



# **Instituto Politécnico Nacional**

## **Escuela Superior de Cómputo**

**Alumno: Hernández Radilla José  
Ángel**

**Materia: Computing selected topics**

**Profesor: Juárez Martínez Genaro**

**Grupo: 3CM19**

**28-Marzo-2021**

# 1. Funciones del archivo automata.py

## 1.0.1. Función reglas

Esta función se usa para evaluar el valor de las células en el instate de tiempo  $t+1$ . La función recibe como parámetros  $r$ , que es un arreglo de 8 elementos con el valor en binario de la regla que se quiere evaluar, los otros 3 parámetros son los valores de las células adyacentes en el instante de tiempo  $t$ . La función consta simplemente de muchas sentencias if para cada uno de los casos "000", "001", ..., "111". Finalmente la función regresa el valor de la célula correspondiente a la regla que se está utilizando.

```
def reglas(r, n1, n2, n3):
    if n1 == n2 == n3 == 1:
        return r[0]
    if n1 == n2 == 1 and n3 == 0:
        return r[1]
    if n1 == 1 and n2 == 0 and n3 == 1:
        return r[2]
    if n1 == 1 and n2 == 0 and n3 == 0:
        return r[3]
    if n1 == 0 and n2 == 1 and n3 == 1:
        return r[4]
    if n1 == 0 and n2 == 1 and n3 == 0:
        return r[5]
    if n1 == 0 and n2 == 0 and n3 == 1:
        return r[6]
    if n1 == 0 and n2 == 0 and n3 == 0:
        return r[7]
```

Figura 1: Función reglas.

## 1.0.2. Función llenar estado inicial

La función recibe cuatro argumentos los cuales son un estado inicial el cual es una cadena de cualquier tamaño que tiene un número en binario que será la primera configuración, los siguientes dos argumentos son el inicio y el fin, estos argumentos no son el tamaño de la cadena sino un rango en el cual la función recorre la cadena para ir colocando los 0's y 1's de la cadena en un arreglo que después será colocado en una matriz. Finalmente el argumento op representa la opción + o -, esta opción permite recorrer la cadena normalmente o empezando desde el fin de la misma, esto porque la función será utilizada por hilos los cuales recorrerán la cadena del inicio a la mitad y del fin a la mitad, para hacer ligeramente más rápido este proceso de llenado de la matriz que posteriormente se graficará. La función regresa un arreglo que corresponde simplemente a la conversión de la cadena a un arreglo de números enteros.

```

def llenar_estado_inicial(inicial, inicio, fin, op):
    arr = []
    inc = 0
    if op == "+":
        for i in range(inicio, fin):
            arr.append(int(inicial[i]))
    else:
        for i in range(fin, inicio):
            arr.append(int(inicial[i]))
    return arr

```

Figura 2: Función llenar estado inicial.

### 1.0.3. Función calcular iteración

Esta función se encarga de calcular el tiempo  $t+i$  de las células, las células son acomodadas en un arreglo que será devuelto por la función. La función recibe 6 argumentos, el primero corresponde a la regla que será utilizada para calcular las iteraciones que corresponde a un arreglo de números enteros, este arreglo es la representación en binario de la regla que se usará. El siguiente argumento es  $i$  que corresponde con la iteración  $i$ -ésima que se quiere calcular, debe de ser un número entero. El tercer argumento corresponde a la matriz que contiene a las iteraciones previamente hechas, recordemos que para calcular la iteración  $t+i$  se tiene que haber calculado previamente la iteración  $t+i-1$ . Los siguientes dos argumentos corresponden al inicio y al fin de la iteración anterior, recordemos que la función funciona con hilos un hilo se encarga de recorrer la mitad de la iteración anterior para calcular la mitad de la nueva iteración y el otro hilo se encarga de recorrer la otra mitad. El argumento  $op$  se utiliza para saber que mitad recorrerá el hilo si la primera mitad o la segunda. Dentro de la función simplemente se crea un arreglo que corresponde a la iteración que se calculará se pregunta por medio de la sentencia `if` que parte de la iteración anterior se recorrerá, finalmente dentro del recorrido se agrega a el arreglo el resultado de evaluar la función `reglas`, recordemos que la función `reglas` devolverá el valor de una célula dados los valores de las tres células (izquierda, central y derecha) de la iteración anterior, se debe tener en cuenta que la célula cero y la última tienen el problema de que no existen células izquierdas o derechas por lo que se ponen sentencias `if` para preguntar por estos casos y asignarles el principio de la célula o final de la célula según sea el caso.

```

def calcular_iteracion(r, i, m, inicio, fin, op):
    arr = []
    if op == "+":
        for j in range(inicio, fin):
            if j == 0:
                arr.append(reglas(r, m[i][len(m[i])-1], m[i][j], m[i][j+1]))
            elif j == len(m[i])-1:
                arr.append(reglas(r, m[i][j-1], m[i][j], m[i][0]))
            else:
                arr.append(reglas(r, m[i][j-1], m[i][j], m[i][j+1]))
    else:
        for j in range(fin, inicio):
            if j == 0:
                arr.append(reglas(r, m[i][len(m[i])-1], m[i][j], m[i][j+1]))
            elif j == len(m[i])-1:
                arr.append(reglas(r, m[i][j-1], m[i][j], m[i][0]))
            else:
                arr.append(reglas(r, m[i][j-1], m[i][j], m[i][j+1]))
    return arr

```

Figura 3: Función calcular iteración.

#### 1.0.4. Función crear matriz hilos

En esta función se crea la matriz que tiene todas la iteraciones y regresa la matriz. Recibe 3 argumentos el primero corresponde con la regla que se utilizará que es una arreglo de enteros que representa el número en binario de la regla correspondiente. El siguiente argumento es el número de iteraciones, debe de ser un número entero, finalmente la condición inicial que es una cadena de cualquier tamaño. La función crear la matriz, después procede a crear dos hilos que copiaran la confición inicial dentro de la matriz. Finalmente se utiliza un ciclo for para calcular todas la iteraciones, se crearán dos hilos para calcular la mitad de la iteración correspondiente, esas mitades son arreglos que se unirás y se pondrán en la matriz.

```
def crear_matriz_hilos(r, iteraciones, inicial):
    matriz = []
    fin = 0
    if len(inicial)%2 == 0:
        fin = (len(inicial)//2)-1
    else:
        fin = len(inicial)//2
    h1 = Hilo(target=llenar_estado_inicial, args=(inicial, 0,fin+1,"+"))
    h2 = Hilo(target=llenar_estado_inicial, args=(inicial, len(inicial),fin+1,"-"))
    h1.start()
    h2.start()
    arr1 = h1.join()
    arr2 = h2.join()
    matriz.append(arr1+arr2)

    for i in range(0,iteraciones):
        h1 = Hilo(target=calcular_iteracion, args=(r, i, matriz,0,fin+1,"+"))
        h2 = Hilo(target=calcular_iteracion, args=(r, i, matriz,len(inicial),fin+1,"-"))
        h1.start()
        h2.start()
        arr1 = h1.join()
        arr2 = h2.join()
        matriz.append(arr1+arr2)
    return matriz
```

Figura 4: Función crear matriz hilos.

## 2. Funciones del archivo archivo.py

### 2.0.1. Función cargar archivo

Esta función recibe la ruta de un archivo, lo abre en modo de lectura para extraer una configuración que estará dada por la regla que se quiere graficar, el número de iteraciones y la configuración inicial. La función lee la primera linea que representa el número de iteraciones y lo convierte a entero, después lee la segunda linea que representa la regla, la regla puede estar escrita en binario o como número decimal, en el caso de ser binario debe de empezar con la letra b, en este caso simplemente cada bit se convertirá a entero y se guardará en un arreglo; En el caso de ser decimal el número se convertirá a binario y posteriormente cada bit se guardará en un arreglo. Finalmente se lee otra linea del archivo que representa la configuración inicial y se regresan el número de iteraciones, el arreglo que representa a la regla a utilizar y la configuración inicial.

```
def cargar_archivo(ruta):
    archivo = open(ruta, 'r')
    arr = []
    iteraciones = int(archivo.readline())
    regla = archivo.readline()
    if regla[0] == "b":
        for i in range(1, len(regla)-1):
            arr.append(int(regla[i]))
    else:
        binario = binarizar(int(regla))
        for i in range(0, len(binario)):
            arr.append(int(binario[i]))
        if len(arr)<8:
            num = 8 - len(arr)
            for i in range(num):
                arr.insert(0,0)
    inicial = archivo.readline()
    archivo.close()
    return iteraciones, arr, inicial
```

Figura 5: Función cargar archivo.

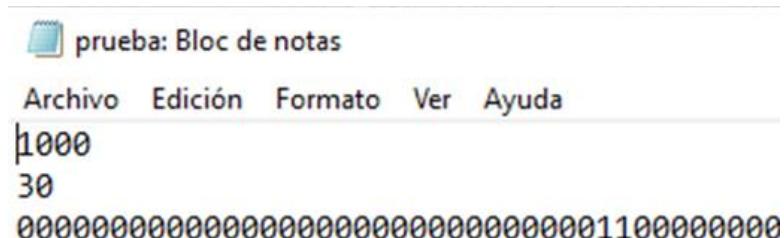


Figura 6: Ejemplo de archivo.

### 2.0.2. Función guardar archivo

La función recibe un objeto de tipo File, el número de iteraciones, la regla que se usará como un arreglo que representa el binario de número de la regla, y una cadena con la configuración inicial. La función simplemente escribe en el archivo cada uno de los parámetros que le fueron dados.

```
def guardar_archivo(archivo, iteraciones, r, inicial):
    archivo.write(str(iteraciones)+"\n")
    reglas = "b"
    for i in r:
        reglas += str(i)
    archivo.write(reglas+"\n")
    archivo.write(inicial)
    archivo.close()
```

Figura 7: Función guardar archivo.

### 3. Métodos más importantes de la clase Ventana del archivo interfaz.py

#### 3.0.1. Clase Ventana y método init

Esta clase es toda la interfaz del programa, en esta encontraremos una serie de métodos que mandan a llamar a las funciones de los archivos automata.py y archivo.py. La clase Ventana posee unos atributos que son las configuraciones de los colores, así como datos correspondientes al autómata, como las iteraciones, la regla que está usando, la configuración inicial, y la matriz.

```
class Ventana():
    def __init__(self):
        self.ventana = Tk()
        self.color_uno = "#000000"
        self.color_cero = "#ffffff"
        self.menu = Menu(self.ventana)
        self.fig = Figure(figsize = (8, 8), dpi = 100)
        self.canvas = FigureCanvasTkAgg(self.fig,master = self.ventana)

        #datos
        self.iteraciones = 0
        self.r = []
        self.inicial = "0"
        self.num_unos = []
        self.matriz = []
```

Figura 8: Método init de la clase ventana.

#### 3.0.2. Método aleatorio

Este método se utiliza para crear una configuración aleatoria, el método recibe una ventana (desde la cual es llamado el método) para cerrarla, un número de iteraciones y un número de células. Lo primero que hace el método es crear una regla aleatoria para ello crea un arreglo de tamaño 8 y a cada elemento le asigna aleatoriamente un 0 o un 1. Después se crea una configuración inicial del tamaño que se especifica, esta configuración es una cadena de 0's y 1's escogidos al azar. Se crea una matriz (con la función crear matriz hilos) y se pinta la matriz. Un detalle importante es que la clase ventana guarda la matriz, la configuración inicial y la regla por si se quiere editar algo.

```

def aleatorio(self,ventana,i,c):
    regla = []
    inicial = ""
    for j in range(8):
        regla.append(random.randint(0,1))
    for j in range(int(c.get())):
        if random.randint(0,1):
            inicial+="1"
        else:
            inicial+="0"
    self.matriz = crear_matriz_hilos(regla,int(i.get()),inicial)
    self.pintar(self.matriz)
    self.iteraciones = int(i.get())
    self.r = regla
    self.inicial = inicial
    ventana.destroy()

```

Figura 9: Función aleatorio.

### 3.0.3. Método op cantidad 1

En este Método se cuentan la cantidad de 1 que hay en la matriz y después se grafican, para ello se recorre la matriz por filas (que representa la iteración i), dentro de este recorrido se recorre para ir contando los 1 que existen dentro de una iteración, estos resultados se guardan en un arreglo, posteriormente este arreglo se grafica con el módulo matplotlib de python (método plot y show).

```

def op_cantidad_1(self):
    arr = []
    for i in range(len(self.matriz)):
        aux = 0
        for j in range(len(self.matriz[0])):
            if self.matriz[i][j]:
                aux += 1
        arr.append(aux)
    plt.title("Cantidad de 1's")
    plt.xlabel("iteracion")
    plt.ylabel("cantidad")
    plt.plot(arr)
    plt.show()

```

Figura 10: Método op cantidad 1.

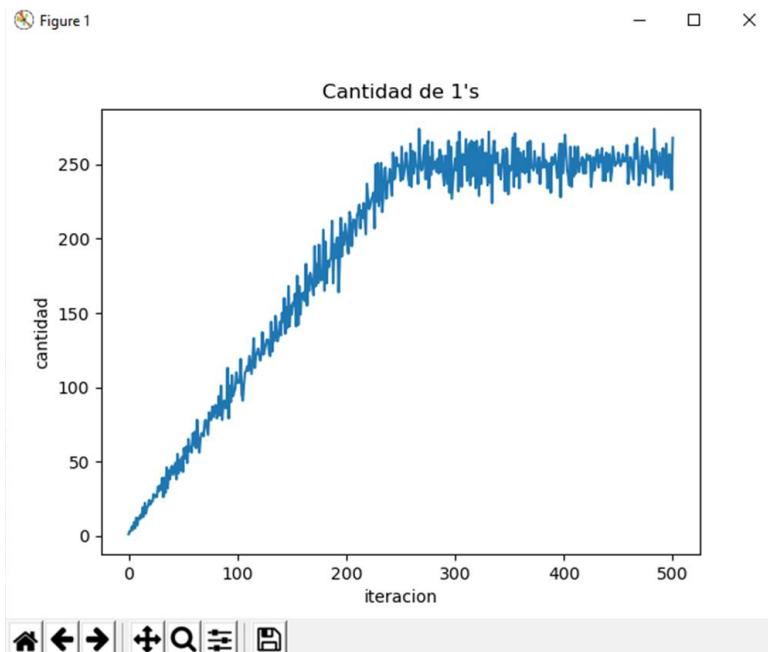


Figura 11: Grafica generada por el programa.

### 3.0.4. Método cambiar color uno/cero

En estos métodos se cambian los colores del uno o del cero, recordando que la clase ventana tiene un atributo para el color del cero y otro para el del uno originalmente puestos a ffffff y 000000 respectivamente. En este método simplemente utilizamos la función askcolor, que nos genera un colorchooser, y obtenemos el valor uno, que es el valor hexadecimal del color y cambiamos el de la clase.

```

def cambiar_color_uno(self):
    self.color_uno = askcolor()[1]
    self.pintar(self.matriz)

```

Figura 12: Método para cambiar el color de los 1's.

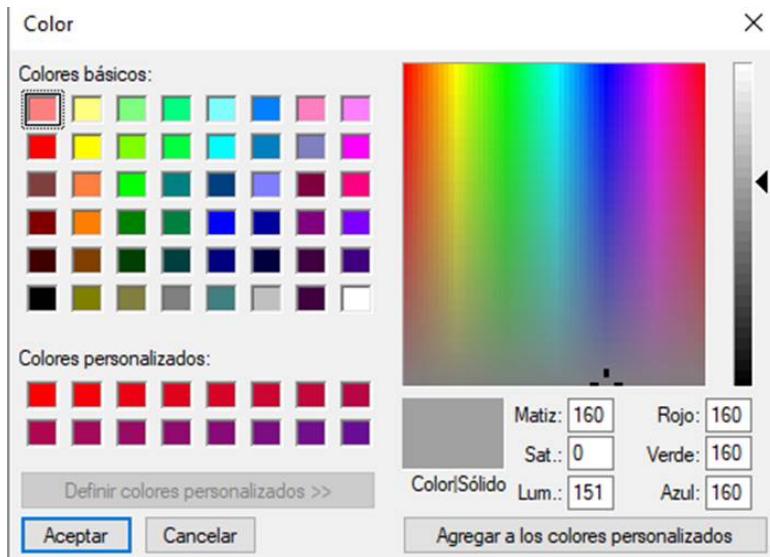


Figura 13: Color chooser.

### 3.0.5. Método pintar

En este Método se pinta la matriz que contiene la celulas, para ello se utiliza el método imshow correspondiente a matplotlib, a este método se le pueden agregar colores por medio de un gradiente que puede definirse con un ColorMap, en este caso la clase ventana tiene unos colores definidos que pueden ser cambiados.

```

def pintar(self, matriz):
    cmap = LinearSegmentedColormap.from_list('mycmap', [self.color_cero, self.color_uno])
    grafica = self.fig.add_subplot(111)
    grafica.imshow(self.matriz, cmap = cmap)
    self.canvas.draw()
    self.canvas.get_tk_widget().pack()

```

Figura 14: Método op cantidad 1.

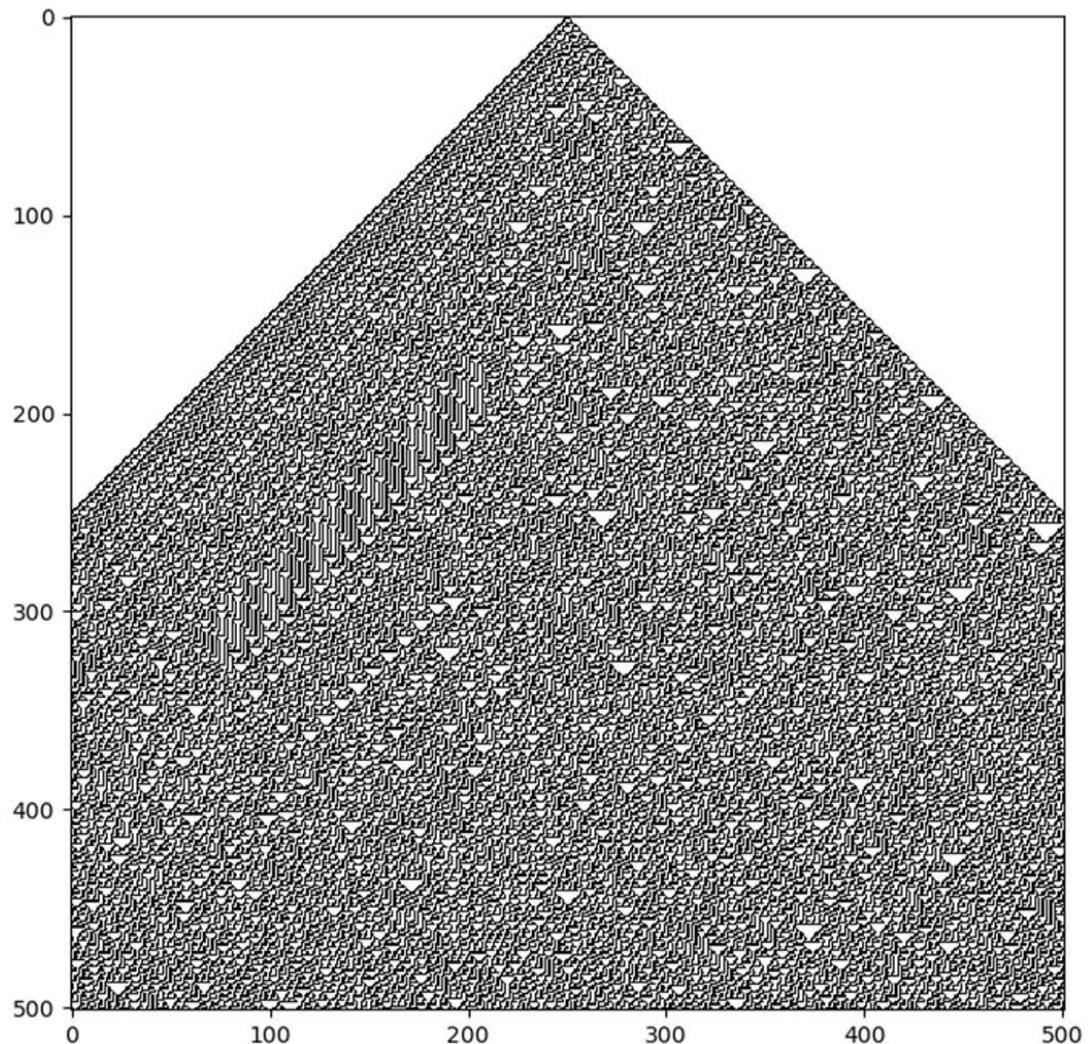


Figura 15: Ejemplo de matriz pintada de 1000 iteraciones y 1000 células.

## 4. Pruebas de reglas

## 5. Código

```
from threading import Thread

class Hilo(Thread):
    def __init__(self, group=None, target=None, name=None,
```

```

        args=(), kwargs={}, Verbose=None):
    Thread.__init__(self, group, target, name, args, kwargs)
    self._return = None
def run(self):
    if self._target is not None:
        self._return = self._target(*self._args,
                                    **self._kwargs)
    return

def join(self, *args):
    Thread.join(self, *args)
    return self._return

def reglas(r, n1, n2, n3):
    if n1 == n2 == n3 == 1:
        return r[0]
    if n1 == n2 == 1 and n3 == 0:
        return r[1]
    if n1 == 1 and n2 == 0 and n3 == 1:
        return r[2]
    if n1 == 1 and n2 == 0 and n3 == 0:
        return r[3]
    if n1 == 0 and n2 == 1 and n3 == 1:
        return r[4]
    if n1 == 0 and n2 == 1 and n3 == 0:
        return r[5]
    if n1 == 0 and n2 == 0 and n3 == 1:
        return r[6]
    if n1 == 0 and n2 == 0 and n3 == 0:
        return r[7]

def llenar_estado_inicial(inicial, inicio, fin, op):
    arr = []
    inc = 0
    if op == "+":
        for i in range(inicio, fin):
            arr.append(int(initial[i]))
    else:
        for i in range(fin, inicio):
            arr.append(int(initial[i]))
    return arr

def calcular_iteracion(r, i, m, inicio, fin, op):
    arr = []
    if op == "+":
        for j in range(inicio, fin):
            if j == 0:
                arr.append(reglas(r, m[i][len(m[i])-1], m[i][j], m[i][j+1]))
            elif j == len(m[i])-1:

```

```

        arr.append( reglas(r,m[i][j-1], m[i][j], m[i][0]) )
    else:
        arr.append( reglas(r,m[i][j-1], m[i][j], m[i][j+1]) )
else:
    for j in range(fin, inicio):
        if j == 0:
            arr.append( reglas(r,m[i][len(m[i])-1], m[i][j], m[i][j+1]) )
        elif j == len(m[i])-1:
            arr.append( reglas(r,m[i][j-1], m[i][j], m[i][0]) )
        else:
            arr.append( reglas(r,m[i][j-1], m[i][j], m[i][j+1]) )
return arr

def crear_matriz_hilos(r, iteraciones, inicial):
    matriz = []
    fin = 0
    if len(inicial)%2 == 0:
        fin = (len(inicial)//2)-1
    else:
        fin = len(inicial)//2
    h1 = Hilo(target=llenar_estado_inicial, args=(inicial, 0,fin+1,"+"))
    h2 = Hilo(target=llenar_estado_inicial, args=(inicial, len(inicial),fin+1,"+"))
    h1.start()
    h2.start()
    arr1 = h1.join()
    arr2 = h2.join()
    matriz.append(arr1+arr2)

    for i in range(0,iteraciones):
        h1 = Hilo(target=calcular_iteracion, args=(r, i, matriz,0,fin+1,"+"))
        h2 = Hilo(target=calcular_iteracion, args=(r, i, matriz,len(inicial),fin+1,"+"))
        h1.start()
        h2.start()
        arr1 = h1.join()
        arr2 = h2.join()
        matriz.append(arr1+arr2)
    return matriz

from automata import reglas

def binarizar(decimal):
    binario = ''
    while decimal // 2 != 0:
        binario = str(decimal % 2) + binario
        decimal = decimal // 2
    return str(decimal) + binario

def cargar_archivo(ruta):

```

```

archivo = open(ruta , 'r')
arr = []
iteraciones = int(archivo.readline())
regla = archivo.readline()
if regla[0] == "b":
    for i in range(1, len(regla)-1):
        arr.append(int(regla[i]))
else:
    binario = binarizar(int(regla))
    for i in range(0, len(binario)):
        arr.append(int(binario[i]))
    if len(arr)<8:
        num = 8 - len(arr)
        for i in range(num):
            arr.insert(0,0)
inicial = archivo.readline()
archivo.close()
return iteraciones , arr , inicial

def guardar_archivo(archivo , iteraciones , r , inicial ):
    archivo.write(str(iteraciones)+"\n")
    reglas = "b"
    for i in r:
        reglas += str(i)
    archivo.write(reglas+"\n")
    archivo.write(inicial)
    archivo.close()

from tkinter import Tk,Menu,ALL,Label,Entry,Button
from automata import crear_matriz_hilos
from tkinter.filedialog import askopenfilename,asksaveasfile
from archivo import cargar_archivo,guardar_archivo,binarizar
import matplotlib.pyplot as plt
from matplotlib.backends.backend_tkagg import (FigureCanvasTkAgg,NavigationTo
from matplotlib.figure import Figure
from functools import partial
import random
from matplotlib.colors import LinearSegmentedColormap
from tkinter.colorchooser import askcolor

class Ventana():
    def __init__(self):
        self.ventana = Tk()
        self.color_uno = "#000000"
        self.color_cero = "#ffffff"
        self.menu = Menu(self.ventana)
        self.fig = Figure(figsize = (8, 8), dpi = 100)
        self.canvas = FigureCanvasTkAgg(self.fig,master = self.ventana)

        #datos
        self.iteraciones = 0

```

```

self.r = []
self.inicial = "0"
self.num_unos = []
self.matriz = []

#colocando menus
self.ventana.config(menu=self.menu)
#archivos
self.op = Menu(self.menu, tearoff=0)
self.op.add_command(label="Cargar_archivo", command=self.cargar_archivo)
self.op.add_command(label="Guardar_archivo", command=self.guardar_archivo)
self.menu.add_cascade(label="Archivo", menu=self.op)
#graficar cantidad 1
self.op1 = Menu(self.menu, tearoff=0)
self.op1.add_command(label="Graficar_cantidad_de_1's", command=self.graficar_cantidad_de_1s)
self.op1.add_command(label="Graficar_cantidad_de_0's", command=self.graficar_cantidad_de_0s)
self.menu.add_cascade(label="Graficar", menu=self.op1)
#graficar aleatorio
self.op2 = Menu(self.menu, tearoff=0)
self.op2.add_command(label="Crear_grafica_aleatoria", command=self.ver_grafica_aleatoria)
self.menu.add_cascade(label="Aleatorio", menu=self.op2)
#colores
self.op3 = Menu(self.menu, tearoff=0)
self.op3.add_command(label="Color_de_0", command=self.cambiar_color_cero)
self.op3.add_command(label="Color_de_1", command=self.cambiar_color_uno)
self.menu.add_cascade(label="Colores", menu=self.op3)

#zoom y opciones de la grafica
self.toolbar = NavigationToolbar2Tk(self.canvas, self.ventana)
self.toolbar.update()
self.canvas.get_tk_widget().pack()

def cambiar_color_cero(self):
    self.color_cero = askcolor()[1]
    self.pintar(self.matriz)

def cambiar_color_uno(self):
    self.color_uno = askcolor()[1]
    self.pintar(self.matriz)

def ventana_aleatorio(self):
    ventana = Tk()
    Label(ventana, text='Numero_de_iteraciones').pack()
    i = Entry(ventana)
    i.pack()
    Label(ventana, text='Cantidad_de_celulas').pack()
    c = Entry(ventana)
    c.pack()
    Label(ventana, text='Regla_(campo_vacio_=aleatoriamente)').pack()
    er = Entry(ventana)

```

```

er . pack ()
Label(ventana , text='Probabilidad _de _unos (campo_vacio _=_50 %) ' ). pack ()
p1 = Entry(ventana)
p1 . pack ()
Button(ventana , text='Aceptar ' , command=partial( self . aleatorio , ventana))
ventana . title ( 'Automata _celular ')
ventana . geometry ("250x200")
ventana . eval( 'tk :: PlaceWindow %_center ' % ventana . winfo_pathname( ventana ))
ventana . resizable( False , False )

def aleatorio( self , ventana , i , c , er , p1):
    regla = []
    inicial = ""
    if er . get () == "" :
        for j in range(8):
            regla . append( random . randint(0 , 1))
    else:
        if er . get ()[0] == "b" :
            for j in range(1 , len(er . get ())):
                regla . append( int(er . get ()[ j ]))
            if len( regla ) < 8:
                for j in range(0,8-len( regla )):
                    regla . insert(0 , 0)
        else:
            binario = binarizar( int(er . get ()))
            for j in binario:
                regla . append( int(j ))
            if len( regla ) < 8:
                for j in range(0,8-len( regla )):
                    regla . insert(0 , 0)
    for j in range( int(c . get ())):
        if p1 . get () == "":
            if random . randint(0 , 1):
                inicial+="1"
            else:
                inicial+="0"
        else:
            if random . randint(0 , 100) < int(p1 . get ()):
                inicial+="1"
            else:
                inicial+="0"
    self . matriz = crear_matriz_hilos( regla , int(i . get ()) , inicial )
    self . pintar( self . matriz )
    self . iteraciones = int(i . get ())
    self . r = regla
    self . inicial = inicial
    ventana . destroy ()

def op_cantidad_0( self ):
    arr = []

```

```

for i in range(len(self.matriz)):
    aux = 0
    for j in range(len(self.matriz[0])):
        if not self.matriz[i][j]:
            aux += 1
    arr.append(aux)
plt.title("Cantidad_de_0's")
plt.xlabel("iteracion")
plt.ylabel("cantidad")
plt.plot(arr)
plt.show()

def op_cantidad_1(self):
    arr = []
    for i in range(len(self.matriz)):
        aux = 0
        for j in range(len(self.matriz[0])):
            if self.matriz[i][j]:
                aux += 1
        arr.append(aux)
    plt.title("Cantidad_de_1's")
    plt.xlabel("iteracion")
    plt.ylabel("cantidad")
    plt.plot(arr)
    plt.show()

def cargar_archivo(self):
    archivo = askopenfilename()
    especificaciones=cargar_archivo(archivo)
    self.matriz = crear_matriz_hilos(especificaciones[1], especificaciones)
    self.pintar(self.matriz)
    self.iteraciones = especificaciones[0]
    self.r = especificaciones[1]
    self.inicial = especificaciones[2]

def guardar_archivo(self):
    archivo = asksaveasfile(mode='w', defaultextension=".txt")
    if archivo is None: #regresa None si se ha cancelado
        return
    guardar_archivo(archivo, self.iteraciones, self.r, self.inicial)

def pintar(self, matriz):
    cmap = LinearSegmentedColormap.from_list('mycmap', [self.color_cero,
    grafica = self.fig.add_subplot(111)
    grafica.imshow(self.matriz, cmap = cmap)
    self.canvas.draw()
    self.canvas.get_tk_widget().pack()

def mostrar(self):
    self.ventana.title('Automata_celular')

```

```
self.ventana.configure(width=800,height=800)
self.ventana.eval('tk::PlaceWindow %s:center' % self.ventana.winfo_pa
self.ventana.resizable(False, False)
self.ventana.mainloop()

v = Ventana()
v.mostrar()
```

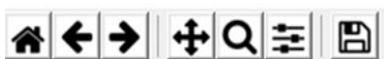
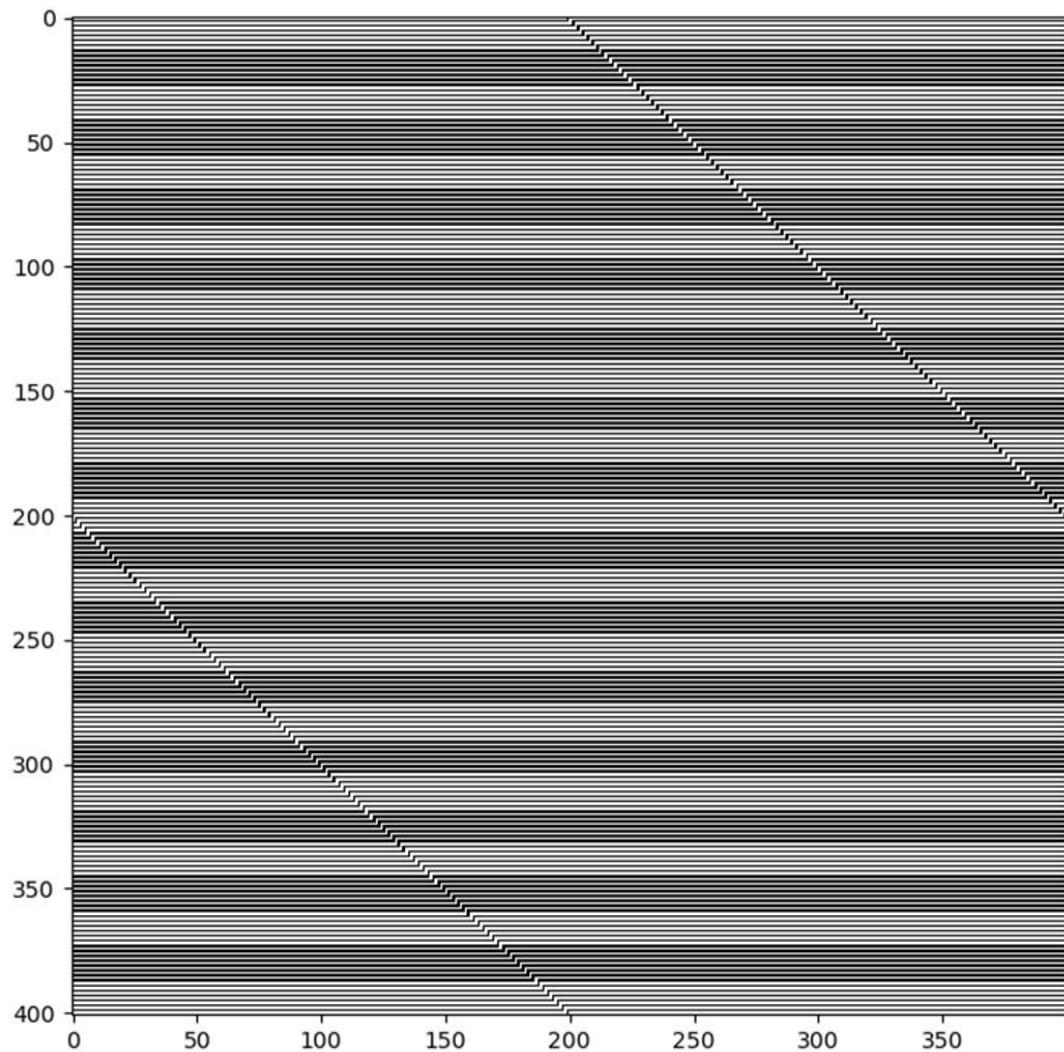
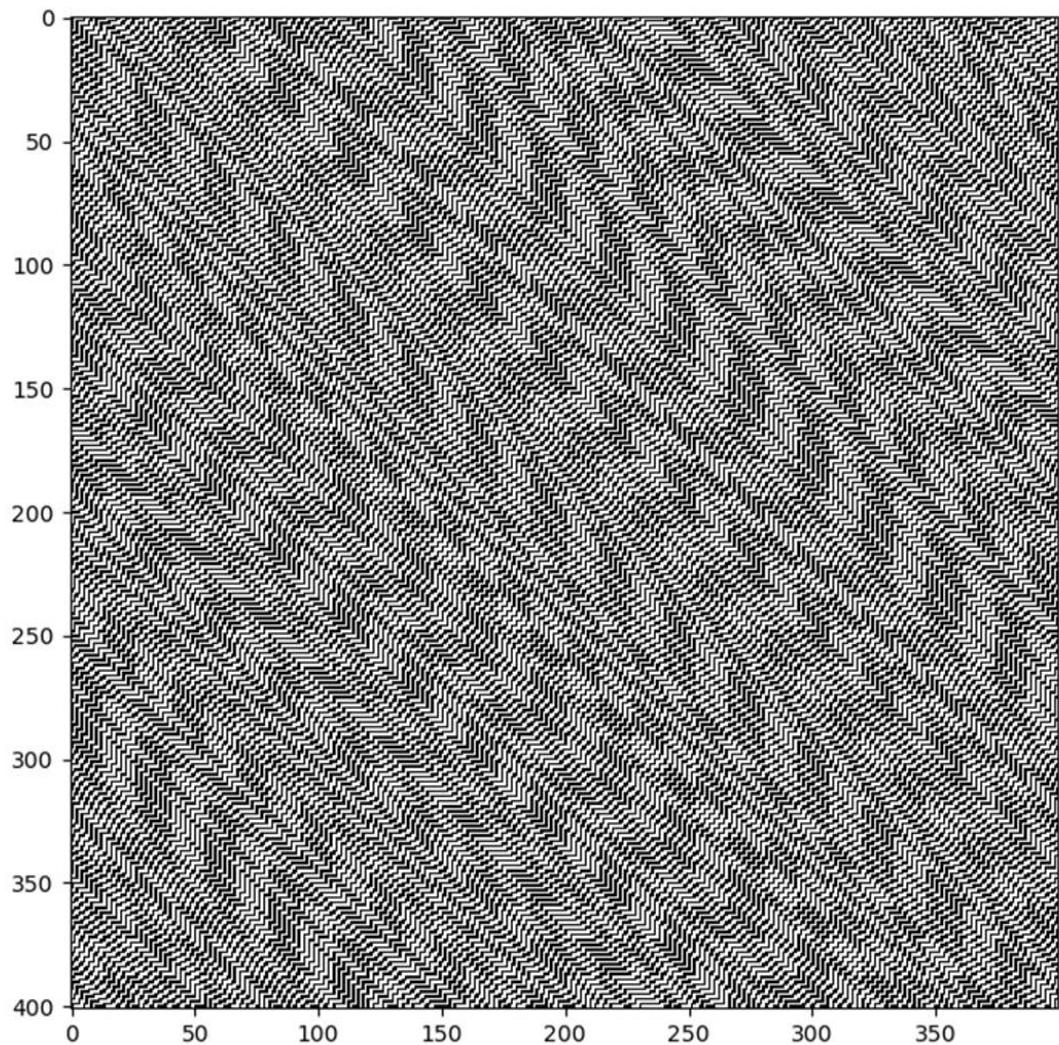


Figura 16: Regla 15 con un 1 en medio.

## Automata celular

Archivo Graficar Aleatorio Colores



x=151.979 y=86.7305 [1]

Figura 17: Regla 15 con 50 % de 1.

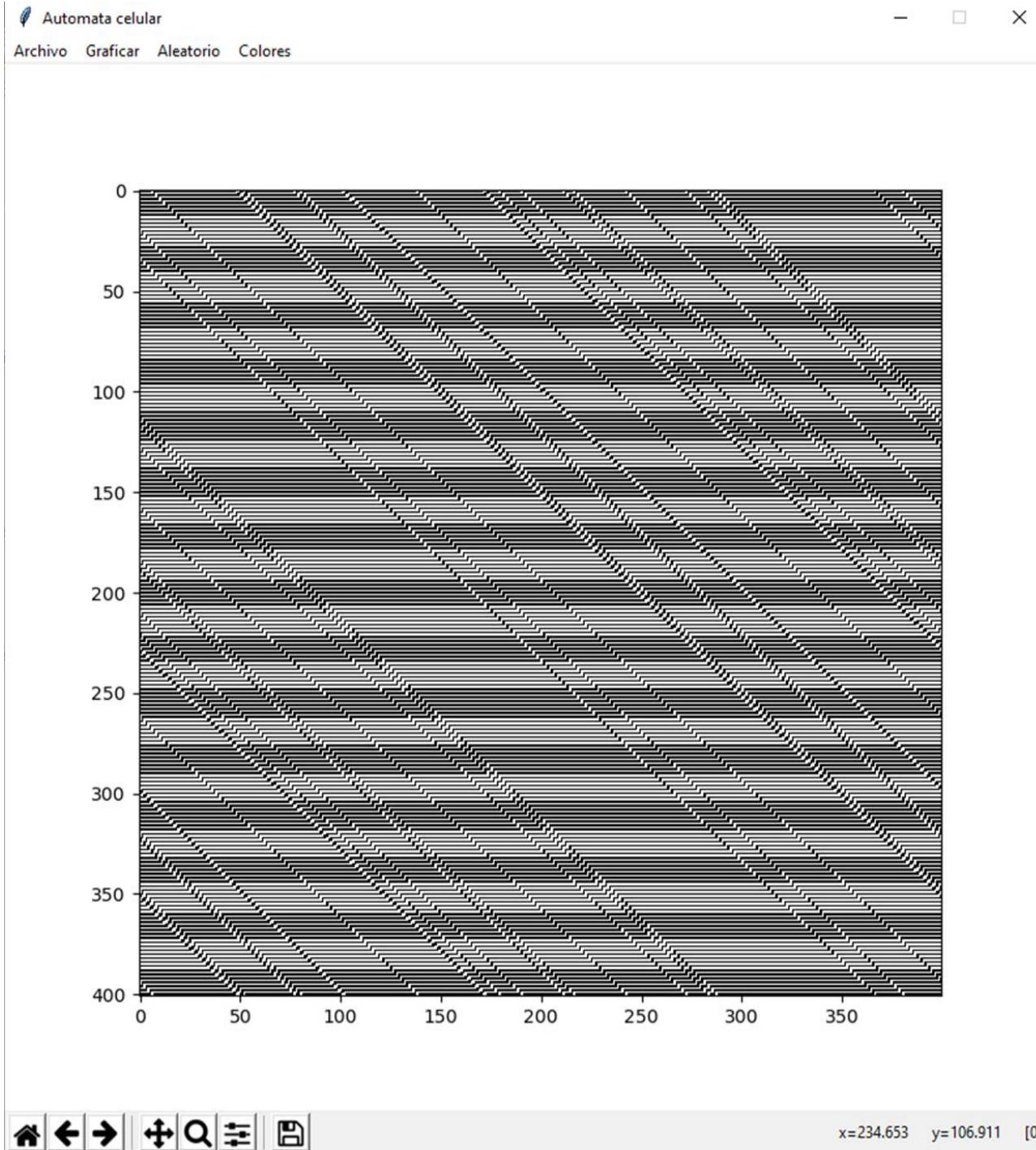
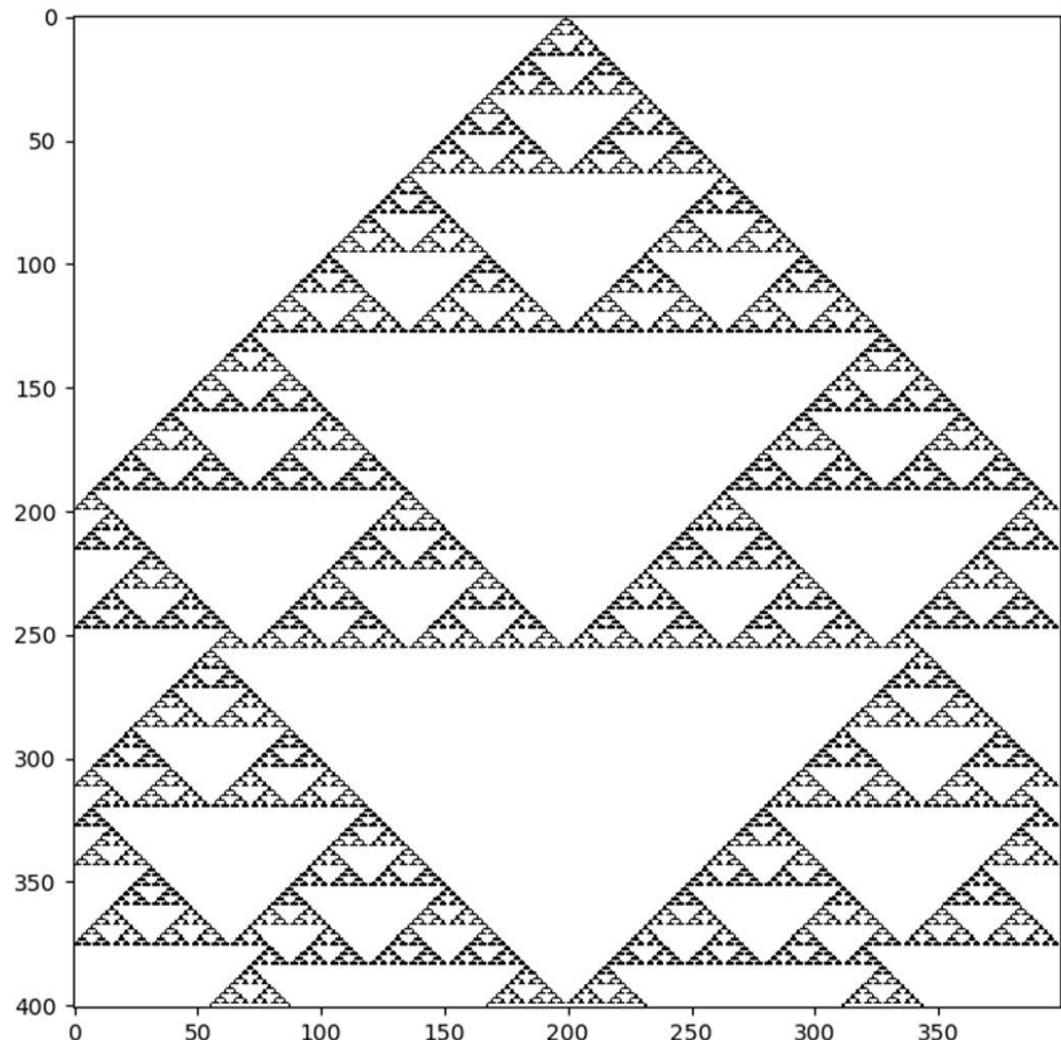


Figura 18: Regla 15 con 95 % de 1.



zoom rect

Figura 19: Regla 22 con un 1 en medio.

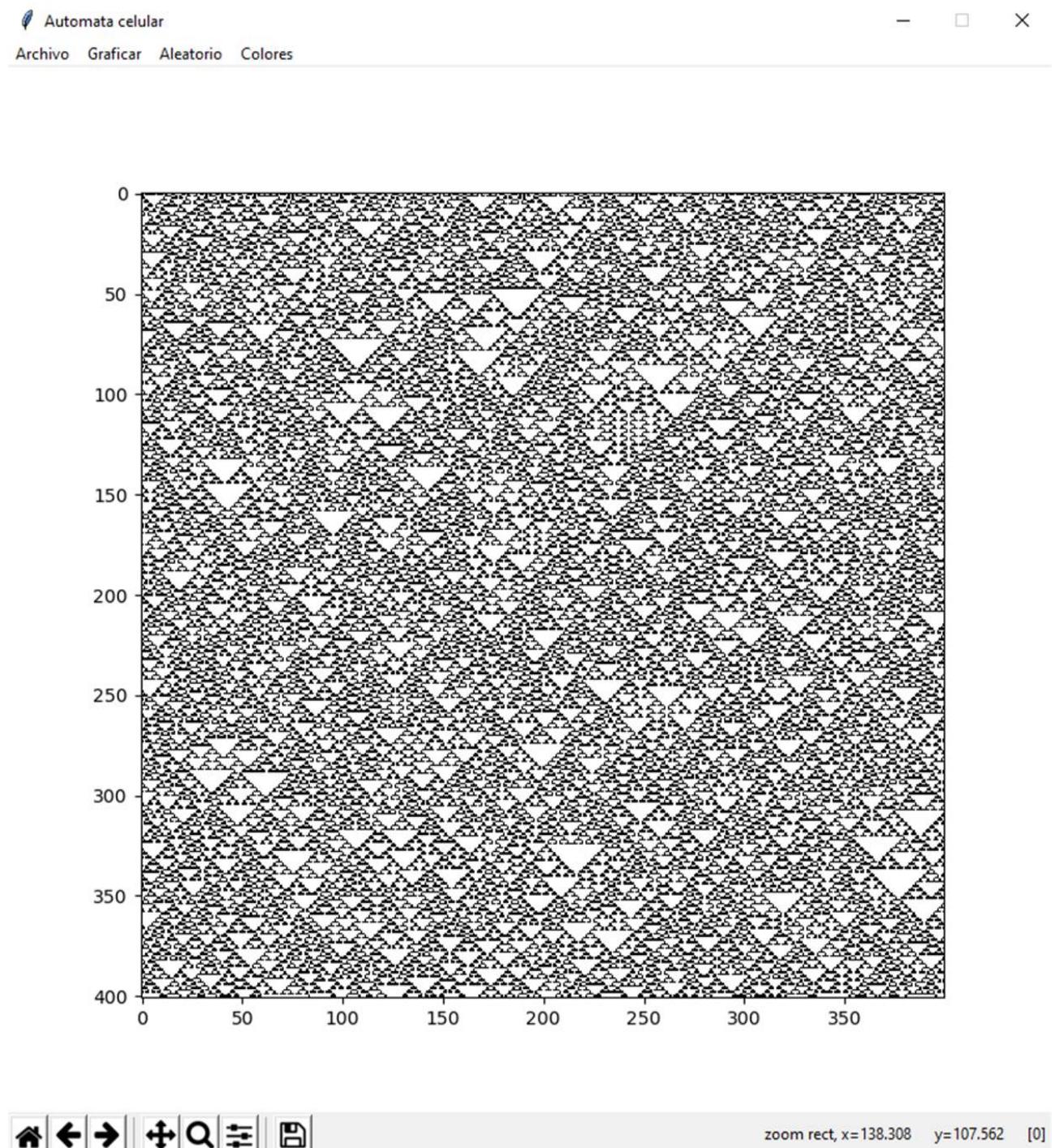


Figura 20: Regla 22 con 50 % de 1.

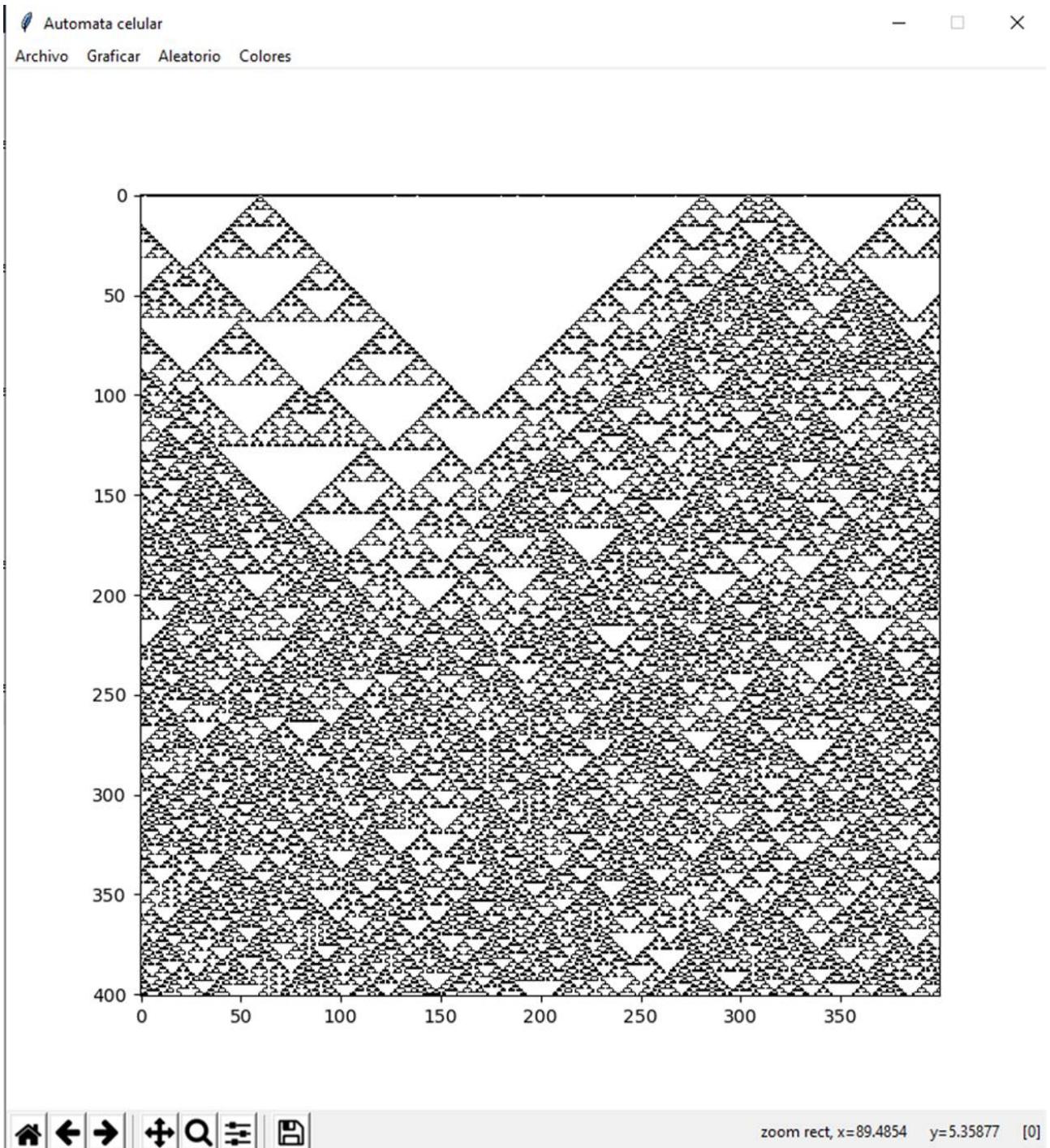


Figura 21: Regla 22 con 95 % de 1.

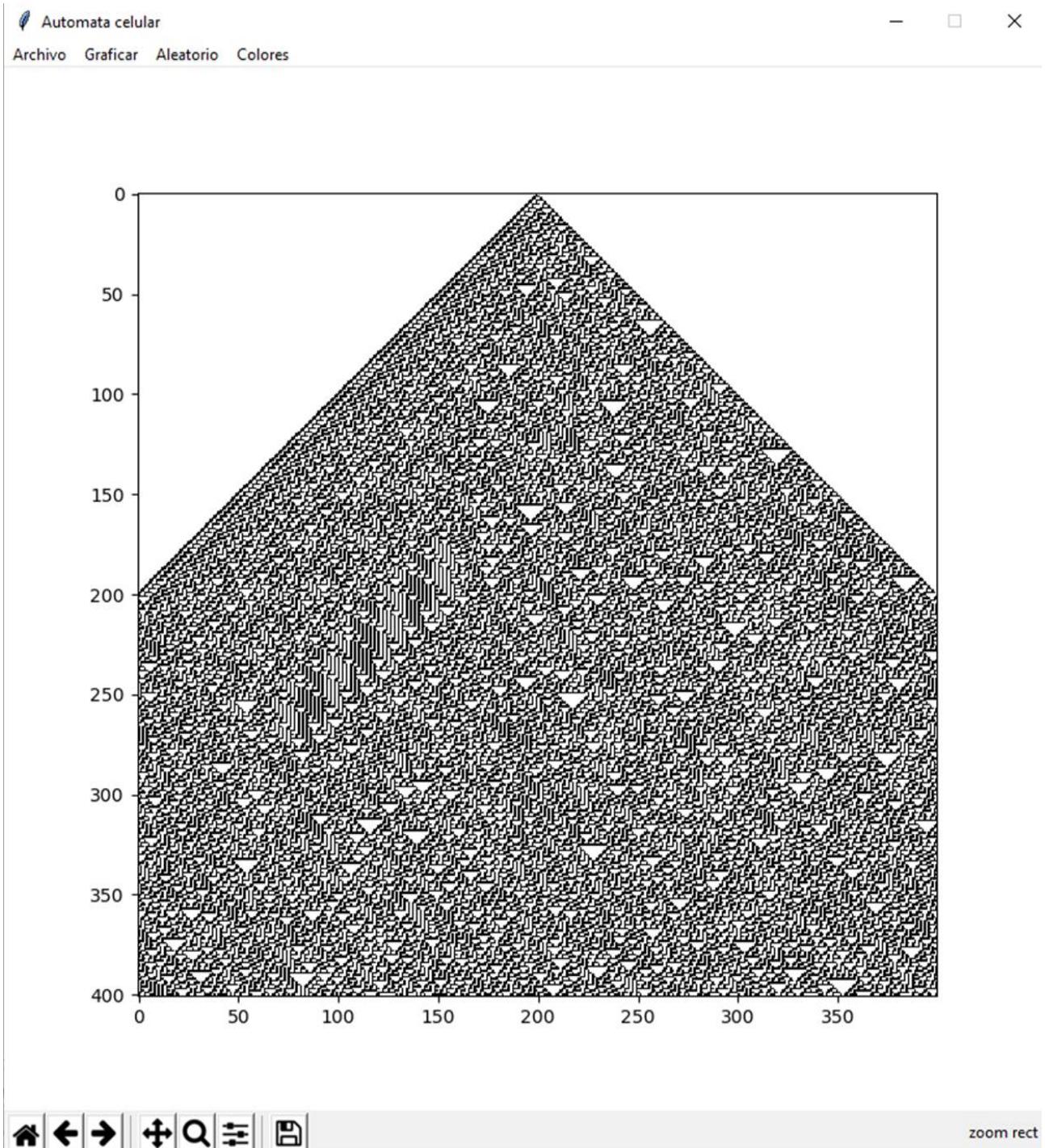


Figura 22: Regla 30 con un 1 en medio.

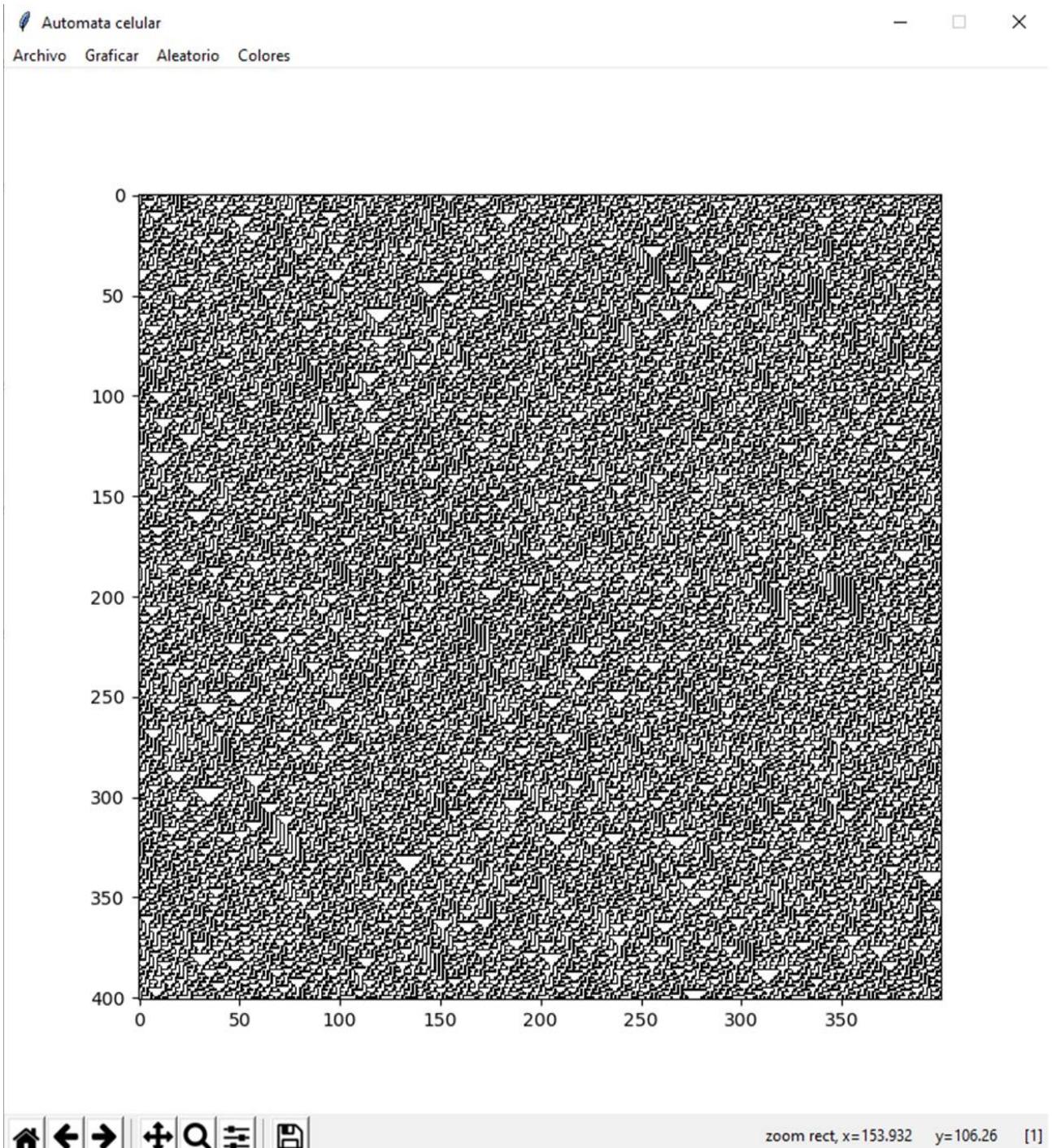


Figura 23: Regla 30 con 50 % de 1.

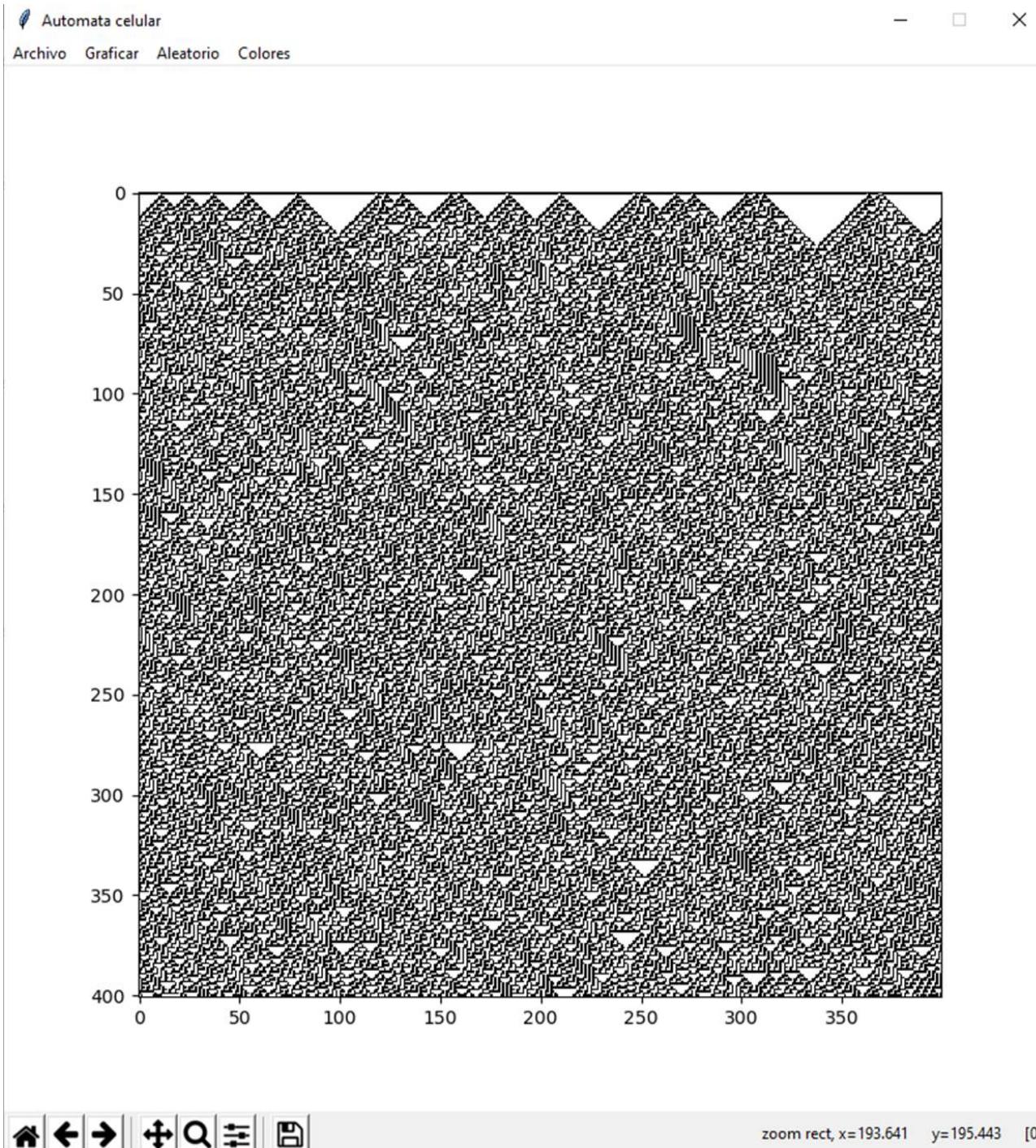
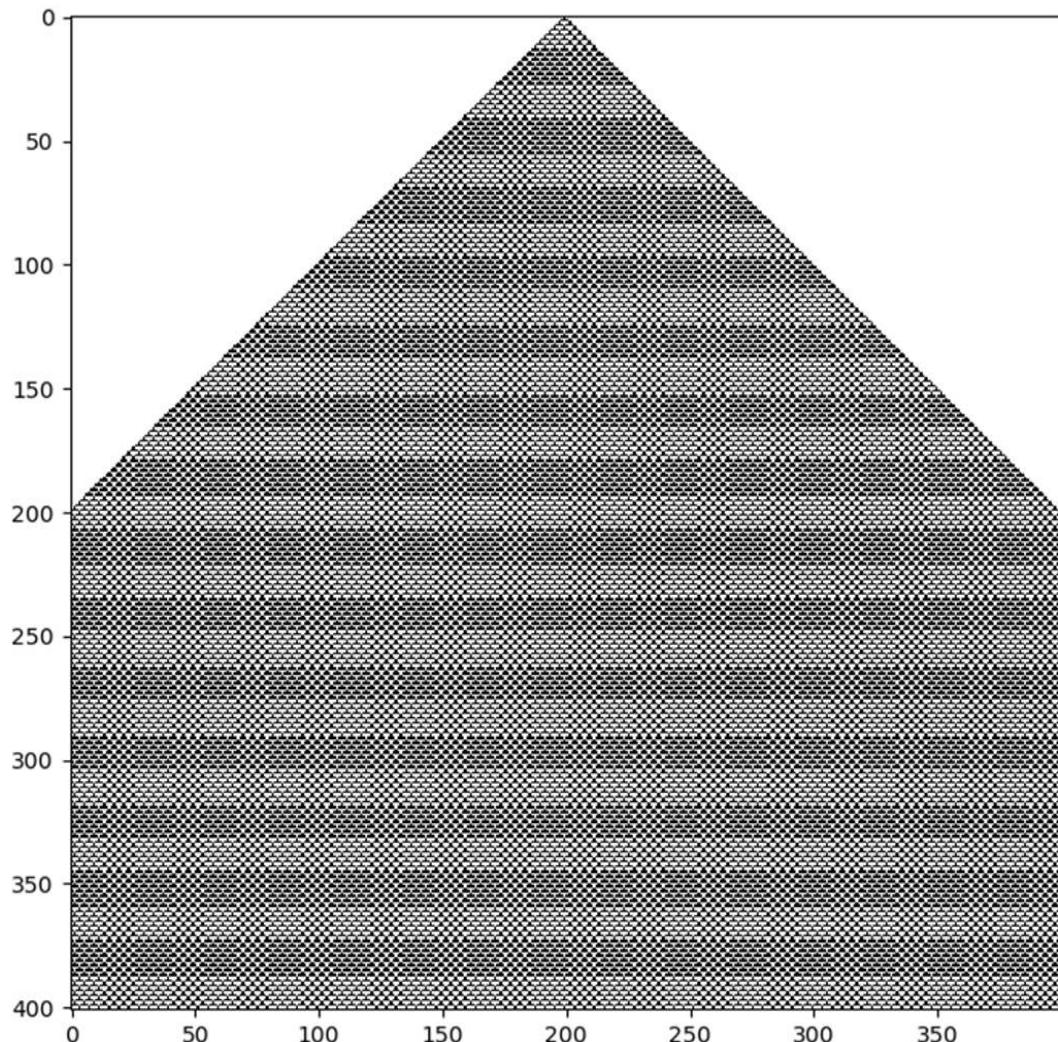


Figura 24: Regla 30 con 95 % de 1.

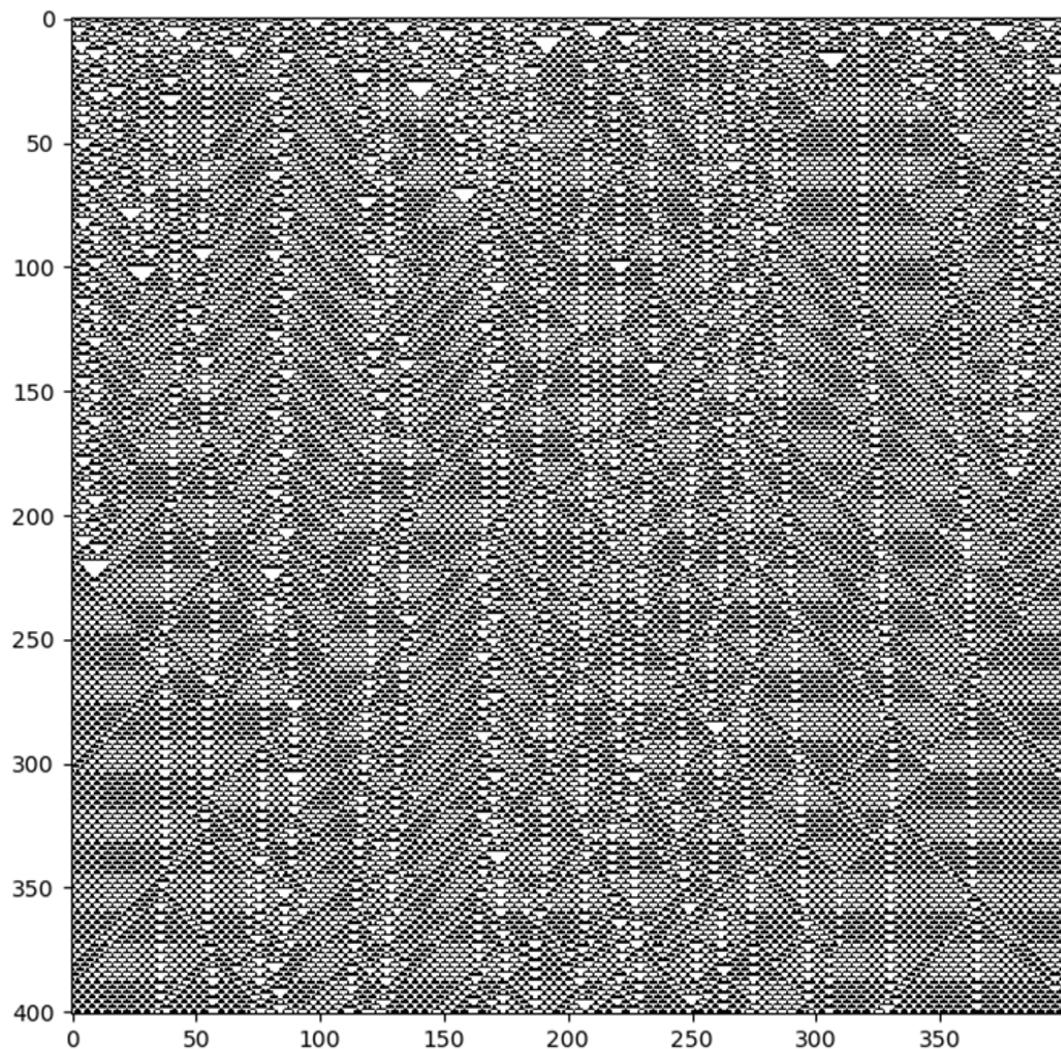
## Automata celular

Archivo Graficar Aleatorio Colores



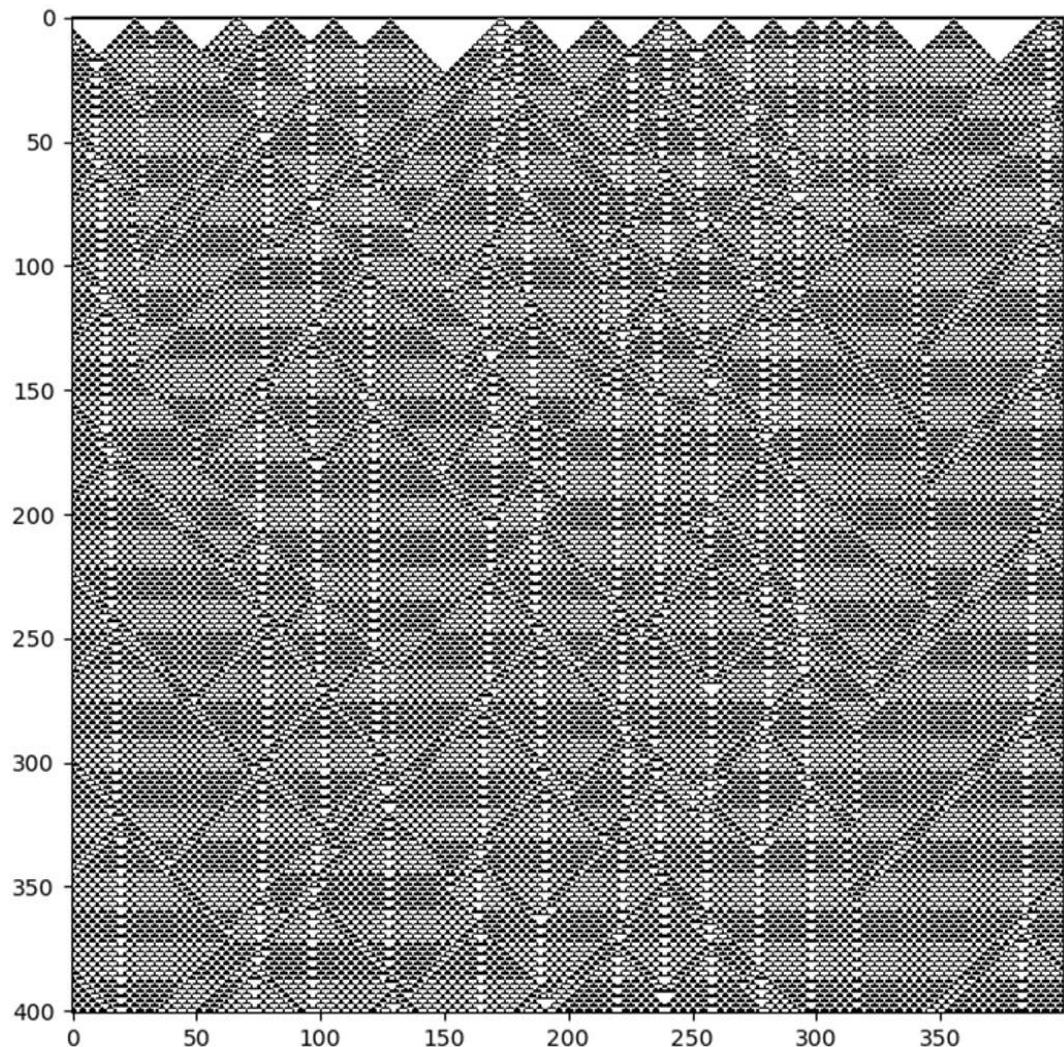
zoom rect

Figura 25: Regla 54 con un 1 en medio.



zoom rect, x=168.253 y=248.172 [1]

Figura 26: Regla 54 con 50 % de 1.



zoom rect, x=144.818 y=132.299 [1]

Figura 27: Regla 54 con 95 % de 1.

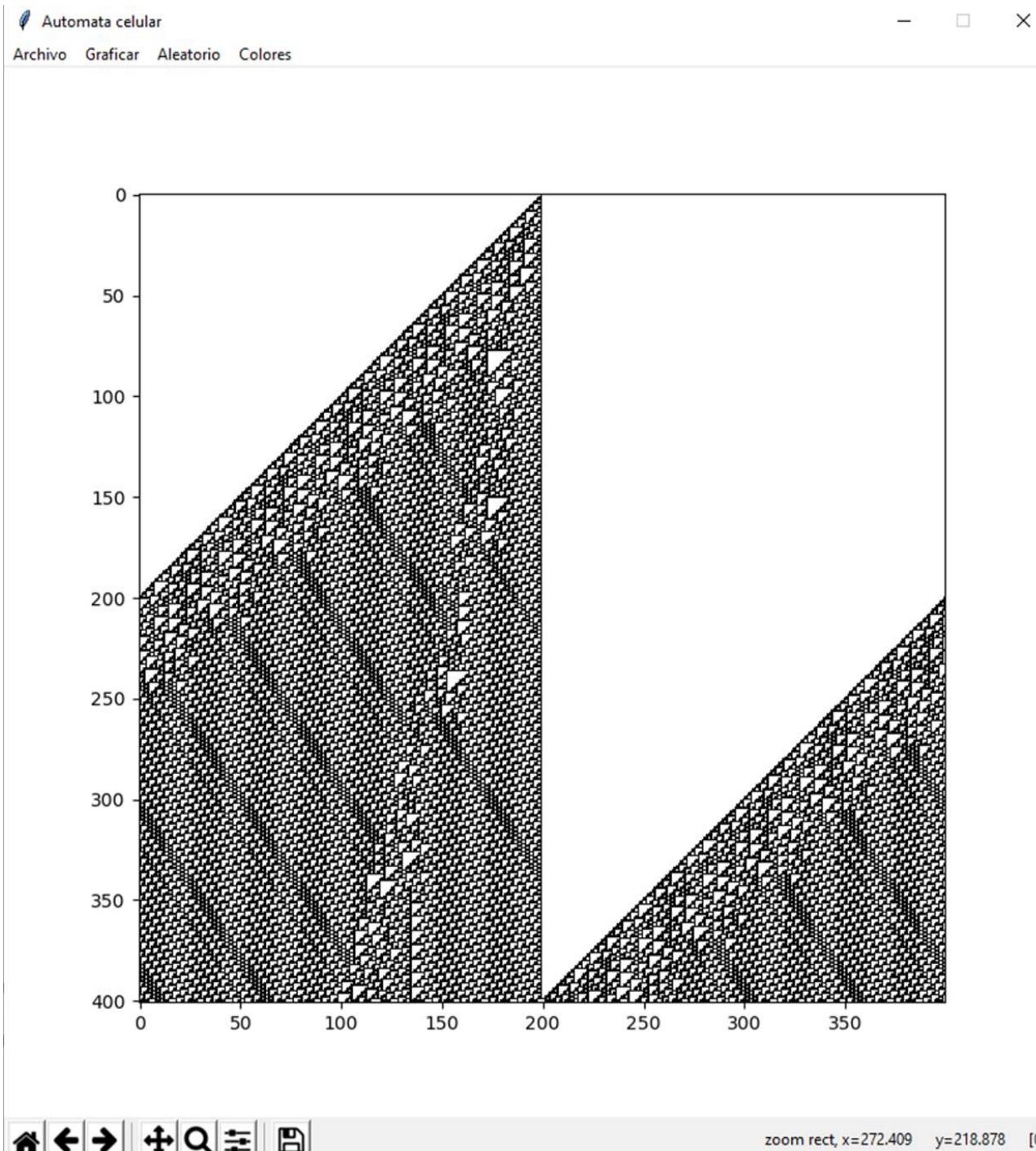


Figura 28: Regla 110 con un 1 en medio.

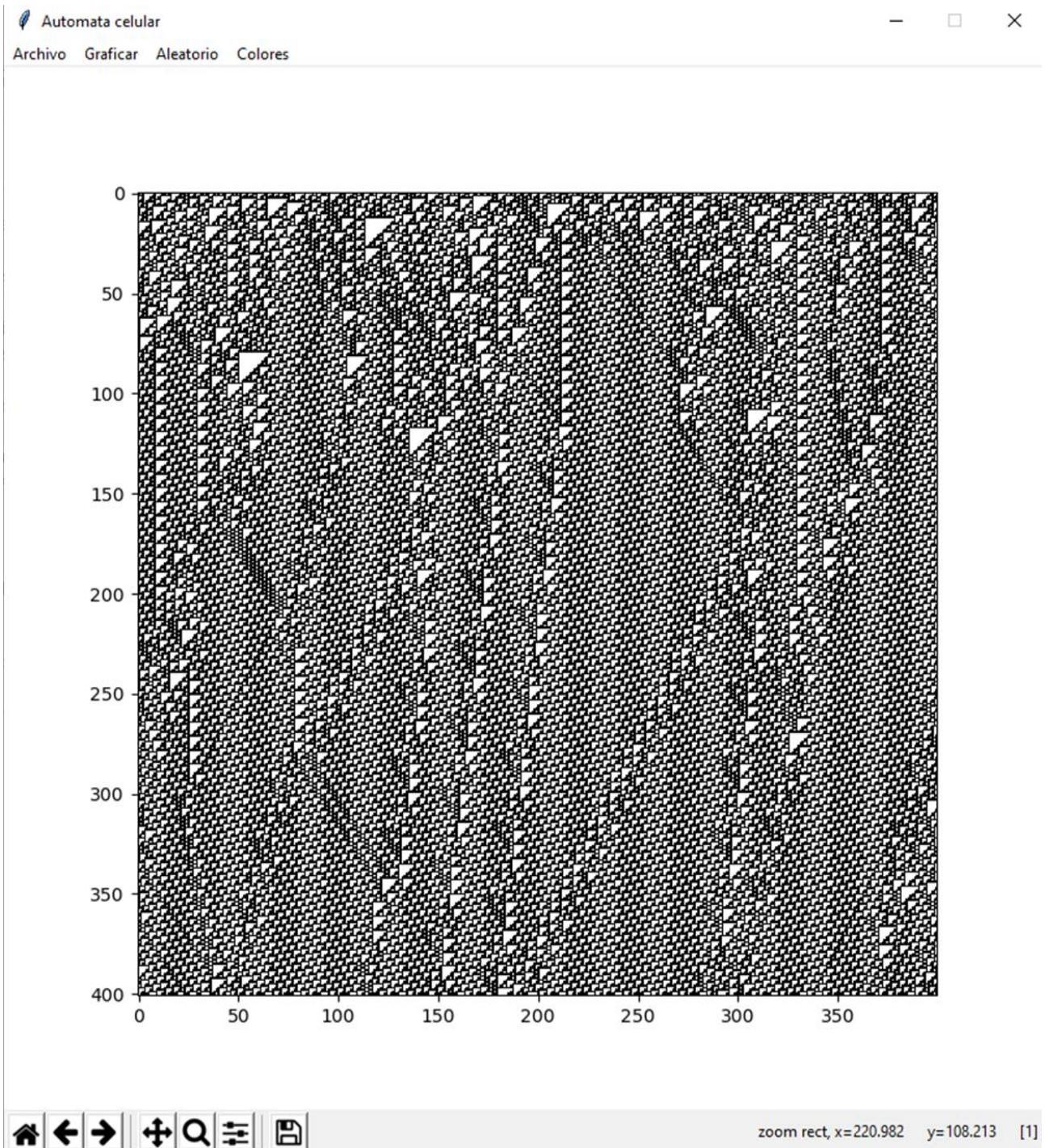
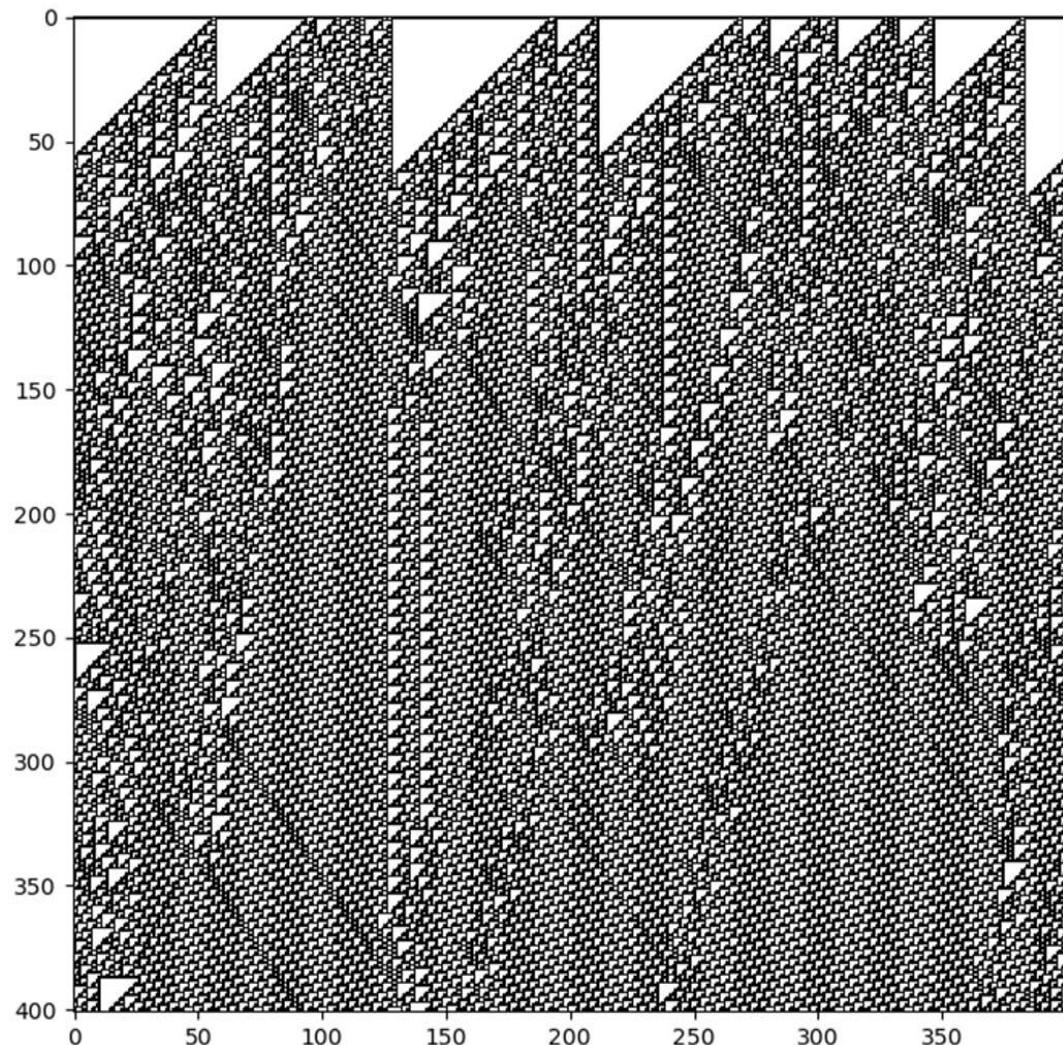


Figura 29: Regla 110 con 50 % de 1.



zoom rect, x=159.14 y=59.3896 [0]

Figura 30: Regla 110 con 95 % de 1.

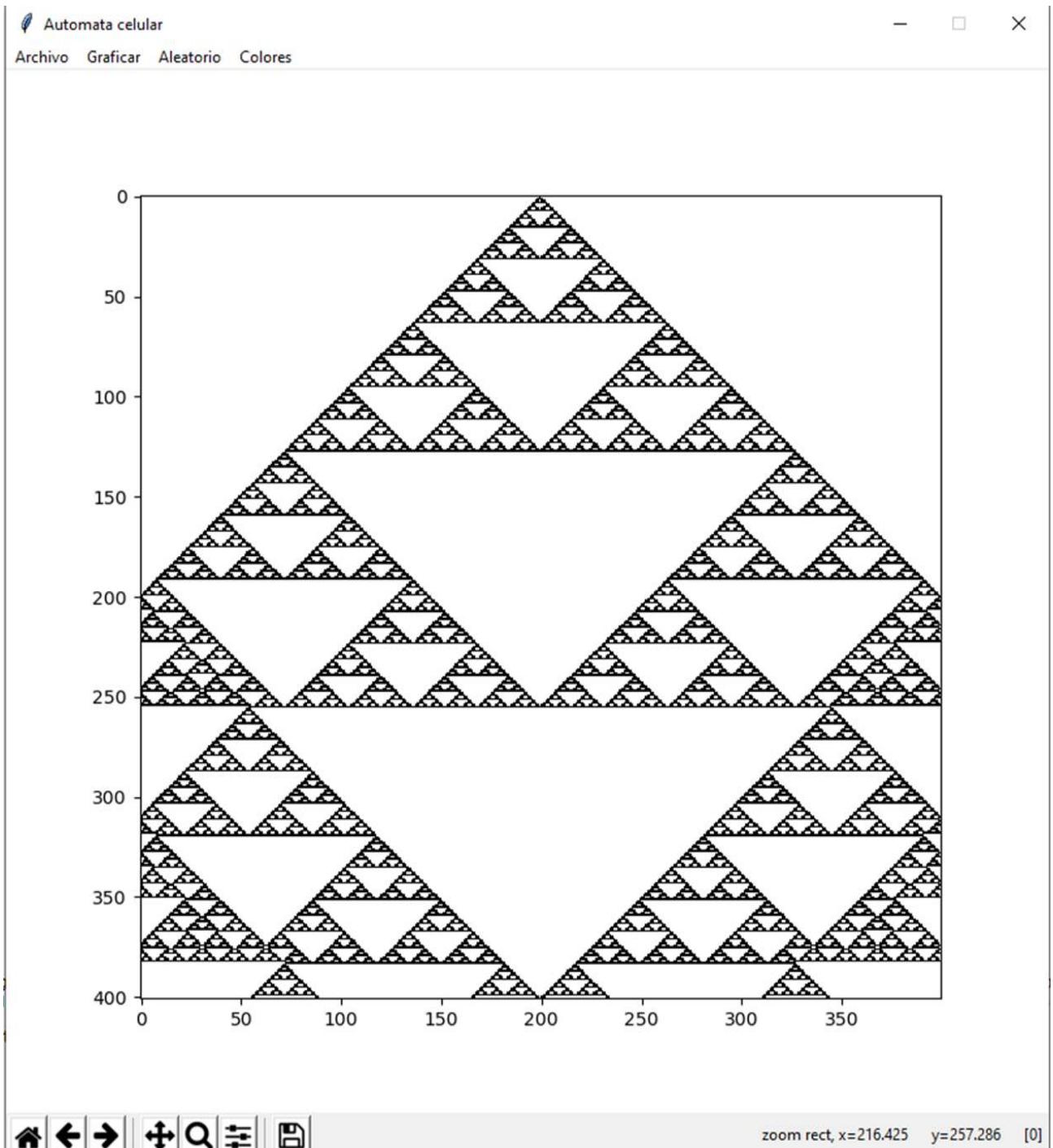


Figura 31: Regla 126 con un 1 en medio.

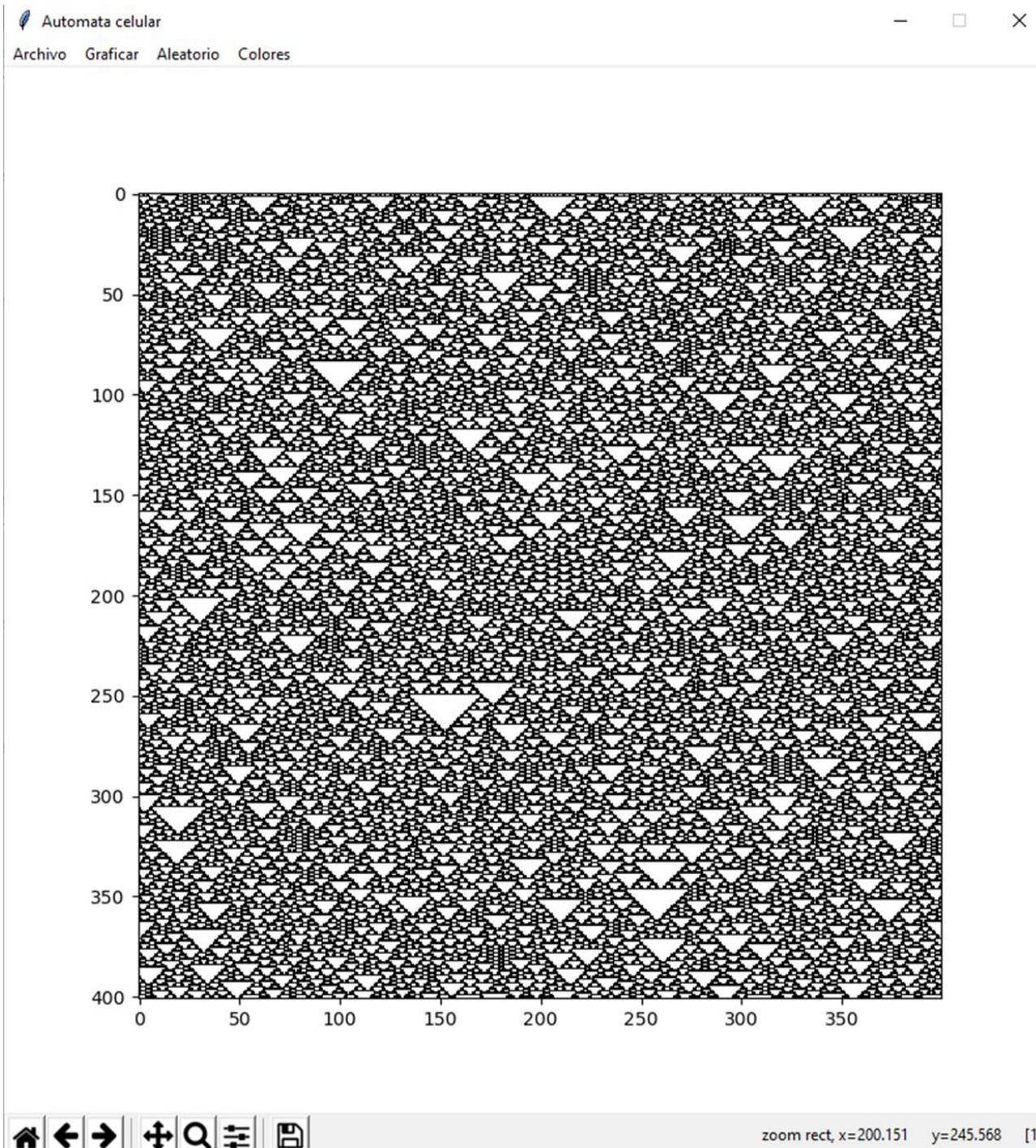


Figura 32: Regla 126 con 50 % de 1.

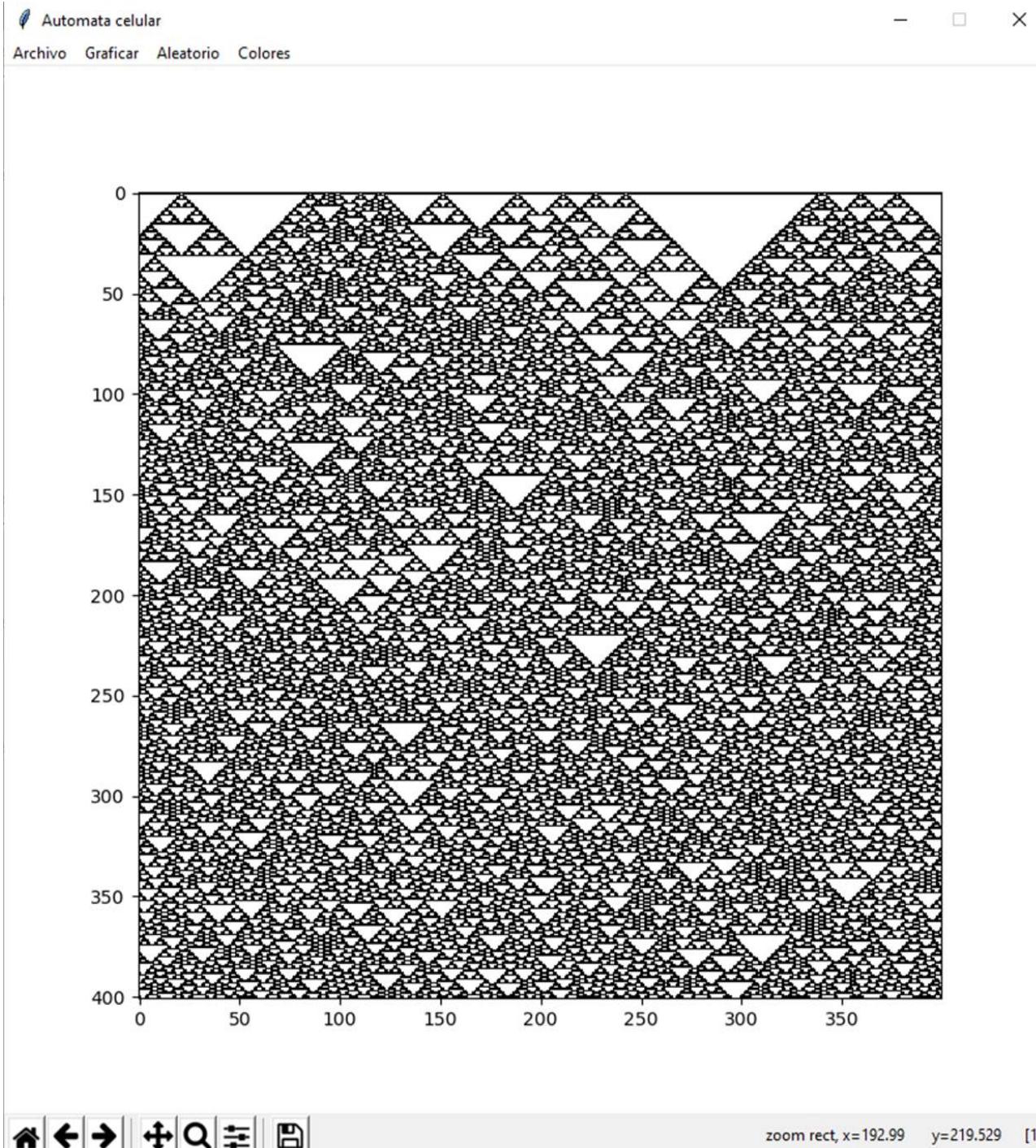


Figura 33: Regla 126 con 95 % de 1.