

# Entrega de trabajos del primer parcial

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

19 de septiembre de 2018

# Índice

|                             |           |
|-----------------------------|-----------|
| <b>1. Maquina de Turing</b> | <b>3</b>  |
| 1.1. Introducción . . . . . | 3         |
| 1.2. Desarrollo . . . . .   | 4         |
| 1.3. Pruebas . . . . .      | 7         |
| 1.4. Conclusiones . . . . . | 12        |
| <b>2. Juego de la Vida</b>  | <b>13</b> |
| 2.1. Introducción . . . . . | 13        |
| 2.2. Desarrollo . . . . .   | 13        |
| 2.3. Pruebas . . . . .      | 20        |
| 2.4. Conclusiones . . . . . | 23        |
| <b>Referencias</b>          | <b>24</b> |

# 1. Maquina de Turing

## 1.1. Introducción

La elaboración de este programa consistió en elaborar un maquina de Turing capaz de duplicar la cantidad de unos en una cadena de unos, es decir, si la cadena que se ingresa es 11 entonces la cadena de salida sera 1111.

Es por esto que la maquina sólo aceptara unos en la entrada mientras que los símbolos de la cinta incluirán a la X y la Y. De esta forma la maquina de Turing para este problema se define como:

$$M = (\{q_0, q_1, q_2, q_3\}, \{1\}, \{1, X, Y, B\}, \delta, q_0, B, \{q_4\})$$

donde  $\delta$  se especifica en la siguiente tabla:

|        | Símbolo   |           |           |           |
|--------|-----------|-----------|-----------|-----------|
| Estado | 1         | X         | Y         | B         |
| 0      | (1, X, R) | -         | (3, 1, R) | -         |
| 1      | (1, 1, R) | -         | (1, Y, R) | (1, Y, L) |
| 2      | (2, 1, L) | (0, 1, R) | (2, Y, L) | -         |
| 3      | -         | -         | (3, 1, R) | -         |

EL funcionamiento de esta maquina se puede entender mejor con el diagrama de la figura 1

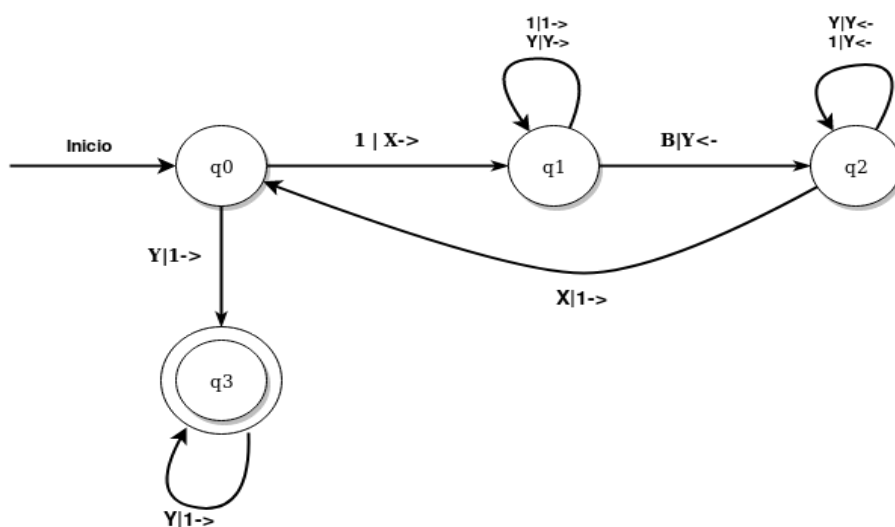


Figura 1: Representación gráfica de las transiciones de la maquina de Turing

El funcionamiento del programa empieza cuando se inserta una cadena de unos y la maquina empezara a trabajar, además de generar una cadena final se muestra una animación del como se están realizando las transiciones en la cinta de la maquina, por otro lado se imprime en consola el historial de movimientos que se hacen, este historial también se guarda en un archivo de texto.

## 1.2. Desarrollo

El código de este programa fue realizado en Python 3.7 y se utilizo la biblioteca tkinter para la parte gráfica.

Archivo: turing.py Esta clase es la que modela la maquina de Turing, los parámetros importantes son: el estado inicial, los estados finales, la cadena de entrada y la función de transición.

```
1 class MaquinaTuring:
2     """MaquinaTuring"""
3     def __init__(self, estado_inicial, estados_finales, cadena,
4         transiciones):
5         self.transiciones = transiciones
6         self.inicial = estado_inicial
7         self.finales = estados_finales
8         self.cinta = list(cadena)
9         self.estado_actual = self.inicial
10        self.apuntador = 0
11        self.blanco = "B"
12        self.direccion = None
13
14    def consumir(self):
15        """Toma un simbolo de la cinta y lo evalua en la funcion
16        de
17        transicion"""
18        if len(self.cinta) - 1 < self.apuntador:
19            caracter = 'B'
20        else:
21            caracter = self.cinta[self.apuntador]
22        tupla = (self.estado_actual, caracter)
23        if tupla in self.transiciones:
24            siguiente = self.transiciones[tupla]
25            if len(self.cinta) - 1 < self.apuntador:
26                self.cinta.append(self.blanco)
27            if self.apuntador < 0:
28                self.cinta.insert(0, self.blanco)
29            self.cinta[self.apuntador] = siguiente[1]
30            if siguiente[2] == "R":
31                self.apuntador += 1
32            else:
```

```

31         self.apuntador -= 1
32
33         self.estado_actual = siguiente[0]
34         self.direccion = siguiente[2]
35         return True
36     else:
37         return False
38
39     def es_final(self):
40         """Metodo para comprobar si nos encontramos en un estado
41         final"""
42         if self.estado_actual in self.finales:
43             if len(self.cinta) - 1 < self.apuntador or self.
44                 apuntador < 0:
45                 return True
46             return False

```

Archivo: diagrama.py Este archivo implementa la clase MaquinaTuring.py y se declaran los parámetros que se pasaran a este archivo así como la captura de la cadena y la escritura del registro de transiciones en consola y en archivo de texto, sin olvidar la animación de dichas transiciones.

```

1  import tkinter as tk
2  import time
3  from tkinter import font as tkfont
4  from turing import MaquinaTuring
5
6  # Tabla de transiciones que modela el automata
7  transiciones = {
8      ("q0", "1"): ("q1", "X", "R"),
9      ("q0", "Y"): ("q3", "1", "R"),
10     ("q1", "1"): ("q1", "1", "R"),
11     ("q1", "Y"): ("q1", "Y", "R"),
12     ("q1", "B"): ("q2", "Y", "L"),
13     ("q2", "1"): ("q2", "1", "L"),
14     ("q2", "X"): ("q0", "1", "R"),
15     ("q2", "Y"): ("q2", "Y", "L"),
16     ("q3", "Y"): ("q3", "1", "R"),
17 }
18
19 entrada = input("Ingrese la cadena de unos: ")
20 maquina = MaquinaTuring("q0", "q3", entrada, transiciones)
21
22 # Configuracion de la ventana
23 gui = tk.Tk()
24 gui.geometry("600x400+100+100")
25 gui.title("Maquina de Turing")
26 c = tk.Canvas(gui, width=600, height=400)
27 c.pack()

```

```

28 bold_font = tkfont.Font(family="Arial", size=24)
29
30 # Principales componentes que se animan
31 control = c.create_rectangle(150, 100, 200, 150, fill="lightblue
    ")
32 flecha = c.create_line(175, 150, 175, 175, arrow=tk.LAST, width
    =3)
33 texto = c.create_text(165, 200, text=''.join(maquina.cinta),
    font=bold_font,
34                        anchor=tk.W)
35 estado = c.create_text(160, 125, text=maquina.estado_actual,
    font=bold_font,
36                        anchor=tk.W)
37
38 archivo = open("salida.txt", "w+")
39 # Mientras no llegues a un estado final continua
40 while not maquina.es_final():
41     print('Cadena: {}'.format(''.join(maquina.cinta)))
42     print('Estado actual: {}, apuntador: {}'.format(maquina.
    estado_actual,
43             maquina.apuntador+1))
44
45     archivo.write('Cadena: {}\n'.format(''.join(maquina.cinta)))
46     archivo.write('Estado actual: {}, apuntador: {}\n'
47             .format(maquina.estado_actual, maquina.
    apuntador+1))
48     if not maquina.consumir():
49         print('*' * 20)
50         archivo.write('*' * 20)
51         archivo.write('\n')
52         break
53     print('Siguiente estado: {}'.format(maquina.estado_actual))
54     print('*'*20)
55
56     archivo.write('Siguiente estado: {}\n'.format(maquina.
    estado_actual))
57     archivo.write('*' * 20)
58     archivo.write('\n')
59
60     gui.update()
61     time.sleep(1)
62     c.itemconfigure(texto, text=''.join(maquina.cinta), anchor=
    tk.W)
63     c.itemconfigure(estado, text=maquina.estado_actual)
64     if maquina.direccion == 'R':
65         c.move(control, 19, 0)
66         c.move(flecha, 19, 0)
67         c.move(estado, 19, 0)
68     else:

```

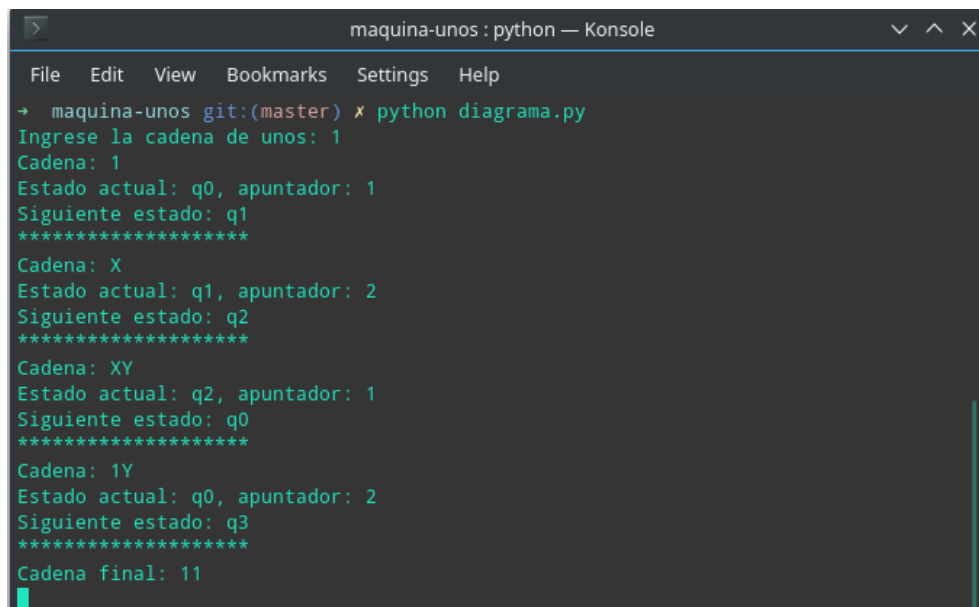
```

69         c.move(control, -19, 0)
70         c.move(estado, -19, 0)
71         c.move(flecha, -19, 0)
72
73     print('Cadena final: {}'.format(''.join(maquina.cinta)))
74     archivo.write('Cadena final: {}\n'.format(''.join(maquina.cinta)
75 ))
76     archivo.close()
77     gui.mainloop()

```

### 1.3. Pruebas

El siguiente ejemplo es la cadena con un solo uno.



```

maquina-unos: python — Konsole
File Edit View Bookmarks Settings Help
→ maquina-unos git:(master) x python diagrama.py
Ingrese la cadena de unos: 1
Cadena: 1
Estado actual: q0, apuntador: 1
Siguiendo estado: q1
*****
Cadena: X
Estado actual: q1, apuntador: 2
Siguiendo estado: q2
*****
Cadena: XY
Estado actual: q2, apuntador: 1
Siguiendo estado: q0
*****
Cadena: 1Y
Estado actual: q0, apuntador: 2
Siguiendo estado: q3
*****
Cadena final: 11

```

Figura 2: Salida en consola

```
diagrama.py x salida.txt x turing.py x
1 Cadena: 1
2 Estado actual: q0, apuntador: 1
3 Siguiente estado: q1
4 *****
5 Cadena: X
6 Estado actual: q1, apuntador: 2
7 Siguiente estado: q2
8 *****
9 Cadena: XY
10 Estado actual: q2, apuntador: 1
11 Siguiente estado: q0
12 *****
13 Cadena: 1Y
14 Estado actual: q0, apuntador: 2
15 Siguiente estado: q3
16 *****
17 Cadena final: 11
18
```

Figura 3: Registro de transiciones

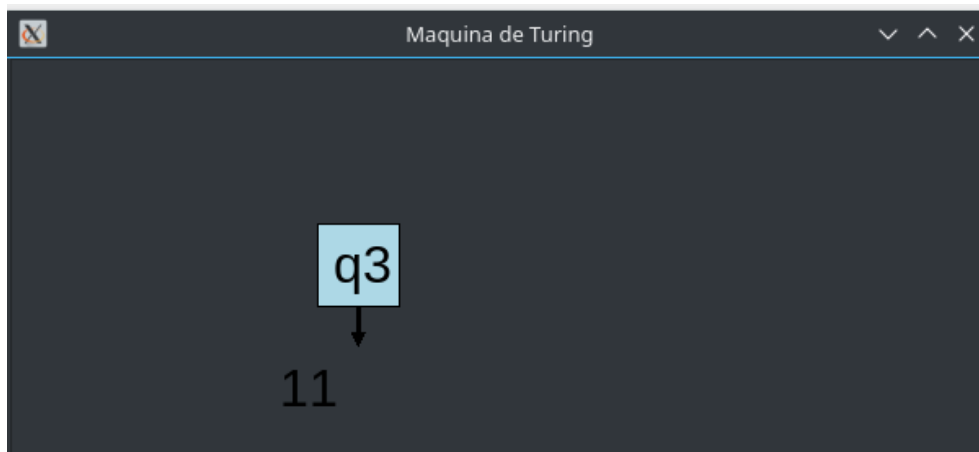


Figura 4: Representación gráfica de las transiciones de la maquina de Turing

En este ejemplo se inserta una cadena con dos unos.



```
maquina-unos : python — Konsole

File Edit View Bookmarks Settings Help

→ maquina-unos git:(master) python diagrama.py
Ingrese la cadena de unos: 11
Cadena: 11
Estado actual: q0, apuntador: 1
Siguiendo estado: q1
*****
Cadena: X1
Estado actual: q1, apuntador: 2
Siguiendo estado: q1
*****
Cadena: X1
Estado actual: q1, apuntador: 3
Siguiendo estado: q2
*****
Cadena: X1Y
Estado actual: q2, apuntador: 2
Siguiendo estado: q2
*****
Cadena: X1Y
Estado actual: q2, apuntador: 1
Siguiendo estado: q0
*****
Cadena: 11Y
Estado actual: q0, apuntador: 2
Siguiendo estado: q1
*****
Cadena: 1XY
Estado actual: q1, apuntador: 3
Siguiendo estado: q1
*****
Cadena: 1XY
Estado actual: q1, apuntador: 4
Siguiendo estado: q2
*****
Cadena: 1XY
Estado actual: q2, apuntador: 3
Siguiendo estado: q2
*****
Cadena: 1XY
Estado actual: q2, apuntador: 2
Siguiendo estado: q0
```

Figura 5: Salida en consola parte uno

```
*****
Cadena: 1XY
Estado actual: q2, apuntador: 3
Siguiendo estado: q2
*****
Cadena: 1XY
Estado actual: q2, apuntador: 2
Siguiendo estado: q0
*****
Cadena: 11Y
Estado actual: q0, apuntador: 3
Siguiendo estado: q3
*****
Cadena: 111Y
Estado actual: q3, apuntador: 4
Siguiendo estado: q3
*****
Cadena final: 1111
```

Figura 6: Salida en consola parte dos

```
diagrama.py x salida.txt x
1 Cadena: ·11
2 Estado actual: ·q0, ·apuntador: ·1
3 Siguiete estado: ·q1
4 *****
5 Cadena: ·X1
6 Estado actual: ·q1, ·apuntador: ·2
7 Siguiete estado: ·q1
8 *****
9 Cadena: ·X1
10 Estado actual: ·q1, ·apuntador: ·3
11 Siguiete estado: ·q2
12 *****
13 Cadena: ·X1Y
14 Estado actual: ·q2, ·apuntador: ·2
15 Siguiete estado: ·q2
16 *****
17 Cadena: ·X1Y
18 Estado actual: ·q2, ·apuntador: ·1
19 Siguiete estado: ·q0
20 *****
21 Cadena: ·11Y
22 Estado actual: ·q0, ·apuntador: ·2
23 Siguiete estado: ·q1
24 *****
25 Cadena: ·1XY
26 Estado actual: ·q1, ·apuntador: ·3
27 Siguiete estado: ·q1
28 *****
29 Cadena: ·1XY
30 Estado actual: ·q1, ·apuntador: ·4
31 Siguiete estado: ·q2
32 *****
33 Cadena: ·1XYY
34 Estado actual: ·q2, ·apuntador: ·3
35 Siguiete estado: ·q2
36 *****
37 Cadena: ·1XYY
38 Estado actual: ·q2, ·apuntador: ·2
39 Siguiete estado: ·q0
40 *****
41 Cadena: ·11YY
42 Estado actual: ·q0, ·apuntador: ·3
43 Siguiete estado: ·q3
44 *****
45 Cadena: ·111Y
46 Estado actual: ·q3, ·apuntador: ·4
47 Siguiete estado: ·q3
48 *****
49 Cadena final: ·1111
50
```

Figura 7: Registro de transiciones

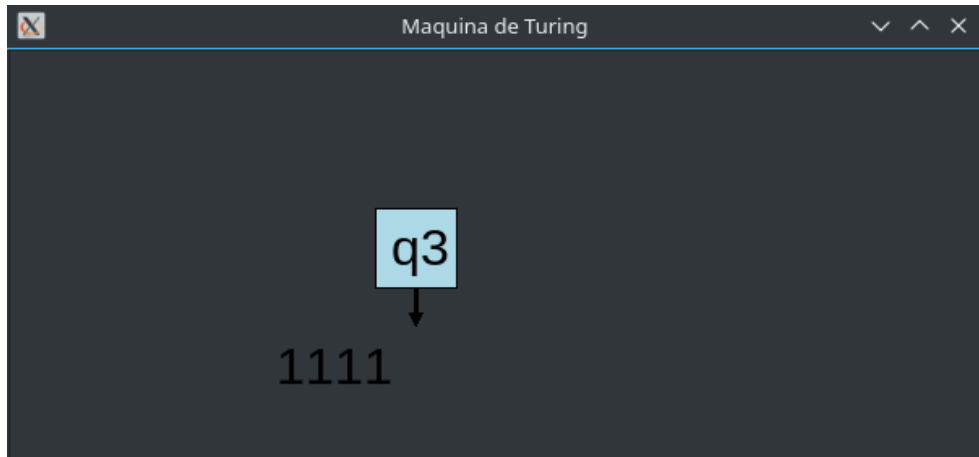


Figura 8: Representación gráfica de las transiciones de la maquina de Turing

#### 1.4. Conclusiones

El hacer este programa probó lo útil y poderosa que es la maquina de Turing en la solución de problemas, ya que como es evidente si un problema no se puede solucionar usando una maquina de Turing entonces es no computable y a pesar que este problema de duplicar una cadena no es difícil el implementar una maquina de Turing resulta en una solución bastante practica y fácil de implementar.

## 2. Juego de la Vida

### 2.1. Introducción

Este programa implementa el Juego de la Vida de Conway con la siguiente funcionalidad. 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.

- 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 1000 por 1000 para poder observar un comportamiento que nos proporcione información.

### 2.2. 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.

```
1 from tkinter import Tk, Canvas, Frame, Button, Entry, Label,
   Scale
2 from tkinter import BOTH, TOP, LEFT, HORIZONTAL
3 import numpy as np
4 from tkcolorpicker import askcolor
5
6 class Ventana(Frame):
7     def __init__(self, parent):
8         Frame.__init__(self, parent)
9         self.parent = parent
10        # Elementos interfaz
```

```

11         self.ceros = "white"
12         self.unos = "black"
13         self.regla = [2, 3, 3, 3]
14         self.e1 = None
15         self.e2 = None
16         self.contador = 0
17         self.colorBtn1 = None
18         self.colorBtn2 = None
19         self.barra = None
20         self.canvas = None
21         # variables del juego de la vida
22         self.pausa = True
23         self.tam = 100
24         self.tam_cuadro = 10
25         self.distribucion = .5
26         self.cuadritos = np.zeros(shape=(self.tam, self.tam),
dtype=int)
27         self.celulas = np.random.randint(2, size=(self.tam, self
.tam), dtype=int)
28         self.historia_x = list()
29         self.historia_y = list()
30         self.tiempo = 0
31         # Historial de unos
32         archivo = open("grafica.txt", "w")
33         archivo.close()
34         self.initUI()
35
36     def iniciar(self):
37         archivo = open("grafica.txt", "w")
38         archivo.close()
39         self.canvas.delete('all')
40         self.update_idletasks()
41         self.pausa = True
42         self.contador = 0
43         self.tiempo = 0
44         self.tam = int(self.e2.get())
45         self.tam_cuadro = 0
46         while self.tam_cuadro*self.tam < 1000:
47             self.tam_cuadro += 1
48         if self.tam_cuadro*self.tam > 1000:
49             self.tam_cuadro -= 1
50         print(self.tam_cuadro)
51         self.distribucion = self.barra.get()/100
52         self.celulas = np.random.choice([1, 0], size=(self.tam,
self.tam), p=[self.distribucion, 1-self.distribucion])
53         self.cuadritos = np.zeros(shape=(self.tam, self.tam),
dtype=int)
54         texto = self.e1.get().split(",")
55         self.regla[0] = int(texto[0])

```

```

56         self.regla[1] = int(texto[1])
57         self.regla[2] = int(texto[2])
58         self.regla[3] = int(texto[3])
59         self.contar_unos()
60         self.re_dibujar()
61
62
63     def contar_unos(self):
64         for i in range(self.tam):
65             for j in range(self.tam):
66                 if self.celulas[i, j] == 1:
67                     self.contador += 1
68
69     def pulsar_cuadrado(self, event):
70         item = self.canvas.find_closest(event.x, event.y)[0]
71         x, y = np.where(self.cuadritos==item)
72         if self.canvas.itemcget(item, "fill") == self.unos:
73             self.canvas.itemconfig(item, fill=self.ceros)
74             self.celulas[x[0]][y[0]] = 0
75         else:
76             self.canvas.itemconfig(item, fill=self.unos)
77             self.celulas[x[0]][y[0]] = 1
78
79
80     def re_dibujar(self):
81         print("REDIBUJAR")
82         for i in range(self.tam):
83             for j in range(self.tam):
84                 if self.celulas[i, j] == 0:
85                     self.cuadritos[i, j] = self.canvas.
create_rectangle(0 + (j * self.tam_cuadro),
86
87                     0 + (i * self.tam_cuadro),
88
89                     self.tam_cuadro + (j * self.tam_cuadro),
90
91                     self.tam_cuadro + (i * self.tam_cuadro),
92
93                     fill=self.ceros, width=0, tag="btncuadrado")
94                 else:
95                     self.cuadritos[i, j] = self.canvas.
create_rectangle(0 + (j * self.tam_cuadro),
96
97                     0 + (i * self.tam_cuadro),
98
99                     self.tam_cuadro + (j * self.tam_cuadro),
100
101                     self.tam_cuadro + (i * self.tam_cuadro),

```

```

95         fill=self.unos, width=0, tag="btncuadruto")
96         self.contador += 1
97
98         self.canvas.tag_bind("btncuadruto", "<Button-1>", self.
99         pulsar_cuadruto)
100         self.update()
101
102     def initUI(self):
103         self.parent.title("Layout Test")
104         self.pack(fill = BOTH, expand = 1)
105
106         self.canvas = Canvas(self, relief = 'raised', width =
107         1200, height = 800)
108         self.canvas.pack(side = LEFT)
109
110         Label(self, text="Regla:").pack(side=TOP)
111         self.e1 = Entry(self, fg="black", bg="white")
112         self.e1.insert(10, "2,3,3,3")
113         self.e1.pack(side=TOP)
114
115         Label(self, text="Tamano:").pack(side=TOP)
116         self.e2 = Entry(self, fg="black", bg="white")
117         self.e2.insert(10, "100")
118         self.e2.pack(side=TOP)
119
120         Label(self, text="Porcentaje de unos").pack(side=TOP)
121         self.barra = Scale(self, from_=0, to=100, orient=
122         HORIZONTAL, tickinterval=50)
123         self.barra.set(50)
124         self.barra.pack(side=TOP)
125
126         btnIniciar = Button(self, text="Iniciar/Reiniciar",
127         command=self.iniciar)
128         btnIniciar.pack(side=TOP)
129
130         button1 = Button(self, text="Pausa/Reanudar", command=
131         self.empezar_dentener)
132         button1.pack(side = TOP)
133
134         self.colorBtn1 = Button(self, text="Selecciona el color
135         de unos", command=self.getColorUnos, bg=self.unos)
136         self.colorBtn1.pack(side = TOP)
137
138         self.colorBtn2 = Button(self, text="Selecciona el color
139         de ceros", command=self.getColorCeros, bg=self.ceros)
140         self.colorBtn2.pack(side=TOP)

```



```

136         btnSave = Button(self, text="Guardar", command=self.
guardar)
137         btnSave.pack(side=TOP)
138
139
140     def actualizar_color_matriz(self):
141         for i in range(self.tam):
142             for j in range(self.tam):
143                 if self.celulas[i][j] == 0:
144                     self.canvas.itemconfig(self.cuadritos[i][j],
fill=self.ceros)
145                 else:
146                     self.canvas.itemconfig(self.cuadritos[i][j],
fill=self.unos)
147
148         self.update_idletasks()
149
150     def getColorUnos(self):
151         color = askcolor()
152         if not color[1] == None:
153             self.unos = color[1]
154             self.colorBtn1.configure(bg=self.unos)
155             self.actualizar_color_matriz()
156
157
158     def getColorCeros(self):
159         color = askcolor()
160         if not color[1] == None:
161             self.ceros = color[1]
162             self.colorBtn2.configure(bg=self.ceros)
163             self.actualizar_color_matriz()
164
165     def guardar(self):
166         # np.savetxt("matriz.txt", self.celulas, fmt="%d")
167         archivo = open("matriz.txt", 'a')
168         archivo.write("tiempo={}\n".format(self.tiempo))
169         for i in range(self.tam):
170             for j in range(self.tam):
171                 archivo.write("{} ".format(self.celulas[i, j]))
172             archivo.write("\n")
173
174         archivo.write("\n")
175         archivo.close()
176
177
178     def cargar(self):
179         self.celulas = np.loadtxt("prueba.txt", dtype=int)
180         self.canvas.delete('all')
181         self.tam = self.celulas.shape[0]

```

```

182         #self.celulas = np.random.randint(2, size=(self.tam,
self.tam), dtype=int)
183         self.cuadritos = np.zeros(shape=(self.tam, self.tam),
dtype=int)
184         print(self.celulas)
185         print(self.tam)
186         self.canvas.configure(width=self.tam, height=self.tam)
187         # self.contar_unos()
188         self.re_dibujar()
189         self.update_idletasks()
190         self.update()
191
192
193     def empezar_dentener(self):
194         print("empezar_detener")
195         self.pausa = not self.pausa
196         self.animacion()
197
198     def animacion(self):
199         if not self.pausa:
200             self.historia_y.append(self.contador)
201             self.historia_x.append(self.tiempo)
202             archivo = open("grafica.txt", "a")
203             archivo.write("{}{}\n".format(self.tiempo, self.
contador))
204             archivo.close()
205             nueva_poblacion = self.celulas.copy()
206             for i in range(self.tam):
207                 print(i)
208                 for j in range(self.tam):
209                     vecinos = (self.celulas[i - 1, j - 1] + self
.celulas[i - 1, j] + self.celulas[i - 1, (j + 1) % self.tam]
210                             + self.celulas[i, (j + 1) % self.
tam] + self.celulas[(i + 1) % self.tam, (j + 1) % self.tam]
211                             + self.celulas[(i + 1) % self.tam
, j] + self.celulas[(i + 1) % self.tam, j - 1] + self.celulas
[i, j - 1])
212                     if self.celulas[i, j] == 1:
213                         if vecinos < self.regla[0] or vecinos >
self.regla[1]:
214                             nueva_poblacion[i, j] = 0
215                             self.canvas.itemconfig(self.
cuadritos[i][j], fill=self.ceros)
216                             self.contador -= 1
217                         else:
218                             if vecinos >= self.regla[2] and vecinos
<= self.regla[3]:
219                                 nueva_poblacion[i, j] = 1

```

```

220         self.canvas.itemconfig(self.
cuadritos[i][j], fill=self.unos)
221         self.contador += 1
222
223         self.celulas[:] = nueva_poblacion[:]
224         self.update_idletasks()
225         print("Termino")
226         self.tiempo += 1
227         self.after(1000, self.animacion)
228
229 def main():
230     root = Tk()
231     root.geometry('1360x750+0+0')
232     app = Ventana(root)
233     app.mainloop()
234
235 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()

```

## 2.3. Pruebas

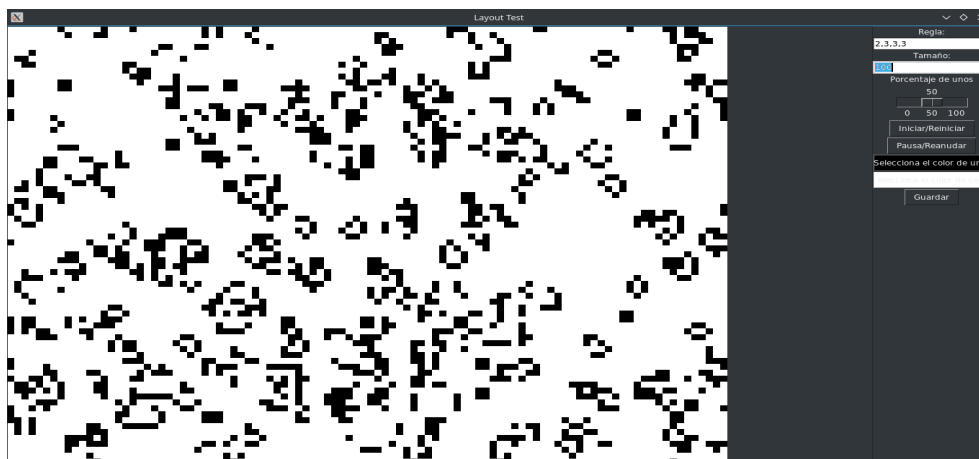


Figura 9: Juego de la vida con la regla 2333

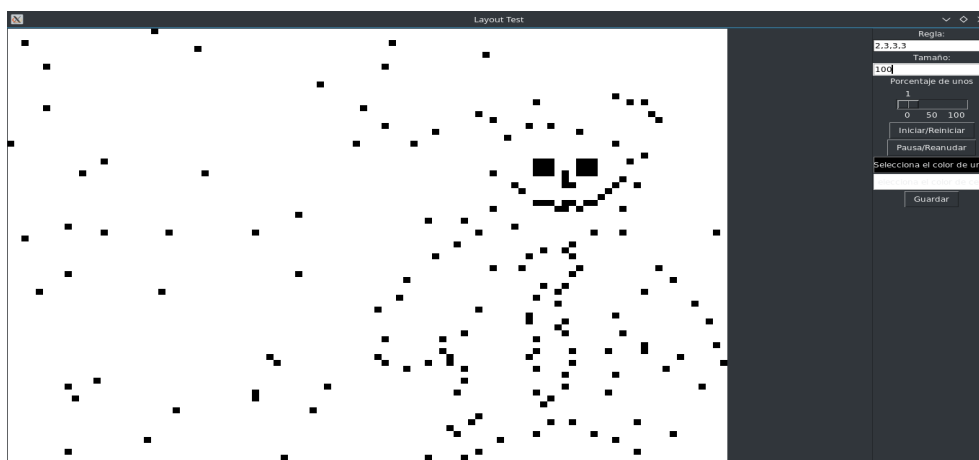


Figura 10: Juego de la vida con células seleccionadas por el usuario

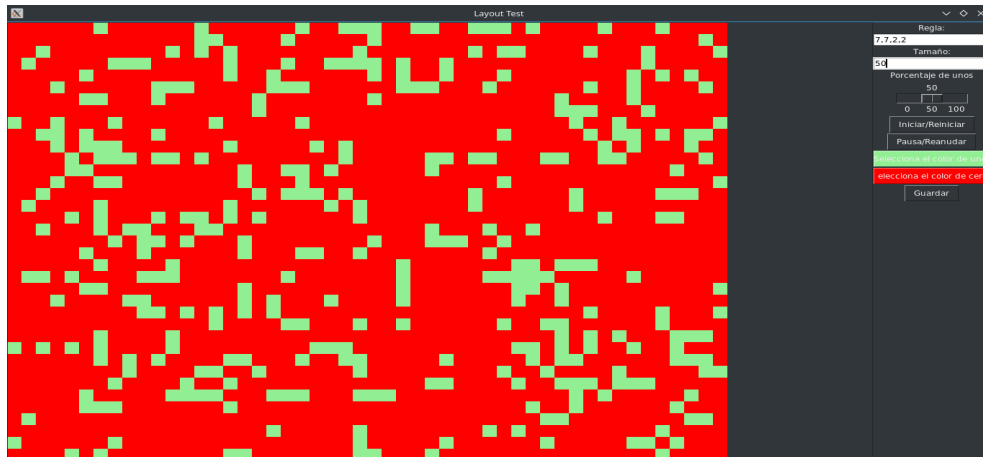
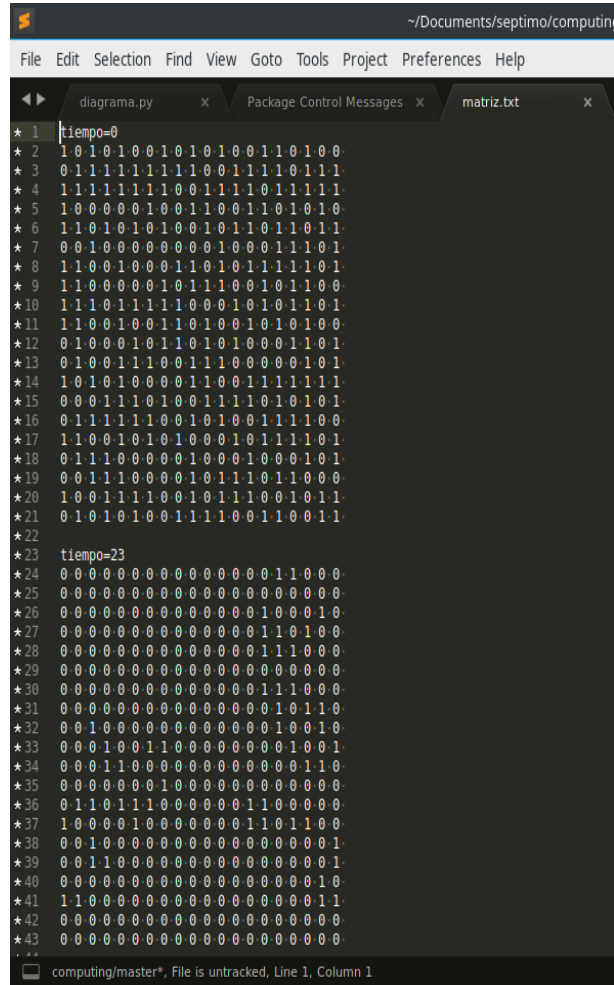


Figura 11: Juego de la vida cambiando los colores y la regla 7722



```

* 1 |tiempo=0
* 2 |1 0 1 0 1 0 0 1 0 1 0 1 0 0 1 1 0 1 0 0
* 3 |0 1 1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1
* 4 |1 1 1 1 1 1 1 1 0 0 1 1 1 1 0 1 1 1 1 1
* 5 |1 0 0 0 0 0 1 0 0 1 1 0 0 1 1 0 1 0 1 0
* 6 |1 1 0 1 0 1 0 1 0 0 1 0 1 1 0 1 1 0 1 1
* 7 |0 0 1 0 0 0 0 0 0 0 0 0 1 0 0 0 1 1 1 0 1
* 8 |1 1 0 0 1 0 0 0 1 1 0 1 0 1 1 1 1 1 1 0 1
* 9 |1 1 0 0 0 0 0 1 0 1 1 1 0 0 1 0 1 1 1 0 0
*10 |1 1 1 0 1 1 1 1 1 0 0 0 1 0 1 0 1 1 1 0 1
*11 |1 1 0 0 1 0 0 1 1 0 1 0 0 1 0 1 0 1 0 1 0
*12 |0 1 0 0 0 1 0 1 1 0 1 0 1 0 0 0 1 1 0 1
*13 |0 1 0 0 1 1 1 0 0 1 1 1 1 0 0 0 0 0 1 0 1
*14 |1 0 1 0 1 0 0 0 0 1 1 0 0 1 1 1 1 1 1 1 1
*15 |0 0 0 1 1 1 0 1 0 0 0 1 1 1 1 0 1 0 1 0 1
*16 |0 1 1 1 1 1 1 0 0 1 0 1 0 0 1 1 1 1 1 0 0
*17 |1 1 0 0 1 0 1 0 1 0 0 0 1 0 1 1 1 1 1 0 1
*18 |0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 0 0 1 0 1
*19 |0 0 1 1 1 0 0 0 0 1 0 1 1 1 0 1 1 0 0 0
*20 |1 0 0 1 1 1 1 0 0 1 0 1 1 1 0 0 1 0 1 1
*21 |0 1 0 1 0 1 0 0 1 1 1 1 0 0 1 1 0 0 1 1
*22 |
*23 |tiempo=23
*24 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 0 0
*25 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*26 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 0 1 0
*27 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0 1 0 0
*28 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
*29 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*30 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1 1 0 0 0
*31 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0 1 1 0
*32 |0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 1 0 0 1 0
*33 |0 0 0 1 0 0 1 1 0 0 0 0 0 0 0 1 0 0 1
*34 |0 0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 1 1 0
*35 |0 0 0 0 0 0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0
*36 |0 1 1 0 1 1 1 0 0 0 0 0 0 1 1 0 0 0 0 0 0
*37 |1 0 0 0 0 1 0 0 0 0 0 0 0 1 1 0 1 1 0 0
*38 |0 0 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
*39 |0 0 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1
*40 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 0
*41 |1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 1 1
*42 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
*43 |0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0

```

Figura 12: Matriz que se guarda en tiempo=0 y tiempo=23

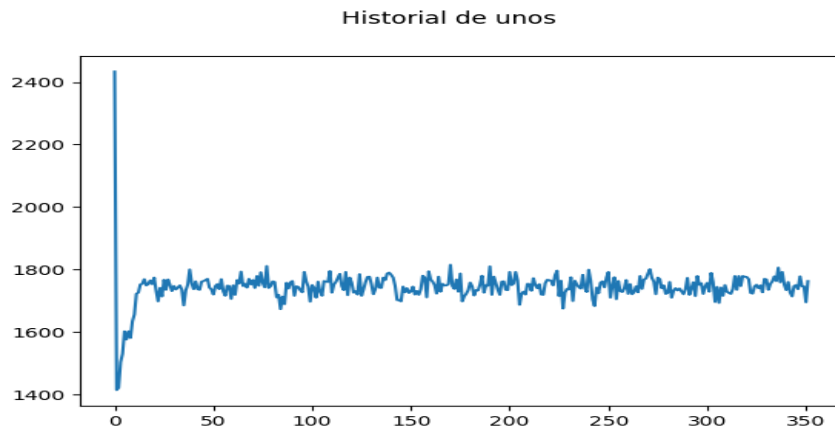


Figura 13: Gráfica que muestra el histórico de unos con la regla 7722

## 2.4. 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.

## Referencias

- [1] J. E. Hopcroft, R. Motwani, and J. D. Ullman, *Introducción a La Teoría De Autómatas, Lenguajes Y Computación*. Addison-Wesley, 2007.