Daily Coding Problem #39

Problem

This problem was asked by Dropbox.

Conway's Game of Life takes place on an infinite two-dimensional board of square cells. Each cell is either dead or alive, and at each tick, the following rules apply:

- Any live cell with less than two live neighbours dies.
- Any live cell with two or three live neighbours remains living.
- Any live cell with more than three live neighbours dies.
- Any dead cell with exactly three live neighbours becomes a live cell.

A cell neighbours another cell if it is horizontally, vertically, or diagonally adjacent.

Implement Conway's Game of Life. It should be able to be initialized with a starting list of live cell coordinates and the number of steps it should run for. Once initialized, it should print out the board state at each step. Since it's an infinite board, print out only the relevant coordinates, i.e. from the top-leftmost live cell to bottom-rightmost live cell.

You can represent a live cell with an asterisk (*) and a dead cell with a dot (`.`).

Solution

This is a straightforward implementation problem, so your solution may differ. Since our board is infinite, we can't create a matrix that represents our whole board.

Instead, we'll represent each cell simply as a pair of cartesian coordinates (row, col). In this solution, we keep the set of cells as a property on our class. Each tick, we create a new set of cells that represents the next generation. We pretty much have to do this so that changing the board doesn't affect the future cells we process from the current generation.

We look at each live cell, compute the number of neighbours for each one, and preserve it according to the rules.

Similarly, we look at all the neighbouring cells of all the live cells, since any of them could potentially become alive due to rule #4. If any of them have exactly 3 neighbours, then we should add them to the set of new cells.

For printing the board, we need to find the top-leftmost cell and the bottom-rightmost cell. These are our boundaries for the board. Then we can print out each row and cell one by one and checking if the current spot is in our set of cells.

It's useful to create some helper functions here. In our case, we have:

- `get_number_of_live_neighbours`
- `get_neighbouring_cells`
- `get_boundaries`

```
class GameOfLife:
    def __init__(self, n, cells=set()):
        # Each cell will be a tuple (row, col)
```

```
        self.cells = cells
        for _ in range(n):
            self.print_board()
            self.next()

    def get_number_of_live_neighbours(self, row, col):
        count = 0
        for cell_row, cell_col in self.cells:
            if abs(cell_row - row) > 1:
                continue
            if abs(cell_col - col) > 1:
                continue
            if cell_row == row and cell_col == col:
                continue
            count += 1
        return count

    def get_neighbouring_cells(self, row, col):
        return set([
            (row - 1, col - 1),
            (row, col - 1),
            (row + 1, col - 1),
            (row - 1, col),
            (row + 1, col),
            (row - 1, col + 1),
            (row, col + 1),
            (row + 1, col + 1),
        ])

    def next(self):
        new_cells = set()
```

```
        # Go through each cell, look for neighbours, decide whether to append to new list
        for row, col in self.cells:
            num_of_neighbours = self.get_number_of_live_neighbours(row, col)
            if 2 <= num_of_neighbours <= 3:
                new_cells.add((row, col))

        potential_live_cells = set()
        for row, col in self.cells:
            potential_live_cells = potential_live_cells.union(self.get_neighbouring_cells(row, col))
        potential_live_cells = potential_live_cells - self.cells

        # Go through each potential live cell, get the number of neighbours, and add if = 3
        for row, col in potential_live_cells:
            num_of_neighbours = self.get_number_of_live_neighbours(row, col)
            if num_of_neighbours == 3:
                new_cells.add((row, col))

        self.cells = new_cells

    def get_boundaries(self):
        top = min(self.cells, key=lambda cell: cell[0])[0]
        left = min(self.cells, key=lambda cell: cell[1])[1]
        bottom = max(self.cells, key=lambda cell: cell[0])[0]
        right = max(self.cells, key=lambda cell: cell[1])[1]
        return top, left, bottom, right

    def print_board(self):
        top, left, bottom, right = self.get_boundaries()
        print('----------------------------------------')
        for i in range(top, bottom + 1):
            for j in range(left, right + 1):
```

```
            if (i, j) in self.cells:
                print('*', end='')
            else:
                print('.', end='')
        print('')
print('---------------------------------------')
```