Universidad De San Carlos de Guatemala
Facultad de Ingeniería
Escuela de Ciencias y Sistemas

Lenguajes Formales y de Programación
Sección “B-”

“MANUAL TÉCNICO”

Carlos Augusto Calderón Estrada

201905515

Objetivos

General:

Brindarle al desarrollador a trabajar en esta aplicación una guía lo más completa y sencilla posible para el mejor entendimiento de lo que se lleva desarrollado en la aplicación y así evitar errores posibles por el uso incorrecto de los algoritmos y/o estructuras.

Específicos:

- Que el desarrollador trabaje en la aplicación como si fuera creada por el/ella.
- Proporcionarle al lector una explicación sencilla y técnica de los procesos algorítmicos y las relaciones de los métodos, funciones y atributos que son esenciales en la aplicación.

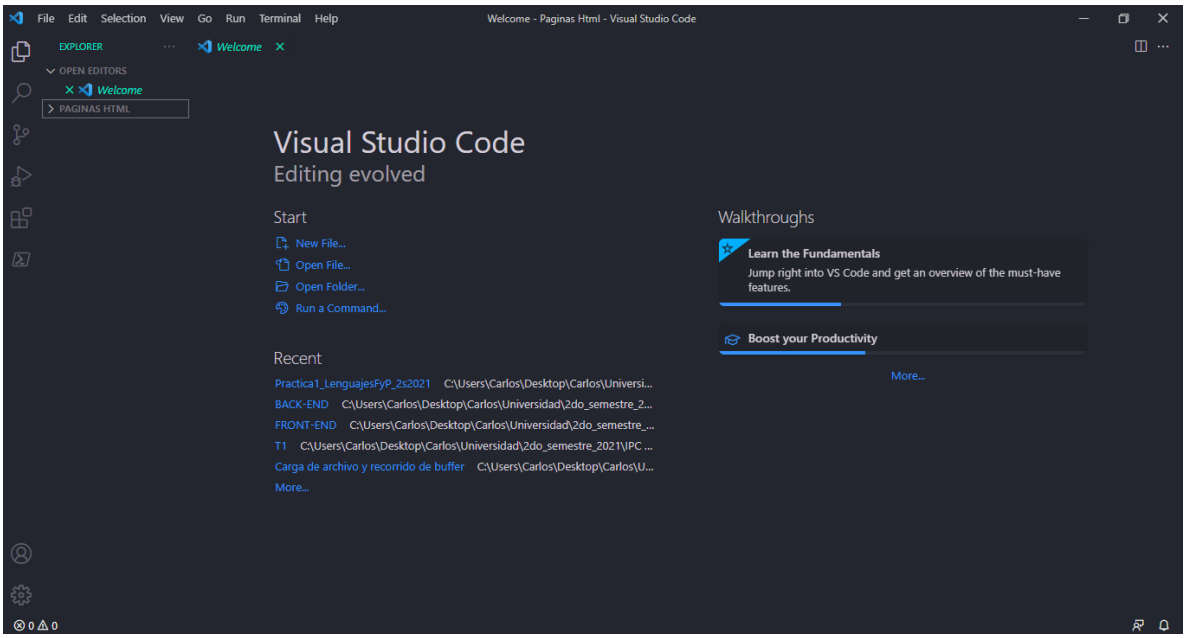
Introducción

Este manual técnico tiene como fin dar a conocer al desarrollador las mejores recomendaciones, requerimientos para un mejor entendimiento al momento de querer realizar modificaciones, optimización de procesos o métodos, indicando el editor de texto que se utilizó para el desarrollo de dicha aplicación.

La aplicación se basa en Analizar un archivo por medio de un autómata finito creado por el desarrollador, en el que se cargara por medio de una ventana emergente y este carga el archivo al programa(todo esto a través de una interfaz gráfica de Python) y se procede a realizar el análisis de dichos datos a utilizar en dicha ejecución, en el que por medio de estructuras de control **IF**, que cualquier desarrollador con un poco de experiencia conoce, se basa el autómata, almacenándolas en variables que serán utilizadas como tokens y son indispensables, luego de realizar el análisis, se generaran html que contendrán un lienzo o cuadro de pixels, estos contendrán un estilo único para cada imagen, y por medio de graphviz generamos la imagen que se visualiza en html, estas mismas podrán ser visualizadas en el apartado de Analizar, con finalidad se generan los reportes de errores y de tokens por medio de escritura de archivo que contienen la lista de errores y de tokens.

Editor de texto utilizado

1. El editor de texto utilizado fue Visual Studio Code gracias a la variedad de extensiones que posee y su fácil uso para el manejo de la nube git, que permite agregar la propia terminal y ejecutar los comandos desde el entorno de VS Code.



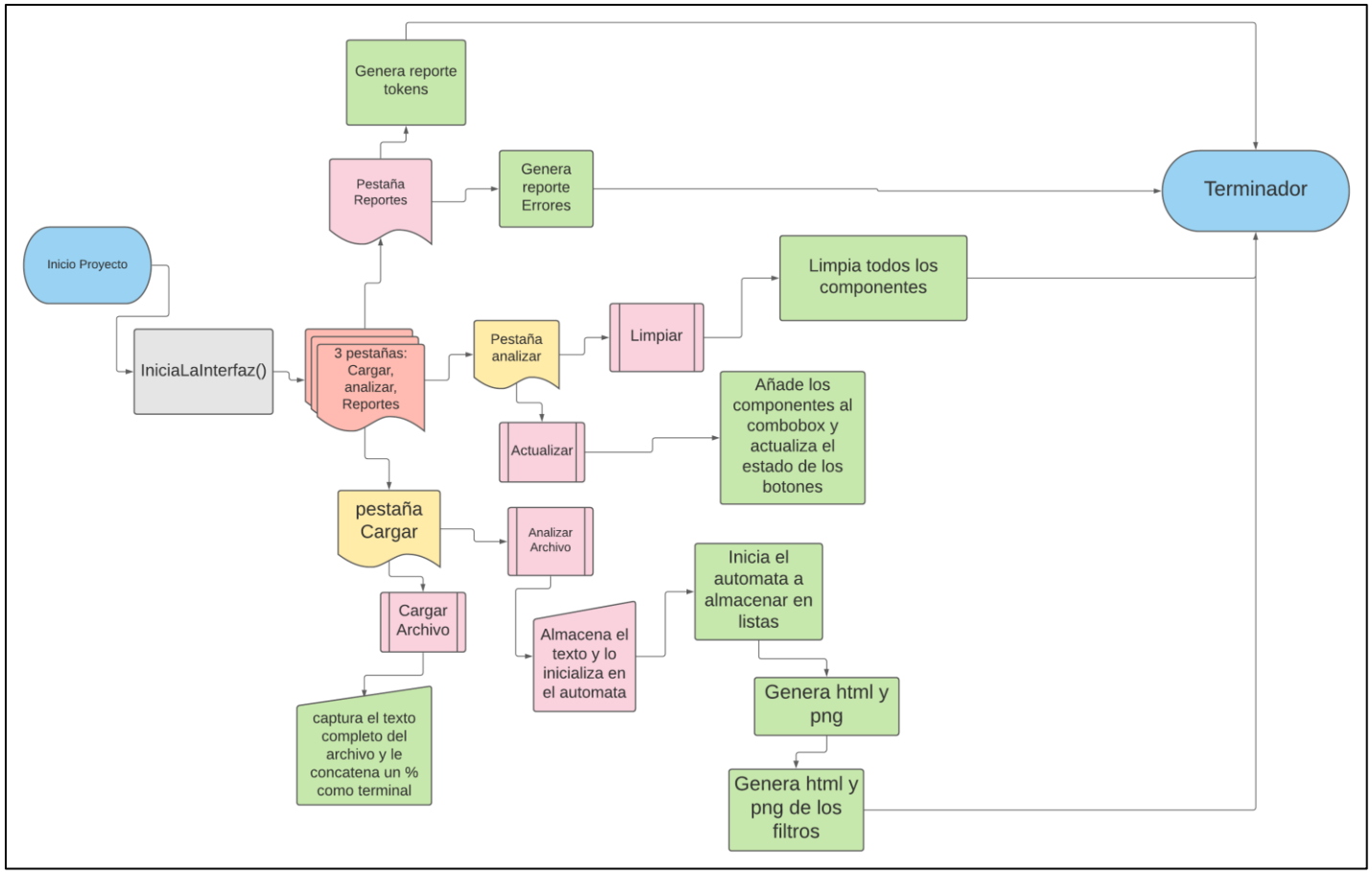
Requerimientos:

- Procesador de 1.6 GHz o más.
- 1 GB de memoria RAM
- Plataformas aprobadas: OS X, Windows 7, Linux(Debian), Linux (Red Hat)
- Microsoft .NET Framework 4.5.2 para VS Code

Sistema Operativo que se llevó a cabo:

Windows 10 de 64 bits

Diagrama de flujo



Lógica del Programa

Main: La clase principal crea el objeto de la interfaz y la ejecuta para poder trabajar en ella, si se cierra, se cierra el programa

```
gestor = Gestor()
if __name__ == "__main__":
    import sys
    app = QtWidgets.QApplication(sys.argv)
    Bixelart = QtWidgets.QDialog()
    ui = InterfazInicio()
    ui.setupUi(Bixelart)
    Bixelart.show()
    sys.exit(app.exec_())
```

Cargar Archivo:

```
def CargarArchivo(self): ##Inicio del metodo
        self.Imagen.clear()## al cargar un nuevo archivo se borran las image
nes actuales
        self.Tokens.clear()## al cargar un nuevo archivo se borran los token
s actuales
        self.Errores.clear()## al cargar un nuevo archivo se borran los erro
es actuales
        global texto
        Tk().withdraw()##importamos la opciones para la ventana emergente
        archivo = filedialog.askopenfile(initialdir="./Archivos Prueba", tit
le="Seleccione un archivo",filetypes=(("Archivos pxla",".pxla"),("ALL files"
,".txt")))
        if archivo is None:
            print('No se selecciono ni un archivo\n')
            return None
        else:
            ##abrimos el archivo y lo leemos
            texto = archivo.read()
            print(texto)
            print('\n\n')
            ##concatenamos el simbolo terminal
            texto+='%'
            print(texto)
            archivo.close()##Cerramos el archivo
            print("Lectura Exitosa")
```

Función isLetra:

```
 def isLetra(self,caracter):##metodo que retorna si es una letra alfabetica
        if((ord(caracter) >= 65 and ord(caracter) <= 90)
        or (ord(caracter) >= 97 and ord(caracter) <= 122)
        or ord(caracter) == 164 or ord(caracter) == 165):
            return True
        else:
            return False
```

Función isNumero:

```
def isNumero(self,caracter):##metodo que retorna si es un digito
        if ((ord(caracter) >= 48 and ord(caracter) <= 57)):
            return True
        else:
            return False
```

Función Analizar:

```
def Analizar(self):##metodo del automata finito que analiza
        global texto
        ##variables y tokens iniciales
        HayError = False
        tempCelda=[]
        tempFiltros=[]
        titulo = ""
        ancho = ''
        alto = ''
        fila = ''
        columna = ''
        poX=''
        poY=''
        valorB = ''
        color= ''
        estado = 0
        lexema = ""
        contadorColumna=0
        contadorFila=1
```

- **Inicio del recorrido del archivo**

```
-   for x in texto: # Inicia la lectura del archivo caracter*caracter
-               if(ord(x)==10):## decision si hay un salto de linea se rei
    nicia contadores
-                   contadorFila+=1
-                   contadorColumna=0
-               contadorColumna+=1
```

- **Ejemplo de los estados en el automata**

```
if (estado==0):## Inicia el automata
                if (self.isLetra(x)==True): ##Verifica si hay letra
                    lexema += x
                    estado = 1## Cambia de estado
            elif (estado==1):##Estado Nuevo

                if (self.isLetra(x)==True):#Verifica si hay letra
                    lexema += x
                    HayError=False
                elif (x == '='): # Verifica si es un signo "="
                    if ((lexema == "TITULO")
                    or (lexema == "ANCHO")
                    or (lexema == "ALTO")
                    or (lexema == "FILAS")
                    or (lexema == "COLUMNAS")
                    or (lexema == "CELDAS")
                    or (lexema == "FILTROS")):##Verifica si es de los tokens
 permitidos
                        estado = 2 ##Si es true, cambia de estado
                    else: ## Si no
                        e =('--
> Error Lexico, se detecto: ' + lexema + ' en ''Fila: '+str(contadorFila)+'
Columna: '+str(contadorColumna)+' favor de revisar')
                        HayError=True
                        lexema=''
                        ##Envia el error de el token no existente
                        self.Errores.append(Errores(str(contadorFila), str(c
ontadorColumna), e))
```

- **Estado de Nueva imagen:**

```
-   elif (estado == 20): ##hay mas imagenes o fin
-                   if(x=='@'): ## si es un @
-                       contadorArroba=1 # Cuenta el numero de arrobas e i
    ncrementa
-                       lexema+=x #agrega al lexema
-                       estado = 22 ##Cambia estado
-                   elif (x=='%'):## si es un %
-                       #lo agrega al simbolo de tokens
-                       self.Tokens.append(Token("SimboloFin",x,contadorFi
    la,contadorColumna))
-                       #agrega la imagen
-                       self.Imagen.append(Imagen(titulo,int(ancho),int(al
    to),int(fila), int(columna),tempCelda,tempFiltros))
-                       estado = 23# cambia estado
-                   elif ord(x) == 32 or ord(x) == 10 or ord(x) == 9: #si
    es espacio tab o salto
-                       pass # lo ignora
-                   else:
-                       e =('--
    > Error Lexico, se detecto ' + x +" en "+"Fila: "+str(contadorFila)+"
    Columna: "+str(contadorColumna)+' Solo se permite valores valores "@"
    y "%" ')
-                       HayError=True ##establece que hay error en la imag
    en
-                       lexema=''
-                       ## envia el error
-                       self.Errores.append(Errores(str(contadorFila), str
    (contadorColumna), e))
```

- **Estado terminal:**

```
if (estado==23):## estado terminal
            if HayError==False:# si no hay error
                self.GenerarArchivosPixeles() # genera los archivos html
            else:# si hay error imprime mensaje en consola
                 print('La imagen final no se genero')
```

- **Generar Archivo Pixel:** Este método genera los html

```
def GenerarArchivosPixeles(self):##Genera los html y manda funcion para imag
enes
        for i in range(0,len(self.Imagen)):## recorre lista imagenes
            filehtml = open("./Htmls/"+self.Imagen[i].Titulo+".html","w")##
abre html
            fileCss = open("./Htmls/css/"+self.Imagen[i].Titulo+".css","w")#
abre css
            pixel = ''# variable
            #Recorre el numero de pixeles a crear
            for c in range(int(self.Imagen[i].Filas)*int(self.Imagen[i].Colu
mnas)):
                #concatena los pixeles
                pixel += '<div class="pixel"></div>\n\n'
            contenidoHTML=(## cuerpo del html
              '<!DOCTYPE html>\n'
                ' <html>\n'
                '<head> \n'
                '<meta charset="utf-8"> \n'
                '<link rel="stylesheet" href="css/'+self.Imagen[i].Titulo+'.
css">\n'
                '<title>Reporte' +self.Imagen[i].Titulo +'</title>\n'
                '</head>\n'
                '<body>\n'
                '<div class="canvas">\n'
                +pixel+
                '</div>\n''</body>\n''</html>\n'
            )
            filehtml.write(str(contenidoHTML))# escribe el archivo html
            cssPixel =''
            contador=0
            #mismo proceso para el css y filtros
fileCss.write(str(contenidoCSS))##escribe css
            ## cierra archivos
            fileCss.close()
            filehtml.close()
            cssPixel=pixel=''
            #genera las imagenes en Graphviz
            self.GenerarPNG(self.Imagen[i].Titulo,str(AlmacenarGrafica))
##Generamos los html de los filtros
        self.HtmlFiltros()
        print("Ver filtros")
```

- **Generar PNG:** Se genera los png por medio de Graphviz

```
def GenerarPNG(self,titulo,texto):
        Ima = Digraph(format='png')
        Ima.node(titulo,label=texto,color='white')
        Ima.render('Imagenes/'+titulo)
```

- **Reporte tokens y errores:** Se basa de igual forma que la escritura de html imágenes, ya que es recorriendo la lista de tokens o errores, pero adicional lo agregamos a una tabla simple y le agregamos el mismo estilo

2. **Interfaz:** Se utilizó PyQt5 como librería drag and drop

```
      #Creacion de Labels

        self.Cargar.setPalette(palette)
        self.Cargar.setObjectName("Cargar")
        self.label = QtWidgets.QLabel(self.Cargar)
        self.label.setGeometry(QtCore.QRect(70, 10, 491, 20))
        font = QtGui.QFont()
        font.setFamily("Arial")
        font.setPointSize(14)
        font.setBold(True)
        font.setWeight(75)
        self.label.setFont(font)
        self.label.setWordWrap(False)
        self.label.setObjectName("label")
        ##Creacion de botones
        self.pushButton = QtWidgets.QPushButton(self.Cargar)
        self.pushButton.setGeometry(QtCore.QRect(60, 40, 481, 101))
        palette = QtGui.QPalette()
        self.pushButton.setFont(font)

        self.pushButton.setCursor(QtGui.QCursor(QtCore.Qt.ArrowCursor))
        self.pushButton.setObjectName("pushButton")

        ## ACCION DEL BOTON CARGAR ARCHIVO
        self.pushButton.clicked.connect(gestor.CargarArchivo)
        self.pushButton_2 = QtWidgets.QPushButton(self.Cargar)
        self.pushButton_2.setGeometry(QtCore.QRect(60, 170, 481, 101))
```

- **Método retranslateUi:** Establece los nombre visibles de los componentes.

```
-   def retranslateUi(self, Bixelart):
-           _translate = QtCore.QCoreApplication.translate
-           Bixelart.setWindowTitle(_translate("Bixelart", "Inicio"))
-           self.label.setText(_translate("Bixelart", "Por Favor presione
    el boton para cargar el archivo"))
-           self.pushButton.setText(_translate("Bixelart", "Cargar Archivo
    "))
-           self.pushButton_2.setText(_translate("Bixelart", "Analizar Arc
    hivo"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.Cargar), _t
    ranslate("Bixelart", "Cargar Archivo"))
-           self.lblImagen.setText(_translate("Bixelart", ""))
-           self.groupBox.setTitle(_translate("Bixelart", "Botones"))
-           self.btnOriginal.setText(_translate("Bixelart", "Original"))
-           self.btnMiX.setText(_translate("Bixelart", "MirrorX"))
-           self.btnMiY.setText(_translate("Bixelart", "MirrorY"))
-           self.btnDMi.setText(_translate("Bixelart", "DoubleMirror"))
-           self.btnActCombo.setText(_translate("Bixelart", "Actualizar"))
-           self.btnLimpiar.setText(_translate("Bixelart", "Limpiar"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.Analizar),
    _translate("Bixelart", "Analizar"))
-           self.pushButton_7.setText(_translate("Bixelart", "Reporte De T
    okens"))
-           self.pushButton_8.setText(_translate("Bixelart", "Reporte De E
    rrores"))
-           self.label_2.setText(_translate("Bixelart", "Por Favor presion
    e el boton para Generar el Reporte deseado"))
-           self.toolBox.setItemText(self.toolBox.indexOf(self.GenerarRepo
    rtes), _translate("Bixelart", "Reportes"))
-
```

función addCombo: Agrega los datos al Combo box

```
def AddCombo(self):

        if self.Combo.count() < 1:
            for i in range(len(gestor.Imagen)):
                self.Combo.addItem(gestor.Imagen[i].Titulo)
        self.ActualizarBotones(self.Combo.currentText())
```

Función Limpiar: Limpia todos los componentes del frame

```
def Limpiar(self):
        self.Combo.clear()
        self.lblImagen.clear()

        self.btnMiX.setEnabled(False)
        self.btnMiY.setEnabled(False)
        self.btnDMi.setEnabled(False)
```

Función ActualizarBotones: actualiza el estado de los filtros según la imagen del combo box

```
def ActualizarBotones(self,TextoActual):
        self.btnMiX.setEnabled(False)
        self.btnMiY.setEnabled(False)
        self.btnDMi.setEnabled(False)
        for i in range(len(gestor.Imagen)):
            if TextoActual == gestor.Imagen[i].Titulo:
                for j in range(len(gestor.Imagen[i].Filtros)):
                    if (gestor.Imagen[i].Filtros[j]=="MIRRORX"):
                        self.btnMiX.setEnabled(True)

                    elif(gestor.Imagen[i].Filtros[j]=="MIRRORY"):
                        self.btnMiY.setEnabled(True)

                    elif(gestor.Imagen[i].Filtros[j]=="DOUBLEMIRROR"):
                        self.btnDMi.setEnabled(True)
```

Mostrar Imagen: Como ejemplo para la muestra de imágenes en el frame tomare esta función ya que las demás (Mostrar ImagenX, ImagenY, ImagenXY) funcionan prácticamente de la misma manera

```
def MostrarImagen(self):
        tituloImagen=self.Combo.currentText()
        pixmap= QtGui.QPixmap('./Imagenes/'+tituloImagen+'.png').scaled(200,
200)
        self.lblImagen.setPixmap(pixmap)
        self.lblImagen.resize(200,200)
```

Librerías Utilizadas

Las librerías que se utilizaron para el desarrollo de esta practica fueron:

Imports locales:

- from Token import Token
- from Errores import Errores
- from Imagen import Imagen
- from Celda import Celda
- from Gestor import *

Imports TKINTER:

Estas librerías fuerin utilizadas para la carga del archivo

- from tkinter import *
- from tkinter import filedialog, Tk
- from tkinter.filedialog import askopenfilename
- from graphviz import *

Imports Interfaz PyQt5:

Estas librerías fueron utilizadas para la creación de la interfaz gráfica.

- from PyQt5 import QtCore, QtGui, QtWidgets

Descripción de Tokens

L = [a-zA-ZñÑ]

D=[0-9]

S=[‘{’, ’}’, ‘ [’, ‘]’, ‘;’, ‘=’, ‘,’]

Se = [‘@’@’@’@’]

Color = ‘#’ (L | D) {6}

Cadena = “(^”)*”

T = (‘T’ ‘R’ ’U’ ‘E’)

F = (‘F’ ‘A’’L’ ‘S’ ‘E’)

#	Token	Descripción	Tipo de dato	ER
1	Reservada	Este token representa una palabra reservada	String	**L+**
2	Cadena	Este token representa el ancho del Lienzo o cuadro de pixeles	String	**“(^”)*”**
3	X	Este token representa la posición en la columna del lienzo	Int [0,9]	**D+**
4	Y	Este token representa la posición en la fila del lienzo	Int [0-9]	**D+**
5	Color	Este token representa el alto del lienzo o cuadro de pixeles	String	**‘#’ (L | D) {6}**
6	Separador	Este token representa el número de filas del lienzo o cuadro de pixeles	String	**Se?**
7	Simbolos	Este token puede representar el principio o final de otro token	String	**S?**

Diagrama de Árbol