Assignment 3: Algorithm Min-Max, MinMax+α-β pruning and Heuristic Min-Max

Aliss Francisco Gabriel

Mendez Camacho Cesar Augusto

- Explanation Of the Heuristic Functions

1. Heuristic Weak

```
def heuristic_weak(board, player):
    black_score, white_score = get_score(board)
    if player == BLACK:
        return black_score - white_score
    else:
        return white_score - black_score

```

In this heuristic function, you simply get the amount of chips that the corresponding player has over its opponent, for example if black has 35 and white 30, the function returns 5 (for the black player). In the case of player being white would indicate that he has 5 chips less than black.

2. **Heuristic_look_for_corners_and_centers**

```
def heuristic_look_for_corners_and_center(board, player):
    value = 0
    for i in range(8):
        for j in range(8):
            if board[i][j] == player:
                if (i == 0 and j == 0) or (i == 7 and j == 7) or \
                   (i == 0 and j == 7) or (i == 7 and j == 0):
                    value += 20 # Corners, value = 20
                elif (i == 3 and j == 3) or (i == 3 and j == 4) or \
                     (i == 4 and j == 3) or (i == 4 and j == 4):
                    value += 5 # Centers value = 5
                elif (i == 0 and j == 1) or (i == 1 and j == 0) or \
                     (i == 7 and j == 1) or (i == 6 and j == 0) or \
                     (i == 0 and j == 6) or (i == 1 and j == 7) or \
                     (i == 7 and j == 6) or (i == 6 and j == 7) or \
                     (i == 1 and j == 1) or (i == 6 and j == 6) or \
                     (i == 6 and j == 1) or (i == 1 and j == 6):
                    value += 0 # positions adjacent to corners,value=0
                else:
                    value += 10 # any other position, value = 10

    return value
```

In this heuristic function the positions of the map are taken into account, we calculate a value, according to the position of the tiles of a player N that are on the map, the corners are worth 20 because these give us more chance to flank the opponent. The centers are valued at 5 because the center of the map serves to expand to the corners and to effectively force the opponent to make bad moves.

The positions adjacent to the corners are valued at 0 because these give away the corners, and we know corners are pretty important.

Finally any other box is worth 10.

3. **heuristic_look_for_corners_and_borders**

```
def heuristic_look_for_corners_and_borders(board, turn):
    value = 0
    for i in range(8):
        for j in range(8):
            if board[(i, j)] == turn:
                if (i == 0 and j == 0) or (i == 7 and j == 7) or \
                   (i == 0 and j == 7) or (i == 7 and j == 0):
                    value += 50 #corners, value = 50
                elif (i == 3 and j == 3) or (i == 3 and j == 4) or \
                     (i == 4 and j == 3) or (i == 4 and j == 4):
                    pass # centers no value
                elif (i == 0 and j == 1) or (i == 1 and j == 0) or \
                     (i == 7 and j == 1) or (i == 6 and j == 0) or \
                     (i == 0 and j == 6) or (i == 1 and j == 7) or \
                     (i == 7 and j == 6) or (i == 6 and j == 7):
                     value -= 1#positions adjacent to corners,value=-1
                elif (i == 1 and j == 1) or (i == 6 and j == 6) or \
                     (i == 6 and j == 1) or (i == 1 and j == 6):
                     value -= 10 #adjacent diagonal to the corner,
value = -10
                elif (i == 0 and j == 2) or (i == 0 and j == 5) or \
                     (i == 7 and j == 2) or (i == 7 and j == 5) or \
                     (i == 2 and j == 0) or (i == 5 and j == 0) or \
                     (i == 2 and j == 7) or (i == 5 and j == 7):
                     value += 5 # border column, value = 5
                elif (i == 0 and j == 3) or (i == 0 and j == 4) or \
                     (i == 7 and j == 3) or (i == 7 and j == 4) or \
                     (i == 3 and j == 0) or (i == 4 and j == 0) or \
                     (i == 3 and j == 7) or (i == 4 and j == 7):
                     value += 2 # border row, value = 2
                else:
                     value += 1 # any other position, value = 1
    return value
```

In this heuristic, the corners are worth more, 50.

The centers have no value, basically because the center slots will give a better access to the border slots. So instead of focusing on the center to get to the border, the focus is changed directly to the borders.

The positions adjacent to the corners have a negative value of -1, now a penalization will be implemented in the case of a ‘dumb move’ that gives the corner away. Only the slots that are adjacent but not diagonal to the corner are used.

Those that are diagonally to the corners have a negative value of -10 since these give the corner to the other player. The penalization is bigger because a diagonal line from corner to corner is more threatening for the player, so giving up this position is potentially more dangerous.

The side edges of the columns have a value of 5, and the bottom and top edges a value of 2. Basically a better value was assigned to the side edges because they give better access to corners than top and bottom edges.

Any other box has a value of 1.

- **Experiments made**

Running regular Min-Max to check amount of states expanded

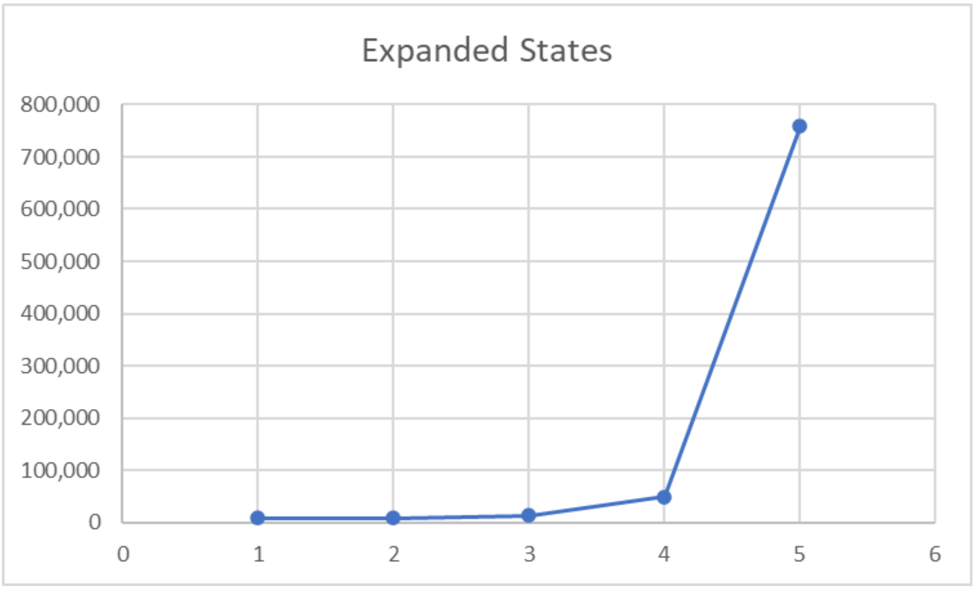
To run the experiment on the computer, the size of the map was changed to 4x4. As the following min-max variations expand all the space of states, running this on an 8x8 board is an unthinkable feat. The 8x8 board has 64 slots and each slot can have

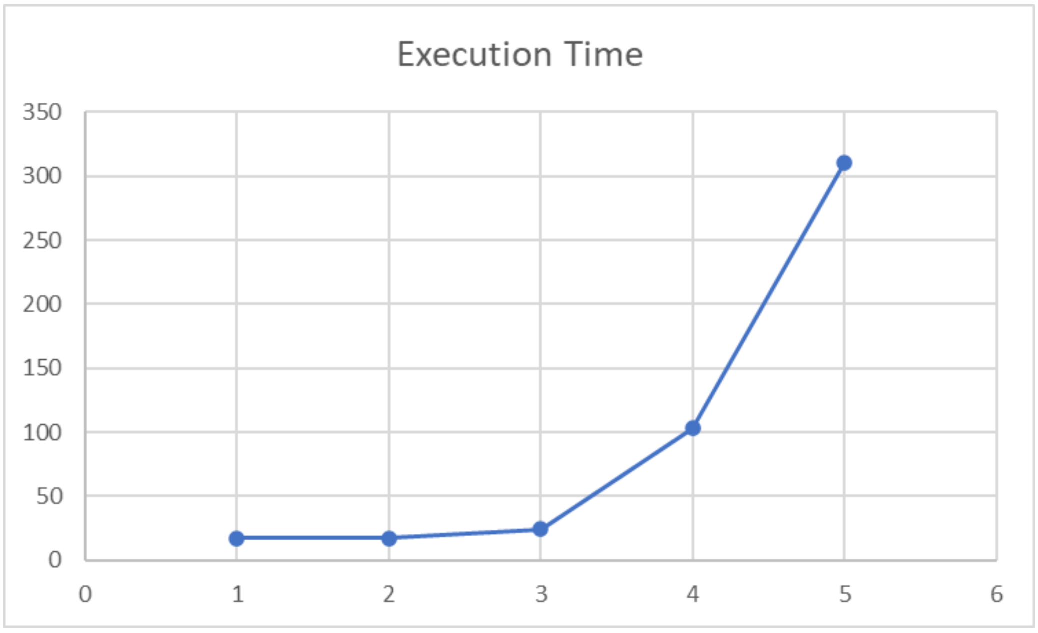
3 states (empty, black, white) so the computer will not be able to work with so many states.

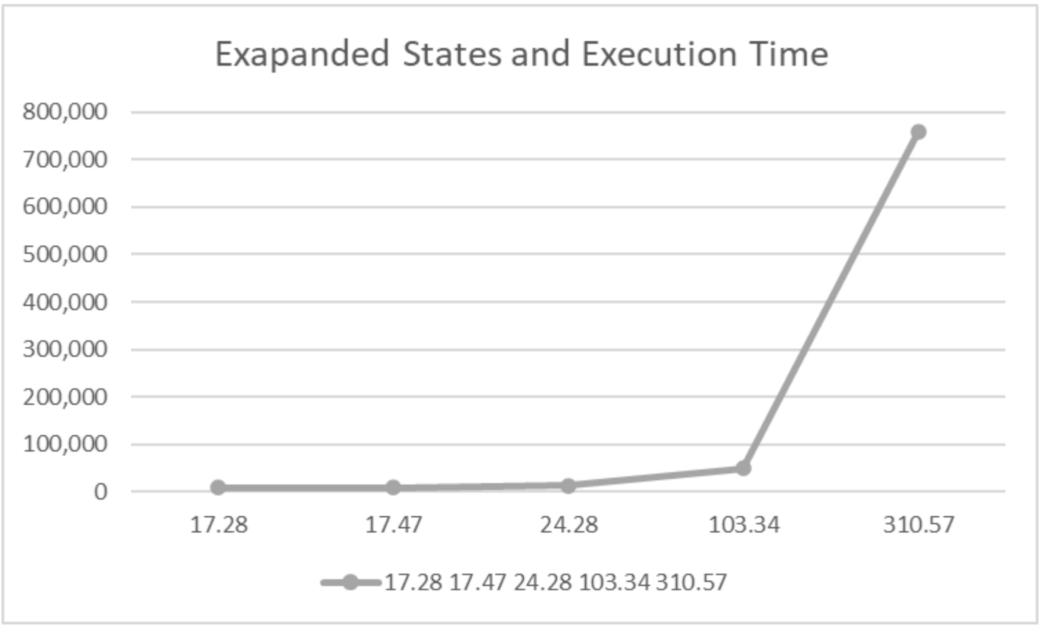
The following table shows the amount of expanded states in 5 different runs in a Player vs Computer experiment:

REGULAR MIN-MAX

Run	Expanded States	Winner	Execution Time
1	13,423	Computer	24.28 [s]
2	49,821	Computer	103.34 [s]
3	8,956	Player	17.28 [s]
4	8,964	Player	17.47 [s]
5	758,373	Computer	310.57 [s]







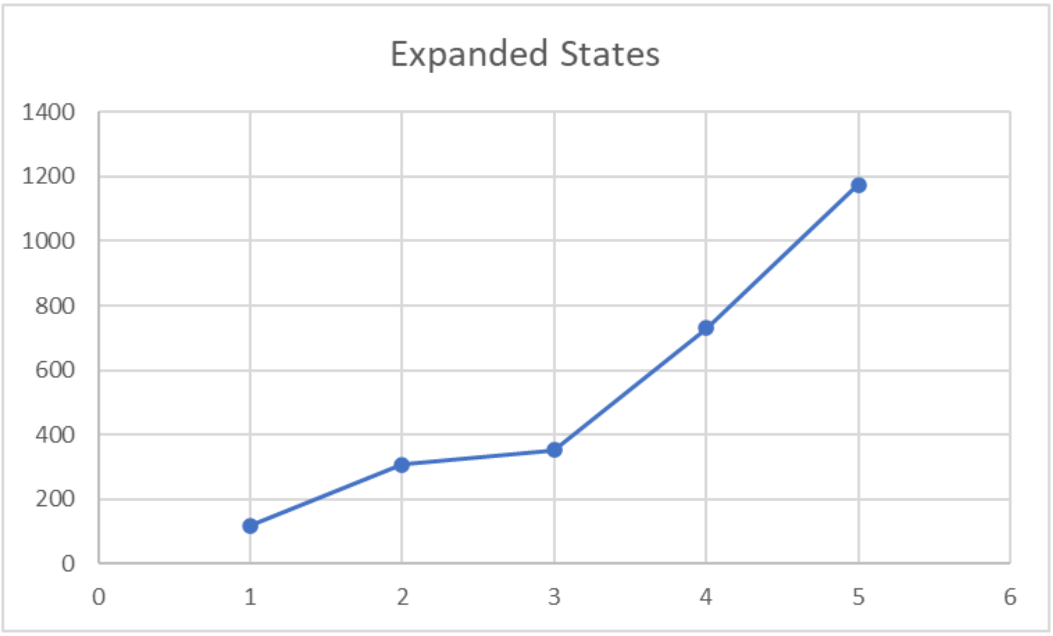
So memory-wise, MinMax is very heavy. A surprising number of 758,373 states were expanded in one of the runs. Time-wise it is also very heavy, the experiment was run at night, so the air was full of impatience making the seconds feel longer. Anyway, taking the anemic factor away, the regular MinMax is really ineffective in terms of memory and time.

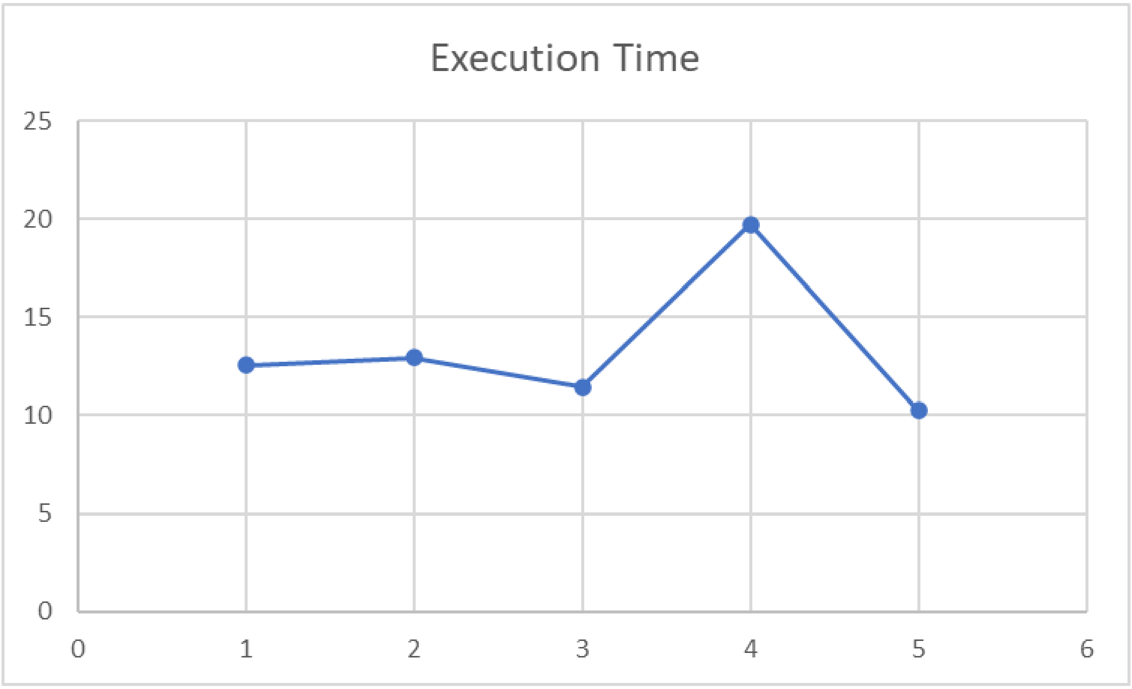
MIN-MAX ALPHA-BETA PRUNING

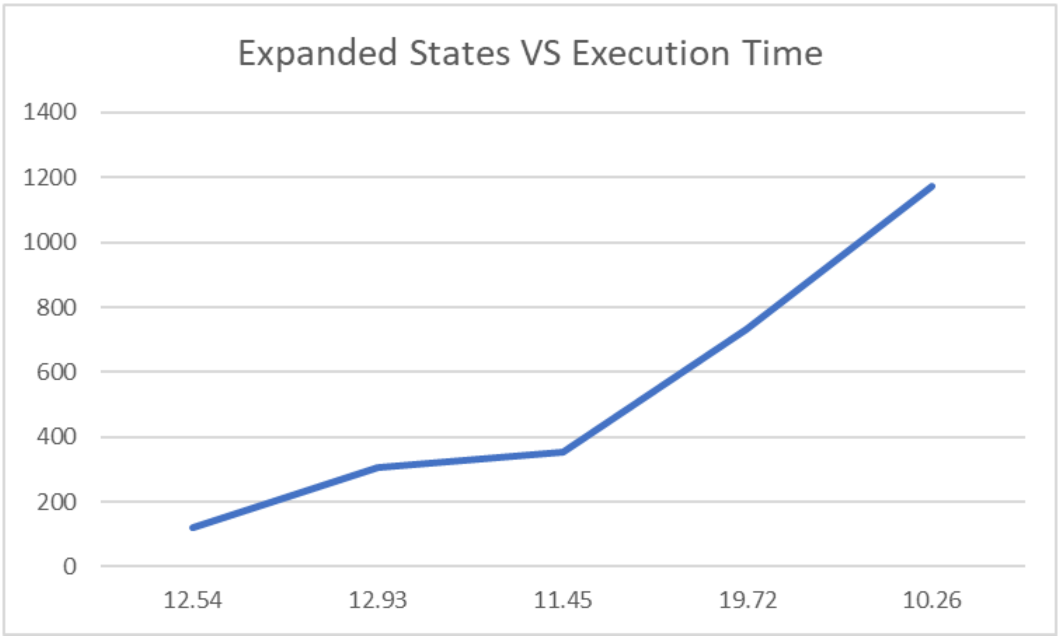
To run this experiment was done the same as in the previous one, reducing the size of the board to one of 4x4, but it was also tested with one 6x6.

4x4 Board

Run	Expanded States	Winner	Execution Time
1	731	Computer	19.72 [s]
2	306	Player	11.45 [s]
3	353	Player	12.93 [s]
4	119	Player	12.54 [s]
5	1175	Computer	10.26 [s]

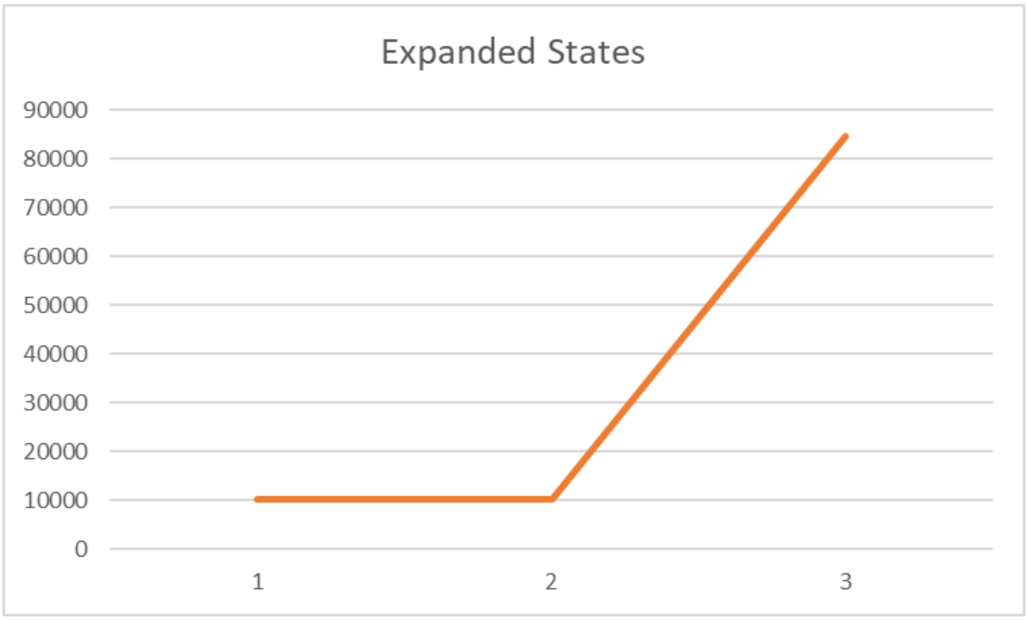


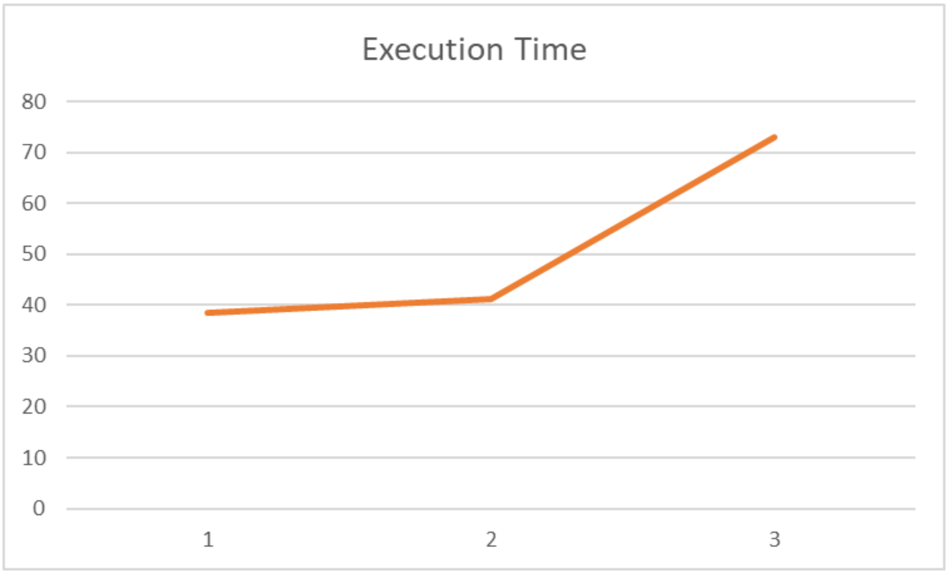


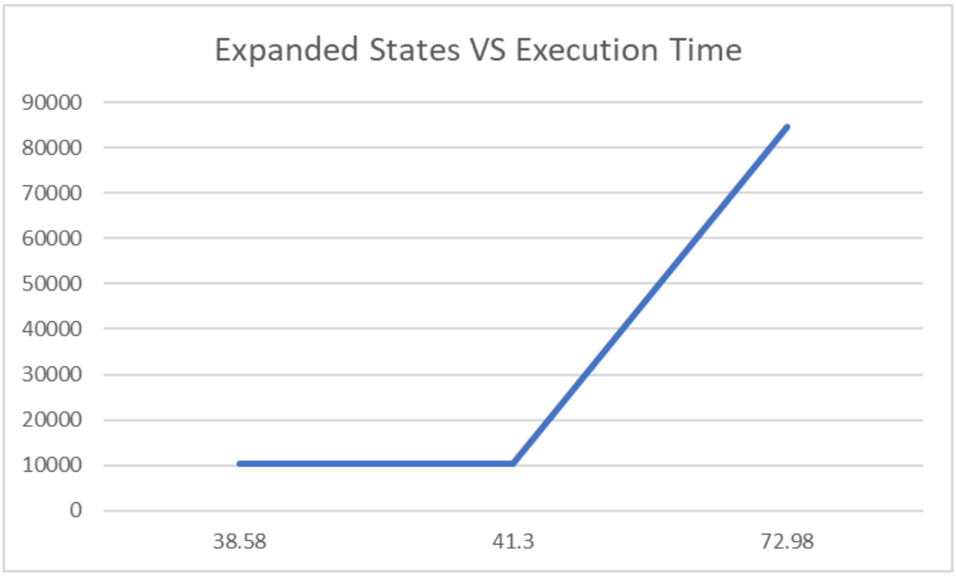


6x6 Board

Run	Expanded States	Winner	Execution Time
1	10,301	Computer	38.58 [s]
2	84,616	Player	72.98 [s]
3	10,310	Computer	41.30 [s]
4	???	???	The program ran for 1 hour and didnt finished the execution







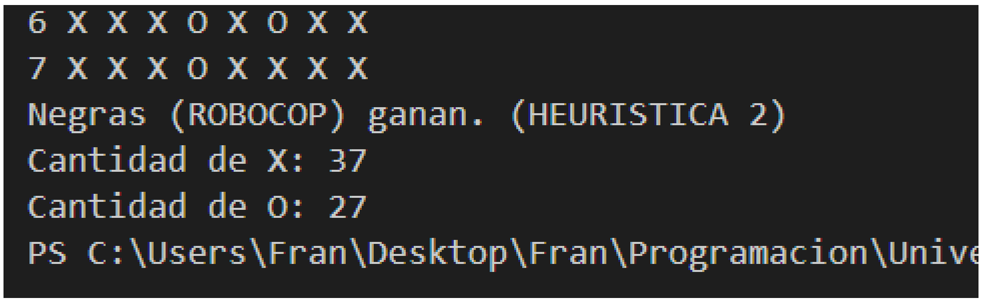
Comparing the 2 tables, you can see that while increasing the size of the board, the number of states increases significantly, and the execution time also, being that in some executions the algorithm took quite a while to find the best move, left running the program for an hour and a half and did not finish running

MIN-MAX ALPHA-BETA PRUNING HEURISTIC

Experiment 1 (AI VS AI)

To determine the best heuristic a cool experiment was made. An implementation of a computer v computer mode was made. This experiment was run 3 times, the first run had a maximum depth of 3, second run had a maximum depth of 4 and the third run had a maximum depth of 6.

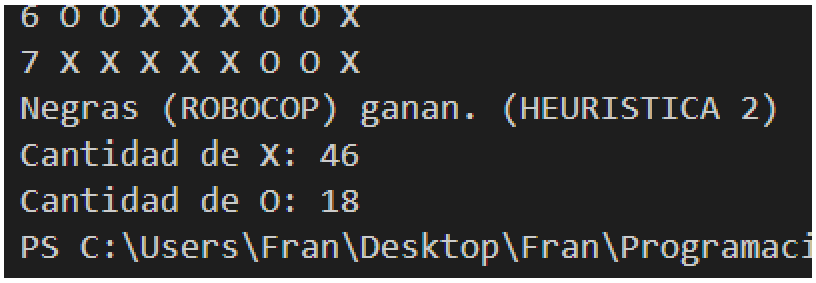
Max Depth 3:



```
6 X X X O X O X X
7 X X X O X X X X
Negras (ROBOCOP) ganan. (HEURISTICA 2)
Cantidad de X: 37
Cantidad de O: 27
PS C:\Users\Fran\Desktop\Fran\Programacion\Unive
```

- **Chips difference:** 10
- **Winner ->** *heuristic_look_for_corners_and_borders*

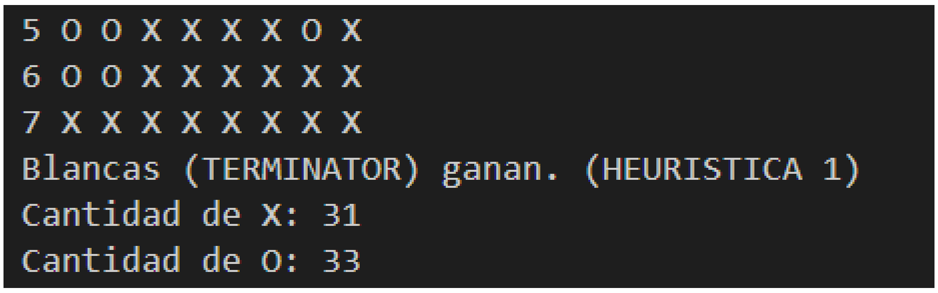
Max Depth 4:



```
6 O O X X X O O X
7 X X X X X O O X
Negras (ROBOCOP) ganan. (HEURISTICA 2)
Cantidad de X: 46
Cantidad de O: 18
PS C:\Users\Fran\Desktop\Fran\Programaci
```

- **Chips difference:** 28
- **Winner ->** *heuristic_look_for_corners_and_borders*

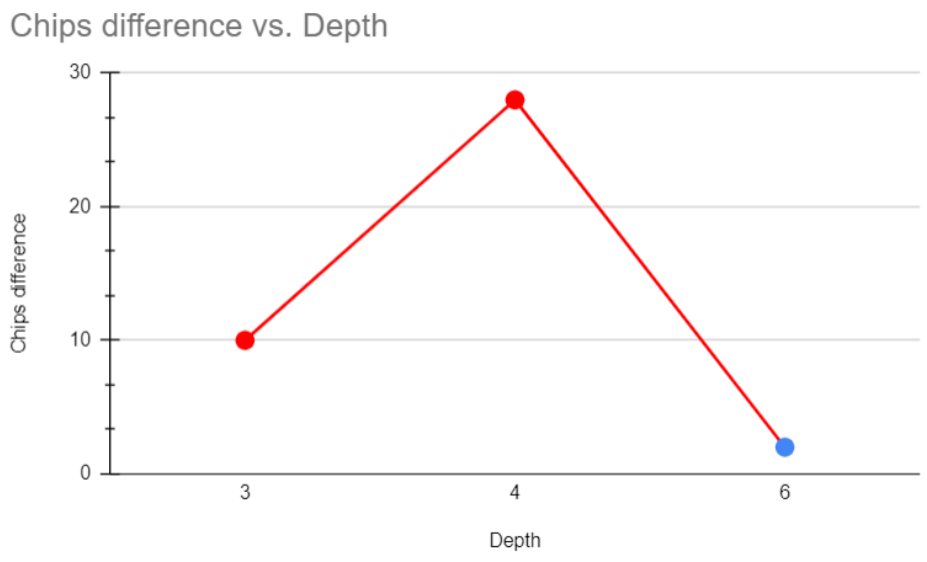
Max Depth 6:



```
5 O O X X X X O X
6 O O X X X X X X
7 X X X X X X X X
Blancas (TERMINATOR) ganan. (HEURISTICA 1)
Cantidad de X: 31
Cantidad de O: 33
```

- **Chips difference:** 2
- **Winner ->** *heuristic_look_for_corners_and_centers*

The following data was plotted, and the following graph was obtained:



Very interestingly, the first heuristic won with the maximum depth at 6 with a little difference of 2 chips. This means that the heuristic's quality may be related to the depth used.

Also it is important to say that the execution time in the last run was about 15 minutes, while the 2nd run used 5 minutes. First run used about 30 seconds.

• Conclusions

- Min-Max with alpha-beta pruning, heuristic and cutoff is by far the best algorithm. Basically, because it is the only one that can run an 8x8 board without expanding a huge and impossible to compute amount of states.
- Time has proven to be a very important factor, it is very annoying to wait 20 seconds or even minutes until you get to play. Min- max with heuristic and pruning gets slow after using more than 4 levels as cutoff limit. And the other versions of min max are pretty slow even with a 4x4 board.
- Min-max with pruning and heuristic plays really good, but its not good enough to beat a skilled human player, at least with the cutoff level that was implemented.