Tic-Tac-Toe

Hugo Veríssimo

February 3, 2024

VER < ! – aka comentarios

Introduction

Tic-tac-toe is a paper-and-pencil game for two players who take turns marking the spaces in a three-by-three grid with X or O. The player who succeeds in placing three of their marks in a horizontal, vertical, or diagonal row is the winner.

O	X	X
X	X	O
O	O	X

In this paper, I want to use my programming skills to dig into the game and learn more about it – things like figuring out how many possible games there are and looking at the stats behind it all.

In pursuit of this exploration, I will harness the capabilities of Python, particularly relying on the Pandas and NumPy libraries.

```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
```

besides that I will be using some code that will not appear here but is in the respetcice file inside the repository on github (link)

```
import games_generator as gg
import statistics_truegames as stg
```

NOTAS NOTAS NOTAS NOTAS NOTAS NOTAS

0	1	2
3	4	5
6	7	8

melhorar comentarios e “ “ ” em games_generator (?)

9! and number into equation aka *dolarsign*

How many games

the first approach will be just the 9! games, which means, player1 plays and then playr2 and so on until all 9 squares are fullfied.

for this i will be using numpy since each game will be represented by a array with nine elements (repseenig each square) and each of this elemnt will be a int representing the move number that fills that square

```
# import games_generator as gg

gg.ALL_tic_tac_toe("attachment(1)_alltictactoe.csv")
```

```
pd.read_csv("attachment(1)_alltictactoe.csv", header = None)
```

```
##         0  1  2  3  4  5  6  7  8
## 0       1  2  3  4  5  6  7  8  9
## 1       1  2  3  4  5  6  7  9  8
## 2       1  2  3  4  5  6  8  7  9
## 3       1  2  3  4  5  6  9  7  8
## 4       1  2  3  4  5  6  8  9  7
## ...    .. .. .. .. .. .. .. .. ..
## 362875  7  9  8  6  5  4  3  2  1
## 362876  8  7  9  6  5  4  3  2  1
## 362877  9  7  8  6  5  4  3  2  1
## 362878  8  9  7  6  5  4  3  2  1
## 362879  9  8  7  6  5  4  3  2  1
##
## [362880 rows x 9 columns]
```

as we can see, we have now the 9! games, lets check a random game for further understaging of the dataframe

```
##        0  1  2  3  4  5  6  7  8
## 21125  1  5  3  6  4  2  9  8  7
```

this game can be translated into this (assuming player1 is X)

0	1	2
3	4	5
6	7	8

→

X	1	2
3	4	O
6	7	8

→

X	1	X
3	O	O
6	7	8

→

X	X	X
O	O	O
6	7	8

→

X	X	X
O	O	O
6	O	X

→

X	X	X
O	O	O
X	O	X

note that the two players succeed placing three of their marks in a horizontal row (row 1 and 2) so they both won the game? of course not, the first one completing the row won the game whhich would stop the game at that move making all the further moves irrelvant scenarios. This means there aren't 362880 (9!) games

since we now have all the possible ways to fullfie the squares lets remove the moves that happen after one of the player wins

```
# import games_generator as gg

gg.TRUE_tic_tac_toe("attachment(2)_truetictactoe.csv")
```

```
pd.read_csv("attachment(2)_truetictactoe.csv", names = (list(range(9))) + ["winner"])
```

```
##         0  1  2  3  4  5  6  7  8  winner
## 0       0  0  0  0  2  4  1  3  5       1
## 1       0  0  0  0  2  4  1  5  3       1
## 2       0  0  0  0  2  4  3  1  5       1
## 3       0  0  0  0  2  4  3  5  1       1
## 4       0  0  0  0  2  4  5  1  3       1
## ...    .. .. .. .. .. .. .. .. ..     ...
```

```
## 255163  9  8  7  6  5  2  4  3  1       1
## 255164  9  8  7  6  5  3  2  1  4       0
## 255165  9  8  7  6  5  3  4  1  2       0
## 255166  9  8  7  6  5  4  2  1  3       1
## 255167  9  8  7  6  5  4  2  3  1       1
##
## [255168 rows x 10 columns]
```

as we can see we got the possiblities down to 255168, a aproximatley 29.7% decrase from the 9! possiblities.

note that i also added a new columns with a value to represent who won the game. the value in the "winner" columns can be 1 or 2 if player 1 or 2 won respectivley or 0 if the games ends up in draw.

statistics HERE HERE HERE nao te esquecas de nomear chuncks

about the attachment(2)

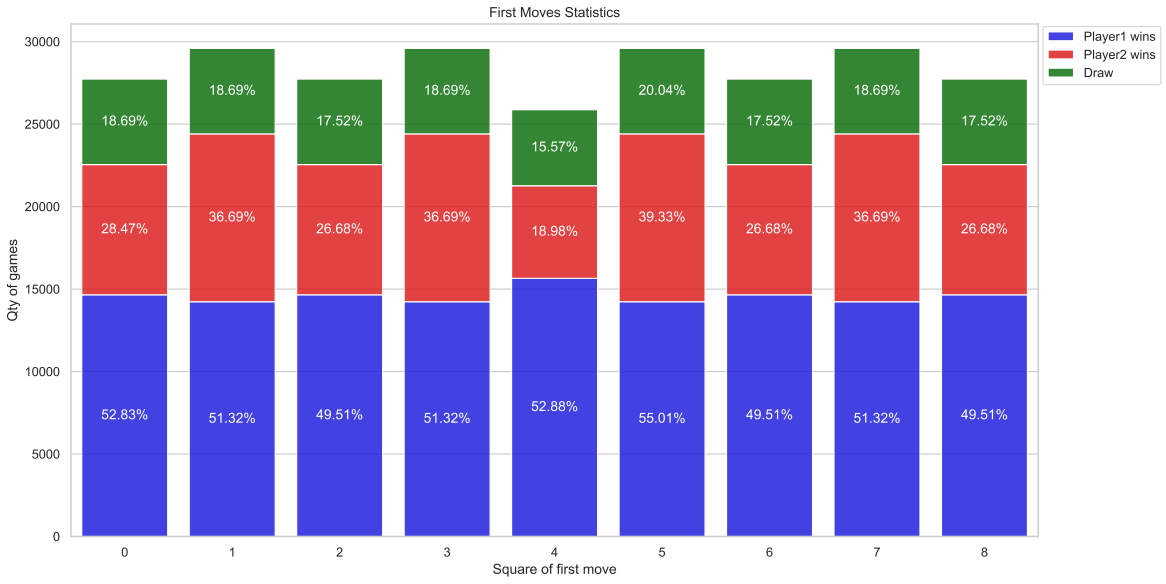
```
true_games = pd.read_csv("attachment(2)_truetictactoe.csv",
                         names = (list(range(9))) + ["winner"])
```

% of wins

quantos começam em cada sitio e % de vitoria se começar lá ESCREVER TOTAL EM CIMA DE CADA BARRA ?

```
# import statistics_truegames as stg

stg.first_moves_statistics(15, 8)
```



how many final games using only X and O

```
# import pandas as pd

def is_it_player1_or_2(x):
    return 0 if x == 0 else (1 if x % 2 == 1 else -1)
```

```
len(true_games.map(is_it_player1_or_2).drop_duplicates())
```

```
## 958
```

use seaborn for graphs

quantidade de V1, V2 e draww

```
len(true_games[true_games["winner"] == 1]), len(true_games[true_games["winner"] == 2]), len(true_games[
```

```
## (131184, 77904, 46080)
```

symmetries

using numpy check symmetries and all that

```
# import numpy as np

def SYMMETRIES_tic_tac_toe(n = 3, savefile = True):
    """
    simetrias bla bla
    n = 3 -> 3 by 3 square
    """

    ###########################################################################
    def _symmetry_squares(n = n):
        """ ex.: if there's vertical symmetry which squares are valid moves """
        middle_line = n//2 + n%2

        vertical = []
        for row in range(n):
            for col in range(middle_line):
                vertical.append(row * n + col)

        horizontal = []
        for row in range(middle_line):
            for col in range(n):
                horizontal.append(row * n + col)

        diagonal = []
        anti_diagonal = []
        for row in range(n):
            for col in range(n):
                if row <= col:
                    diagonal.append(row * n + col)

                if col <= (n-1) - row:
                    anti_diagonal.append(row * n + col)

        return vertical, horizontal, diagonal, anti_diagonal
    ###########################################################################

    ###########################################################################
    symmetry_squares = _symmetry_squares(n)
```

```
    def _valid_moves(TTT, n = n, symmetry_squares = symmetry_squares):
        middle_line = n//2 + n%2

        # define all 4 axes of symmetry and verify them (True or False)
        vertical = np.all([np.transpose(TTT)[i] == np.transpose(TTT)[-i-1] for i in range(middle_line)]
        horizontal = np.all([TTT[i] == TTT[-i-1] for i in range(middle_line)])
        diagonal = np.all(TTT == np.transpose(TTT))
        anti_diagonal = np.all(np.fliplr(TTT) == np.transpose(np.fliplr(TTT)))
        axes_of_symmetry = vertical, horizontal, diagonal, anti_diagonal

        # which are the valid moves
        all_possible_moves = set(range(n*n))
        invalid_moves = []
        for i in range(4):
            if axes_of_symmetry[i]:
                invalid_moves_i = all_possible_moves - set(symmetry_squares[i])
                invalid_moves += invalid_moves_i
        valid_moves = all_possible_moves - set(invalid_moves)

        return list(valid_moves)
    ##############################################################################

    ##############################################################################
    def _its_a_win(TTT, n = n):
        lines = np.vstack([TTT, np.transpose(TTT), np.diag(TTT), np.diag(np.fliplr(TTT))])
        # if 0 -> 0 ; if impar -> 1 ; if par -> -1
        X_vs_O = np.where(lines == 0, 0, np.where(lines % 2 == 1, 1, -1))
        sums = np.sum(X_vs_O, axis=1)
        return n in np.abs(sums)
    ##############################################################################

    TTT_moves = np.zeros(n**2)
    TTT_symmetries = []

    ongoing_games1 = [TTT_moves]
    move = 1
    while move < n**2:
        ongoing_games2 = []
        for game in ongoing_games1:
            # verify possible moves
            valid_moves = _valid_moves(game.reshape(n,n))
            # play them
            for valid_move in valid_moves:
                if game[valid_move] != 0:
                    pass
                else:
                    game[valid_move] = move
                    # verify if its a winning move
                    if _its_a_win(game.reshape(n,n)):
                        TTT_symmetries.append(game.copy())
                    else:
                        ongoing_games2.append(game.copy())
```

```
                    game[valid_move] = 0
        # next move
        move += 1
        ongoing_games1 = ongoing_games2

    # last move
    move = n**2
    for game in ongoing_games1:
        # was move (n**2 - 1) a winning move ?
        if _its_a_win(game.reshape(n,n)):
            TTT_symmetries.append(game.copy())
        else:
            TTT_symmetries.append(np.where(game == 0, n**2, game).copy())

    if savefile:
        np.savetxt("attachment(3)_symmetriestictactoe.csv", TTT_symmetries, delimiter=",", fmt="%d")
        return "attachment(3) saved", len(TTT_symmetries)
    else:
        return TTT_symmetries, len(TTT_symmetries)


SYMMETRIES_tic_tac_toe(n = 3)
```

```
## ('attachment(3) saved', 31896)
```

```
pd.read_csv("attachment(3)_symmetriestictactoe.csv", header = None)
```

```
##        0  1  2  3  4  5  6  7  8
## 0      1  2  4  3  0  0  5  0  0
## 1      1  2  0  3  4  0  5  0  0
## 2      1  2  0  3  0  4  5  0  0
## 3      1  2  0  3  0  0  5  4  0
## 4      1  2  0  3  0  0  5  0  4
## ...   .. .. .. .. .. .. .. .. ..
## 31891  9  2  8  6  1  7  4  3  5
## 31892  8  2  7  9  1  6  4  3  5
## 31893  9  2  7  8  1  6  4  3  5
## 31894  8  2  9  7  1  6  4  3  5
## 31895  9  2  8  7  1  6  4  3  5
##
## [31896 rows x 9 columns]
```

bla bla 31896

```
# ver como cresce aquilo das simetrias com n += 1

"""
lista = []
for i in range(1, 4):
    lista.append(SYMMETRIES_tic_tac_toe(n = i, savefile=False)[1])
    print(str(i) + " OK")
    #print using {} la no meio a dizer tp n por n square ...
lista
"""
```

```
## '\nlista = []\nfor i in range(1, 4):\n    lista.append(SYMMETRIES_tic_tac_toe(n = i, savefile=False)
```