Conway's Game of Life

by Nikola Viazmenski

Due 11/10/2015

Contents

1 Overview of the Game

Conway's Game of Life was created in 1970 by British mathematician John Horton Conway. The Game is based around "cells", which are either in a state of life, or which are dead. Which cells are alive and which are dead are determined by the basic rules of the Game, which are described below. By design, the Game is a zero-player game - all calculations are done by the computer, with a user only giving the initial arrangement of the cells - and in some cases, even this is not so, and the computer generates the initial arrangement. The rules of the Game are as follows:

1. Any living cell with less than two neighbors will die, presumably due to underpopulation.
2. Any living cell with greater than three neighbors will die, presumably due to overpopulation.
3. Any living cell with two or three neighbors survives to the next generation.
4. Any dead cell with precisely three neighbors will come to life.

The Game and its derivatives are the most famous kinds of what are known as "cellular automatons" - computerized simulations of life.

2 Code produced in class

The following is code produced in the class, which simulates the Game of Life.

2.1 Sections of the Code

2.1.1 Section 1

Section 1 is rather simplistic. It imports the libraries needed for the program, namely random, graphics, and Tkinter, as well as the tkMessageBox module. Random is used to generate the random locations of the cells which start alive, graphics allows for the Game to be rendered and represented in a window, and tkinter is relevant to the user input at the beginning of the Game.

```
import random
from graphics import *
from Tkinter import *
import tkMessageBox
```

2.1.2 Section 2

Section 2 opens a window using Tkinter where the user can input the ratio of living cells to total cells which they would like to create a game with.

```
font = "times", 25, "bold"
lTitle = Label(wMain, text = "Conway's Game of Life", font = font)
lTitle.place(x = 120, y = 30)

font2 = "times", 20
lOption = Label(wMain, text = "Number of Starting Cells", font = font2)
lOption.place(x = 145, y = 80)

font3 = "times", 15
lOption2 = Label(wMain, text = "1/", font = font3)
lOption2.place(x = 155, y = 110)
```

```
eEntry = Entry(wMain, bd = 2)
eEntry.place(x = 175, y = 110)
```

2.1.3 Section 3

Section 3 generates the opening sequence in which the Game is played, with the function "gameCallBack". The subfunction "startUpCallBack" randomly generates where the cells will be starting in a state of life, based around the size of the game field and the ratio of cells that the user selected to start as alive. The following subfunction, "firstLoadCallBack", actually generates the numerical field upon which the game will be played.

```
def gameCallBack(entry):
    p = []
   alives = []
   size = 35

   game = GraphWin("The Game Of Life", 400, 400)
   game.setCoords(-.2, 0, size, size + .2)

   def startUpCallBack():
        live = 1.0/int(entry) * (size**2)
        for i in range(int(live)):
         x = int(random.uniform(0, size))
         y = int(random.uniform(0, size))
            alives.append([x, y])
        return alives
   def firstLoadCallBack():
         startUpCallBack()
         for i in range(size):
            a = []
            for r in range(size):
               a.append(0)
            p.append(a)
         for i in range(size):
            for r in range(size):
               for q in range(len(alives)):
                  if alives[q][0] == r:
                        if alives[q][1] == i:
                           p[i][r] = 1
     return p
   firstLoadCallBack()
```

2.1.4 Section 4

Section 4 is the large function "loaderCallBack". This is where the game is rendered into a window from the graphics library, the rules are defined, and the

calculations based on the rules are processed, the results of which are appended onto the new array which is then drawn as the new state of the game.

```
    def loaderCallBack(p):
        green = color_rgb(82, 245, 103)
      red = color_rgb(247, 34, 49)
      black = color_rgb(0, 0, 0)
        while True:
            for i in range(size):
            for r in range(size):
               point = Rectangle(Point(r, i), Point(r + 1, i + 1))
               if p[i][r] == 0:
                     point.setFill(black)
               if p[i][r] == 1:
                     point.setFill(green)
               point.draw(game)
            x = []
         for i in range(size):
            g = []
            for r in range(size):
               g.append(p[i][r])
            x.append(g)
         for i in range(size):
            for r in range(size):
            ss = 0
                    if not i == (size - 1) and not i == 0:
                  if not r == (size - 1) and not r == 0:
                            ss = ss + p[i + 1][r - 1]
                     ss = ss + p[i + 1][r]
                     ss = ss + p[i + 1][r + 1]
                     ss = ss + p[i][r - 1]
                     ss = ss + p[i][r + 1]
                     ss = ss + p[i - 1][r - 1]
                     ss = ss + p[i - 1][r]
                     ss = ss + p[i - 1][r + 1]
                    if i == (size - 1):
                  if not r == (size - 1) and not r == 0:
                     ss = ss + p[0][r - 1]
                     ss = ss + p[0][r]
                     ss = ss + p[0][r + 1]
                     ss = ss + p[i][r - 1]
                     ss = ss + p[i][r + 1]
                     ss = ss + p[i - 1][r - 1]
                     ss = ss + p[i - 1][r]
                     ss = ss + p[i - 1][r + 1]
               if i == (size - 1):
                  if r == (size - 1):
                     ss = ss + p[0][r - 1]
                     ss = ss + p[0][r]
                     ss = ss + p[0][0]
```

```
      ss = ss + p[i][r - 1]
      ss = ss + p[i][0]
      ss = ss + p[i - 1][r - 1]
      ss = ss + p[i - 1][r]
      ss = ss + p[i - 1][0]
if i == (size - 1):
   if r == 0:
      ss = ss + p[0][size - 1]
      ss = ss + p[0][r]
      ss = ss + p[0][r + 1]
      ss = ss + p[i][size - 1]
      ss = ss + p[i][r + 1]
      ss = ss + p[i - 1][size - 1]
      ss = ss + p[i - 1][r]
      ss = ss + p[i - 1][r + 1]
if i == 0:
   if not r == (size - 1) and not r == 0:
      ss = ss + p[i][r - 1]
      ss = ss + p[i][r]
      ss = ss + p[i][r + 1]
      ss = ss + p[i][r - 1]
      ss = ss + p[i][r + 1]
      ss = ss + p[size - 1][r - 1]
      ss = ss + p[size - 1][r]
      ss = ss + p[size - 1][r + 1]
     if i == 0:
   if r == (size - 1):
      ss = ss + p[i][r - 1]
      ss = ss + p[i][r]
      ss = ss + p[i][0]
      ss = ss + p[i][r - 1]
      ss = ss + p[i][0]
      ss = ss + p[size - 1][r - 1]
      ss = ss + p[size - 1][r]
      ss = ss + p[size - 1][0]
     if i == 0:
   if r == 0:
      ss = ss + p[i][size - 1]
      ss = ss + p[i][r]
      ss = ss + p[i][r + 1]
      ss = ss + p[i][size - 1]
      ss = ss + p[i][r + 1]
      ss = ss + p[size - 1][size - 1]
      ss = ss + p[size - 1][r]
      ss = ss + p[size - 1][r + 1]
if not i == (size - 1) and not i == 0:
   if r == (size - 1):
      ss = ss + p[i][r - 1]
      ss = ss + p[i][r]
      ss = ss + p[i][0]
```

```
                    ss = ss + p[i][r - 1]
                    ss = ss + p[i][0]
                    ss = ss + p[i][r - 1]
                    ss = ss + p[i][r]
                    ss = ss + p[i][0]
                  if not i == (size - 1) and not i == 0:
                 if r == 0:
                    ss = ss + p[i][size - 1]
                    ss = ss + p[i][r]
                    ss = ss + p[i][r + 1]
                    ss = ss + p[i][size - 1]
                    ss = ss + p[i][r + 1]
                    ss = ss + p[i][size - 1]
                    ss = ss + p[i][r]
                    ss = ss + p[i][r + 1]
              if p[i][r] == 1:
                 if ss < 2:
                    x[i][r] = 0
                 if ss > 3:
                    x[i][r] = 0
                 if not ss > 3 and not ss < 2:
                    x[i][r] = 1
                 if p[i][r] == 0:
                    if ss == 3:
                       x[i][r] = 1
                    if not ss == 3:
                       x[i][r] == 0

     p = []
     for i in range(size):
        g = []
        for r in range(size):
           g.append(x[i][r])
        p.append(g)

  loaderCallBack(p)
def startCallBack():
  gameCallBack(int(eEntry.get()))
```

2.1.5 Section 5

Section 5 of the code places the button which initializes the game on the user input window, and sets some parameters for the game window.

```
bBegin = Button(wMain, text = "Begin", command = startCallBack)
bBegin.place(x = 210, y = 150)

wMain.minsize(width = 500, height = 210)
wMain.maxsize(width = 500, height = 210)
```

```
wMain.title("Conway's Game of Life")
wMain.mainloop()
```

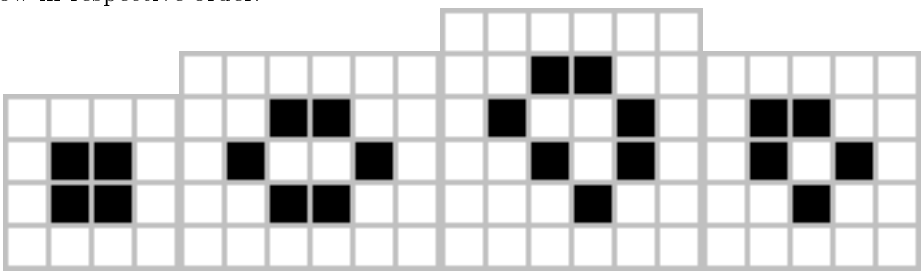
3 Patterns Observed in the Game

3.1 Stagnant patterns

3.1.1 Still Lifes

Still life patterns are patterns that neither grow nor shrink, but rather, when undisturbed, will simply stay static.

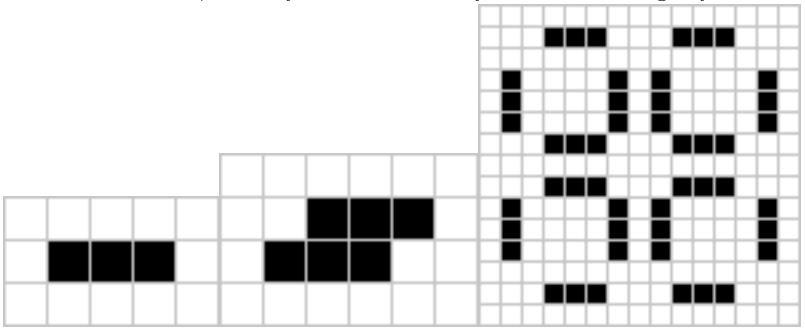
Examples include the Block, the Beehive, the Loaf, and the Boat, pictured below in respective order.



3.1.2 Oscillators

Oscillator patterns are patterns which switch back and forth between a set of states. The amount of different states that they have, or the amount of changes between a first state and the oscillator undergoes before it returns to its first state, is referred to as its period.

Examples include the Blinker (which is very frequently observed in mass quantities when a game is nearing complete stagnation), the Toad, and the Pulsar. Note that the oscillators can only be pictured in their base state due to technical limitations, so they cannot be truly shown in full glory.

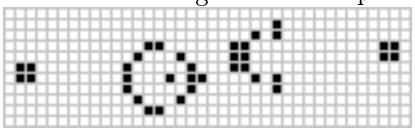


3.2 Translator patterns

Translator patterns, known as "spaceships", are patterns that will consistently move themselves across the game field. The speed at which they travel is compared to the speed of light - that is, a spaceship traveling at the speed of light is traveling one cell per turn. Spaceships can be further classified into their direction of movement, such as diagonal movement and orthogonal movement. Two of the simplest and most well-known spaceships are the glider and the lightweight spaceship, pictured below. Again, due to technical limitations, they can only be displayed here in one state.

3.3 Spawner patterns

Spawner patterns, referred to as "guns", are patterns that do not move around but infinitely produce smaller patterns, specifically spaceships. The first gun to be discovered was Gosper's glider gun, which was discovered by Bill Gosper in 1970. The significance of guns is that they can allow for fundamentally unlimited growth. The starting state of Gosper's glider gun is pictured below.



4 Variations of the Game

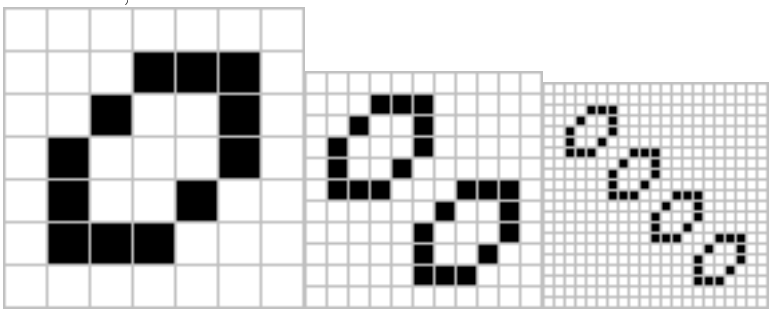
The Game's rules have been changed various times, in order to cultivate new and interesting patterns using the same general architecture of the living/dying rules, but changing the number of neighbors for which a cell lives and dies.

4.1 Notation used in variations

The rules of Conway's Game state that a cell is born if it has three neighbors, dies if it has less than two or more than three neighbors, and lives if it has two or three neighbors. The rules for this game are then denoted as 23/3 - two or three neighbors for a cell to live, three neighbors for a new cell to be born.

4.2 HighLife

HighLife is a variation on Conway's Game of Life. In HighLife, the rule is 23/36. This means that a cell survives if it has two or three neighbors, and a new cell is born if it has either three or six neighbors. This variation on the game still allows a majority of common classic patterns to exist, but also allows for easy creation of what is known as a "replicator" - a group of cells that that continually creates more and more copies of itself. Pictured below are a replicator, this replicator after 12 turns, and after an additional 24 turns.



5 Conclusion

Conway's Game of Life was and still is one of the most advanced and deep programs ever to be created. Its patterns have attracted the attention of and hypnotized thousands of people, its variations are deepening the entire universe of cellular automata, and more and more patterns and guns and the like are being discovered every day. As of the time of writing, LifeWikihas over 750 patterns intimately documented, and more are being discovered every day.

References

LifeWiki
"Conway's Game of Life." LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015. "Blinker." - LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015. "Glider." - LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015. "Gosper Glider Gun." - LifeWiki. N.p., n.d. Web. 9 Nov. 2015. "HighLife." - LifeWiki. N.p., n.d. Web. 9 Nov. 2015. "Lightweight Spaceship." - LifeWiki. N.p., n.d. Web. 9 Nov. 2015. "Pulsar." - LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015. "Spaceship." - LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015. "Toad." - LifeWiki. N.p., Nov. 2015. Web. 9 Nov. 2015.