Como no he encontrado ninguna librería para dibujar grafos que me fuera útil he decidido intentar construir el grafo a partir de rectangle shapes , lines y labels.

ScrollPane tiene eventos por defecto para las teclas izquierda y derecha por lo que hay que añadirle un event filter para que atienda a nuestro evento antes que al que esta por defecto. Aun con esto el evento por defecto no desaparece.

grafo.addEventFilter(KeyEvent.KEY\_RELEASED,

event ->handleKeyAction(event));

nodo raíz situado en el punto medio de la suma de los 17 nodos -1 que es la raíz ya que en el peor caso la raiz tiene número de nodos-1 hijos.

## Clase Nodo V0.1

Contiene once atributos:

**private String símbolo**: es el símbolo que se va a representar en este Nodo.

**private Rectangle rectangle**: es el rectángulo sobre el que va a estar el símbolo y representa al nodo.

**private Nodo parent**: es el nodo padre.

**private LinkedList<Nodo> children**: son todos los hijos del nodo, estos hijos representan los nodos que se van a ejecutar a causa de este nodo.

**private Double posX**: posición en el eje X del nodo.

**private Double posY**: posición en el eje Y del nodo.

**private Line line**: línea que va a juntar el nodo padre con este nodo.

**private Label label**: etiqueta que va a mostrar el símbolo del nodo.

**private HashMap<String , Nodo> hermanos**: mapa de nodos que serán los nodos en la pila que suelen aparecer al lado del nodo ejecutado. Están insertados en función de su símbolo.

**private Nodo LeftSibling**: hermano izquierdo ya ejecutado del nodo.

**private Rectangle rectRgla**: rectángulo que engloba a todos los nodos de la regla.

**public Nodo(String simbolo,Nodo parent)**

En el constructor se asignan el símbolo y el padre recogidos y se inicializa los hermanos, los hijos y el rectángulo del nodo con un tamaño 50x50.

## Clase Nodo V0.2

**Atributos nuevos:**

**private Boolean isTerminal:** true si es terminal y false si no.

**Constructor:**

Se ha añadido el campo de Boolean isTerminal para recoger si el nodo es terminal y se asigna al atributo correspondiente.

## Clase Nodo V0.3

**Atributos nuevos:**

**private int font:** contiene el tamaño de fuente de la etiqueta.

**private ArrayList<Line> lines:** contiene las líneas que unen un nodo con sus hijos en el caso del traductor ascendente.

**Constructor:**

Se ha añadido el campo de int font a la cabecera y se asigna al atributo correspondiente. Si la letra se puede salir del recuadro se aumenta el recuadro. Se inicializa la lista de líneas.

## Clase Nodo V0.4

**Atributos nuevos:**

**private Double widthRectRgla:** guarda el ancho del rectángulo que engloba los nodos de la misma regla.

**private String value:** guarda el valor del símbolo.

**Constructor:**

Se añade el campo String value a la cabecera y se le asigna al atributo value.

## Clase Nodo V0.5

Constructor:

Se aumenta el ancho de los nodos y se añade borde Ya no cambia de tamaño dependiendo del tamaño de letra.

## Clase Grafo V0.1

Tiene tres atributos:

**private HashMap<Integer,Nodo> nodos**: es un mapa que guarda los nodos ejecutados en función de paso en el que aparecieron .

**private FicheroXML ejemplo**: guarda los datos obtenidos del fichero XML que tiene los datos del traductor.

**private double posXAnterior**: guarda la posición en el eje X del ultimo nodo ejecutado que se ha insertado. Se utiliza para evitar que los nodos se solapen.

**public Grafo(FicheroXML xml)**

El constructor recoge el fichero XML y lo asigna, además, inicializa el mapa de nodos y la posición anterior como la mitad del numero de nodos -1 por el tamaño de los nodos. (Pendiente de revisión, ya que si el nodo raíz tiene una regla podemos saber cuántos hijos tendrá y calcular mejor su posición).

**public Nodo insertarNodo(Nodo parent,Pane panelPadre,String simbolo,Double posX,Double posY)**

Recoge el nodo que va a ser el padre, el panel donde se va a dibujar, el símbolo correspondiente al nodo y las posiciones en el eje X e Y.

Crea el nodo, asigna el color y la posición del rectángulo y la guarda. Crea la etiqueta con el símbolo, le asigna la posición y guarda la etiqueta. Después guarda el nodo en el mapa de nodos. A continuación, si no es nodo raíz crea la línea que junta el padre con el nodo, la asigna al nodo y pinta en el panel el rectángulo, la etiqueta y la línea. Si el nodo es raíz solo dibuja el rectángulo del nodo y la etiqueta.

Por último, devuelve el nodo creado.

**public HashSet<Nodo> insertarNodoNotExec(Nodo parent,Pane panelPadre,Regla regla,Double posX,Double posY,Nodo hijo)**

Recoge el padre de los nodos a insertar , el panel donde se van a dibujar, la regla de donde vienen sus símbolos, las posiciones en el eje X e Y y el nodo ejecutado que va a permanecer como primer hermano.

Inicializa el conjunto de nodos que se va a devolver y crea un array con todos los símbolos de la regla. A continuación, por cada uno de los símbolos si no corresponden al símbolo del primer hermano, se crea un nodo con ese símbolo, se le asigna el color, una opacidad del 50%, una posición en el eje X que será la posición x introducida mas un pequeño espacio de 10 px mas el ancho del nodo y la posición en el eje Y introducida. Después se crea la etiqueta, se le asigna su posición y se guarda en el nodo. Por último, se guarda el nodo en el conjunto que se va a devolver, en el mapa de hermanos y se dibuja en el panel el rectángulo y la etiqueta.

Devuelve el conjunto de nodos no ejecutados insertados.

**public HashSet<Nodo> insertarNodoNotExec(Pane panelPadre,Collection<Nodo> nodosHermanos,Double posX,Double posY,Nodo hijo)**

Recoge el panel sobre el que se pinta, la colección de nodos no ejecutados que se va a repintar, la posición en los ejes X e Y y el primer hermano.

Funciona igual que el anterior salvo porque no se crea el nodo ni la etiqueta, sino que se recolocan.

**public void eliminarNodo(Nodo nodoElim,Pane panelPadre,int number)**

Recoge el nodo a eliminar, el panel sobre el que se va a pintar y el paso en el que está el traductor.

Primero se borran el rectángulo, la línea y la etiqueta del panel. Después, si no es un nodo raíz se elimina el nodo de los hijos del padre, y si el nodo a eliminar es el primer hijo de la raíz se le pone una posición X por defecto, si no lo es se le la posición X anterior. Por último, se elimina el nodo del mapa de nodos.

**public LinkedList<Nodo> eliminarNodoNotExec(Nodo nodoElim,Pane panelPadre)**

Recoge el nodo ejecutado que corresponde al primer hermano y el panel del que se va a borrar.

Inicializa la lista de nodos que se va a devolver. A continuación, por cado nodo de los hermanos no ejecutados del nodo ejecutado se añade a una lista. Si esta lista no esta vacía por cada uno de los nodos de la lista se borra el rectángulo y la etiqueta del panel.

Devuelve la lista de nodos no ejecutados que se han borrado.

**public void construir(int contador,int pasoSolicitado,Pane panelPadre )**

Recoge el paso desde el que se va a construir, el paso hasta el que se va a construir, no incluido este, y el panel sobre el que se va a pintar.

Por cada paso ,desde el paso inicial introducido(contador) hasta el solicitado, si es el primer paso (correspondiente a la raíz del árbol) se inserta un nodo y se pone la posición anterior a un valor por defecto (0 o 10 , está por decidir), sino, se recupera el padre del nodo que se va a insertar, si el padre no tiene hijos, se inserta el hijo en la posición X anterior y la posición Y será el doble de la altura del rectángulo más la posición Y del padre, se añade el nodo a los hijos del padre y se obtiene la regla correspondiente al nodo, si esta regla nos nula se insertan los nodos no ejecutados correspondientes al lado del nodo ejecutado. Después se inserta el rectángulo correspondiente a la regla con el ancho del rectángulo del nodo más el de todos los nodos no ejecutados hermanos y se actualiza la posición anterior a la del nodo ejecutado insertado.

Si el padre ya tenía hijos, inserta el nodo en la posición X anterior más el ancho del rectángulo más una distancia de separación por defecto y la posición Y será el doble de la altura del rectángulo más la posición Y del padre y se añade el nodo a los hijos del padre. Después se eliminan los nodos no ejecutados del hermano izquierdo y se insertan los de este nodo. A continuación, se obtiene el rectángulo de la regla que tiene el hermano ejecutado izquierdo y si el primer hermano ejecutado no está en la posición por defecto se le añade al ancho del rectángulo la distancia del ultimo nodo no ejecutado del anterior nodo ejecutado más el ancho del nodo ejecutado más el ancho de los nodos hermanos no ejecutados más las distancias de separación menos la posición del primer hermano ejecutado. Si el nodo está en la posición por defecto se le añade el ancho del nodo más el ancho de los hermanos no ejecutados más las distancias de separación. Se guarda el rectángulo en la regla y se actualiza la posición anterior.

**public void eliminar(int contador,int pasoSolicitado,Pane panelPadre )**

Recoge el paso inicial desde el que se va a empezar a borrar, el paso hasta el que se borra y el panel del que se van a borrar los nodos.

Por todo el intervalo de pasos desde contador -1 hasta el paso final si el nodo que se va a eliminar no es nodo raíz y es el primero de los hijos se eliminan los nodos hermanos no ejecutados, después se busca el nodo no ejecutado del hermano izquierdo que corresponde al símbolo del nodo a eliminar, si existe dicho nodo se vuelve a pintar en el sitio que le corresponde. Por último, se elimina el nodo a eliminar.

**public Nodo NodoNotExec(Nodo nodo)**

Recoge el nodo que se va a buscar.

Si el nodo no es la raíz se busca en los hermanos no ejecutados del hermano izquierdo.si no se encuentra dicho nodo devuelve null.

## Clase Grafo V0.2

Igual que la V0.1 salvo en el método construir donde se asigna tres eventos al rectángulo que engloba los símbolos de la misma regla en el grafo:

-Click: Cambia la forma de la regla en el panel de la gramática.

-Entered (el cursor entra en el cuadrado): cambia el color del recuadro y dibuja el rectángulo correspondiente a la misma regla en la gramática.

-Exited (el cursor entra en el cuadrado): cambia el color del recuadro al original y borra el rectángulo correspondiente a la misma regla en la gramática.

## Clase Grafo V0.3

Introducción de nuevos atributos:

**private CadenaEntrada cadena:** cadena de entrada.

**private HashMap<String,Integer>stepProcess:** mapa que guarda en que paso se procesan los elementos de la cadena.

**private int contador:** representa el paso que se va a ejecutar.

**private Pane panelPadre:** panel sobre el que se dibuja el grafo.

Constructor:

Se añaden los campos de panelPadre y cadenaEntrada a la cabecera del método y se asigna el valor de los nuevos atributos en el cuerpo. Además, se llama al método que recoge en que paso se procesan los elementos de la cadena y al método que añade el evento de ir a esos pasos al pulsar en la cadena.

Métodos:

**public void obtainStepsProcess():**

Por cada estado de la cadena se separan los elementos procesados de los pendientes. Si hay elementos procesados se guarda el ultimo, y si el elemento procesado es diferente del ultimo elemento procesado guardas en el mapa el paso en el que se ha procesado en función del elemento procesado y guardas el elemento procesado para compararlo con los siguientes elementos. Después, aumentas el paso en el que estas.

**public void addHandlingListennerChain():**

obtienes los rectángulos de la cadena y por cada uno le añades un evento que se activa dando click. Cuando se activa el evento obtengo el rectángulo clickeado y si su id es EOF el paso al que ir es el último, si no, uso el id del rectángulo para obtener el paso. Si el paso es mayor que el contador construyo hasta ese elemento, si es menor, elimino hasta llegar a ese paso.

El evento no funciona si clickeas la etiqueta.

Se elimina el campo panelPadre de todas las llamadas que lo usaban ya que es un atributo asignado en el constructor.

En la función **eliminarNodo(Nodo nodoElim,int number)** se supone number como el paso a eliminar y no number -1.

En los métodos **construir** y **eliminar** se elimina contador de la cabecera ya que se ha introducido como atributo en la clase. Además, en ambos métodos, al final, se actualiza el contador y la cadena, y por último se devuelve el paso en el que se ha quedado.

Se han añadido el get y el set del contador.

## Clase Grafo V0.4

Atributos nuevos:

**private Configuracion config:** guarda la configuración de los elementos de la aplicación (colores, tamaños de letra y zoom)

**Constructor:** recoge la configuración en la cabecera y la asigna al atributo correspondiente.

**Métodos:**

**insertarNodo:** Cuando crea un nodo nuevo se le añade a la llamada si el símbolo es terminal. Si es terminal se les añade los colores correspondientes a los terminales de fondo al recuadro y el de letra a la etiqueta, si no, los de los no terminales. Siempre se le añade el tamaño de letra del árbol.

**public Boolean isTerminal(String symbol):**

Recoge un símbolo para comprobar si es terminal y devuelve verdadero si lo es y falso si no.

Busca el símbolo en el mapa de símbolos del ejemplo (FicheroXML) y cuando lo encuentra devuelve si es terminal.

**insertarNodoNotExec:** añade a la etiqueta el tamaño correspondiente y determina si el símbolo es terminal, después añade los colores correspondientes al rectángulo y a la etiqueta.

**public void updateGraph():**

Actualiza el grafo después de haber cambiado la configuración.

Elimina el grafo hasta el paso cero y después vuelve a construirlo hasta el paso en el que estaba.

## Clase Grafo V0.5

Atributos:

**private double posYAnterior:** guarda la posición en el eje y para el traductor ascendente.

**private double heigth:** guarda la altura de los rectángulos para el traductor ascendente.

**private double width:** guarda la anchura de los rectángulos para el traductor ascendente.

**private String tipoTraductor:** guarda el tipo de traductor que se va a representar.

**private int nivelAnterior:** guarda el ultimo nivel del árbol en el que se ha dibujado.

Construtor:

Se añade el tipo de traductor a la cabecera. Se inicializa **posYAnterior** a 300 **(revisar), tipoTraductor** al valor recogidoy **nivelAnterior**, **heigth** y **width** a 0.

Métodos:

**public Nodo insertarNodoA(String parents,String simbolo,Double posX,Double posY):**

Funciona igual que **insertarNodo** con la diferencia que en vez de recoger un nodo padre recoge la String con el símbolo y los elementos con los que se va a relacionar. Al final del método en vez de insertar una línea inserta una línea por cada elemento con el que se relaciona.

**construir(int pasoSolicitado ):**

Se va a dividir en dos métodos construirAsc y construirDesc. Elegirá cual se ejecutará en función del tipo de traductor.

**public int construirAsc(int pasoSolicitado):**

Por cada paso desde contador hasta el paso solicitado sin incluir obtendrá el nivel en el árbol del símbolo que se va a procesar.

Si es el primer paso se obtendrá el símbolo de la lista de pasos y se insertará en la posición (10 ,300). Se actualiza el ultimo nivel, se guarda el ancho y el alto del nodo y se actualiza la última posición X.

En otro caso:

Si el nivel anterior es mayor que el actual la posición Y del nuevo nodo será la resta del doble de la altura del nodo más 10 por la diferencia de los niveles.

Si el nivel anterior es menor que el actual la posición Y del nuevo nodo será la suma del doble de la altura del nodo más 10 por la diferencia de los niveles.

Si el símbolo no está conectado a ningún otro, su posición X será la anterior mas el ancho de un nodo más 10. Y se insertara el nodo sin padre.

Si el símbolo esta conectado a otros se obtiene el primero de esos nodos y se obtiene el numero de elementos con los que están conectados.

Si tiene más de un hijo su posición X será la posición X de su primer hijo más la media de la suma de la anchura de sus hijos mas 10 por cada hijo menos el primero.

Si solo tiene un hijo su posición X será la misma que la de este.

Por ultimo se inserta el nodo con la String con el símbolo y los elementos con los que se va a relacionar.

Se actualiza el nivel.

Devuelve el paso solicitado.

**public int construirDesc(int pasoSolicitado):**

Hará lo mismo que construir hacia hasta ahora.

**public int eliminar(int pasoSolicitado ):**

Se va a dividir en dos métodos eliminarAsc y eliminarDesc. Elegirá cual se ejecutará en función del tipo de traductor.

**public void eliminarDesc(int pasoSolicitado):**

Hará lo mismo que eliminar hacia hasta ahora.

**public void eliminarAsc(int pasoSolicitado ):**

Por cada paso desde el contador al solicitado se obtiene el nodo a eliminar.

Si es el primer nodo introducido se borra del panel y se actualiza la posición X a 10, la posición Y a 300 y el nivelAnterior a 0.

Si el nodo a eliminar no tiene nodos conectados se borra el nodo y se actualiza la posición X, la posición Y y el nivelAnterior al valor que tenía en el paso anterior.

Si tiene nodos conectados además de lo anterior elimina todas las líneas que corresponden al nodo.

Por ultimo se elimina el nodo.

## Clase Grafo V0.6

**Atributos:**

**private HashMap<Integer, Integer> sibling**: guarda el hermano izquierdo de un nodo según el paso en el que se procesan.

**private HashMap<Integer,Double>posXAnteriores:** guarda la posición en el eje X de cada nodo para poder recuperarla cuando se elimine un nodo.

**Constructor:**

Se inicializan los mapas sibling y posXanterior. Además, se llama a la función **englove**.

**Métodos:**

**insertarNodo(Nodo parent,String simbolo,Double posX,Double posY,String value):**

Se ha añadido a la cabecera el campo value y se le pasa al constructor del nodo.

**insertarNodoA(String parents,String simbolo,Double posX,Double posY,String value):**

Se ha añadido a la cabecera el campo value y se le pasa al constructor del nodo.

**public void assignSiblings(String[] siblings):**

Asigna los hermanos derechos a los nodos que producen otro nodo.

Recoge el array de String que contiene el índice de los nodos productores.

Por cada índice si es el primero lo guarda en el mapa sibling con valor nulo si no lo guarda con el índice del nodo (el del hemano derecho).

**public void assignAllSiblings():**

Recoge los hermanos derechos de todos los nodos si tienen.

Por cada paso si tiene nodos conectados recoge los índices de los hermanos derechos de dichos nodos.

**public void assingRectanglesEvents(Rectangle rectReg):** Se asignan los eventos correspondientes a los recuadros que engloban los nodos de una misma regla. Estos eventos hacen lo mismo que hacían cuando se asignaban en construir.

**construirAsc(int pasoSolicitado):**

Al insertarNodoA se le pasa el valor del nodo también.

Si es el primer paso guarda la posición X, se crea el rectángulo que va ha englobar la regla, se coloca y se le añade la opacidad al 50%. Después, Se asigna el rectángulo al nodo, se dibuja en el panel, se le asigna a una regla en la gramática, se le asignan los eventos y se guarda el ancho del rectángulo en el nodo.

Si no es el primero

Si no está conectado a otros nodos se guarda la posición X, se asigna al nodo su hermano derecho y se inicializa a null el rectángulo, si el hermano derecho no es nulo obtengo el rectángulo que engloba los nodos de la misma regla.

Si el rectángulo existe se le amplia la anchura para que englobe el nuevo nodo y se asigna al nodo el rectángulo y su nueva anchura.

Si no existe se crea.

Si está conectado se guarda la posición X, se asigna al nodo su hermano derecho y se inicializa a null el rectángulo, si el hermano derecho no es nulo obtengo el rectángulo que engloba los nodos de la misma regla.

Si el rectángulo existe se le amplia la anchura para que englobe el nuevo nodo y se asigna al nodo el rectángulo y su nueva anchura.

Si no existe se crea.

En cada paso al terminar se actualizan los valores de los nodos. (se hace igual para construirDesc).

**eliminarAsc(int pasoSolicitado ):**

Si es el primer hermano se borra el rectángulo que engloba la regla.

Si no y no es el ultimo paso se recupera el ancho del rectángulo antes de insertar el nodo a eliminar (el ancho de rectángulo que tiene guardado el hermano derecho).

**public void updatedValues(List<Simbolo> symbolsUpdated):**

Actualiza todos los valores nuevos de los nodos.

Por cada símbolo con un valor nuevo se obtiene el nodo que corresponde a cada símbolo y se le asigna el valor nuevo.

## Clase Grafo V0.7

Métodos:

**insertarNodo(Nodo parent,String simbolo,Double posX,Double posY,String value):**

Cambia la forma de calcular la posición de la etiqueta para que se mantenga lo más centrada posible aun cuando cambie el tamaño.

**insertarNodoA(String parents,String simbolo,Double posX,Double posY,String value):**

Cambia la forma de calcular la posición de la etiqueta para que se mantenga lo más centrada posible aun cuando cambie el tamaño.

**insertarNodoNotExec(Nodo parent,Regla regla,Double posX,Double posY,Nodo hijo):**

Cambia la forma de calcular la posición de la etiqueta para que se mantenga lo más centrada posible aun cuando cambie el tamaño.

**insertarNodoNotExec(Collection<Nodo> nodosHermanos,Double posX,Double posY,Nodo hijo):**

Cambia la forma de calcular la posición de la etiqueta para que se mantenga lo más centrada posible aun cuando cambie el tamaño.

**construirAsc(int pasoSolicitado):**

Aumenta el margen aumentando la posición X antes de iniciar.

**construirDesc(int pasoSolicitado):**

Aumenta el margen aumentando la posición X antes de iniciar.

**public Double widthNoExecuteNodes(Nodo nodo):**

Recibe un nodo y calcula el ancho total de los hermanos.

## Clase Gramática V0.1

Atributos:

**private FicheroXML ejemplo:** Datos recabados del archivo XML.

**private ArrayList<Regla> reglas:** Lista con las reglas de la gramática.

**private int posYAnterior:** Posición de la regla anterior en el eje Y para poder situar las reglas de forma correcta.

**private HashMap<Rectangle,Regla>relRectRegla:** Mapa que contiene como se relacionan los rectángulos del grafo con las reglas de la gramática.

**private HashMap<String,Rectangle> ruleRectGramm:**  Mapa que relaciona los rectángulos y las reglas de la gramática.

**private HashMap<String,ArrayList<Label>> reglaLabel:** Mapa que relaciona cada regla con sus etiquetas.

**private HashMap<String,Boolean>formaReglas:** Mapa que relaciona las reglas con su forma. True si tienen la forma larga con las acciones y false si tiene la forma corta.

**private HashMap<String,Regla> idRegla:** Mapapara obtener la misma regla pero con los símbolos que la corresponde ya que las reglas de la lista de la gramática no los tiene.

**private Pane panelPadre:** panel donde se va a dibujar la gramática.

Constructor:

**public Gramatica(FicheroXML ejemplo,Pane panelPadre):** Inicializa los mapas y recoge y asigna el FicheroXML y el panelPadre.

Métodos:

**public ArrayList<Label> formarReglaCorta(Regla regla):**

Recoge la regla y devuelve las etiquetas que forman la forma corta de la regla.

Obtiene la regla con símbolos, por cada símbolo crea una etiqueta con ese símbolo y lo añade al resultado.

**public ArrayList<Label> formarReglaLarga(Regla regla):**

Recoge la regla y devuelve las etiquetas que forman la forma larga de la regla con las acciones.

Obtiene la regla con símbolos y por cada símbolo si no se acaban los símbolos y si la posición del símbolo coincide con la de la acción se crea dos etiquetas una con el símbolo correspondiente y otra con la acción que va detrás del símbolo. Se añaden al resultado y se suma uno a la posición. Si no coincide se crea una única etiqueta con el valor de símbolo y se añade al resultado. Después se suma uno a la posición en la regla.

**public void construir():**

Por cada regla de la lista de reglas, guarda la regla en un mapa según su id**,** crea la lista de etiquetas de la forma corta, y por cada etiqueta las sitúa, las dibuja y actualiza la posición de la siguiente en el eje X. Cuando termina actualiza la posición de la siguiente regla en el eje Y, y guarda las etiquetas y la forma en sus respectivos mapas.

**public void cambiarFormaRegla(Regla regla):**

Recoge la regla cuya forma se va a cambiar.

Se obtiene la posición de la regla en el eje Y y se borra del panel. Si la regla tiene la forma larga se cambia por la forma corta, se actualizan los mapas y se acorta el cuadrado correspondiente, sino se cambia por la larga, se actualizan los mapas, y se alarga el recuadro correspondiente.

**public void drawRectangle(Regla regla):**

Recoge la regla en la cual se va a pintar el rectángulo.

Se obtiene la posición de la regla en los ejes X e Y, se calcula el ancho de la regla, se crea y se dibuja el rectángulo que se va a dibujar en la gramática. Además, se guarda en un mapa según el id de la regla a la que engloba.

**public Rectangle erasedRectangle(Regla regla):**

Recoge la regla en la cual se va a borrar el rectángulo y devuelve el rectángulo borrado.

Obtiene el rectángulo a través del id de la regla y lo elimina tanto del mapa como del panel.

## Clase Gramática V0.2

**Atributos nuevos:**

**private Configuracion config:** guarda la configuración de los elementos de la aplicación (colores, tamaños de letra y zoom)

**private HashMap<String,ArrayList<String>> ruleActions:** Mapa que guarda las acciones de cada regla en función del id de las reglas.

**Constructor:** añade el campo de Configuración config a la cabecera y se lo asigna al atributo. Además, se inicializa ruleActions.

**Métodos:**

**formarReglaCorta:** añade los colores correspondientes a los terminales y no terminales.

**formarReglaLarga:** añade los colores correspondientes a los terminales, no terminales y a las acciones. Además, añade la acción a la regla correspondiente en ruleActions.

**public boolean isTerminal(Regla regla,String simbolo):**

Recoge el símbolo y la regla a la que corresponde el símbolo para decidir si es un terminal, devuelve true, o si es un no terminal, devuelve false.

Busca en los símbolos de la regla el símbolo recogido y devuelve si es terminal.

**public void updateGrammar():**

Por cada regla, por cada etiqueta que la forma, cambio el color y el tamaño de letra según sea terminal, no terminal o una acción.

**public boolean isActions(ArrayList<String> actions,String actionSearched):**

Recoge las acciones correspondientes a la regla y la acción que se va a comprobar. Devuelve true si es una acción y false si no.

Comprueba si la acción pertenece a la lista de acciones recogida si esta lista no es nula.

## Clase Gramática V0.3

Métodos:

**construir ():**

Muestra la gramática con las acciones.

**updateGrammar():**

Se cambia la forma de la regla dos veces para asegurarse de que la gramática se actualiza después de cambiar el tamaño de letra.

## Clase CadenaEntrada V0.1

**private List<String> cadenaPorPaso:** lista que contiene el estado de la cadena en cada paso.

**private double posX:** posición en el eje X del elemento de la cadena anterior.

**private Pane panelPadre:** panel donde se va a dibujar la cadena.

**private HashMap<String,Rectangle> rectanglesChain:** mapa con los rectángulos de la cadena en función de su símbolo.

**private HashMap<String,String> rectanglesText:** mapa con el símbolo de los rectángulos en función del id del rectángulo.

Constructor:

**public CadenaEntrada(List<String> cadenaPorPaso,Pane panelPadre):**

Recoge el estado de la cadena en cada paso y el panel sobre el que se va a dibujar.

Se asignan a posX una posición inicial de 10 pixeles y a panelPadre el recogido, por ultimo se inicializan los mapas.

Métodos:

**public void construir():**

Se obtiene los símbolos que forman la cadena y se inicializa la posición en el eje Y a 50. Por cada uno de los símbolos de la cadena creo una etiqueta y un rectángulo para el símbolo. El rectángulo se crea con un ancho del tamaño del símbolo mas 10 pixeles y con un alto del tamaño del símbolo. Después, se le da una opacidad al recuadro del 50%, se colocan en el panel la etiqueta y el recuadro, se la da al recuadro el símbolo como id, se añade el recuadro a los mapas y se dibujan la etiqueta y el recuadro. Para finalizar el bucle se actualiza la posición X anterior.

**public void actualizarCadena(int step):**

Recoge el paso al que se tiene que actualizar la cadena.

Se obtienen los símbolos ya procesados y los añado a un conjunto. Después, por cada clave de rectanglesChain, si la clave pertenece al conjunto la opacidad del recuadro pasara al 100%, si no, al 50%. Si el paso es el ultimo se cambia la opacidad del recuadro de EOF al 100%, si no, al 50%.

## Clase CadenaEntrada V0.2

Atributos nuevos:

**private Configuracion config:** guarda los datos de configuración de la aplicación.

**private HashMap<String,Label>labels:** se relaciona cada etiqueta con su contenido.

En el constructor se añade a la cabecera Configuracion config y en el cuerpo se asigna y se inicializa labels.

Cambios en los métodos:

**Construir:**

A la hora de crear los rectángulos y las etiquetas se le añade el color de los terminales a los rectángulos y el de letra pendiente a las etiquetas. Además, añado la etiqueta a labels.

**ActualizarCadena:**

Paso de recoger los elementos pendientes a coger los leídos. Cuando he comprobado si están pendientes o leídos le añado el color correspondiente a la etiqueta.

## Clase CadenaEntrada V0.3

Atributos:

Eliminación de **private HashMap<String,String> rectanglesText** y de todas las llamadas consiguientes.

**private int Font:** contiene el tamaño de fuente de la cadena.

Constructor:

Se le elimina la asignación de **rectanglesText** y se añade la asignación de Font.

Métodos:

**construir():** se cambia el tamaño fuente de la etiqueta antes de calcular el ancho y el alto necesario para el rectángulo.

**actualizarCadena(int step):** se asigna el tamaño de fuente a la etiqueta cuando se le asigna el color. Se reasigna el color del rectángulo y el tamaño correspondiente (se usa cuando él font cambia) y se posiciona de nuevo en caso de que font haya cambiado.

## Clase FXMLDocumentController V0.1

**Atributos:**

private ScrollPane grafo: panel con scroll que va a contener al grafo.

private ScrollPane gramática: panel con scroll que va a contener a la gramática

private ScrollPane cadenaEntrada: panel con scroll que va a contener a la cadena de entrada.

private Button masZoom: botón que corresponde al aumento de zoom.

private Slider sliderZoom: slider que regula el zoom.

private FicheroXML ejemplo: guarda la información correspondiente del traductor después de procesar el XML.

private Grafo graph: grafo que se va a representar.

private Gramatica gramar:gramatica que se va a representar.

private CadenaEntrada entryChain: cadena de entrada que se va a representar.

private Pane paneGrafo: panel que va a contener al grafo y que se va a insertar en el panel con scroll.

private Pane paneGramatica: panel que va a contener a la gramática y que se va a insertar en el panel con scroll.

private Pane paneCadenaEntrada: panel que va a contener a la cadena de entrada y que se va a insertar en el panel con scroll.

private Configuracion configuration: contiene los datos de configuración de la aplicación.

private Group graphGroup: grupo que contiene el panel del grafo ya que para que funcione el scroll después de escalar el panel es necesario insertarlo en un grupo.

**El constructor es sustituido por public void initialize(URL url, ResourceBundle rb):**

Llama a la función elegirArchivo diciéndole que el tipo de archivo tiene que ser XML.

**Métodos:**

**public void zoom(double zoom ):**

Recoge el porcentaje de escalado y escala el panel que contiene el grafo en los ejes X e Y.

**private void elegirArchivo(String tipo):**

Permite elegir y abrir un traductor.

Recoge el tipo de archivo que se necesita abrir.

Crea un selector de archivo le añade un titulo y un directorio inicial. Se le añade el filtro según el cual solo se van a mostrar archivos con extensión XML. Se obtiene el path del archivo seleccionado. Se crea un nuevo FicheroXML y se carga el traductor con esa ruta. Se carga la configuración. Se crea los paneles y se asignan el del grafo al grupo y después al panel con scroll y el resto directamente a panel con scroll. Al panel con scroll se le añade un filtro de eventos para que realice primero el evento de teclado de la aplicación y después el de mover el scroll. Se crea la cadena de entrada con los datos de la cadena de entrada del ejemplo, su panel y la configuración, y se construye la cadena de entrada. Se crea la gramática con los datos de la gramática del ejemplo, su panel y la configuración, y se construye la gramatica. Se crea el grafo con los datos del grafo del ejemplo,la gramatica, la cadena de entrada, su panel, la configuración y el tipo de traductor que se va a representar. Después se le añade un evento que reacciona cuando su valor cambia al slider del zoom, dicho evento efectua el zoom y lo guarda en el archivo de configuración. Se inicializa el slider con el valor de configuración y se aplica dicho valor como zoom al panel del grafo.

**private void handleMasMenosZoom(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón de más o al botón de menos zoom.

Si el botón pinchado es el de mas zoom se aumenta en 10 el valor del zoom.

Si no se disminuye en 10.

Y se guarda en configuración el zoom.

**public void handleOpenConfiguration(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al submenú de configuración llamado editar configuración.

Abre una nueva ventana y paraliza la anterior hasta que esta se ha cerrado.Cuando se cierra se vuelve a cargar la nueva configuración y se actualizan el grafo , la gramática y la cadena de entrada.

**public void handleDefaultConfiguration(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al submenú de configuración llamado restablecer configuración por defecto.

Vuelve a cargar la configuración por defecto, actualiza el grafo, la gramática, la cadena de entrada y el zoom. Por último, guarda la nueva configuración.

**private void handleCargarArchivo(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click a Traductor en el submenú cargar de archivo.

Llama a la función que te permite cargar otro archivo.

**private void handleAnteriorAction(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón Anterior y a Anterior del menú de ejecución.

Si hay un paso anterior retrocede a él.

**private void handleSiguienteAction(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón Siguiente y a Siguiente del menú de ejecución.

Si hay un paso posterior avanza a él.

**private void handleIrFinAction(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón Fin y a Fin del menú de ejecución.

Avanza hasta el final del procesado de la cadena

**private void handleIrInicioAction(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón inicio y a inicio del menú de ejecución.

Retrocede hasta el inicio del procesado de la cadena.

**private void handleKeyAction(KeyEvent event):**

Este método reacciona dependiendo de la tecla que se pulse.

Si es la flecha derecha retrocede un paso el procesamiento de la cadena

Si es la izquierda avanza un paso.

Si es la tecla inicio retrocede hasta el paso inicial.

Si es la tecla fin avanza hasta el último paso.

## Clase FXMLDocumentController V0.2

**Atributos:**

**private TextField inputZoom:** Representa el campo de texto donde se introduce un valor concreto de zoom.

**private Button inicioButton:** Representa el botón de inicio.

**private Button finButton:** Representa el botón de fin.

**private Button anteriorButton:** Representa el botón anterior.

**private Button siguienteButton:** Representa el botón siguiente**.**

**Métodos:**

**handleMasMenosZoom(ActionEvent event):**

Se actualiza el valor de inputZoom.

**private void handleInputZoom(ActionEvent event):**

Evento que se activa al introducir un valor en el campo de texto del zoom. Hace efectivo el zoom, actualiza el slider del zoom y guarda el nuevo zoom en la configuración.

**public void handleOpenHelp(ActionEvent event):**

Abre el manual de usuario.

**handleAnteriorAction(ActionEvent event):**

Cambia el color de los botones siguiente y final si esta al final de la ejecución a gris y si no a negro. Cambia el color de los botones anterior e inicio si esta al inicio de la ejecución a gris y si no a negro.

**handleSiguienteAction(ActionEvent event):**

Cambia el color de los botones siguiente y final si esta al final de la ejecución a gris y si no a negro. Cambia el color de los botones anterior e inicio si esta al inicio de la ejecución a gris y si no a negro.

**handleIrFinAction(ActionEvent event):**

Cambia el color de los botones siguiente y final si esta al final de la ejecución a gris y si no a negro. Cambia el color de los botones anterior e inicio si esta al inicio de la ejecución a gris y si no a negro.

**handleIrInicioAction(ActionEvent event):**

Cambia el color de los botones siguiente y final si esta al final de la ejecución a gris y si no a negro. Cambia el color de los botones anterior e inicio si esta al inicio de la ejecución a gris y si no a negro.

**handleKeyAction(KeyEvent event):**

Cambia el color de los botones siguiente y final si esta al final de la ejecución a gris y si no a negro. Cambia el color de los botones anterior e inicio si esta al inicio de la ejecución a gris y si no a negro.

**Constructor:**

Inicia el color de los botones inicio y anterior a gris y los de fin y siguiente a negro.

## Clase FXMLDocumentControllerConfig V0.1

**Atributos:**

**private Rectangle noTerminal1:** rectángulo que representa el primer no terminal del árbol de ejemplo.

**private Rectangle noTerminal2:** rectángulo que representa el segundo no terminal del árbol de ejemplo.

**private Rectangle terminal:** rectángulo que representa el segundo no terminal del árbol de ejemplo.

**private Rectangle readChainR:** rectángulo que representa la parte leída de la cadena en la cadena de entrada de ejemplo.

**private Rectangle pendChainR:** rectángulo que representa la parte pendiente de la cadena en la cadena de entrada de ejemplo.

**private Label labelreadChain:** etiqueta que representa la parte leída de la cadena en la cadena de entrada de ejemplo.

**private Label labelpendChain:** etiqueta que representa la parte pendiente de la cadena en la cadena de entrada de ejemplo

**private Label labelNoTerminal1:** etiqueta que representa el primer no terminal del árbol de ejemplo.

**private Label labelNoTerminal2:** etiqueta que representa el segundo no terminal del árbol de ejemplo.

**private Label labelTerminal:** etiqueta que representa el terminal del árbol de ejemplo.

**private Label grammarNoTerminal1:** etiqueta que representa el primer no terminal de la gramática de ejemplo.

**private Label grammarTerminal:** etiqueta que representa el terminal de la gramática de ejemplo.

**private Label grammarNoTerminal2:** etiqueta que representa el segundo no terminal de la gramática de ejemplo.

**private Label action:** etiqueta que representa la acción de la gramática de ejemplo.

**private ComboBox sizeFuenteArbol:** combo box que te permite elegir el tamaño de la letra del árbol.

**private ComboBox sizeFuenteTraduc:** combo box que te permite elegir el tamaño de la letra de la gramática.

**private ComboBox sizeFuenteCadena:** combo box que te permite elegir el tamaño de la letra de la cadena de entrada.

**private ComboBox fountTypeSemanticAct:** combo box que te permite elegir el tipo de letra de las acciones de la gramática.

**private ComboBox sizeFountSemanticAct**: combo box que te permite elegir el tamaño de la letra de las acciones de la gramática.

**private ColorPicker backgroundColorTerminals:** selector de color que te permite elegir el color de los recuadros de los terminales.

**private ColorPicker fontColorTerminals:** selector de color que te permite elegir el color de la letra de los terminales.

**private ColorPicker backgroundColorNoTerminals:** selector de color que te permite elegir el color de los recuadros de los no terminales.

**private ColorPicker fontColorNoTerminals:** selector de color que te permite elegir el color de la letra de los terminales.

**private ColorPicker readPart:** selector de color que te permite elegir el color de la letra de la parte leída de la cadena de entrada.

**private ColorPicker pendPart:** selector de color que te permite elegir el color de la letra de la parte pendiente de la cadena de entrada.

**private ColorPicker colorFountSemanticAct:** selector de color que te permite elegir el color de la letra de las acciones semánticas de la gramática.

**private javafx.scene.control.Button aceptButton:** botón de aceptar.

**private javafx.scene.control.Button cancelButton;** botón de cancelar.

**private Grafo arbolEjemplo:** árbol de ejemplo.

**private String colorBackgroundTerminals:** Color del rectángulo de los terminales.

**private String colorTextTerminals:** Color del texto de las etiquetas de los terminales.

**private String colorBackgroundNoTerminals:** Color del rectángulo de los no terminales.

**private String colorTextNoTerminals:** Color del texto de las etiquetas de los no terminales.

**private String colorAction:** Color del texto de las etiquetas de las acciones.

**private String colorPendChain:** Color del texto de las etiquetas de la parte pendiente de la cadena.

**private String colorReadChain:** Color del texto de las etiquetas de la parte leída de la cadena.

**private int actionSize:** tamaño de letra de las acciones.

**private int treeSize:** tamaño de letra de las etiquetas del árbol.

**private int grammarSize:** tamaño de letra de las etiquetas de la gramática.

**private int chainSize:** tamaño de letra de las etiquetas de la cadena de entrada.

**private String actionType:** tipo de letra de las acciones.

**private Configuracion lectConf:** configuración de la aplicación antes de los cambios.

**public void initialize(URL url, ResourceBundle rb):**

Es el sustituto al constructor.

Carga la configuración antigua y asigna los datos correspondientes a los atributos. Después, configura lo ejemplos según los anteriores datos. Por cada combo box añade los valores que va a permitir elegir, el valor que va a mostrar y un evento que se accionara cuando cambie el valor del combo box. Este evento asigna el valor al atributo correspondiente y realiza los cambias en los ejemplos para poder verlos. Por último, pone el valor que se está usando en los selectores de color y en el combo box del tipo de letra.

**public void handleAcept(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón aceptar.

Guarda la configuración nueva y cierra la ventana.

**public void handleCancel(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click al botón cancelar.

Cierra la ventana.

**public void handleChooseColor(ActionEvent event):**

Este método es un evento que se asigna en el XML FXMLDocument y reacciona al darle click a un selector de color.

Dependiendo el selector de color usado actualiza los ejemplos y asigna el nuevo valor al atributo correspondiente.

## Clase FXMLDocumentControllerConfig V0.2

**Constructor:**

Los datos de los combo box pasan a ser Strings para poder introducirlos mediante campos de texto.