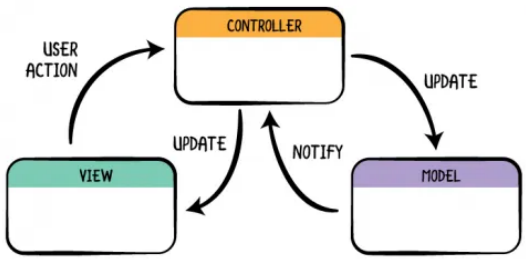
Manual Técnico

Patrón de diseño de la aplicación

Para la aplicación se decidió utilizar el patrón de diseño MVC (model view controller) permitiendo así separar y organizar de mejor forma el código utilizado para la interfaz de usuario, como esta es controlada y el procesamiento de datos del back end.



Clase App

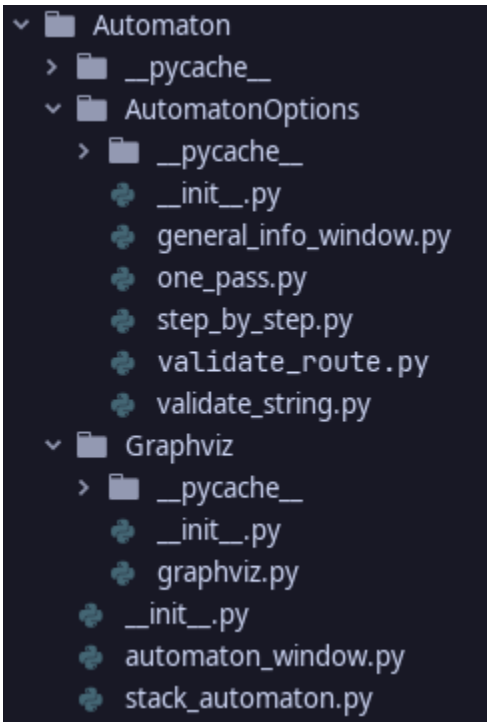
Esta clase es la raíz de toda la aplicación contiene un método útil para cambiar de ventana a una nueva y cerrar la actual, también contiene los arreglos de las gramáticas y autómatas de pila cargadas al sistema. Por esta razón es necesario pasar como argumento esta clase a otras ventanas para no perder el hilo principal de la aplicación y mantener todos los datos guardados correctamente.

```
class App(Tk):
    def __init__(self) -> None:
        Tk.__init__(self)
        self._frame = None
        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)
        self.title("Spark Stark")

        # App variables
        self.gramatics_objects = []
        self.automaton_objects = []

    def switch_frame(self, frame_class):
        new_frame = frame_class
        if self._frame is not None:
            self._frame.destroy()
        self._frame = new_frame
        self.columnconfigure(0, weight=1)
        self.rowconfigure(0, weight=1)
```

Módulo Autómata



En este módulo principal se encuentran todos los archivos necesarios para las ventanas, clases y funciones necesarias para llevar a cabo las tareas de la aplicación correspondientes a los autómatas de pila

Módulo stack_automaton

Clases Principales y usos:

1. Stack: Una clase que se comporta como una pila con los métodos push y pop.
2. Transition: Clase que almacena las transiciones correspondientes a un autómata de pila, además de incluir un método que facilita la comparación de 2 transiciones para verificar si son distintas o iguales.
3. StackAutomaton: Es el autómata de pila y toda la información correspondiente a uno. Además de tener validaciones para cada uno de sus datos también posee un método para comprobar si una cadena es válida o no trabajando en conjunto con la clase Stack y la clase Transition.

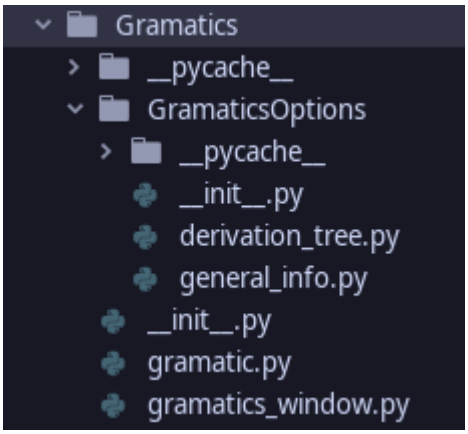
Modulo Graphviz

Posee distintas funciones que reciben como entrada ya sea un objeto autómata de pila o una lista de transiciones para devolver una cadena lista para ser guardada en un archivo .dot de graphviz.

Resto de módulos

El resto de módulos sirven principalmente para gestionar la interfaz de usuario y como la información es gestionada de los módulos anteriormente mencionados.

Módulo Gramatics



En este módulo principal se encuentran todos los archivos para las ventanas, clases y funciones necesarias para llevar a cabo las tareas de la aplicación correspondientes a las gramáticas libres de contexto.

Módulo gramatic

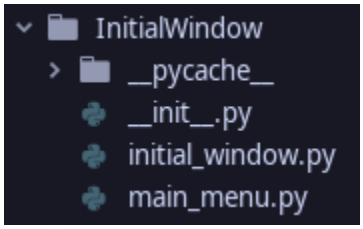
Clases principales y sus usos:

1. Production: Almacena la información de una producción de una gramática libre de contexto.
2. Gramatic: Almacena la información correspondiente de una gramática libre de contexo, además de proporcionar validaciones para cada uno de sus valores.

Resto de módulos

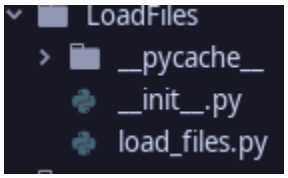
El resto de módulos sirven principalmente para gestionar la interfaz de usuario y como la información es gestionada de los módulos anteriormente mencionados.

Módulo InitialWindow



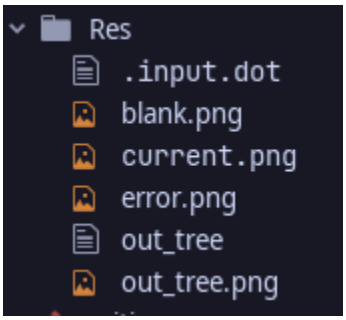
En este módule se guardan los módulos correspondientes a la pantalla de spash y al menú principal de la aplicación.

Módulo LoadFiles



El módulo de carga de datos es muy similar tanto para las gramáticas libres de contexto y los autómatas de pila, lo único que cambia en ambos es en la forma en la que se tiene que leer el archivo. Por lo que ambos métodos fueron condensados en un archivo y al crear una nueva instancia del controller de esta clase es necesario indicar con una variable booleana si se trata de cargar una gramática o un autómata.

Módulo Res



Se guardan algunos recursos temporales y permanentes necesarios para la aplicación.