Graphs #1: Tic-Tac-Toe
Oct. 18, 2016

The Game of Tic-Tac-Toe

Tic-tac-toe is a classic paper-and-pencil game where players alternate placing Xs and Os on a 3x3 grid. Today, we will explore **state space representation**, **transitions**, and how **graphs** are all over solving a game of tic-tac-toe.

State Space

A state space is a mathematical representation of a physical system. For a given representation, a state space must be able to represent all possible combination of possibilities of the physical system.

For tic-tac-toe, we will use a state space to represent the board of a tic-tac-toe game. Our state space will be:

- An array of length nine (9) of one-character strings
- Each character is either an 'X', 'O', or '-'
- Each character represents one square on the board
- The board/string encoding is done in the following way:

X	X	O
X	O	
O	O	X

➔

[0]	[1]	[2]
[3]	[4]	[5]
[6]	[7]	[8]

Based on the figure above, the current state in our state space is:
`["X", "X", "O", "X", "O", "-", "O", "O", "X"]`

Q1: Which of the following states are winning?

(a): `["X", "X", "O", "X", "O", "-", "O", "O", "X"]`

(b): `["X", "O", "X", "O", "X", "O", "-", "-", "-"]`

(c): `["X", "X", "-", "O", "O", "-", "X", "O", "-"]`

Q4: What set of indexes must be checked to determine if a state is a winning state?

Q5: Complete the following code to place the indexes into a list of list of numbers:

```
setOfIndexes = [



]
```

Q6: Assuming that `board` contains the string of our current state, the following code extracts the character at the specified index for each index in our set of indexes. What if-statements need to be written to check if the state is a winning state and return 1 if "X" has won, -1 if "O" has won, and zero if no one has won?

```
for indexes in setOfIndexes:
  p1 = board[ indexes[0] ]
  p2 = board[ indexes[1] ]
  p3 = board[ indexes[2] ]

  if


  if
```

Code It:

Complete the getBoardState function based on the coding done above in **compute.py** inside of demo_tictactoe. *You can check your results by running py/compute.py from the command line.*

Traversing the Game States
In tic-tac-toe, the game changes state by the current player (either X or O) marking their symbol in an empty square (represented by "-").

To allow us to computationally explore the tic-tac-toe game, we must have a function that generates the next game states from a given state. *This function should only concern itself about generating the immediate next state (not all future states).*

Consider the following state:

X	X	O
	X	
O		

Current board state (as shown above):
```
["X", "X", "O", "-", "X", "-", "O", "-", "-"]
```

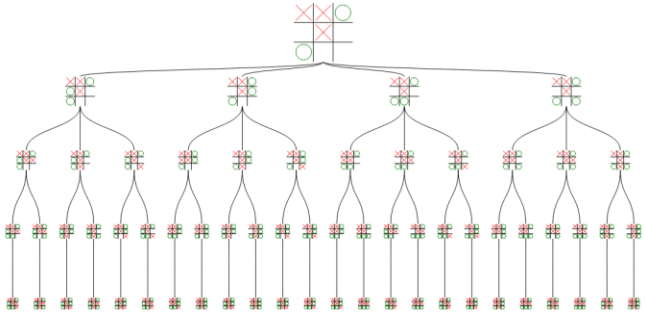
Next board states (with "O" as the current player, or `turn`):
```
["X", "X", "O", "O", "X", "-", "O", "-", "-"]
["X", "X", "O", "-", "X", "O", "O", "-", "-"]
["X", "X", "O", "-", "X", "-", "O", "O", "-"]
["X", "X", "O", "-", "X", "-", "O", "-", "O"]
```

Q7: What algorithm can be used to create a list of new states?

Q8: Complete the following code to find a list of next states, given a state stored in `board` and the current player stored in `turn`.

```
def getNextStates(board, turn):
```

Finding Every Move
Suppose we want to create a tree of every move, not just the *next move*? For example, the following **tree** shows every transition of game states from our example board:



Our Algorithm:

Activity 8 (Mid-week Homework):
Use NLP to label the sentiment of each node in your story form Activity 7. We'll use this data in class on Thursday to find various paths through your story.