The Game of Life - John Horton Conway

Juanita Gómez
Pontificia Universidad Javeriana

February 11, 2018

1 Introduction: The Game of Life

Mathematicians have been concerned with designing tools that allow modeling real-life phenomena through mathematical models. Computing is one of the most used resources for this purpose since they allow to represent algorithms that show the evolution of these phenomena.

The Game of Life is a zero-player game that was designed in 1970 by John Horton Conway, a British mathematician. It is said that this game is a cellular automata, since it is a mathematical model of a dynamic system that evolves in discrete steps, that is, quantifiable units of time with integer values at regular intervals. It consists of a set of cells that acquire different values that change according to mathematical expressions that are determined by the states of the neighboring cells. It has also been typecast in the category of simulation games because it aims to imitate real-life processes.

1.1 J.H. Conway

John H. Conway is one of the most prominent theoreticians in the study of finite groups and one of the most important knot theorists in the world. He has written or co-written more than ten books and over one hundred and thirty journal articles on a wide variety of mathematical topics. He has done important work in number theory, game theory, coding theory, in the creation of new numbering systems, including "surreal numbers" and is widely known as the inventor of the "game of life". Born in 1937, Conway received his Ph.D. in 1967 from the University of Cambridge where he was until 1986 at Princeton University, where he met John von Neumann, a mathematician who impelled the creation of his "Game of Life"

1.2 Origin

The mathematician John Horton Conway, designer of the game, intended to solve a problem that was presented in the 40s by John Von Newman, a Hungarian mathematician who tried to create a hypothetical machine that was capable of building copies of itself. Neumann arrived at a mathematical model of the machine that was developed on a grid with several complex rules. Initially this was interpreted as a set of cells that grew, reproduced and died with the passage of time for what were known as cellular automata. Conway's purpose was then to simplify the Neumann model. His idea was to start with a simple configuration of cells and observe how they change according to certain rules. These rules were chosen after a long period of experimentation, in which he basically wanted to achieve the following:

- There should not be an initial pattern for which there is a simple test that the population can grow without limit.
- There should be initial patterns that appear to grow without limit
- There should be simple initial patterns that grow and change for a considerable period of time before ending either by the death of all cells or by the establishment of cells that end in constant oscillations.

Basically, what Conway was looking for was that its rules made the behavior of the population unpredictable. In this way, he invented the Game of Life in 1970 and was shown to the public for the first time in October of that year, through an article that was published in the Scientific American magazine, in the Mathematical Games column by Martin Gardner. Once published, it quickly attracted the attention of those who knew it due to the emergence of such complex patterns based on simple rules. Additionally, it drew attention to its similarity with some of the evolutionary processes that determine the birth and decadence of the societies of living beings.

2 Method: The Game Rules

The Game of the life, develops in theory an infinite board that is known like "world", but in the practice has been developed in a board with fixed dimensions, that is divided in cells by means of a reticle.

Each of the cells contains what is known as "cell" and is surrounded by 8 squares, 2 that are laterally adjacent, 2 vertically adjacent and 4 adjacent diagonally. These 8 boxes are known as "neighborhood", which determines the state of the cell in the next generation. In this way, the game is based on the evolution of the successive states of the cells, in which the conditions of a state depend only on the conditions of the previous state. Therefore, it is not necessary to input data during the development of the game, but the initial state of it, is what determines its evolution. The participation of a user in the game only consists in the determination of its initial state, creating what is called "initial population" or "zero generation".

The possible states of each one of the cells of the game are, alive cell that can be considered a "logical one" or dead cell considered like the "logical zero". The state of each cell depends on its current state and the current state of its 8 neighboring cells, following some rules. A cell dying leaves the cell it occupied empty.

The rules established by Conway for the development of the game were:

- Survival: Living cells that have 2 or 3 living neighbors survive the next generation (their state remains unchanged)
- Death: Living cells that have less than 2 neighbors die by isolation or solitude and those that have more than three living neighboring cells die from overpopulation in the next turn.
- Birth: A dead (empty) cell that has exactly 3 living neighboring cells, will become a living cell in the next turn (birth of a new individual).

Once the initial state of the cells is established, iterations are carried out according to the established rules, thus giving the evolution of the population in which the following results can be obtained:

- Extinction: After a finite number of iterations, all members of the population disappear.
- Stabilization: After a finite number of iterations, the population is stabilized in two ways: The remaining cells remain in their state constantly or it may happen that the remaining cell forms are oscillating.
- Constant growth: The population grows turn after shift and an infinite number of generations is maintained.

3 Implementation in Python and Matlab

In this section we show the code created to model this phenomenon, in Python and Matlab, two software tools that allow us to visualize the evolution of the game. In both the same algorithm was used, with modifications only in terms of the language of the platforms. Therefore, each of the sections of the code will be explained only in one of the IDEs, along with its operation.

3.1 Matlab

Below is the code used in Matlab to design the Game of Life program.

```matlab

Title = ['///////============================================================///////\n'...
'///////                         WELCOME TO                          ///////\n'...
'///////                                                             ///////\n'...
'///////                     THE GAME OF LIFE                        ///////\n'...
'///////=============================================================///////\n'...
'///////                 Presented by: Juanita Gomez                 ///////\n'...
'///////=============================================================///////\n'];

separator = '\n///////=================================================================///////';
fprintf(Title);
fprintf('\nPresented by: Juanita Gomez');
r = input('\nChoose the probability that a cell will start alive: ');
m = input('\nChoose the size of the square matrix to use: ');
state = 0;
a='#';
b=' ';
%Matrix Creation
TheGameOfLife=char(ones(m));
for i=1:m
    for j=1:m
        y = randi([0 100],1,1);
        if (y>r)
            TheGameOfLife(i,j)=b;
        else
            TheGameOfLife(i,j)=a;
        end
    end
end
%Matrix Evolution (Infinite Cycle)
while 1>0
    clc;
    fprintf(Title);
    fprintf('\nYou chose an initial probability of life of: %d ',r);
    fprintf('\nYou chose a square matrix of size: %d', m);
    fprintf(separator);
    display(TheGameOfLife);
    fprintf(separator);
    fprintf('\nState: %d', state);
    fprintf('\nThanks for playing\nTo end the game press (ctrl + c)\nUntil next time\n');
    state = state + 1;
    pause(0.1);
    %Each of the iterations of the game
    for i=1:m
        for j=1:m
            v=0;
            if i+1<m+1
                if TheGameOfLife(i+1,j)==a
                    v=v+1;
                end
                if j+1<m+1
                    if TheGameOfLife(i+1,j+1)==a
                        v=v+1;
                    end
                end
                if j-1>0
                    if TheGameOfLife(i+1,j-1)==a
                        v=v+1;
                    end
                end
            end
            if i-1>0
                if TheGameOfLife(i-1,j)==a
                    v=v+1;
                end
                if j+1<m+1
                    if TheGameOfLife(i-1,j+1)==a
                        v=v+1;
                    end
                end
                if j-1>=1
                    if TheGameOfLife(i-1,j-1)==a
                        v=v+1;
                    end
                end
            end
            if j+1<m+1
                if TheGameOfLife(i,j+1)==a
                    v=v+1;
                end
            end
            if j-1>0
                if TheGameOfLife(i,j-1)==a
                    v=v+1;
                end
            end

            %Find alive cells
            if TheGameOfLife(i,j)==a
                %1Living cells that have less than 2 neighbors die
                if v<2
                    TheGameOfLife(i,j)=b;
                end
                %2Living cells that have 2 or 3 living neighbors survive the next generation
                if (v==2||v==3)
                    TheGameOfLife(i,j)=a;
                end
                %3Living cells that have more than 3 neighbors die
                if v>3
                    TheGameOfLife(i,j)=b;
                end
            end
            %Find dead cells
            if TheGameOfLife(i,j)==b;

                %4A dead (empty) cell that has exactly 3 neighbors becomes a living cell
```

```
                if v==3
                    TheGameOfLife(i,j)=a;
                end
            end
        end
    end
end
```

3.1.1 Title and configuration

In this part of the code, the titles of the program are shown and the user is welcomed to start the game. As mentioned above, the user's participation in the game only consists of determining its initial state, creating the "initial population". For this, two parameters have been implemented: The first is the size of the square matrix in which it will be developed, and the second is the probability that a cell is alive in an initial state. This means that the user will determine how many living cells will appear in the initial population through their probability, for example if he chose a matrix of size 10x10 and a probability of 30, then 30 living cells and 70 dead cells will appear. In this part of the program, a 'state' counter is also presented that will allow the user to see the number of iterations that the program has been running since it started. Finally define the variables a and b that represent the 'alive' and 'dead' states of the cells and are defined as characters to be visualized in the matrix. In this case the numeral character was chosen for the living state and the space for the dead state.

```
Title =
['///////==================================================================///////\n'...
'///////                          WELCOME TO                               ///////\n'...
'///////                                                                   ///////\n'...
'///////                       THE GAME OF LIFE                            ///////\n'...
'///////=================================================================///////\n'...
'///////                  Presented by. : Juanita Gomez                    ///////\n'...
'///////=================================================================///////\n'];

separator = '\n///////==================================================================///////'

fprintf(Title);
fprintf('\nPresented by: Juanita Gomez');
r = input('\nChoose the probability that a cell will start alive: ');
m = input('\nChoose the size of the square matrix to use: ');

state = 0;
a='#';
b=' ';
```

3.1.2 Creation of the initial matrix

In this part of the code, an array of mXm size is created, dimensions previously set by the user. To determine the state of the initial matrix, a cycle is used in which the matrix is traversed cell by cell and for each one a random number is assigned between 0 and 100. According to the probability 'r' chosen by the user, if the random value is greater than r the cell will be dead in the initial state; if on the contrary the value is less than r, the cell will be alive in its initial state. In this way there will be 'r' living cells in the matrix but they will be in completely random positions every time the program is run.

```
TheGameOfLife=char(ones(m));
for i=1:m
    for j=1:m
        y = randi([0 100],1,1);
        if (y>r)
            TheGameOfLife(i,j)=b;
        else
            TheGameOfLife(i,j)=a;
        end
    end
end
```

3.1.3 General program evolution

Once the initial matrix is created, the evolution of the cells begins. This algorithm is presented as an infinite cycle that is defined with a 'while' that is always true (while 1> 0), so that the program does not stop unless the user wants it. After the while the first thing that appears is a command that allows to "clean" the screen where the program is run, in such a way that in each of the iterations of the program, the matrix appears in the same place as in the previous iteration. In this part of the code, the program information is presented to the user: The title appears in each of the iterations, along with information about the probability and size of the initial matrix chosen by the user and the variable 'state' is shown which reflects the number of iterations that the program has been since it started, which is added one in each iteration of the cycle. An announcement is also presented informing the user how the program should end, in this case by pressing ctrl + c. Finally, the last part of this section is a command that allows you to pause the program before each iteration so that the evolution of the program can be carefully visualized.

```
while 1>0
    clc;
    fprintf(Title);
    fprintf('\nYou chose an initial probability of life of: %d ',r);
    fprintf('\nYou chose a square matrix of size: %d', m);
    fprintf(separator);
    display(TheGameOfLife);
    fprintf(separator);
    fprintf('\nState: %d', state);
    fprintf('\nThanks for playing\nTo end the game press (ctrl + c)\nUntil next time\n');
    state = state + 1;
    pause(0.3);
```

3.1.4 Each iteration of the game

Each iteration of the program is determined by a cycle that runs through each of the cells of the matrix, performing 2 procedures:

- Neighbors counting: In this process the status of the eight neighbors of each of the cells is verified and those that are alive are counted through a counter 'v' that is initialized at the beginning of the cycle and using the comparison of the states initially defined as 'a' and 'b'. To perform this count, it is necessary to be careful when crossing the boxes of the matrix that are on the lateral or vertical edges of the matrix, so as not to leave the limits of the matrix. For this, before comparing the state of a neighbor of a cell it is necessary to verify that this neighbor exists within the limits of the matrix. If it exists, its state is compared with 'a' and 'b' to know if it is alive, if it does not exist, it will proceed to verify the other neighbors.

- Determination of cell status: Once the count of the living neighbors of the cell is completed, it is verified whether this cell is in its 'alive' or 'dead' state, since depending on this and its number of living neighbors its status will be determined in the next turn. In case the cell is alive, it is verified if the number of neighbors is greater than 3 or less than 2, in which case the cell becomes a dead cell, changing its state from 'a' to 'b'. If the number of live neighbors is exactly 2 or 3, the cell remains in its current state the next turn. In case the cell is dead, it is only verified if the number of living neighbors is exactly equal to 3, in which case it becomes a living cell going from 'b' to 'a'. Otherwise its state remains the same.

```
    %Each of the iterations of the game
    for i=1:m
        for j=1:m
            v=0;
            if i+1<m+1
                if TheGameOfLife(i+1,j)==a
                    v=v+1;
                end
                if j+1<m+1
                    if TheGameOfLife(i+1,j+1)==a
                        v=v+1;
                    end
                end
                if j-1>0
                    if TheGameOfLife(i+1,j-1)==a
                        v=v+1;
                    end
                end
            end
            if i-1>0
                if TheGameOfLife(i-1,j)==a
                    v=v+1;
                end
                if j+1<m+1
                    if TheGameOfLife(i-1,j+1)==a
                        v=v+1;
                    end
                end
                if j-1>=1
                    if TheGameOfLife(i-1,j-1)==a
                        v=v+1;
                    end
                end
            end
            if j+1<m+1
                if TheGameOfLife(i,j+1)==a
                    v=v+1;
                end
            end
```

```
                if j-1>0
                    if TheGameOfLife(i,j-1)==a
                        v=v+1;
                    end
                end

                %Find alive cells
                if TheGameOfLife(i,j)==a
                    %1Living cells that have less than 2 neighbors die
                    if v<2
                        TheGameOfLife(i,j)=b;
                    end
                    %2Living cells that have 2 or 3 living neighbors survive the next generation
                    if (v==2||v==3)
                        TheGameOfLife(i,j)=a;
                    end
                    %3Living cells that have more than 3 neighbors die
                    if v>3
                        TheGameOfLife(i,j)=b;
                    end
                end
                %Find dead cells
                if TheGameOfLife(i,j)==b;

                    %4A dead (empty) cell that has exactly 3 neighbors becomes a living cell
                    if v==3
                        TheGameOfLife(i,j)=a;
                    end
                end
            end
        end
```

3.2 Python

Below we present the Python code used for the program.

```

# -*- coding: utf-8 -*-
import os
import time
import random
import platform



Title = ('///////============================================================///////\n'
          '///////                        WELCOME TO                         ///////\n'
          '///////                                                           ///////\n'
          '///////                     THE GAME OF LIFE                      ///////\n'
          '///////============================================================///////\n'
          '///////                Presented by: Juanita Gomez                ///////\n'
          '///////============================================================///////\n');

separator = ('\n///////============================================================///////')

footer = ('Thanks for playing\nTo end the game press (ctrl + .)\nUntil next time\n')




if platform.system() == 'Darwin' or platform.system() == 'Linux' :
    os.system('clear')
elif platform.system() == 'win32' :
    os.system('cls')

print(Title)

r = input('Choose the probability that a cell will start alive: ')
m = input('Choose the size of the square matrix to use: ')
if m > 35 :
    print('Note that for that matrix size it may be necessary to\nreadjust the screen')
    time.sleep(2)
n=0
a=str(' #')
b=str('  ')

#Matrix Creation
matriz = [[0 for x in range(m)] for y in range(m)]
i=0;
for i in range(0,m):
    for j in range(0,m):
        y=random.randint(0,100)
        if (y>r):
            matriz[i][j]=b
        else:
            matriz[i][j]=a

#Matrix evolution (Infinite Cycle)
while 1>0:

    if platform.system() == 'Darwin' or platform.system() == 'Linux' :
        os.system('clear')
    elif platform.system() == 'win32' :
        os.system('cls')

    print (Title)

    print('You chose an initial probability of life of: ' + str(r));
    print('You chose a square matrix of size: ' + str(m));

    print (separator + '\n')
    print('\n'.join([''.join(['{:2}'.format(item) for item in row])
          for row in matriz]))
    print (separator + '\n')
    print (Iteration number:+ str(n) )
    print (footer)
```

```
        n += 1
        time.sleep(0.1)

    #Each iteration of the program
    for i in range(0,m):
        j=0
        for j in range(0,m):
            v=0;
            if i+1<m:
                if matriz[i+1][j]==a:
                    v=v+1
                if j+1<m:
                    if matriz[i+1][j+1]==a:
                        v=v+1
                if j-1>=0:
                    if matriz[i+1][ j-1]==a:
                        v=v+1
            if i-1>=0:
                if matriz[i-1][j]==a:
                    v=v+1
                if j+1<m:
                    if matriz[i-1][j+1]==a:
                        v=v+1
                if j-1>=0:
                    if matriz[i-1][j-1]==a:
                        v=v+1
            if j+1<m:
                if matriz[i][j+1]==a:
                    v=v+1
            if j-1>=0:
                if matriz[i][j-1]==a:
                    v=v+1

            #Fin alive cells
            if matriz[i][j]==a:

                #1Living cells that have less than 2 neighbors die
                if v<2:
                    matriz[i][j]=b
                #2Living cells that have 2 or 3 living neighbors survive the next generation
                if (v==2 or v==3):
                    matriz[i][j]=a
                #3Living cells with more than 3 neighbors die
                if v>3:
                    matriz[i][j]=b

            #Find dead cells
            if matriz[i][j]==b:

                #4A dead (empty) cell that has exactly 3 neighbors becomes a living cell
                if v==3:
                    matriz[i][j]=a
```

3.2.1 Title and configuration

In this part of the code, same as in matlab, the titles of the program are shown and the user is welcomed to start the game, asking him to insert the size of the square matrix and the probability that a cell is alive in an initial state. There is also the iteration counter and the definition of the variables a and b as explained above. Additional to what we saw in matlab, we set a warning when the sizes of the matrix are very large since it may be necessary to adjust the screen. In that case, there is a pause time that allows the user to read the warning. For this, unlike matlab, it was necessary to use a function that depends on the operating system, so it is necessary to identify it in this part of the process.

```

# -*- coding: utf-8 -*-
import os
import time
import random
import platform



Title = ('///////============================================================///////\n'
         '///////                          WELCOME TO                        ///////\n'
         '///////                                                            ///////\n'
         '///////                       THE GAME OF LIFE                     ///////\n'
         '///////============================================================///////\n'
         '///////                 Presented by: Juanita Gomez                ///////\n'
         '///////============================================================///////\n');

separator = ('\n///////============================================================///////')

footer = ('Thanks for playing\nTo end the game press (ctrl + .)\nUntil next time\n')

if platform.system() == 'Darwin' or platform.system() == 'Linux' :
    os.system('clear')
elif platform.system() == 'win32' :
    os.system('cls')

print(Title)

r = input('Choose the probability that a cell will start alive: ')
m = input('Choose the size of the square matrix to use: ')
if m > 35 :
    print('Note that for that matrix size it may be necessary to\nreadjust the screen')
    time.sleep(2)

```

```
n=0
a=str(' #')
b=str('  ')
```

3.2.2 Creation of the initial matrix

In this part of the code, as in matlab, the matrix is created and its initial state is determined.

```

#óCreacin de matriz
matriz = [[0 for x in range(m)] for y in range(m)]
i=0;
for i in range(0,m):
    for j in range(0,m):
        y=random.randint(0,100)
        if (y>r):
            matriz[i][j]=b
        else:
            matriz[i][j]=a
```

3.2.3 General program evolution

In this part of the code, an infinite cycle is created in the same way as in matlab, the command 'Clean the screen' appears and the program information is presented to the user which includes the title, probability and size of the matrix , and the number of iterations that the program has been since it started. Finally, there is the command that allows you to pause the program before each iteration.

```
#Matriz Evolution (Infinite cycle)
while 1>0:

    if platform.system() == 'Darwin' or platform.system() == 'Linux' :
       os.system('clear')
    elif platform.system() == 'win32' :
       os.system('cls')

    print (Title)

    print('You chose an initial probability of life of: ' + str(r));
    print('You chose a square matrix of size: ' + str(m));

    print (separator + '\n')
    print('\n'.join([''.join(['{:2}'.format(item) for item in row])
          for row in matriz]))
    print (separator + '\n')
    print ('Iteration number:' + str(n) )
    print (footer)

    n += 1
    time.sleep(0.1)
```

3.2.4 Each iteration of the program

In the Python code, each iteration of the program is also determined by a cycle that runs through each of the cells in the matrix, performing the 2 procedures explained above: Neighbors Counting, and determining the cell's status. The only differences in this part of the code with matlab, are the syntax, and the differences in the limits of the cycles due to the difference in the numbering of the boxes of a matrix in each program. In matlab, the matrices start from row and column 1, while in python their numbering starts at 0.

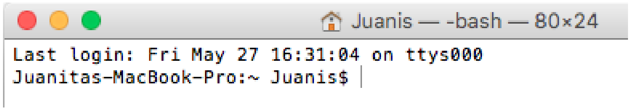
```
  #Each iteration of the program
    for i in range(0,m):
        j=0
        for j in range(0,m):
            v=0;
            if i+1<m:
                if matriz[i+1][j]==a:
                   v=v+1
                if j+1<m:
                    if matriz[i+1][j+1]==a:
                       v=v+1
                if j-1>=0:
                    if matriz[i+1][j-1]==a:
                       v=v+1
            if i-1>=0:
                if matriz[i-1][j]==a:
                   v=v+1
                if j+1<m:
                    if matriz[i-1][j+1]==a:
                       v=v+1
                if j-1>=0:
                    if matriz[i-1][j-1]==a:
                       v=v+1
            if j+1<m:
                if matriz[i][j+1]==a:
                   v=v+1
            if j-1>=0:
                if matriz[i][j-1]==a:
                       v=v+1
```

```
            #Fin alive cells
            if matriz[i][j]==a:

                #1Living cells that have less than 2 neighbors die
                if v<2:
                    matriz[i][j]=b
                #2Living cells that have 2 or 3 living neighbors survive the next generation
                if (v==2 or v==3):
                    matriz[i][j]=a
                #3Living cells with more than 3 neighbors die
                if v>3:
                    matriz[i][j]=b

            #Find dead cells
            if matriz[i][j]==b:

                #4A dead (empty) cell that has exactly 3 neighbors becomes a living cell
                if v==3:
                    matriz[i][j]=a
```

4 Example

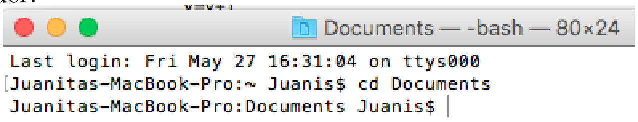
Next we will show an example of the program in python and we will review what happens in the first 5 iterations of the code. In the first place, it should be clarified that to open the program, it is necessary to do it in the terminal of the computer. For which, we open the terminal, access the folder where the file is located and run it from there. This is shown below.

4.1 Open the terminal



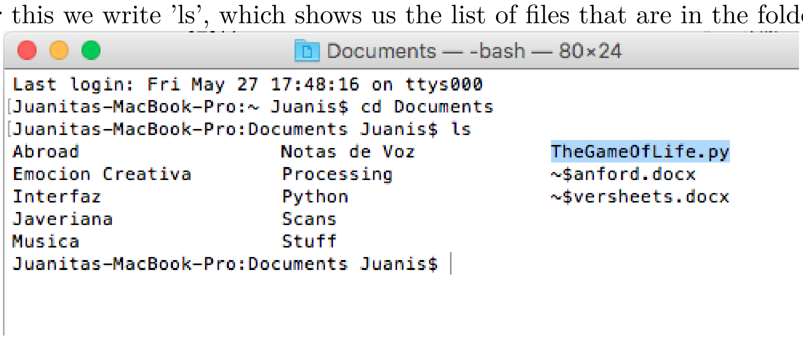
4.2 Open the folder where the file is

For this it is necessary to write 'cd' which means 'change directory', along with the name of the folder.



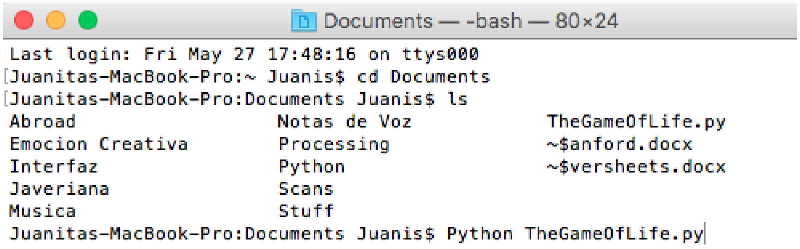
4.3 Verify that the file is in the folder

For this we write 'ls', which shows us the list of files that are in the folder to which we accessed



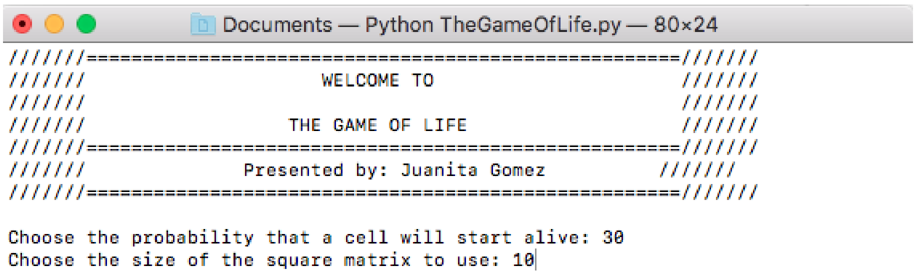
4.4 Run the file

For this it is necessary to write the name of the program 'Python' along with the name of the file separated by a space, that is 'TheGameOfLife.py'.



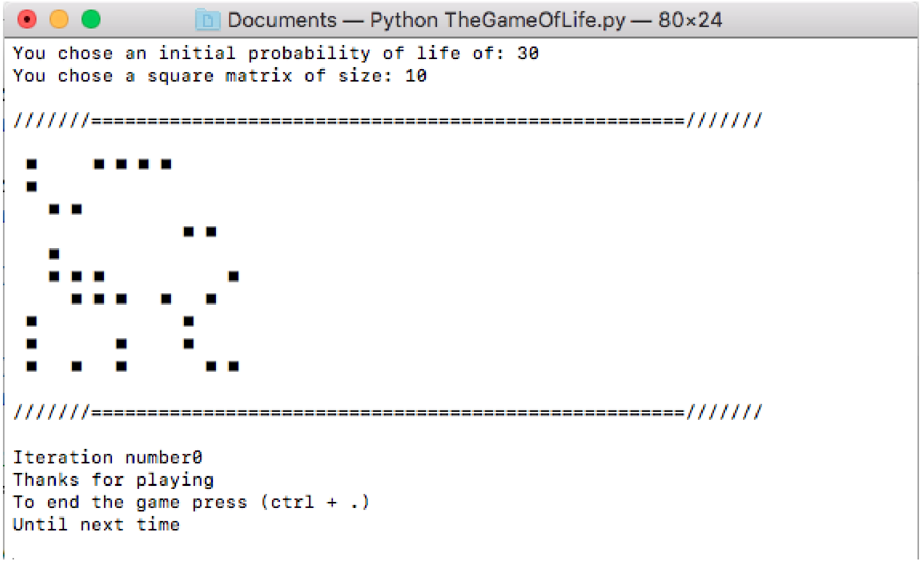
4.4.1 Request information to the user

The first window to run the program, shows the titles of this along with the questions of the data to the user. In this case, the initial probability of life of 30 and size of the matrix of 10X10 was placed.



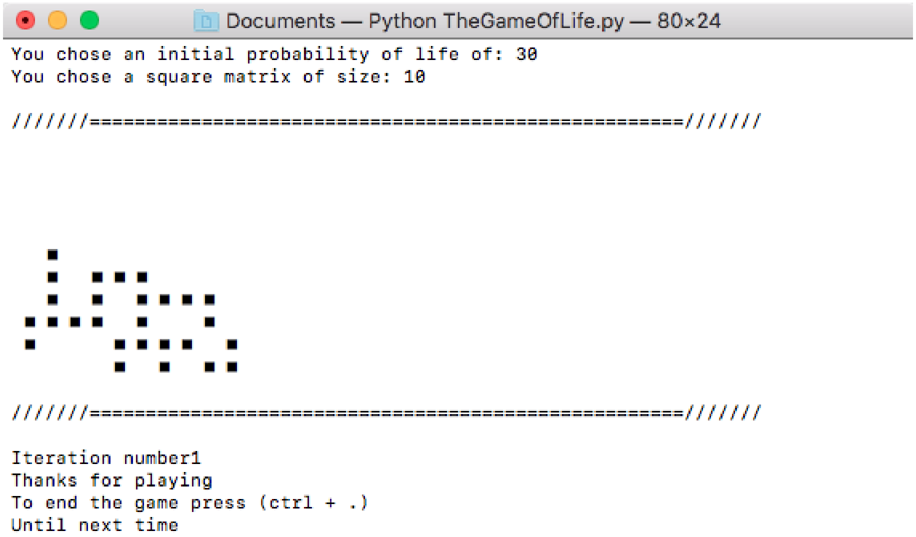
4.5 First matrix

On the screen appears a matrix of 10X10, which has 30 cells alive since this value corresponds to 30 percent of the total cells. The counter and instructions to end the program also appear.



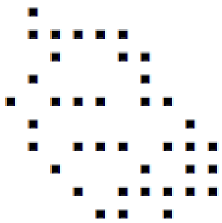
4.6 First iteration

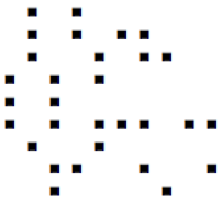
In the first iteration we observed that the program information is shown to the user, along with the titles and the number of iterations that changed from 0 to 1. The new matrix now has 28 cells alive after following the rules of Conway.

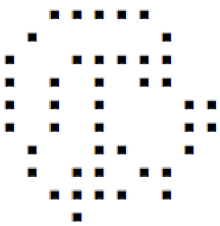


4.7 Program Evolution

Next we present the following 3 iterations of the program where we can visualize the transformations of the matrix.







4.8 End of the program

In this case, after 197 iterations, the matrix begins to be constant so that from there all the following matrices are equal. There is a stabilization of the population and since the program has an infinite duration, it is the user who now decides when he stops the program using the ctrl +

command.

References

[1] Melissa Gymrek, *Conway's Game of Life*, `http://web.mit.edu/sp.268/www/2010/lifeSlides.pdf`, (2010)

[2] David Alejandro Reyes Gómez, *Descripción y Aplicaciones de los Autómatas Celulares*, `http://delta.cs.cinvestav.mx/~mcintosh/cellularautomata/Summer_Research_files/Arti_Ver_Inv_2011_DARG.pdf`, (2011)

[3] Manuel Romero Dopico, *EL JUEGO DE LA VIDA*, `http://www.it.uc3m.es/jvillena/irc/practicas/09-10/04mem.pdf`, (s.f)

[4] Dierk Schleicher, *Interview with John Horton Conway*, `http://www.ams.org/notices/201305/rnoti-p567.pdf`, (2013)

[5] Martin Gardner, *MATHEMATICAL GAMES*, Scientific American 223, (1970)