**Universidad Nacional de Costa Rica**


**Escuela de Informática**

**Curso:**
**Paradigmas de Programación**

**Profesor: Mag. Georges Alfaro S.**

**Proyecto #1**

**Estudiantes:**
**Roger Addiel Amador Villagra 116360595**
**David Josué Quesada Ordóñez 402340105**
**Grupo 1pm**

**II Ciclo 2018**

## Introducción

El problema consiste en construir un programa que pueda aplicar un conjunto de reglas de producción o sustitución a una hilera de entrada para producir un resultado determinado.

El conjunto de reglas a aplicar forma un sistema de reescritura o sustitución llamado **algoritmo de Markov.**

El programa desarrollado es similar al Yad Studio, pero con características diferentes, iniciando por el lenguaje en el que fue desarrollado el actual programa (Python) en comparación al Yad Studio (C++).

## Desarollo

Interfaz principal:

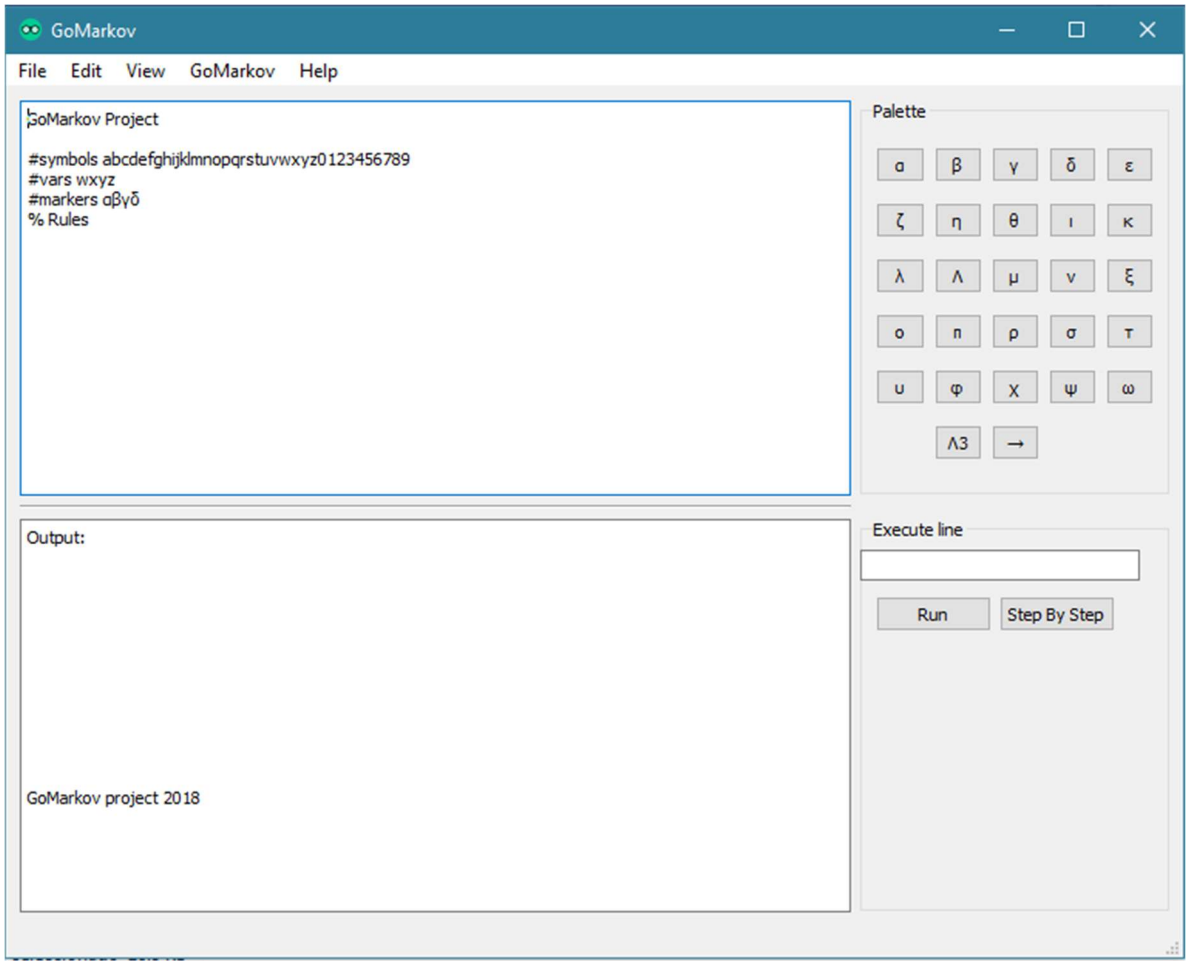



Imagen 1 – Interfaz principal

Como se observa en la imagen anterior, la interfaz cuenta con un submenú en la parte superior. Este submenú se divide 5 secciones: “File”, “Edit”, “View”, “GoMarkov” y “Help”. Estas secciones incluyen todo lo necesario que un editor como estos realice. Tales como: escribir, copiar, cortar, pegar, abrir, guardar, etc.

Primer cuadro de texto:

En el primer cuadro de texto se podrá escribir los símbolos, variables, marcadores y el conjunto de reglas.

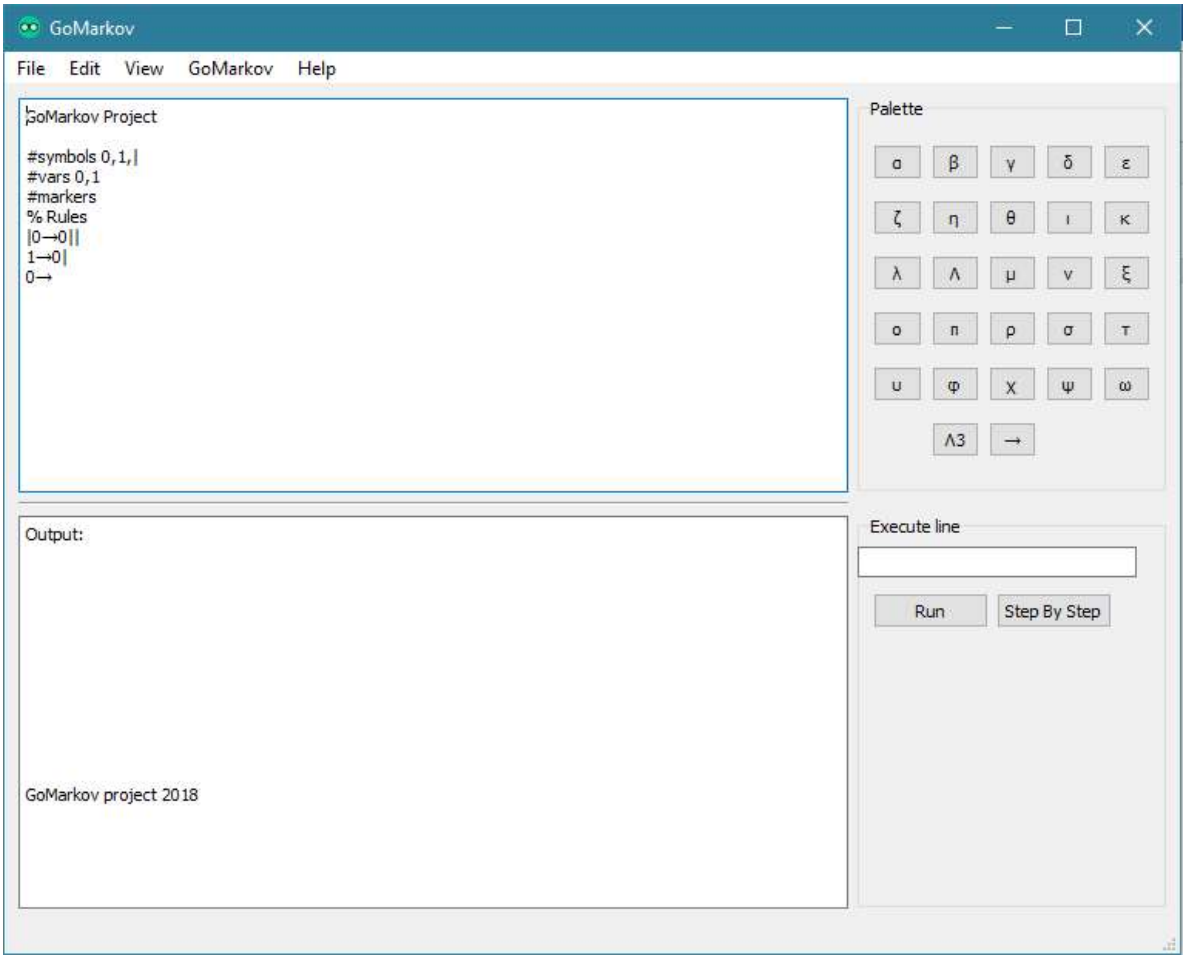



Imagen 2 – Primer cuadro de texto

Tal y como se observa en la imagen anterior, este cuadro de texto contiene los símbolos, variables, marcadores y reglas que necesito para el algoritmo en ejecución. En este caso, el algoritmo que pasa números binario al conjunto unario.

Línea de ejecución: En esta sección se permite poder poner la hilera a la que se estará evaluando.

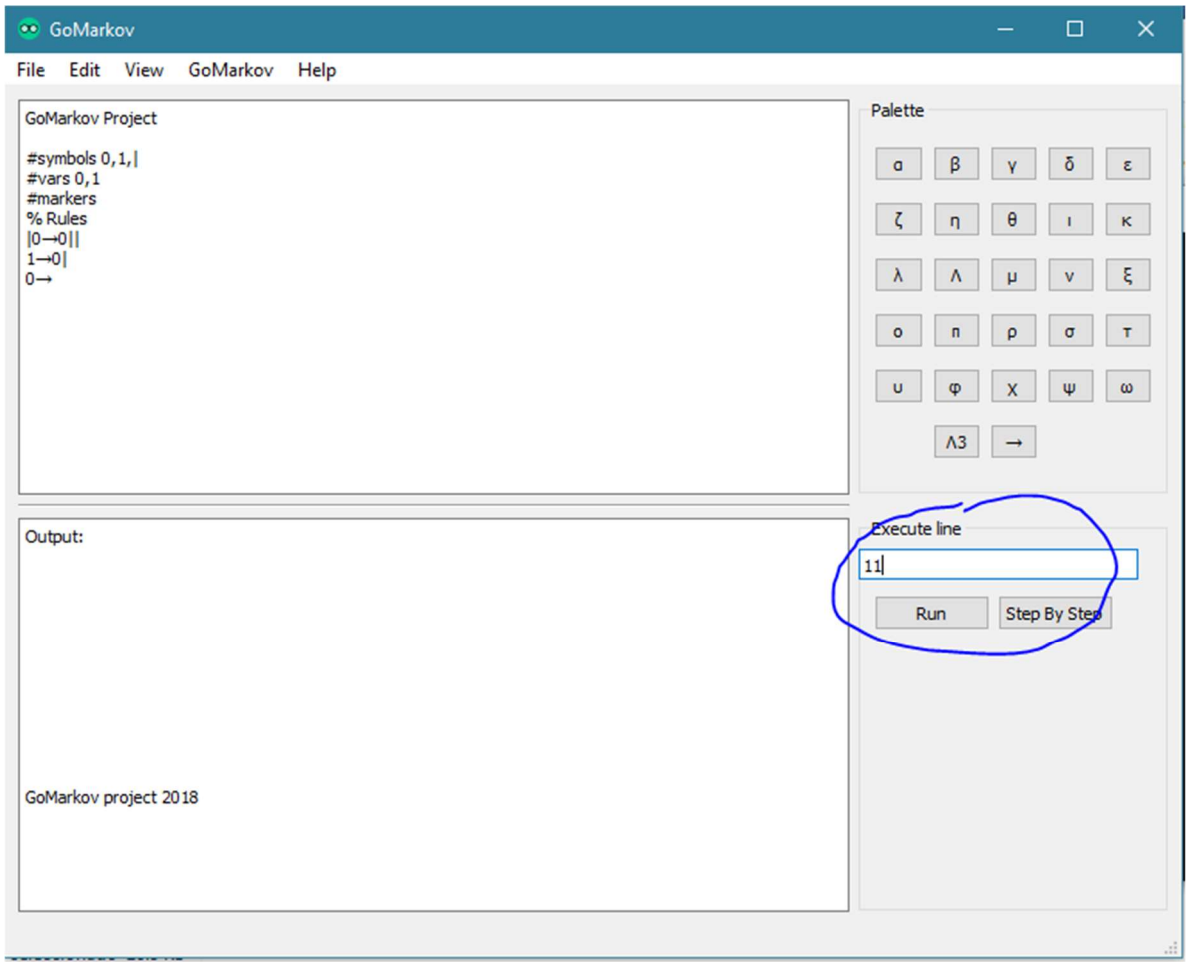



Imagen 3 – Línea de ejecución

Segundo cuadro de texto: En el segundo cuadro de texto se podrá observar nuestro resultado.

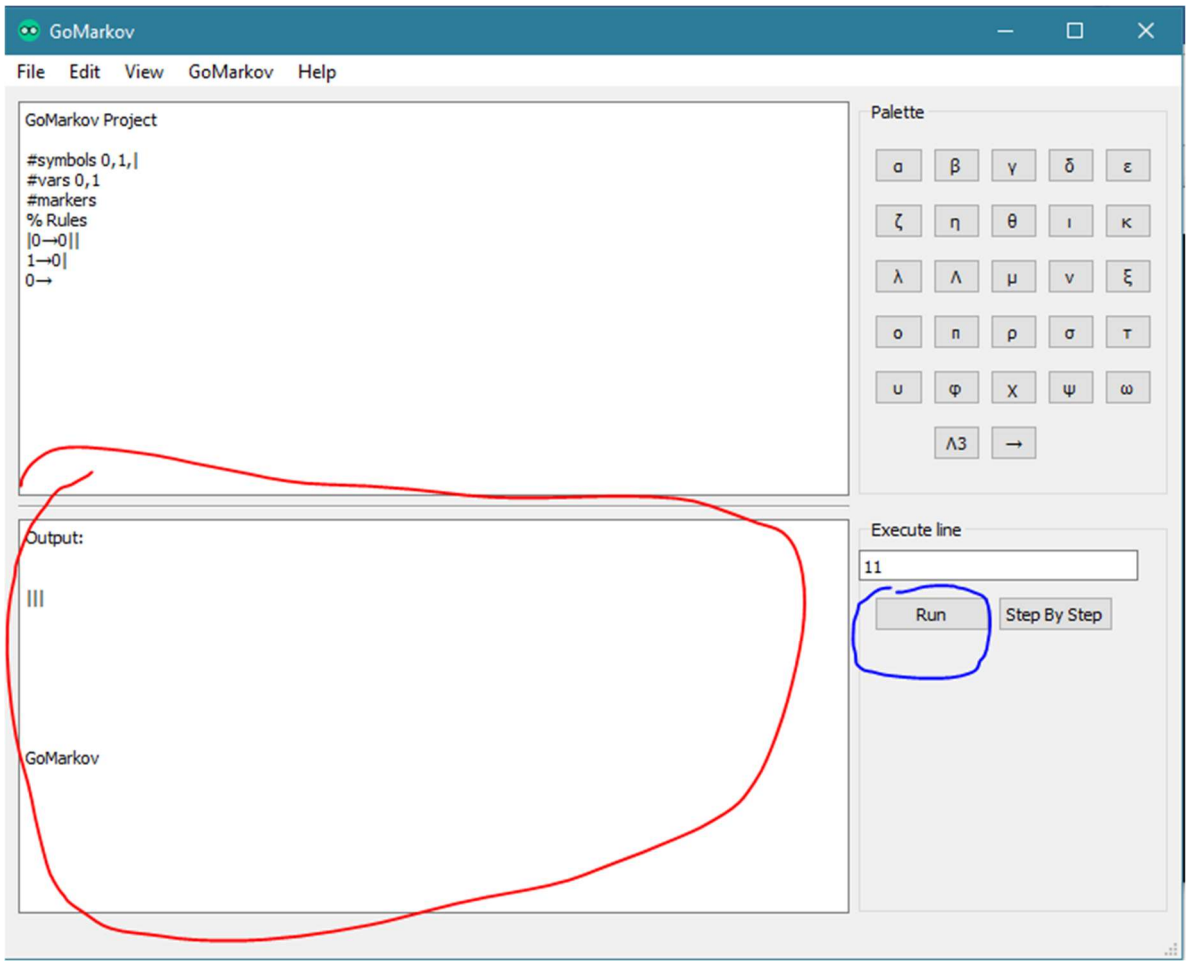



Imagen 4 – Segundo cuadro de texto

El segundo cuadro de texto es el cuadro que está encerrado en color rojo. Esta muestra la salida que se obtuvo al presionar el botón “Run” (encerrado en color azul). Veremos que sucede al presionar el botón “Step By Step”.

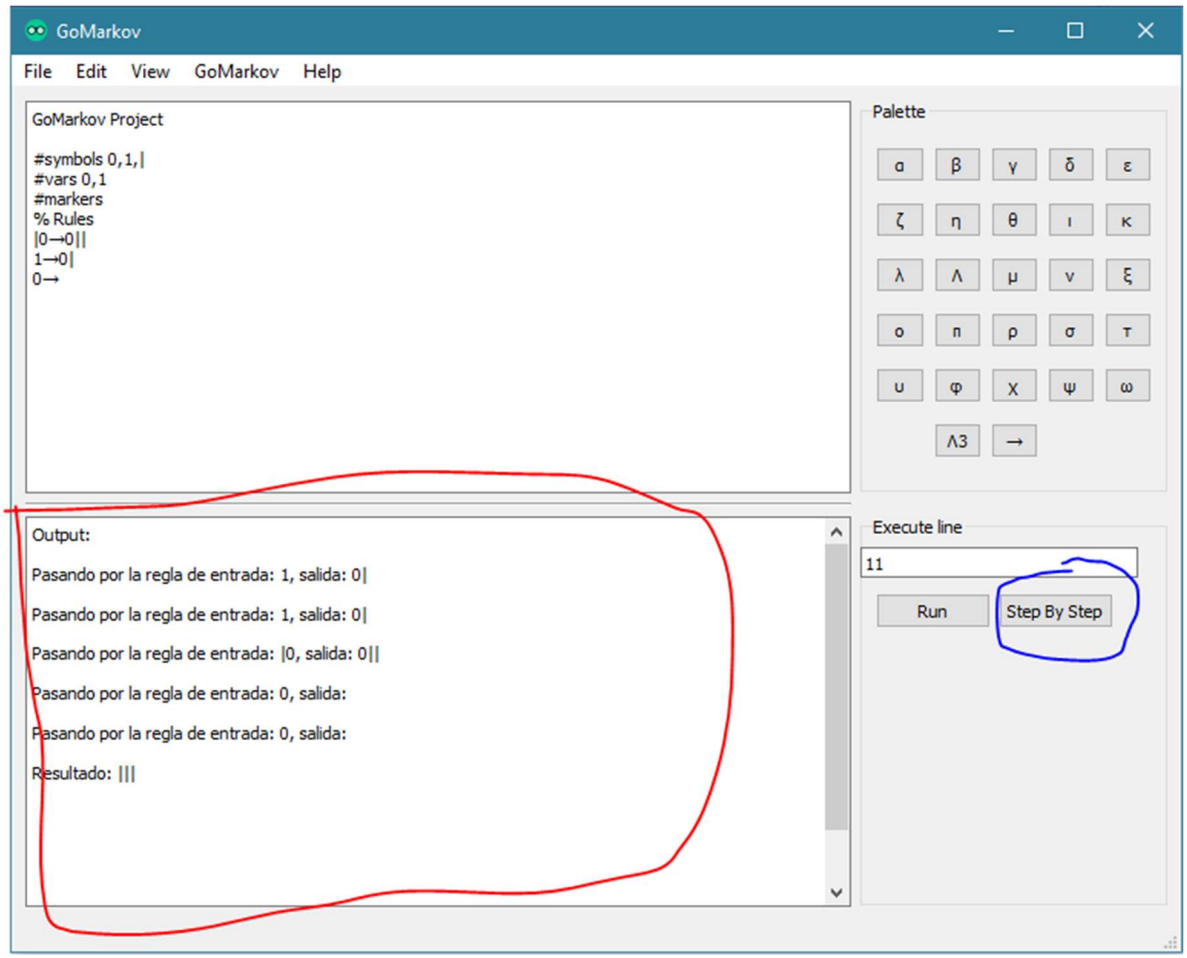


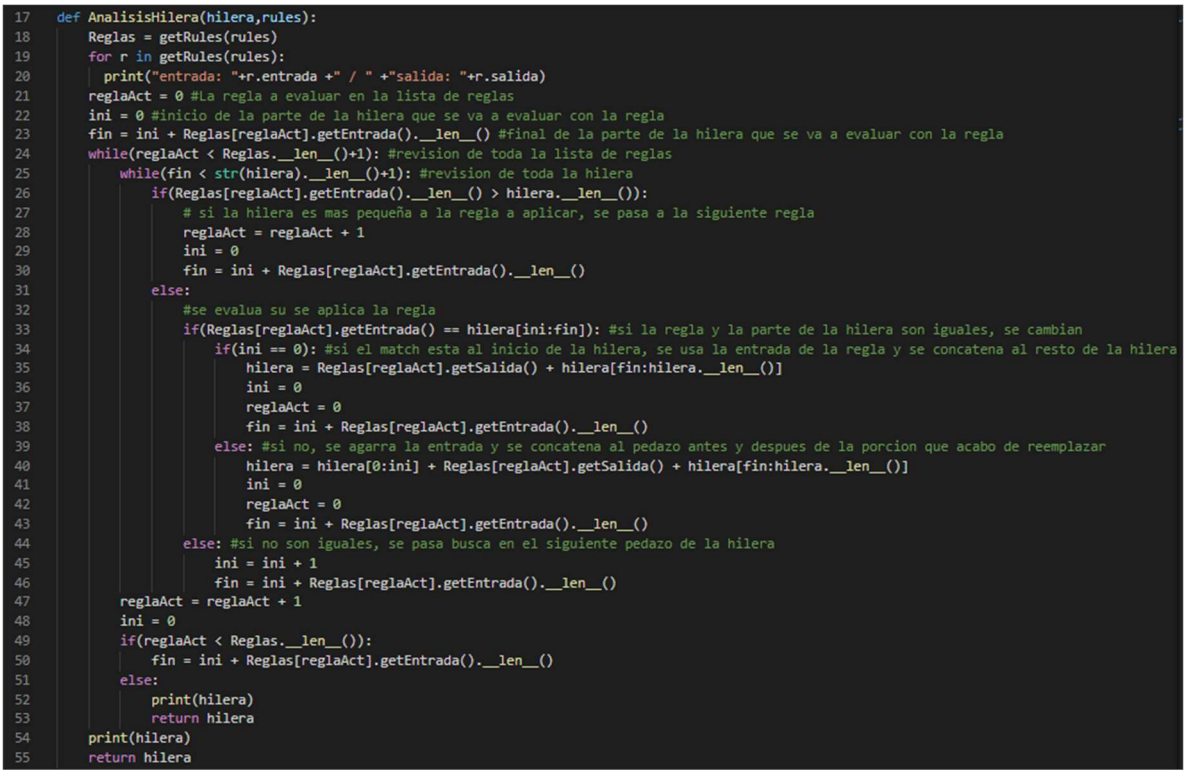

Imagen 5 – Opción Step By Step

En la imagen anterior se logra observar el resultado obtenido al presionar la opción “Step By Step”. Esta muestra paso por paso el algoritmo usado para determinar el resultado final.

Como se observa en la imagen 4 y en la imagen 5, ambos resultados son el mismo.

**Algoritmos principales:**

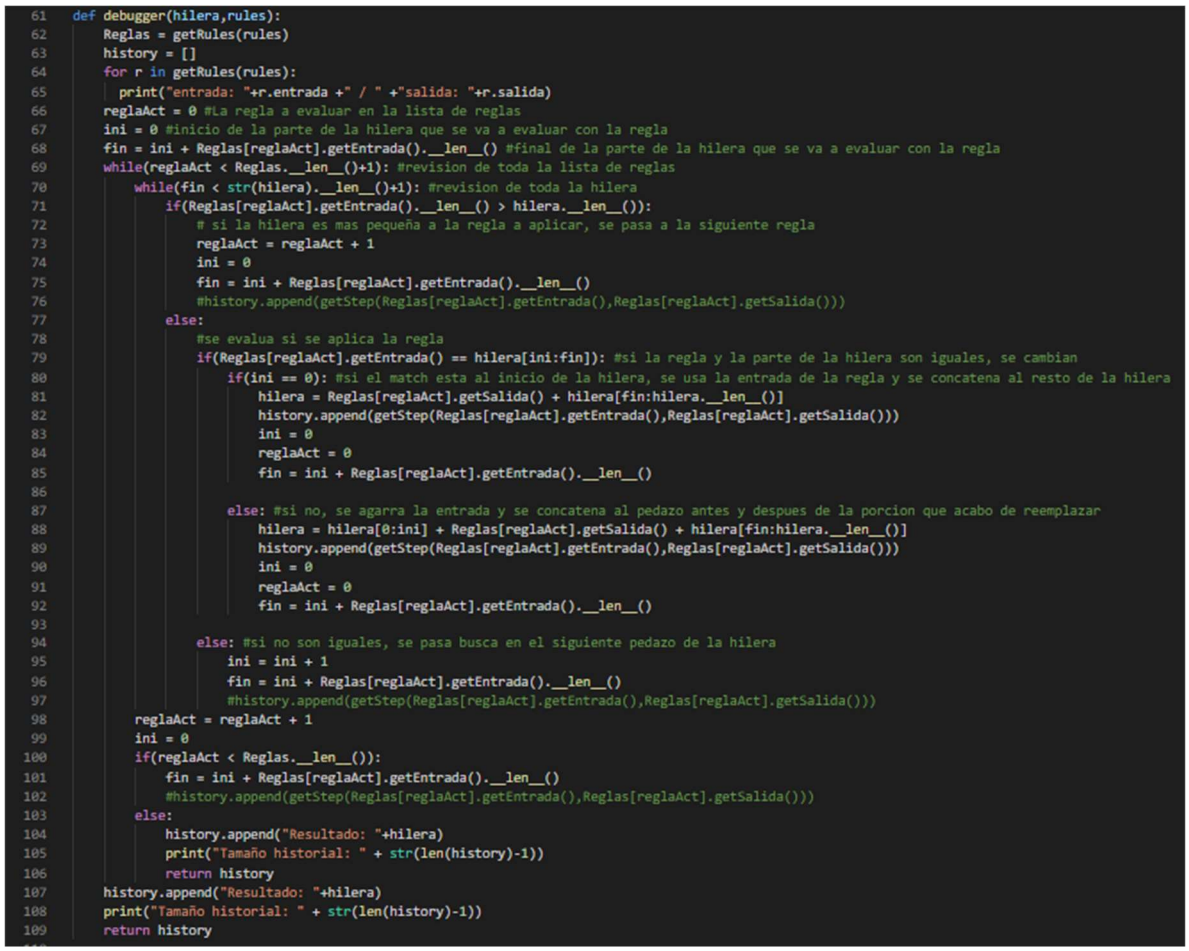
El algoritmo que determina el resultado obtenido se llama “AnalisisHilera”, mismo que veremos a continuación:



```
def AnalisisHilera(hilera,rules):
    Reglas = getRules(rules)
    for r in getRules(rules):
      print("entrada: "+r.entrada +" / " +"salida: "+r.salida)
    reglaAct = 0 #La regla a evaluar en la lista de reglas
    ini = 0 #inicio de la parte de la hilera que se va a evaluar con la regla
    fin = ini + Reglas[reglaAct].getEntrada().__len__() #final de la parte de la hilera que se va a evaluar con la regla
    while(reglaAct < Reglas.__len__()+1): #revision de toda la lista de reglas
        while(fin < str(hilera).__len__()+1): #revision de toda la hilera
            if(Reglas[reglaAct].getEntrada().__len__() > hilera.__len__()):
                # si la hilera es mas pequeña a la regla a aplicar, se pasa a la siguiente regla
                reglaAct = reglaAct + 1
                ini = 0
                fin = ini + Reglas[reglaAct].getEntrada().__len__()
            else:
                #se evalua su se aplica la regla
                if(Reglas[reglaAct].getEntrada() == hilera[ini:fin]): #si la regla y la parte de la hilera son iguales, se cambian
                    if(ini == 0): #si el match esta al inicio de la hilera, se usa la entrada de la regla y se concatena al resto de la hilera
                        hilera = Reglas[reglaAct].getSalida() + hilera[fin:hilera.__len__()]
                        ini = 0
                        reglaAct = 0
                        fin = ini + Reglas[reglaAct].getEntrada().__len__()
                    else: #si no, se agarra la entrada y se concatena al pedazo antes y despues de la porcion que acabo de reemplazar
                        hilera = hilera[0:ini] + Reglas[reglaAct].getSalida() + hilera[fin:hilera.__len__()]
                        ini = 0
                        reglaAct = 0
                        fin = ini + Reglas[reglaAct].getEntrada().__len__()
                else: #si no son iguales, se pasa busca en el siguiente pedazo de la hilera
                    ini = ini + 1
                    fin = ini + Reglas[reglaAct].getEntrada().__len__()
        reglaAct = reglaAct + 1
        ini = 0
        if(reglaAct < Reglas.__len__()):
            fin = ini + Reglas[reglaAct].getEntrada().__len__()
        else:
            print(hilera)
            return hilera
    print(hilera)
    return hilera
```

Imagen 6 – Algoritmo AnalisisHilera

Este es el algoritmo que se ejecuta al presionar el botón “run”. Además, a continuación, veremos la variante de este algoritmo para el “paso por paso”.



```
def debugger(hilera,rules):
    Reglas = getRules(rules)
    history = []
    for r in getRules(rules):
      print("entrada: "+r.entrada +" / " +"salida: "+r.salida)
    reglaAct = 0 #La regla a evaluar en la lista de reglas
    ini = 0 #inicio de la parte de la hilera que se va a evaluar con la regla
    fin = ini + Reglas[reglaAct].getEntrada().__len__() #final de la parte de la hilera que se va a evaluar con la regla
    while(reglaAct < Reglas.__len__()+1): #revision de toda la lista de reglas
        while(fin < str(hilera).__len__()+1): #revision de toda la hilera
            if(Reglas[reglaAct].getEntrada().__len__() > hilera.__len__()):
                # si la hilera es mas pequeña a la regla a aplicar, se pasa a la siguiente regla
                reglaAct = reglaAct + 1
                ini = 0
                fin = ini + Reglas[reglaAct].getEntrada().__len__()
                #history.append(getStep(Reglas[reglaAct].getEntrada(),Reglas[reglaAct].getSalida()))
            else:
                #se evalua si se aplica la regla
                if(Reglas[reglaAct].getEntrada() == hilera[ini:fin]): #si la regla y la parte de la hilera son iguales, se cambian
                    if(ini == 0): #si el match esta al inicio de la hilera, se usa la entrada de la regla y se concatena al resto de la hilera
                        hilera = Reglas[reglaAct].getSalida() + hilera[fin:hilera.__len__()]
                        history.append(getStep(Reglas[reglaAct].getEntrada(),Reglas[reglaAct].getSalida()))
                        ini = 0
                        reglaAct = 0
                        fin = ini + Reglas[reglaAct].getEntrada().__len__()

                    else: #si no, se agarra la entrada y se concatena al pedazo antes y despues de la porcion que acabo de reemplazar
                        hilera = hilera[0:ini] + Reglas[reglaAct].getSalida() + hilera[fin:hilera.__len__()]
                        history.append(getStep(Reglas[reglaAct].getEntrada(),Reglas[reglaAct].getSalida()))
                        ini = 0
                        reglaAct = 0
                        fin = ini + Reglas[reglaAct].getEntrada().__len__()

                else: #si no son iguales, se pasa busca en el siguiente pedazo de la hilera
                    ini = ini + 1
                    fin = ini + Reglas[reglaAct].getEntrada().__len__()
                    #history.append(getStep(Reglas[reglaAct].getEntrada(),Reglas[reglaAct].getSalida()))
        reglaAct = reglaAct + 1
        ini = 0
        if(reglaAct < Reglas.__len__()):
            fin = ini + Reglas[reglaAct].getEntrada().__len__()
            #history.append(getStep(Reglas[reglaAct].getEntrada(),Reglas[reglaAct].getSalida()))
        else:
            history.append("Resultado: "+hilera)
            print("Tamaño historial: " + str(len(history)-1))
            return history
    history.append("Resultado: "+hilera)
    print("Tamaño historial: " + str(len(history)-1))
    return history
```

Imagen 7 – Algoritmo debugger

## ¿Qué hace el programa?

Este programa reconoce cualquier tipo de algoritmo y su funcionalidad es similar a la del Yad Studio.

## ¿Qué no hace el programa?

De los 10 algoritmos a probar, sólo 3 dan error. Sin embargo, el lenguaje de estos algoritmos está correctamente definidos, pero el programa no los interpreta bien por el algoritmo que los interpreta, ya que es un poco diferente a como lo manejan otros programas, por ejemplo: Yad Studio.

Los 3 algoritmos que no identifica bien son:

1- Duplicar hilera
2- Suma de decimales
3- Multiplicación de binarios

Para el fácil manejo de estos algoritmos, los incluimos en una carpeta llama “algoritmos”, para que sea de fácil manejo al momento de probarlos.

**¿Qué hay que cambiar para que estos algoritmos funcionen?**

Para que estos algoritmos funciones hay que mejorar su lenguaje, ya que el algoritmo utilizado para leer e interpretar la hilera mediante el conjunto de reglas, se comprueba que trabaja al 100%, pero de una manera diferente a como está en el lenguaje de esos tres algoritmos.

## Conclusiones

El proyecto elaborado fue un gran reto para nosotros; iniciando por el lenguaje de programación utilizado, ya que, ninguno de los dos los habíamos usado anteriormente. Sin embargo, eso nos hizo investigar e interesarnos por aprender este lenguaje.

La realización de un programa que interpretara reglas dio un salto a lo que habitualmente se hace. Esto nos hizo trabajar duro para primero entender lo que es estaba haciendo y también evaluarlo de una manera correcta.