Homework #6: The Meaning of `life` **Due: 11/20**

1. [100 points] Using a `tkinter Canvas` for a "board", implement a Python program *pylife* which is an interactive version of John Conway's famous cellular automaton game "Life". For more information about this game, take a look at

   `http://www.math.com/students/wonders/life/life.html`

   The rules of Life are very simple. (You heard it here first! `B-)`) You start with a rectangular array of cells. Each cell is either "alive" (black) or "dead" (white). Initially, all cells are dead.

   Every cell has eight neighbors. To make a new generation, apply these rules to each cell:

   (a) If the cell is alive and has fewer than two or more than three living neighbors, the cell dies (from loneliness or overcrowding, respectively).

   (b) If the cell is dead and has exactly three living neighbors, the cell revives.

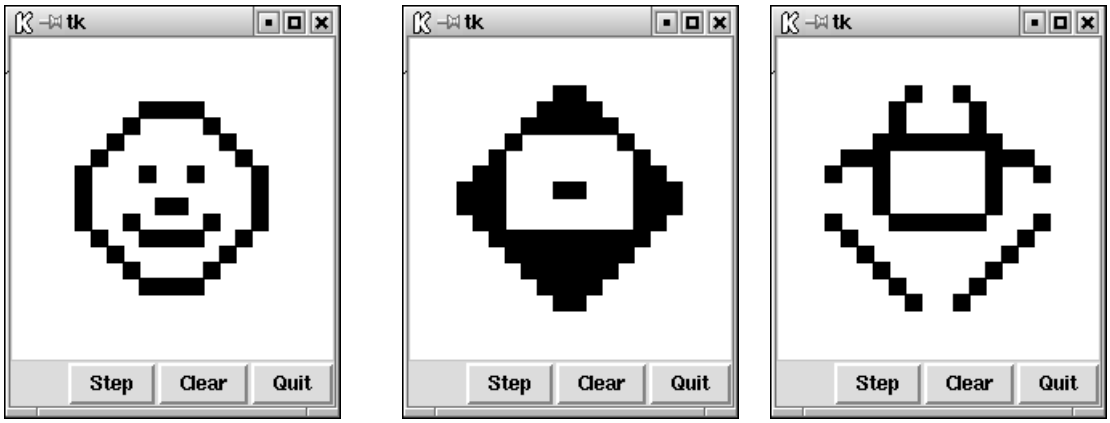
   Build a class `CellArray` to implement these rules. Interpret "living" as 1 and "dead" as 0. Add a `step()` method to the class to perform a single iteration of these rules. The neighbors of a cell are the eight cells that surround it, except that on the border, neighbors that fall outside the array are considered dead. (We'll call these "zero boundary conditions".) `CellArray` should not itself make any `tkinter` calls (or call anything that does).

   `pylife` should have a GUI that consists of:

   - a `CellCanvas` widget (a child of `Canvas`) that allows the user to enter and modify a `CellArray` (initially all dead) and to see successive generations
   - a `Step` button to replace the current generation with the next
   - a `Clear` button to erase the canvas and start (again) with all cells dead
   - a `Quit` button to exit the application

   The user may click on the `CellCanvas` widget to kill and resurrect (click toggles the cell status) cells at any time.

   Here are three successive generations of the "happy face" initial configuration. Your program should produce these:



generation 0 (you input this) generation 1 generation 2

20×20 cells is a good size for the `CellCanvas`. Smaller than that doesn't show interesting effects and larger that that is tedious to enter.

Up to this point is worth 90 points. The remaining 10 points come from your choice of one of the following enhancements:

- Add some kind of functionality (including incorporating it into the GUI) for saving a configuration to and restoring a configuration from a file. (Include any interesting configurations with your submission.)
- Add a `Run` button to automatically show successive generations. (Use a `tkinter` timer to keep it from going too fast.)
- Add a set of playback-like controls, buttons for
  - `Reverse` (`Run` backward)
  - `Step Backward`
  - `Step Forward` (replacing `Step`)
  - `Forward` (`Run` forward)

  use the usual arrow key label icons, if you like, and a `Slider` to manually see any generation created so far.
- Add a button to change the boundary conditions from zero to "wraparound" (aka "toroidal"): Border cells wrap around to the other side, so right-hand neigbors of the cells in the rightmost column are the cells in the leftmost column, etc.
- Assume an infinite (zero boundary conditions) canvas and expand and contract the user's view of it (always keeping it above the initial size) so that all cells are always visible.
- Any other enhancement approved by the instructor.

Be sure to include a `README` file in your tarball describing which of these you are implementing. You may implement additional functionality for extra credit, up to a maximum of 120 points.