Jeu de la vie de Conway

<sxh python; title : Conway_Game_of_Life_2D-JK-2012.py> #!/usr/bin/env python # -*- coding: utf-8 -*- “”“A minimal implementation of Conway's Game of Life.

source : http://www.exolete.com/code/life modified by par Jérémie Knoops, BA2 chimie UMONS, 2011-2012 cf. http://fr.wikipedia.org/wiki/Jeu_de_la_vie & http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life Each cell's survival depends on the number of occupied nearest and next-nearest neighbours (calculated in Grid::step). A living cell dies of overcrowding or loneliness if it has more than three or fewer than two neighbours; a dead cell is brought to life if it has exactly three neighbours (determined in Cell::setNextState).

```
Iain Haslam, June 2005.
```

”“” from Tkinter import * import time

#

Definition des cellules

class Cell(Label):

```
DEAD = 0
LIVE = 1

def __init__(self,parent):
    Label.__init__(self,parent,relief="raised",width=2,borderwidth=1)
    self.bind("<Button-1>", self.toggle)
    self.displayState(Cell.DEAD)
```

```
def toggle(self,event):
    self.displayState(1-self.state)
```

```
def setNextState(self,numNeighbours):
    """Work out whether this cell will be alive at the next iteration."""
    if self.state==Cell.LIVE and \
        (numNeighbours>3 or numNeighbours<2):
        self.nextState = Cell.DEAD
    elif self.state==Cell.DEAD and numNeighbours==3:
        self.nextState = Cell.LIVE
    else:
        self.nextState = self.state
```

```
def stepToNextState(self):
    self.displayState(self.nextState)
```

```
  def displayState(self,newstate):
      self.state = newstate
      if self.state==Cell.LIVE:
          self["bg"] = "black"
      else:
          self["bg"] = "white"
```

#

Definition de la grille

class Grid:

```
  def __init__(self,parent,sizex,sizey):
      self.sizex = sizex
      self.sizey = sizey
      #numpy.zeros(sizex,sizey) is a better choice,
      #but an additional dependency might be rude...
      self.cells = []
      for a in range(0,self.sizex):
          rowcells = []
          for b in range(0,self.sizey):
              c = Cell(parent)
              c.grid(row=b, column=a)
              rowcells.append(c)
          self.cells.append(rowcells)
  def step(self):
      """Calculate then display the next iteration of the game of life.
```

This function uses wraparound boundary conditions.

```
      """
      cells = self.cells
      for x in range(0,self.sizex):
          if x==0: x_down = self.sizex-1
          else: x_down = x-1
          if x==self.sizex-1: x_up = 0
          else: x_up = x+1
          for y in range(0,self.sizey):
              if y==0: y_down = self.sizey-1
              else: y_down = y-1
              if y==self.sizey-1: y_up = 0
              else: y_up = y+1
              sum = cells[x_down][y].state + cells[x_up][y].state + \
                  cells[x][y_down].state + cells[x][y_up].state + \
                  cells[x_down][y_down].state + cells[x_up][y_up].state + \
```

```
                cells[x_down][y_up].state + cells[x_up][y_down].state
            cells[x][y].setNextState(sum)
    for row in cells:
        for Cell in row:
            Cell.stepToNextState()
    print self.calc()
def clear(self):
    for row in self.cells:
        for Cell in row:
            Cell.displayState(Cell.DEAD)
def modify(self,Coord):
    self.clear()
    for (x,y) in Coord:
        self.cells[x][y].displayState(Cell.LIVE)
```

def calc(self):

```
    n=0
    for row in self.cells:
        for Cell in row:
            if Cell.state==Cell.LIVE:
                n=n+1
    return n
def multistep(self):
    text1=KBvar1.get()
    try:
        ns=int(text1)
    except ValueError:
        ns = 1
    text2=KBvar2.get()
    try:
        delay=int(text2)
    except ValueError:
        delay = 0
    for a in range(ns):
        time.sleep(delay)
        self.step()
        self.update()
```

```
def update(self):
    for row in self.cells:
            for Cell in row:
                Cell.update_idletasks()
```

#

Programme principal

root = Tk() if name == “main”:

```
Figures=[("Blinker",((0,1),(1,1),(2,1))),("Glider",((0,2),(1,0),(2,1),(1,2),
(2,2))),("R-Pentomino",((0,1),(1,0),(1,1),(1,2),(2,0)))]
  upframe = Frame(root)
  upframe.grid(row=0,column=0)
  middleFrame =Frame(root)
  middleFrame.grid(row=1,column=0)
  bottomFrame= Frame(root)
  bottomFrame.grid(row=2,column=0)
  gr = Grid(upframe,30,30)
  for i,fig in enumerate(Figures):
        Button(middleFrame,
                text=fig[0],
                command=lambda toto=fig:
                  gr.modify(toto[1])). \
                  grid(row=i,column=0)
```

```
  ###ajout
  textlab1=Label(middleFrame, text='Number of steps:', width=15, height=2,
fg="black")
  textlab1.grid(row=0,column=1)
  KBvar1=StringVar()
  KB1=Entry(middleFrame,textvariable=KBvar1,width=5)
  KB1.grid(row=0,column=2)
  textlab2=Label(middleFrame, text='Delay(sec):', width=15, height=2,
fg="black")
  textlab2.grid(row=1,column=1)
  KBvar2=StringVar()
  KB2=Entry(middleFrame,textvariable=KBvar2,width=5)
  KB2.grid(row=1,column=2)
  ###
  buttonStep = Button(bottomFrame,text="Step",command=gr.multistep)
  buttonStep.grid(row=1,column=1)
  buttonCalc = Button(bottomFrame,text="Calculate",command=gr.calc)
  buttonCalc.grid(row=1,column=2)
  buttonClear = Button(bottomFrame,text="Clear",command=gr.clear)
  buttonClear.grid(row=1,column=3)
  buttonQuit = Button(bottomFrame,text="Quit",command=root.destroy)
  buttonQuit.grid(row=1,column=4)
```

```
  root.mainloop()
```

</sxh>

Références

- http://www.exolete.com/code/life
- http://fr.wikipedia.org/wiki/Jeu_de_la_vie
- http://en.wikipedia.org/wiki/Conway%27s_Game_of_Life