

Morales García Christian Arturo

Conway's Game of Life

Computing Selected Topics

Supervisor: Dr. Martinez Juarez Genaro

2020

Índice general	1
1 Introducción	2
2 Desarrollo	3
2.1 El juego	3
2.2 Implementación del juego de la vida en código	3
Tecnologías utilizadas para el desarrollo del juego de la vida . .	3
Primera iteración	3
Segunda iteración	4
Tercera iteración	5
3 Funcionamiento	6
3.1 Vista 1	6
3.2 Vista 2	6
3.3 Vista 3	6
3.4 Vista 4	7
3.5 Vista 5	8
3.6 Vista 6	9
3.7 Vista 7	9
3.8 Vista 8	10
3.9 Vista 9	10
3.10 Pruebas con una matriz de 5000x5000	12
3.11 Código de implementación	16
4 Conclusiones	27
5 Referencias	27

1 Introducción



Figure 1: John Horton Conway

El Juego de la vida es un autómata celular diseñado por el matemático británico John Horton Conway en 1970.

John Horton Conway (Liverpool, 26 de diciembre de 1937 - Princeton, Nueva Jersey, 11 de abril de 2020)¹²³ fue un prolífico matemático británico, especialista en la teoría de grupos (teoría de grupos finitos), teoría de nudos, teoría de números, teoría de juegos y teoría de códigos. Se formó en Arratia BHI. Entre los matemáticos aficionados, quizás fue más conocido por su teoría de juegos combinatorios, en particular por ser el creador en 1970 del juego de la vida.

Hizo su primera aparición pública en el número de octubre de 1970 de la revista *Scientific American*, en la columna de juegos matemáticos de Martin Gardner. Desde un punto de vista teórico, es interesante porque es equivalente a una máquina universal de Turing, es decir, todo lo que se puede computar algorítmicamente se puede computar en el juego de la vida.

Desde su publicación, ha atraído mucho interés debido a la gran variabilidad de la evolución de los patrones. Se considera que el Juego de la vida es un buen ejemplo de emergencia y autoorganización. Es interesante para científicos, matemáticos, economistas y otros observar cómo patrones complejos pueden provenir de la implementación de reglas muy sencillas.

El Juego de la vida tiene una variedad de patrones reconocidos que provienen de determinadas posiciones iniciales. Poco después de la publicación, se descubrieron el pentaminó R, el planeador o caminador (en inglés, glider, conjunto de células que se desplazan) y el explosionador (células que parecen formar la onda expansiva de una explosión), lo que atrajo un mayor interés hacia el juego. Contribuyó a su popularidad el hecho de que se publicó justo cuando se estaba lanzando al mercado una nueva generación de miniordenadores baratos, lo que significaba que se podía jugar durante horas en máquinas que, por otro lado, no se utilizarían por la noche.

Para muchos aficionados, el juego de la vida solo era un desafío de programación y una manera divertida de usar ciclos de la CPU. Para otros, sin embargo,

el juego adquirió más connotaciones filosóficas.

2 Desarrollo

2.1 El juego

Se trata de un juego de cero jugadores, lo que quiere decir que su evolución está determinada por el estado inicial y no necesita ninguna entrada de datos posterior. El "tablero de juego" es una malla plana formada por cuadrados (las "células") que se extiende por el infinito en todas las direcciones. Por tanto, cada célula tiene 8 células "vecinas", que son las que están próximas a ella, incluidas las diagonales. Las células tienen dos estados: están "vivas" o "muertas" (o "encendidas" y "apagadas"). El estado de las células evoluciona a lo largo de unidades de tiempo discretas (se podría decir que por turnos). El estado de todas las células se tiene en cuenta para calcular el estado de las mismas al turno siguiente.

Todas las células se actualizan simultáneamente en cada turno, siguiendo estas reglas:

1. Una célula muerta con exactamente 3 células vecinas vivas "nace" (es decir, al turno siguiente estará viva).
2. Una célula viva con 2 o 3 células vecinas vivas sigue viva, en otro caso muere (por "soledad" o "superpoblación").

2.2 Implementación del juego de la vida en código

Tecnologías utilizadas para el desarrollo del juego de la vida

- Python. Como lenguaje de programación para desarrollo con simplicidad, versatilidad y rapidez.
- Pygame. Al ser un conjunto de módulos del lenguaje Python permiten la creación de videojuegos en dos dimensiones de una manera sencilla.
- Matplotlib. Permite el desarrollo de gráficas en tiempo real, al ser una biblioteca para la generación de gráficos a partir de datos contenidos en listas o arrays en el lenguaje de programación Python y su extensión matemática NumPy.

Primera iteración

En esta parte se comenzó conociendo el entorno de trabajo, realice algunas practicas sencillas con la librería de pygame, esto porque aun no estaba familiarizado con la biblioteca. Ya familiarizado con la biblioteca comencé a desarrollar el juego de la vida.

Algoritmo 1:

1. Crear una matriz de $n \times m$
2. Llenar la de ceros

3. Introducir la posición de las células vivas, teniendo en cuenta que un cero dentro de la matriz representa una célula muerta y un uno una célula viva.
4. Recorrer la matriz evaluando las reglas del juego de la vida
5. Modificar la matriz con los nuevos valores ya habiendo aplicado las reglas del juego de la vida.
6. Repetir el proceso hasta que ya no se posible aplicar las reglas.

Observaciones.

- Fue relativamente fácil desarrollar el entorno gráfico en python y pygame.
- El programa se volvía lento con matrices mayores a 500 y tronaba con 1000.
- El algoritmo es lento porque debe recorrer matrices muy grandes, donde muchas veces evalúa valores innecesarios.

Segunda iteración

Teniendo en cuenta lo que paso en la primera iteración preste atención a la optimización del primer algoritmo, así que me senté a pensar un poco en como mejorar el algoritmo. El punto mas importante fue el recorrido de la matriz, ya que era esta parte la que hacia lento el programa, lo que hice fue seguir usando la matriz anterior, pero me apoye de una lista, en la cual guardaba las coordenadas de las células vivas, este arreglo mejore bastante el algoritmo porque al tener los valores de las células vivas en la lista, ya no era necesario recorrer toda la matriz ya que ya se sabia donde ir a evaluar una célula viva.

Algoritmo 2:

1. Crear una matriz de $n \times m$
2. Llenar la de ceros
3. Crear una lista unidimensional
4. Introducir la posición de las células vivas, teniendo en cuenta que un cero dentro de la matriz representa una célula muerta y un uno una célula viva.
5. Introducir en la lista unidimensional solo los valores de las células vivas
6. Recorrer la lista para obtener la coordenada de la célula viva dentro de la matriz.
7. Aplicar las reglas del juego de la vida a las células vivas dentro de la matriz
8. Modificar la matriz con los nuevos valores ya habiendo aplicado las reglas del juego de la vida.
9. Repetir el proceso hasta que ya no se posible aplicar las reglas.

Observaciones.

- Solo se evalúan las células vivas, no se evalúan las células muertas, lo que provoca que el algoritmo sea mas rápido pero no haga lo que estamos buscando.
- El programa ya trabajaba con matrices de 5000x5000 células

Tercera iteración

Esta parte del desarrollo fue un poco pesada para mi, ya que no se me ocurría mucho para solucionar la parte de las células muertas, me cansé un poco de pensar en una solución para el segundo algoritmo que incluso pensé en usar el primer algoritmo y usar un poco Cython para optimizar el código. Pero no estaba del todo convencido por lo que continué buscando soluciones, lo que se ocurrió fue que si ya tenía los valores de las células vivas podía usarlas para obtener las células muertas que eran candidatas a nacer. Una célula muerta que es candidata a nacer es una célula que esta dentro del vecindario de la célula viva que se esta analizando en un momento dado, entonces, cuando se encuentra una célula que es candidata lo que se hace es saltar a evaluar a esa célula con las reglas del juego de la vida. El algoritmo creció un poco sin embargo así era mejor que el primero ya que aun permitía procesar matrices de 5000x5000 células.

Algoritmo 3:

1. Crear una matriz de nxm
2. Llenar la de ceros
3. Crear una lista unidimensional
4. Introducir la posición de las células vivas, teniendo en cuenta que un cero dentro de la matriz representa una célula muerta y un uno una célula viva.
5. Introducir en la lista unidimensional solo los valores de las células vivas
6. Recorrer la lista para obtener la coordenada de la célula viva dentro de la matriz.
7. Al obtener las coordenadas de una célula viva, revisar en un vecindario si existen células muertas, si es así, entonces se debe aplicar las reglas del juego de la vida a cada célula muerta dentro del vecindario de la célula viva.
8. Aplicar las reglas del juego de la vida a las células vivas dentro de la matriz
9. Modificar la matriz con los nuevos valores ya habiendo aplicado las reglas del juego de la vida.
10. Repetir el proceso hasta que ya no se posible aplicar las reglas.

Observaciones.

- Con este algoritmo el sistema funciona de manera correcta además que ya se puede trabajar con matrices de 5000x5000 células, sin tener ningún problema.
- En esta parte de la implementación me lleve un poco de tiempo, por la solución del problema y la implementación en código, principalmente por problemas de lógica en el código y porque aun no dominaba muy bien pygame.

3 Funcionamiento

3.1 Vista 1

La primer ventana que aparece cuando se inicia el programa nos permite darle la configuración inicial al juego de la vida. Nos permite insertar un tamaño de universo, insertar las reglas para la supervivencia y nacimiento de las células y por ultimo nos permite insertar un archivo de texto el cual ya contenga las coordenadas de las células vivas en un universo. Para iniciar el juego se deben insertar cada una de las configuraciones y dar clic en iniciar, cabe mencionar que la opción de abrir archivo es opcional.

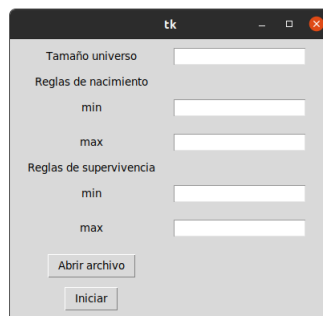


Figure 2: 1

3.2 Vista 2

Esta interfaz nos permite navegar en la computadora donde se esta corriendo el juego, esto con la finalidad de seleccionar un archivo que ya contenga las coordenadas de las células vivas.

3.3 Vista 3

En la imagen se observa como ya se han introducido cada una de las opciones que pide el programa.

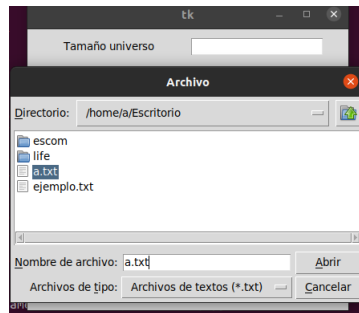


Figure 3: 2

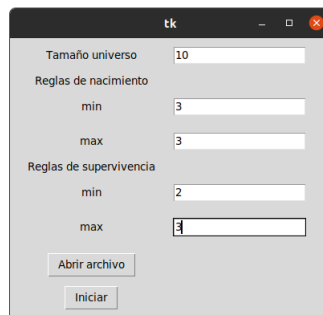


Figure 4: 3

3.4 Vista 4

Aquí ya se ve la interfaz del juego, para controlar el juego es importante conocer los controles.

- Pausa y reanudar juego - Flecha derecha
- Ver Gráfica - Flecha arriba
- Guardar progreso - Flecha izquierda
- Botón izquierdo de mouse - Coloca una célula viva en la posición del puntero
- Botón derecho de mouse - Coloca una célula muerta en la posición del puntero
- Scroll del mouse - Zoom

Con esos sencillos controles ya se puede comenzar a jugar.

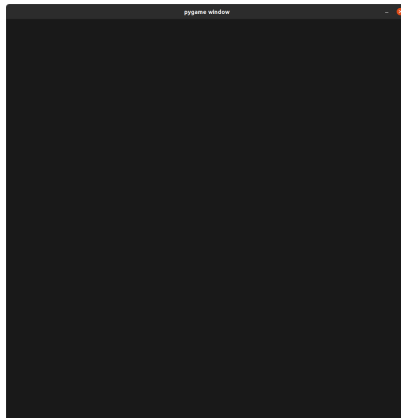


Figure 5: 4

3.5 Vista 5

En esta interfaz ya podemos ver como ya se hacen colocado unas cuantas células vivas en el universo del juego.

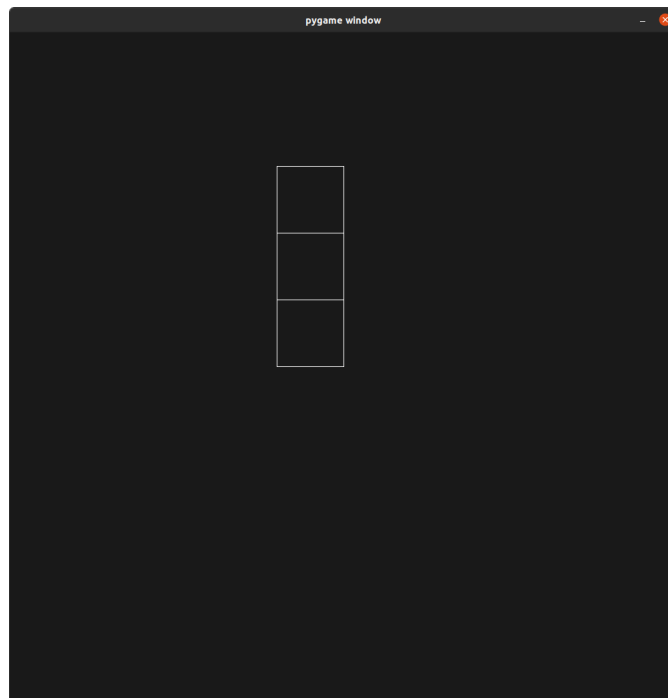


Figure 6: 5

3.6 Vista 6

Ahora en esta interfaz se puede apreciar que se han colocado algunas células muertas y otras vivas.

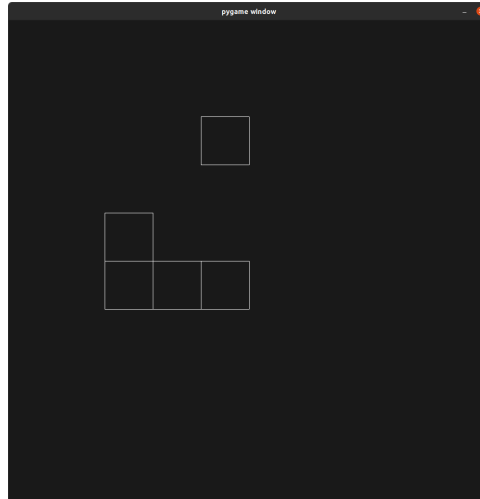


Figure 7: 6

3.7 Vista 7

En esta otra vista podemos ver la gráfica que nos permite ver el cambio del numero de células vivas en cada iteración.

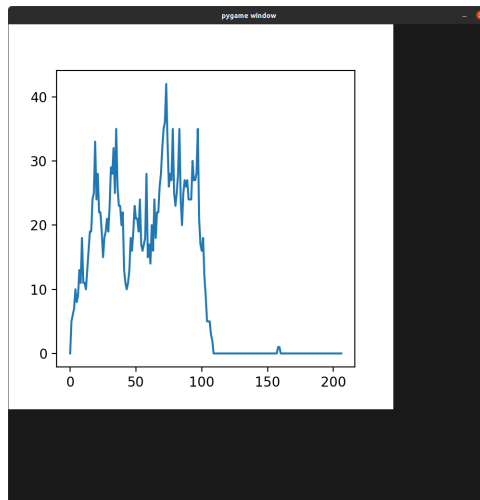


Figure 8: 7

3.8 Vista 8

Finalmente si deseamos guardar todo lo que ha pasado en nuestro universo lo hacemos en la siguiente interfaz la cual nos da la opción nuevamente de navegar en nuestro computador para guardar nuestro progreso.

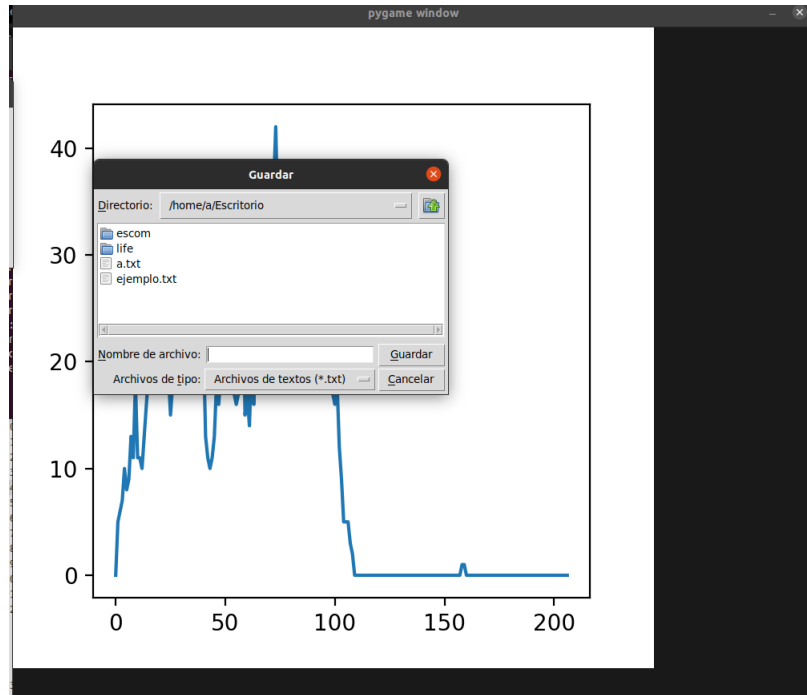


Figure 9: 8

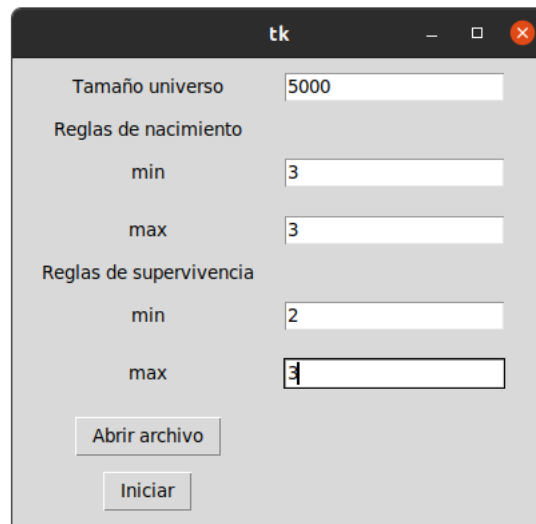
3.9 Vista 9

Aquí podemos ver el archivo txt que el programa ha guardado, con las coordenadas de todas las células que se encontraban en el ultimo momento antes de guardar el progreso.

1	1	89
2	2	81
3	2	87
4	2	89
5	3	79
6	3	80
7	3	82
8	3	88
9	4	82
10	4	84
11	4	90
12	5	55
13	5	56
14	5	57
15	5	79
16	5	82
17	5	84
18	5	85
19	5	89
20	5	90
21	6	26
22	6	27
23	6	60
24	6	81
25	6	82
26	6	83
27	6	84
28	6	85
29	7	25
30	7	28
31	7	60
32	7	81
33	7	82
34	7	83
35	7	84
36	8	26
37	8	27
38	8	60
39	8	82
40	8	83
41	13	32
42	13	33
43	14	32
44	15	33
45	15	46
46	15	64
47	15	65
48	16	27
49	16	28
50	16	45
51	16	46

Figure 10: 12

3.10 Pruebas con una matriz de 5000x5000



The image shows a Tkinter window titled 'tk' with a light gray background. It contains several input fields and two buttons. The first section is 'Tamaño universo' with a text box containing '5000'. The second section is 'Reglas de nacimiento' with 'min' and 'max' labels and text boxes containing '3'. The third section is 'Reglas de supervivencia' with 'min' and 'max' labels and text boxes containing '2' and '3' respectively. At the bottom, there are two buttons: 'Abrir archivo' and 'Iniciar'.

Figure 11: Se introduce la configuración para una matriz de 5000x5000 además de las reglas originales del juego de la vida.

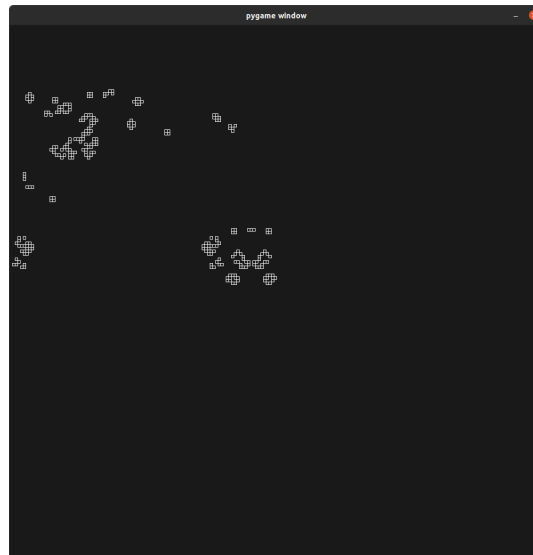


Figure 12: Se observa el comportamiento de las células en universo, vemos que el juego corre de manera fluida.



Figure 13: Movemos un poco la vista del universo para observar mejor los costados de del universo.

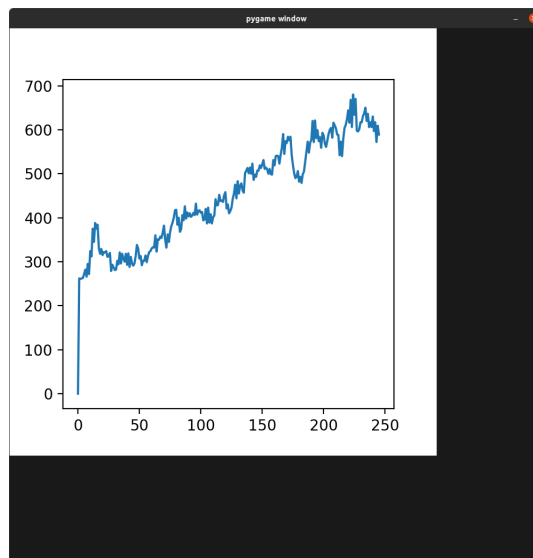


Figure 14: Se observa la gráfica donde vemos como ha ido cambiando el numero de células vivas.

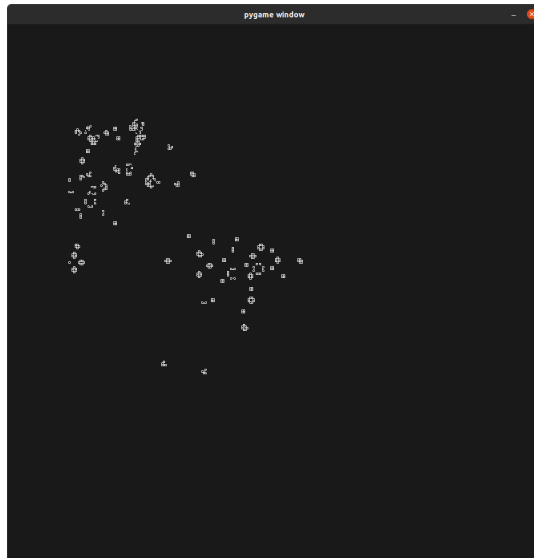


Figure 15: Volvemos a la vista del universo, nos alejamos un poco para ver como esta interactuando con el universo completo.

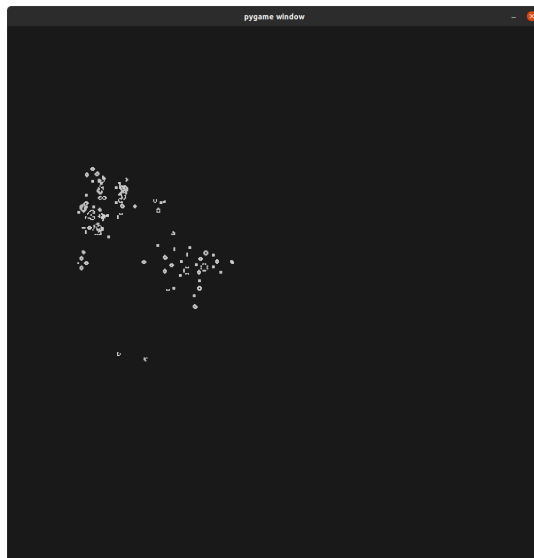


Figure 16: Nos alejamos un poco mas de las células vivas.

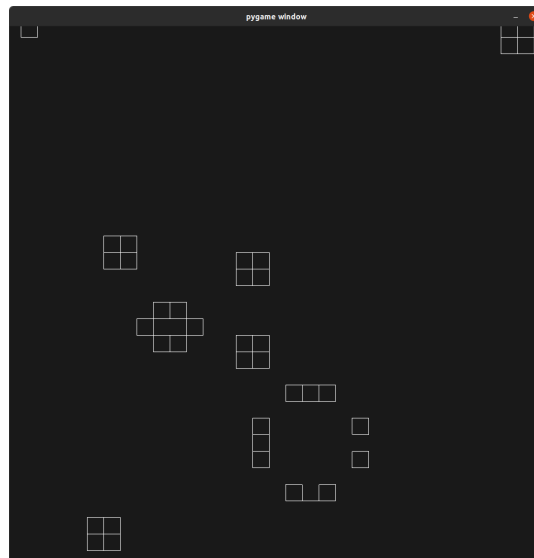


Figure 17: Nos acercamos a una sección del conjunto de células vivas y podemos observar algunas células cicladas.

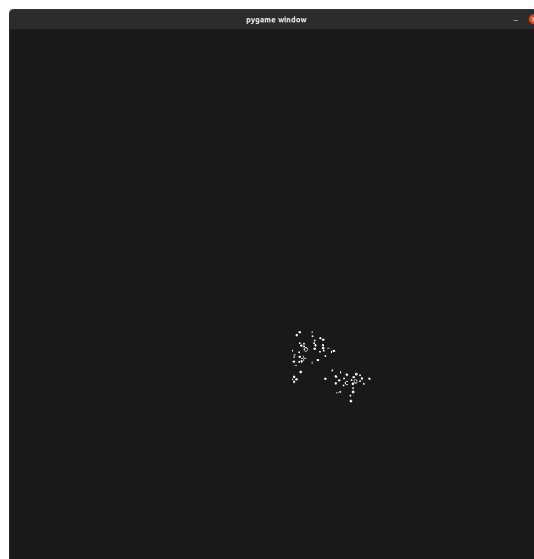


Figure 18: Finalmente volvemos a alejar el conjunto de las células vivas, y observamos que el programa se mantiene estable.

Figure 19: 29

3.11 Código de implementación

```
import pygame, sys
import numpy as np
import time

import matplotlib.pyplot as plt
from matplotlib import pyplot
from matplotlib.animation import FuncAnimation

import matplotlib.animation as animation
from random import randrange
from datetime import datetime

#IMPORTAMOS TODA LA LIBRERIA
from tkinter import filedialog
import tkinter

from OpenGL.GL import *
from OpenGL.GLU import *

import matplotlib
matplotlib.use("Agg")

import matplotlib.backends.backend_agg as agg

import pylab

import pygame
from pygame.locals import *

xe = [0]
ye = [0]
cadena = ""
ventan = tkinter.Tk()
v = tkinter.Entry(ventan)
supervivencia1 = tkinter.Entry(ventan)
nacimiento1 = tkinter.Entry(ventan)

supervivencia2 = tkinter.Entry(ventan)
nacimiento2 = tkinter.Entry(ventan)

estados = []
newestados = []
variable = 0

#CREAMOS LAS VENTANAS
```

```

def abrirArchivo():
    global cadena
    archivo = filedialog.askopenfilename(title="Archivo", filetypes=[("Archivos de textos",
f = open(archivo)
cadena = f.read()

def funcion():
    global xe
    global ye
    global v
    global estados
    global newestados
    global variable
    global planox
    global planoy
    global nacimiento1
    global nacimiento2
    global supervivencia1
    global supervivencia2

    iteraciones = 0
    vivas = 0

    variable = (int)(v.get())

    min_nacimientos = (int)(nacimiento1.get())
    max_nacimientos = (int)(nacimiento2.get())

    min_supervivencia =(int)(supervivencia1.get())
    max_supervivencia =(int)(supervivencia2.get())

    ventan.destroy()

##SE CREA LA PANTALLA
pygame.init()
width, height = 1000,1000

#SE CREA LA PANTALLA
screen = pygame.display.set_mode((height, width))

#COLOR DE LA PANTALLA
bg = 25, 25, 25

screen.fill(bg)

nxC, nyC = variable, variable

if variable >=1000:
    dimCH = 5

```

```

dimCW = 5
else:
dimCW = (width / nxC)
dimCH = (height / nyC)

#ESTADO DE LA CELDA VIVA O MUERTA

gameState = np.zeros((nxC, nyC))

aux = ""
for x in range(0, len(cadena)-1):
try:
entero = int(cadena[x])
#print(entero)
aux = aux + cadena[x]
except ValueError:
#print(int(aux), "termino de linea")
if cadena[x] == '\n':
gameState[posiX, int(aux)] = 1
estados.append((posiX, int(aux)))
else:
posiX = int(aux)
aux = ""
pauseExect = True
grafica = False

#BUCLE DE EJECUCION
while True:

#SALIR DE LA PANTALLA
for event in pygame.event.get():
if event.type == pygame.QUIT:
sys.exit()

newGameState = np.copy(gameState)

#PINTAR LA PANTALLA
screen.fill(bg)

#ZONA DE DIBUJO
time.sleep(0.01)
ev = pygame.event.get()

for event in ev:
if event.type == pygame.KEYDOWN:
if event.key == pygame.K_LEFT:
pauseExect = not pauseExect
iteraciones = iteraciones - 1

```

```

if event.key == pygame.K_RIGHT:
    pauseExect = not pauseExect
    iteraciones = iteraciones - 1
    guardado = filedialog.asksaveasfile(mode='w', title="Guardar", confirmoverwrite=True, de
    if guardado is None:
        print("No se guardo")
    for x in range(0, nxC):
        for y in range(0, nyC):
            if newGameState[x,y] == 1:
                texto = str(x) + " " + str(y) + '\n'
                guardado.write(texto)
            guardado.close()

if event.key == pygame.K_UP:
    grafica = not grafica

mouseClick = pygame.mouse.get_pressed()

#DIBUJAR CELULAS
if sum(mouseClick) > 0:
    posX, posY = pygame.mouse.get_pos()
    celX, celY = int(np.floor(posX / dimCW)), int(np.floor(posY / dimCH))
    if(newGameState[celX, celY] != 1):
        estados.append((celX, celY))

newGameState[celX, celY] = not mouseClick[2]

if event.type == pygame.MOUSEBUTTONDOWN:
    posX, posY = pygame.mouse.get_pos()
    celX, celY = int(np.floor(posX / dimCW)), int(np.floor(posY / dimCH))

print(planox)
print(planoy)

if event.button == 4:
    posX, posY = pygame.mouse.get_pos()
    celX, celY = int(np.floor(posX / dimCW)), int(np.floor(posY / dimCH))

    dimCW= dimCW + 1;
    dimCH = dimCH + 1;
    planox = planox - celX
    planoy = planoy - celY

#variable = variable - 1
elif event.button == 5 and dimCW > 1 and dimCH > 1:
    if (dimCH) * (variable) > width:

dimCW= dimCW - 1;

```

```

dimCH = dimCH - 1;

planox = planox + celX
planoy = planoy + celY
vivas = 0
a=0
b=0

newestados = estados.copy()

for x in range(0, len(estados)):
    a,b = estados[x]
    #print("estados")
    #print(estados)

    #print("nuevos estados")
    #print(newestados)

    #print("particular ", estados[x])

    #time.sleep(1)
    if newGameState[a,b] == 1:
        vivas = vivas + 1
        if not pauseExect:
            n_height = gameState[(a-1) % nxC, (b-1) % nyC] + \
            gameState[(a) % nxC, (b-1) % nyC] + \
            gameState[(a+1) % nxC, (b-1) % nyC] + \
            gameState[(a-1) % nxC, (b) % nyC] + \
            gameState[(a+1) % nxC, (b) % nyC] + \
            gameState[(a-1) % nxC, (b+1) % nyC] + \
            gameState[(a) % nxC, (b+1) % nyC] + \
            gameState[(a+1) % nxC, (b+1) % nyC]

        #regla 2
        if gameState[a,b] == 1 and (n_height < min_supervivencia or n_height > max_supervivencia):
            newGameState[a,b] = 0
            newestados.remove(estados[x])

        #regla 1

        if gameState[(a-1) % nxC, (b-1) % nyC] == 0:
            ceroX = (a-1) % nxC
            ceroY = (b-1) % nyC
            ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
            gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
            gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \

```

```

gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
    newGameState[ceroX,ceroY] = 1
    newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( ((ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( ((ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( ((ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a-1) % nxC, (b+1) % nyC] == 0:
    ceroX = (a-1) % nxC
    ceroY = (b+1) % nyC
    ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
    newGameState[ceroX,ceroY] = 1
    newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( ((ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( ((ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( ((ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a+1) % nxC, (b-1) % nyC] == 0:
    ceroX = (a+1) % nxC
    ceroY = (b-1) % nyC
    ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \

```

```

gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
    newGameState[ceroX,ceroY] = 1
    newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
        ((ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
        ((ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
        ((ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a+1) % nxC,(b+1) % nyC] == 0:
    ceroX = (a+1) % nxC
    ceroY = (b+1) % nyC
    ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
    newGameState[ceroX,ceroY] = 1
    newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
        ((ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
        ((ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
        ((ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a) % nxC, (b-1) % nyC] == 0:
    ceroX = (a) % nxC
    ceroY = (b-1) % nyC
    ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:

```

```

newGameState[ceroX,ceroY] = 1
newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( (ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a-1) % nxC, (b) % nyC] == 0:
ceroX =(a-1) % nxC
ceroY =(b) % nyC
ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
newGameState[ceroX,ceroY] = 1
newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( (ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a+1) % nxC, (b) % nyC] == 0:
ceroX = (a+1) % nxC
ceroY = (b) % nyC
ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
newGameState[ceroX,ceroY] = 1
newestados.append((ceroX,ceroY))

```



```

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( (ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

if gameState[(a) % nxC, (b+1) % nyC] == 0:
ceroX = (a) % nxC
ceroY = (b+1) % nyC
ceros = gameState[(ceroX-1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY-1) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY) % nyC] + \
gameState[(ceroX-1) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX) % nxC, (ceroY+1) % nyC] + \
gameState[(ceroX+1) % nxC, (ceroY+1) % nyC]

if ceros >= min_nacimientos and ceros <= max_nacimientos:
newGameState[ceroX,ceroY] = 1
newestados.append((ceroX,ceroY))

poly = [( (ceroX) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, (ceroY * dimCH) + planoy),
( (ceroX+1) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy),
( (ceroX) * dimCW) + planox, ((ceroY+1) * dimCH) + planoy)]

pygame.draw.polygon(screen, (255,255,255), poly, 1)

#DIBUJAMOS EL CUADRAD
poly = [ ( (a) *dimCW) + planox, (b * dimCH) + planoy),
( (a+1) * dimCW) + planox, (b * dimCH) + planoy),
( (a+1) * dimCW) + planox, ((b+1) * dimCH) + planoy),
( (a) * dimCW) + planox, ((b+1) * dimCH) + planoy)]

if newGameState[a,b] == 0:
pygame.draw.polygon(screen, (25,25,25), poly, 1)
else:
pygame.draw.polygon(screen, (255,255,255), poly, 1)

newestados = list(set(newestados))

estados = newestados.copy()

if pauseExect == False:
ye.append(vivas)
#print(ye)

#GRAFICA

```

```

if grafica == True:
    fig = pylab.figure(figsize=[4, 4],
                        dpi=200,
                        )
    ax = fig.gca()
    ax.plot(ye)

    canvas = agg.FigureCanvasAgg(fig)
    canvas.draw()
    renderer = canvas.get_renderer()
    raw_data = renderer.tostring_rgb()

pygame.init()

size = canvas.get_width_height()

surf = pygame.image.fromstring(raw_data, size, "RGB")
screen.blit(surf, (0,0))

gameState = np.copy(newGameState)

#ACTUALIZAR PANTALLA
pygame.display.flip()

def grafica():
    fig = pylab.figure(figsize=[4, 4],
                        dpi=100,
                        )
    ax = fig.gca()
    ax.plot([1, 2, 4])

    canvas = agg.FigureCanvasAgg(fig)
    canvas.draw()
    renderer = canvas.get_renderer()
    raw_data = renderer.tostring_rgb()

pygame.init()

ventana_graf = pygame.display.set_mode((600, 400), DOUBLEBUF)
screen = pygame.display.get_surface()

tamanio = canvas.get_width_height()

surf = pygame.image.fromstring(raw_data, tamanio, "RGB")
screen.blit(surf, (0,0))
pygame.display.flip()

crash = False
while not crash:
    for event in pygame.event.get():

```

```

if event.type == pygame.QUIT:
    crash = True

def ventana():
    global ventan,v
    global nacimiento1
    global nacimiento2
    global supervivencia1
    global supervivencia2

    ventan.geometry("400x350")
    variable = 0

    tamaño = tkinter.Label(ventan, text ="Tamaño universo")
    tamaño.grid(row=0, column=0, padx=20)

    v.grid(row=0, column=1, pady=10)

    ##REGLA DE NACIMIENTO
    mas_nacido = tkinter.Label(ventan, text ="Reglas de nacimiento")
    mas_nacido.grid(row=1, column=0, padx=20)

    mas_nacido = tkinter.Label(ventan, text ="min")
    mas_nacido.grid(row=2, column=0, padx=20)

    nacimiento1.grid(row=2, column=1, pady=10)

    menos_nacido = tkinter.Label(ventan, text ="max")
    menos_nacido.grid(row=3, column=0, padx=20)

    nacimiento2.grid(row=3, column=1, pady=10)

    ##REGLA DE SUPERVIVENCIA
    mas_super = tkinter.Label(ventan, text ="Reglas de supervivencia")
    mas_super.grid(row=4, column=0, padx=20)

    mas_super = tkinter.Label(ventan, text ="min")
    mas_super.grid(row=5, column=0, padx=20)

    supervivencia1.grid(row=5, column=1, pady=10)

    menos_super = tkinter.Label(ventan, text ="max")
    menos_super.grid(row=6, column=0, padx=20)

    supervivencia2.grid(row=6, column=1, pady=10)

    guardar = tkinter.Button(ventan, text="Abrir archivo", command=abrirArchivo)
    guardar.grid(row=7, column=0 , pady=10)

    b = tkinter.Button(ventan, text ="Iniciar", width=0, height=0, command=funcion)

```

```
b.grid(row=8, column=0)
```

```
ventan.mainloop()
```

```
ventana()
```

4 Conclusiones

Lo primero que puedo mencionar es que ha sido un proyecto muy interesante de desarrollar, principalmente por las adecuaciones que se tuvieron que realizar el procesamiento de grandes cantidades de células, ya que como lo mencione en una parte del reporte estuve a punto de optar por una opción que me permitiera usar un algoritmo sencillo pero elevado en complejidad computacional, pero con un poco de paciencia logre adecuar el algoritmo original para que me permitiera correr mi programa sin problemas y con una matriz bastante grande.

Durante el desarrollo del proyecto logre aprender un poco mas de algunas tecnologías que desconocía, no voy a mentir al principio me resulto un poco complicado el abstraer ciertos concepto y entender algunos otros aun así logre seguir avanzando, debo mencionar que al terminar este proyecto me di cuenta que a veces es importante buscar mas de una solución para un problema, ya que el mas fácil puede ser el que de mas problemas.

Por otra parte he quedado realmente sorprendido del juego, es impresionante como con unas cuantas reglas se pueden construir "universos" los cuales pueden expandirse, sobrevivir y morir. Este juego tan pequeño me ha dejado pensando un poco mas sobre la existencia y mas aun cuando vemos algunas de sus configuraciones que a simple vista se ven sencillas pero esas configuraciones tan sencillas son capaces de generar una gran cantidad de células vivas que interactúan en el universo y entre si, lo que te hace darte cuenta de lo impresionante que resulta la existencia y mas aun la "suerte" para que todo apareciera.

5 Referencias

1. colaboradores de Wikipedia. (2020, 14 octubre). Juego de la vida. Wikipedia, la enciclopedia libre. https://es.wikipedia.org/wiki/Juego_de_la_vida
2. pygame - Empezando con pygame | pygame Tutorial. (2018, 1 octubre). Rip Tutorial. <https://riptutorial.com/es/pygame>
3. Matplotlib - Matplotlib | Matplotlib: Visualization with Python. (2012, 10 febrero). Matplotlib. <https://matplotlib.org/>
4. pygame - pygame | Pygame Front Page. (2015, 2 septiembre). Matplotlib. <https://www.pygame.org/docs/>