Beyond just the Game

Specifications (Tic-Tac-Toe Game)

- Try to be as functional as possible without being stupid (OOP is more natural in this application... but I think a more functional/mathematical solution could be more efficient) This makes it an interesting challenge (def will need mutability, but limit when possible)

- Operations should be in constant time. Have them be straight-up numerical operations when possible

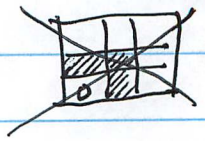
- I don't want the most obvious solution for checking victory conditions (each point recursively checking its neighbors). Seems inefficient/inelegant

- Don't use Ruby — already used for FizzBuzz and I want to play with something else => given some OOP/mutability is probably unavoidable, likely use Python (has nice dict, etc. too)

Problems

Victory Condition

Figure out if a player has won ⇒ don't use recursive checking of neighbors



no ok here, gets recursively more painful w/ larger grids (e.g. connect 4 or Go)

Collision / occupied cell

Unless I can figure out an abstract mathematical representation of the whole grid, probably need OOP to track state of each grid cell

VI

Probably just go for text-based, doesn't use frameworks and is also faster to program

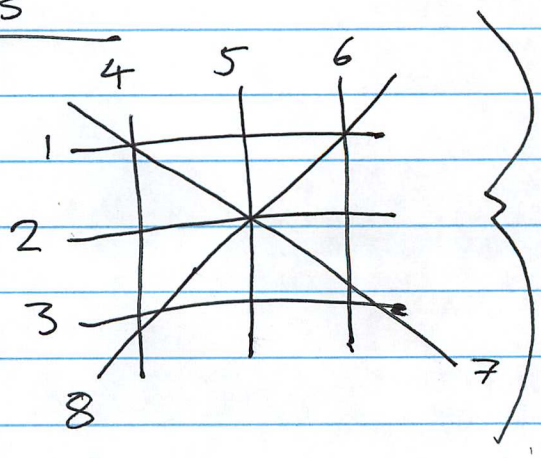
Print selectable/named cells?

a b c
d e f ⇒ d e f
g h i g h i

Notes: If I do this correctly / how I want, logic should be easily expandable to Connect 4. Need more thought for Go (much more/ too ambitious here in timeframe)

Distributing Points

(1,4,7)	(1,5,0)	(1,6,8)
(2,4,0)	(2,5,7,8)	(2,6,0)
(3,4,8)	(3,5,0)	(3,6,7)



Center cell is a different solution — linked lists for creating arrays for storing rows? (to make them variable in size for two-intersection & grid points vs. 3-intersection (4 for center)

Allocating array indexes to the grid

Is there a mathematical ~~algo~~ algorithm for creating the grid's references (vs. manually coding them? (would be useful for Connect 4 as well)

Probably based on the row/column size

Cells are always on a row or col, but not necessarily on a diagonal

grid rows = cell row

grid col = cell col + 3

grid diag = ?

Added cells		
5	6	7
6	7	8
7	8	9

algorithmically determine diagonals

Algorithmically Determining Diagonals

All cells are on a row & column by definition. Not all cells are on a diagonal

Add r+c

	4	5	6
1	5	6	7
2	6	7	8
3	7	8	9

All odd #'s, problem is determining middle cell. Hopefully there's a mathematical solution vs. defining a special case for middle cell...

Add r+c (not consecutive) #s for row→col

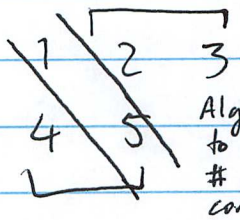
	1	2	3
1	2	3	4
2	3	4	5
3	4	5	6

All evens, same problem

Consecutive (count by col)

1	2	3
4	5	6
7	8	9

All odd, middle is finally diff than top-right to bottom-left diag.



1 = 1

1+4 = 5

Algorithm to determine # of consecutive cells to move across

[max(col) - 1 + 2

to account for 1st cell

to account for next row, second col

Function specifying this grid

= col + (row - 1) · 3

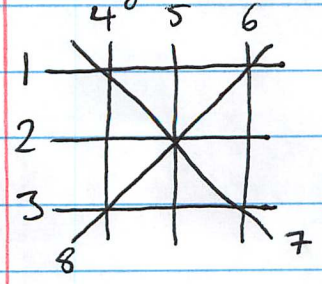
max cols

Function that determines middle cell (for any # of r & c)

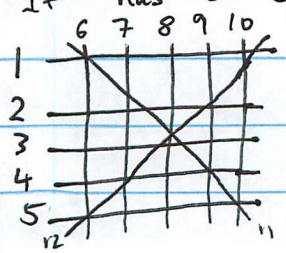
= (max(r|c) + 1) / 2

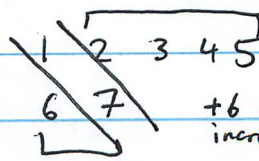
odd

Victory Routes



If has 5 x 5





+6 to increment diagonal

5 - 1 + 2 = 6 seems to be general

Design

Text-based, turn-based (X, O : X goes 2nd)
Make printing after each input clean
Selections should be intuitive & clear
(even w/o point & click)

Choosing:

```
$>
a  b  c
d  e  f
g  h  i

Player X turn. Pick cell to tic (q to quit):
```

Main:

```
* Tic-Tac-Toe Game! *

Menu selection
1) New Game
2) Quit
```

Game Over:

```
$>
a  b  X
O  X  f
X  O  i
Player X wins! (n for new game, q to quit)
```