Nueva Guatemala de la Asunción

Manual de Técnico

Proyecto 1: Pixel Art

Elaborado por: Estuardo Gabriel Son Mux
Carné: 202003894
Fecha: 11/09/2021

Alcances y Objetivos del Programa

El programa realizado tiene como objetivo el análisis de los datos ingresados en un archivo *.pxla el cual contiene la información para la creación de una o varias imágenes simulando pixel art, así mismo la creación de un archivo html que contiene la lista de tokens y errores que se presenten durante el análisis del archivo.

Esto se logra mediante el análisis léxico del contenido del archivo y la utilización de la librería cairosvg para la creación de las imágenes.

Especificaciones Técnicas

Requisitos de Hardware

- Computadora con todos sus componentes para su correcto funcionamiento
- Laptop

Requisitos de Software

Sistema Operativo

El programa fue desarrollado en una laptop con sistema operativo Windows 10, 8GB de ram, Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz.

Nombre del dispositivo	DESKTOP-LJSJP7U
Procesador	Intel(R) Core(TM) i3-4005U CPU @ 1.70GHz 1.70 GHz
RAM instalada	8.00 GB
Id. del dispositivo	515F0098-1032-4937-BDB1-C191D818D118

Lenguaje de Programación

Python

Librería Cairosvg

IDE o Editor de Código

Visual Estudio

Lógica del Programa

Archivos Utilizados

El programa consta de siete archivos para el funcionamiento del mismo. Así mismo, cuenta con una imagen *.ico el cual es el icono de la ventana y una carpeta llamada Reportes donde se almacena el reporte Html y la imagen generada.

Principal.py

En este archivo se encuentra inicializada la clase de ventana para que sea posible su visualización, así como la implementación de procesos de la clase Ventana.

```
from tkinter import *
import Ventana

if __name__=="__main__":
    listaTokens=list()
    ventana=Tk()
    Interfaz=Ventana.Ventana(ventana)

    btnCargar = Interfaz.crearBoton("Cargar Archivos",20,95,1,1)
    btnAnalizar = Interfaz.crearBoton("Analizar Archivos", 20, 100,96,1)
    btnReportes = Interfaz.crearBoton("Ver Reportes", 20, 95,196,1)
    btnSalir = Interfaz.crearBoton("Salir", 20, 70,406,1)

    btnOriginal= Interfaz.crearBoton("Original", 80, 116, 40, 80,"#CBCFCA")
    btnMirrorX = Interfaz.crearBoton("Mirror X", 80, 116, 40, 160,"#CBCFCA")
    btnMirrorY = Interfaz.crearBoton("Mirror Y", 80, 116, 40, 240,"#CBCFCA")
    btnDMirror = Interfaz.crearBoton("Double Mirror", 80, 116, 40, 320,"#CBCFCA")

    Interfaz.crearLienzo();

    ventana.mainloop()
```

Ventana.py

En este archivo se encuentran todas las configuraciones de la ventana principal del programa.

```
from cairosvg import svg2png
from tkinter import filedialog
from tkinter import *
from tkinter import ttk
from PIL import Image
import Analizador
import os

class Ventana:
    def __init__(self, ventana):
        self.lienzo=None
        self.contenido=""
        self.procesar=None
        self.ventana = ventana
        self.ventana.title("Bitxelart")
        self.ventana.iconbitmap("icono.ico")
        self.frame = Frame(self.ventana,bg="grey",width=750,height=600)
        self.frame.pack()
        self.listaImagen= ttk.Combobox(self.frame, state="readonly")
        self.listaImagen.place(height=20, width=115,x=291,y=1)
```

En este archivo también se encuentran las configuraciones de los distintos botones que se encuentran en la ventana, como ejemplo la creación del archivo Html y la visualización de las imágenes dentro de la aplicación.

```
def verOriginal(self):
    for imagen in self.procesar.listaImagenes:
        if imagen.titulo==self.listaImagen.get():
            svg2png(bytestring=imagen.crearImagen(), write_to='Reportes\imagen.png')
            redimensionar=Image.open("Reportes\imagen.png")
            redimensionar = redimensionar.resize((500, 500), Image.ANTIALIAS)
            redimensionar.save("Reportes\imagen.png","png")
            imagens = PhotoImage(file="Reportes\imagen.png")
            self.crearLienzo(imagens)

def verX(self):
    for imagen in self.procesar.listaImagenes:
        if imagen.titulo==self.listaImagen.get() and imagen.mirrorX==True:
            svg2png(bytestring=imagen.crearMirrorX(), write_to='Reportes\imagen.png')
            redimensionar = Image.open("Reportes\imagen.png")
            redimensionar = redimensionar.resize((500, 500), Image.ANTIALIAS)
            redimensionar.save("Reportes\imagen.png", "png")
            imagens = PhotoImage(file="Reportes\imagen.png")
            self.crearLienzo(imagens)

def verY(self):
    for imagen in self.procesar.listaImagenes:
        if imagen.titulo==self.listaImagen.get() and imagen.mirrorY==True:
            svg2png(bytestring=imagen.crearMirrorY(), write_to='Reportes\imagen.png')
```

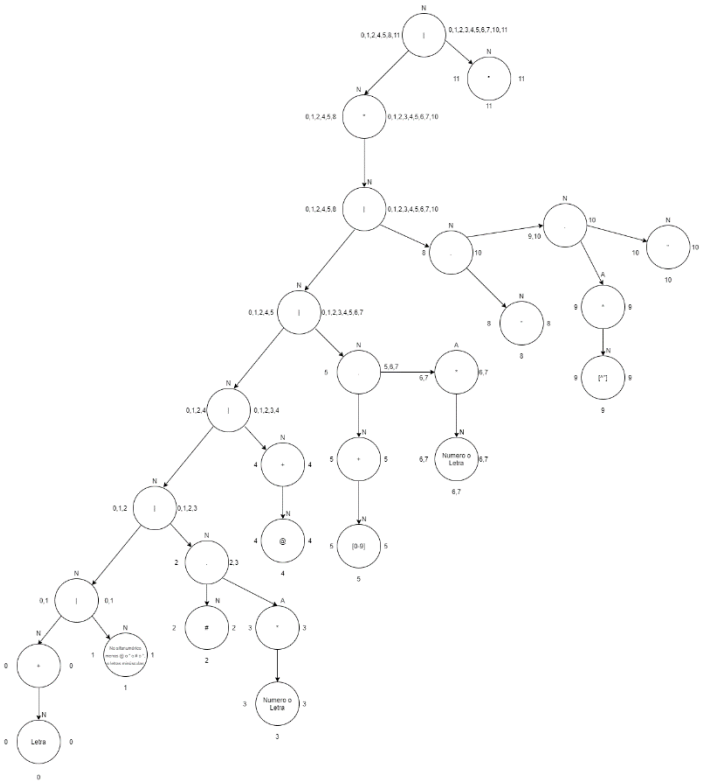
Analizador.py

En este archivo se encuentran los procesos encargados del análisis léxico del archivo que sea ingresado, reconociendo los tokens que formen parte del lenguaje o caracteres que no pertenezcan al lenguaje. Así mismo se encarga de crear los objetos de tipo imagen y almacenarlos en una lista.

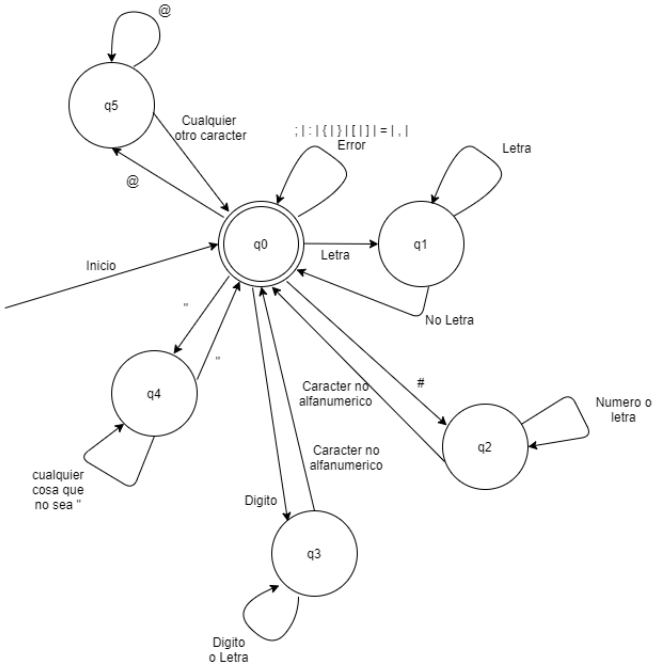
Los tokens que reconoce el analizador se presentan en la siguiente tabla:

Token	Lexema	Patrón
TITULO	TITULO	TITULO
ALTO	ALTO	ALTO
ANCHO	ANCHO	ANCHO
FILAS	FILAS	FILAS
COLUMNAS	COLUMNAS	COLUMNAS
CELDAS	CELDAS	CELDAS
FILTROS	FILTROS	FILTROS
Filtro	DOUBLEMIRROR	MIRRORX | MIRRORY | DOUBLEMIRROR
Llave Agrupación	{	{ | }
Corchetes	[	[|]
Punto y Coma	;	;
Coma	,	,
Entero	30	[0-9]+
Separador de Imagen	@@@@	@{4}
String	“Pokeball”	"[^"]*"
Igual	=	=
Booleano	TRUE	TRUE | FALSE
Codigo Color Hex	#92D050	#[a-fA-F0-9]{6}

Para la elaboración del autómata se utilizo el siguiente diagrama de árbol:



Que dio como resultado el siguiente autómata:



```
import Token
import Errores
import Posicion
import Imagen
import re

class Analizador:
    def __init__(self, contenido):
        self.contenido = contenido
        self.listaTokens = list()
        self.listaErrores = list()
        self.listaImagenes = list()
        self.imagen=Imagen.Imagen()
        self.comprobar=""
        self.posicion=None

    def reconocerTokens(self):
        centinela="°"
        texto=self.contenido+centinela
        linea=1
        noCaracter=1
        cadena=""
        estado=0
        i=0
```

```
def reconocerTokens(self):
    centinela="°"
    texto=self.contenido+centinela
    linea=1
    noCaracter=1
    cadena=""
    estado=0
    i=0

    while i<len(texto):
        c=texto[i]
        if estado==0:
            if c=="{":
                if self.comprobar=="CELDAS=":
                    self.comprobar+="{"

                self.listaTokens.append(Token.Token("Llave A
                noCaracter+=1

            elif c=="}":
                self.listaTokens.append(Token.Token("Llave A
                noCaracter+=1
```

Imagen.py

Esta clase es de utilidad para la creación de objetos de tipo imagen almacenando sus datos de nombre, celdas, alto, ancho, filas y columnas, también se incluyen los filtros aplicables.

```
class Imagen:
    def __init__(self):
        self.titulo=""
        self.mirrorX=False
        self.mirrorY=False
        self.doubleMirror=False
        self.width=0
        self.height=0
        self.filas=0
        self.columnas=0
        self.listaPosiciones=list()

    def setTitulo(self, titulo):
        self.titulo=titulo

    def setWidth(self, width):
        self.width=width

    def setHeight(self, height):
        self.height=height

    def setFilas(self, fila):
        self.filas=fila
```

Posicion.py

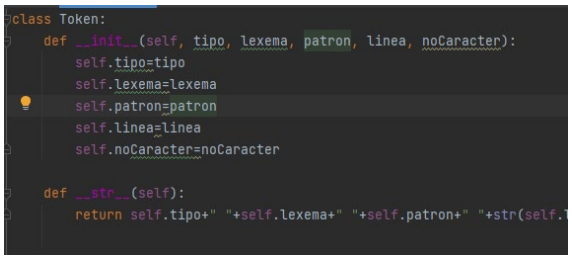
Esta clase almacena las posiciones de cada celda, los colores y si la verificación de si debe ser o no pintada.

```
class Posicion:
    def __init__(self):
        self.x=float('inf')
        self.y=float('inf')
        self.pintar=False
        self.color="#ffffff"

    def __str__(self):
        return str(self.x)+" "+str(self.y)+" "+str(self.pintar)+" "+self.color
```

Token.py

Después de reconocer un token el analizador léxico de la clase analizador crea un objeto de la clase token que contiene el token reconocido, el patrón al que pertenece, la fila y la posición en la que se encuentra dentro de la fila.

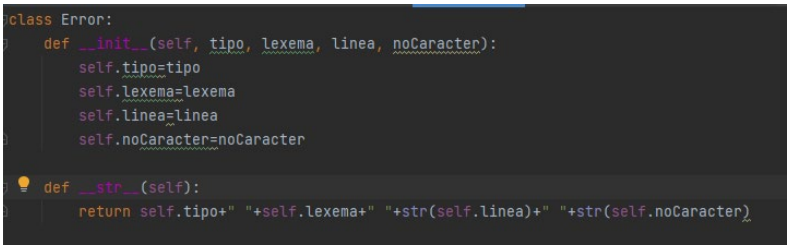


```
class Token:
    def __init__(self, tipo, lexema, patron, linea, noCaracter):
        self.tipo=tipo
        self.lexema=lexema
        self.patron=patron
        self.linea=linea
        self.noCaracter=noCaracter

    def __str__(self):
        return self.tipo+" "+self.lexema+" "+self.patron+" "+str(self.
```

Errores.py

En el caso de que el analizador léxico detecte una cadena o carácter que no pertenezca al lenguaje, se creara un objeto de la clase errores que contiene la posición y fila en la que se encuentra.



```
class Error:
    def __init__(self, tipo, lexema, linea, noCaracter):
        self.tipo=tipo
        self.lexema=lexema
        self.linea=linea
        self.noCaracter=noCaracter

    def __str__(self):
        return self.tipo+" "+self.lexema+" "+str(self.linea)+" "+str(self.noCaracter)
```

Flujo del Programa

Se inicia el programa con una ventana principal la cual es invocada en el archivo Principal.py y se forma con la clase Ventana, es atreves de esta que se llega a las funciones las diferentes funciones del programa.

La función de carga de archivo accede la clase Analizador.

En la clase Analizador se crean objetos a partir de las clases Imagen, Posiciones, Token y Errores los cuales almacenan los datos de las imágenes especificadas en el archivo de entrada.

La clase Imagen contiene en sus atributos una lista de posiciones que será la encargada de almacenar los objetos Posiciones creados por la clase Analizador.