Artificial Intelligence Project

-(Anurag Srivastava 2019IMT020)

Title of the project: GRAVITRIPS

Abstract Of The Project:

In this project i have made Game based on the similar iconic 2 player Russian Gravitrips board game of the 80's which was played between the two humans over a physical board ,I have made the computer game on the same logic only in this case the game is played between a Human(User) and a AI (Machine) where the machine aims to win the game against the Human or atleast Tie the game at the very best.

Prerequisite to run the code:

-python3.9 version
-In this project I have used the following Python Libraries-

1. numpy
2. random
3. math
4. pygame
5. sys

GitHub Repository Link to Project:-
https://github.com/StuartRyder/Gravitrips

Introduction Of The Project:

This is a 2-player game played between the User and the AI where the objective is for the AI to achieve 4 balls in a Row/Column or Diagonal Position while preventing the human from doing the same and the one between AI and Human who achieves both the objective first win’s the game.

Rules of the game:-

In this Game there would be a grid of 6 rows and 7 columns and the objective of AI is to drop the ball from the sliding row on the top into one of the 7 valid column such that it achieves the objective of 4 consecutive AI balls in a Row/column/any Diagonal .

Ball colours;-

User-Grey
AI-White
Grid colour-Black

How to Play:-

Before starting, machine(code) decide randomly which of among the user and AI will have the first turn and will be the beginner and then moves are made alternatively, one by turn.

Turns require in placing new pieces on the board; pieces slide downwards from the sliding row at the top and fall down(due to gravity) to the last row or piling up on the last piece introduced in the same column. So, in every turn the introduced piece may be placed at most on seven different squares.Once one Column/Row is filled neither the User nor the AI can fill/replace the existing ball from the cell.

The winner is the first player who gets a straight line made with four own pieces and no gaps between them.

Methodology:

To make the AI work I have implemented the **Minimax Algorithm with Alpha-Beta Pruning** I also learned the **pygame library of the python** while doing this project so i have also tried to implement the Graphic interfacing to the project as well.

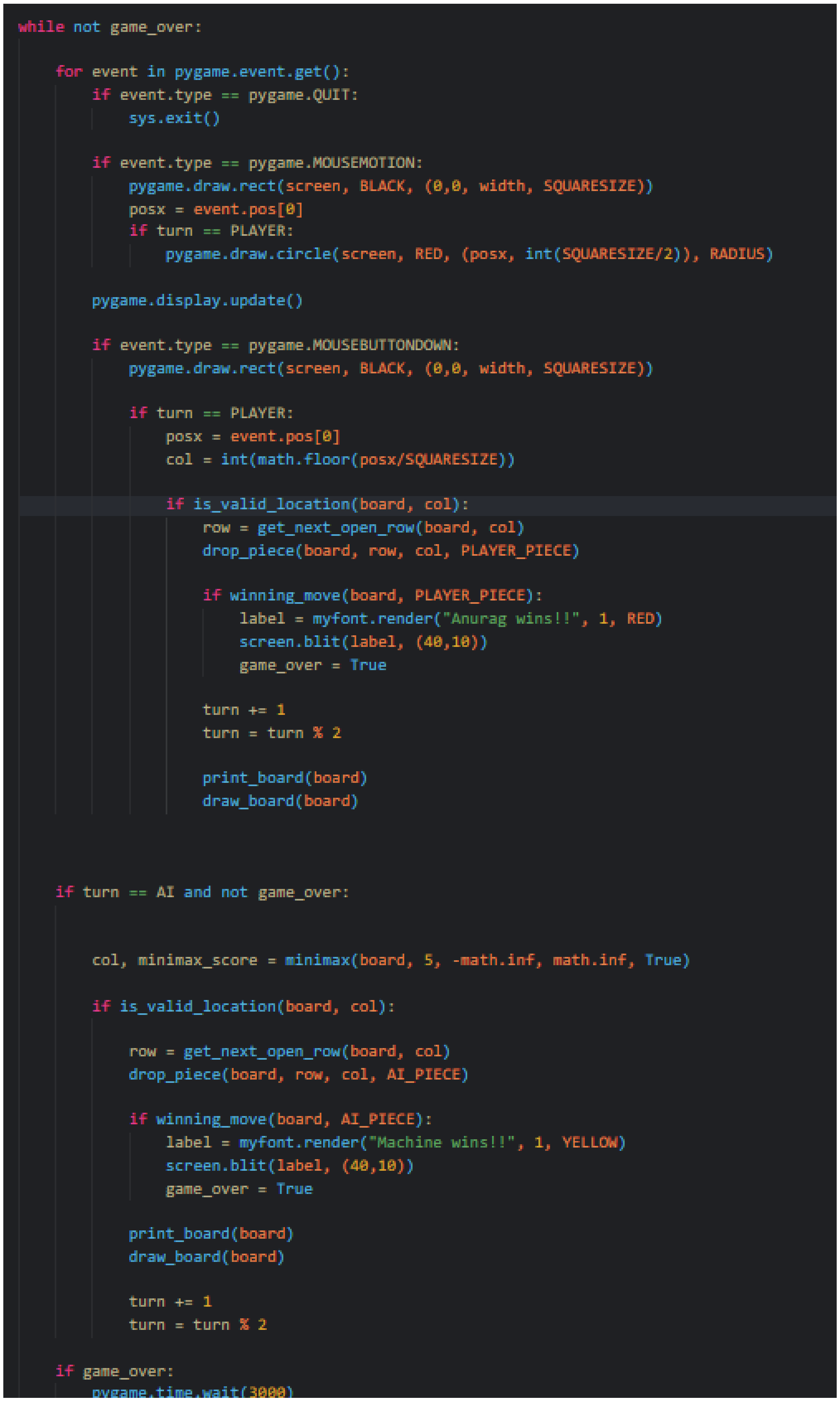
Now I will show you the Steps in which the code game works:-

Step 1-

```
turn = random.randint(PLAYER, AI)
```

In the first step we will initialize our turn variable(variable which keeps a record as to whos turn it is Player or AI) by randint() function which random decides who will go first.

Step 2-



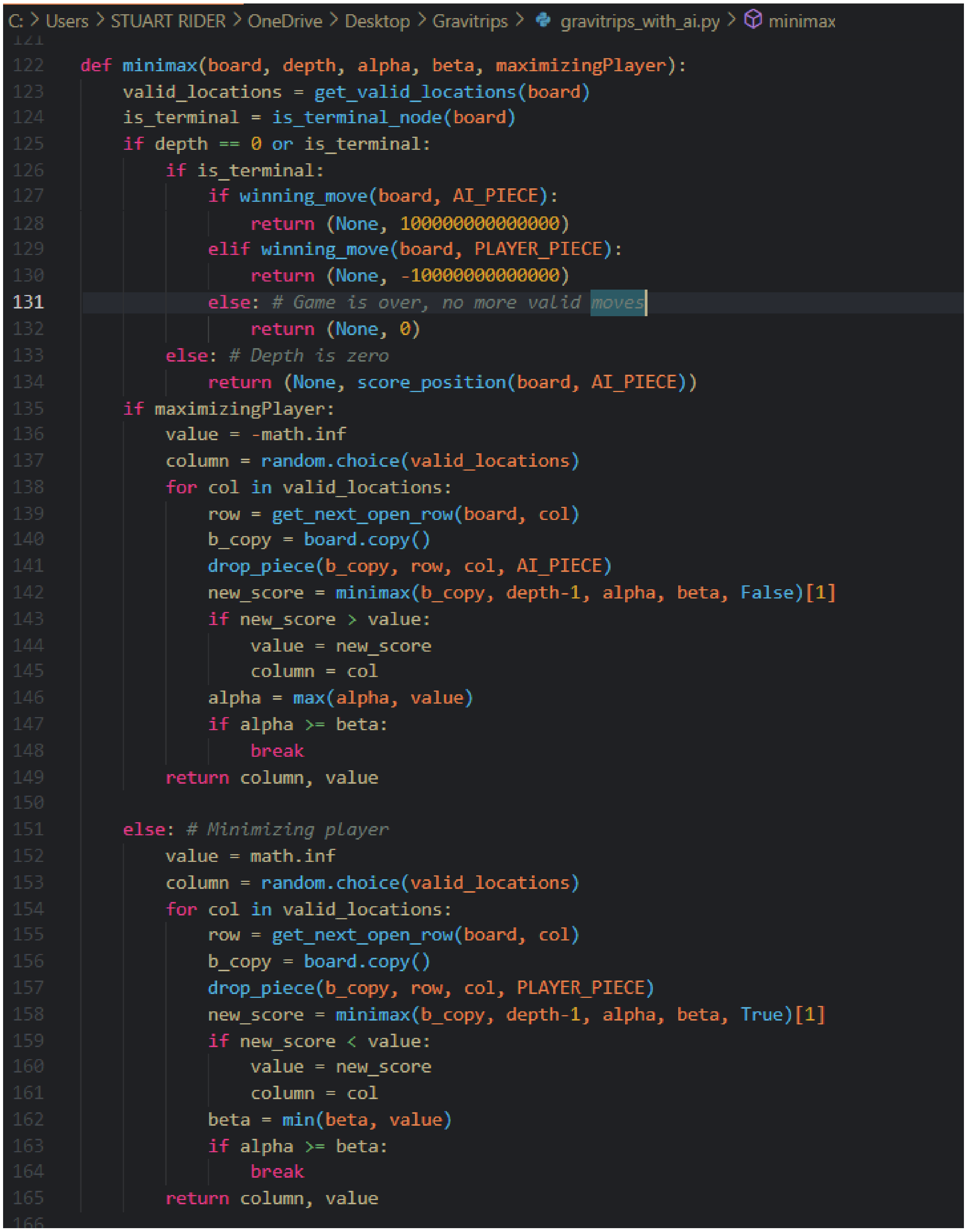
```python
while not game_over:

    for event in pygame.event.get():
        if event.type == pygame.QUIT:
            sys.exit()

        if event.type == pygame.MOUSEMOTION:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))
            posx = event.pos[0]
            if turn == PLAYER:
                pygame.draw.circle(screen, RED, (posx, int(SQUARESIZE/2)), RADIUS)

        pygame.display.update()

        if event.type == pygame.MOUSEBUTTONDOWN:
            pygame.draw.rect(screen, BLACK, (0,0, width, SQUARESIZE))

            if turn == PLAYER:
                posx = event.pos[0]
                col = int(math.floor(posx/SQUARESIZE))

                if is_valid_location(board, col):
                    row = get_next_open_row(board, col)
                    drop_piece(board, row, col, PLAYER_PIECE)

                    if winning_move(board, PLAYER_PIECE):
                        label = myfont.render("Anurag wins!!", 1, RED)
                        screen.blit(label, (40,10))
                        game_over = True

                    turn += 1
                    turn = turn % 2

                    print_board(board)
                    draw_board(board)



    if turn == AI and not game_over:


        col, minimax_score = minimax(board, 5, -math.inf, math.inf, True)

        if is_valid_location(board, col):

            row = get_next_open_row(board, col)
            drop_piece(board, row, col, AI_PIECE)

            if winning_move(board, AI_PIECE):
                label = myfont.render("Machine wins!!", 1, YELLOW)
                screen.blit(label, (40,10))
                game_over = True

            print_board(board)
            draw_board(board)

            turn += 1
            turn = turn % 2

    if game_over:
        pygame.time.wait(3000)
```

In the Second Step we start the infinite loop which keeps on going endlessly until game_over variable becomes false which as a matter of fact becomes false when either the user press the exit button of the game window on the top left corner , or after 3000 s after anyone amongst the AI and Person wins or after it results in a tie.

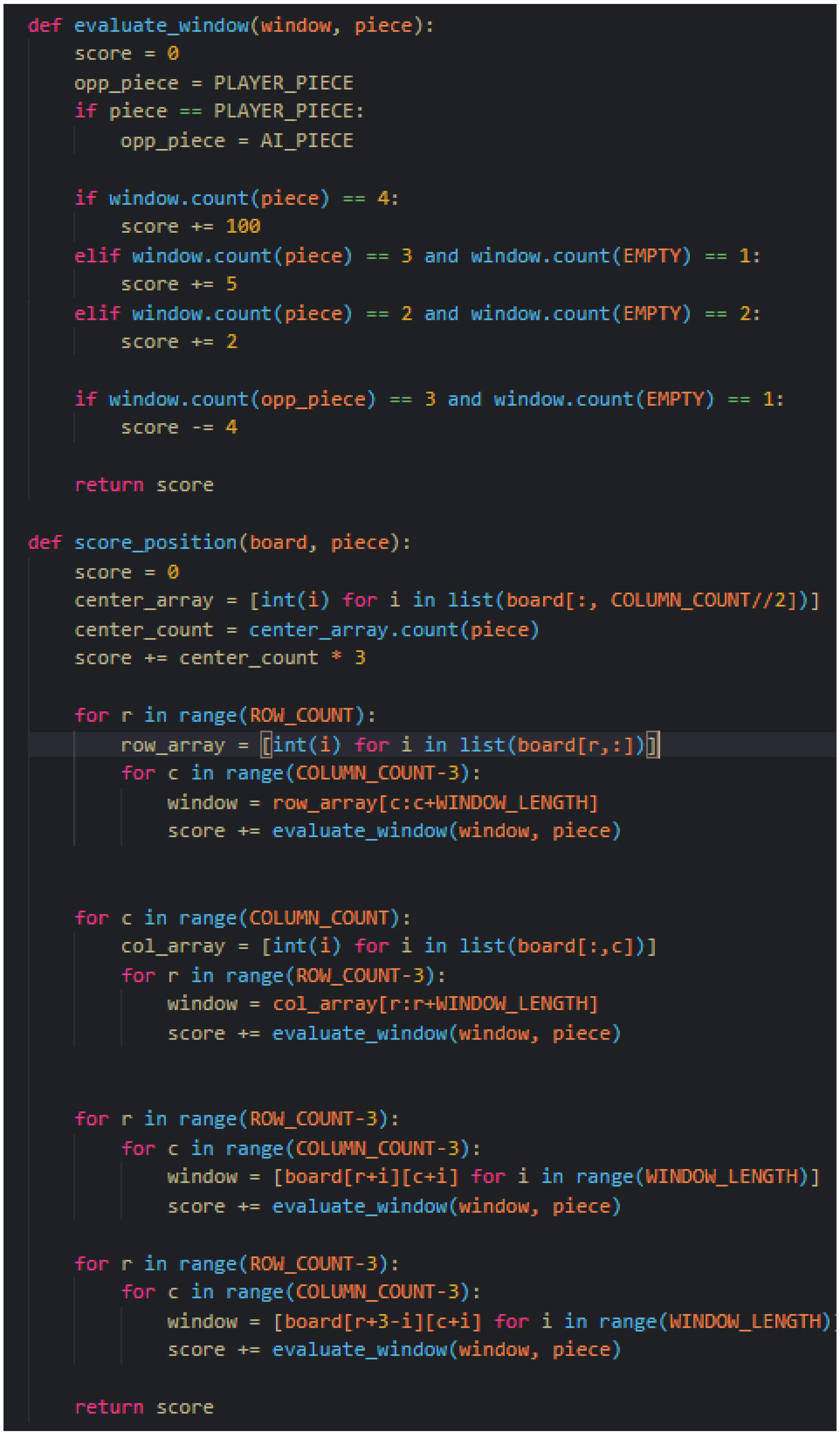
In the while loop the loop continuously keeps on running and keeps a track of person input and AI input using if-else and turn variable and keeps on calling required update functions for both the scenarios in the respective case which are defined within the respective if clauses and keeps on updating turn variable for the next loop until the terminate condition is achieved.

Step 3-

PTO


C: > Users > STUART RIDER > OneDrive > Desktop > Gravitrips > gravitrips_with_ai.py > minimax

```python
def minimax(board, depth, alpha, beta, maximizingPlayer):
    valid_locations = get_valid_locations(board)
    is_terminal = is_terminal_node(board)
    if depth == 0 or is_terminal:
        if is_terminal:
            if winning_move(board, AI_PIECE):
                return (None, 100000000000000)
            elif winning_move(board, PLAYER_PIECE):
                return (None, -10000000000000)
            else: # Game is over, no more valid moves
                return (None, 0)
        else: # Depth is zero
            return (None, score_position(board, AI_PIECE))
    if maximizingPlayer:
        value = -math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, AI_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, False)[1]
            if new_score > value:
                value = new_score
                column = col
            alpha = max(alpha, value)
            if alpha >= beta:
                break
        return column, value

    else: # Minimizing player
        value = math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, PLAYER_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, True)[1]
            if new_score < value:
                value = new_score
                column = col
            beta = min(beta, value)
            if alpha >= beta:
                break
        return column, value
```



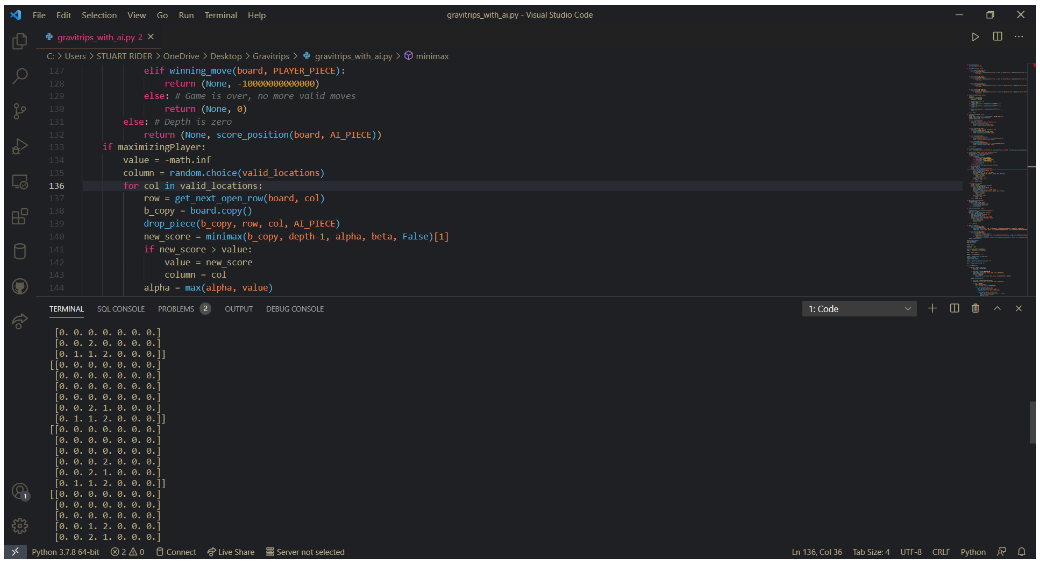
```python
def evaluate_window(window, piece):
    score = 0
    opp_piece = PLAYER_PIECE
    if piece == PLAYER_PIECE:
        opp_piece = AI_PIECE

    if window.count(piece) == 4:
        score += 100
    elif window.count(piece) == 3 and window.count(EMPTY) == 1:
        score += 5
    elif window.count(piece) == 2 and window.count(EMPTY) == 2:
        score += 2

    if window.count(opp_piece) == 3 and window.count(EMPTY) == 1:
        score -= 4

    return score

def score_position(board, piece):
    score = 0
    center_array = [int(i) for i in list(board[:, COLUMN_COUNT//2])]
    center_count = center_array.count(piece)
    score += center_count * 3

    for r in range(ROW_COUNT):
        row_array = [int(i) for i in list(board[r,:])]
        for c in range(COLUMN_COUNT-3):
            window = row_array[c:c+WINDOW_LENGTH]
            score += evaluate_window(window, piece)


    for c in range(COLUMN_COUNT):
        col_array = [int(i) for i in list(board[:,c])]
        for r in range(ROW_COUNT-3):
            window = col_array[r:r+WINDOW_LENGTH]
            score += evaluate_window(window, piece)


    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+i][c+i] for i in range(WINDOW_LENGTH)]
            score += evaluate_window(window, piece)

    for r in range(ROW_COUNT-3):
        for c in range(COLUMN_COUNT-3):
            window = [board[r+3-i][c+i] for i in range(WINDOW_LENGTH)
            score += evaluate_window(window, piece)

    return score
```

Minimax function:-
This is the logic function which implements **minimax alpha-beta pruning** which is helping the AI to decide which move to make next move . In this my Heuristic is `100000000000000 and -100000000000000` which is basically a large number which results in the final conclusion of termination of the minimax recursive function this number is for helping the AI decide whether the future move made will result in making the AI win or the Human win.

We keep on changing the score using evaluate_window() function on the comparison count of on the board is such that 4,3,2 orientations are filled in a consecutive positions for that we are incrementing the score variable by a arbitrary number and if the opponent piece is such that it is in a position that 3 pieces are in row then it negates(-) the score with a value , for the algorithm to compare it with the existing score and decide the future move and thus facilitating the that AI to make its next move.

Finally, this minimax function helps the AI make a smart and knowledge based move which would help it win.

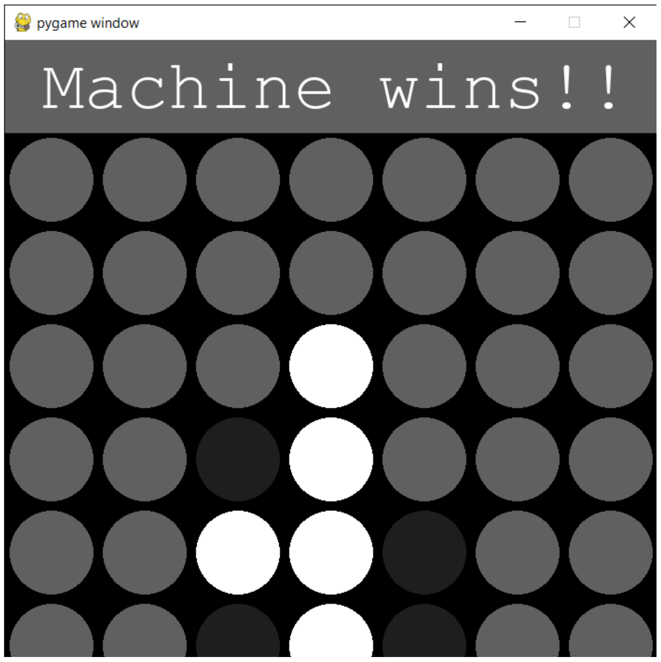
Result and Discussion:



```python
            elif winning_move(board, PLAYER_PIECE):
                return (None, -10000000000000)
            else: # Game is over, no more valid moves
                return (None, 0)
        else: # Depth is zero
            return (None, score_position(board, AI_PIECE))
    if maximizingPlayer:
        value = -math.inf
        column = random.choice(valid_locations)
        for col in valid_locations:
            row = get_next_open_row(board, col)
            b_copy = board.copy()
            drop_piece(b_copy, row, col, AI_PIECE)
            new_score = minimax(b_copy, depth-1, alpha, beta, False)[1]
            if new_score > value:
                value = new_score
                column = col
            alpha = max(alpha, value)
```

```
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 2. 0. 0. 0. 0.]
 [0. 1. 1. 2. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 2. 1. 0. 0. 0.]
 [0. 1. 1. 2. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 2. 0. 0. 0.]
 [0. 0. 2. 1. 0. 0. 0.]
 [0. 1. 1. 2. 0. 0. 0.]]
[[0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 0. 0. 0. 0. 0.]
 [0. 0. 1. 2. 0. 0. 0.]
 [0. 0. 2. 1. 0. 0. 0.]
```

Backend matrix of the game



GUI-Interface window of the game

So as we can clearly see in the Screenshots that machine wins the game against the human by making just 5 move . This Heuristics works with 99% of success rate and results in AI winning the game and only 1% of the cases it occurs that it loses or ties the game.
My heuristic is good as it provides such a value to score for incrementing which creates a large difference between the values of the score variable which eventually help the algorithm decide the next move but not too large that it makes the file heavy and slow.

Conclusion of the project:

This project was a great source of learning as i get to learn and implement the heuristics search paradigm in a real-world via AI algorithms although I ended up making my project in Minimax-Alpha-Beta pruning but still in the time of research i get to learn a lot of other AI algorithms as well, I also got to learn a lot of newer python libraries which i never needed previously in my projects and coding in general some of which were PyQt5 and pygame library .
With this I end my project .

Thanking you
Anurag Srivastava
(2019IMT-020)

--------------------------------Thank You---