A representation of the "Game of Life"

Neel Ramani

November 11, 2015

This report contains information about Conway's Game of Life.

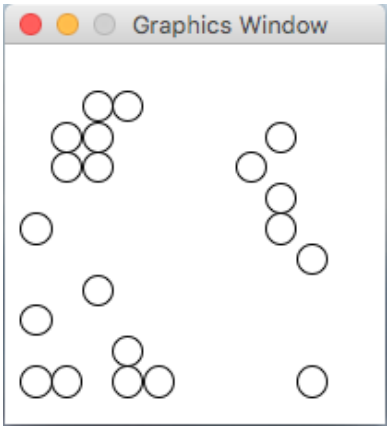

Figure 1: This is an example of the "Conway's Game of Life".

About the Report

This report shows:

1)What is Conway's Game of Life

2)What are the rules of the Conway's Game of Life

3)Python Code to create Conway's Game of Life

4)How the code works and the output

1) What is Conway's Game of Life

The Game of Life, also known simply as Life, is a cellular automaton devised by the British mathematician John Horton Conway in 1970.The "game" is a zero-player game,meaning that its evolution is determined by its initial state, requiring no further input. One interacts with the Game of Life by creating an initial configuration and observing how it evolves or, for advanced players, by creating patterns with particular properties.

2) What are the rules of the Conway's Game of Life

The universe of the Game of Life is an infinite two-dimensional orthogonal grid of square cells, each of which is in one of two possible states, alive or dead. Every cell interacts with its eight neighbours, which are the cells that are horizontally, vertically, or diagonally adjacent. At each step in time, the following transitions occur:

1)Any live cell with fewer than two live neighbours dies, as if caused by under-population.

2)Any live cell with two or three live neighbours lives on to the next generation.

3)Any live cell with more than three live neighbours dies, as if by over-population.

4)Any dead cell with exactly three live neighbours becomes a live cell, as if by reproduction.

In the code shown below, the Game of Life created in Python uses the same set of rules and the grid created here depends on the input.

3) PYTHON CODE

```
import random
from graphics import *


def empty(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[0]
        a=a+[b]
    return a


def fill(a,p):
    N=len(a)
    for i in range(N):
        for j in range(N):
            if random.uniform(0,1)<p:
                a[i][j]=1

def update(A,B):
    N=len(A)
    for i in range(N):
        for j in range(N):
            neigh=A[(i-1)%N][(j-1)%N]+A[(i-1)%N][j]+
                  A[(i-1)%N][(j+1)%N]+A[i][(j-1)%N]+
                  A[i][(j+1)%N]+A[(i+1)%N][(j-1)%N]+
                  A[(i+1)%N][j]+A[(i+1)%N][(j+1)%N]
            if A[i][j]==0:
                if neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
            else:
                if neigh==2 or neigh==3:
                    B[i][j]=1
                else:
                    B[i][j]=0
```

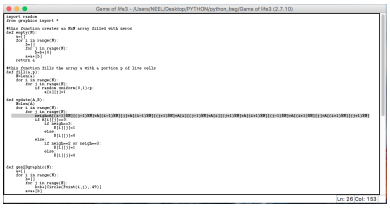

Figure 2: The highlighted section in the code as shown in the picture will be needed to change it in the code given below in order for it to function properly.

Please read the caption of the image carefully. After copying the code in Python, press enter and then you will be prompted to enter a number which will be your grid dimension.

The number you enter will be the grid size. For example, if you input the number 4, the grid created will be 4x4 size. Be sure to enter only positive integers.

The bigger the grid size is, the more computing it will take. Input of an integer **between 4 to 10** will be appropriate for a normally configured computer to show the Game of Life.

```
def gen2Dgraphic(N):
    a=[]
    for i in range(N):
        b=[]
        for j in range(N):
            b=b+[Circle(Point(i,j),.49)]
        a=a+[b]
    return a

def push(B,A):
    N=len(A)
    for i in range(N):
        for j in range(N):
            A[i][j]=B[i][j]

def drawArray(A,a,window):
    N=len(A)
    for i in range(N):
        for j in range(N):
            if A[i][j]==1:
                a[i][j].undraw()
                a[i][j].draw(window)
            if A[i][j]==0:
                a[i][j].undraw()

N=int(input("Enter the grid dimension  "))
win = GraphWin()
win.setCoords(-1,-1,N+1,N+1)
grid=empty(N)
grid2=empty(N)
circles=gen2Dgraphic(N)
fill(grid,0.3)

while True:
    drawArray(grid,circles,win)
    update(grid,grid2)
    push(grid2,grid)
```

4) How the code works and the output

Firstly, The code creates an array of the dimension you put in.Then, it fills the array with zero and changes random elements of the array to 1 which creates an array of 1s and 0s of a particular dimension.When this is done, the rules are applied. If there is 1 in the array, the cell is alive else the cell is dead, if the cell is alive, a circle is graphed into the window. Then it checks the neighbours and co-relates it to the rules as stated.Finally, the new array we get is graphed and then the new array is made old and the process keeps on going.

```
>>>
Enter the grid dimension
```

This will be the first outcome when you run the code. You need to input a positive integer in order for the code to start the Game of Life with a particular grid size

After this is done, a new window will appear and the first array or the first state will be graphed and then it will **keep updating itself**.