

Entrega de trabajos del segundo parcial

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

11 de noviembre de 2018

Índice

1. Autómata celular (Regla de life y difusión)	3
1.1. Introducción	3
1.2. Práctica a realizar	3
1.3. Desarrollo	6
1.4. Pruebas	20
1.4.1. Análisis de poblaciones	22
1.4.2. Insertar mosaicos	42
1.5. Conclusiones	45
2. Árboles	46
2.1. Introducción	46
2.2. Desarrollo	46
2.3. Pruebas	49
2.4. Conclusiones	53
3. Hormiga de Langton	54
3.1. Introducción	54
3.2. Práctica a realizar	55
3.2.1. Hormiga de Langton original	55
3.2.2. Hormiga de Langton modificada	55
3.3. Desarrollo	56
3.4. Pruebas	71
3.4.1. Hormiga de Langton original	71
3.4.2. Hormiga de Langton modificada	73
3.5. Conclusiones	76
Referencias	77

1. Autómata celular (Regla de life y difusión)

Los autómatas celulares(AC) surgen en la década de 1940 con John Von Neumann, que intentaba modelar una máquina que fuera capaz de auto-replicarse, llegando así a un modelo matemático de dicha maquina con reglas complicadas sobre una red rectangular. Inicialmente fueron interpretados como conjunto de células que crecían, se reproducían y morían a medida que pasaba el tiempo. A esta similitud con el crecimiento de las células se le debe su nombre.[1]

Un autómata celular se caracteriza por contar con los siguientes elementos:

1. Arreglo regular. Ya sea un plano de dos dimensiones o un espacio n-dimensional, este es el espacio de evoluciones, y cada división homogénea del arreglo es llamada célula.
2. Conjunto de estados. Es finito y cada elemento o célula del arreglo toma un valor de este conjunto de estados. También se denomina alfabeto. Puede ser expresado en valores o colores.
3. configuración inicial. Consiste en asignar un estado a cada una de las células del espacio de evolución inicial del sistema.
4. Vecindades. Define el conjunto contiguo de células y posición relativa respecto a cada una de ellas. A cada vecindad diferente corresponde un elemento del conjunto de estados.
5. Función local. Es la regla de evolución que determina el comportamiento del A. C. Se conforma de una célula central y sus vecindades. Define como debe cambiar de estado cada célula dependiendo de los estados anteriores de sus vecindades. Puede ser una expresión algebraica o un grupo de ecuaciones.

1.1. Introducción

1.2. Práctica a realizar

Este programa implementar la simulación de un autómata celular. Los puntos importantes a señalar son que cuenta con una interfaz gráfica para el usuario en la cual aparecen los unos (célula viva) y ceros (célula muerta) que son el principal elemento en este autómata celular y que son representados como pequeños cuadros que cambian su tamaño de acuerdo a la población que se tenga. Las características de este simulador son las siguientes:

- Permitir seleccionar el tamaño de la población de la matriz de unos y ceros.
- Permitir seleccionar la regla que se utilizara en cada iteración de la simulación.
- Se podrá elegir la distribución de unos que habrá en la matriz.
- Se podrá cambiar los colores de los ceros y los unos de la simulación.
- Permitir guardar la matriz que se esta trabajando en un archivo de texto.
- Se mostrara el cambio de unos que hay a lo largo de cada iteración.

El objetivo que se tiene es mostrar una matriz de hasta 1000 por 1000 para poder observar un comportamiento que nos proporcione información.

Para el segundo parcial se realizo una mejora de este programa 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.

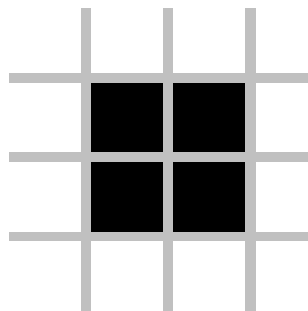


Figura 1: Patrón estático que se genera con la regla 2333

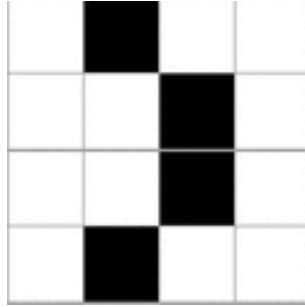


Figura 2: Glider que se genera con la regla 7722

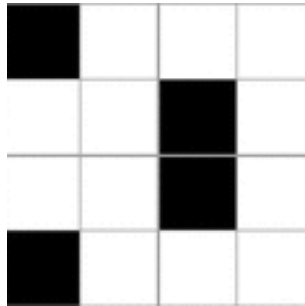


Figura 3: Glider que se genera con la regla 7722

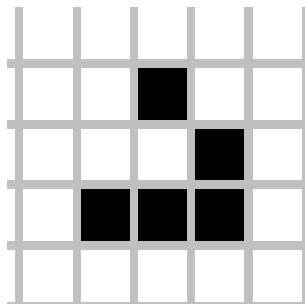


Figura 4: Glider que se genera con la regla 2333

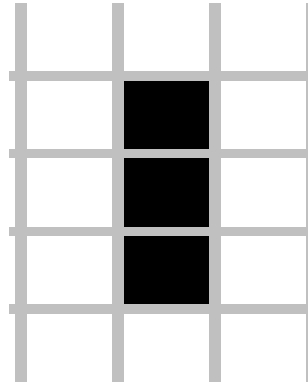


Figura 5: Oscilador que se genera con la regla 2333

1.3. 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.

```

1  from tkinter import Tk, Canvas, Frame, Button, Entry, Label,
    Scale, filedialog
2  from tkinter import BOTH, TOP, LEFT, HORIZONTAL, Y, RIGHT,
    VERTICAL, PhotoImage
3  from tkinter import Scrollbar
4  import numpy as np
5  from tkcolorpicker import askcolor
6  import datetime
7  import time
8
9
10 dict_tipos = {
11     "nada": 0,
12     "cubo": 1,
13     "glider": 2,
14     "glider2": 3,
15     "glider3": 4,
16     "oscilador": 5,
17 }
18
19
20 class Ventana(Frame):
21     def __init__(self, parent):
22         Frame.__init__(self, parent)
23         self.parent = parent
24

```

```

25     # Elementos interfaz
26     self.ceros = "white"
27     self.unos = "black"
28     self.regla = [2, 3, 3, 3]
29     self.e1 = None
30     self.e2 = None
31     self.contador = 0
32     self.colorBtn1 = None
33     self.colorBtn2 = None
34     self.barra = None
35     self.canvas = None
36     self.cubo_image = None
37     self.glider = None
38     self.glider2 = None
39     self.glider3 = None
40     self.oscilador = None
41     # 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.el.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_cuadrilo(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
353                     0 + (i * self.tam_cuadro),
354
355                     self.tam_cuadro + (j * self.tam_cuadro),
356
357                     self.tam_cuadro + (i * self.tam_cuadro),
358
359                     fill=self.ceros, width=0, tag="btncuadruto")
360                 else:

```

```

357         self.cuadritos[i, j] = self.canvas.
create_rectangle(0 + (j * self.tam_cuadro),
358
359         0 + (i * self.tam_cuadro),
360
361         self.tam_cuadro + (j * self.tam_cuadro),
362
363         self.tam_cuadro + (i * self.tam_cuadro),
364
365         fill=self.unos, width=0, tag="btncuadruto")
366
367     self.canvas.tag_bind("btncuadruto", "<Button-1>", self.
pulsar_cuadruto)
368     self.update_idletasks()
369
370     def init_ui(self):
371         self.parent.title("Juego de la vida")
372         self.pack(fill=BOTH, expand=1)
373
374         self.canvas = Canvas(self, relief='raised', width=1000,
height=1000)
375         scroll = Scrollbar(self, orient=VERTICAL)
376         scroll.pack(side=RIGHT, fill=Y)
377         scroll.config(command=self.canvas.yview)
378
379         self.canvas.config(yscrollcommand=scroll.set)
380         self.canvas.pack(side=LEFT)
381
382         Label(self, text="Regla:").pack(side=TOP)
383         self.e1 = Entry(self, fg="black", bg="white")
384         self.e1.insert(10, "2,3,3,3")
385         self.e1.pack(side=TOP)
386
387         Label(self, text="Tamaño:").pack(side=TOP)
388         self.e2 = Entry(self, fg="black", bg="white")
389         self.e2.insert(10, "100")
390         self.e2.pack(side=TOP)
391
392         Label(self, text="Porcentaje de unos").pack(side=TOP)
393         self.barra = Scale(self, from_=0, to=100, orient=
HORIZONTAL, tickinterval=50)
394         self.barra.set(50)
395         self.barra.pack(side=TOP)
396
397         btn_iniciar = Button(self, text="Iniciar/Reiniciar",
command=self.iniciar)
398         btn_iniciar.pack(side=TOP)

```

```

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
453     archivo",
454                                     filetypes=((("CSV", "*.
455     csv"), ("Archivo de texto", "*.txt")), ("Todos los
456     archivos", ".*.*")))
457     return ga
458
459 def actualizar_color_matriz(self):
460     for i in range(self.tam):
461         for j in range(self.tam):
462             if self.celulas[i][j] == 0:
463                 self.canvas.itemconfig(self.cuadritos[i][j],
464                 fill=self.ceros)
465             else:
466                 self.canvas.itemconfig(self.cuadritos[i][j],
467                 fill=self.unos)
468
469     self.update_idletasks()
470
471 def get_color_unos(self):
472     color = askcolor()
473     if not color[1] is None:
474         self.unos = color[1]
475         self.colorBtn1.configure(bg=self.unos)
476         self.actualizar_color_matriz()
477
478 def get_color_ceros(self):
479     color = askcolor()
480     if not color[1] is None:
481         self.ceros = color[1]
482         self.colorBtn2.configure(bg=self.ceros)
483         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.el.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_detener(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()
```

Archivo: grafica.py En este archivo se encuentra el código que se encarga de graficar la historia de unos a lo largo de la animación, al igual que el archivo anterior se utilizó python 3, sin embargo en la graficación se realizó con la biblioteca matplotlib.

```
1 import matplotlib.pyplot as plt
2 import matplotlib.animation as animation
3
4 fig = plt.figure('Historial de unos')
5 fig.suptitle("Historial de unos")
6 ax1 = fig.add_subplot(1, 1, 1)
7 def animacion(i):
8     info = open("grafica.txt", "r").read()
9     lineas = info.split("\n")
10    xs = []
11    ys = []
12
13    for linea in lineas:
14        if len(linea) > 1:
15            x,y = linea.split(",")
16            xs.append(int(x))
17            ys.append(int(y))
18    ax1.clear()
19    ax1.plot(xs, ys)
20
21 ani = animation.FuncAnimation(fig, animacion, interval=1000)
22
23 plt.show()
```

1.4. Pruebas

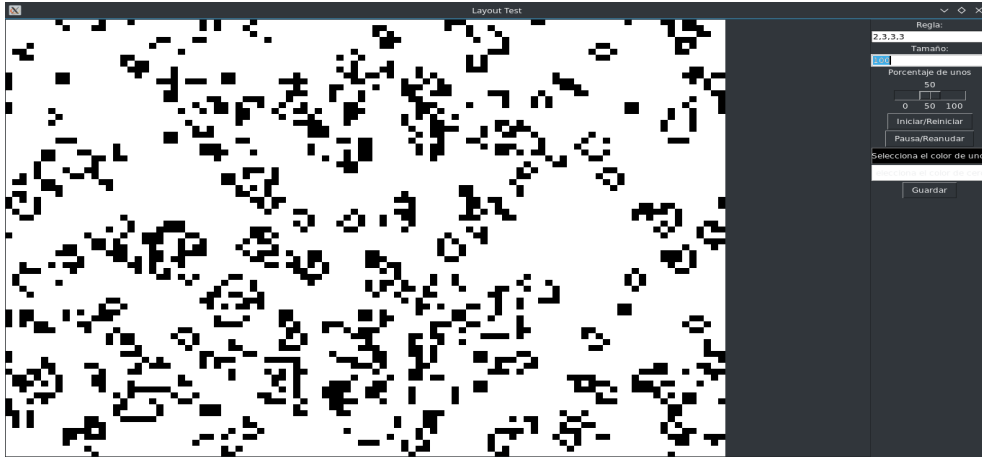


Figura 6: Juego de la vida con la regla 2333

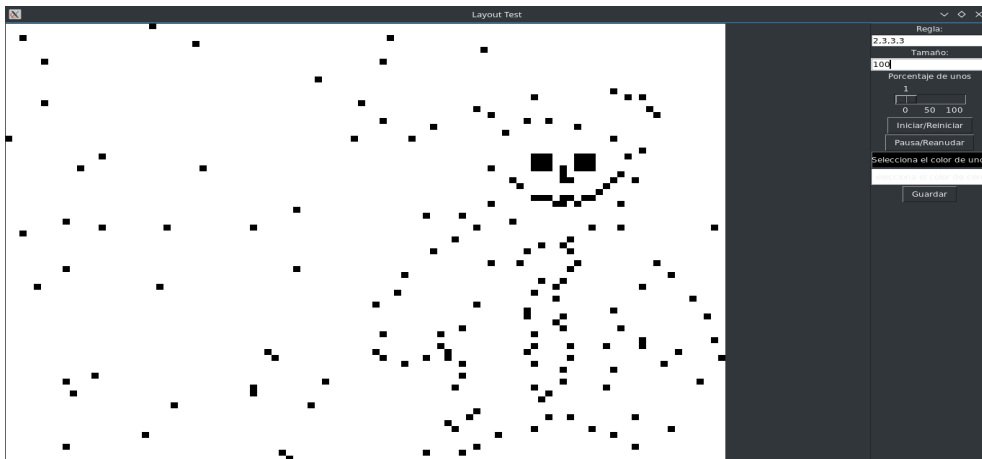


Figura 7: Juego de la vida con células seleccionadas por el usuario

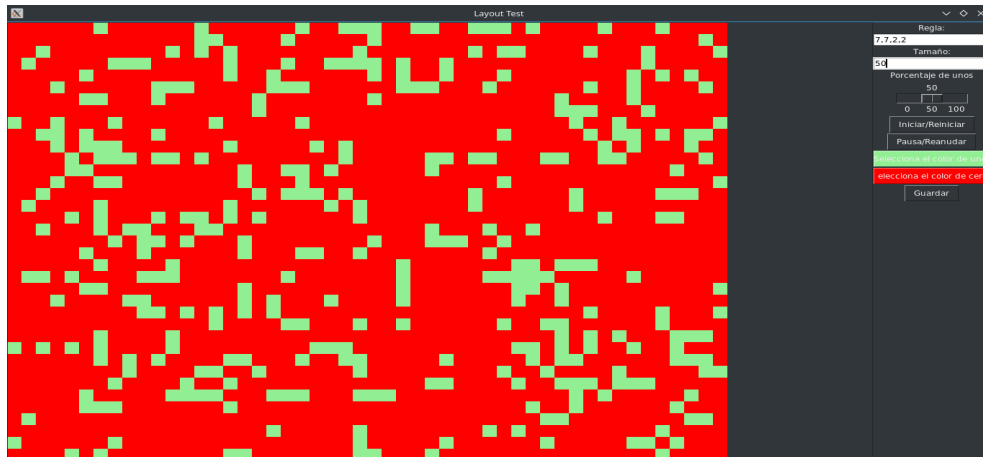
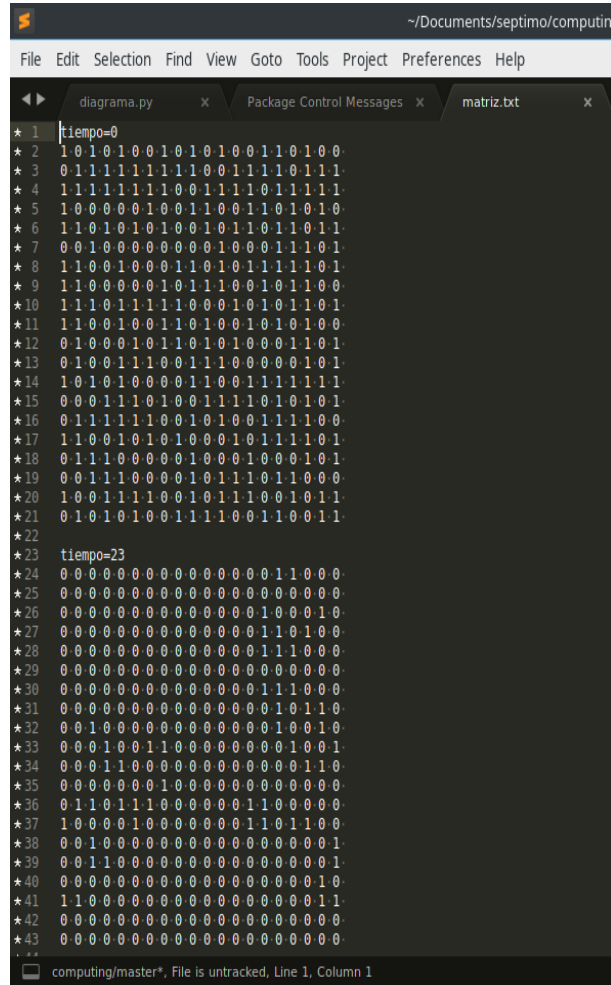


Figura 8: Juego de la vida cambiando los colores y la regla 7722



The image shows a code editor window with the title bar "~Documents/septimo/computing". The menu bar includes File, Edit, Selection, Find, View, Goto, Tools, Project, Preferences, and Help. The editor has three tabs: "diagrama.py", "Package Control Messages", and "matriz.txt". The "matriz.txt" tab is active, displaying a binary matrix. The matrix is divided into two sections by a line: "tiempo=0" and "tiempo=23". Each section contains 20 rows of binary data, with each row preceded by a line number from 1 to 43. The status bar at the bottom indicates "computing/master*, File is untracked, Line 1, Column 1".

```
* 1 tiempo=0
* 2 101010001010100110100
* 3 011111111110011110111
* 4 11111111001111011111
* 5 10000010011001101010
* 6 11010101001011011011
* 7 00100000000100011101
* 8 110010001101010111101
* 9 11000001011100101100
*10 11101111100010101101
*11 11001001101001010100
*12 01000101101010001101
*13 01001110011100000101
*14 10101000011001111111
*15 00011101001111010101
*16 01111110010100111100
*17 11001010100010111101
*18 01110000010001000101
*19 00111000010111011000
*20 10011110010111001011
*21 01010100111100110011
*22
*23 tiempo=23
*24 000000000000000011000
*25 00000000000000000000
*26 000000000000000100010
*27 000000000000000110100
*28 000000000000000111000
*29 00000000000000000000
*30 000000000000000111000
*31 00000000000000010110
*32 00100000000000010010
*33 00010011100000001001
*34 00011000000000000110
*35 00000001000000000000
*36 01101110000001100000
*37 10000100000001101100
*38 00100000000000000001
*39 00110000000000000001
*40 00000000000000000010
*41 11000000000000000011
*42 00000000000000000000
*43 00000000000000000000
```

Figura 9: Matriz que se guarda

1.4.1. Análisis de poblaciones

En esta parte se hicieron pruebas con la regla de life y difusión aumentando la densidad de la población de 10 en 10 por ciento hasta llegar al máximo de 90 por ciento debido que en un 100 por ciento no se aprecia nada al igual que en cero, las pruebas de realizaron tras 500 o 1000 generaciones en una matriz de 100 por 100.

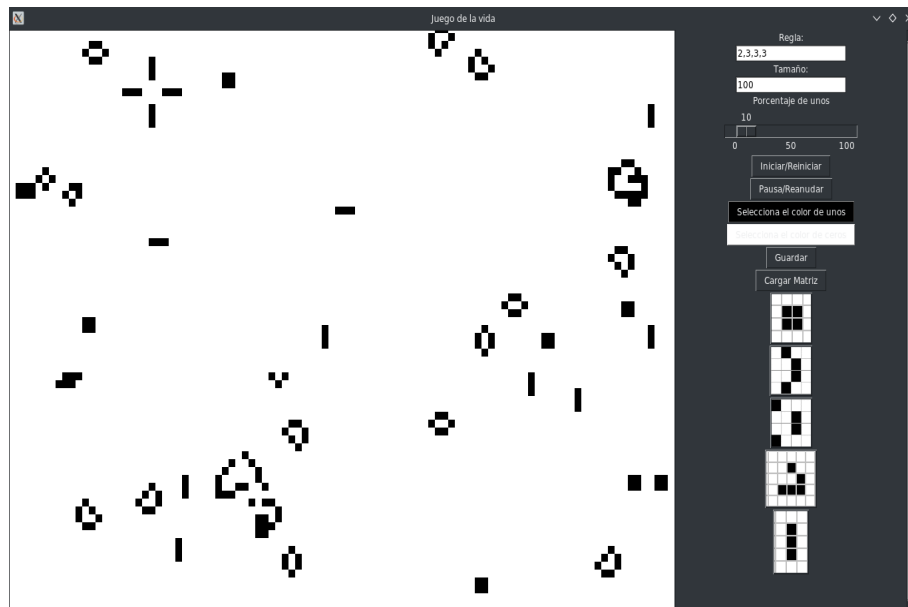


Figura 10: Regla de life con una probabilidad de unos de 10 %

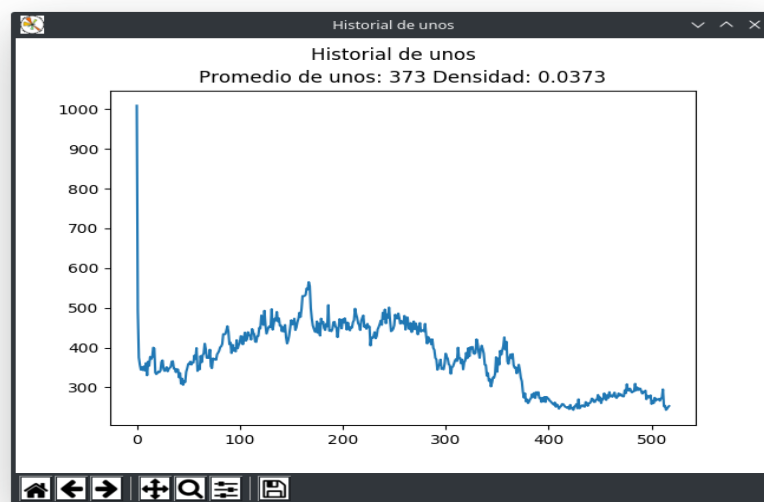


Figura 11: Comportamiento de la población de la simulación anterior

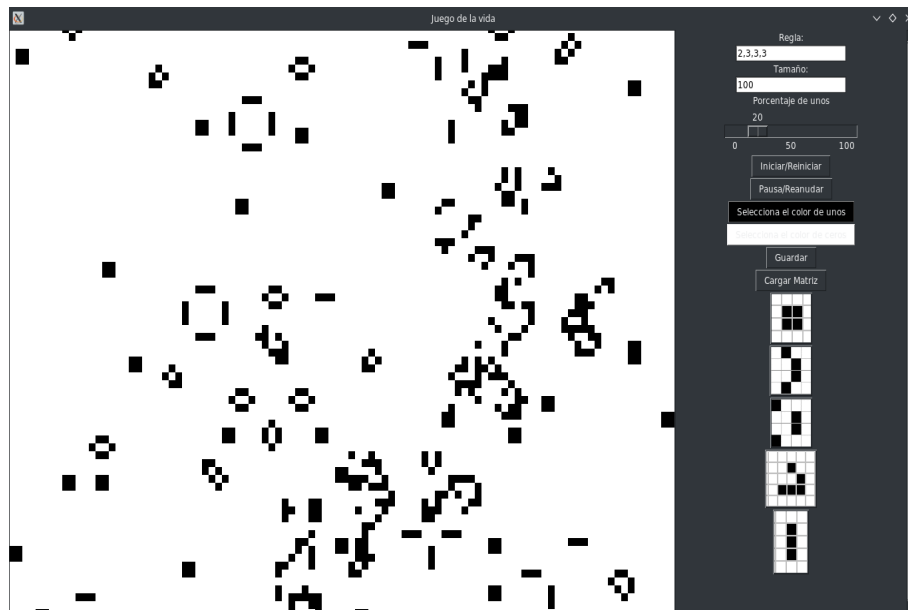


Figura 12: Regla de life con una probabilidad de unos de 20 %

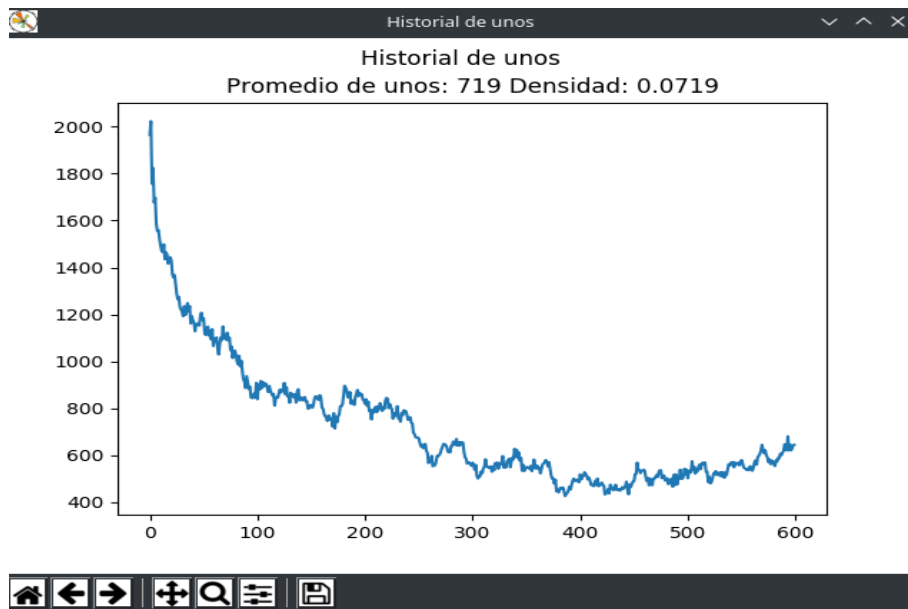


Figura 13: Comportamiento de la población de la simulación anterior

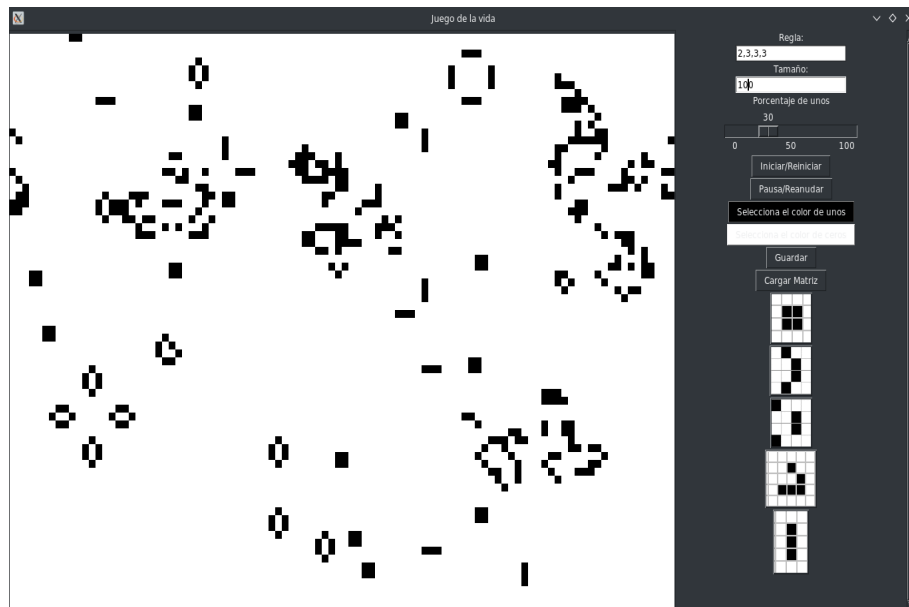


Figura 14: Regla de life con una probabilidad de unos de 30 %

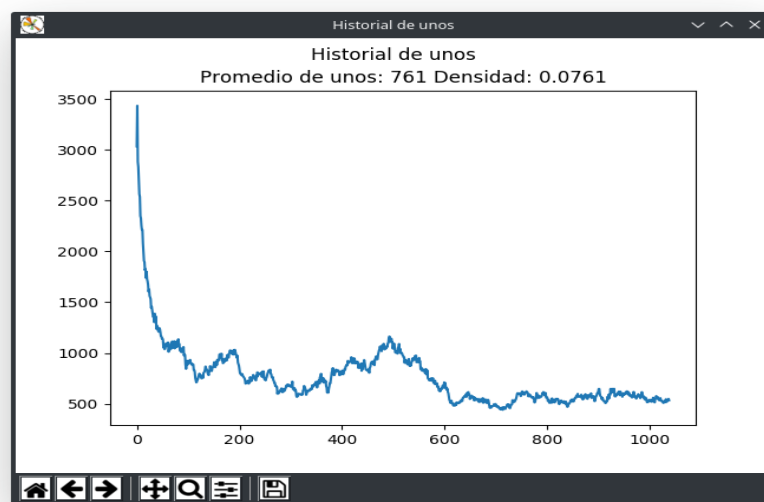


Figura 15: Comportamiento de la población de la simulación anterior

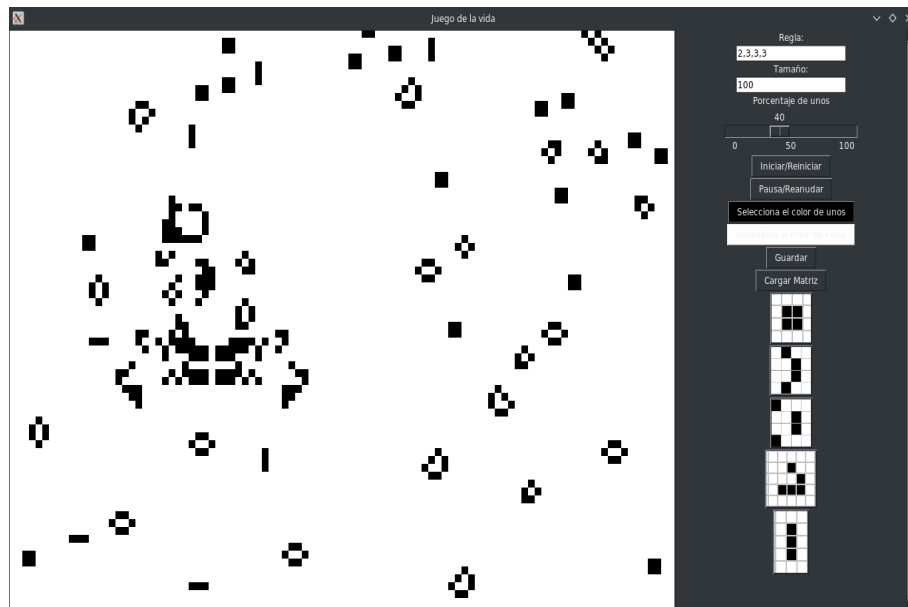


Figura 16: Regla de life con una probabilidad de unos de 40 %

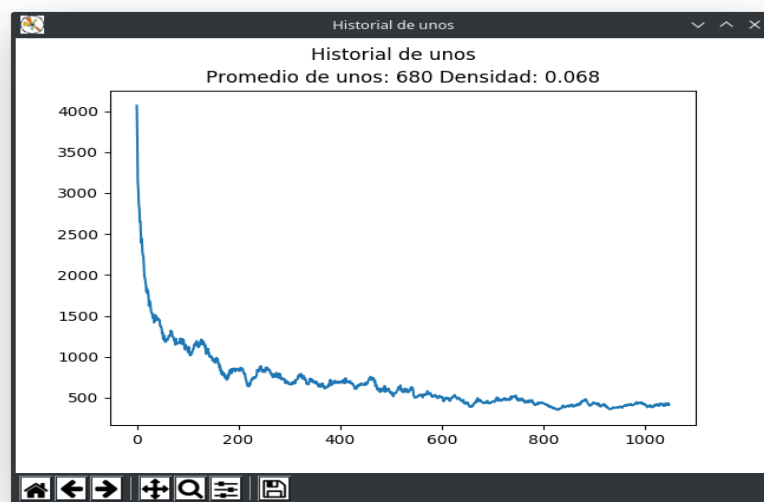


Figura 17: Comportamiento de la población de la simulación anterior

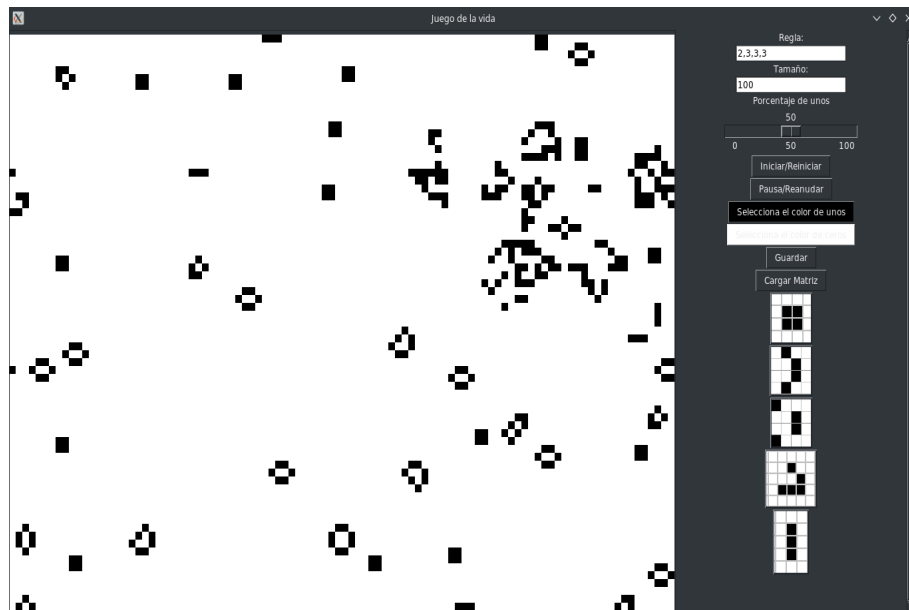


Figura 18: Regla de life con una probabilidad de unos de 50 %

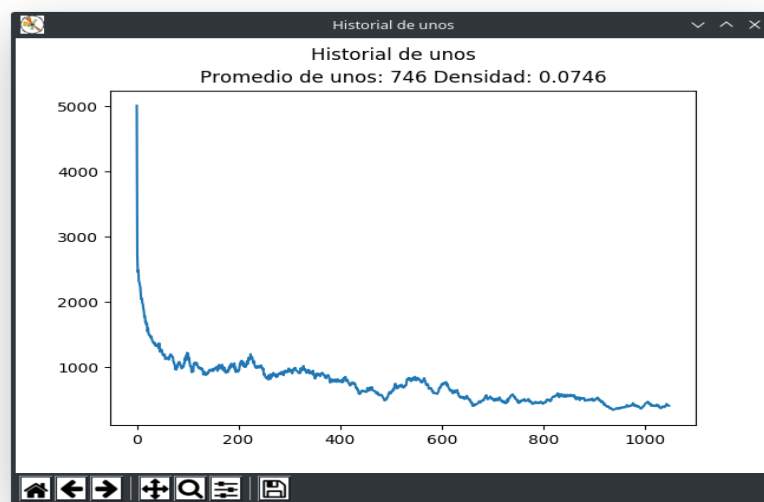


Figura 19: Comportamiento de la población de la simulación anterior

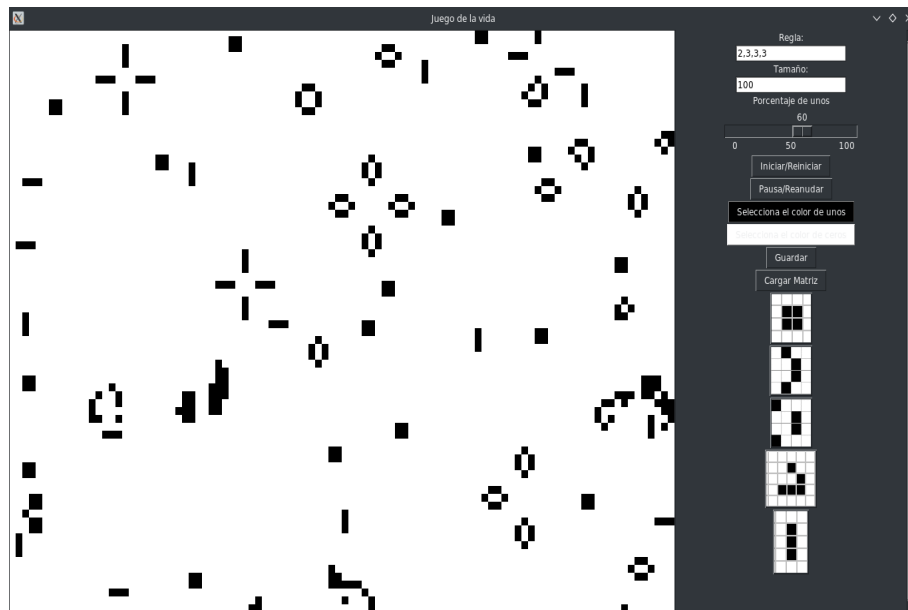


Figura 20: Regla de life con una probabilidad de unos de 60 %

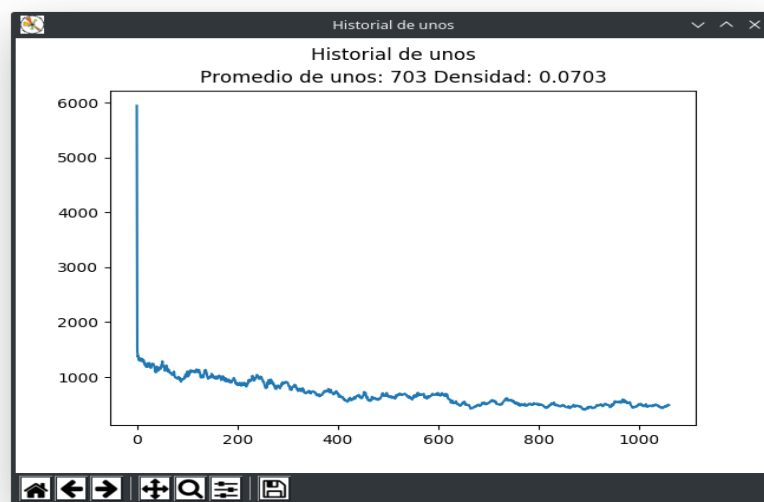


Figura 21: Comportamiento de la población de la simulación anterior

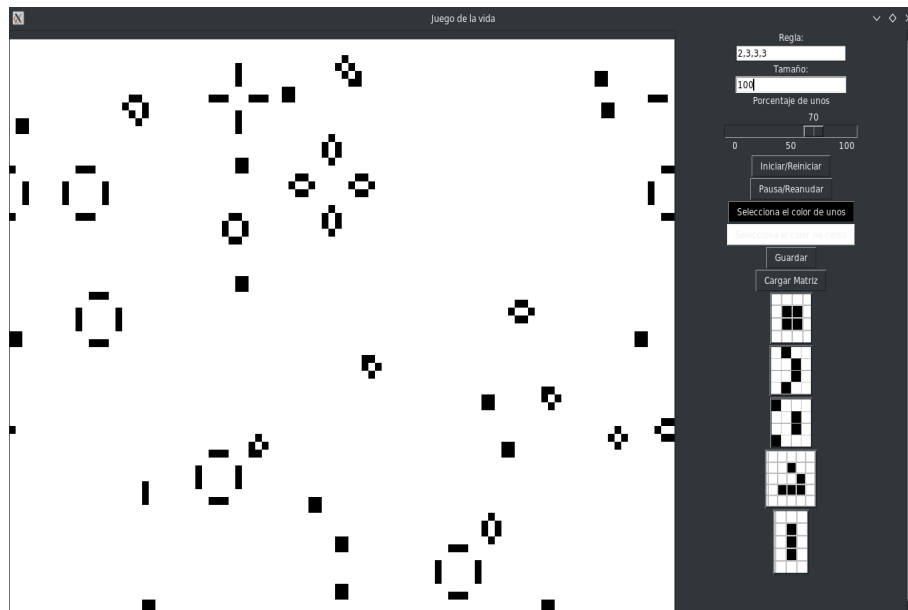


Figura 22: Regla de life con una probabilidad de unos de 70 %

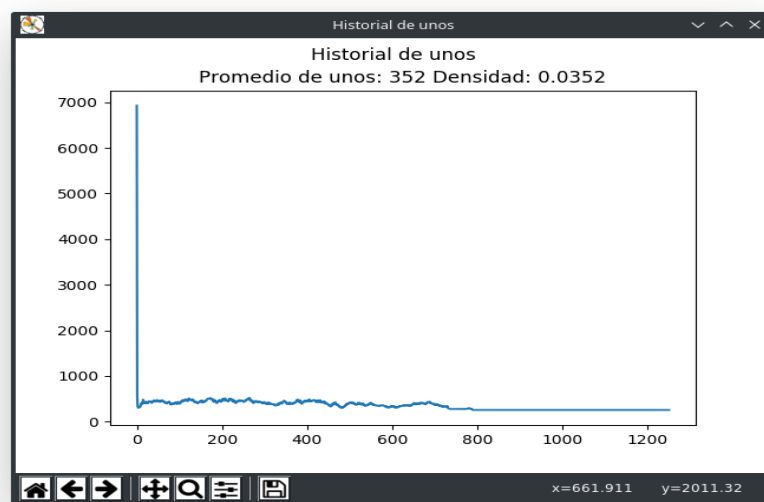


Figura 23: Comportamiento de la población de la simulación anterior

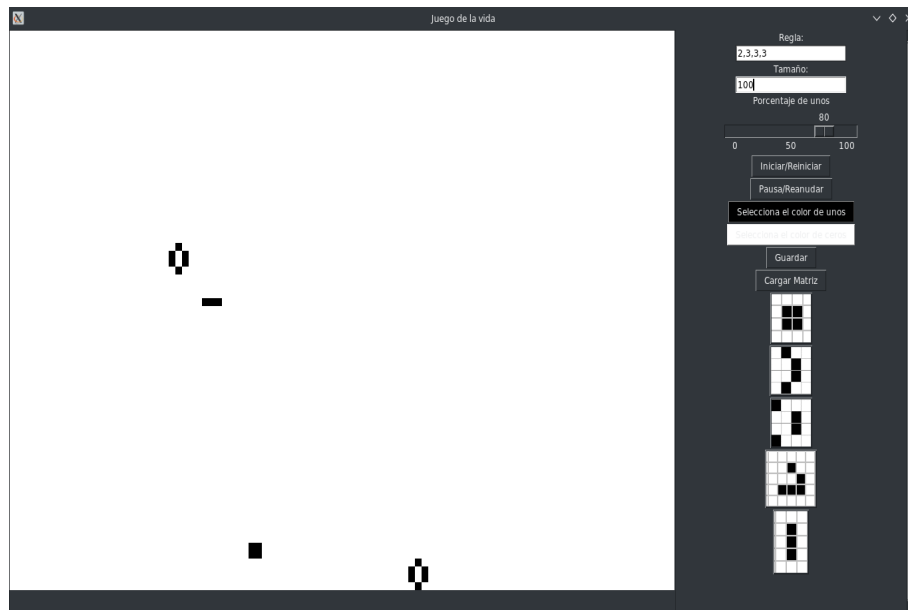


Figura 24: Regla de life con una probabilidad de unos de 80 %

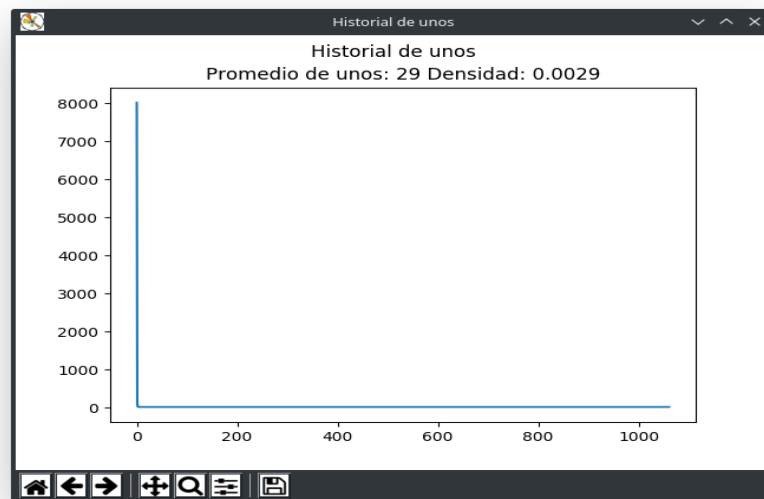


Figura 25: Comportamiento de la población de la simulación anterior



Figura 26: Regla de life con una probabilidad de unos de 90 %

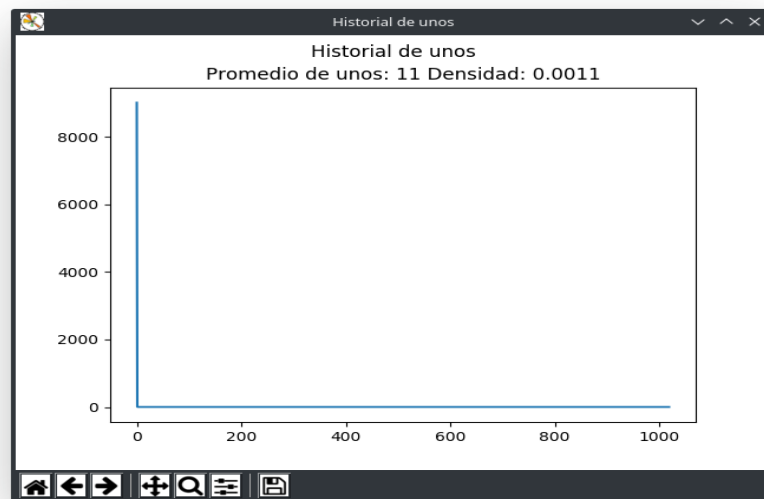


Figura 27: Comportamiento de la población de la simulación anterior

En todas las simulaciones se encuentra un comportamiento similar en el cual la población inicial es muy alta y de manera brusca disminuye y a partir de ahí decrece de manera lenta.

En la figura 22 se aprecia así que el valor de densidad de la población tiende a la regla de life, el valor es 0.0352, el resto de simulaciones tiende a un valor similar pero debido a que solo se trabajaron con 1000 generaciones no se logra apreciar por completo.



Figura 28: Regla de difusión con una probabilidad de unos de 10 %

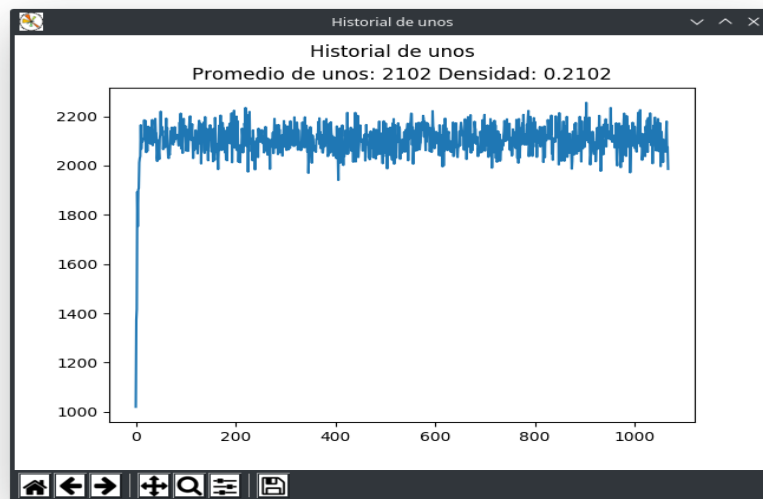


Figura 29: Comportamiento de la población de la simulación anterior



Figura 30: Regla de difusión con una probabilidad de unos de 20 %

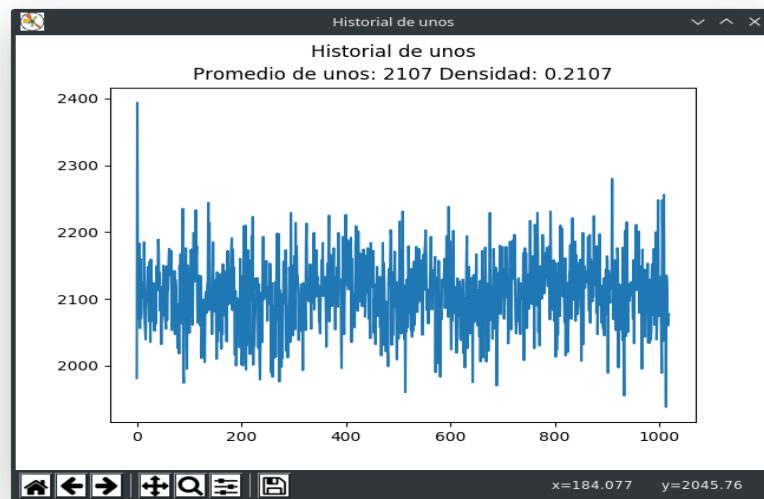


Figura 31: Comportamiento de la población de la simulación anterior



Figura 32: Regla de difusión con una probabilidad de unos de 30 %

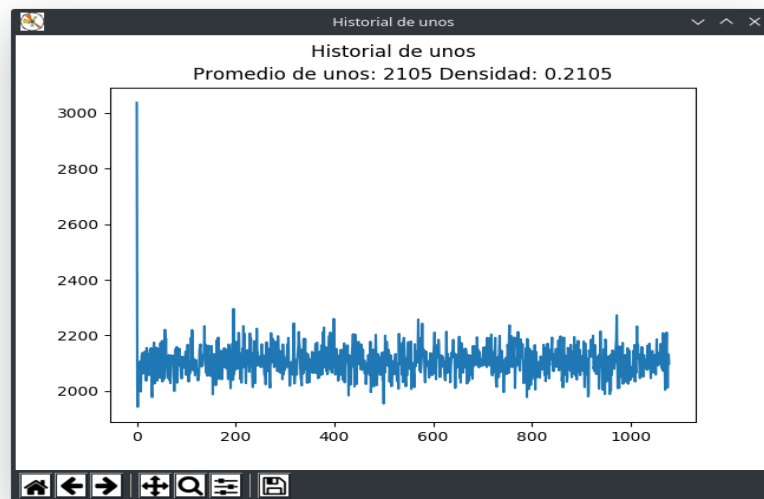


Figura 33: Comportamiento de la población de la simulación anterior



Figura 34: Regla de difusión con una probabilidad de unos de 40 %

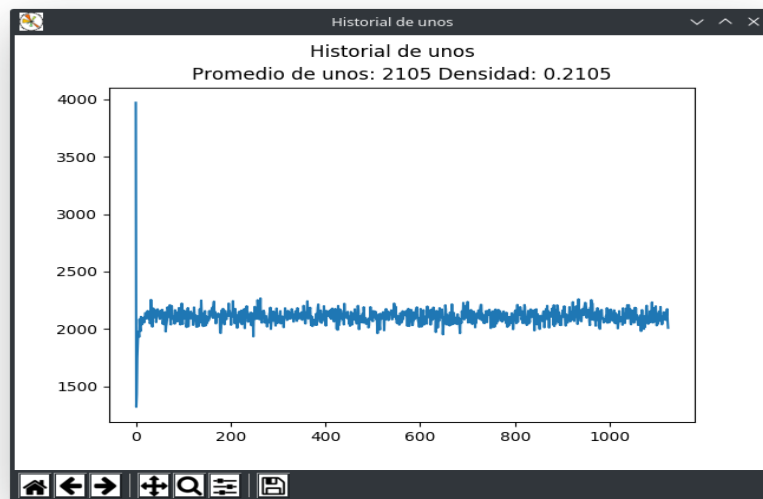


Figura 35: Comportamiento de la población de la simulación anterior

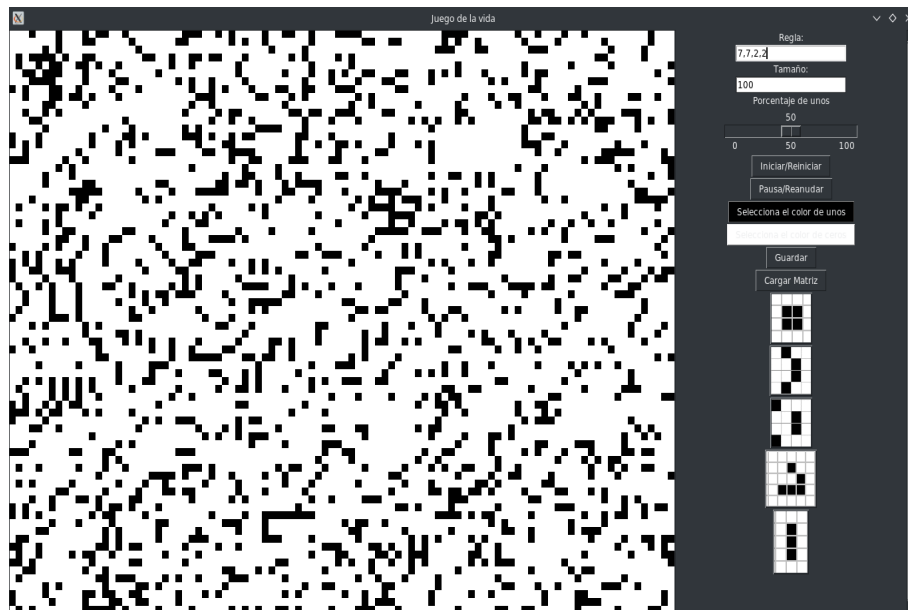


Figura 36: Regla de difusión con una probabilidad de unos de 50 %

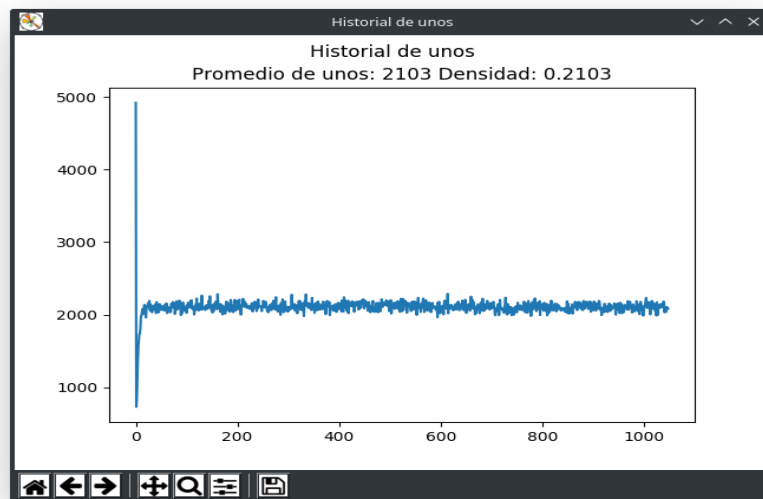


Figura 37: Comportamiento de la población de la simulación anterior



Figura 38: Regla de difusión con una probabilidad de unos de 60 %

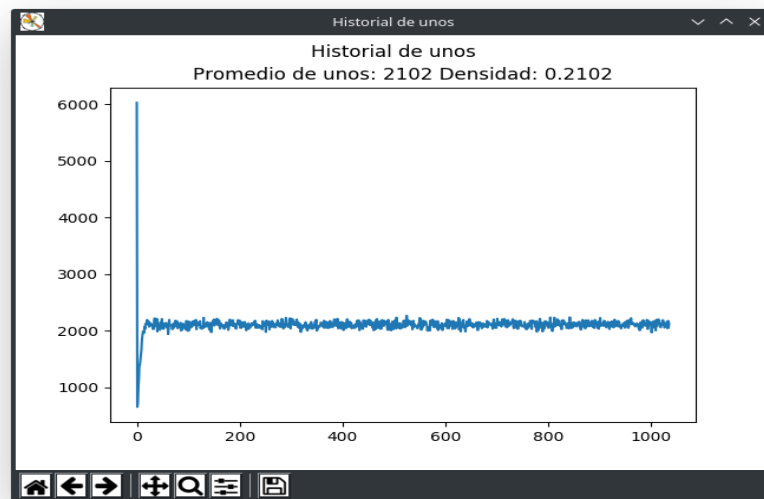


Figura 39: Comportamiento de la población de la simulación anterior



Figura 40: Regla de difusión con una probabilidad de unos de 70 %

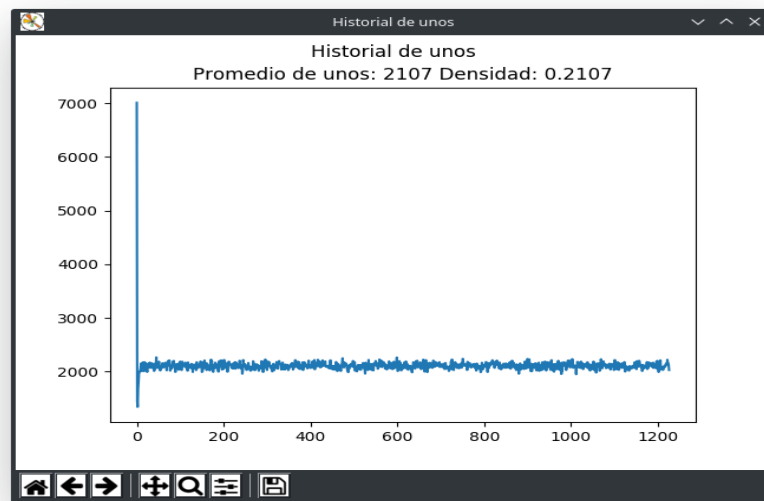


Figura 41: Comportamiento de la población de la simulación anterior



Figura 42: Regla de difusión con una probabilidad de unos de 80 %

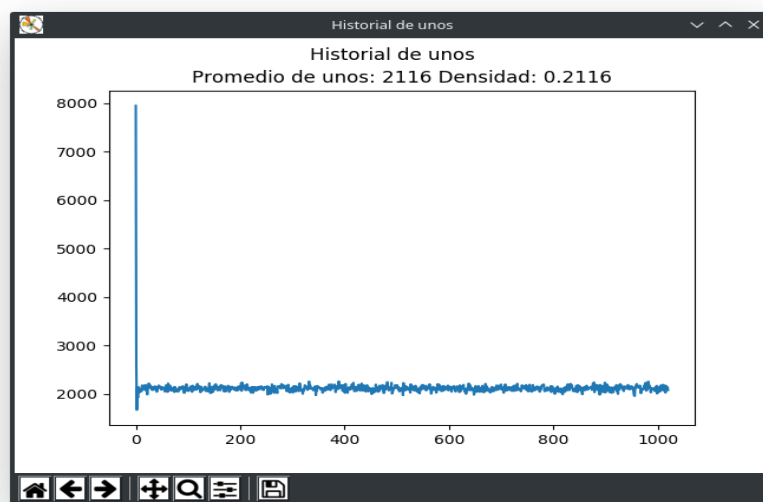


Figura 43: Comportamiento de la población de la simulación anterior



Figura 44: Regla de difusión con una probabilidad de unos de 90 %

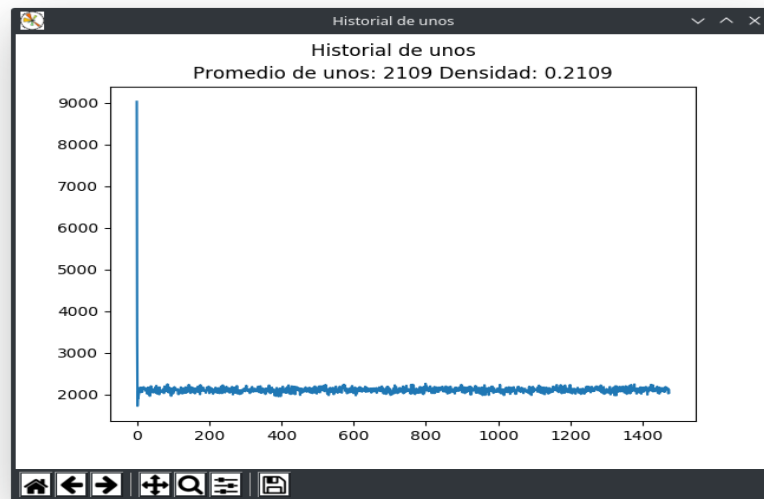


Figura 45: Comportamiento de la población de la simulación anterior

Para la regla de difusión se podría decir que el comportamiento de la po-

blación es más estable que la de life debido a que en todas las simulaciones se llega a la densidad de población de 0.21 y el comportamiento de la gráfica es igual en todas las probabilidades de uno que se probaron en donde la población oscila entre un límite mayor y uno menor cerca de los 2000 individuos vivos lo cual es un comportamiento interesante.

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

1.4.2. Insertar mosaicos

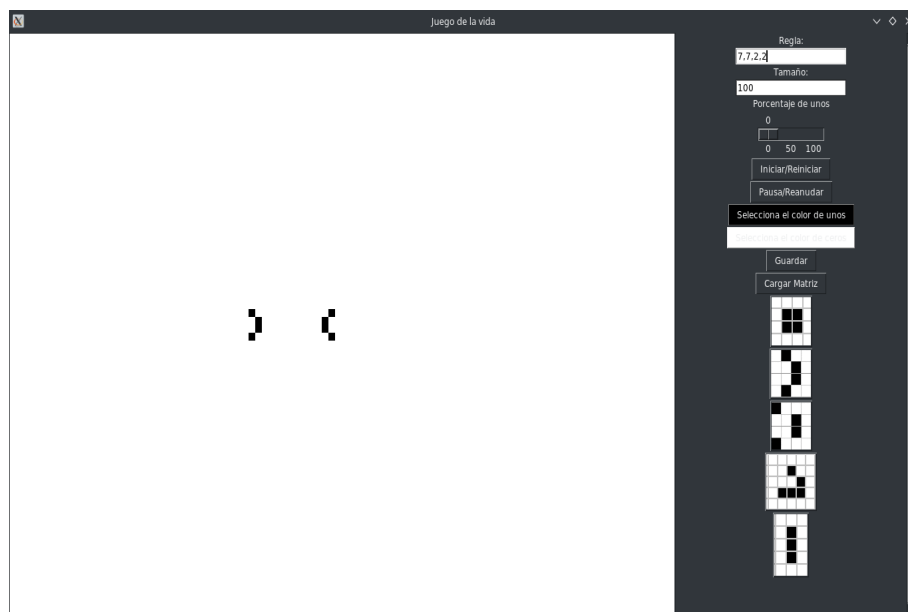


Figura 46: Probando la regla 7722 con dos glider de frente

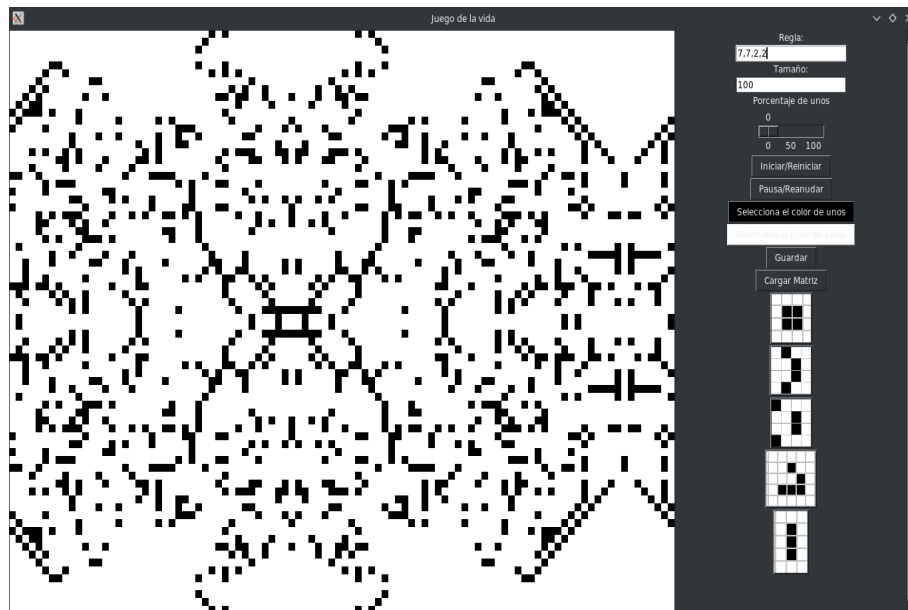


Figura 47: Resultado producido por la configuración anterior

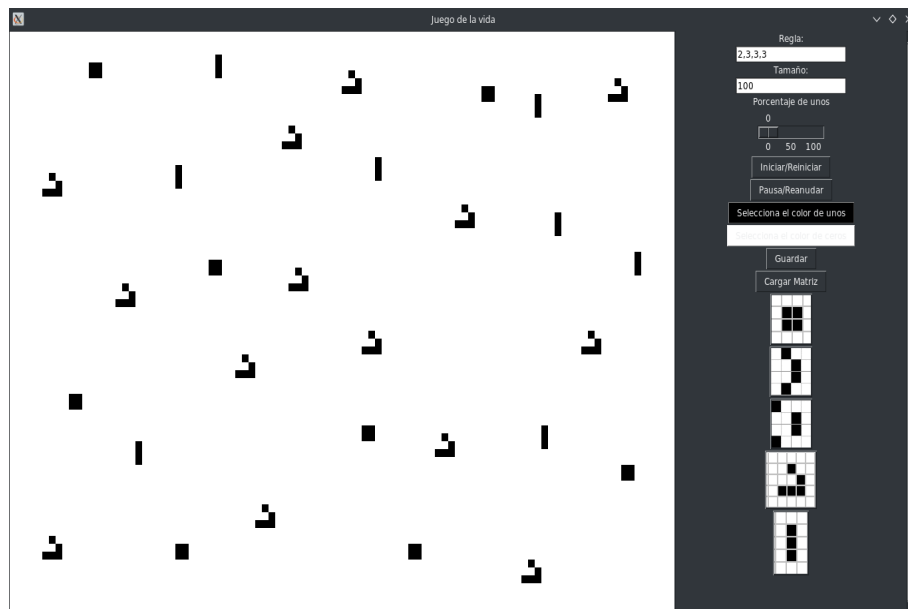


Figura 48: Probando gliders, osciladores y patrones estáticos con la regla 2333

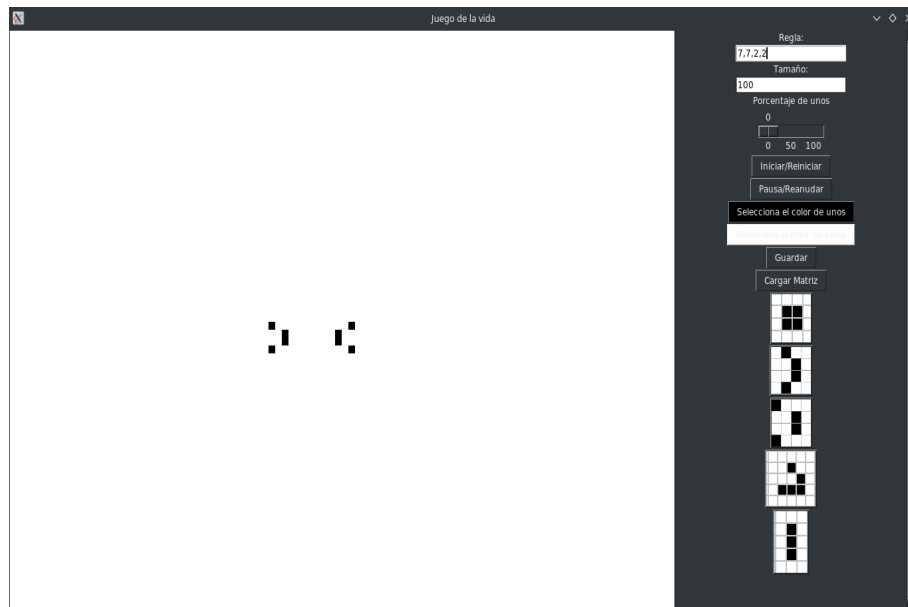


Figura 49: Probando la regla 7722 con dos glider de frente

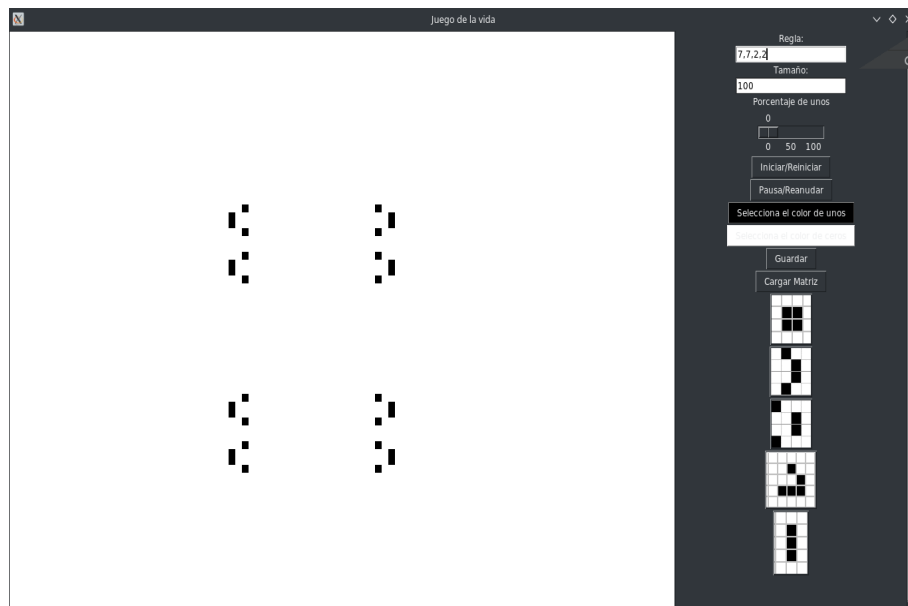


Figura 50: Resultado producido por la configuración anterior

1.5. Conclusiones

Al hacer y probar este programa se pudieron solucionar dos cuestiones, la primera es si existe una configuración en la cual el número de células no se termine, y la respuesta a esto es que exista un oscilador el cual cambia su posición a lo largo del tiempo, otra posible opción es que haya figuras como un bloque formado por 4 cuadros en el cual no hay cambios.

La siguiente cuestión es si existe una configuración en la cual la población crezca indefinidamente. Para lograr esto es indispensable tener un espacio que sea infinito en el cual se puedan propagar las células a través del tiempo, ya que si no se cuenta con esto en algún punto se tendrán tantos elementos vivos que empezaran a morir por sobre población.

Respecto a los patrones, 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 generaciones 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 específicamente 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.

```
1 import numpy as np
2 import sys
3 import webbrowser
4
5 dict_tipos = {
6     "life": 1,
7     "diffusion": 2,
8 }
9
10
11 class Arboles:
12     def __init__(self, _tam=2, _tipo=dict_tipos["life"]):
13         self.tam = tam
14         self.tipo = tipo
15         self.vida = [2, 3, 3, 3]
16         self.diffusion = [7, 7, 2, 2]
17         self.regla = self.diffusion
18         self.longitud = self.tam * self.tam
19         self.max = 2 ** self.longitud
20         self.formato = "{:0{}b}".format(self.longitud)
21         if self.tipo == dict_tipos["life"]:
22             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:

```

```

71         cadena[k] = "1"
72         k += 1
73     return "".join(cadena)
74
75     def generar(self):
76         temp_nom = "diffusion"
77         if self.tipo == dict_tipos["life"]:
78             temp_nom = "life"
79         nombre_archivo = "tam-{}-{}".format(self.tam, temp_nom)
80         archivo = open("{}_wls".format(nombre_archivo), "w")
81
82         archivo.write("Graph[{"")
83         for i in range(self.max):
84             cadena = self.numero_cadena(i)
85             m = self.cadena_matriz(cadena)
86             m_sig = self.obtener_siguiete(m)
87             cadena_sig = self.matriz_cadena(m_sig)
88             if i == self.max-1:
89                 archivo.write("{}{} -> {}{} ".format(cadena,
90 cadena_sig))
91             else:
92                 archivo.write("{}{} -> {}{} , ".format(cadena,
93 cadena_sig))
94             if self.tam == 2:
95                 archivo.write('}', VertexLabels->Automatic,
96 GraphLayout -> "RadialEmbedding"]')
97             else:
98                 archivo.write('}', GraphLayout -> "RadialEmbedding"]'
99 )
100         archivo.close()
101         ejecutable = open("ejecutable-{}-{}.bat".format(self.tam,
102 temp_nom), "w")
103         ejecutable.write("set MATHEPATH=C:\\Program Files\\
104 Wolfram Research\\Mathematica\\11.3\\n")
105         ejecutable.write("set PROJECTPATH=C:\\Users\\reymy\\
106 Documents\\septimo\\computing-selected-topics\\arboles\\n")
107         ejecutable.write(""%MATHEPATH%\\wolframscript.exe" -
108 cloud -print -format PNG -file ')
109         ejecutable.write(""%PROJECTPATH%\\{}.wls" > {}.png'.
110 format(nombre_archivo, nombre_archivo))
111         ejecutable.close()
112
113 tam = int(sys.argv[1])
114 tipo = int(sys.argv[2])
115
116 life2 = Arboles(tam, tipo)
117 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 MATHEPATH=C:\Program Files\Wolfram Research\Mathematica
    \11.3
2 set PROJECTPATH=C:\Users\reymy\Documents\septimo\computing-
    selected-topics\arboles
3 "%MATHEPATH%\wolframscript.exe" -cloud -print -format PNG -file
    "%PROJECTPATH%\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.

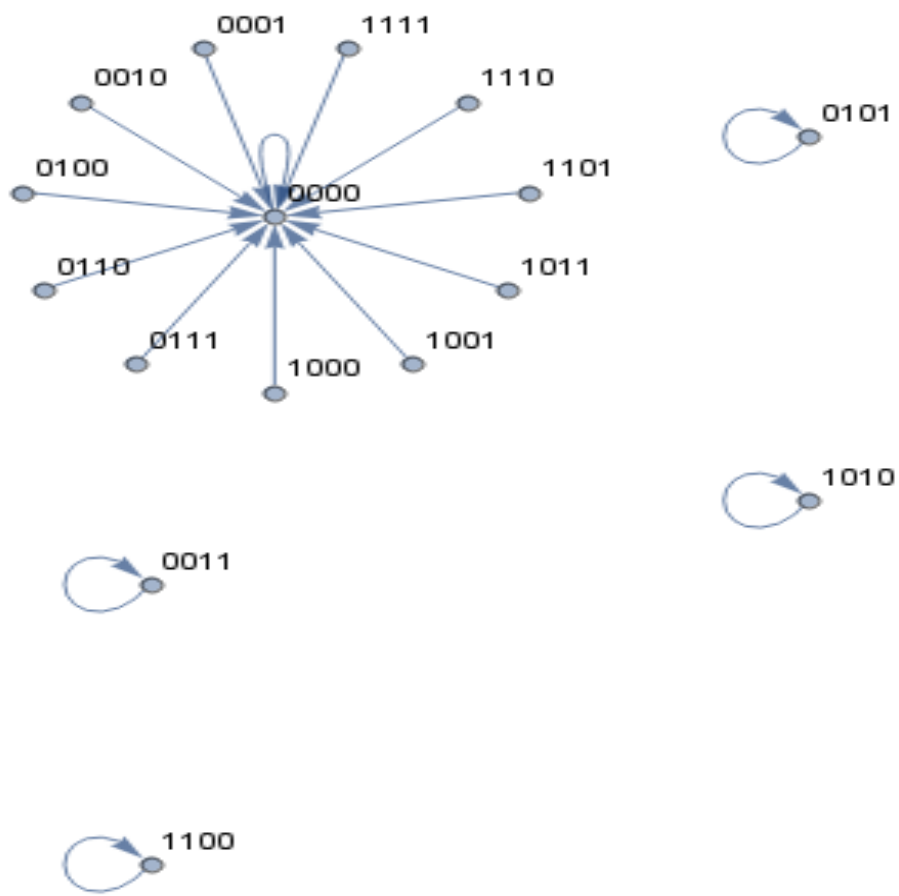


Figura 51: Árboles generados en una matriz de 2x2 con la regla de life

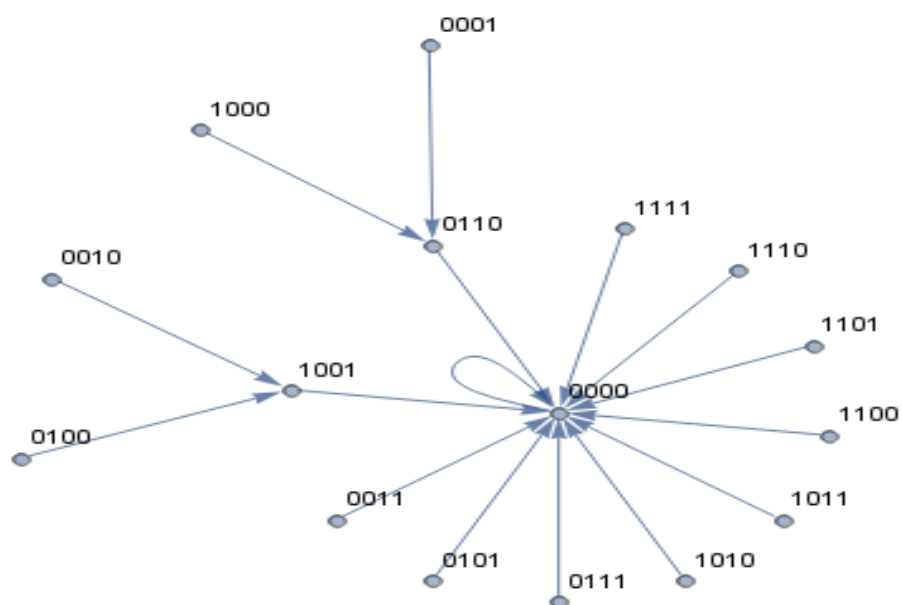


Figura 52: Árboles generados en una matriz de 2x2 con la regla de difusión

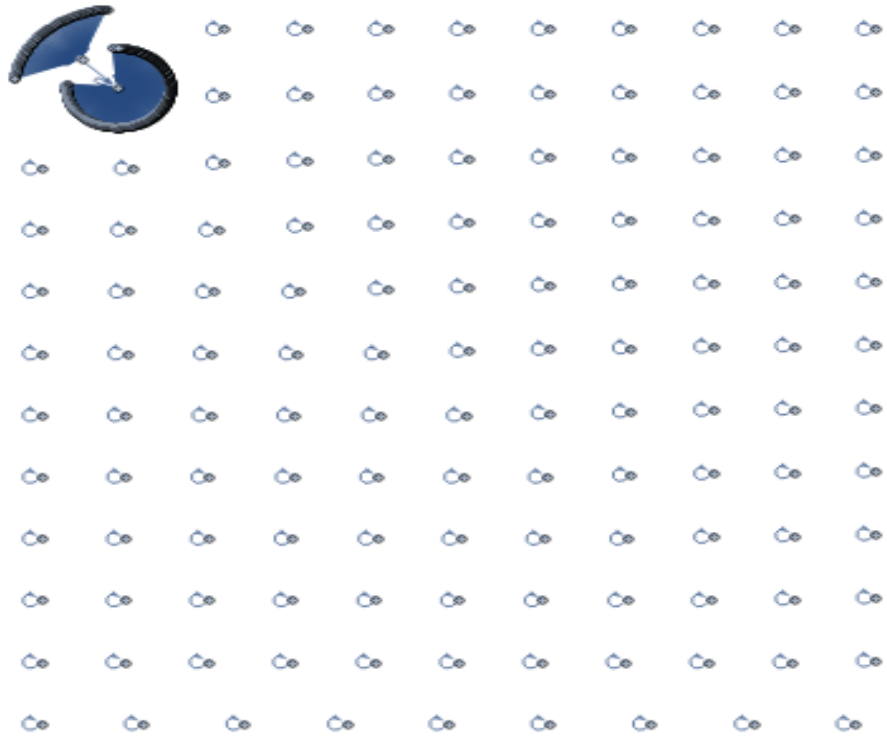


Figura 53: Árboles generados en una matriz de 3x3 con la regla de life

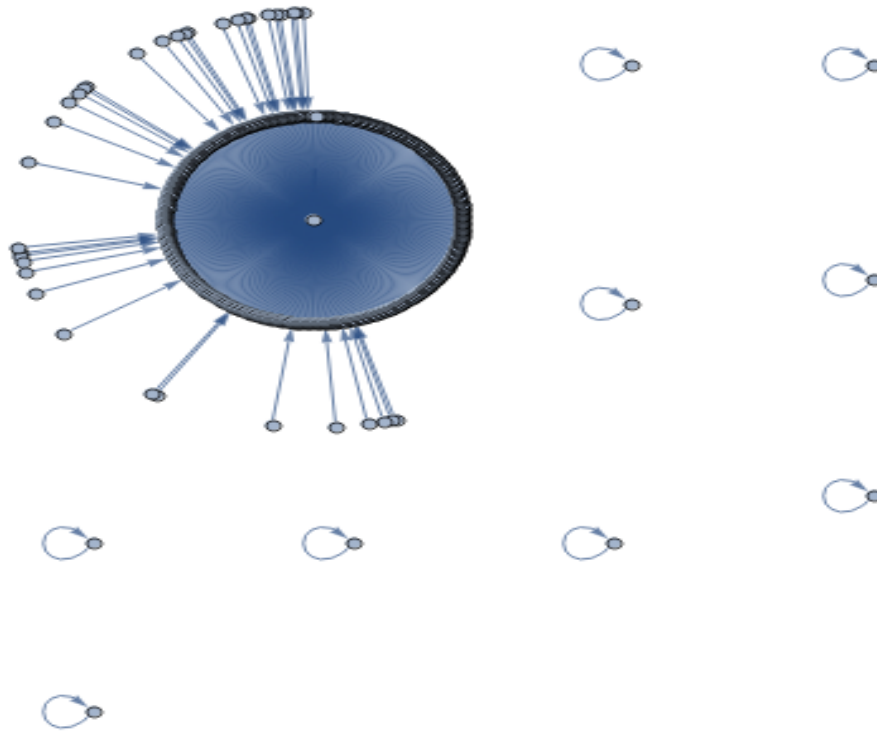


Figura 54: Árboles generados en una matriz de 3x3 con la regla de difusión

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 específica 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.

3. Hormiga de Langton

3.1. Introducción

La hormiga de Langton es un autómata celular desarrollado por Chris Langton en 1986, Langton se inspiró en la bioquímica, exploró la posibilidad de implementar la lógica molecular del estado viviente generando así una bioquímica artificial basada en la interacción entre moléculas artificiales. Langton dijo que el comportamiento global de una sociedad es un fenómeno emergente que surge de todas las interacciones locales de sus miembros.

El comportamiento complejo puede surgir de la interacción de las partes muy simples. Por lo que utilizó una colonia de hormigas como modelo para una forma variante de un autómata celular, en donde cada célula puede cambiar de estado, en virtud de los estados de las otras células. [2]

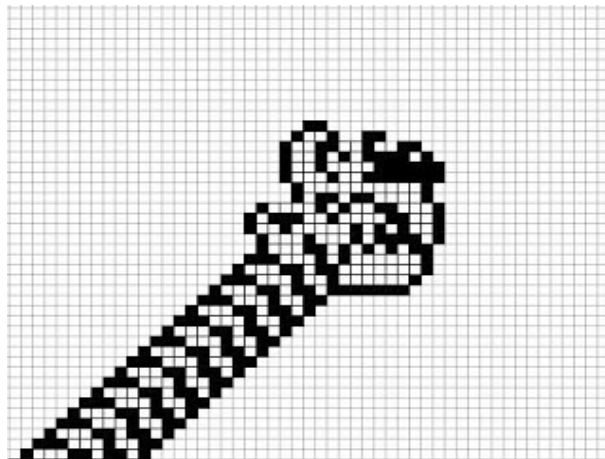


Figura 55: Ejemplo de una hormiga de Langton

Las hormigas del modelo clásico se mueven en un entorno que consta de células en donde cada una de estas células se encuentra en uno de los dos posibles estados (viva o muerta, 0 o 1), la hormiga viaja en línea recta en el espacio siguiendo las siguientes reglas:

Si se encuentra con una célula muerta, hace un giro a la derecha y sale de la célula invirtiendo el estado de la misma.

Si se encuentra con una célula viva, gira a la izquierda y sale de la célula invirtiendo su estado.

De esta forma la hormiga deja un rastro.^a medida que se mueve.

3.2. Práctica a realizar

En esta práctica se trabajaron dos versiones diferentes de la hormiga de Langton, la primera que es la implementación clásica de este autómata celular y la segunda versión en la cual se cuenta con tres tipos de hormigas y de restricciones que modifican el comportamiento del autómata original. Es importante señalar que ambas versiones tienen funcionalidades diferentes como el poder graficar y manipular de forma particular la simulación.

3.2.1. Hormiga de Langton original

Las principales características de esta versión es la posibilidad de insertar hormigas por parte del usuario en la posición que se desee además de poder cambiar el color del camino generado por la hormiga. Para poder apreciar de una forma más clara el comportamiento de la hormiga los movimientos que se realizan tienen un color realizado.

- Si la hormiga esta viendo hacia abajo el color de la hormiga es azul.
- El color sera rojo si la hormiga se encuentra viendo hacia arriba.
- Para la izquierda se tiene el color verde.
- Y para la derecha el color asociado sera el amarillo.

Finalmente, se tiene la posibilidad de generar una cantidad de hormigas basada en una probabilidad de nacimiento, por lo que cada hormiga que se genera tendrá un color diferente. Es importante señalar que el tamaño del espacio en donde se trabaja puede cambiar y las dimensiones de cada célula se ajustan al tamaño del espacio.

3.2.2. Hormiga de Langton modificada

Esta versión modificada tiene todo la funcionalidad de la versión con excepción de que cada hormiga tenga un color diferente, en este caso los colores generados por las hormigas depende del tipo de hormiga del que se trate. Para las hormigas normales el color asociado es el blanco, para las soldado sera el naranja y para las reinas sera el morado.

El comportamiento es el mismo que en la versión clásica, sin embargo se tienen las siguientes restricciones:

- Las hormigas tienen un tiempo de vida de 100 iteraciones, posterior a esto mueren.

- Cada tipo de hormiga tiene una probabilidad de nacimiento asociada, los valores por defecto son 90 % para las normales, 8 % para las soldado y 2 % para las reinas.
- También se tiene la posibilidad de cambiar estos valores en la interfaz.
- Si una hormiga reina y una soldado se encuentran, se reproducen y nace una nueva hormiga dependiendo de las probabilidades anteriores.
- Se tiene un programa para poder visualizar la población de hormigas separando por tipo de hormiga.

3.3. Desarrollo

El programa fue desarrollado en Python 3 utilizando la biblioteca para desarrollar interfaces Tkinter y la biblioteca para graficar llamada matplotlib junto al uso de numpy que es una biblioteca que permite manipular matrices de una forma sencilla.

Archivo: hormiga.py

Este archivo contiene las clases Hormiga, Soldado y Reina, las cuales representan a cada tipo de hormiga y tienen métodos y atributos para que su implementación sea más sencilla.

```

1 colores_dict = {
2     "N": "red",
3     "S": "blue",
4     "E": "yellow",
5     "O": "green",
6 }
7
8 tipos_dict = {
9     "obrero": 1,
10    "soldado": 2,
11    "reina": 3,
12 }
13
14
15 class Hormiga:
16     def __init__(self, x=0, y=0, limite=0):
17         """ Direcciones:
18             N -> Norte
19             S -> Sur
20             E -> Este
21             O -> Oeste
22         """
23         self.x = x

```



```

24     self.y = y
25     self.limite = limite
26     self.orientacion = 'S'
27     self.color = "white"
28     self.tipo = tipos_dict["obrero"]
29     self.vida = 0
30
31     def mover(self, direccion):
32         self.vida += 1
33         if direccion == 0:
34             if self.orientacion == 'S':
35                 self.orientacion = 'O'
36             elif self.orientacion == 'O':
37                 self.orientacion = 'N'
38             elif self.orientacion == 'N':
39                 self.orientacion = 'E'
40             else:
41                 self.orientacion = 'S'
42         else:
43             if self.orientacion == 'S':
44                 self.orientacion = 'E'
45             elif self.orientacion == 'E':
46                 self.orientacion = 'N'
47             elif self.orientacion == 'N':
48                 self.orientacion = 'O'
49             else:
50                 self.orientacion = 'S'
51
52         if self.orientacion == 'S':
53             self.y += 1
54         elif self.orientacion == 'E':
55             self.x += 1
56         elif self.orientacion == 'N':
57             self.y -= 1
58         else:
59             self.x -= 1
60
61         self.y = self.checar_limite(self.y)
62         self.x = self.checar_limite(self.x)
63
64     def checar_limite(self, coord):
65         if coord < 0:
66             return self.limite - 1
67         if coord == self.limite:
68             return 0
69         return coord
70
71     def cambiar(self):
72         if self.orientacion == 'S':

```

```

73         self.orientacion = 'O'
74     elif self.orientacion == 'O':
75         self.orientacion = 'N'
76     elif self.orientacion == 'N':
77         self.orientacion = 'E'
78     else:
79         self.orientacion = 'S'
80
81
82 class Soldado(Hormiga):
83     def __init__(self, x=0, y=0, limite=0):
84         super().__init__(x, y, limite)
85         self.color = "orange"
86         self.tipo = tipos_dict["soldado"]
87
88
89 class Reina(Hormiga):
90     def __init__(self, x=0, y=0, limite=0):
91         super().__init__(x, y, limite)
92         self.color = "purple"
93         self.tipo = tipos_dict["reina"]

```

Archivo: main.py

Aquí se encuentra la implementación clásica de la hormiga de Langton junto a la interfaz en donde se despliega la simulación

```

1 from tkinter import Tk, Frame, Canvas, Button, Label, Entry,
   Scale, Scrollbar
2 import tkinter as tk
3 from tkcolorpicker import askcolor
4 import numpy as np
5 import random as pyrandom
6 from hormiga import Hormiga, colores_dict
7
8
9 class Ventana(Frame):
10     def __init__(self, parent):
11         # Elementos de la interfaz
12         Frame.__init__(self, parent)
13         self.grid(row=0, column=0)
14         self.parent = parent
15         self.canvas = None
16         self.input_tam = None
17         self.barra = None
18         self.default_color = "white"
19         self.btn_color = None
20
21         # Elementos de control
22         self.cuadros = None
23         self.matriz = None

```

```

24         self.tam = 500
25         self.tam_cuadro = 2
26         self.hormigas = list()
27         self.pausa = True
28         self.distribucion = .05
29
30     def init_ui(self):
31         self.parent.title("Hormiga de Lagnton")
32         self.pack(fill=tk.BOTH, expand=1)
33
34         self.canvas = Canvas(self, relief='raised', width=1000,
height=1000)
35         scroll = Scrollbar(self, orient=tk.VERTICAL)
36         scroll.pack(side=tk.RIGHT, fill=tk.Y)
37         scroll.config(command=self.canvas.yview)
38
39         self.canvas.config(yscrollcommand=scroll.set)
40         self.canvas.pack(side=tk.LEFT)
41
42         Label(self, text="Tamaño:", font=(20,)).pack(side=tk.
TOP)
43         self.input_tam = Entry(self, fg="black", bg="white")
44         self.input_tam.insert(10, "100")
45         self.input_tam.pack(side=tk.TOP)
46
47         Label(self, text="Porcentaje de hormigas", font=(20,)).
pack(side=tk.TOP)
48         self.barra = Scale(self, from_=0, to=100, orient=tk.
HORIZONTAL, tickinterval=50)
49         self.barra.set(5)
50         self.barra.pack(side=tk.TOP)
51
52         self.btn_color = Button(self, text="Color de la hormiga"
, command=self.get_color, bg=self.default_color)
53         self.btn_color.pack(side=tk.TOP)
54
55         btn_iniciar = Button(self, text="Iniciar/Reiniciar",
command=self.iniciar, font=(20,))
56         btn_iniciar.pack(side=tk.TOP)
57
58         btn_pausa = Button(self, text="Reanudar/Pausa", command=
self.empezar_detener, font=(20,))
59         btn_pausa.pack(side=tk.TOP)
60
61         Label(self, text="Relacion de colores y \n posicion de
las hormiga:", font=(20,)).pack(side=tk.TOP)
62         Label(self, text="Abajo", bg="blue", font=(20,)).pack(
side=tk.TOP)
63         Label(self, text="Arriba", bg="red", font=(20,)).pack(

```

```

side=tk.TOP)
64     Label(self, text="Izquierda", bg="green", font=(20,)).
pack(side=tk.TOP)
65     Label(self, text="Derecha", bg="yellow", fg="black",
font=(20,)).pack(side=tk.TOP)
66
67     def iniciar(self):
68         print("iniciar")
69         self.hormigas[:] = []
70         self.canvas.delete('all')
71         self.update_idletasks()
72
73         self.tam = int(self.input_tam.get())
74         self.tam_cuadro = 0
75         while self.tam_cuadro * self.tam < 1000:
76             self.tam_cuadro += 1
77         if self.tam_cuadro * self.tam > 1000:
78             self.tam_cuadro -= 1
79
80         self.distribucion = self.barra.get() / 100
81
82         self.pausa = True
83         self.cuadros = np.zeros(shape=(self.tam, self.tam),
dtype=int)
84         self.matriz = np.random.choice([1, 0], size=(self.tam,
self.tam), p=[self.distribucion, 1-self.distribucion])
85         self.redibujar()
86
87     def get_color(self):
88         color = askcolor()
89         if not color[1] is None:
90             self.default_color = color[1]
91             self.btn_color.configure(bg=self.default_color)
92
93     def empezar_detener(self):
94         print("empezar_detener")
95         self.pausa = not self.pausa
96         self.animacion()
97
98     def animacion(self):
99         if not self.pausa:
100             conjunto = set()
101             for hormiga in self.hormigas:
102                 if self.matriz[hormiga.y, hormiga.x] == 0:
103                     if (hormiga.y, hormiga.x) not in conjunto:
104                         self.matriz[hormiga.y, hormiga.x] = 1
105                         conjunto.add((hormiga.y, hormiga.x))
106                         self.canvas.itemconfig(self.cuadros[hormiga.
y, hormiga.x], fill=hormiga.color)

```

```

107         hormiga.mover(0)
108     else:
109         if (hormiga.y, hormiga.x) not in conjunto:
110             self.matriz[hormiga.y, hormiga.x] = 0
111             conjunto.add((hormiga.y, hormiga.x))
112             self.canvas.itemconfig(self.cuadros[hormiga.
y, hormiga.x], fill="black")
113             hormiga.mover(1)
114             self.canvas.itemconfig(self.cuadros[hormiga.y,
hormiga.x], fill=colores_dict[hormiga.orientacion])
115
116         self.update_idletasks()
117         self.after(5, self.animacion)
118
119     def borrar_cuadrado(self, event):
120         print("borrar_cuadrado")
121         item = self.canvas.find_closest(event.x, event.y)[0]
122         y, x = np.where(self.cuadros == item)
123         contador = 0
124         for hormiga in self.hormigas:
125             if hormiga.x == x and hormiga.y == y:
126                 if self.matriz[hormiga.y, hormiga.x] == 1:
127                     self.canvas.itemconfig(item, fill="white")
128                 else:
129                     self.canvas.itemconfig(item, fill="black")
130             break
131         contador += 1
132         if contador < len(self.hormigas):
133             self.hormigas.pop(contador)
134
135     def pulsar_cuadrado(self, event):
136         print("pulsar_cuadrado")
137         item = self.canvas.find_closest(event.x, event.y)[0]
138         y, x = np.where(self.cuadros == item)
139         crear = True
140         for hormiga in self.hormigas:
141             if hormiga.x == x and hormiga.y == y:
142                 hormiga.cambiar()
143                 crear = False
144                 self.canvas.itemconfig(item, fill=colores_dict[
hormiga.orientacion])
145             break
146
147         if crear:
148             hormiga = Hormiga(x[0], y[0], self.tam)
149             hormiga.color = self.default_color
150             self.hormigas.append(hormiga)
151             self.canvas.itemconfig(item, fill=colores_dict[
hormiga.orientacion])

```

```

152
153     def redibujar(self):
154         print("redibujar")
155         for i in range(self.tam):
156             for j in range(self.tam):
157                 if self.matriz[i, j] == 1:
158                     self.matriz[i, j] = 0
159                     hormiga = Hormiga(j, i, self.tam)
160                     hormiga.orientacion = np.random.choice(['N',
161 'S', 'E', 'O'])
162                     hormiga.color = "#%06x" % pyrandom.randint
163 (0, 0xFFFFFF)
164                     self.cuadros[i, j] = self.canvas.
165 create_rectangle(0 + (j * self.tam_cuadro),
166
167                     0 + (i * self.tam_cuadro),
168
169                     self.tam_cuadro + (j * self.tam_cuadro),
170
171                     self.tam_cuadro + (i * self.tam_cuadro),
172
173                     fill=colores_dict[hormiga.orientacion],
174
175                     width=0, tag="btncuadrado")
176                     self.hormigas.append(hormiga)
177                 else:
178                     self.cuadros[i, j] = self.canvas.
179 create_rectangle(0 + (j * self.tam_cuadro),
180
181                     0 + (i * self.tam_cuadro),
182
183                     self.tam_cuadro + (j * self.tam_cuadro),
184
185                     self.tam_cuadro + (i * self.tam_cuadro),
186
187                     fill="black", width=0, tag="btncuadrado")
188                     self.canvas.tag_bind("btncuadrado", "<Button-1>", self.
189 pulsar_cuadrado)
190                     self.canvas.tag_bind("btncuadrado", "<Button-3>", self.
191 borrar_cuadrado)
192
193                     self.update_idletasks()
194
195
196 def main():
197     root = Tk()
198     root.geometry("1360x750+0+0")
199     app = Ventana(root)
200     app.init_ui()

```

```

186     app.mainloop()
187
188
189 main()

```

Archivo: alternativo.py

Esta es la versión modificada de la hormiga de Langton con funcionalidad diferente a la clásica.

```

1 from tkinter import Tk, Frame, Canvas, Button, Label, Entry,
   Spinbox, Scrollbar, Radiobutton, IntVar, Scale, StringVar
2 import tkinter as tk
3 import numpy as np
4 from hormiga import Soldado, Hormiga, Reina, colores_dict,
   tipos_dict
5 import datetime
6 import time
7
8
9 class Ventana(Frame):
10     def __init__(self, parent):
11         Frame.__init__(self, parent)
12         self.parent = parent
13         self.canvas = None
14         self.input_tam = None
15         self.barra = None
16         self.barra_normal = None
17         self.barra_soldado = None
18         self.barra_reina = None
19         self.label_probabilidades = None
20
21         self.tiempo_vida = 50
22         self.cuadros = None
23         self.matriz = None
24         self.tam = 100
25         self.tam_cuadro = 10
26         self.hormigas = list()
27         self.pausa = True
28         self.distribucion = .05
29         self.mi_var = IntVar()
30         self.mi_var.set(1)
31         self.radio1 = None
32         self.radio2 = None
33         self.radio3 = None
34         self.tiempo = 0
35         self.contador = [0, 0, 0]
36         self.nom_archivo = "{}.csv".format(self.obtener_hora())
37         self.archivo = None
38         self.probabilidades = [.9, .08, .02]
39

```

```

40     def init_ui(self):
41         self.parent.title("Hormiga de Lagnton")
42         self.pack(fill=tk.BOTH, expand=1)
43
44         self.canvas = Canvas(self, relief='raised', width=1000,
45 height=1000)
46         scroll = Scrollbar(self, orient=tk.VERTICAL)
47         scroll.pack(side=tk.RIGHT, fill=tk.Y)
48         scroll.config(command=self.canvas.yview)
49
50         self.canvas.config(yscrollcommand=scroll.set)
51         self.canvas.pack(side=tk.LEFT)
52
53     Label(self, text="Tamanio:", font=(20,)).pack(side=tk.
54 TOP)
55     self.input_tam = Entry(self, fg="black", bg="white")
56     self.input_tam.insert(10, "100")
57     self.input_tam.pack(side=tk.TOP)
58
59     Label(self, text="Porcentaje de hormigas", font=(20,)).
60 pack(side=tk.TOP)
61     self.barra = Scale(self, from_=0, to=100, orient=tk.
62 HORIZONTAL, tickinterval=50)
63     self.barra.set(5)
64     self.barra.pack(side=tk.TOP)
65
66     Label(self, text="Tipo de hormiga:", font=(20,)).pack(
67 side=tk.TOP)
68     self.radio1 = Radiobutton(self, text="Obrera", variable=
69 self.mi_var, value=1, command=self.seleccion,
70 indicatoron=False, selectcolor
71 ="white", font=(20,), fg="black")
72     self.radio2 = Radiobutton(self, text="Soldado", variable
73 =self.mi_var, value=2, command=self.seleccion,
74 indicatoron=False, selectcolor
75 ="orange", font=(20,), fg="black")
76     self.radio3 = Radiobutton(self, text="Reina", variable=
77 self.mi_var, value=3, command=self.seleccion,
78 indicatoron=False, selectcolor
79 ="purple", font=(20,), fg="black")
80     self.radio1.pack(side=tk.TOP)
81     self.radio2.pack(side=tk.TOP)
82     self.radio3.pack(side=tk.TOP)
83     self.radio1.select()
84     self.radio2.deselect()
85     self.radio3.deselect()
86
87     self.label_probabilidades = Label(self, text="Suma de
88 probabilidades: 100%", font=(20,))

```



```

77         self.label_probabilidades.pack(side=tk.TOP)
78
79         Label(self, text="Probabilidad de hormigas normales:",
80 font=(20,)).pack(side=tk.TOP)
81         valor1 = StringVar(self)
82         valor1.set("90")
83         self.barra_normal = Spinbox(self, from_=0, to=100,
84 command=self.mover_spinner, textvariable=valor1)
85         self.barra_normal.pack(side=tk.TOP)
86
87         Label(self, text="Probabilidad de hormigas soldado:",
88 font=(20,)).pack(side=tk.TOP)
89         valor2 = StringVar(self)
90         valor2.set("8")
91         self.barra_soldado = Spinbox(self, from_=0, to=100,
92 command=self.mover_spinner, textvariable=valor2)
93         self.barra_soldado.pack(side=tk.TOP)
94
95         Label(self, text="Probabilidad de hormigas reina:", font
96 =(20,)).pack(side=tk.TOP)
97         valor3 = StringVar(self)
98         valor3.set("2")
99         self.barra_reina = Spinbox(self, from_=0, to=100,
100 command=self.mover_spinner, textvariable=valor3)
101         self.barra_reina.pack(side=tk.TOP)
102
103         btn_iniciar = Button(self, text="Iniciar/Reiniciar",
104 command=self.iniciar, font=(20,))
105         btn_iniciar.pack(side=tk.TOP)
106
107         btn_pausa = Button(self, text="Reanudar/Pausa", command=
108 self.empezar_detener, font=(20,))
109         btn_pausa.pack(side=tk.TOP)
110
111         Label(self, text="Relacion de colores y \n posicion de
112 las hormiga:", font=(20,)).pack(side=tk.TOP)
113         Label(self, text="Abajo", bg="blue", font=(20,)).pack(
114 side=tk.TOP)
115         Label(self, text="Arriba", bg="red", font=(20,)).pack(
116 side=tk.TOP)
117         Label(self, text="Izquierda", bg="green", font=(20,)).
118 pack(side=tk.TOP)
119         Label(self, text="Derecha", bg="yellow", fg="black",
120 font=(20,)).pack(side=tk.TOP)
121
122     def mover_spinner(self):
123         print("moviendo normal")
124         aux1 = int(self.barra_normal.get())
125         aux2 = int(self.barra_soldado.get())

```

```

113         aux3 = int(self.barra_reina.get())
114         self.proabilidades[0] = aux1 / 100
115         self.proabilidades[1] = aux2 / 100
116         self.proabilidades[2] = aux3 / 100
117         valor = aux1 + aux2 + aux3
118         texto = "Suma de probabilidades {} {}".format(valor)
119         self.label_probabilidades.configure(text=texto)
120
121     def iniciar(self):
122         print("iniciar")
123         self.nom_archivo = "{}.csv".format(self.obtener_hora())
124         self.archivo = open(self.nom_archivo, "w")
125         self.archivo.close()
126         self.contador[:] = [0, 0, 0]
127         self.tiempo = 0
128         self.hormigas[:] = []
129         self.canvas.delete('all')
130         self.update_idletasks()
131
132         self.tam = int(self.input_tam.get())
133         self.tam_cuadro = 0
134         while self.tam_cuadro * self.tam < 1000:
135             self.tam_cuadro += 1
136         if self.tam_cuadro * self.tam > 1000:
137             self.tam_cuadro -= 1
138
139         self.distribucion = self.barra.get() / 100
140         self.proabilidades[0] = int(self.barra_normal.get()) /
141         self.proabilidades[1] = int(self.barra_soldado.get()) /
142         self.proabilidades[2] = int(self.barra_reina.get()) /
143
144         self.pausa = True
145         self.cuadros = np.zeros(shape=(self.tam, self.tam),
146 dtype=int)
147         self.matriz = np.random.choice([1, 0], size=(self.tam,
148 self.tam), p=[self.distribucion, 1-self.distribucion])
149         self.redibujar()
150
151     def seleccion(self):
152         print(str(self.mi_var.get()))
153
154     def crear_hormiga(self, j, i):
155         tipo = np.random.choice([1, 2, 3], p=self.proabilidades
)
156         if tipo == 1:
157             hormiga = Hormiga(j, i, self.tam)

```

```

156         self.contador[0] += 1
157     elif tipo == 2:
158         hormiga = Soldado(j, i, self.tam)
159         self.contador[1] += 1
160     else:
161         hormiga = Reina(j, i, self.tam)
162         self.contador[2] += 1
163     hormiga.orientacion = np.random.choice(['N', 'S', 'E', 'O'])
164     return hormiga
165
166     @staticmethod
167     def obtener_hora():
168         return datetime.datetime.fromtimestamp(time.time()).
169         strftime('%Y-%m-%d_%H:%M%S')
170
171     def redibujar(self):
172         print("redibujar")
173         for i in range(self.tam):
174             for j in range(self.tam):
175                 if self.matriz[i, j] == 1:
176                     self.matriz[i, j] = 0
177                     hormiga = self.crear_hormiga(j, i)
178                     self.cuadros[i, j] = self.canvas.
179                     create_rectangle(0 + (j * self.tam_cuadro),
180
181                     0 + (i * self.tam_cuadro),
182
183                     self.tam_cuadro + (j * self.tam_cuadro),
184
185                     self.tam_cuadro + (i * self.tam_cuadro),
186
187                     fill=hormiga.color,
188
189                     width=0, tag="btncuadrado")
190                     self.hormigas.append(hormiga)
191                 else:
192                     self.cuadros[i, j] = self.canvas.
193                     create_rectangle(0 + (j * self.tam_cuadro),
194
195                     0 + (i * self.tam_cuadro),
196
197                     self.tam_cuadro + (j * self.tam_cuadro),
198
199                     self.tam_cuadro + (i * self.tam_cuadro),
200
201                     fill="black", width=0, tag="btncuadrado")
202                     self.canvas.tag_bind("btncuadrado", "<Button-1>", self.
203                     pulsar_cuadrado)

```

```

191         self.canvas.tag_bind("btncuadrado", "<Button-3>", self.
borrar_cuadrado)
192         self.update_idletasks()
193         print(self.contador)
194
195     def borrar_cuadrado(self, event):
196         print("borrar_cuadrado")
197         item = self.canvas.find_closest(event.x, event.y)[0]
198         y, x = np.where(self.cuadros == item)
199         contador = 0
200         for hormiga in self.hormigas:
201             if hormiga.x == x and hormiga.y == y:
202                 self.contador[hormiga.tipo-1] -= 1
203                 if self.matriz[hormiga.y, hormiga.x] == 1:
204                     self.canvas.itemconfig(item, fill="white")
205                 else:
206                     self.canvas.itemconfig(item, fill="black")
207             break
208             contador += 1
209         if contador < len(self.hormigas):
210             self.hormigas.pop(contador)
211
212     def pulsar_cuadrado(self, event):
213         print("pulsar_cuadrado")
214         item = self.canvas.find_closest(event.x, event.y)[0]
215         y, x = np.where(self.cuadros == item)
216         crear = True
217         for hormiga in self.hormigas:
218             if hormiga.x == x and hormiga.y == y:
219                 hormiga.cambiar()
220                 crear = False
221                 self.canvas.itemconfig(item, fill=colores_dict[
hormiga.orientacion])
222             break
223
224         if crear:
225             if self.mi_var.get() == 1:
226                 hormiga = Hormiga(x[0], y[0], self.tam)
227                 self.contador[0] += 1
228             elif self.mi_var.get() == 2:
229                 hormiga = Soldado(x[0], y[0], self.tam)
230                 self.contador[1] += 1
231             else:
232                 hormiga = Reina(x[0], y[0], self.tam)
233                 self.contador[2] += 1
234             self.hormigas.append(hormiga)
235             self.canvas.itemconfig(item, fill=colores_dict[
hormiga.orientacion])
236

```

```

237     def animacion(self):
238         if not self.pausa:
239             archivo = open(self.nom_archivo, "a")
240             archivo.write("{}{}{}{}\n".format(self.tiempo,
self.contador[0], self.contador[1], self.contador[2]))
241             archivo.close()
242             reinas = list()
243             soldados = list()
244             cont = 0
245             for hormiga in self.hormigas:
246                 if hormiga.tipo == tipos_dict["reina"]:
247                     reinas.append(cont)
248                 elif hormiga.tipo == tipos_dict["soldado"]:
249                     soldados.append(cont)
250                 cont += 1
251             for i in reinas:
252                 for j in soldados:
253                     if self.hormigas[i].x == self.hormigas[j].x
and self.hormigas[i].y == self.hormigas[j].y:
254                         self.hormigas.append(self.
crear_hormiga(self.hormigas[i].x, self.hormigas[i].y))
255
256             conjunto = set()
257             for hormiga in self.hormigas:
258                 if hormiga.vida == self.tiempo_vida:
259                     self.contador[hormiga.tipo-1] -= 1
260                     self.hormigas.remove(hormiga)
261                     continue
262                 if self.matriz[hormiga.y, hormiga.x] == 0:
263                     if (hormiga.y, hormiga.x) not in conjunto:
264                         self.matriz[hormiga.y, hormiga.x] = 1
265                         conjunto.add((hormiga.y, hormiga.x))
266                         self.canvas.itemconfig(self.cuadros[hormiga.
y, hormiga.x], fill=hormiga.color)
267                         hormiga.mover(0)
268                 else:
269                     if (hormiga.y, hormiga.x) not in conjunto:
270                         self.matriz[hormiga.y, hormiga.x] = 0
271                         conjunto.add((hormiga.y, hormiga.x))
272                         self.canvas.itemconfig(self.cuadros[hormiga.
y, hormiga.x], fill="black")
273                         hormiga.mover(1)
274                         self.canvas.itemconfig(self.cuadros[hormiga.y,
hormiga.x], fill=colores_dict[hormiga.orientacion])
275
276             self.update_idletasks()
277             self.after(100, self.animacion)
278             self.tiempo += 1
279

```

```

280     def empezar_detener(self):
281         print("empezar_detener")
282         self.pausa = not self.pausa
283         self.animacion()
284
285
286     def main():
287         root = Tk()
288         root.geometry("1360x750+0+0")
289         app = Ventana(root)
290         app.init_ui()
291         app.mainloop()
292
293
294     main()

```

Archivo: grafica.py

El código de este archivo permite graficar la cantidad de hormigas de cada tipo de la versión modifica en donde se tienen tres tipos de hormigas.

```

1  import matplotlib.pyplot as plt
2  import matplotlib.animation as animation
3  import sys
4
5  fig = plt.figure("Historia de la hormiga de Langton")
6  fig.suptitle("Historia de la hormiga de Langton")
7  fig.add_axes()
8  ax1 = fig.add_subplot(1, 1, 1)
9  archivo = sys.argv[1]
10
11
12  def animacion(i, *args):
13      info = open(args[0], "r").read()
14      lineas = info.split("\n")
15      xs = []
16      obreras = []
17      soldados = []
18      reinas = []
19      for linea in lineas:
20          if len(linea) > 1:
21              x, obrera, soldado, reina = linea.split(",")
22              xs.append(int(x))
23              obreras.append(int(obrera))
24              soldados.append(int(soldado))
25              reinas.append(int(reina))
26
27      ax1.clear()
28      ax1.plot(xs, obreras, label="Obreras")
29      ax1.plot(xs, soldados, label="Soldados")
30      ax1.plot(xs, reinas, label="Reinas")

```

```

31     legend = ax1.legend()
32     legend.get_frame()
33     ax1.set_xlabel('Generacion')
34     ax1.set_ylabel('Cantidad')
35
36
37 ani = animation.FuncAnimation(fig, animacion, interval=100,
38                               fargs=(archivo,))
39 plt.show()

```

3.4. Pruebas

3.4.1. Hormiga de Langton original

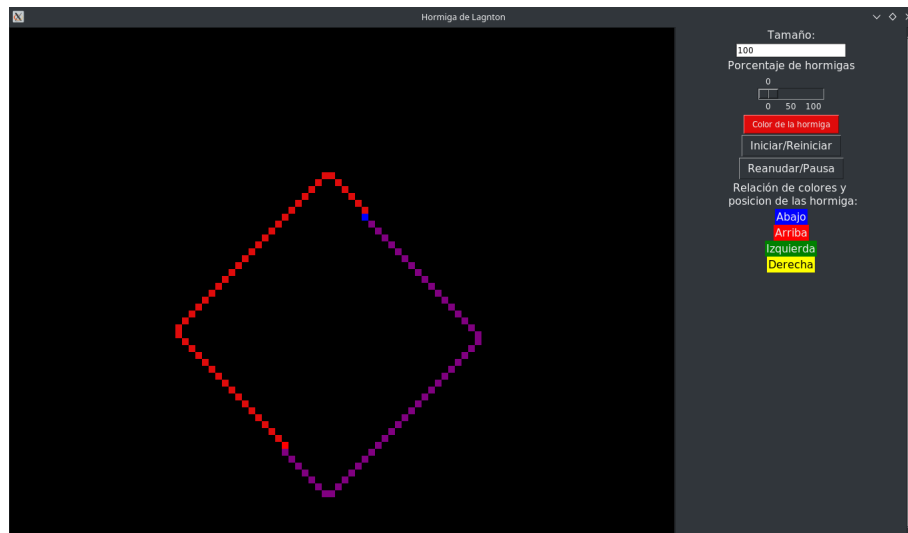


Figura 56: Dos hormigas insertadas por el usuario y con colores diferentes, en un espacio de 100x100

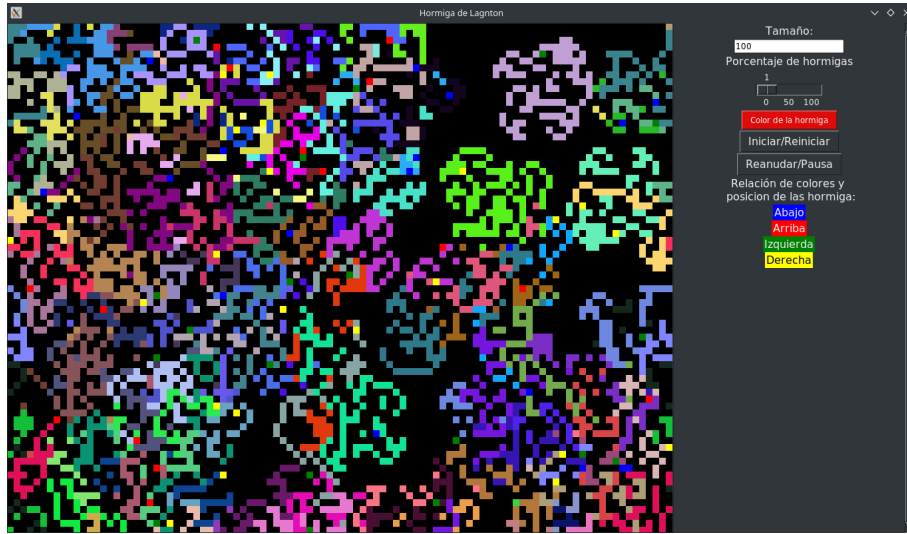


Figura 57: Varias hormigas generadas aleatoriamente en un espacio de 100x100

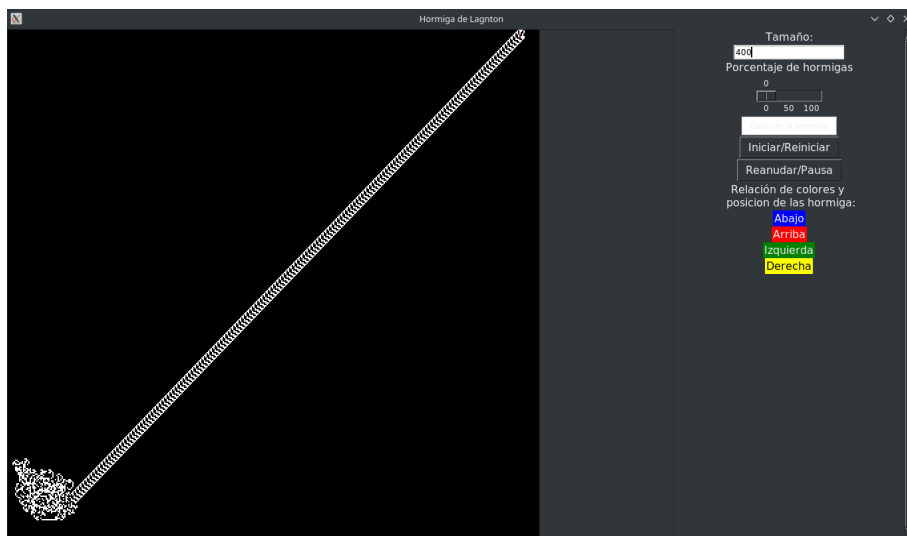


Figura 58: Hormiga que se mueve al “infinito” después de varias iteraciones en un espacio de 400x400

3.4.2. Hormiga de Langton modificada

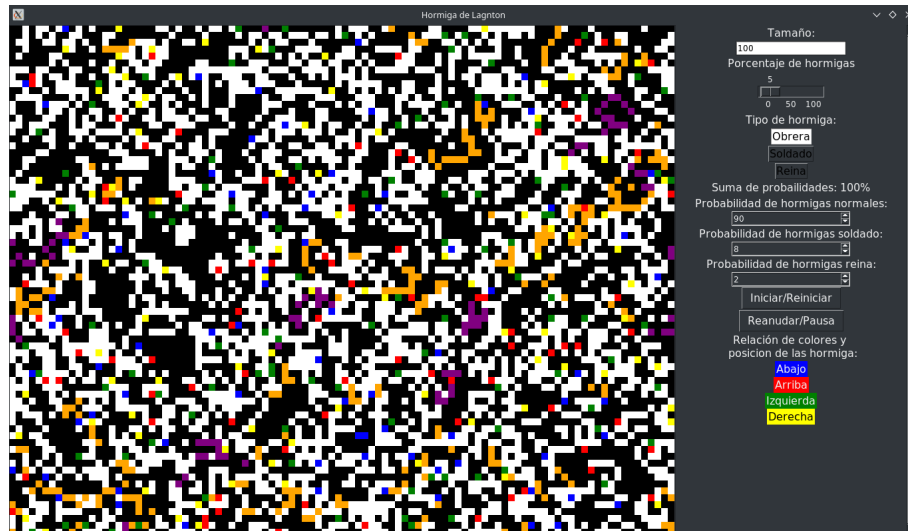


Figura 59: Probabilidad por defecto

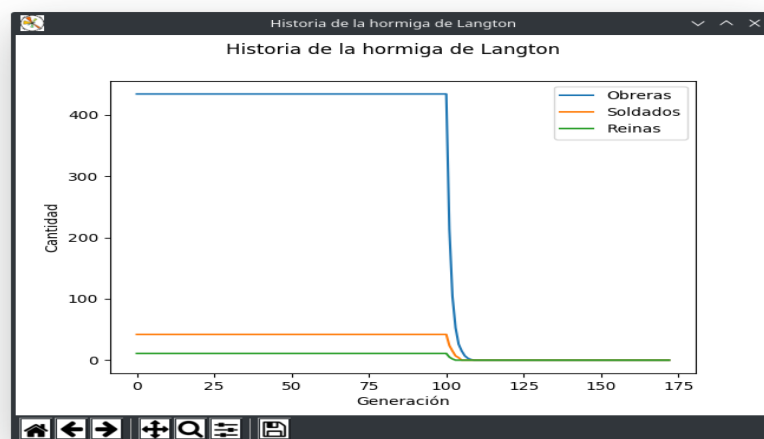


Figura 60: Gráfica de la colonia anterior

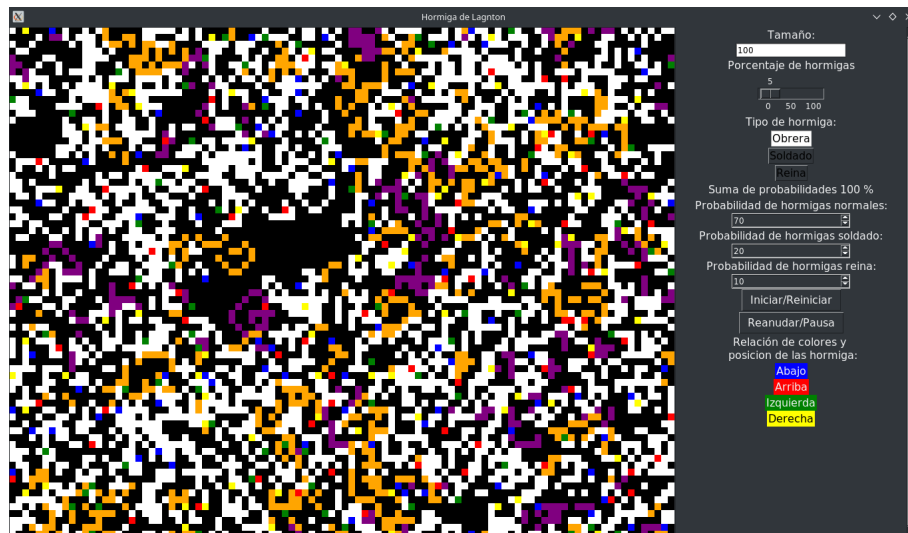


Figura 61: Probabilidad de 70, 20, 10

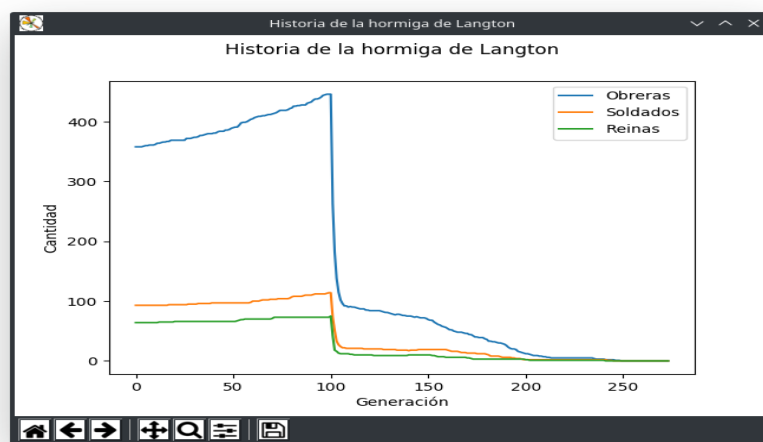


Figura 62: Gráfica de la colonia anterior

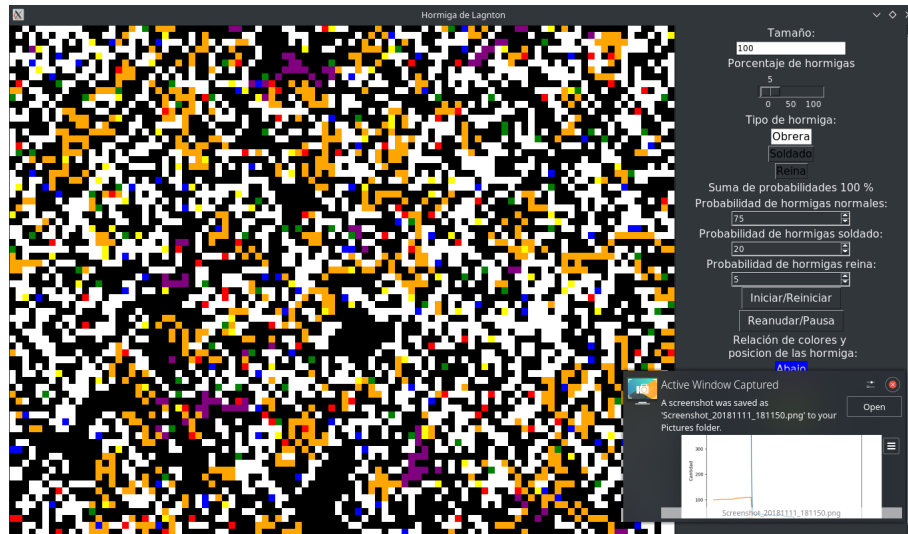


Figura 63: Probabilidad de 75, 25, 5

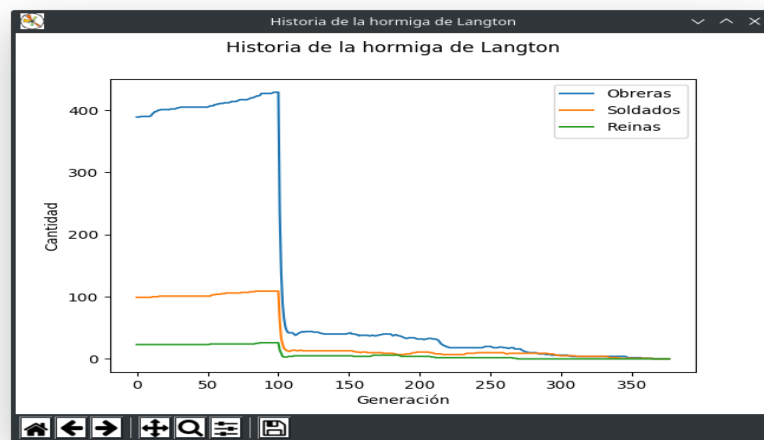


Figura 64: Gráfica de la colonia anterior

Es importante señalar que el porcentaje de hormigas reina no pude ser muy alto, por lo general no mayor a 10 porque entonces se dispara la cantidad de hormigas muy rápidamente debido a que empiezan a nacer más reinas y junto con las soldado aumentan sin control.

3.5. Conclusiones

La hormiga de Langton es otro de ejemplo de un autómata celular bastante interesante, los comportamientos que se generan en este son bastante peculiares y complejos ya que estos tratan de describir toda una colonia. Respecto a la implementación no hubo muchos problemas, sin embargo, el rendimiento del programa no es tan bueno como se esperaba debido a que entre mayor sea el espacio a trabajar más lenta se vuelve la simulación.

En cuanto a las pruebas hechas en la versión modificada se aprecia que debido a la restricción de tiempo de vida de las hormigas la colonia tiende a morir bastante rápido, incluso en aquella en la cual hay muchas hormigas reinas y soldado, por lo que la única forma de resolver esto es que el tiempo de vida de las hormigas sea muy alta para tener el tiempo suficiente para que generen nuevas hormigas.

El hecho de que la probabilidad de que la hormiga que se genere sea una reina es muy baja es otro problema ya que al no nacer tantas reinas la colonia no se puede mantener.

Referencias

- [1] Reyes Gómez, D., *Descripción y Aplicaciones de los Autómatas Celulares*. cinvestav, 2011.
- [2] De Felipe Vargas, D. and Sanchez Salazar C., *Comportamiento colectivo no trivial implementado en robots de bajo costo: el caso de “La hormiga de Langton”*. Instituto Politécnico Nacional, 2016.