Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

Introduction:

In this project we were assigned to create an Artificial Intelligence agent that played connect 4 using the minimax algorithm, which is also known as the negamax algorithm. I chose to design and implement the program using Python with the Pycharm IDE. My results of the program I feel were pretty successful and I ended up with an AI that is pretty good at playing connect 4. Another interesting feature I chose to add was AI vs AI just to see how they stacked up. It seemed to be a pretty even split for which AI is the winner.

Core:
My minimax algorithm is similar to the one that Dr. Soule did in class as I was struggling to understand the minimax until he did it in class. After having that example and studying the algorithm from outside sources I feel like I have a better understanding.

```
rows = 6
cols = 7
maxdepth = 6
alpha = -9999
beta = 9999
```

The above snip is to give context to some global variables used in the algorithm. My algorithm goes to a maximum depth of 6 and it incrementally goes there by calling the respective mini and max functions. I was able to implement alpha and beta pruning, which has importance not only in efficiency but also keeping the minimax function optimal. It does this by preventing the search from going down branches of the tree that are meaningless in calculations to a solution.

```
#Return the best column to place the chip in
def chooseMove(b):
    bestmove = None
    bestvalue = float("-inf")
    tempValues = []
    for c in range(0, cols):
        if(b[0][c] == 0):   #0 is top row, make sure column is not full
            value = mini(b,c,1) #b = current board, c = column for move, 1 is the depth
            tempValues.insert(0, value)
            print(str(c) + " value= " + str(value))
            if(value > bestvalue): #If the current value is better then the best prior value assign it
                bestvalue = value
                bestmove = c
            if all ([v == 0 for v in tempValues]): #If all the AI calculations come out to zero
                bestmove = random.randint(0,6) #Choose random column to place in
    return bestmove
```

The next snip here is the choose move function. The idea behind this function is to use the minimax algorithm to return the best move for the AI. It does so by incrementing through the

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

columns, using the minimax function on each column, and finding the best solution. If there is no best solution I have the AI introduce a random play into the game to make it more interesting.

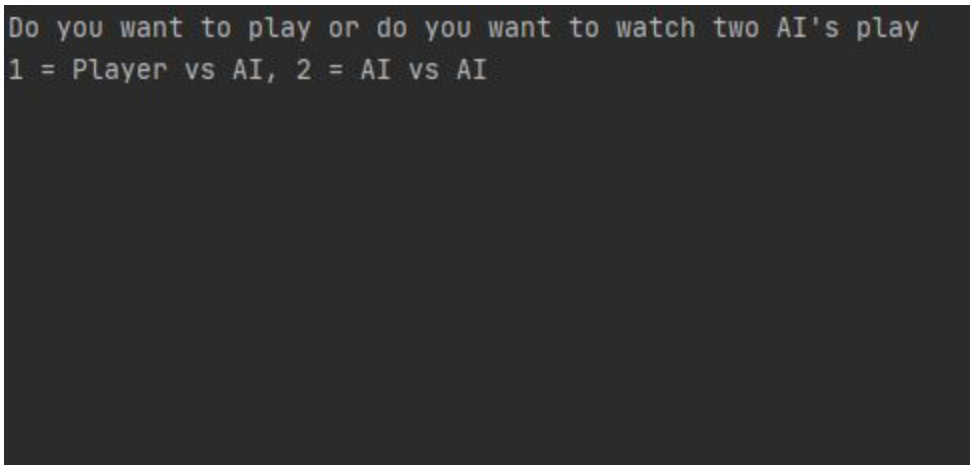
```
#find minimum value but if no win or loss call maxi function
def mini(b, c, depth):
    global beta
    global alpha
    newb = copy.deepcopy(b) #make temporary new board to look ahead
    makeMove(newb, c, 'X') #Place AI piece
    s = checkWinOrEval(newb)
    #s = score_move(newb, 'X')
    if(s == 1):
        return s #winner
    if(s == -1):
        return s #winner
    if(depth == maxdepth):
        return s #currently always 0
    worstvalue = float("inf")
    for c in range (0,cols):
        if(b[0][c] == 0): #Make sure column is empty
            value = maxi(newb,c,depth+1)
            if(value < worstvalue):
                worstvalue = value
            beta = min(beta,value) #alpha beta pruning
            if beta <= alpha:
                break
    return worstvalue
```

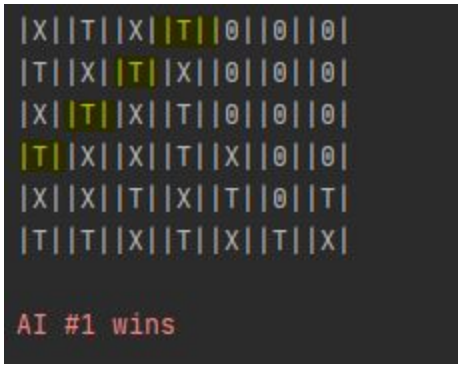
Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

```
def maxi(b, c, depth):
    global beta
    global alpha
    newb = copy.deepcopy(b) #make temporary new board to look ahead
    makeMove(newb, c, 'T') #Place player piece
    s = checkWinOrEval(newb)
    #s = score_move(newb, 'X')
    if(s == 1):
        return s # AI wins
    if(s == -1):
        return s # player wins
    if(depth == maxdepth):
        return s #currently always 0
    bestvalue = float("-inf")
    for c in range (0,cols):
        if(b[0][c] == 0): #Make sure column is empty
            value = mini(newb,c,depth+1)
            if(value > bestvalue):
                bestvalue = value
            alpha = max(alpha, value) #alpha beta pruning
            if beta <= alpha:
                break
    return bestvalue
```

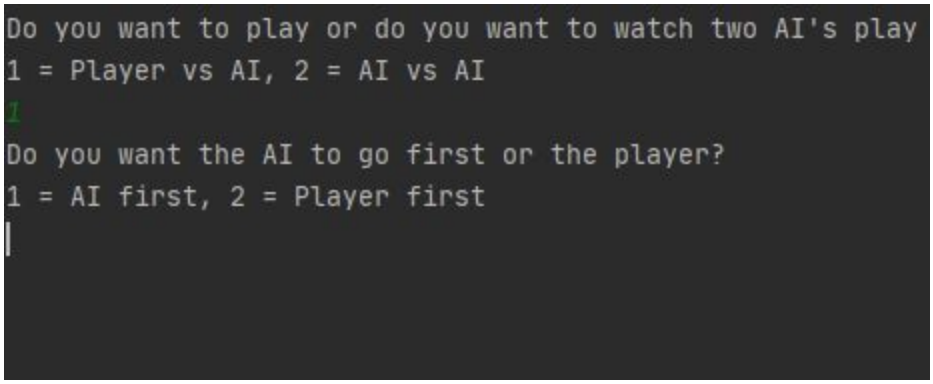
The next two functions are similar but have some differences in either maximizing the value or minimizing the value. The mini max algorithm works by checking future game state and then it makes the move that maximizes the minimum value of the position resulting from the opponent's possible future moves. Along with the minimax algorithm I was able to implement alpha beta pruning which helps remove extra states and an AI vs AI feature (Shown below)

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2





For the AI vs Player option it asks the user who would like to go first (shown below)



My evaluation function (its long):

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

```
def checkWinOrEval(b):
    score =0
    #Horizontal, left to right
    for i in range(rows):
        for j in range(cols-3):
            if(b[i][j] == 'X' and b[i][j+1] == 'X' and b[i][j+2] == 'X' and b[i][j+3] == 'X'):
                score = 1
                return score #AI 1 win
            if(b[i][j] == 'T' and b[i][j+1] == 'T' and b[i][j+2] == 'T' and b[i][j+3] == 'T'):
                score = -1
                return score #Player win/ AI #2 win
            if (b[i][j] == 'X' and b[i][j + 1] == 'X' and b[i][j+2] == 'X' and b[i][j+3] == 0):
                score = 0.5
                return score
            if (b[i][j] == 'T' and b[i][j+1] == 'T' and b[i][j+2] == 'T' and b[i][j+3] == 0):
                score -0.5
                return score
            if (b[i][j] == 'X' and b[i][j + 1] == 'X' and b[i][j+2] == 0 and b[i][j+3] == 0):
                score = 0.2
                return score
            if (b[i][j] == 'T' and b[i][j+1] == 'T' and b[i][j+2] == 0 and b[i][j+3] == 0):
                score = 0.2
                return score
```

```
    #horizontal, right to left
    for i in reversed(range(rows)):
        for j in range(cols - 3):
            if (b[i][j] == 'X' and b[i][j - 1] == 'X' and b[i][j - 2] == 'X' and b[i][j - 3] == 'X'):
                score = 1
                return score  # AI 1 win
            if (b[i][j] == 'T' and b[i][j - 1] == 'T' and b[i][j - 2] == 'T' and b[i][j - 3] == 'T'):
                score = -1
                return score  # Player win/ AI #2 win
            if (b[i][j] == 'X' and b[i][j - 1] == 'X' and b[i][j - 2] == 'X' and b[i][j - 3] == 0):
                score = 0.5
                return score
            if (b[i][j] == 'T' and b[i][j - 1] == 'T' and b[i][j - 2] == 'T' and b[i][j - 3] == 0):
                score - 0.5
                return score
            if (b[i][j] == 'X' and b[i][j - 1] == 'X' and b[i][j - 2] == 0 and b[i][j - 3] == 0):
                score = 0.2
                return score
            if (b[i][j] == 'T' and b[i][j - 1] == 'T' and b[i][j - 2] == 0 and b[i][j - 3] == 0):
                score = 0.2
                return score
```

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

```
    #Vertical
    for i in range(rows-3):
        for j in range(cols):
            if (b[i][j] == 'X' and b[i+1][j] == 'X' and b[i+2][j] == 'X' and b[i+3][j] == 'X'):
                score = 1
                return score  # AI 1 win
            if (b[i][j] == 'T' and b[i+1][j] == 'T' and b[i+2][j] == 'T' and b[i+3][j] == 'T'):
                score = -1
                return score  # Player win/ AI #2 win
            if (b[i][j] == 'X' and b[i+1][j] == 'X' and b[i+2][j] == 'X' and b[i+3][j] == 0):
                score = 0.5
                return score
            if (b[i][j] == 'T' and b[i+1][j] == 'T' and b[i+2][j] == 'T' and b[i+3][j] == 0):
                score - 0.5
                return score
            if (b[i][j] == 'X' and b[i+1][j] == 'X' and b[i+2][j] == 0 and b[i+3][j] == 0):
                score = 0.2
                return score
            if (b[i][j] == 'T' and b[i+1][j] == 'T' and b[i+2][j] == 0 and b[i+3][j] == 0):
                score = 0.2
                return score
```

```
#Diagonal
for i in range(rows - 3):
    for j in range(3,cols):
        if (b[i][j] == 'X' and b[i + 1][j-1] == 'X' and b[i + 2][j-2] == 'X' and b[i + 3][j-3] == 'X')
                score = 1
                return score  # AI 1 win
        if (b[i][j] == 'T' and b[i + 1][j-1] == 'T' and b[i + 2][j-2] == 'T' and b[i + 3][j-3] == 'T')
                score = -1
                return score  # Player win/ AI #2 win
        if (b[i][j] == 'X' and b[i + 1][j-1] == 'X' and b[i + 2][j-2] == 'X' and b[i + 3][j-3] == 0):
                score = 0.5
                return score
        if (b[i][j] == 'T' and b[i + 1][j-1] == 'T' and b[i + 2][j-2] == 'T' and b[i + 3][j-3] == 0):
                score - 0.5
                return score
        if (b[i][j] == 'X' and b[i + 1][j-1] == 'X' and b[i + 2][j-2] == 0 and b[i + 3][j-3] == 0):
                score = 0.2
                return score
        if (b[i][j] == 'T' and b[i + 1][j-1] == 'T' and b[i + 2][j-2] == 0 and b[i + 3][j-3] == 0):
                score = 0.2
                return score
```

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

```
#Diagonal #2
for i in range(3,rows):
    for j in range(3, cols):
        if (b[i][j] == 'X' and b[i -1][j-1] == 'X' and b[i - 2][j-2] == 'X' and b[i - 3][j-3] == 'X'):
                score = 1
                return score  # AI 1 win
        if (b[i][j] == 'T' and b[i - 1][j-1] == 'T' and b[i - 2][j-2] == 'T' and b[i - 3][j-3] == 'T'):
                score = -1
                return score  # Player win/ AI #2 win
        if (b[i][j] == 'X' and b[i - 1][j-1] == 'X' and b[i - 2][j-2] == 'X' and b[i - 3][j-3] == 0):
                score = 0.5
                return score
        if (b[i][j] == 'T' and b[i - 1][j-1] == 'T' and b[i - 2][j-2] == 'T' and b[i - 3][j-3] == 0):
                score - 0.5
                return score
        if (b[i][j] == 'X' and b[i - 1][j-1] == 'X' and b[i - 2][j-2] == 0 and b[i - 3][j-3] == 0):
                score = 0.2
                return score
        if (b[i][j] == 'T' and b[i - 1][j-1] == 'T' and b[i - 2][j-2] == 0 and b[i - 3][j-3] == 0):
                score = 0.2
                return score

return score
```

My idea behind the evaluation function was to have a score variable that kept track of the score and returned it depending on the circumstances. I have 5 main parts that are divided up into horizontal (left to right), horizontal (right to left), vertical, and the two diagonals. I would say it's not the most efficient function but it seems to work well for most cases.

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2
Here are some examples of gameplay:

```
|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||0||0|
|T||T||X||X||0||0||0|
```

```
The AI is planning its move
0 value= 0.2
1 value= 0.2
2 value= 0.2
3 value= 0.2
4 value= 0.5
5 value= 0
6 value= 0.2
|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||0||0|
|T||T||X||X||X||0||0|

Move: 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|
|T||0||0||0||0||0||0|
|T||T||X||X||X||0||0|

The AI is planning its move
0 value= 0.2
1 value= 0.5
2 value= 0.5
3 value= 0.5
4 value= 0.5
5 value= 1
6 value= 0.5
|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|
|T||0||0||0||0||0||0|
|T||T||X||X||X||X||0|

AI wins
Move:
```

Hunter Hawkins Stark
CS 470 Artificial Intelligence
Summer 2020 Dr. Soule
Project #2

So in the first example here the AI and player start off with rows 0-3 filled. The first two are filled by the player and the second two are filled by the AI. When it receives this board it evaluates and determines the best play is 0.5 to enhance its progress to 3 in a row rather than blocking the player from getting 2 in a row. After doing that the player makes a mistake of dropping a piece into row #0. Once that happens the AI then detects that it has a winning move of value 1 in row #5. The program seems to take wins when possible and block losses when possible, however it doesn't always 100% block losses. I'm not entirely sure why it fails to occasionally block a winning move from the player.

Conclusion:

As for my program's playability it seems to be pretty engaging. It's not perfect as occasionally it will miss a winning move for the opposing player or AI but it is still entertaining and the majority of the time it makes educated moves. As for improvements I would like to make the interface a little better and improve it to have various difficulties and the hardest being an undefeatable AI. The program has its strengths and weaknesses like any others produced, but I'm proud of progress and have learned a lot along the way. My hope is to keep working on this outside of class to consistently improve my AI.