gof Documentation

Release 0.1.0

julien tayon

Sep 27, 2017

Contents

- source : https://github.com/jul/game_of_life/
- ticketting : https://github.com/jul/game_of_life/issues
- docs : http://gof.readthedocs.org

Warning:

Please don't

- from module import *
- if you do so (which is bad) don't forget to define __all__

I guess I have no excuse others than lazyness. But I find it so easier to use.

CHAPTER 1

Pathetic excuse #1

I am doing a conway game of life's Domain Specific Language, and this is the easiest way to do it.

CHAPTER 2

Pathetic excuse #2

Pretty easy to use and code :)

CHAPTER 3

Pathetic excuse #3

I am a rebel.

CHAPTER 4

Game of life is a serious game

Contents:

Direct in the fun

Quick install

First install the package either from github:: git clone git://github.com/jul/game_of_life.git

And do what you have to do :)

Or:: pip install gof

I made a package to write less docs.

To dive directly in the core of the topic:: python -i -mgof.demo

You'll have a *pseudo* animation (could work on windows, but I am lazy), of a cellular automata. But this is not fun, you have to manipulate to really have fun.

Since you use *python -i* at the end of the demo you are left with an interactive session

Quick tour

Let's use all the functions:

First seeing is believing::

```
>>> print grid
 '    '
_..............
 ..............
 ..............
 ..............
 ..............
-...X..........
 ...X..........
 ...X..........
 ..............
 XX...XXX.....X
-..............
 ...X..........
 ...X..........
 ...X..........
 ..............
-..............
```

So I may have overloaded __str__ so that you have a nice output. If you want to know more about the grid object:

```
>>> help(grid)
```

It does not tells you : grid.size_x, grid._size_y are attributes where the dimension of the matrix are stored.

Now, you want to clean the matrix, to play:

```
>>> bleach(grid, 20,40)
>>> print grid
```

This should show you a nice empty grid.

Before you play the game of life, you want to draw patterns on your grid. (The one I defined are not exhautive, you can draw your own.) Let's add a glider, an oscillator, and a fixed block:

```
>>> at(grid, 10,20, glider)
>>> at(grid, 5,5, oscillator)
>>> at(grid, 15,25, still)
```

and see the result ::

```
>>> print grid
 '    '    '    '    '    '    '    '
-........................................
 ........................................
 ........................................
 ........................................
 ........................................
-.....XXX................................
 ........................................
 ........................................
 ........................................
 ........................................
-....................X.X.................
 ......................X.................
 .....................XX.................
 ........................................
 ........................................
-.........................XX.............
 .........................XX.............
 ........................................
 ........................................
 ........................................
```

let's see how it evolves ::

```
>>> evolve(grid, 10, 5)
```

not stable yet? Let’s play 10 more iterations, slower::

```
>>> evole(grid, 10, 2)
```

Boring, want more surprises?::

```
>>> bleach(grid, 20,40, Bitmap(1<<20*40))
>>> dirty(grid, 10)
```

It adds pattern randomly on the grid

Then, just sit back and play 200 iterations at 5 times the slow speed ::

```
>>> evolve(grid, 200,5)
```

You may have stable result around 100-200 iterations. What it the Bitmap by the way ?

Well, then fun part is matrix is just a view on anything that looks like a mutable sequence, and an int is a mutable sequence of bits, no ?

When (and only when) using Bitmap you can make::

```
>>> print "{0:b}".format(grid.matrix._int)
1000000001100000000000000000000000000111000000000000000001011000001000000000000110000010000000000
```

The quickstart is over :)

Have fun

Installing

To play with the source, you should use:

```
mkdir gof
cd gof
virtualenv .
. bin/activate
git clone git://github.com/jul/game_of_life.git

#.... Have Fun using gof interactively ...

deactivate
```

Using gof interactively

python

Boring simple:

```
python -i -mgof
```

bpython

bpython is an interactive console for python available normally as a binary package in your prefered binary distribution, or with pip install. Its strength lies in the completion, syntax highlihting, the ability to save your session either locally, or snapshot it on bpaste in one keystroke (with its screen output):

```
bpython -i -mgof
```

ipython

ipython is pretty much identical to bpython except it is the regular interactive session for pylab. So ... it is the tool of choice for dynamic plotting. you loose the cool saving part when you want to exchange your code.

Using ipython is a bit more complicated once you have typed:

```
ipython -i
```

you have to manually type

```
>>> from gof.console import *
>>> intro()
```

Forewords

Please read the forewords, and sign with your blood you are fully aware this is code is not PEP8 compliant, and not pythonic. It pretty much brings back the fun of a BASIC like language for playing with game of life.

For the sake of fun some twists to purity are pretty much acceptable at my opinion. It is not a serious project: it is a game.

Warning:

In serious projects :

- NEVER import *
- NEVER **use global variables**

If you do so, even me will curse your descendants mustache for 7 generations.

Using the playground

Gof console is leaving you with a depressing empty shell you want to play with.

Global *evil* variables

- DEAD or ALIVE : because True and False are too mainstream;
- grid : your playground;
- pixel : the pattern to draw a pixel;
- oscillator, still, glider : a non exhaustive list of singular patterns in game

Global *evil* function

intro()

A cheat sheet of the console

help

It is a builtin of python interpreter. Abuse it, this one might be useful:

```
>>> help grid
```

bleach(grid, x, y, atlernative_empty_mutable_array=x*y*[DEAD])

Cleans the grid of any living cells, and set time to 0

if X and Y is specified, resize the grid.

You can specify an alternative 1D sequence of mutable as a backend. Just remember to allocate enough space.

at(grid, x,y ,pattern=pixel, state=ALIVE)

Set at the position x,y the pattern given as an argument in the grid

dirty(times=1, list_of_pattern)

Set one random pattern at a random position in the grid.

evolve(grid, time=100,speed)

Make the grid evolve n times according to Conway's Rules. Default speed is waiting 1sec after each print. If speed is "unseeable" then, It will evolve quietly at full speed.

CHAPTER 5

Changelog

v0.1.0

- usable console
- pypi package, because it make doc shorter.

v0.1.1

- spotted a bug in reset (TOFIX)
- default backend is hint
- changed the code so that conway rules are only a turing machine encoded as an int
- added the grid.mutate(nb_flip=n) that flip randomly nbits in the conway code

v 0.1.2

- Added another demo demo2 to show the result of flipping n bit on a cellular automata

CHAPTER 6

Indices and tables

- genindex
- modindex
- search