Entrega de trabajos del segundo parcial

Barrera Pérez Carlos Tonatihu
Profesor: Genaro Juárez Martínez
Computing Selected Topics
Grupo: 3CM8

4 de noviembre de 2018

Índice

1. Juego de la vida

1.1. Introducción

Este programa es la mejora del hecho en el primer parcial ya que en esta versión se agrega la opción de insertar mosaicos ya definidos y de poder cargar un archivo con una configuración inicial del juego de la vida. Otra funcionalidad que faltaba en la versión anterior y que se agrego en este programa es una barra para poder moverse a lo largo de toda la sección de simulación y así poder apreciar la simulación por completo.

Los patrones que se eligieron son los más comunes que se pueden encontrar con estas dos reglas y que generan nuevos patrones a lo largo de la simulación del programa. Los mosaicos que se agregaron son algunos que se generan con la regla de life y con la regla de difusión. Las siguientes figuras son las cinco que se pueden insertar en el simulador.

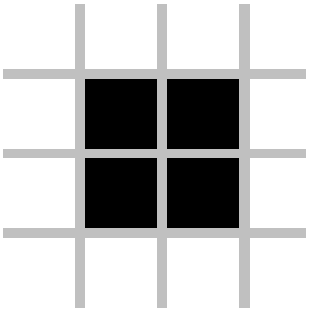

Figura 1: Patrón estático que se genera con la regla 2333

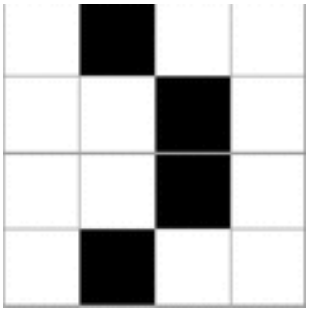

Figura 2: Glider que se genera con la regla 7722

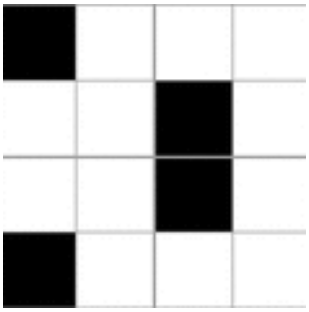

Figura 3: Glider que se genera con la regla 7722

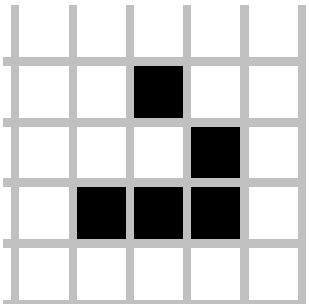

Figura 4: Glider que se genera con la regla 2333

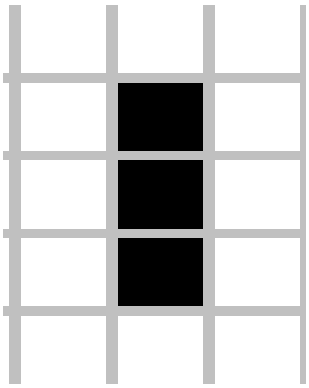

Figura 5: Oscilador que se genera con la regla 2333

1.2. Desarrollo

Archivo: gol.py En este archivo se encuentra la clase que controla todo el juego de la vida, desde el como se muestra en pantalla a como trabaja. El código fue desarrollado en python 3 y se utilizo la biblioteca tkinter.

```
from tkinter import Tk, Canvas, Frame, Button, Entry, Label,
    Scale, filedialog
from tkinter import BOTH, TOP, LEFT, HORIZONTAL, Y, RIGHT,
    VERTICAL, PhotoImage
from tkinter import Scrollbar
import numpy as np
from tkcolorpicker import askcolor
import datetime
import time


dict_tipos = {
    "nada": 0,
    "cubo": 1,
    "glider": 2,
    "glider2": 3,
    "glider3": 4,
    "oscilador": 5,
}


class Ventana(Frame):
    def __init__(self, parent):
        Frame.__init__(self, parent)
        self.parent = parent

        # Elementos interfaz
        self.ceros = "white"
        self.unos = "black"
        self.regla = [2, 3, 3, 3]
        self.e1 = None
        self.e2 = None
        self.contador = 0
        self.colorBtn1 = None
        self.colorBtn2 = None
        self.barra = None
        self.canvas = None
        self.cubo_image = None
        self.glider = None
        self.glider2 = None
        self.glider3 = None
        self.oscilador = None
        # variables del juego de la vida
```

```
42          self.pausa = True
43          self.tam = 100
44          self.tam_cuadro = 10
45          self.distribucion = .5
46          self.cuadritos = np.zeros(shape=(self.tam, self.tam),
    dtype=int)
47          self.celulas = np.random.randint(2, size=(self.tam, self
    .tam), dtype=int)
48          self.tiempo = 0
49          self.tipo_insertar = dict_tipos["nada"]
50          # Historial de unos
51          self.nom_archivo = None
52
53      def iniciar(self):
54          self.nom_archivo = "{}.csv".format(self.obtener_hora())
55          archivo = open(self.nom_archivo, "w")
56          archivo.close()
57          self.canvas.delete('all')
58          self.update_idletasks()
59          self.pausa = True
60          self.contador = 0
61          self.tiempo = 0
62          self.tam = int(self.e2.get())
63          self.tam_cuadro = 0
64          while self.tam_cuadro*self.tam < 1000:
65              self.tam_cuadro += 1
66          if self.tam_cuadro*self.tam > 1000:
67              self.tam_cuadro -= 1
68          self.distribucion = self.barra.get()/100
69          self.celulas = np.random.choice([1, 0], size=(self.tam,
    self.tam), p=[self.distribucion, 1-self.distribucion])
70          self.cuadritos = np.zeros(shape=(self.tam, self.tam),
    dtype=int)
71          texto = self.e1.get().split(",")
72          self.regla[0] = int(texto[0])
73          self.regla[1] = int(texto[1])
74          self.regla[2] = int(texto[2])
75          self.regla[3] = int(texto[3])
76          self.contar_unos()
77          print(self.contador)
78          self.re_dibujar()
79
80      def contar_unos(self):
81          for i in range(self.tam):
82              for j in range(self.tam):
83                  if self.celulas[i, j] == 1:
84                      self.contador += 1
85
86          print("contador_unos : {}".format(self.contador))
```

```
87
88      def pulsar_cuadrito(self, event):
89          item = self.canvas.find_closest(event.x, event.y)[0]
90          i, j = np.where(self.cuadritos == item)
91          print("{}, {}".format(i[0], j[0]))
92          if self.tipo_insertar == dict_tipos["nada"]:
93              if self.canvas.itemcget(item, "fill") == self.unos:
94                  self.canvas.itemconfig(item, fill=self.ceros)
95                  self.celulas[i[0]][j[0]] = 0
96                  self.contador -= 1
97              else:
98                  self.canvas.itemconfig(item, fill=self.unos)
99                  self.celulas[i[0]][j[0]] = 1
100                 self.contador += 1
101         elif self.tipo_insertar == dict_tipos["cubo"]:
102             print("cubo")
103             print(self.celulas)
104             self.insertar_cubo(i[0], j[0])
105             print(self.celulas)
106         elif self.tipo_insertar == dict_tipos["glider"]:
107             print("glider")
108             self.insertar_glider(i[0], j[0])
109         elif self.tipo_insertar == dict_tipos["glider2"]:
110             print("glider2")
111             self.insertar_glider_dos(i[0], j[0])
112         elif self.tipo_insertar == dict_tipos["glider3"]:
113             print("glider3")
114             self.insertar_glider_tres(i[0], j[0])
115         elif self.tipo_insertar == dict_tipos["oscilador"]:
116             print("oscilador")
117             self.insertar_oscilador(i[0], j[0])
118
119         self.tipo_insertar = dict_tipos["nada"]
120
121     def insertar_cubo(self, x1, y1):
122         x2 = x1 + 1
123         y2 = y1 + 1
124         item1 = self.cuadritos[x1, y1]
125         item2 = self.cuadritos[x1, y2]
126         item3 = self.cuadritos[x2, y1]
127         item4 = self.cuadritos[x2, y2]
128
129         if self.celulas[x1, y1] == 0:
130             self.celulas[x1, y1] = 1
131             self.contador += 1
132             self.canvas.itemconfig(item1, fill=self.unos)
133         if self.celulas[x1, y2] == 0:
134             self.celulas[x1, y2] = 1
135             self.contador += 1
```

```
136             self.canvas.itemconfig(item2, fill=self.unos)
137         if self.celulas[x2, y1] == 0:
138             self.celulas[x2, y1] = 1
139             self.contador += 1
140             self.canvas.itemconfig(item3, fill=self.unos)
141         if self.celulas[x2, y2] == 0:
142             self.celulas[x2, y2] = 1
143             self.contador += 1
144             self.canvas.itemconfig(item4, fill=self.unos)
145 
146     def insertar_oscilador(self, i1, j1):
147         i0 = i1 - 1
148         i2 = i1 + 1
149         item0 = self.cuadritos[i0, j1]
150         item1 = self.cuadritos[i1, j1]
151         item2 = self.cuadritos[i2, j1]
152 
153         if self.celulas[i0, j1] == 0:
154             self.celulas[i0, j1] = 1
155             self.contador += 1
156             self.canvas.itemconfig(item0, fill=self.unos)
157 
158         if self.celulas[i1, j1] == 0:
159             self.celulas[i1, j1] = 1
160             self.contador += 1
161             self.canvas.itemconfig(item1, fill=self.unos)
162 
163         if self.celulas[i2, j1] == 0:
164             self.celulas[i2, j1] = 1
165             self.contador += 1
166             self.canvas.itemconfig(item2, fill=self.unos)
167 
168     def insertar_glider(self, i1, j1):
169         j2 = j1 + 1
170         i2 = i1 + 1
171         i3 = i2 + 1
172         i4 = i3 + 1
173 
174         item1 = self.cuadritos[i1, j1]
175         item2 = self.cuadritos[i1, j2]
176         item3 = self.cuadritos[i2, j1]
177         item4 = self.cuadritos[i2, j2]
178         item5 = self.cuadritos[i3, j1]
179         item6 = self.cuadritos[i3, j2]
180         item7 = self.cuadritos[i4, j1]
181         item8 = self.cuadritos[i4, j2]
182 
183         if self.celulas[i1, j1] == 0:
184             self.celulas[i1, j1] = 1
```

```
185             self.contador += 1
186             self.canvas.itemconfig(item1, fill=self.unos)
187         if self.celulas[i1, j2] == 1:
188             self.celulas[i1, j2] = 0
189             self.contador -= 1
190             self.canvas.itemconfig(item2, fill=self.ceros)
191 
192         if self.celulas[i2, j1] == 1:
193             self.celulas[i2, j1] = 0
194             self.contador -= 1
195             self.canvas.itemconfig(item3, fill=self.ceros)
196         if self.celulas[i2, j2] == 0:
197             self.celulas[i2, j2] = 1
198             self.contador += 1
199             self.canvas.itemconfig(item4, fill=self.unos)
200 
201         if self.celulas[i3, j1] == 1:
202             self.celulas[i3, j1] = 0
203             self.contador -= 1
204             self.canvas.itemconfig(item5, fill=self.ceros)
205         if self.celulas[i3, j2] == 0:
206             self.celulas[i3, j2] = 1
207             self.contador += 1
208             self.canvas.itemconfig(item6, fill=self.unos)
209 
210         if self.celulas[i4, j1] == 0:
211             self.celulas[i4, j1] = 1
212             self.contador += 1
213             self.canvas.itemconfig(item7, fill=self.unos)
214         if self.celulas[i1, j2] == 1:
215             self.celulas[i1, j2] = 0
216             self.contador -= 1
217             self.canvas.itemconfig(item8, fill=self.ceros)
218 
219     def insertar_glider_dos(self, i1, j1):
220         i2 = i1 + 1
221         i3 = i2 + 1
222         i4 = i3 + 1
223         j2 = j1 + 1
224         j3 = j2 + 1
225 
226         item1 = self.cuadritos[i1, j1]
227         item2 = self.cuadritos[i1, j2]
228         item3 = self.cuadritos[i1, j3]
229         item4 = self.cuadritos[i2, j1]
230         item5 = self.cuadritos[i2, j2]
231         item6 = self.cuadritos[i2, j3]
232         item7 = self.cuadritos[i3, j1]
233         item8 = self.cuadritos[i3, j2]
```

```
234         item9 = self.cuadritos[i3, j3]
235         item10 = self.cuadritos[i4, j1]
236         item11 = self.cuadritos[i4, j2]
237         item12 = self.cuadritos[i4, j3]
238 
239         if self.celulas[i1, j1] == 0:
240             self.celulas[i1, j1] = 1
241             self.contador += 1
242             self.canvas.itemconfig(item1, fill=self.unos)
243         if self.celulas[i1, j2] == 1:
244             self.celulas[i1, j2] = 0
245             self.contador -= 1
246             self.canvas.itemconfig(item2, fill=self.ceros)
247         if self.celulas[i1, j3] == 1:
248             self.celulas[i1, j2] = 0
249             self.contador -= 1
250             self.canvas.itemconfig(item3, fill=self.ceros)
251 
252         if self.celulas[i2, j1] == 1:
253             self.celulas[i2, j1] = 0
254             self.contador -= 1
255             self.canvas.itemconfig(item4, fill=self.ceros)
256         if self.celulas[i2, j2] == 1:
257             self.celulas[i2, j2] = 0
258             self.contador -= 1
259             self.canvas.itemconfig(item5, fill=self.ceros)
260         if self.celulas[i2, j3] == 0:
261             self.celulas[i2, j3] = 1
262             self.contador += 1
263             self.canvas.itemconfig(item6, fill=self.unos)
264 
265         if self.celulas[i3, j1] == 1:
266             self.celulas[i3, j1] = 0
267             self.contador -= 1
268             self.canvas.itemconfig(item7, fill=self.ceros)
269         if self.celulas[i3, j2] == 1:
270             self.celulas[i3, j2] = 0
271             self.contador -= 1
272             self.canvas.itemconfig(item8, fill=self.ceros)
273         if self.celulas[i3, j3] == 0:
274             self.celulas[i3, j3] = 1
275             self.contador += 1
276             self.canvas.itemconfig(item9, fill=self.unos)
277 
278         if self.celulas[i4, j1] == 0:
279             self.celulas[i4, j1] = 1
280             self.contador += 1
281             self.canvas.itemconfig(item10, fill=self.unos)
282         if self.celulas[i4, j2] == 1:
```

```
283             self.celulas[i4, j2] = 0
284             self.contador -= 1
285             self.canvas.itemconfig(item11, fill=self.ceros)
286         if self.celulas[i4, j3] == 1:
287             self.celulas[i4, j3] = 0
288             self.contador -= 1
289             self.canvas.itemconfig(item12, fill=self.ceros)
290 
291     def insertar_glider_tres(self, i2, j2):
292         i1 = i2 - 1
293         i3 = i2 + 1
294         j1 = j2 - 1
295         j3 = j2 + 1
296 
297         item1 = self.cuadritos[i1, j1]
298         item2 = self.cuadritos[i1, j2]
299         item3 = self.cuadritos[i1, j3]
300         item4 = self.cuadritos[i2, j1]
301         item5 = self.cuadritos[i2, j2]
302         item6 = self.cuadritos[i2, j3]
303         item7 = self.cuadritos[i3, j1]
304         item8 = self.cuadritos[i3, j2]
305         item9 = self.cuadritos[i3, j3]
306 
307         if self.celulas[i1, j1] == 1:
308             self.celulas[i1, j1] = 0
309             self.contador -= 1
310             self.canvas.itemconfig(item1, fill=self.ceros)
311         if self.celulas[i1, j2] == 0:
312             self.celulas[i1, j2] = 1
313             self.contador += 1
314             self.canvas.itemconfig(item2, fill=self.unos)
315         if self.celulas[i1, j3] == 1:
316             self.celulas[i1, j2] = 0
317             self.contador -= 1
318             self.canvas.itemconfig(item3, fill=self.ceros)
319 
320         if self.celulas[i2, j1] == 1:
321             self.celulas[i2, j1] = 0
322             self.contador -= 1
323             self.canvas.itemconfig(item4, fill=self.ceros)
324         if self.celulas[i2, j2] == 1:
325             self.celulas[i2, j2] = 0
326             self.contador -= 1
327             self.canvas.itemconfig(item5, fill=self.ceros)
328         if self.celulas[i2, j3] == 0:
329             self.celulas[i2, j3] = 1
330             self.contador += 1
331             self.canvas.itemconfig(item6, fill=self.unos)
```

```
332
333         if self.celulas[i3, j1] == 0:
334             self.celulas[i3, j1] = 1
335             self.contador += 1
336             self.canvas.itemconfig(item7, fill=self.unos)
337         if self.celulas[i3, j2] == 0:
338             self.celulas[i3, j2] = 1
339             self.contador += 1
340             self.canvas.itemconfig(item8, fill=self.unos)
341         if self.celulas[i3, j3] == 0:
342             self.celulas[i3, j3] = 1
343             self.contador += 1
344             self.canvas.itemconfig(item9, fill=self.unos)
345
346     def re_dibujar(self):
347         print("REDIBUJAR")
348         for i in range(self.tam):
349             for j in range(self.tam):
350                 if self.celulas[i, j] == 0:
351                     self.cuadritos[i, j] = self.canvas.
    create_rectangle(0 + (j * self.tam_cuadro),
352
            0 + (i * self.tam_cuadro),
353
            self.tam_cuadro + (j * self.tam_cuadro),
354
            self.tam_cuadro + (i * self.tam_cuadro),
355
            fill=self.ceros, width=0, tag="btncuadrito")
356                 else:
357                     self.cuadritos[i, j] = self.canvas.
    create_rectangle(0 + (j * self.tam_cuadro),
358
            0 + (i * self.tam_cuadro),
359
            self.tam_cuadro + (j * self.tam_cuadro),
360
            self.tam_cuadro + (i * self.tam_cuadro),
361
            fill=self.unos, width=0, tag="btncuadrito")
362
363         self.canvas.tag_bind("btncuadrito", "<Button-1>", self.
    pulsar_cuadrito)
364         self.update_idletasks()
365
366     def init_ui(self):
367         self.parent.title("Juego de la vida")
368         self.pack(fill=BOTH, expand=1)
369
```

```
370         self.canvas = Canvas(self, relief='raised', width=1000,
    height=1000)
371         scroll = Scrollbar(self, orient=VERTICAL)
372         scroll.pack(side=RIGHT, fill=Y)
373         scroll.config(command=self.canvas.yview)
374 
375         self.canvas.config(yscrollcommand=scroll.set)
376         self.canvas.pack(side=LEFT)
377 
378 
379         Label(self, text="Regla:").pack(side=TOP)
380         self.e1 = Entry(self, fg="black", bg="white")
381         self.e1.insert(10, "2,3,3,3")
382         self.e1.pack(side=TOP)
383 
384         Label(self, text="Tamanio:").pack(side=TOP)
385         self.e2 = Entry(self, fg="black", bg="white")
386         self.e2.insert(10, "100")
387         self.e2.pack(side=TOP)
388 
389         Label(self, text="Porcentaje de unos").pack(side=TOP)
390         self.barra = Scale(self, from_=0, to=100, orient=
    HORIZONTAL, tickinterval=50)
391         self.barra.set(50)
392         self.barra.pack(side=TOP)
393 
394         btn_iniciar = Button(self, text="Iniciar/Reiniciar",
    command=self.iniciar)
395         btn_iniciar.pack(side=TOP)
396 
397         button1 = Button(self, text="Pausa/Reanudar", command=
    self.empezar_dentener)
398         button1.pack(side=TOP)
399 
400         self.colorBtn1 = Button(self, text="Selecciona el color
    de unos", command=self.get_color_unos, bg=self.unos)
401         self.colorBtn1.pack(side=TOP)
402 
403         self.colorBtn2 = Button(self, text="Selecciona el color
    de ceros", command=self.get_color_ceros, bg=self.ceros)
404         self.colorBtn2.pack(side=TOP)
405 
406         btn_save = Button(self, text="Guardar", command=self.
    guardar)
407         btn_save.pack(side=TOP)
408 
409         btn_cargar = Button(self, text="Cargar Matriz", command=
    self.cargar)
410         btn_cargar.pack(side=TOP)
```

```
411
412         self.cubo_image = PhotoImage(file="./data/cuadrado.png")
413         btn_cubo = Button(self, image=self.cubo_image, command=
    self.seleccionar_cubo)
414         btn_cubo.pack(side=TOP)
415
416         self.glider = PhotoImage(file="./data/glider.png")
417         self.glider = self.glider.subsample(2)
418         btn_glider = Button(self, image=self.glider, command=
    self.seleccionar_glider)
419         btn_glider.pack(side=TOP)
420
421         self.glider2 = PhotoImage(file="./data/glider2.png")
422         self.glider2 = self.glider2.subsample(2)
423         btn_glider2 = Button(self, image=self.glider2, command=
    self.seleccionar_glider_dos)
424         btn_glider2.pack(side=TOP)
425
426         self.glider3 = PhotoImage(file="./data/glider3.png")
427         btn_glider3 = Button(self, image=self.glider3, command=
    self.seleccionar_glider_tres)
428         btn_glider3.pack(side=TOP)
429
430         self.oscilador = PhotoImage(file="./data/oscilador.png")
431         btn_osilador = Button(self, image=self.oscilador,
    command=self.seleccionar_oscilador)
432         btn_osilador.pack(side=TOP)
433
434     def seleccionar_glider(self):
435         self.tipo_insertar = dict_tipos["glider"]
436
437     def seleccionar_glider_dos(self):
438         self.tipo_insertar = dict_tipos["glider2"]
439
440     def seleccionar_glider_tres(self):
441         self.tipo_insertar = dict_tipos["glider3"]
442
443     def seleccionar_oscilador(self):
444         self.tipo_insertar = dict_tipos["oscilador"]
445
446     def seleccionar_cubo(self):
447         self.tipo_insertar = dict_tipos["cubo"]
448
449     @staticmethod
450     def abrir_archivo():
451         print("abrir archivo")
452         ga = filedialog.askopenfilename(title="Selecciona un
    archivo",
```

```
453                                             filetypes=(("CSV", "*.
    csv"), ("Archivo de texto", "*.txt"),
454                                                        ("Todos los
    archivos", "*.*")))
455         return ga
456
457     def actualizar_color_matriz(self):
458         for i in range(self.tam):
459             for j in range(self.tam):
460                 if self.celulas[i][j] == 0:
461                     self.canvas.itemconfig(self.cuadritos[i][j],
     fill=self.ceros)
462                 else:
463                     self.canvas.itemconfig(self.cuadritos[i][j],
     fill=self.unos)
464
465         self.update_idletasks()
466
467     def get_color_unos(self):
468         color = askcolor()
469         if not color[1] is None:
470             self.unos = color[1]
471             self.colorBtn1.configure(bg=self.unos)
472             self.actualizar_color_matriz()
473
474     def get_color_ceros(self):
475         color = askcolor()
476         if not color[1] is None:
477             self.ceros = color[1]
478             self.colorBtn2.configure(bg=self.ceros)
479             self.actualizar_color_matriz()
480
481     def guardar(self):
482         temp_nom = "respaldo-{}.csv".format(self.obtener_hora())
483         archivo = open(temp_nom, 'a')
484         for i in range(self.tam):
485             for j in range(self.tam):
486                 archivo.write("{} ".format(self.celulas[i, j]))
487             archivo.write("\n")
488
489         archivo.write("\n")
490         archivo.close()
491
492     def cargar(self):
493         print("Cargar archivo")
494         temp_archivo = self.abrir_archivo()
495         self.celulas = np.loadtxt(temp_archivo, dtype=int)
496         self.canvas.delete('all')
497         self.nom_archivo = "{}.csv".format(self.obtener_hora())
```

```
498         archivo = open(self.nom_archivo, "w")
499         archivo.close()
500         texto = self.e1.get().split(",")
501         self.regla[0] = int(texto[0])
502         self.regla[1] = int(texto[1])
503         self.regla[2] = int(texto[2])
504         self.regla[3] = int(texto[3])
505         self.tam = self.celulas.shape[0]
506         self.cuadritos = np.zeros(shape=(self.tam, self.tam),
    dtype=int)
507         self.tam_cuadro = 0
508         self.contador = 0
509         while self.tam_cuadro * self.tam < 1000:
510             self.tam_cuadro += 1
511         if self.tam_cuadro * self.tam > 1000:
512             self.tam_cuadro -= 1
513         self.contar_unos()
514         self.re_dibujar()
515 
516     def empezar_dentener(self):
517         print("empezar_detener")
518         self.pausa = not self.pausa
519         self.animacion()
520 
521     def animacion(self):
522         if not self.pausa:
523             archivo = open(self.nom_archivo, "a")
524             archivo.write("{},{}\n".format(self.tiempo, self.
    contador))
525             archivo.close()
526             nueva_poblacion = self.celulas.copy()
527             for i in range(self.tam):
528                 for j in range(self.tam):
529                     vecinos = self.revisar_vecinos(i, j)
530                     if self.celulas[i, j] == 1:
531                         if vecinos < self.regla[0] or vecinos >
    self.regla[1]:
532                             nueva_poblacion[i, j] = 0
533                             self.canvas.itemconfig(self.
    cuadritos[i][j], fill=self.ceros)
534                             self.contador -= 1
535                     else:
536                         if self.regla[2] <= vecinos <= self.
    regla[3]:
537                             nueva_poblacion[i, j] = 1
538                             self.canvas.itemconfig(self.
    cuadritos[i][j], fill=self.unos)
539                             self.contador += 1
540 
```

```
541             self.celulas[:] = nueva_poblacion[:]
542             self.update_idletasks()
543             print("Termino el t={}".format(self.tiempo))
544             self.tiempo += 1
545             self.after(50, self.animacion)
546 
547     def revisar_vecinos(self, i, j):
548         vecinos = self.celulas[i - 1, j - 1]
549         vecinos += self.celulas[i - 1, j]
550         vecinos += self.celulas[i - 1, (j + 1) % self.tam]
551         vecinos += self.celulas[i, (j + 1) % self.tam]
552         vecinos += self.celulas[(i + 1) % self.tam, (j + 1) %
    self.tam]
553         vecinos += self.celulas[(i + 1) % self.tam, j]
554         vecinos += self.celulas[(i + 1) % self.tam, j - 1]
555         vecinos += self.celulas[i, j - 1]
556         return vecinos
557 
558     @staticmethod
559     def obtener_hora():
560         return datetime.datetime.fromtimestamp(time.time()).
    strftime('%Y-%m-%d_%H:%M:%S')
561 
562 
563 # 2 2 7 7
564 def main():
565     root = Tk()
566     root.geometry('1360x750+0+0')
567     app = Ventana(root)
568     app.init_ui()
569     app.mainloop()
570 
571 
572 main()
```

1.3. Pruebas

Las pruebas que se realizaron fueron insertar los mosaicos ya definidos y ver como se comportan con las reglas de vida y de difusión.

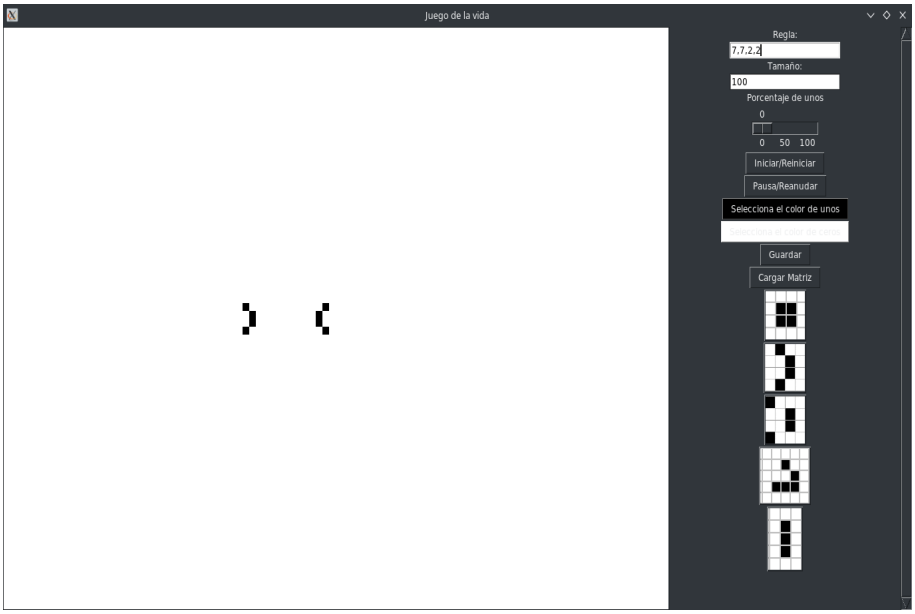

Figura 6: Probando la regla 7722 con dos glider de frente

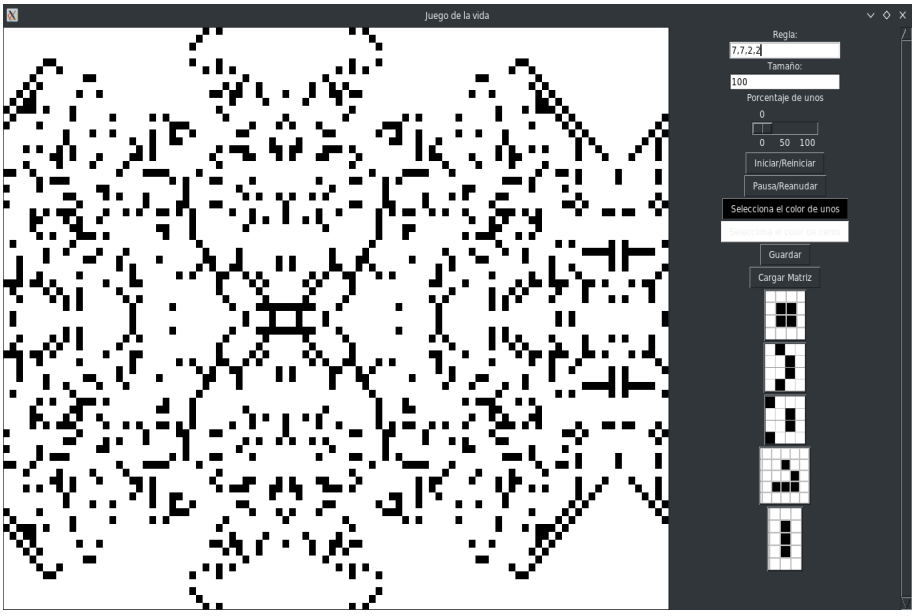

Figura 7: Resultado producido por la configuración anterior

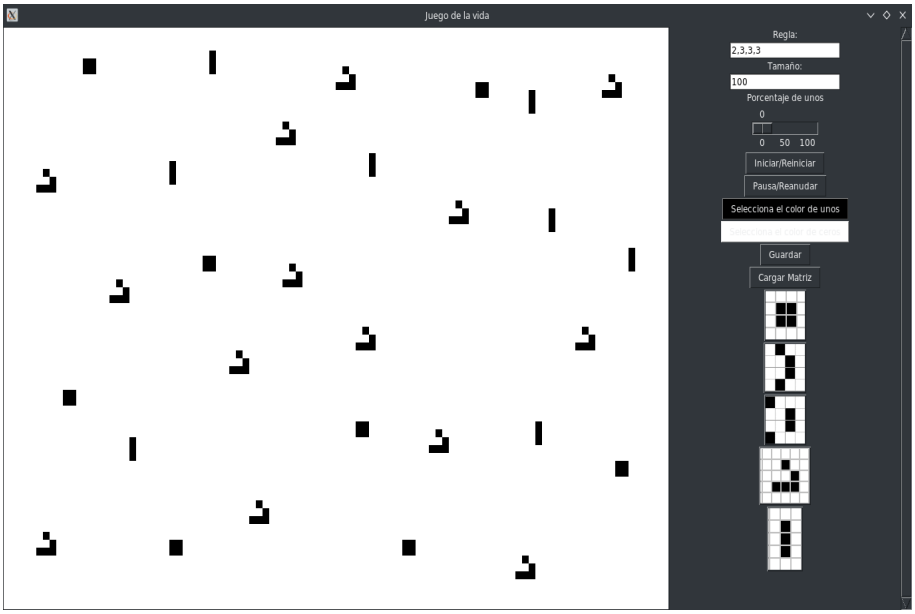

Figura 8: Probando gliders, osciladores y patrones estáticos con la regla 2333

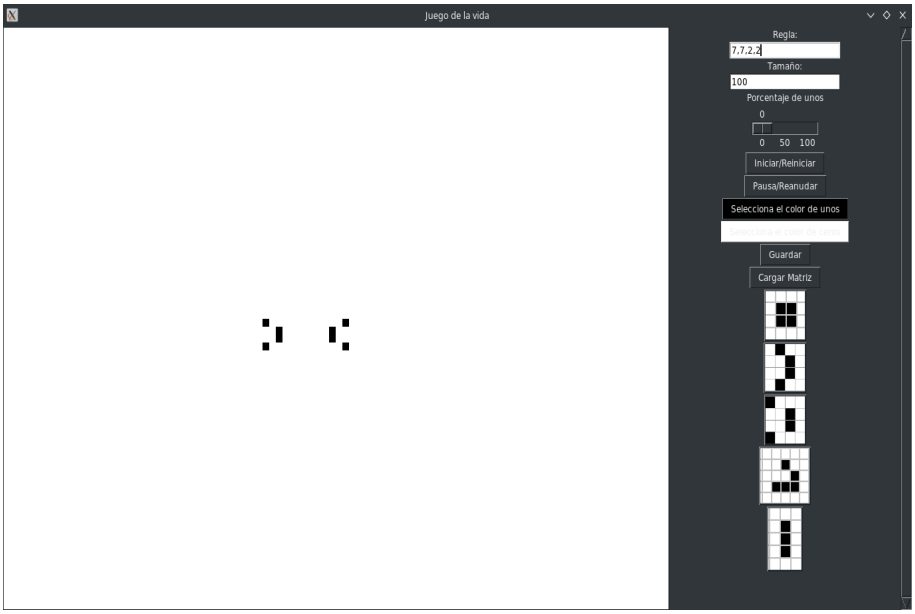

Figura 9: Probando la regla 7722 con dos glider de frente

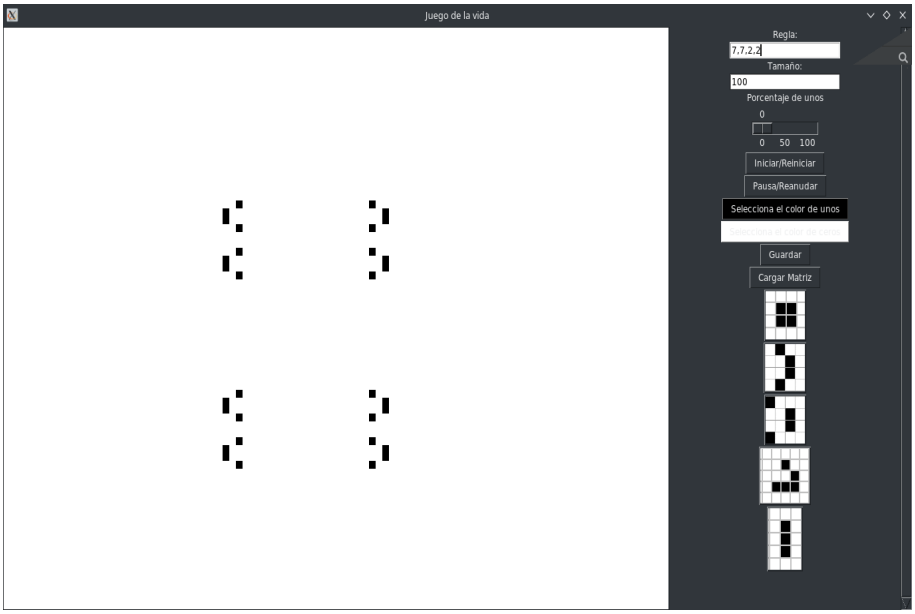

Figura 10: Resultado producido por la configuración anterior

1.4. Conclusiones

Los comportamientos que se generan al insertar distintos patrones en las dos reglas que tenemos resultan bastante interesante principalmente los gliders ya que estos son los que generan las figuras más complejas a partir de figuras tan simples y pequeñas y que con unas pocas generación crecen de una forma muy acelerada

2. Árboles

2.1. Introducción

Este programa se encarga de generar los árboles que se crean con la regla de life y la regla de difusión una matriz de 2x2 hasta 7x7, sin embargo en este caso solo se alcanzo hasta 4x4. El programa fue desarrollado en Python especificamente para Windows 10 ya que los archivos que se generan solo funcionan en Windows ya que se trabajo con la versión de WolframScript para este sistema operativo.

Al ejecutar el programa se obtienen los scripts necesarios para crear las imagenes de cada matriz, cada script se debe de ejecutar para poder producir la imagen del árbol, tambien se realizo una página web sencilla para poder comparar los grafos de cada regla.

2.2. Desarrollo

Archivo: arboles.py En este archivo se encuentra el código responsable de generar todas las transiciones entre todas las configuraciones de cada matriz para la regla de life o la regla de difusión, además genera los script de Windows y scripts de Wolfram necesarios para conectarse a Wolfram cloud y obtener las imágenes de los árboles.

```
 import numpy as np
import sys
import webbrowser

dict_tipos = {
    "life": 1,
    "diffusion": 2,
}


class Arboles:
    def __init__(self, _tam=2, _tipo=dict_tipos["life"]):
        self.tam = tam
        self.tipo = tipo
        self.vida = [2, 3, 3, 3]
        self.diffusion = [7, 7, 2, 2]
        self.regla = self.diffusion
        self.longitud = self.tam * self.tam
        self.max = 2 ** self.longitud
        self.formato = "{{:0{}b}}".format(self.longitud)
        if self.tipo == dict_tipos["life"]:
            self.regla = self.vida
```

```
23
24      def obtener_siguiente(self, m):
25          nueva_matriz = m.copy()
26          for i in range(self.tam):
27              for j in range(self.tam):
28                  suma = self.revisar_vecinos(i, j, m)
29                  if m[i, j] == 1:
30                      if suma < self.regla[0] or suma > self.regla
    [1]:
31                          nueva_matriz[i, j] = 0
32                  else:
33                      if self.regla[2] <= suma <= self.regla[3]:
34                          nueva_matriz[i, j] = 1
35          return nueva_matriz
36
37      def revisar_vecinos(self, i, j, m):
38          vecinos = m[i - 1, j - 1]
39          vecinos += m[i - 1, j]
40          vecinos += m[i - 1, (j + 1) % self.tam]
41          vecinos += m[i, (j + 1) % self.tam]
42          vecinos += m[(i + 1) % self.tam, (j + 1) % self.tam]
43          vecinos += m[(i + 1) % self.tam, j]
44          vecinos += m[(i + 1) % self.tam, j - 1]
45          vecinos += m[i, j - 1]
46          return vecinos
47
48      def numero_cadena(self, numero):
49          return self.formato.format(numero)
50
51      @staticmethod
52      def cadena_numero(cadena):
53          return int(cadena, 2)
54
55      def cadena_matriz(self, cadena):
56          m = np.zeros(shape=(self.tam, self.tam), dtype=int)
57          k = 0
58          for i in range(self.tam):
59              for j in range(self.tam):
60                  if cadena[k] == '1':
61                      m[i, j] = 1
62                  k += 1
63          return m
64
65      def matriz_cadena(self, matriz):
66          cadena = ["0"] * self.longitud
67          k = 0
68          for i in range(self.tam):
69              for j in range(self.tam):
70                  if matriz[i, j] == 1:
```

```
                    cadena[k] = "1"
                k += 1
        return "".join(cadena)

    def generar(self):
        temp_nom = "diffusion"
        if self.tipo == dict_tipos["life"]:
            temp_nom = "life"
        nombre_archivo = "tam-{}-{}".format(self.tam, temp_nom)
        archivo = open("{}.wls".format(nombre_archivo), "w")

        archivo.write("Graph[{")
        for i in range(self.max):
            cadena = self.numero_cadena(i)
            m = self.cadena_matriz(cadena)
            m_sig = self.obtener_siguiente(m)
            cadena_sig = self.matriz_cadena(m_sig)
            if i == self.max-1:
                archivo.write('"{}" -> "{}"'.format(cadena, cadena_sig))
            else:
                archivo.write('"{}" -> "{}", '.format(cadena, cadena_sig))
        if self.tam == 2:
            archivo.write('}, VertexLabels->Automatic, GraphLayout -> "RadialEmbedding"]')
        else:
            archivo.write('}, GraphLayout -> "RadialEmbedding"]')
        archivo.close()
        ejecutable = open("ejecutable-{}-{}.bat".format(self.tam, temp_nom), "w")
        ejecutable.write("set MATHE_PATH=C:\\Program Files\\Wolfram Research\\Mathematica\\11.3\n")
        ejecutable.write("set PROJECT_PATH=C:\\Users\\reymy\\Documents\\septimo\\computing-selected-topics\\arboles\n")
        ejecutable.write('"%MATHE_PATH%\\wolframscript.exe" -cloud -print -format PNG -file ')
        ejecutable.write('"%PROJECT_PATH%\\{}.wls" > {}.png'.format(nombre_archivo, nombre_archivo))
        ejecutable.close()


tam = int(sys.argv[1])
tipo = int(sys.argv[2])

life2 = Arboles(tam, tipo)
life2.generar()
```

```
110 webbrowser.open("file:///C:/Users/reymy/Documents/septimo/
    computing-selected-topics/arboles/index.html")
```

Ejemplo de script de Wolfram para Windows generado por el código anterior.

```
1 Graph[{"0000" -> "0000", "0001" -> "0110", "0010" -> "1001",
    "0011" -> "0000", "0100" -> "1001", "0101" -> "0000", "0110"
    -> "0000", "0111" -> "0000", "1000" -> "0110", "1001" ->
    "0000", "1010" -> "0000", "1011" -> "0000", "1100" -> "0000",
     "1101" -> "0000", "1110" -> "0000", "1111" -> "0000"},
    VertexLabels->Automatic, GraphLayout -> "RadialEmbedding"]
```

Ejemplo de script de Windows 10 generado por el código de python anterior. Este es el script que se genera para la regla de difusión de una matriz de 2x2 que ejecuta el script de wolfram y crea la imagen correspondiente de los árboles.

```
1 set MATHE_PATH=C:\Program Files\Wolfram Research\Mathematica
    \11.3
2 set PROJECT_PATH=C:\Users\reymy\Documents\septimo\computing-
    selected-topics\arboles
3 "%MATHE_PATH%\wolframscript.exe" -cloud -print -format PNG -file
    "%PROJECT_PATH%\tam-2-diffusion.wls" > tam-2-diffusion.png
```

2.3. Pruebas

Las pruebas se realizaron hasta una matriz de 4x4, sin embargo este tamaño de matriz es muy grande y no se puede generar una imagen de todos los arboles de este tamaño de matriz con ninguna de las dos reglas que se trabajaron.

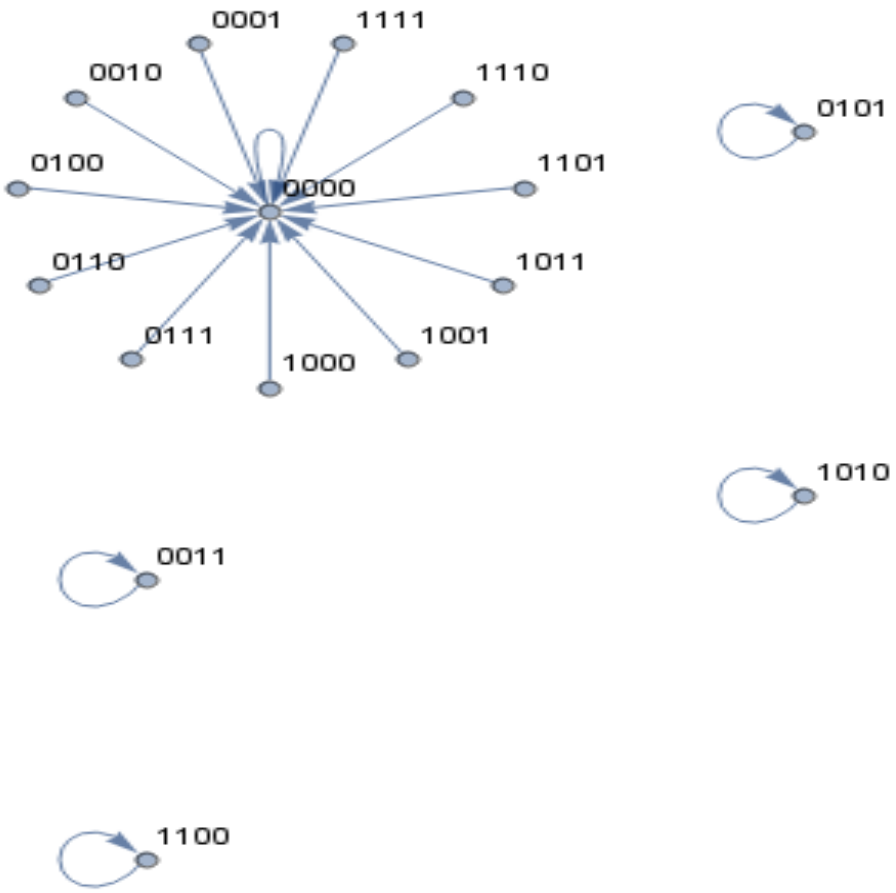

Figura 11: Árboles generados en una matriz de 2x2 con la regla de life

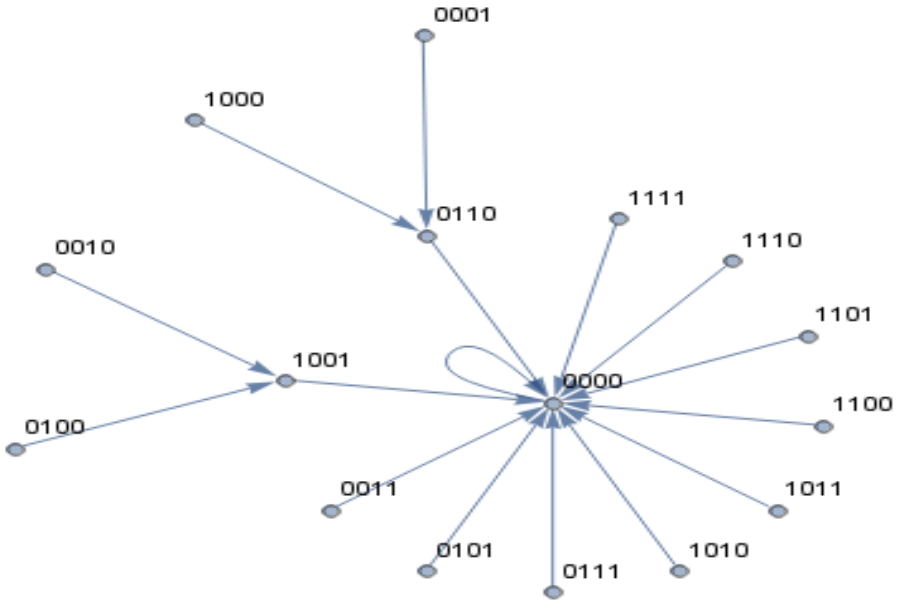

Figura 12: Árboles generados en una matriz de 2x2 con la regla de difusión

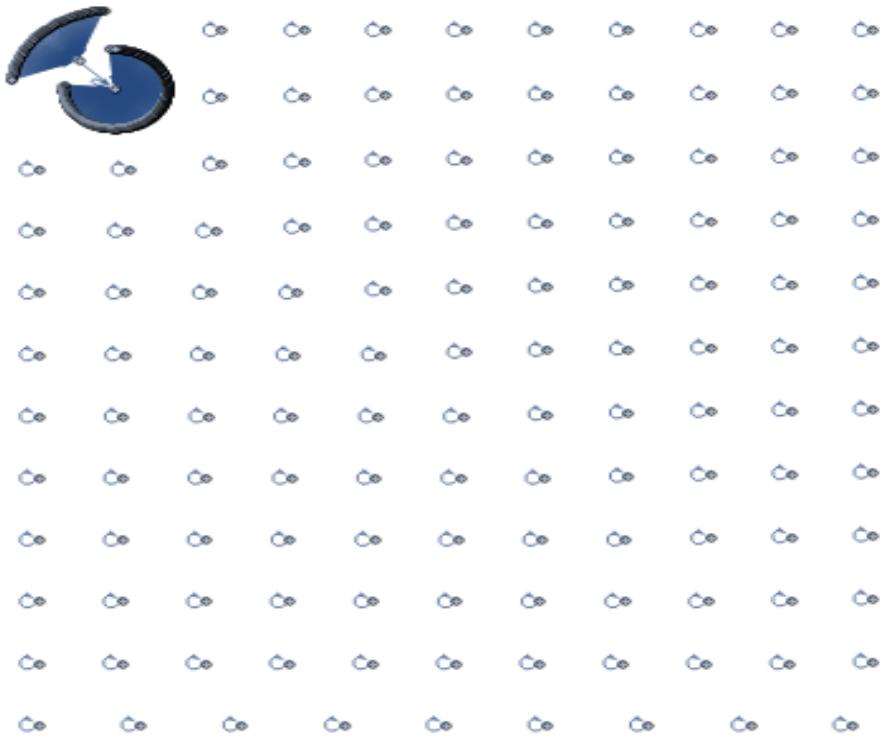

Figura 13: Árboles generados en una matriz de 3x3 con la regla de life

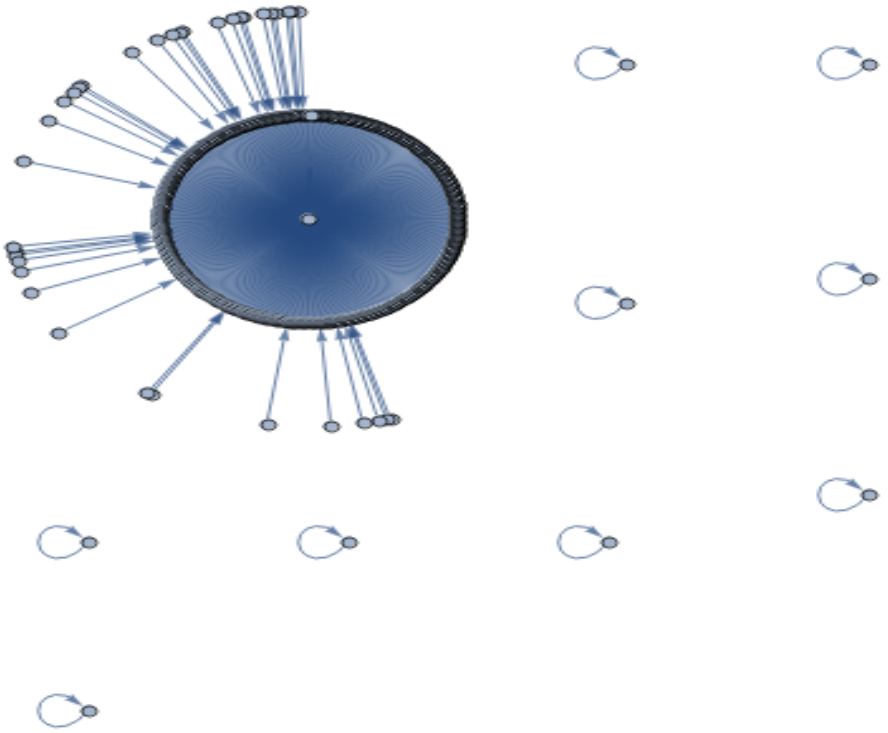

Figura 14: Árboles generados en una matriz de 3x3 con la regla de difusión

2.4. Conclusiones

A pesar de que solo se pudo observar el comportamiento en matrices de 2x2 y de 3x3 es fácil identificar el comportamiento característico de la regla de life y de la regla de difusión.

En difusión los árboles se encuentran más concentrados en menos grupos mientras que en life se generan muchos árboles lo cual es interesante considerando que en la regla de difusión la población tiende a crecer. Por lo que existe una relación en estas dos características.

Se podría decir que ya que de una configuración en especifica de la regla de difusión esta crece por lo que pasa por más configuraciones distintas que si se trabajara la misma configuración en life y es por esto que la cantidad de árboles es menor en la regla de difusión que en la de life.