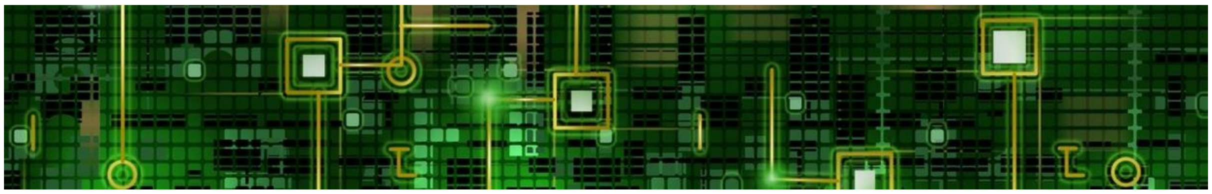
Instituto Tecnológico y de Estudios Superiores de Monterrey

Campus Querétaro



Final Challenge
Unbeatable Tic Tac Toe

Intelligent Systems (TC2011)

Professor:

PhD. Ruben Stranders

Student:

Marco Antonio Reyes Fabela **Enrollment:** A01208429

Justification

Why did I choose this problem?

I chose this problem because, at first, I wanted to do a hardware implementation of a neurological network in a FPGA, however, I underestimated the challenge (especially the algorithm's floating-point implementation) and the time was not enough. Therefore, I decided to implement an unbeatable Tic Tac Toe player that will also be a challenge, since I've always been curious about how AI is programmed in strategy videogames.

Why is my solution useful?

The solution is useful because the processing time is not long and it meets the objective that has been set. The game rules allow that, if both contenders play a perfect game, there will not be another option but a draw.

In order to ensure that it is unbeatable, it has been tested against google Tic Tac Toe player in the 3 existing difficulties as well as 5 human users. The results are listed below:

- Against the Google tic tac toe (easy level): The implementation won 10 out of 10 matches switching the starter turn.
- Against the Google tic tac toe (medium level): The implementation won 8 matches and ended in draw 2 times, switching the starter turn.
- Against the Google tic tac toe (impossible level): The implementation allowed 10 draws out of 10 matches, switching the starter turn.
- Against humans, the implementation won 2 matches and ended in 5 draws.

Thus, the results obtained confirm that the Tic Tac Toe is, until now, unbeatable.

Why is AI a good way of solving the problem

AI may not be the best way to solve this problem due to the level of memory that is required and the processing speed. However, this is an "easy" implementation, so it could be solved with out use minimax algorithm or AI.

This way the code is shorter and is easy to modify, meaning this algorithm could be used as a generic function for any non-cooperative game of two players.

What does my code need to implement a connect 4?

In order to implement a perfect player form using the actual code, it is necessary to change the size of the matrix, the search function when a player wins and the maximum depth allowed before it can be considered as a tie game.

It could also be used as a generic implementation for any possible board size, even for a three-dimensional board. The only limitations will be:

- The other games shall follow the same rules as tic tac toe
- The processing time as well as the required memory may increase for all the possible solutions.

Related problems

A research of other related problems was made and many implementations of Tic tac toe were found using:

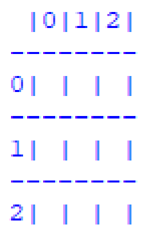
- Python AI 1 (Cruz, 2018)
- Python AI 2 (Pathak, 2017)
- Python AI 3 (omegadeep10, 2018)
- Numpy (Yip, 2015)
- A combination of minimax and numpy in the same implementation (Austin, 2017)
- An implementation without AI, just with reactive decision set (Akyshnik, 2017)
- Connect 4 implementations using a minimax approach (jam1garner, 2017) (Loyola, 2015) (Albertson, 2018)
- 3D Tic Tac Toe implementations (Wu, 2017) (Schee, 2014)

Documentation

Instructions

To run this program the 3.7 version of python (32 bits) will be needed. After the download, the user will be ready to play. It is recommended to use the native Python IDE to run this program.

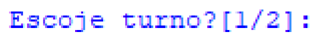
Since the interface settings use Spanish as a default, the commands and the main functionality of the interface will be explained in this language.



```
 |0|1|2|
--------
0| | | |
--------
1| | | |
--------
2| | | |
```

Figure 1: the Board

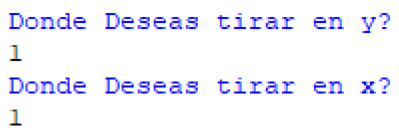
In the Figure 1, the board shows a 3*3 matrix that is numerated in ascendant order from 0,0 in the top left corner to 2,2 in the right bottom corner.



```
Escoje turno?[1/2]:
```

Figure 2

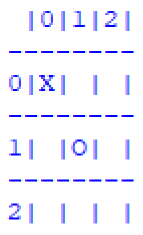
In the Figure 2, the game asks the player which turn they want to choose.



```
Donde Deseas tirar en y?
1
Donde Deseas tirar en x?
1
```

Figure 3

In the Figure 3, the first question asks to the y-coordinate (row) and the second question asks the x-coordinate (column). These values will be used in the current turn.



```
 |0|1|2|
--------
0|X| | |
--------
1| |O| |
--------
2| | | |
```

Figure 4

In the 1,1 coordinate of the matrix, the 'O' represents the Human move, the 'X' represents the AI's answer. The Figures 3 and 4 go on until the end of the match.

Analysis

The implementation was tested against 4 different implementations for the same game, the google tic tac toe, two dofferent implementations using minimax algorithm and one implementation without the use of an AI algorithm.

In the Figure 5, the results of the matches against the Google Tic Tac Toe are shown.

As it can be seen in the graphic, in the Easy level, this implementation wasn't defeated; in Medium level, it won 8 matches and ended in 2 tied matches and in the impossible level, the 10 matches ended in draw.

However, against humans, it won 2 times and ended in tied 8 times.

The conclusion of these results is that the goal of be unbeatable was achieved.

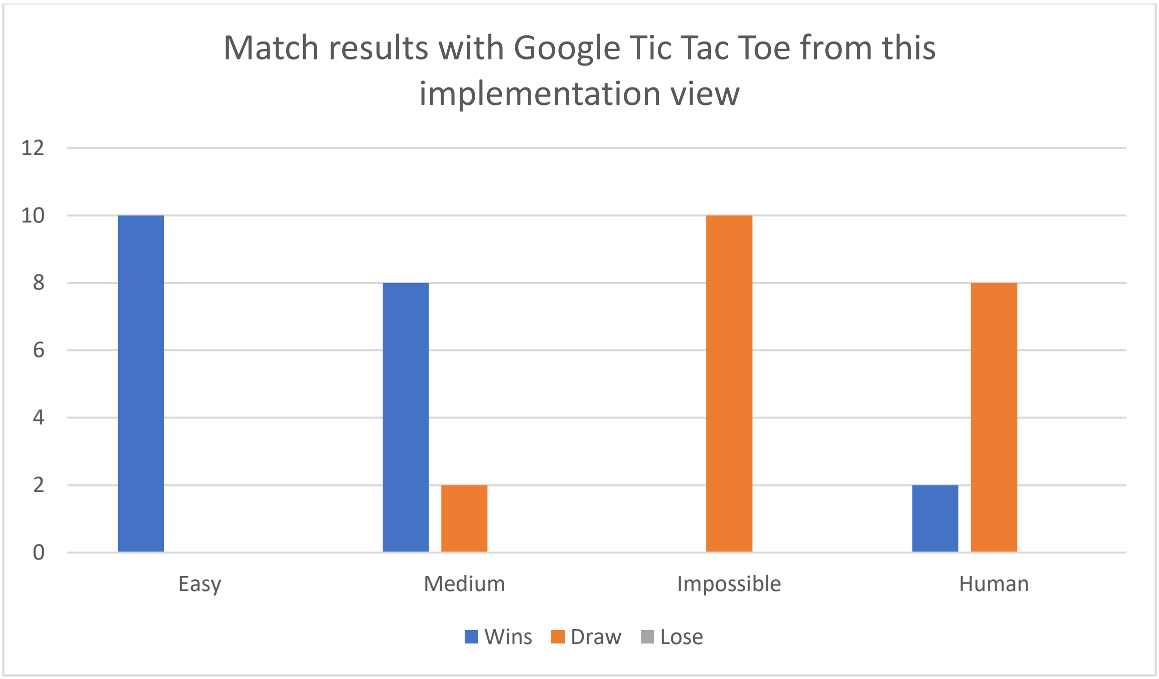


Figure 5

In the next Figure, we can compare the performance against the other 3 implementations. Against the Other python AI 1 (Cruz, 2018), it is possible to see that all matches ended in draw.

Comparing against the Other python AI 2 (Pathak, 2017), it is possible to see that it won all the times, so it is easy to assume that there could be a problem in the minimax implementation of Other python AI 2 since it shouldn't be defeated.

However, against the Other "no AI" implementation (Akyshnik, 2017) it is possible to see that it won 3 times. When the AI implementation started, the game was won. When the "no AI" implementation started, the match ended in a draw.

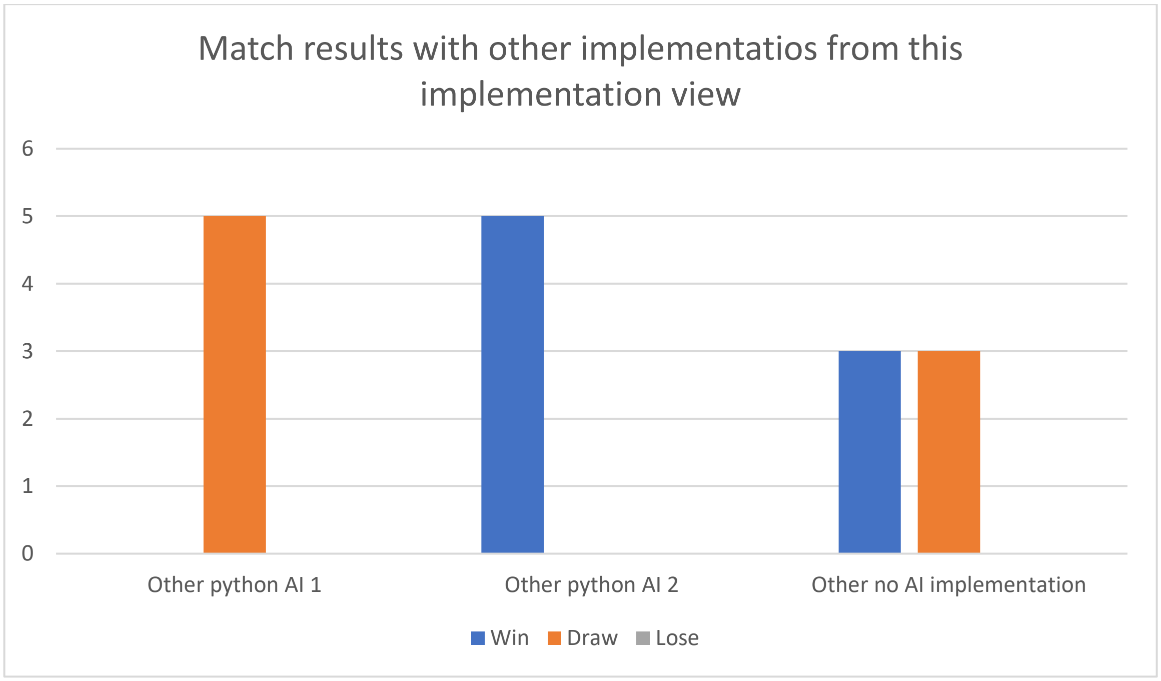


Figure 6

The required processing time is shown in the next Figures (min and max). Of course, this time could change a bit depenfing on the processor's workload.

It is highly remarkable that this implementation was the second fastest. Due to the processing time and the fact that it is unbeatable, it is possible to say that it is the best one of the implementations that were tested.

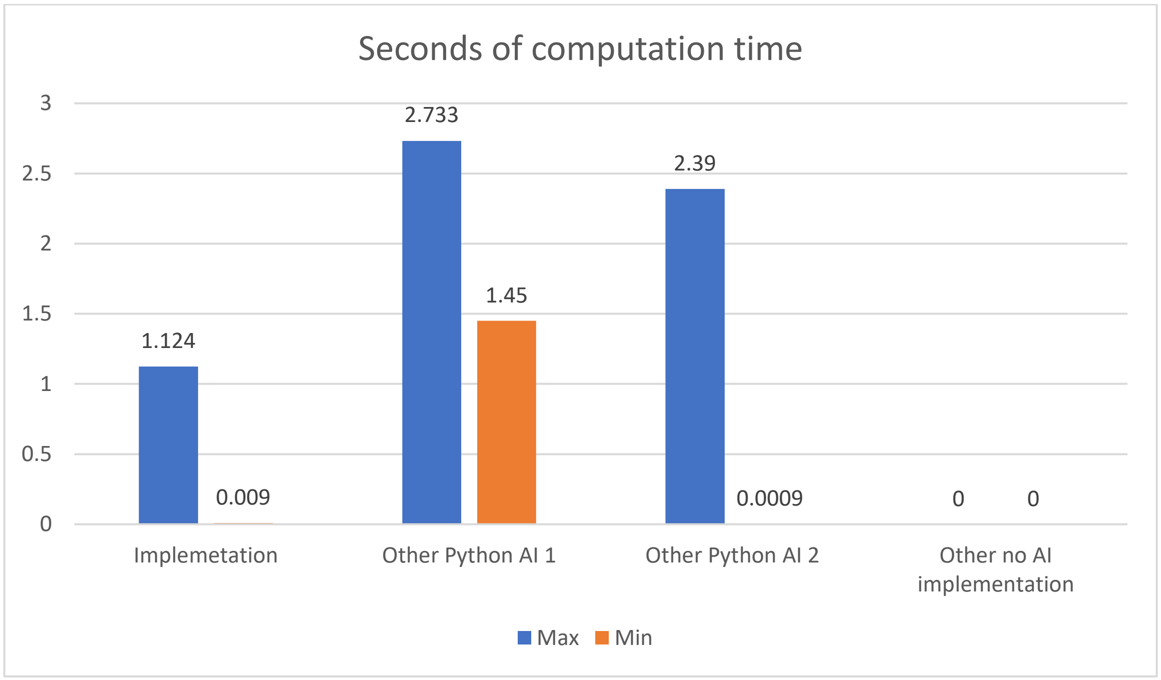

Seconds of computation time
3
2.5
2
1.5
1
0.5
0
1.124
0.009
2.733
1.45
2.39
0.0009
0
0
Implemetation
Other Python AI 1
Other Python AI 2
Other no AI implementation
Max
Min

Figure 7

Referencias

Akyshnik. (6 de 1 de 2017). *TicTacToe in Python3 w/ Simple AI*. Obtenido de codereview: https://codereview.stackexchange.com/questions/151871/tictactoe-in-python3-w-simple-ai

Albertson, K. (10 de April de 2018). *kevinAlbs/Connect4*. Obtenido de github: https://github.com/kevinAlbs/Connect4

Austin, J. (27 de July de 2017). *ja3067/tic-tac-toe-minimax*. Obtenido de github: https://github.com/ja3067/tic-tac-toe-minimax

Cruz, C. (27 de 10 de 2018). *Cledersonbc/tic-tac-toe-minimax*. Obtenido de github: https://github.com/Cledersonbc/tic-tac-toe-minimax/tree/master/py_version

jam1garner. (30 de sep de 2017). *jam1garner/Connect-4-AI*. Obtenido de github: https://github.com/jam1garner/Connect-4-AI

Loyola, N. (21 de Nov de 2015). *nloyolag/connect4-ai*. Obtenido de github: https://github.com/nloyolag/connect4-ai

omegadeep10. (2 de 6 de 2018). *omegadeep10/tic-tac-toe*. Obtenido de github: https://github.com/omegadeep10/tic-tac-toe

Pathak, O. (15 de 11 de 2017). *OmkarPathak/Tic-Tac-Toe-Using-Minimax*. Obtenido de github: https://github.com/OmkarPathak/Tic-Tac-Toe-Using-Minimax

Schee, M. v. (11 de May de 2014). *mevdschee/python-tictactoe*. Obtenido de github: https://github.com/mevdschee/python-tictactoe

Wu, J. (28 de March de 2017). *JonathanWu1120/3d-tic-tac-toe*. Obtenido de github: https://github.com/JonathanWu1120/3d-tic-tac-toe

Yip, G. (12 de 11 de 2015). *geoffreyyip/numpy-tictactoe*. Obtenido de github: https://github.com/geoffreyyip/numpy-tictactoe

Appendix

```
import itertools
import copy
import random

def checarBoard(m,value):
    for x in range(0,3):
        if m[x][0]== value and m[x][1]== value and m[x][2] ==
value:
            return 1
    for y in range(0,3):
        if m[0][y]== value and m[1][y]== value and m[2][y] ==
value:
            return 1
    if m[0][0]== value and m[1][1]== value and m[2][2] ==
value:
        return 1
    if m[0][2]== value and m[1][1]== value and m[2][0] ==
value:
        return 1
    return 0

def eva(m):
    if checarBoard(m,'X'):
        return 1
    if checarBoard(m,'O'):
        return -1
    return 0

def printBoard(m):
    p=" |0|1|2|"
    l="--------"
    print(m)
    print(p)
    for i in range(0,3):
        print(l)
        pri= str(i)+"|"
        for c in range(0,3):
            if '0' != m[i][c]:
                pri+= str(m[i][c])+"|"
            else:
                pri+= " "+"|"
        print(pri)

def changeBoard(m,x,y,value):
    if m[x][y]=='0':
        m[x][y]=value
        return 1
    return 0

def validarResp():
    x=-1
    while x < 0 or x>=3:
        print("Donde Deseas tirar en y?")
        x=int(input())
    y=-1
    while y < 0 or y>=3:
        print("Donde Deseas tirar en x?")
        y=int(input())
    return x,y

def turnoHumano(m):
    printBoard(m)

    x,y=validarResp()
    valido=changeBoard(m,x,y,'O')
    #print(valido)
    while valido == 0:
        print("Movimiento NO valido Casilla ocupada")
        #print(x)
        #print(y)
        x,y=validarResp()
        valido=changeBoard(m,x,y,'O')
    if checarBoard(m,'O'):
        print("Conseguiste lo imposible le ganaste al jugador
perfecto de TicTacToe!!!!")



def celdasValidas(m):
    celdas=[]
    for x in range(0,3):
        for y in range(0,3):
            if m[x][y] == '0':
                celdas.append([x, y])
    return celdas

def minimax(m, deep, flag):
    if flag == 1:
        final = [-10,-1,-1]
    else:
        final =[10,-1,-1]
    if deep==0 or abs(eva(m)):
        return [eva(m),-1,-1]
    for move in celdasValidas(m):
        x,y = move

        if flag == 1:
            changeBoard(m,x,y,'X')
        else:
            changeBoard(m,x,y,'O')
        punt=minimax(m,deep-1,-flag)
        m[x][y] = '0'
        punt[1]=x
        punt[2]=y
        if flag==1:
            if punt[0] > int(final[0]):
                final= punt
        else:
            if punt[0]< int(final[0]):
                final= punt
    return final

def IAturno(m):
    deep=celdasValidas(m)
    if len(deep) == 0:
        return 0
    print(len(deep))
    if len(deep) == 9:
        x = random.choice([0,1,2])
        y = random.choice([0,1,2])
    else:
        holiwis = minimax(m,len(deep),1)
        s,x,y=holiwis
    changeBoard(m,x,y,'X')
```

```
def funcion(m,prof,path,selec):
    costo=0
    path=[]
    lis ={}
    dos=2
    while len(celdasValidas(m))>0 and (not checarBoard(m,'X')
and not checarBoard(m,'X')):

        if selec == '2':
            IAturno(m)
            selec=0
        turnoHumano(m)
        c=celdasValidas(m)
        IAturno(m)
    printBoard(m)
    if checarBoard(m,'X') == 1:
        print("PERDISTE CONTRA EL MEJOR JUGADOR NO HAY DE QUE
AVERGONZARSE")
    else:
        if checarBoard(m,'X') == 1:
            print("NO SE POR QUE LO ANOTO SI NUNCA NADIE LO
VERA")
        else:
            print("HICISTE LO MEJOR POSIBLE")
    m=[['0','0','0'],['0','0','0'],['0','0','0']]


m=[['0','0','0'],['0','0','0'],['0','0','0']]
printBoard(m)
prof=0
Estado=0
```

```
path=[]
while 1:
    m=[['0','0','0'],['0','0','0'],['0','0','0']]
    selec = input('Escoje turno?[1/2]: ')
    funcion(m,prof,path,selec)
```