Rosy: un entorno de desarrollo integrado para proyectos Arduino

Macoritto Torcivia, Martín

Proyecto de grado

Facultad de Ingeniería e Informática

- 2022 -

Título:

Ingeniero en Informática

Profesor guía:

Nombre: Ing. Rivera Bernsdorff, Fernando Lucas **Firma: _____**

Alumno:

Nombre: Macoritto Torcivia, Martín **Firma: _____**

Tribunal evaluador:

Nombre: _____ **Firma: _____**

Nombre: _____ **Firma: _____**

Nombre: _____ **Firma: _____**

Fecha de exposición: ____/____/____

Dedicatorias

A toda la comunidad del software libre que se empeña en crear y brindar software de calidad a los usuarios con el propósito de ayudar y servir a los demás.

A lo estudiantes de todos los niveles educativos que día a día se esmeran y esfuerzan en aprender nuevos conocimientos y capacidades técnicas para conseguir sus objetivos.

Agradecimientos

Al Ing. Rivera Bernsdorff, Fernando Lucas, mi profesor guía por aceptar la dirección de este proyecto de grado. Gracias por su constante apoyo y por transmitir con gran generosidad sus conocimientos y experiencias.

A mis padres Carmen y Alberto y mis hermanas María Fernanda y Claudia, los seres de luz con quienes tuve la gracia de compartir toda mi vida. Gracias por enseñarme los mejores valores de la vida.

A mis amigos Ralf, Gianluca, Antonio Nahuel y Anibal Ricardo por acompañarme durante toda la carrera y compartir su hermosa amistad.

A las profesoras Talamé y Cardoso por dictar la materia que me inspiro a realizar este proyecto.

A Luis Haro por brindar la información sobre las materias dictadas en las escuelas técnicas.

Índice

Abstract ... 1
1. Introducción ... 2
1.1. Breve descripción del problema ... 2
1.2. Motivación ... 2
1.3. Procedimiento a realizar ... 2
1.4. Criterios de éxito ... 3
2. Estado de la cuestión ... 4
2.1. Marco teórico ... 4
2.1.1. IDE ... 4
2.1.2. Arduino ... 4
2.1.3. Lenguaje de programación ... 5
2.1.4. Compilador ... 6
2.1.5. Intérprete y Depurador ... 8
2.1.6. Software libre ... 9
2.1.7. Código abierto ... 9
2.2. Soluciones Similares ... 9
2.2.1. Arduino IDE ... 10
2.2.2. Visuino ... 11
2.2.3. S4A ... 11
2.2.4. Minibloq ... 12
3. Definición del problema ... 13
3.1. Introducción ... 13
3.2. Objetivos ... 13
3.2.1. Objetivo general ... 13
3.2.2. Objetivos específicos ... 13
3.3. Alcance ... 14
3.4. Alternativas tecnológicas ... 15
3.4.1. Sistemas operativos ... 15
3.4.2. Lenguaje de programación ... 15
4. Solución propuesta ... 16
4.1. Definición de la solución ... 16
4.2. Metodología ... 17
4.3. Planificación del proyecto ... 18
4.3.1. WBS ... 18
4.3.2. Diccionario WBS ... 20

4.3.3. Recursos necesarios ... 28
4.3.4. Gantt ... 29
4.3.5. Análisis de costos ... 31
4.3.6. Análisis de riesgo ... 34
4.3.7. Factibilidad ... 36
5. Desarrollo de la solución ... 37
5.1. Funciones para sensores y actuadores ... 37
5.1.1. Método para calcular la distancia (sensor ultrasónico) ... 37
5.1.2. Función de sondeo (servomotor) ... 38
5.2. Tokens y palabras reservadas ... 39
5.2.1. Tokens de Pini ... 39
5.2.2. Tokens de uso común ... 40
5.2.3. Tokens de Rosy ... 40
5.3. Gramática del lenguaje ... 42
5.3.1. Reglas gramaticales de Pini ... 42
5.3.2. Reglas gramaticales de Rosy ... 45
5.4. Analizador léxico y sintáctico ... 58
5.4.1. Diferenciación de tipos de pines ... 59
5.4.2. Librerías de terceros ... 59
5.4.3. Manejo de variables globales, funciones y métodos ... 60
5.4.4. Precedencia ... 61
5.4.5. Depurador de errores ... 61
5.4.6. Proceso de compilación ... 63
5.5. Interfaz gráfica de usuario (GUI) ... 64
5.5.1. Interfaz principal de la aplicación ... 64
5.5.2. Interfaz de selección de componentes ... 68
5.5.3. Resaltador de sintaxis ... 69
5.5.4. Archivo de preferencias ... 70
5.6. Sitio web y documentación ... 70
1. Página de inicio ... 70
2. Descargas ... 71
3. Documentación ... 71
4. Información adicional ... 77
6. Resultados ... 78
7. Conclusiones y futuras líneas de investigación ... 84
8. Bibliografía ... 85
Anexos ... 87

Índice de ilustraciones

Ilustración 1: Placa Arduino UNO [Autor de fotografía Macoritto Torcivia, 2020] 5

Ilustración 2: Esquema de un compilador [Tomado de Louden, 2005] 7

Ilustración 3: Fases de un compilador [Tomado de Louden, 2005] 7

Ilustración 4: Arduino IDE [Captura de pantalla tomada por Macoritto Torcivia] 10

Ilustración 5: Visuino [Tomado de https://www.visuino.com] 11

Ilustración 6: Scratch for Arduino [Tomado de http://s4a.cat/index_es.html] 12

Ilustración 7: Minibloq [Tomado de http://blog.minibloq.org] 12

Ilustración 8: Diagrama del funcionamiento de Rosy IDE 16

Ilustración 9: Modelo incremental de n incrementos [Tomado de Pressman, 2010] 18

Ilustración 10: Diagrama WBS 19

Ilustración 11: Gantt del proyecto (resumido) 30

Ilustración 12: Matriz de riesgos 35

Ilustración 13: Funcionamiento del sensor ultrasónico [Tomado de Luis Llamas, 2015] 37

Ilustración 14: Robot Arduino con sensor ultrasónico acoplado a un servomotor [Autor de fotografía Macoritto Torcivia, 2022] 38

Ilustración 15: Funcionamiento de la regla read 59

Ilustración 16: Estructura del lenguaje de Arduino basado en C++ 60

Ilustración 17: Depurador de archivos Lex de Pini y Rosy 62

Ilustración 18: Depurador del archivo Yacc de Rosy 62

Ilustración 19: Diagrama de proceso de compilación 63

Ilustración 20: Diagrama del proceso de compilación de Pini 63

Ilustración 21: Diagrama de proceso de compilación de Rosy 64

Ilustración 22: Interfaz principal de Rosy IDE 65

Ilustración 23: Opciones del menú Archivo 65

Ilustración 24: Opciones del menú Editar 66

Ilustración 25: Opciones del menú Pines 66

Ilustración 26: Opciones del menú Ayuda 66

Ilustración 27: Mensaje de error de sintaxis de Rosy 67
Ilustración 28: Mensaje de carácter ilegal en Pini 67
Ilustración 29: Interfaz de selección de componentes 68
Ilustración 30: Ventana para configurar un componente 69
Ilustración 31: Código generado por la interfaz de selección de componentes 69
Ilustración 32: Página de inicio del sitio web de Rosy 70
Ilustración 33: Botones de acceso rápido de la página principal 71
Ilustración 34: Página de descarga de Rosy IDE 71
Ilustración 35: Documentación de Rosy IDE 72
Ilustración 36: Comparación de códigos 72
Ilustración 37: Buenas practicas del lenguaje 73
Ilustración 38: Imágenes de los sensores y actuadores compatibles con Rosy IDE 73
Ilustración 39: Sintaxis para el sensor ultrasónico 74
Ilustración 40: Explicación del ciclo condicional if 74
Ilustración 41: Ejemplos para definir funciones y métodos 75
Ilustración 42: Estructura general del código en Rosy 75
Ilustración 44: Arduino Explicación de las funciones específicas de Rosy 76
Ilustración 43: Explicación de las funciones de 76
Ilustración 45: Ejemplo de la función READ 77
Ilustración 46: página de información adicional de Rosy IDE 77
Ilustración 47: Código necesario en Rosy y Pini para medir la distancia con un sensor ultrasónico 78
Ilustración 48: Código necesario en C++ para medir la distancia con un sensor ultrasónico 78
Ilustración 49: Código generado por Arduino IDE [captura de pantalla tomada por Macoritto Torcivia] 79
Ilustración 50: Declaración de pines en Arduino 79
Ilustración 51: Declaración de variables en Arduino 79
Ilustración 52: Ciclo void en Arduino 80
Ilustración 53: Ciclo setup en Arduino 80
Ilustración 54: Ventana de configuración para el sensor ultrasónico 81

Ilustración 55: Código en Rosy para obtener la distancia mediante el uso de un sensor ultrasónico ... 81

Ilustración 56: Interfaz de configuración de pines para el módulo joystick ... 82

Ilustración 57: Modelo de placa Arduino UNO con colores de referencia en los pines utilizada en el IDE ... 83

Índice de tablas

Tabla 1: Costos de los RRHH ... 31

Tabla 2: Costos del hardware y servicios ... 31

Tabla 3: Flujo de caja ... 33

Abstract

El presente proyecto de grado se basa en la creación e implementación de un entorno de desarrollo integrado, llamado Rosy (analogía de Robot easy), para proyectos que utilicen la plataforma Arduino. Su finalidad es brindar un programa sencillo, eficiente y de fácil uso para todo tipo de usuarios, principalmente estudiantes y profesores de instituciones primarias, secundarias y/o universitarias. De esta manera se busca facilitar e incentivar el desarrollo y la programación de robots o cualquier tipo de proyecto basado en Arduino.

Para llevarlo a cabo, se desarrollará un lenguaje de programación que utilice una sintaxis coloquial similar a la de SQL, una interfaz gráfica para facilitar la conexión y configuración de componentes a la placa Arduino y un editor de texto con un depurador que indique los errores de sintaxis o gramaticales.

El conjunto de estas herramientas conforma el IDE (entorno de desarrollo integrado) Rosy, el cual será publicado con una licencia de software libre para permitir que todo usuario pueda utilizarlo con las libertades que este otorga y sin la necesidad de pagar por una licencia de uso.

1. Introducción

1.1. Breve descripción del problema

La popularidad de Arduino creció rápidamente durante los últimos años, debido a su simpleza y practicidad para enseñar robótica, electrónica o programación. En algunos países, la robótica es un conocimiento que la mayoría de estudiantes obtienen durante su transcurso en establecimientos educativos. En cambio, en Argentina la situación es diferente, pocos estudiantes adquieren estos conocimientos durante su formación académica, y cuando ingresan a instituciones de nivel superior (universidades/terciarios) les resulta complejo aprender sobre programación, electrónica o robótica sin antes haber tenido una base teórica y práctica de estos conceptos.

El programa desarrollado brinda una solución a esta problemática, al ofrecer un entorno para la creación de proyectos basados en Arduino que resulta práctico y eficaz para la mayoría de los usuarios, tanto aquellos que poseen cocimientos avanzados o básicos.

1.2. Motivación

La idea original del proyecto surgió gracias al trabajo final de Compiladores, una materia de cuarto año de Ingeniería en Informática. El proyecto consistía en la elaboración de un traductor de código desarrollado en PLY (Python Lex-Yacc) que hacía uso de una sintaxis proporcionada por la cátedra. El objetivo era obtener un código fuente en C++ para ser interpretado por un robot construido con Arduino.

Por otra parte, en la materia Sistemas Expertos de quinto año de ingeniería en Informática, se programó un robot con el fin de seguir una línea negra para resolver un laberinto. A pesar de que el robot utilizado ya estaba ensamblado, escribir la lógica del mismo sin conocimientos previos del lenguaje de Arduino resultó ser complejo.

Debido a estas situaciones se llegó a la idea de crear un entorno de desarrollo integrado mediante el uso de PLY para simplificar la programación y el ensamblado de proyectos Arduino.

1.3. Procedimiento a realizar

Para llevar a cabo este proyecto, se establecieron una serie de tareas a realizar, necesarias para su implementación:

1. Análisis e investigación del estado de la cuestión y de soluciones similares existentes.
2. Definición de los tokens del lenguaje, es decir, las palabras reservadas y los símbolos que componen al mismo.
3. Creación de la sintaxis del lenguaje con el uso de los tokens definidos en el paso anterior.
4. Desarrollo de un editor de texto que emplee el lenguaje definido previamente y facilite su uso mediante una interfaz gráfica.

1.4. Criterios de éxito

Los criterios de éxito serán utilizados al finalizar el proyecto para determinar si este cumple con los factores propuestos. Ellos son:

1. Cumplir con los objetivos planteados en el proyecto dentro de los límites del alcance establecido.
2. Aplicar correctamente las metodologías de trabajo definidas para el desarrollo e implementación de la solución propuesta.
3. Optimizar el proceso de escritura de código mediante la reutilización de la configuración de los pines de la placa.
4. Que el software desarrollado sea fácil de manejar e instalar tanto en sistemas operativos Windows como GNU/Linux.

2. Estado de la cuestión

2.1. Marco teórico

En el presente capítulo, se detallarán los temas y conceptos teóricos más importantes para comprender el desarrollo de este proyecto.

2.1.1. IDE

Un Entorno de Desarrollo Integrado, también conocido por sus siglas en inglés IDE (*Integrated Development Environment*), es un conjunto de herramientas y programas que simplifican las tareas realizadas por los desarrolladores en un solo software.

Generalmente, un IDE está compuesto por un editor de texto, el cual permite modificar el código de un proyecto escrito en cierto lenguaje de programación. Además, poseen un compilador que lleva a cabo la ejecución del programa desarrollado de manera local, es decir, sin la necesidad de recurrir a otro software. También cuenta con un depurador capaz de detectar los errores del código, tanto semánticos como sintácticos, y marcarlos visualmente para identificarlos con facilidad. Por último, tienen una Interfaz Gráfica de Usuario, GUI (*Graphical User Interface*), que le permite al desarrollador interactuar visualmente con todas las herramientas del IDE, con el fin de facilitar ciertas acciones que, sin esta ayuda, requieren líneas de comando. [RedHat, 2020]

2.1.2. Arduino

Arduino es una plataforma de código abierto que utiliza placas electrónicas con microcontroladores para el desarrollo de diversos proyectos. [Arduino.cc, 2020]

Estas placas son programadas mediante un software que utiliza un lenguaje de programación propio basado en C++, el cual provee funciones y variables específicas que permiten manipular de manera sencilla las conexiones a la placa. Esto permite manejar y controlar diversos actuadores, sensores y periféricos genéricos.

Un actuador es un dispositivo capaz de realizar una acción física que modifique el entorno. Por ejemplo, un motor de corriente continua que desplaza a un robot de una posición X a una posición Y. En cambio, un sensor es un elemento que capta y/o detecta variaciones físicas un entorno determinado. Por ejemplo, un sensor ultrasónico emite ondas sonoras inaudibles, al rebotar sobre una superficie lisa éstas vuelven hacia el sensor y con un pequeño cálculo se puede determinar la distancia a la que se encuentra el objeto con el que impactó la onda. [Russell & Norvig, 2004]

Por otra parte, un periférico es un dispositivo de entrada/salida que permite una comunicación eficiente entre el usuario y la placa. Por ejemplo, una pantalla LCD que muestra los valores obtenidos por un sensor de temperatura. [Morris Mano, 1998]

Las placas de Arduino más populares y utilizadas en Argentina son la Nano, Mega y Uno, esta última es más requerida, sobre todo por quienes recién inician en el mundo de Arduino. Básicamente la diferencia entre estas tres placas es el tamaño de las mismas y la cantidad de pines que poseen.

En la Ilustración 1, se muestra una placa Arduino Uno R3, la misma posee 6 pines analógicos en su parte inferior izquierda y 12 pines digitales del lado derecho. También cuenta con 3 pines GND (*ground*) y 1 pin VCC de 5V. Esta placa puede ser alimentada por una fuente de corriente continua de 7 a 12V (recomendado) o una batería de 9V. Además, tiene un puerto USB tipo B que permite la conexión a un computador para cargar programas en la placa y monitorear la misma.

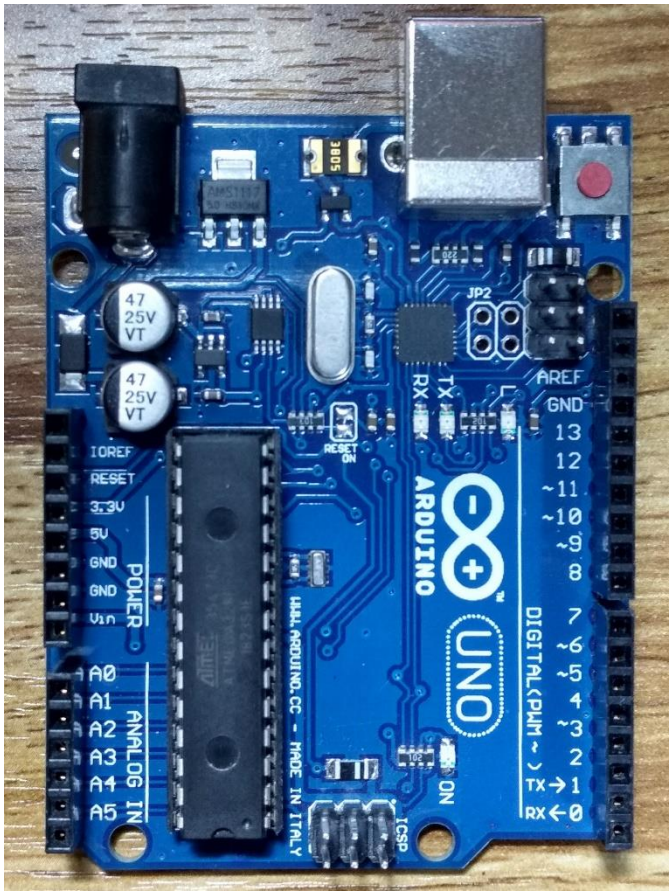


Ilustración 1: Placa Arduino UNO [Autor de fotografía Macoritto Torcivia, 2020]

2.1.3. Lenguaje de programación

Un lenguaje de programación es el conjunto de *tokens* (palabras reservadas y símbolos) que tienen una sintaxis y semántica definidas, por lo general de manera coloquial, con el fin de facilitar la comunicación entre un programador y un computador mediante la escritura de líneas de código que especifican la lógica y estructura de datos del programa.

Cuando surgieron las primeras computadoras con programa almacenado, de la mano de John von Neumann, fue necesario escribir secuencias de códigos para que éstas ejecuten las ordenes o cálculos deseados por los operadores. En un comienzo, estos programas eran escritos en código máquina, secuencias numéricas que representan las operaciones reales que el computador iba a realizar, pero esta forma de escribir era sumamente compleja y ocupaba demasiado tiempo. Ante este problema, surgió Assembler, también conocido como lenguaje ensamblador, que solucionaba la complejidad de escribir en código máquina al implementar instrucciones más cortas y simbólicas que realizaban las mismas acciones. Assembler era muy

rápido y eficiente, tanto que en la actualidad aún se lo utiliza, en especial para tareas que requieren gran velocidad y brevedad en el código. Pero estas ventajas no quitaban el hecho de ser un lenguaje difícil de aprender y sobre todo complejo a la hora de programar, ya que depende de la arquitectura del computador que se utilice. A finales de los años 50 surge un nuevo lenguaje de programación, FORTRAN, desarrollado por un equipo de IBM. Este nuevo lenguaje fue un gran éxito, gracias a que utilizaba una sintaxis mucho más sencilla que la de Assembler, y además era similar a la notación matemática y al lenguaje coloquial. [Louden, 2005]

Al pasar los años, con el auge de la programación orientada a objetos (POO) y los nuevos conceptos de la ingeniería de software, surgieron nuevos lenguajes de programación, cada vez más potentes y fáciles de usar, tales como C++, Java y Python entre otros. En la actualidad existen diversos tipos de lenguajes de programación, algunos de ellos son compilados, mientras que otros son interpretados, pero lo que realmente los diferencia en términos de cual es más fácil de usar y aprender que otro, es su sintaxis y complejidad en la estructura de datos.

No es lo mismo enseñarle a programar en Java o C a alguien que apenas inició en el mundo de la programación, que enseñarle en Python o Scratch. Estos dos últimos lenguajes son mucho más fáciles de aprender que los primeros, sobre todo Scratch, que cuenta con una interfaz gráfica de bloques que facilita la escritura de código, principalmente orientada a niños y adolescentes. Por otro lado, la sintaxis de Python es sencilla y breve, algo fundamental para una persona que aprende a programar, ya que reduce la escritura de código y facilita la comprensión del mismo. Por ejemplo, la siguiente línea de código escrita en Java muestra la oración *Hello Word* en pantalla,

System.out.println(“Hello Word”);

cabe aclarar que para que esta línea funcione es necesario definir previamente el método *main(String[] args)* y compilar el programa. En cambio, la línea de código

print(“Hello Word”)

escrita en Python hace lo mismo que la de Java vista anteriormente, con la diferencia de que esta requiere menos código y (al ser Python un lenguaje interpretado) no es necesario compilar el programa. Al comparar estos ejemplos, podemos observar que el lenguaje de programación Python es más sencillo y fácil de utilizar que Java, sobre todo para las personas que comienzan a programar por primera vez.

2.1.4. Compilador

Un compilador es un programa de computadora que traduce un código escrito en cierto lenguaje de programación a otro equivalente en un lenguaje diferente. El código que toma como entrada se conoce como programa fuente, que suele ser un lenguaje de alto nivel (como Java o C), mientras que la salida, por lo general, es un código de lenguaje intermedio (como ByteCode en el caso de Java) o bajo nivel (como lenguaje máquina en el caso de C), llamado programa objetivo (Ilustración 2). [Louden, 2005; Ruiz Catalán, 2010; Ocaña, 2015]

Ilustración 2: Esquema de un compilador [Tomado de Louden, 2005]

Un compilador está compuesto por seis fases o etapas que ejecutan diferentes operaciones lógicas para realizar el proceso de traducción (Ilustración 3). Cada fase se puede estudiar por separado, pero trabajan en secuencia junto con tres componentes auxiliares. [Louden, 2005]

A modo practico, solo se detallarán las dos primeras fases del compilador (analizador léxico y analizador sintáctico respectivamente), ya que estas son las que se utilizarán para realizar el proyecto[1].

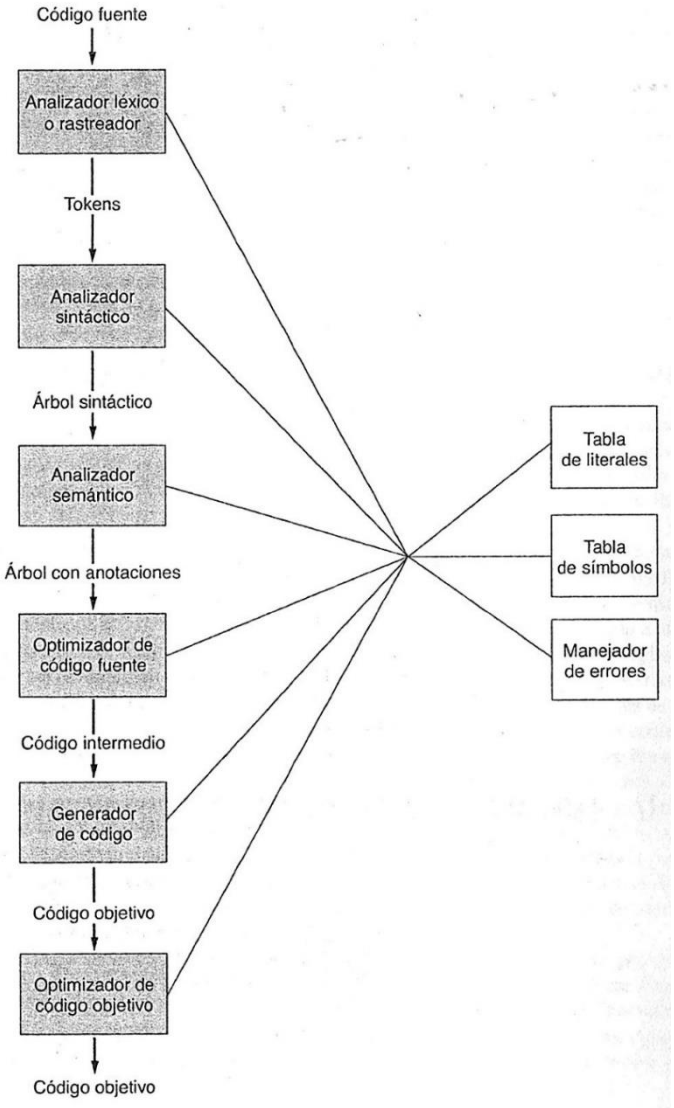


Ilustración 3: Fases de un compilador [Tomado de Louden, 2005]

[1] En la sección 3.3. Alcance correspondiente al capítulo **3. Definición del problema** se explicarán los fundamentos por los cuales sólo se toman las dos primeras fases de un compilador.

2.1.4.1. Analizador léxico

Es la primera fase de un compilador, también conocida como rastreador o *scanner*. Su función consiste en leer carácter por carácter el código fuente para generar tokens (secuencias de caracteres ordenadas), los cuales pueden ser palabras reservadas del lenguaje, como *while*, *if*, *for*, etc. o símbolos, como punto y coma, paréntesis, entre otros. Un *token* está compuesto por dos partes: tipo y lexema, este último corresponde al valor que toma el *token*. Por ejemplo, si en el lenguaje definimos una variable con el nombre X, esta es de tipo identificador y su lexema es X. [Ruiz Catalán, 2010; Louden, 2005]

La forma práctica en la que se implementa el analizar léxico es mediante el uso de expresiones regulares. "Las expresiones regulares representan patrones de cadenas de caracteres. Una expresión regular r se encuentra completamente definida mediante el conjunto de cadenas con las que concuerda". [Louden, 2005]

2.1.4.2. Analizador sintáctico

Es la segunda fase de un compilador, también conocida como *parser*. Se encarga de obtener los tokens generados por el analizador léxico y corroborar que la estructura del programa sea correcta, es decir, que respeten el orden correspondiente definido en la gramática del lenguaje de programación. Si el analizador detecta que la estructura del programa es correcta, entonces genera un árbol sintáctico, también conocido como árbol de análisis gramatical. Caso contrario, genera un informe de error. [Ruiz Catalán, 2010; Louden, 2005]

Este proceso se realiza de manera práctica implementando gramáticas libres de contexto. "Una gramática libre de contexto es una especificación para la estructura sintáctica de un lenguaje de programación. Una especificación así es muy similar a la especificación de la estructura léxica de un lenguaje utilizando expresiones regulares, excepto que una gramática libre de contexto involucra reglas de recursividad". [Louden, 2005]

2.1.5. Intérprete y Depurador

Un intérprete, al igual que un compilador, es un programa capaz de traducir un código escrito en un lenguaje a otro. La diferencia está en que el intérprete analiza línea por línea el código fuente, la convierte en código objeto (lenguaje de máquina) y la ejecuta, mientras que el compilador lee todo el código fuente, genera un archivo con el código equivalente en lenguaje maquina (código objeto) y luego permite su ejecución. [Louden, 2005; Ruiz Catalán, 2010]

"Los intérpretes también se utilizan con frecuencia en situaciones relacionadas con la enseñanza o con el desarrollo de software, donde los programas son probablemente traducidos y vueltos a traducir muchas veces. Por otra parte, es preferible usar un compilador si lo que importa es la velocidad de ejecución, ya que el código objeto compilado es siempre más rápido que el código fuente interpretado, en ocasiones hasta por un factor de 10 o más". [Louden, 2005]

"Un depurador es un programa que puede utilizarse para determinar los errores de ejecución de un programa compilado. A menudo está integrado con un compilador en un IDE" [Louden, 2005]

2.1.6. Software libre

La FSF (*Free Software Foundation*) define al software libre como todo programa que otorgue al usuario las cuatro libertades esenciales:

- Libertad 0: libertad de ejecutar el programa como se desee y para los fines que se deseen.
- Libertad 1: libertad de estudiar el funcionamiento del programa y modificarlo (para esto es necesario tener acceso al código fuente).
- Libertad 2: libertad de redistribuir copias del programa.
- Libertad 3: libertad de distribuir copias del programa con modificaciones (para esto es necesario tener acceso al código fuente).

Si un programa cumple con estas cuatro libertades de manera adecuada, se dice que es software libre. [FSF, 2019]

Que un software sea gratuito no significa que sea software libre, a pesar de que en inglés *free* signifique gratis. El software libre, por otro lado, busca respetar la libertad de los usuarios y la comunidad, esto quiere decir que "los usuarios tienen la libertad de ejecutar, copiar, distribuir, estudiar, modificar y mejorar el software". [FSF, 2019]

Para lograr esto, la FSF dispone de varias licencias de software libre que otorgan las cuatro libertades. La más utilizada en la mayoría de los proyectos de software libre es la *GNU General Public License* (Licencia Pública General de GNU) o GPLv3.

2.1.7. Código abierto

Por lo general, suele confundirse a menudo los términos software libre con código abierto. Tener acceso al código fuente de un programa no otorga las cuatro libertades del software libre.

La diferencia entre estos términos reside en un aspecto más filosófico y ético. El código abierto u *Open Source* se centra en las ventajas prácticas de liberar el código fuente y permitir modificaciones con el fin de obtener programas más potentes y confiables. En cambio, el software libre se enfoca en proteger y respetar la libertad del usuario al utilizar software. [Stallman, 2020]

2.2. Soluciones Similares

Actualmente existen diversos IDE y software que facilita la programación en Arduino. Algunos de ellos son de código abierto y sin costo, otros de uso bajo licencia no libre y pagos, unos poseen interfaz gráfica y compiladores locales, pero todos tienen sus ventajas y desventajas al utilizarlos. A continuación, se describirán los más populares y utilizados por la comunidad.

2.2.1. Arduino IDE

Arduino cuenta con un software oficial de código abierto que permite programar sus placas, llamado Arduino IDE. Este programa está desarrollado en Java y se encuentra disponible para los sistemas operativos Linux, MacOS y Windows.

El lenguaje de programación que utiliza está basado en C++ y cuenta con variables y funciones específicas que permiten controlar de manera más sencilla la placa. A pesar de esto y de poseer los tipos de dato básicos (*String*, *int*, *boolean*, etc.), el lenguaje carece de algunas estructuras de datos más sofisticadas como *ArrayLists*. En el caso de las listas bidimensionales, se pueden implementar mediante el uso de punteros, tarea que no es fácil para un usuario que inicia con Arduino.

Este software, además, consta de un compilador local, el cual facilita el proceso de carga de un programa a la placa. Por otra parte, el editor de código integrado al IDE es muy básico y no cuenta con auto llenado predictivo ni marca los errores en tiempo real (Ilustración 4). Para corroborar que el código escrito no tenga errores de sintaxis es necesario compilar el programa, lo cual conlleva un tiempo extra innecesario que otros IDE evitan al marcar los errores en el mismo editor de texto.

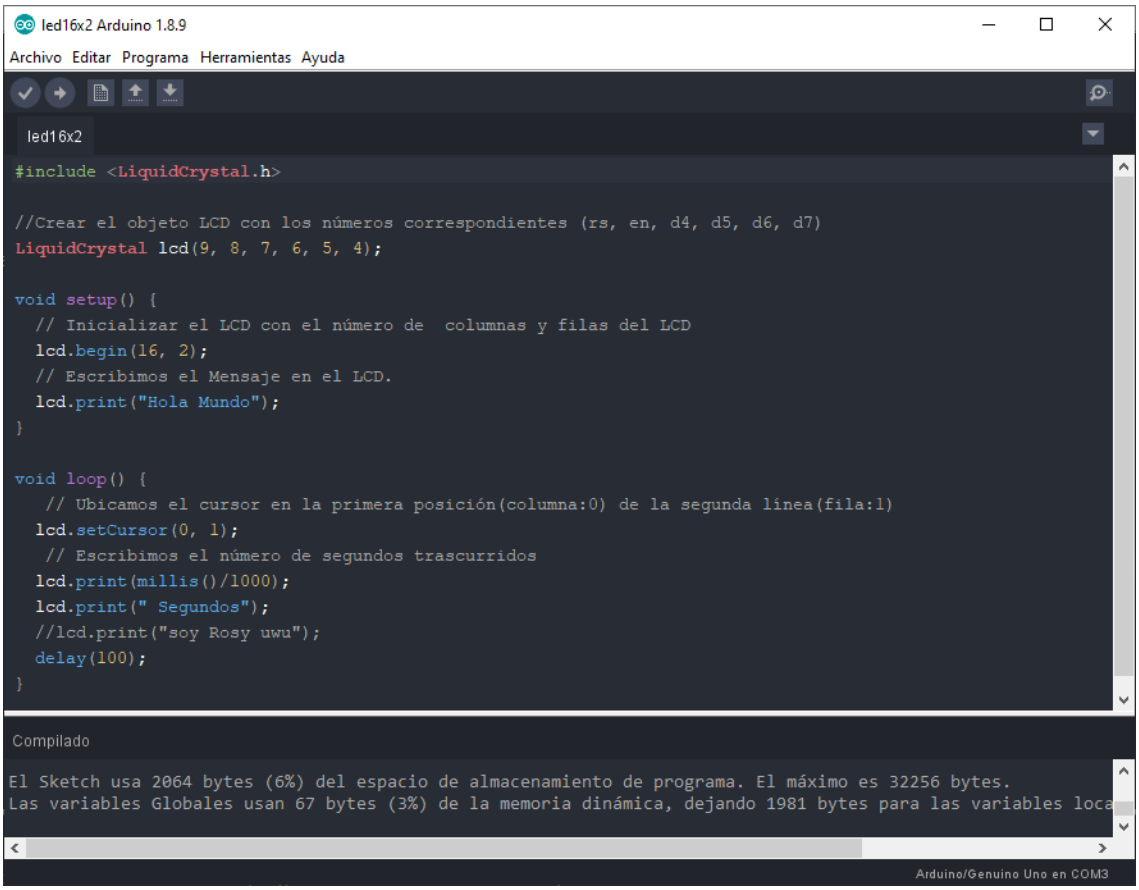


Ilustración 4: Arduino IDE [Captura de pantalla tomada por Macoritto Torcivia]

2.2.2. Visuino

Visuino es un IDE desarrollado por Mitov Software que permite programar placas Arduino (entre otras) de manera visual. Este software cuenta con una interfaz gráfica que facilita la programación al poder arrastrar componentes y conectarlos (Ilustración 5).

Los diseños de las placas y los periféricos son muy genéricos (similares a diagramas de clases), lo cual dificulta el desarrollo a personas que no están familiarizadas con electrónica y Arduino. Por otra parte, este software cuenta un excelente sistema de monitoreo de sensores que otros IDE no poseen.

Visuino solo está disponible para el sistema operativo Windows con uso bajo licencia no libre en sus versiones *Pro* y *free*.

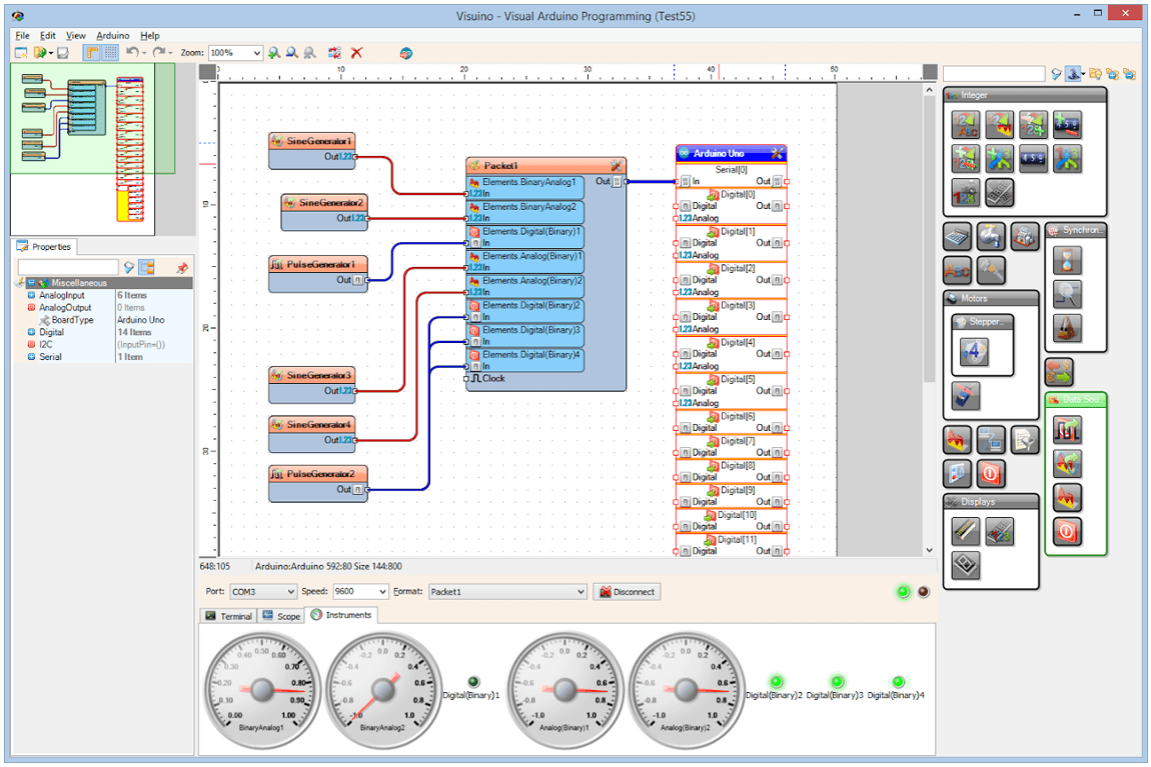


Ilustración 5: Visuino [Tomado de https://www.visuino.com]

2.2.3. S4A

Scratch for Arduino, abreviado S4A, es un software (basado en Scratch) que utiliza una interfaz gráfica de bloques para la codificación de proyectos en Arduino (Ilustración 6). Se encuentra disponible para los sistemas operativos Linux, MacOS y Windows.

Al igual que Scratch, este programa cuenta con una estructura de bloques y conectores que reemplazan a la escritura de código tradicional. Esto resulta muy práctico a la hora de programar para los niños y adolescentes que inician en el mundo de Arduino, pero no tanto para alguien con cierto nivel de experiencia. Otras soluciones similares a S4A son ArduBlock y Minibloq.

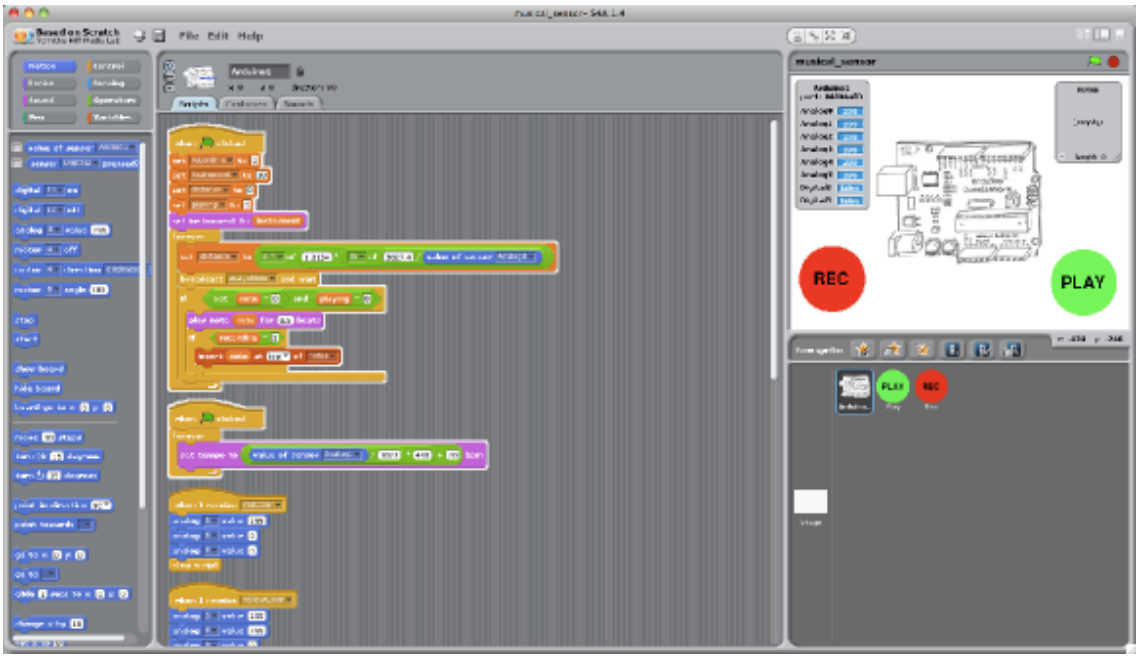


Ilustración 6: Scratch for Arduino [Tomado de http://s4a.cat/index_es.html]

2.2.4. Minibloq

Minibloq es un software de código abierto para el desarrollo de proyectos Arduino (y otras plataformas) que posee una interfaz gráfica de bloques (Ilustración 7). Actualmente está disponible en versión oficial para el sistema operativo Windows, y en versión preliminar para Linux. A diferencia de S4A, Minibloq cuenta con un generador de código fuente en tiempo real.

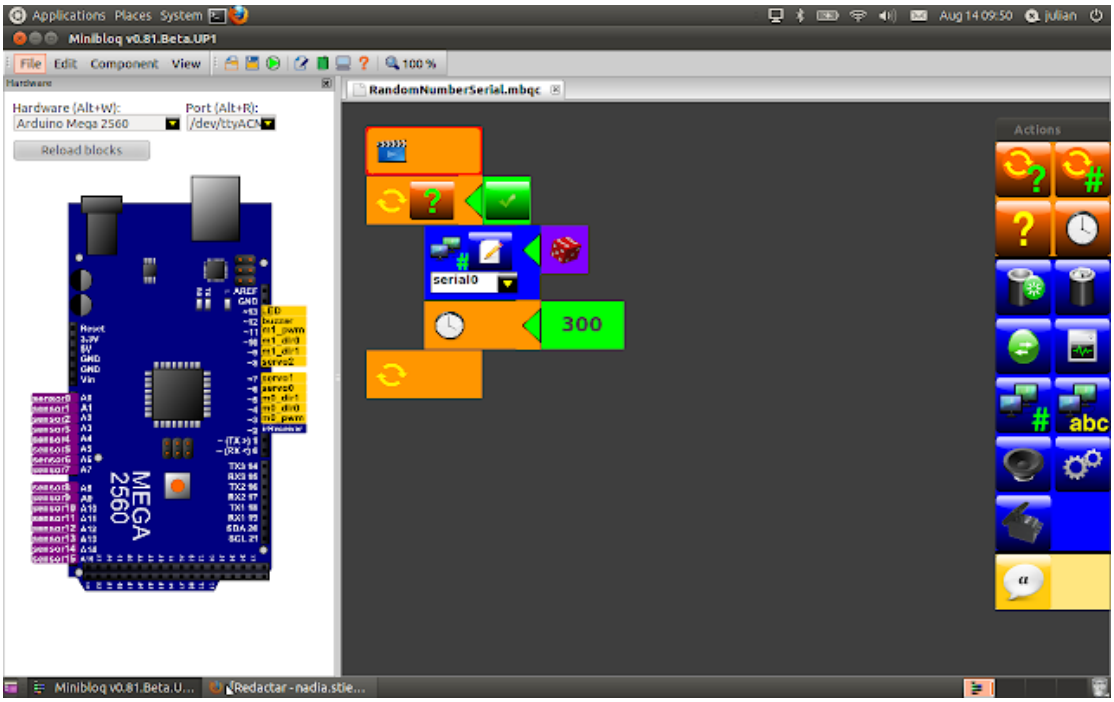


Ilustración 7: Minibloq [Tomado de http://blog.minibloq.org]

3. Definición del problema

3.1. Introducción

La plataforma Arduino facilita el control de circuitos y elementos electrónicos, pero programarlos suele resultar confuso para personas con poca experiencia en el rubro. Además, el IDE oficial de Arduino no provee de una interfaz gráfica y su editor de texto es escaso, mientras que alternativas como S4A o Visuino cuentan con editores gráficos para la configuración de pines, su lógica de programación basada en bloques resulta molesto y tedioso para alguien que posee conocimientos de programación, pero no de Arduino o electrónica.

Por esto se busca implementar un programa original que reúna las mejores características y herramientas de los IDE existentes para el desarrollo e implementación de proyectos Arduino, basándose en las mejores prácticas y estándares de programación, con el fin de brindar un entorno de desarrollo integrado sencillo y práctico para personas con o sin experiencia en programación, electrónica o robótica.

3.2. Objetivos

3.2.1. Objetivo general

Desarrollar un IDE (*Integrated Development Environment*) de uso libre y gratuito con el fin de facilitar la creación de proyectos realizados en Arduino.

3.2.2. Objetivos específicos

- Implementar un archivo de extensión que simplifique y almacene la configuración de los pines de la placa Arduino.
- Desarrollar un lenguaje de programación que ayude a escribir el código fuente del proyecto mediante el uso de la configuración de pines.
- Desarrollar un compilador que genere un archivo de extensión ino a partir del código fuente desarrollado.
- Desarrollar un depurador de programa que indique la ubicación y los tipos de errores en el código fuente.
- Diseñar una interfaz gráfica que facilite la configuración de pines e implemente diseños realistas de las placas Arduino, sensores y actuadores más utilizados.
- Desarrollar una página web que contenga la documentación del proyecto y almacene el código fuente y los archivos descargables del IDE.

3.3. Alcance

Desarrollar un compilador en su totalidad es una tarea muy compleja, "la mayoría de los científicos y profesionales de la computación nunca escribirían un compilador completo" [Louden, 2005]. Además, el compilador requerido en este proyecto no necesita generar un programa objetivo en lenguaje maquina o un código en lenguaje intermedio, sino obtener un archivo en C++ con la gramática del lenguaje de Arduino.

Para llevar a cabo dicha tarea, las dos primeras fases del proceso de un compilador son más que suficientes para generar un archivo traducido a C++ (con extensión ino), mediante la interpretación de un código escrito en el lenguaje de programación implementado. Por lo tanto, solo se desarrollarán estas dos fases, las cuales engloban:

- Un analizador léxico o rastreador (*scanner*) que reconozca tokens (palabras reservadas y tabla de símbolos) definidos mediante expresiones regulares o establecidos como constantes.

- Un analizador sintáctico (*parser*) que reciba los tokens provenientes del analizador léxico y, mediante el uso de los mismos y las definiciones de precedencias determine que la estructura del programa es acorde a la diseñada.

Estas fases generarán como salida el archivo de extensión ino a partir del código fuente escrito en el lenguaje de programación desarrollado, el cual constara de dos partes: la primera que contenga la lógica del programa y la segunda que almacene la configuración de los pines.

El depurador que se desarrollará en este proyecto estará limitado a indicar la ubicación (número de línea) y el tipo de error (carácter invalido o error de sintaxis) en el código fuente. Además, será integrado junto al compilador para facilitar su uso y mejorar el rendimiento del IDE.

La GUI implementada en el proyecto solo incluirá los diseños de los componentes y placas necesarias para el ensamblado y montaje de un robot. Estos componentes además de ser fáciles de conseguir en el marcado, suelen utilizarse para enseñar los principios básicos de la robótica y crear una gran variedad de robots, tanto simples como avanzados, con el fin de programarlos para que realicen diferentes funciones. Los elementos a incluir son:

- Arduino Uno R3
- Puente H L298N (para conectar 2 motores CC[2])
- CNY70 (sensor de color)
- LM35 (sensor de temperatura)
- HC-SR04 (sensor ultrasónico)
- Fotoresistor (sensor de luz)
- Servomotor SG90
- Matriz LED 8x8
- Módulo display de 4 dígitos
- Módulo joystick
- Buzzer
- LED's

[2] Corriente continua

- Pulsadores

3.4. Alternativas tecnológicas

3.4.1. Sistemas operativos

La elección del sistema operativo donde se llevará a cabo la solución planteada es indiferente, ya que hoy en día, tanto en un sistema Windows como en GNU/Linux se puede desarrollar el software propuesto con las mismas herramientas y tecnologías necesarias en ambas plataformas.

Por otro lado, el IDE será desarrollado para los sistemas operativos Windows 7 en adelante y distribuciones GNU/Linux basadas en Debian/Ubuntu que soporten el lenguaje de programación Python 3.8 o versiones superiores.

3.4.2. Lenguaje de programación

El lenguaje de programación que se utilice para el desarrollo deberá contar con librerías que permitan escribir tokens, expresiones regulares y gramáticas libre de contexto para facilitar el proceso e implementación del lenguaje planteado.

El lenguaje de programación Java cuenta con las librerías JFlex y Cup para generar los analizadores léxicos y sintácticos respectivamente. Para definir los tokens con JFlex se debe escribir un archivo de extensión .lex con los mismos, siguiendo un orden especifico e importando librerías externas de java al inicio del archivo. Escribir este archivo es molesto debido a que los editores de código actuales reconocen al mismo como un archivo de texto plano y, por ende, no cuentan con funciones tales como autocompletado y resaltadores de sintaxis.

Por otra parte, el analizador sintáctico en Java se genera utilizando la librería Cup, que de manera similar a JFlex se debe escribir un archivo de extensión .cup donde se definan los terminales, no terminales, la gramática y la precedencia del lenguaje, respetando ese mismo orden. Este archivo, al contrario que el archivo .lex es reconocido por los editores de código y cuenta con funciones de autocompletado y resaltado de sintaxis, pero es necesario descargar la extensión en el editor para poder utilizar dichas funciones. Una vez que se tiene ambos archivos (.lex y .cup) se deben compilar con un comando de la librería Cup para generar el código en Java y poder utilizar las gramáticas con los tokens definidos.

Para el lenguaje de programación Python existe una librería llamada PLY (Python Lex Yacc) la cual permite escribir los tokens y las gramáticas utilizando el mismo lenguaje Python, sin la necesidad de escribir archivos adicionales y compilarlos. Para el analizador léxico se definen los tokens y las expresiones regulares que se utilizarán en un archivo Python, y este se debe importar en otro archivo (también Python) donde se define el analizador sintáctico escribiendo las gramáticas libres de contexto como funciones de Python.

Analizando el procedimiento necesario para desarrollar un analizar léxico y sintáctico en ambos lenguajes, se llegó a la conclusión de que PLY en Python es más práctico y sencillo de utilizar que JFlex y Cup de Java, requiriendo menos archivos y facilitando la implementación al no tener que compilar cada vez que se quiera modificar el lenguaje.

4. Solución propuesta

4.1. Definición de la solución

Para llevar a cabo este proyecto, el IDE planteado en el objetivo general deberá contar con una interfaz gráfica, la cual permitirá realizar la configuración de los pines de la placa, es decir, la conexión de los distintos periféricos, actuadores y sensores con Arduino. Para esto se diseñarán modelos gráficos de los componentes definidos en el alcance. Además, se incluirá un modelo de la placa Arduino Uno R3 que permitirá seleccionar fácilmente los pines en los que se desea conectar los componentes, mediante el uso de una guía de colores que identifiquen el tipo de pin (analógico, digital o pwm) con las ranuras de la placa dónde pueden ser conectados. De esta manera no es necesario escribir código y se optimiza el almacenamiento de la configuración de los pines para reutilizarlos en futuros proyectos que compartan el mismo hardware.

Por otra parte, se diseñará un lenguaje de programación con una sintaxis similar a la de SQL y Python, con el fin de evitar el uso de símbolos tales como paréntesis, corchetes y llaves en código escrito. Esto facilitará a los usuarios con poca experiencia en programación definir la lógica del programa, ya que el lenguaje a desarrollar será más coloquial que los existentes.

A diferencia del lenguaje de programación oficial de Arduino, donde es necesario escribir un ciclo void para configurar los pines declarados previamente y, por otra parte, definir un ciclo *loop* para programar la lógica, el lenguaje propuesto permite reutilizar la configuración de pines realizada en la interfaz gráfica, para que solo sea necesario codificar la lógica del programa. Esta característica permite que se puedan implementar diferentes funciones para un mismo hardware sin la necesidad de volver a definir y configurar los pines en un ciclo *void*.

El depurador del IDE se definirá junto al compilador, de esta manera cuando se realice la compilación del programa, el mismo identificará los errores de sintaxis y semántica en el código (en el caso que los tuviera) con el fin de informar al usuario para su corrección. Por otra parte, el compilador generará como salida un código objetivo (equivalente al desarrollado) en el lenguaje de programación de Arduino, el cual se basa en C++.

En la Ilustración 8 se muestra un diagrama que detalla el funcionamiento del IDE mediante la implementación del editor de texto (que se utiliza como entrada para el código que contiene la lógica del programa), la GUI (que contiene la configuración de pines) y el compilador (que genera el código fuente en C++).

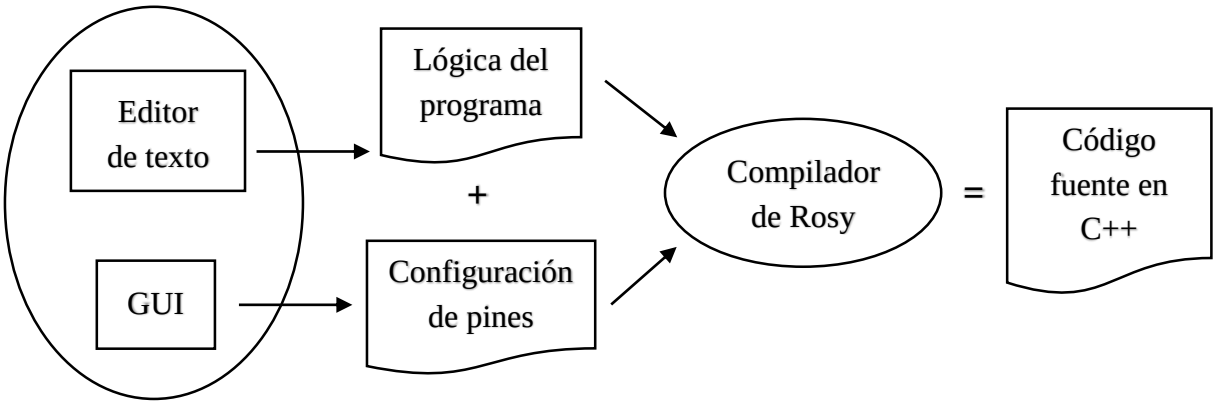


Ilustración 8: Diagrama del funcionamiento de Rosy IDE

El proyecto se realizará con el uso de la librería PLY (Python Lex Yacc). Se escogió esta herramienta debido a que es una alternativa práctica y potente, gracias a que utiliza el lenguaje de programación Python, cuya sintaxis facilita la escritura e interpretación del código desarrollado. Además, al ser un lenguaje interprete, la compilación del código fuente es más rápida que la de un lenguaje compilado.

Otra de las características que destaca a PLY es su simpleza en la forma de escribir las gramáticas y los tokens. En soluciones similares como JFlex y Cup de Java, para escribir la gramática era necesario crear un archivo de extensión *.lex* y *.cup* con las gramáticas del lenguaje, los tokens y las precedencias, luego se debía compilar dichos archivos para generar el código Java que interprete el lenguaje. Este proceso en el desarrollo se vuelve engorroso a medida que crece el lenguaje, mientras que en PLY, las gramáticas se escriben como una función de Python en el mismo archivo que se realiza el *parse* y la interpretación.

En cuanto a la GUI, ya que se utilizará Python para el desarrollo del compilador, con la librería grafica tkinter que viene integrada a Python y utiliza TCL para la definición de ventanas, botones, textos e imágenes. Los diseños de los componentes se representarán mediante imágenes a tamaño escala de los mismos.

El editor de texto contará con funciones básicas de edición como ser un buscador de líneas, para facilitar la visualización de errores gramaticales o de sintaxis y atajos de teclado para deshacer y rehacer acciones. Las palabras reservadas serán de colores, para obtener así una mejor visualización del código escrito y diferenciarlas de nombres de variables o funciones exclusivas de Arduino.

4.2. Metodología

La planificación del proyecto se llevará a cabo mediante la metodología del PMI (*Project Management Institute*), utilizando como referencia la guía del PMBOK quinta edición, mientras que para el desarrollo del mismo se implementará la una metodología incremental. Se optó por este modelo debido a que combina flujos de procesos lineales con paralelos que se ajustan al desarrollo de funcionalidades en simultaneo, lo cual permite progresar más rápido y reducir los tiempos de entrega del software, inclusive, cuando no se posee la cantidad de recursos humanos óptima para el proyecto.

El modelo incremental se enfoca en el avance por incrementos de un software con requerimientos iniciales bien definidos pero que requieren un mayor esfuerzo durante el desarrollo (Ilustración 9). Un incremento es un producto funcional (parte del software final) que opera de manera parcial y se utiliza para evaluar el progreso del proyecto de manera dinámica. Un incremento puede llevarse a cabo de manera paralela a otro, esto permite agilizar el desarrollo del proyecto y obtener una mejor retroalimentación. Todos los incrementos que se realizan durante el desarrollo conforman el producto final. [Pressman, 2010]

Esta metodología divide el trabajo en un conjunto de actividades estructurales:

1. Comunicación: inicio de actividad, recopilación de requisitos y seguimiento
2. Planeación: estimación, programación y análisis de riesgo
3. Modelado: análisis y diseño

4. Construcción: codificación y pruebas
5. Despliegue: entrega y retroalimentación

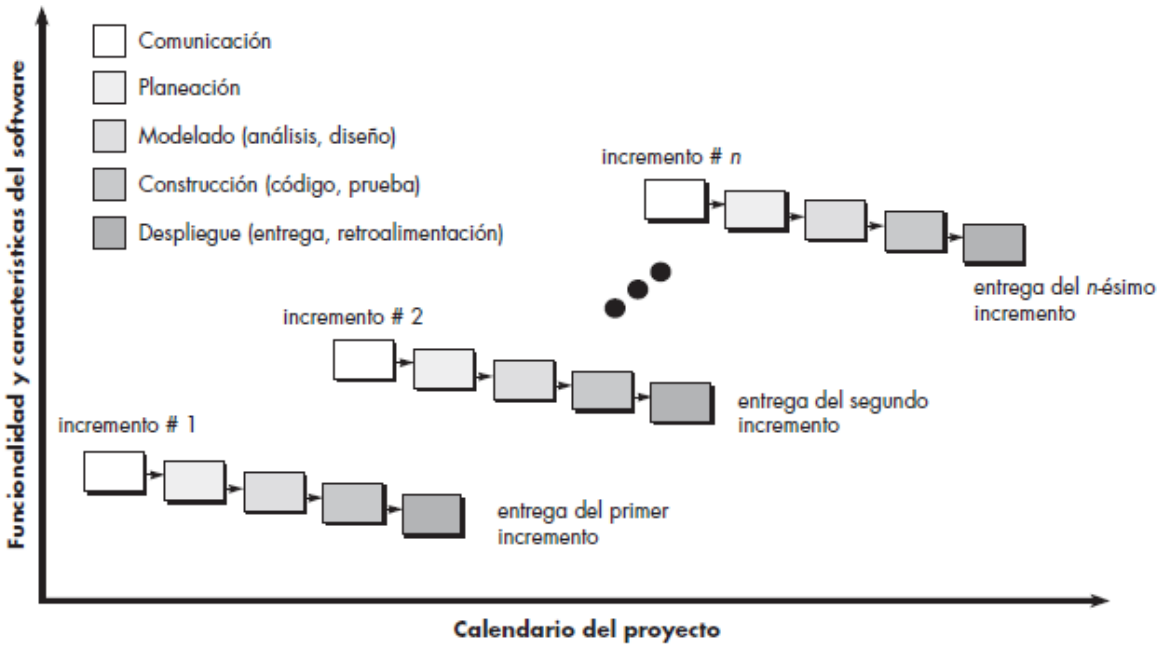


Ilustración 9: Modelo incremental de n incrementos [Tomado de Pressman, 2010]

Debido a que en este proyecto no hay una interacción continua con el cliente, ya que está destinado a la comunidad, la etapa de comunicación del modelo incremental se utilizará para una retroalimentación en cuanto al aprendizaje sobre la marcha del trabajo. Además, esto permitirá que se realicen ajustes a la definición original en caso de ser necesarios.

4.3. Planificación del proyecto

4.3.1. WBS

El WBS *(Work Breakdown Structure*), también conocido como EDT (Estructura de Descomposición del Trabajo) es una estructura jerárquica de la descomposición de las tareas definidas en el alcance con el fin de completar los objetivos del proyecto. [PMI, 2013]

En la Ilustración 10 se muestra el WBS del proyecto, el cual se divide en cinco incrementos. En cada uno se llevan a cabo tareas de análisis, diseño, codificación, pruebas e implementación, donde el resultado final del incremento es una parte fundamental del software propuesto que se utilizara como base funcional para el siguiente incremento.

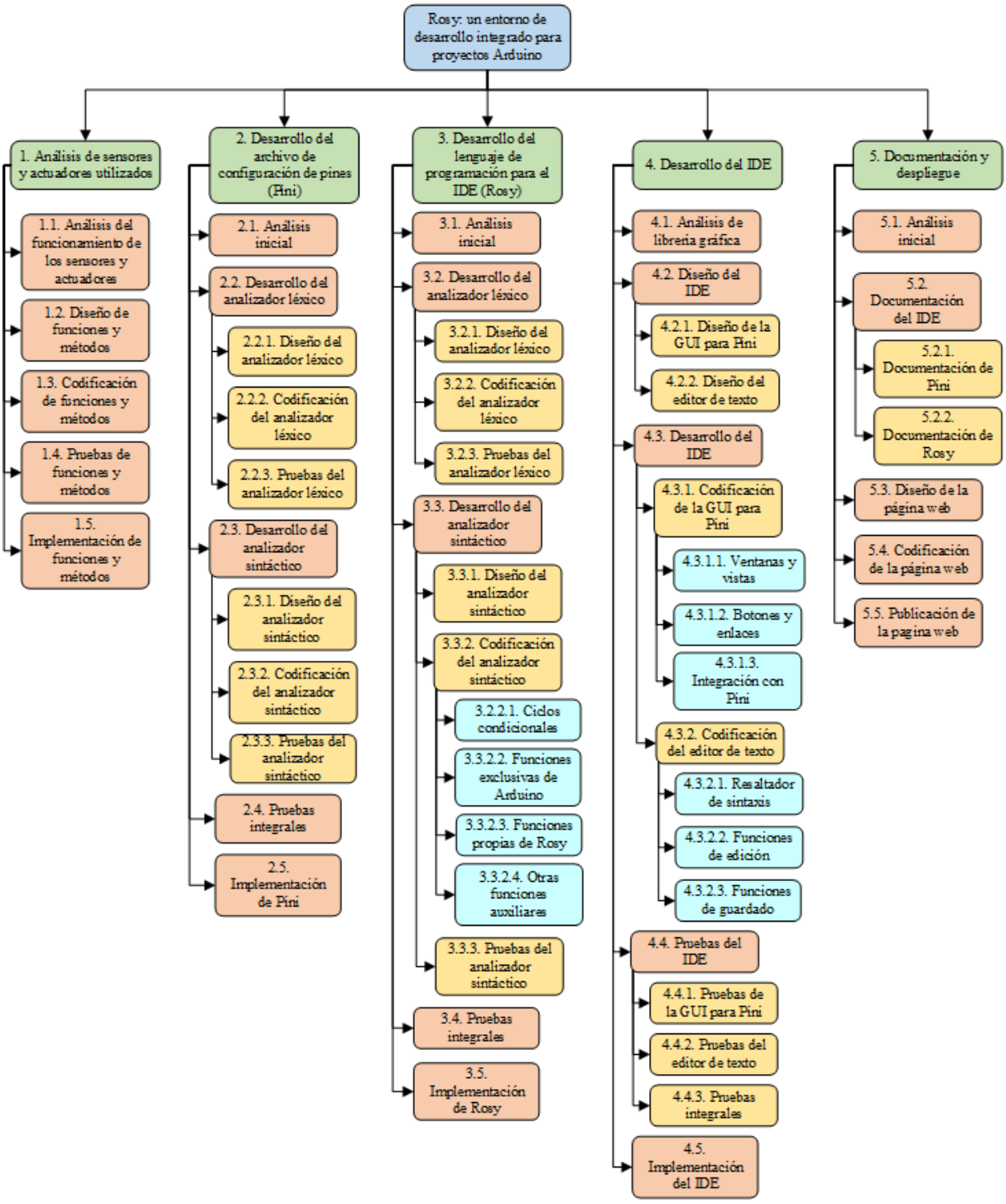


Ilustración 10: Diagrama WBS

4.3.2. Diccionario WBS

El diccionario WBS es un documento que proporciona información de cada componente del WBS, en el cual se describe detalladamente lo qué se realizará durante esa activad y los entregables que validarán la misma [PMI, 2013]. A continuación, se detalla el diccionario WBS del proyecto.

I. Análisis de sensores y actuadores utilizados

El primer incremento del proyecto consiste en el análisis de los sensores y actuadores que se van a utilizar en el IDE, los cuales fueron listados en el capítulo 3. El objetivo de ésta actividad es desarrollar funciones y métodos que faciliten el uso de los componentes con el fin de simplificar la gramática del lenguaje mediante el uso de los mismos.

I. Análisis del funcionamiento de los sensores y actuadores

Durante la etapa de análisis, se investigará sobre el funcionamiento de los componentes con el fin de comprender sus funciones para poder diseñar métodos eficaces en el lenguaje de Arduino y luego utilizarlas en los próximos incrementos.

II. Diseño de funciones y métodos

Una vez finalizado el análisis del funcionamiento de los sensores y actuadores utilizados se diseñarán funciones y procedimientos que simplifiquen su uso y faciliten la configuración de pines requerida para conectarse a la placa Arduino.

III. Codificación de funciones y métodos

En base al diseño de la actividad anterior, se codificarán las funciones y procedimientos en el lenguaje C++ especifico de Arduino para asegurar la compatibilidad con todas los modelos de placa Arduino.

IV. Pruebas de funciones y métodos

Se realizan pruebas unitarias por cada función y procedimiento desarrollado con el objetivo de asegurar su correcto funcionamiento. Si se encuentran errores, se volverá a la actividad de Diseño o Codificación en caso de ser necesario.

V. Implementación de funciones y métodos

El primer incremento se dará por concluido una vez que se obtengan las funciones y métodos de los componentes y sensores utilizados en el proyecto, los cuales serán implementados durante los siguientes incrementos.

II. Desarrollo del archivo de configuración de pines (Pini)

El segundo incremento consiste en el desarrollo del archivo de configuración de pines para la placa denominado Pini. Este archivo cumplirá la función de almacenar las conexiones a la placa Arduino de los sensores y actuadores que se utilicen para un proyecto en particular, con el fin de permitir la reutilización de dicha configuración en proyectos que requieran el mismo hardware.

I. Análisis inicial

Durante la etapa de análisis inicial se investigará sobre el funcionamiento del ciclo *setup* de un programa desarrollado en Arduino, el cual sirve para establecer los parámetros iniciales de los pines definidos previamente, que serán utilizados durante la ejecución del programa.

II. Desarrollo del analizador léxico

El desarrollo del analizador léxico de Pini consiste en el diseño, codificación y pruebas de los tokens y palabras reservadas que serán utilizadas por el analizador sintáctico en la definición de las gramáticas.

i. Diseño del analizador léxico

En la etapa de diseño del analizador léxico se definirán los tokens, palabras claves y expresiones regulares necesarias para configurar los sensores y actuadores definidos en el alcance.

ii. Codificación del analizador léxico

En base al diseño de la actividad anterior se codificarán los tokens, palabras claves y expresiones regulares en Python mediante el uso de la librería Lex, perteneciente a PLY, la cual facilita el desarrollo de analizadores léxicos. Esta herramienta, al mismo tiempo, permite identificar cuando se ingresa un token no definido, generando un “error” de sintaxis del tipo carácter inválido que será utilizado por el depurador cuando se compile el programa.

iii. Pruebas del analizador léxico

Se realizan pruebas ingresando tokens y palabras no definidas con el objetivo de controlar el resultado del analizador léxico y verificar que el mensaje de error sea correcto. En caso de encontrar errores que no fueron contemplados durante la codificación y el diseño, se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

III. Desarrollo del analizador sintáctico

Esta etapa requiere que se completen las actividades anteriores donde se lleva a cabo el desarrollo del analizador léxico, ya que para definir la gramática de Pini, se utilizan los tokens y palabras claves generadas por Lex.

i. Diseño del analizador sintáctico

El diseño del analizador sintáctico consiste en definir todas las gramáticas regulares que se utilizarán en el archivo de configuración de pines, junto a la declaración de las precedencias y los terminales de las mismas, para evitar conflictos de ambigüedad durante la compilación.

ii. Codificación del analizador sintáctico

En base al diseño realizado en la etapa anterior, se codificarán las gramáticas regulares y se establecerán las precedencias de las mismas en Python mediante el uso de la librería Yacc, perteneciente a PLY. Se importarán los token y palabras claves desarrollados previamente para definir la gramática del lenguaje, además, se simplificará la conexión de los componentes a la placa al hacer uso de las funciones y métodos implementados durante el primer incremento.

iii. Pruebas del analizador sintáctico

Se realizarán pruebas para determinar el correcto funcionamiento del compilador y validar los errores del tipo "error de sintaxis" y "final de línea inesperado". En caso de encontrar errores que no fueron contemplados durante la codificación y el diseño, se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

IV. Pruebas integrales

Una vez finalizada las etapas de desarrollo de los analizadores léxicos y sintácticos, se realizarán pruebas que integren ambos componentes con el fin de corroborar su correcto funcionamiento en conjunto. En caso que surjan errores no previstos se podrá volver a las etapas anteriores para realizar las correcciones y los cambios necesarios.

V. Implementación de Pini

La etapa de implementación de Pini dará por concluido el segundo incremento, al obtener como entregable el programa funcional que permite generar un código fuente en C++ de la configuración de pines especificadas en un archivo que utiliza el lenguaje desarrollado.

III. Desarrollo del lenguaje de programación para el IDE (Rosy)

El tercer incremento consiste en el desarrollo del lenguaje de programación para la lógica del programa, denominado Rosy. Éste utilizará el archivo de configuración de pines desarrollado en el incremento anterior.

I. Análisis inicial

Durante la etapa de análisis inicial se investigará sobre el funcionamiento del ciclo *void* de un programa desarrollado en Arduino, el cual sirve para definir la lógica del programa utilizando los pines configurados previamente en el ciclo *setup*.

II. Desarrollo del analizador léxico

El desarrollo del analizador léxico de Rosy consiste en el diseño, codificación y pruebas de los tokens y palabras reservadas que serán utilizadas por el analizador sintáctico en la definición de las gramáticas.

i. Diseño del analizador léxico

En la etapa de diseño del analizador léxico se definirán los tokens, palabras claves y expresiones regulares necesarias para la gramática y sintaxis del lenguaje.

ii. Codificación del analizador léxico

En base al diseño de la actividad anterior se codificarán los tokens, palabras claves y expresiones regulares en Python mediante el uso de la librería Lex, perteneciente a PLY, la cual facilita el desarrollo de analizadores léxicos. Esta herramienta, al mismo tiempo, permite identificar cuando se ingresa un token no definido, generando un "error" de sintaxis del tipo carácter inválido que será utilizado por el depurador cuando se compile el programa.

a. Ciclos condicionales

Se definirán los ciclos condónales básicos de un lenguaje de programación tales como *if, for, while* y *do while*. Cada ciclo permitirá el uso de una condición lógica simple de tipo AND y OR con operadores mayor, menor, distinto, igual, mayor o igual y menor o igual.

b. Funciones exclusivas de Arduino

Se codificarán las funciones exclusivas de Arduino en el lenguaje Rosy y, además, se simplificará el uso de analogRead y digitalRead al definir una única función llamada READ tanto para pines digitales como analógicos.

c. Funciones propias de Rosy

Estas funciones serán diseñadas con el objetivo de facilitar el uso de los componentes, actuadores y sensores definidos en el alcance del proyecto.

d. Otras funciones auxiliares

Las funciones auxiliares definidas ayudarán o complementarán al resto de funciones y ciclos condicionales definidos anteriormente.

iii. Pruebas del analizador léxico

Se realizan pruebas ingresando tokens y palabras no definidas con el objetivo de controlar el resultado del analizador léxico y verificar que el mensaje de error sea correcto. En

caso de encontrar errores que no fueron contemplados durante la codificación y el diseño, se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

III. Desarrollo del analizador sintáctico

Esta etapa requiere que se completen las actividades anteriores donde se lleva a cabo el desarrollo del analizador léxico, ya que, para definir la gramática de Rosy se utilizan los tokens y palabras claves generadas por Lex.

i. Diseño del analizador sintáctico

El diseño del analizador sintáctico consiste en definir todas las reglas gramaticales que se utilizarán en el lenguaje de programación propuesto, junto a la declaración de las precedencias y los terminales de las mismas, para evitar conflictos de ambigüedad durante la compilación.

ii. Codificación del analizador sintáctico

En base al diseño realizado en la etapa anterior, se codificarán las reglas gramaticales y se establecerán las precedencias de las mismas en Python mediante el uso de la librería Yacc, perteneciente a PLY. Se importarán los token y palabras claves desarrollados previamente para definir la gramática del lenguaje, además, se simplificará la programación de los actuadores y sensores al hacer uso de las funciones y métodos implementados durante el primer incremento.

iii. Pruebas del analizador sintáctico

Se realizarán pruebas para determinar el correcto funcionamiento del compilador y validar los errores del tipo “error de sintaxis” y “final de línea inesperado”. En caso de encontrar errores que no fueron contemplados durante la codificación y el diseño, se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

IV. Pruebas integrales

Una vez finalizada las etapas de desarrollo de los analizadores léxicos y sintácticos, se realizarán pruebas que integren ambos componentes con el fin de corroborar su correcto funcionamiento en conjunto. En caso que surjan errores no previstos se podrá volver a las etapas anteriores para realizar las correcciones y los cambios necesarios.

V. Implementación de Rosy

La etapa de implementación de Rosy dará por concluido el tercer incremento al obtener como entregable el programa funcional que permite generar un código fuente en C++ para ser interpretado por el IDE de Arduino. Este código se obtiene compilando lo programado en Rosy junto a la configuración de pines definida en Pini.

IV. Desarrollo del IDE

El cuarto incremento consiste en el desarrollo del IDE en sí mismo, implementando el archivo de configuración de pines Pini y el lenguaje de programación Rosy desarrollados en los incrementos anteriores. El IDE contará con un editor de texto para Rosy y una interfaz gráfica para Pini.

I. Análisis de librería gráfica

Durante la etapa de análisis se investigará sobre el funcionamiento de tkinter, la librería gráfica, que se utilizará para el desarrollo del IDE con el fin de comprender los procesos y funciones requeridas por tkinter para llevar a cabo las siguientes etapas de forma óptima, aprovechando las herramientas que brinda esta librería de Python.

II. Diseño del IDE

El diseño del IDE engloba al diseño de la GUI para Pini y al diseño del editor de texto.

i. Diseño de la GUI para Pini

En esta etapa se diseñará la interfaz gráfica para la configuración de pines, la cual tendrá imágenes a escala[3] de los componentes definidos en el alcance del proyecto. Cuando el usuario hace click sobre la figura del componente que desea agregar, podrá seleccionar él o los pines mediante los cuales se conectara a la placa, además, deberá indicar un nombre para identificar y diferenciar al componente cuando se programe la lógica del programa.

ii. Diseño del editor de texto

El diseño del editor de texto contará con resaltador de sintaxis, que permitirá diferenciar las palabras reservadas, los nombres de las funciones y las variables definidas durante la escritura del código. Además, se incluirán funciones para deshacer y rehacer cambios, atajos de teclado para copiar y pegar texto y un buscador de línea de código.

III. Desarrollo del IDE

El desarrollo del IDE engloba el desarrollo de la GUI para Pini y el editor de texto.

i. Codificación de la GUI para Pini

En la etapa de codificación de la interfaz gráfica para Pini se la desarrollará siguiendo el diseño de la actividad anterior. Se programarán las acciones de los botones y se implementará el compilador de Pini desarrollado durante el segundo incremento.

[3] La escala utilizada equivale a 1 cm igual a 1 pixel

a. Ventanas y vistas

El primer paso para la codificación de la GUI es definir las ventanas y vistas que tendrá la misma. Se utilizará una ventana principal dividida en dos vistas, la cual contendrá el editor de texto y una figura de la placa Arduino UNO con sus pines remarcados en colores según sean analógicos, digitales o pwm. También, contará con una ventana que contenga las figuras de los sensores y actuadores definidos en el alcance.

b. Botones y enlaces

Luego se codificarán los botones que permitan abrir y cerrar las vistas, ventas y los menús de la interfaz gráfica, junto a los enlaces que lleven al sitio web de la documentación del lenguaje.

c. Integración con Pini

Por último, se integrarán los botones y las figuras de la GUI con el generador sintáctico de Pini para que, al hacer click sobre una, se autocomplete el código de la misma con los pines configurados.

ii. Codificación del editor de texto

En la etapa de codificación del editor de texto para Rosy se desarrollará el mismo en base al diseño de la actividad anterior. Además, se programará la lógica para las funciones deshacer, rehacer, los atajos de teclado y el buscador de líneas. Esta etapa utiliza el compilador de Rosy desarrollado en el tercer incremento. También se programarán los botones para compilar el código junto a la configuración de pines y los cuadros de dialogo de errores de sintaxis del depurador.

a. Resaltador de sintaxis

El resaltador de sintaxis de programará con el fin de marcar de distintos colores los tokens y palabras reservadas de la gramática de Rosy, definidas previamente, para facilitar la lectura y comprensión del código.

b. Funciones de edición

Se implementarán funciones de edición de texto tales como deshacer cambios, rehacer cambios y un buscador de líneas de código para agilizar la escritura de los programas.

c. Funciones de guardado

El editor de texto contará con funciones de guardar un nuevo archivo, guardar cambios y abrir archivos existentes, además, se incorporarán controles para evitar el cierre accidental del editor en caso que se encuentren cambios sin guardar.

IV. Pruebas del IDE

Las pruebas del IDE engloban las pruebas de la GUI para Pini, las del editor de texto y las pruebas integrales.

ii. Pruebas de la GUI para Pini

Se realizarán pruebas individuales para determinar el correcto funcionamiento de la interfaz gráfica de Pini en conjunto a su compilador. En caso que surjan errores no previstos, se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

ii. Pruebas del editor de texto

Se realizarán pruebas que determinen el correcto funcionamiento del editor de texto junto a todas sus funciones desarrolladas. Además, se verificará que el compilador genere correctamente el programa fuente en C++ tomando el código escrito en el editor de texto. En caso que surjan errores no previstos se podrá volver a una de las etapas anteriores para realizar las correcciones necesarias.

iii. Pruebas integrales

Una vez finalizadas las pruebas individuales de la interfaz gráfica y el editor de texto, se procederá a realizar pruebas integrales que aseguren el correcto funcionamiento de ambas herramientas en conjunto. En caso que surjan errores no previstos, se podrá volver a las etapas anteriores para realizar las correcciones y los cambios necesarios.

V. Implementación del IDE

Esta etapa dará por concluido el cuarto incremento al obtener como entregable funcional el código fuente del IDE y sus correspondientes archivos ejecutables para los sistemas operativos Windows y GNU/Linux definidos en el alcance.

V. Documentación y despliegue

El último incremento del proyecto consiste en la documentación del funcionamiento del IDE, junto a un instructivo para su instalación en los diferentes sistemas operativos. Todo esto estará disponible en una página web junto a los archivos ejecutables del IDE.

I. Análisis inicial

Durante la etapa de análisis inicial se plantearán ejemplos prácticos sobre la utilización del IDE, empleando los diferentes sensores y actuadores definidos en el alcance, con el fin de mostrar el funcionamiento de los mismos y la implementación de los métodos que facilitan su uso.

II. Documentación del IDE

La documentación del IDE engloba la documentación del archivo de configuración de pines Pini y del lenguaje de programación Rosy.

i. Documentación de Pini

Se documentará el uso de la interfaz gráfica junto a un ejemplo de cómo utilizar cada componente definido en el alcance y los pasos necesarios para incluirlo en la lógica del programa.

ii. Documentación de Rosy

Se documentará la estructura del lenguaje, su sintaxis y la definición y uso de cada una de las funciones que poseen los actuadores y sensores incluidos en el alcance del proyecto. También se incluirá una descripción de los tipos de datos admitidos por el lenguaje, los ciclos iterativos válidos y cómo definir variables de tipo global y constantes. Cada función de los componentes contará con un ejemplo práctico en donde se muestre el uso y la sintaxis de la misma.

III. Diseño de la página web

Se diseñará una página web que contenga la documentación realizada durante las actividades anteriores con sus respectivos ejemplos prácticos y que, además, aloje los archivos ejecutables del IDE para su descarga e instalación en los correspondientes sistemas operativos.

IV. Codificación de la página web

Durante la etapa de codificación se programarán los botones de descarga de los archivos ejecutables del IDE y se realizarán las distintas vistas que contengan la documentación y los instructivos para instalar el programa en cada sistema operativo.

V. Publicación de la página web

La última etapa del proyecto dará por concluido al mismo, al obtener como entregable funcional la página web que albergue la documentación y los enlaces de descarga de los archivos ejecutables del IDE.

4.3.3. Recursos necesarios

Para llevar a cabo este proyecto son necesarios los siguientes recursos humanos y tecnológicos:

- Un líder de proyecto: es el encargado de dirigir y controlar el avance del proyecto. Su principal función consiste en revisar los entregables de cada iteración y corroborar que estos cumplan con lo especificado en la etapa de análisis y diseño. Además de estas tareas, el líder de proyecto o Project Manager deberá realizar reuniones semanales de

no más de dos horas con el fin de controlar el seguimiento del proyecto con los demás integrantes del equipo de desarrollo.

- Un analista funcional: su función se basa en llevar a cabo las investigaciones de las etapas de análisis, recopilar información y establecer los requerimientos tecnológicos para cada incremento. A su vez, se encarga de documentar el software resultante de los incrementos.

- Un analista programador: cumple el rol de supervisar a los programadores durante las etapas de desarrollo y prueba de cada iteración del proyecto. Además, es el responsable de realizar el diseño de las fases del programa.

- Dos programadores: son los encargados de la codificación y prueba para el desarrollo del proyecto. Debido a que la codificación es la fase que más tiempo conlleva en un desarrollo de software, los programadores se repartirán las tareas con el fin de aprovechar el tiempo disponible para completar el proyecto.

- Un tester: es un programador con experiencia en QA[4] que se dedica de manera exclusiva a probar las funcionalidades del software desarrollado y encontrar posibles fallos en las mismas para su posterior corrección por los programadores.

- Dos computadoras portátiles: es necesario la adquisición de dos computadoras portátiles, las cuales serán utilizadas por todo el equipo de trabajo, pero especialmente por los desarrolladores. De esta manera los profesionales contaran con los recursos y programas necesarios para el desarrollo y pruebas del proyecto. Las computadoras deberán poseer un hardware actualizado con el fin de que el software requerido se ejecute sin inconvenientes.

- Conexión a internet: además de las computadoras portátiles se requiere una conexión a internet de por lo menos 50MB para descargar el software necesario para el desarrollo del proyecto, llevar a cabo las investigaciones de la etapa de análisis y documentar los resultados de cada actividad.

4.3.4. Gantt

El diagrama de Gantt muestra la relación entre el tiempo, las tareas del WBS y los recursos necesarios para llevar a cabo el proyecto en tiempo y forma. En el diagrama especificado, un día laboral equivale a 8 horas.

El proyecto tiene una duración de 121 días hábiles, es decir 968 horas, lo que equivale a 6 meses y 1 día aproximadamente. En la Ilustración 11, se muestra una versión resumida del mismo, el detallado se encuentra en los Anexos.

[4] Corresponde a las siglas del proceso o conjunto de actividades *Quality Assurance*, que tienen como objetivo garantizar y asegurar la calidad de un software.

Rosy IDE: un entorno de desarrollo integrado para proyectos Arduino	**121 días**
1. Análisis de sensores y actuadores utilizados	**14 días**
1.1. Análisis del funcionamiento de los sensores y actuadores	2 días
1.2. Diseño de funciones y métodos	3 días
1.3. Codificación de funciones y métodos	5 días
1.4. Pruebas de funciones y métodos	3 días
1.5. Implementación de funciones y métodos	1 día
2. Desarrollo del archivo de configuración de pines (Pini)	**25 días**
2.1. Análisis inicial	2 días
2.2. Desarrollo del analizador léxico	8 días
2.3. Desarrollo del analizador sintáctico	12 días
2.4. Pruebas integrales	2 días
2.5. Implementación de Pini	1 día
3. Desarrollo del lenguaje de programación para el IDE (Rosy)	**45 días**
3.1. Análisis inicial	2 días
3.2. Desarrollo del analizado léxico	9 días
3.3. Desarrollo del analizador sintáctico	31 días
3.4. Pruebas integrales	4 días
3.5. Implementación de Rosy	1 día
4. Desarrollo del IDE	**36 días**
4.1. Análisis de librería gráfica	2 días
4.2. Diseño del IDE	8 días
4.3. Desarrollo del IDE	15 días
4.4. Pruebas del IDE	12 días
4.5. Implementación del IDE	2 días
5. Documentación y despliegue	**20 días**
5.1. Análisis inicial	2 días
5.2. Documentación del IDE	6 días
5.3. Diseño de la página web	3 días
5.4. Codificación de la página web	7 días
5.5. Publicación de la página web	2 días

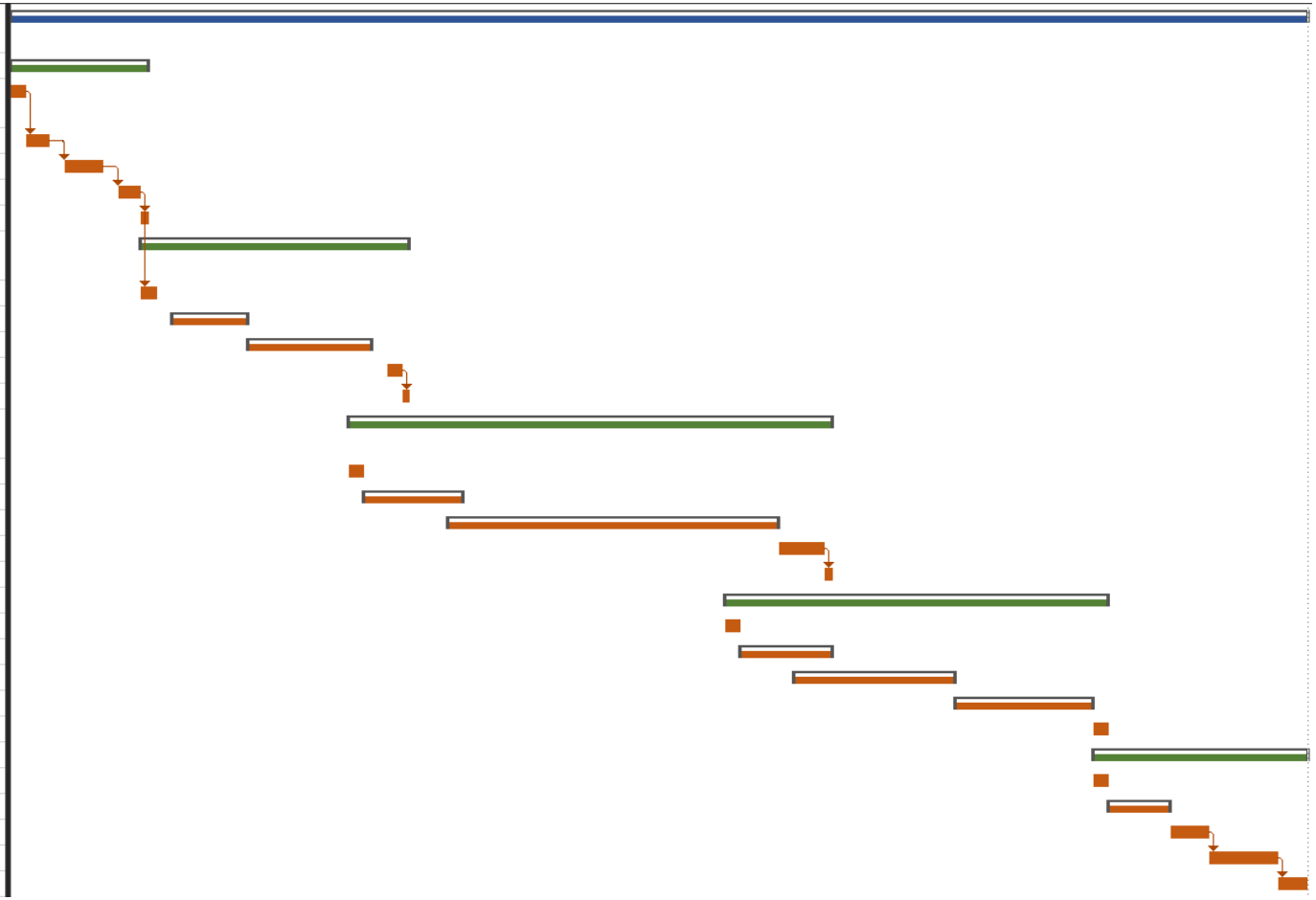


Ilustración 11: Gantt del proyecto (resumido)

4.3.5. Análisis de costos

En base a los recursos y los tiempos estimados en el diagrama de Gantt, se calculan los costos necesarios para el desarrollo del proyecto. Los mismos fueron valuados en pesos argentinos ($ARS) con sus respectivos montos correspondientes a la fecha 28/07/2022. En la Tabla 1 se detallan los costos de cada rol de RRHH, los cuales fueron tomados en base a la ficha oficial de COPAIPA (Consejo Profesional de Agrimensores, Ingenieros y Profesiones Afines). En la Tabla 2 se detallan los costos de hardware y servicios.

Tabla 1: Costos de los RRHH

RRHH			
Función	**Costo/h**	**Hs.**	**Total**
Project Manager	$ 1560,00	56	$ 87.360,00
Analista Funcional	$ 1040,00	128	$ 133.120,00
Analista Programador	$ 870,00	216	$ 187.920,00
Programador 1	$ 690,00	288	$ 198.720,00
Programador 2	$ 690,00	288	$ 198.720,00
Tester	$ 690,00	264	$ 182.160,00
			$ 988.000,00

Tabla 2: Costos del hardware y servicios

Hardware y Servicios			
Detalle	**Cant.**	**Precio**	**Total**
Notebook Lenovo V15 ADA, Intel Core i3, 4GB RAM, 1TB, sin S.O.	2	$ 99.999,00	$ 199.998,00
Licencia Ubuntu 20.04	2	$ 0,00	$ 0,00
Licencia Software Arduino 1.8	2	$ 0,00	$ 0,00
Licencia editor de texto Atom	2	$ 0,00	$ 0,00
Licencia Visual Studio Code	2	$ 0,00	$ 0,00
Hosting Firebase	1	$ 0,00	$ 0,00
Nombre de dominio rosy.com.ar	1	$ 450,00	$ 450,00
Internet Claro 50MB x7194 mes (primer mes gratis)	6	$ 1.199,00	$ 7.194,00
Kit Arduino UNO R3 + Protoboard + resistencias + fotoresistor + pulsadores + cables + LEDs	1	$ 8.888,00	$ 8.888,00
Sensor de color CNY70	1	$ 375,00	$ 375,00
Sensor de temperatura LM35	1	$ 645,00	$ 645,00
Buzzer	1	$ 375,00	$ 375,00
Servomotor SG90	1	$ 800,00	$ 800,00
Sensor ultrasónico HC-SR04	1	$ 600,00	$ 600,00

Motor DC 3V-6V	2	$ 900,00	$ 1.800,00
Puente H L298N	1	$ 1.180,00	$ 1.180,00
Módulo display de 4 dígitos	1	$ 500,00	$ 500,00
Módulo joystick	1	$ 490,00	$ 490,00
Matriz LED de 8x8	1	$ 880,00	$ 880,00
			$ 224.175,00

Debido a que en el desarrollo del proyecto se utilizan herramientas de software libre y gratuito no hay costos adicionales referidos a licencias de uso. Sumando los costos de los recursos humanos, el hardware y los servicios necesarios se obtiene un total de **$ 1.212.175** (un millón doscientos doce ciento setenta y cinco pesos argentinos).

4.3.5.1 Costo beneficio

La esencia de este proyecto se basa en la filosofía del software libre y tiene como objeto ser de utilidad a la comunidad, especialmente la estudiantil, para agilizar el desarrollo y aprendizaje de la robótica y programación. Incluir una licencia de uso paga contradice estos fundamentos y principios y, además, en el objetivo general del proyecto se establece que el software desarrollado será gratuito, por ende, no tendría sentido abonar por el mismo o pagar una cuota para poder utilizarlo.

Por lo tanto, las ganancias del software desarrollado en el proyecto no pueden medirse de manera directa en forma monetaria, sino que se debe estimar el valor del beneficio que se obtiene al hacer uso del mismo.

Para encontrar el valor que genera su uso, se tiene en cuenta un factor de vital importancia en todo proyecto: el tiempo. Rosy IDE, al contar con una interfaz gráfica intuitiva y un lenguaje de programación coloquial, resulta más sencillo de aprender que otras soluciones alternativas, lo que significa que su curva de aprendizaje es menor y requiere menos tiempo y esfuerzo para los usuarios. A su vez, el almacenamiento de la configuración de pines en archivos Pini permite reutilizarlos en proyectos similares y evita volver a escribir el mismo código, lo cual ahorra bastante tiempo de desarrollo.

Teniendo en cuenta la ventaja en términos de tiempo, tanto de aprendizajes como de desarrollo que proporciona el uso de Rosy frente a otras alternativas, se plantea el siguiente análisis.

Según el Ministerio de Educación de la Nación, en Argentina existe un total de 1445 escuelas técnicas, de las cuales 577 tienen orientaciones afines al proyecto desarrollado, es decir, informática, electrónica o electromecánica. [Ministerio de Educación de la Nación Argentina, 2017]

Las materias dictadas en las escuelas técnicas relacionadas a la informática, robótica o electrónica, como por ejemplo **Introducción a la programación** o **Laboratorio de**

dispositivos digitales programables[5], poseen una carga horaria de 5 horas cátedras, las cuales equivalen aproximadamente a 3 horas reloj a la semana, lo que da un total de 96 horas al año[6].

Si en dichas materias se utiliza Rosy IDE al menos durante una hora práctica de las 3 horas semanales (total de 32 horas al año), se facilitaría el aprendizaje de las temáticas planteadas a los alumnos. Esto reduciría al menos un tercio del tiempo que tardarían en realizar la práctica al utilizar otro software, lo cual da como resultado un valor aproximado de 10 horas y 30 minutos.

El sueldo promedio de un profesor con dedicación de 20 horas mensuales es de $ 40.000 (cuarenta mil pesos argentinos) aproximadamente, lo que equivale a $ 2.000 (dos mil pesos argentinos) por hora. De esta manera, al realizar la multiplicación de las horas ahorradas que genera el uso de Rosy IDE (10,5 h.) por el sueldo/hora de un profesor se obtiene un total de $ 21.000 (veintiún mil pesos argentinos) al año. Éste valor se toma como referencia con el fin de asignar un monto a los beneficios generados por el uso del proyecto, en el cual el tiempo que se ahorra el profesor al utilizar Rosy, puede ser ocupado en profundizar la temática de la materia o avanzar en el dictado de otras unidades curriculares. [Sindicato de Investigadores y Docentes de la UTN, 2022]

Al realizar este cálculo por cada escuela técnica del país (577 escuelas) el resultado final es de $ 12.117.000 (doce millones ciento diecisiete mil pesos argentinos) y, si bien el presente caso es ideal, bastaría con que solo el 10% de las escuelas técnicas (58 escuelas aproximadamente) utilicen el software durante un año. De esta manera se obtiene un monto de $ 1.218.000 (un millón doscientos dieciocho mil pesos argentinos), el cual cubre el costo total del proyecto equivalente a $ 1.212.175 (un millón doscientos doce ciento setenta y cinco pesos argentinos).

4.3.5.2. Flujo de caja

En base al análisis realizado anteriormente, se obtiene el siguiente flujo de caja:

- TNA = 95%
- TEM = 7,92%
- Ingreso mensual = $ 101.500,00[7]

Tabla 3: Flujo de caja

Mes	Ingresos	Egresos	Total	Acumulado	Valor Actual	VAN
0	$ -	-$ 113.240,97	-$ 113.240,97	-$ 113.240,97	-$ 113.240,97	-$ 113.240,97
1	$ -	-$ 201.031,27	-$ 201.031,27	-$ 314.272,24	-$ 186.283,80	-$ 299.524,77
2	$ -	-$ 203.478,94	-$ 203.478,94	-$ 517.751,18	-$ 174.719,92	-$ 474.244,69

[5] Información brindada por el Profesor Luis Haro, Vice Director de la E.N.E.T. N° 3 Gral. Martín M. de Güemes, Salta Capital
[6] Las 24 horas se obtienen multiplicando las 3 horas semanales por las 32 semanas (8 meses) del ciclo lectivo.
[7] Resultado de dividir el monto total de $ 1.218.000 obtenido en el análisis de la sección 4.3.5.1 por los 12 meses de un año.

3	$ -	-$ 267.703,60	-$ 267.703,60	-$ 785.454,78	-$ 213.004,43	-$ 687.249,13
4	$ -	-$ 208.791,27	-$ 208.791,27	-$ 994.246,05	-$ 153.942,40	-$ 841.191,52
5	$ -	-$ 217.928,94	-$ 217.928,94	-$ 1.212.174,99	-$ 148.584,87	-$ 990.083,84
6	$ 160.323,50	$ -	$ 160.323,50	-$ 1.051.851,49	$ 101.500,00	-$ 888.583,84
7	$ 173.015,77	$ -	$ 173.015,77	-$ 878.835,72	$ 101.500,00	-$ 787.083,84
8	$ 186.712,86	$ -	$ 186.712,86	-$ 692.122,86	$ 101.500,00	-$ 685.583,84
9	$ 201.494,29	$ -	$ 201.494,29	-$ 490.628,57	$ 101.500,00	-$ 584.038,84
10	$ 217.445,92	$ -	$ 217.445,92	-$ 273.182,65	$ 101.500,00	-$ 482.583,84
11	$ 234.660,39	$ -	$ 234.660,39	-$ 38.522,26	$ 101.500,00	-$ 381.083,84
12	$ 253.237,67	$ -	$ 253.237,67	$ 214.715,41	$ 101.500,00	-$ 279.583,84
13	$ 273.285,65	$ -	$ 273.285,65	$ 488.001,06	$ 101.500,00	-$ 178.083,84
14	$ 294.920,77	$ -	$ 294.920,77	$ 782.921,83	$ 101.500,00	-$ 76.583,84
15	$ 318.286,66	$ -	$ 318.286,66	$ 1.101.190,49	$ 101.500,00	**$ 24.916,16**
16	$ 343.464,93	$ -	$ 343.464,93	$ 1.444.655,42	$ 101.500,00	$ 126.416,16
17	$ 370.655,90	$ -	$ 370.655,90	$ 1.815.311,33	$ 101.500,00	$ 227.916,16

Al definir una tasa nominal anual (TNA) del 95% junto a una tasa efectiva mensual (TEM) del 7,92% y un ingreso mensual de $ 101.500 (ciento un mil quinientos pesos argentinos), el valor actual neto (VAN) cambia de negativo a positivo en el mes 15. Esto indica que, en menos de dos años de haber iniciado el desarrollo del proyecto, el mismo comienza a ser rentable ya que al obtener un VAN positivo, el valor generado supera a los costos totales.

4.3.6. Análisis de riesgo

Para el análisis de los riesgos del proyecto se confeccionó una matriz, donde se muestra por cada uno, su tipo, la fuente y las consecuencias, el impacto en el desarrollo del proyecto y la probabilidad de ocurrencia, medidas en B (baja), M (media), A(alta). También se definen las posibles respuestas en caso de que el riesgo ocurra y quien es el responsable a cargo de la acción de dicha respuesta (Ilustración 12).

La evaluación del riesgo se define en base al impacto y la probabilidad del mismo. Se utiliza el comienzo de la serie de Fibonacci (1, 2, 3, 5, 8, 13) para estimar el valor obtenido, siendo 1 el más bajo y 13 el riesgo más alto en este caso.

Los riesgos del N° 5 al 8 son inherentes al desarrollo de la solución, pero debido a su importancia e impacto en la implementación de la misma y, teniendo en cuenta que no pueden ser eliminados de manera previa al inicio del proyecto y que se disuadirán durante el transcurso del mismo, se optó por incluirlos como riesgos del proyecto.

Matriz de riesgos

N° de riesgo	Tipo de riesgo	Riesgo		Impacto	Probabilidad	Evaluación		Respuesta	Responsable de la acción de respuesta
		Fuente	Consecuencia			Valor	Nivel		
1	Externo	Devaluacion de la moneda nacional	Costos estimados del proyecto incorrectos	M	A	8	Alto	Se deberá solicitar un contrato con los proveedores para congelar precios que se ajusten al costo estimado	Project Manager
2	Externo	Falta de personal capacitado	Sobrecargo de actividades sobre un miembro del equipo	A	B	5	Medio	Se tendrá en cuenta a otros profesionales del rubro como posible reemplazo	Project Manager
3	Externo	Fallo de fábrica en las computadoras adquiridas	Retraso en las tareas de prueba y codificación	M	B	2	Bajo	Hacer valer la garantia correspondiente del producto	Project Manager
4	Interno	Falta de interes en el proyecto por parte del equipo de trabajo	Malos resultados en las tareas y retrasos en el desarrollo	B	M	3	Medio	Motivación del personal	Project Manager
5	Técnico	Palabras reservadas y simbolos mal definidos	Errores al programar las expresiones regulares	M	B	2	Bajo	Revisión de la lista de tokens	Analista programador
6	Técnico	Gramática libre del contexto mal implementada	Errores e ineficiencia en el código del analizador sintáctico	M	B	2	Bajo	Revisión de la gramática del lenguaje	Analista programador
7	Técnico	Fallos en la implementación del compilador	Errores al generar el codigo objetivo	A	M	8	Alto	Revisión del código fuente de los analizadores léxico y sintáctico	Programador
8	Técnico	Mala implementación de la GUI	Errores e ineficiencia en el uso del IDE	A	B	5	Medio	Revisión de la documentación de la herramienta para la creación de GUI	Programador

Ilustración 12: Matriz de riesgos

4.3.7. Factibilidad

En el estudio de factibilidad se analizarán cuatro factores para determinar si el proyecto puede llevarse a cabo sin ninguna restricción o impedimento que afecten o imposibiliten su desarrollo.

4.3.7.1. Factibilidad Técnica

El proyecto es factible técnicamente, ya que no existen restricciones que impidan su desarrollo. Se cuenta con todo el hardware y software necesario para su implementación, además existe una gran demanda laboral del tipo de profesionales que se requieren para conformar el equipo de trabajo, lo cual facilita el reclutamiento de los recursos humanos para el desarrollo del proyecto.

4.3.7.2. Factibilidad Operativa

Este proyecto es factible operacionalmente, debido al creciente uso de la plataforma Arduino. Esto trae consigo nuevos usuarios interesados en aprender robótica, electrónica o programación, lo cual favorece al desarrollo del proyecto, ya que el IDE planteado es una alternativa a las soluciones actuales del mercado.

4.3.7.3. Factibilidad Legal

En el marco legal, el software propuesto ni infringe ninguna norma, ley o decreto existente en la provincia y el país. Además, cumple con las licencias de uso de los programas utilizados para su desarrollo. Por lo tanto, el proyecto es factible legalmente.

4.3.7.4. Factibilidad Ambiental

Respecto al marco ambiental, el desarrollo y la implementación del proyecto no perjudican ni afectan en ningún aspecto al medio ambiente y su entorno.

Teniendo en cuenta el análisis de los cuatros factores principales, el desarrollo de este proyecto es factible en su totalidad.

5. Desarrollo de la solución

5.1. Funciones para sensores y actuadores

Durante la primera etapa del desarrollo de la solución propuesta se realizaron pruebas y ejercicios prácticos mediante el uso de los sensores y actuadores definidos en el alcance. El objetivo de estas prácticas era familiarizarse con los mismos para así obtener una visión más amplia de su funcionamiento en Arduino y, de esta manera, definir funciones y métodos que faciliten el uso y la programación de componentes avanzados. Los métodos que se desarrollaron se describen a continuación.

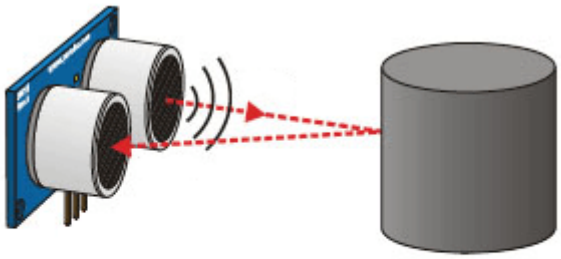
5.1.1. Método para calcular la distancia (sensor ultrasónico)

El objetivo de este método es obtener la distancia en centímetros (cm) que existe entre el sensor ultrasónico y un objeto situado a la misma distancia.

Un sensor ultrasónico mide, en microsegundos (µs), el tiempo que tarda en enviar y recibir un pulso sonoro. La velocidad del sonido es de 343 m/s (metros por segundo), por lo tanto, tenemos que

$$343\frac{m}{s} \times 100\frac{cm}{m} \times \frac{1}{1000000}\frac{s}{\mu s} = \frac{1}{29{,}2}\frac{cm}{\mu s}$$

el sonido demora 29,2 microsegundos en recorrer un centímetro. El tiempo es igual a la distancia recorrida sobre la velocidad, en el caso del sensor ultrasónico se debe multiplicar por 2 ya que el sonido emitido por el mismo realiza un trayecto de ida y vuelta (Ilustración 13).



Tiempo = 2 * (Distancia / Velocidad)
Distancia = Tiempo · Velocidad / 2

Ilustración 13: Funcionamiento del sensor ultrasónico
[Tomado de Luis Llamas, 2015]

Por lo tanto, la fórmula para obtener la distancia en centímetros mediante el uso de un sensor ultrasónico es

$$Distancia\ [cm] = \frac{Tiempo\ [\mu s]}{29{,}2\ \times 2}$$

De esta manera se logra reducir el código necesario para medir distancias al implementar la siguiente función en C++

```
int DISTANCE(int arr[])
{
    digitalWrite(arr[0], LOW);
    delayMicroseconds(5);
    digitalWrite(arr[0], HIGH);
    delayMicroseconds(10);
    digitalWrite(arr[0], LOW);
    return (pulseIn(arr[1], HIGH) / 58.4);
}
```

donde el vector *arr* almacena los pines correspondientes al *TRIGGER* y *ECHO* del sensor, respectivamente. La primera línea de la función tiene como objetivo asegurar que el *TRIGGER* del sensor se encuentre apagado, luego se espera 5 µs y se envía un pulso de 10 µs. La función retorna la distancia calculada con el tiempo que demoro el sensor en recibir el pulso sonoro sobre el tiempo que demora el sonido en recorrer un centímetro multiplicado por 2.

5.1.2. Función de sondeo (servomotor)

Los servomotores son utilizados generalmente en robots simples como eje para un sensor ultrasónico, el cual permite realizar giros en 180° con el fin de medir la distancia que existe en ambos laterales del robot, como se muestra en la Ilustración 14.

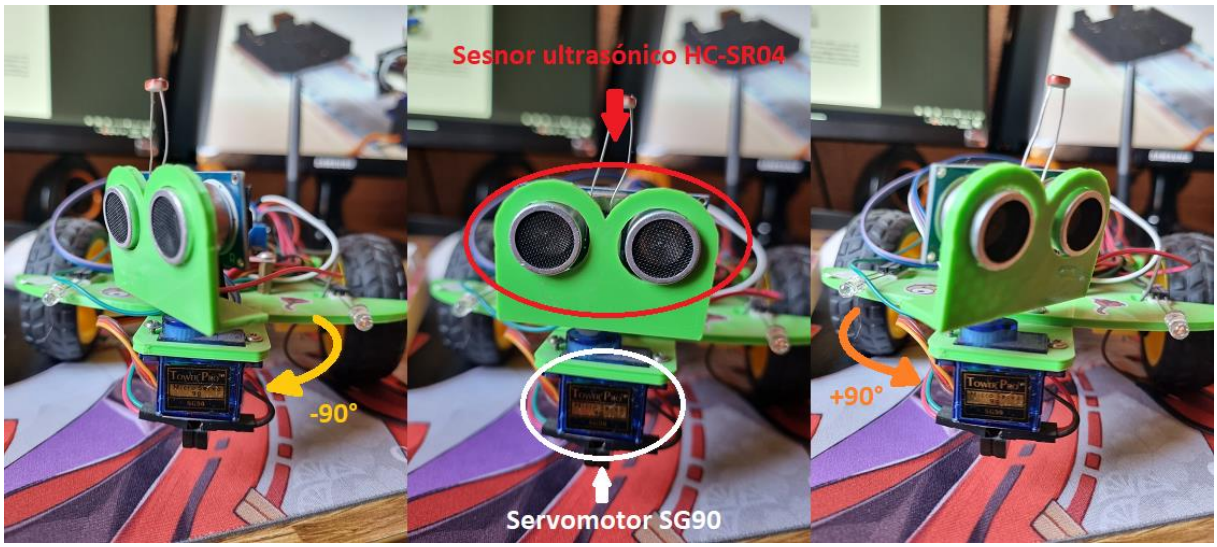


Ilustración 14: Robot Arduino con sensor ultrasónico acoplado a un servomotor [Autor de fotografía Macoritto Torcivia, 2022]

La función de sondeo implementada para servomotores simula el comportamiento de un radar, realizando un barrido desde un ángulo inicial hasta un ángulo final con una velocidad de giro especifica. Esto permite a los robots obtener una "visión" más amplia del entorno en el que se encuentran.

Esta función requiere tres parámetros (ángulo mínimo, ángulo máximo y velocidad de giro) que son utilizados en un ciclo *for*, el cual inicia en el ángulo mínimo y va incrementando su valor con un *delay* equivalente a la velocidad de giro hasta llegar al ángulo máximo. Mientras menor sea la velocidad de giro más suave será el efecto del sondeo. A continuación, se muestra el código en C++ de la función

```
void SOUNDING(Servo ser, int min, int max, int vel, bool soft)
{
  for (int i = min; i <= max; i++) {
    ser.write(i);
    delay(vel);
  }
  if (soft) {
    for (int i = max; i >= min; i--) {
      ser.write(i);
      delay(vel);
    }
  }
}
```

donde la variable *ser* representa al servomotor y el valor booleano *soft* es una variable que se utiliza para realizar un giro suave de vuelta al ángulo mínimo del radar, es decir, a la misma velocidad de giro con la que realizo el barrido inicial. En el caso que se envíe con un valor *false*, dicho giro se realiza de manera abrupta ya que el servomotor cambia de la posición máxima a la mínima sin ningún *delay*.

5.2. Tokens y palabras reservadas

5.2.1. Tokens de Pini

El archivo de configuración de pines (Pini) solo utiliza el símbolo "**,**" (coma) en su sintaxis, ya que la misma, para definir la configuración de los sensores y actuadores, no requiere complejidad y solo es necesario un símbolo que denote una separación visual en los *tokens*.

Las palabras reservadas que se utilizaron en la definición de *tokens* para Pini son:

1. **PIN**: cumple la función de #define en C++ para declarar pines
2. **OUT**: indica que un pin es de salida, equivalente a *OUTPUT* en C++
3. **INP**: indica que un pin es de entrada, equivalente a *INPUT* en C++
4. **INP_PULL**: equivalente a *INPUT_PULLUP* en C++
5. **ULTRASONIC**: declara una variable correspondiente a un sensor ultrasónico
6. **JOYSTICK**: declara una variable correspondiente a un módulo joystick
7. **SERVO**: declara una variable correspondiente a un servomotor
8. **DISPLAY**: declara una variable correspondiente a un módulo display de 4 dígitos
9. **MATRIX**: declara una variable correspondiente a una matriz LED de 8x8
10. **INTENSITY**: se utiliza para definir la intensidad lumínica en una matriz LED 8x8 o un módulo display de 4 dígitos

Los tipos de datos exclusivos para el archivo Pini corresponden a la numeración de pines disponibles en la placa Arduino UNO, es decir, de 2 a 13 para pines digitales y de A0 a A5 para pines analógicos. Ambos fueron definidos mediante las siguientes expresiones regulares:

- **[2-9] | [1] [0-3]** → expresión regular para pines digitales entre 2 y 13
- **[A] [0-5]** → expresión regular para pines analógicos entre A0 y A5

5.2.2. Tokens de uso común

Tanto el analizador sintáctico de Pini como el de Rosy utilizan los mismos *tokens* para definir números enteros e identificadores:

- **[a-zA-Z_] [a-zA-Z_0-9]*** → expresión regular para identificadores
- **[0-9]+** → expresión regular para números enteros

Los identificadores o ID, deben comenzar con un carácter alfabético y luego pueden continuar con ningún o varios caracteres alfanuméricos.

También se implementó una función para omitir comentarios en el código de Pini y Rosy. Dichos comentarios deben iniciar con **//** en caso de ser de una sola línea y, los comentarios de múltiples líneas se inician con **/*** y se finalizan con ***/**. Estos últimos también pueden ser utilizados en una sola línea de código. La expresión regular que admite ambos comentarios es la siguiente:

(/* ([^*] | [\r\n] | (*+ ([^*/] | [\r\n]))) **+/) | (//. *)

5.2.3. Tokens de Rosy

La sintaxis de Rosy cuenta con los siguientes símbolos:

- **,** → coma
- **.** → punto
- **(** → paréntesis izquierdo
- **)** → paréntesis derecho

La coma, al igual que en Pini se utiliza para separar *tokens* y palabras reservadas aportando una separación visual de los mismos. El punto, en cambio cumple la función de asociar un nombre de variable con un método o función correspondiente. Por ejemplo:

```
ultra.DISTANCE
```

esta línea de código indica que la variable “ultra”, correspondiente a un sensor ultrasónico, previamente configurado, ejecute el método *DISTANCE*, el cual devuelve la distancia en cm entre el sensor ultrasónico y un objeto situado a la misma altura.

Los paréntesis izquierdo y derecho se utilizan para agrupar expresiones matemáticas o condiciones lógicas y permiten realizar operaciones o cálculos complejos.

Además de los cuatro símbolos mencionados anteriormente, Rosy implementa los siguientes símbolos que para representar operadores matemáticos:

- = → asignación
- + → suma

- **-** → resta
- **/** → división
- ***** → multiplicación
- **^** → potencia
- **%** → módulo
- **>** → mayor
- **<** → menor
- **>=** → mayor o igual
- **<=** → menor o igual
- **<>** → distinto
- **==** → igual

Rosy, a diferencia de Pini, cuenta con más identificadores para tipos de datos, tales como números decimales, binarios y cadenas de caracteres. A continuación, se detallan las expresiones regulares de los mismos:

- **([0-9]+ [.]) [0-9]+** → expresión regular para números decimales
- **B[01][01][01][01][01][01][01][01]** → expresión regular para números binarios de 8 dígitos
- **[\"][^"\n]* [\"]|[\'][^\'\n]* [\']** → expresión regular para cadenas de caracteres incluidas entre comillas dobles o comillas simples

5.2.3.1. Palabras reservadas

La sintaxis de Rosy posee sesenta y ocho palabras reservadas que consiguen en conjunto una gran versatilidad para definir y establecer reglas gramaticales que faciliten y simplifiquen la escritura de código en Arduino. Estas se agrupan en seis categorías, según la función que cumplen en el lenguaje.

1. Generales

- DELAY
- PRINT
- GLOBAL
- CONST
- TRUE
- FALSE
- AND
- OR
- NOT
- FUNCTION
- METHOD
- RETURN
- THEN
- END
- CALL
- WITH
- TO
- IN
- RANGE
- FROM

2. Tipos de datos

- STRING
- INT
- LONG
- BYTE

- CHAR
- FLOAT
- BOOLEAN

3. Bucles o ciclos

- IF
- ELSE
- ELIF
- WHILE
- FOR
- DO
- DECREMENT

4. Funciones matemáticas

- ABS
- MAX
- MIN
- SQRT
- MAP
- COS
- SIN
- TAN
- RANDOM

5. Funciones de Arduino

- HIGH
- LOW
- READ
- WRITE
- NO
- TONE
- PULSE

6. Funciones propias

- DISTANCE
- X_AXIS
- Y_AXIS
- PUSH
- ANGLE
- ROTATE
- SOUNDING
- SCOREBOARD
- NUMBER
- DOT
- COLUMN
- ROW
- TEXT
- CLEAR
- SOFT
- PLUS
- TIMES
- ALL

5.3. Gramática del lenguaje

5.3.1. Reglas gramaticales de Pini

A continuación, se detallan las reglas gramaticales utilizadas en el analizador sintáctico del archivo de configuración de pines.

1) Inicio

La regla start da inicio a la gramática del lenguaje de Pini. En ella se invocan a las reglas de cada componente definido para la configuración de pines. Es recursiva y permite la función vacía o ε (épsilon).

```
start ::= pin_digital start
        | pin_analog start
        | ultrasonic start
        | joystick start
        | servo start
        | display start
        | matrix start
        |
```

2) Pin digital

Asigna a una variable de un tipo de entrada/salida (definidas en la función io) específica, el valor de un pin digital entre 2 y 13. Esta regla reduce en una línea el proceso de configurar pines analógicos.

```
pin_digital ::= 'PIN' 'digital' ',' 'id' ',' io
```

3) Entrada/salida

Define el tipo de entrada/salida de un pin digital.

```
io ::= 'OUT'
     | 'INP'
     | 'INP_PULL'
```

4) Pin analógico

Asigna a una variable el valor de un pin analógico entre A0 y A5.

```
pin_analog ::= 'PIN' 'analog' ',' 'id'
```

5) Ultrasónico

Define un sensor ultrasónico en una variable. Necesita dos valores de pines digitales correspondientes al *TRIGGER* y *ECHO* del sensor respectivamente.

```
ultrasonic ::= 'ULTRASONIC' 'digital' ',' 'digital' ',' 'id'
```

6) Joystick

Define un módulo joystick en una variable. Los dos primeros valores analógicos corresponden a los ejes X e Y del *stick* analógico del módulo y el valor digital al botón del *stick* analógico.

```
joystick ::= 'JOYSTICK' 'analog' ',' 'analog' ',' 'digital' ',' 'id'
```

7) Servomotor

Define un servomotor en una variable. El valor del pin debe corresponder a un pin digital pwm.

```
servo ::= 'SERVO' 'digital' ',' 'id'
```

8) Display 4 dígitos

Define un módulo display de 4 dígitos en una variable. Los valores digitales corresponden al *CLK* y *DIO* del módulo respectivamente. Se puede configurar la intensidad lumínica del mismo mediante la regla intensity display.

```
display ::= 'DISPLAY' 'digital' ',' 'digital' ',' 'id'
intensity_display
```

9) Intensidad del display

Esta regla se utiliza para regular la intensidad lumínica de un módulo display de 4 dígitos. Permite la función vacía en caso que no se requiera configurar la misma.

```
intensity_display ::= ',' 'INTENSITY' number
                    |
```

10) Matriz LED 8x8

Define una matriz LED de 8x8 en una variable. Los tres valores digitales corresponden a las entradas *DIN*, *CLK* y *CS* de la matriz respectivamente, mientras que el valor numérico inicial hace referencia a la cantidad de matrices conectadas a la placa Arduino. Se puede configurar la intensidad lumínica de la misma mediante la regla intensity matrix.

```
matrix ::= 'MATRIX' number ',' 'digital' ',' 'digital' ',' 'digital'
',' 'id' intensity_matrix
```

11) Intensidad de la matriz

Esta regla se utiliza para regular la intensidad lumínica de una matriz LED 8x8. Permite la función vacía en caso que no se requiera configurar la misma. Gramaticalmente es igual que la regla para configurar la intensidad de un display de 4 dígitos, pero lógicamente se comportan de manera distinta ya que al generar el código fuente para estos componentes la configuración de la intensidad se realiza de manera diferente.

```
intensity_matrix ::= ',' 'INTENSITY' number
                   |
```

12) Número

Esta regla se utiliza como terminal general para números enteros y digitales.

```
number ::= 'digital'
         | 'entero'
```

5.3.2. Reglas gramaticales de Rosy

El lenguaje de Rosy, al ser más versátil y amplio que el lenguaje de Pini presenta una mayor cantidad de reglas (65 en total), y a su vez, más complejas. Para simplificar la descripción de las mismas, se agruparon en siete categorías, las cuales representan una funcionalidad particular del lenguaje.

I. Generales

Las reglas generales se utilizan en la estructura principal del lenguaje y conforman la misma. Se debe respetar dicha estructura para el correcto funcionamiento de Rosy, además, ésta mantiene un orden en el programa escrito, el cual facilita la lectura del mismo. La estructura principal está compuesta por las **variables globales**, luego siguen las **funciones y métodos** y por último la **lógica del programa**.

1) Inicio

La regla start da inicio a la gramática del lenguaje de Rosy. Es la que define la estructura principal del programa donde *globals* es la regla para variables globales, *functions_methods* la regla de funciones y métodos y *body* la regla que define la lógica del programa. Si alguna de etas tres reglas se escriben en otro orden el depurador marcara un error de sintaxis.

```
start ::= globals functions_methods body
```

2) Variables globales

Esta regla define las variables globales. Es recursiva, lo cual permite declarar una o más variables globales. También acepta la función vacía en caso de no declarar ninguna variable global.

```
globals ::= global_variable globals
          |
```

3) Funciones y métodos

Se utiliza para declarar funciones y métodos en el programa para luego ser utilizados en el cuerpo del mismo. Esta regla permite declarar uno o varios metidos o variables mediante la recursividad o ninguno a través de la función vacía.

```
functions_methods ::= function functions_methods
                    | method functions_methods
                    |
```

4) Función

Define una función con o sin parámetros.

```
function ::= 'FUNCTION' 'id' ',' fm_parameters[8] 'THEN' body 'END'
'FUNCTION'
```

[8] Detallada en la **regla 4** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**

5) Método

Define un método con o sin parámetros. Se debe especificar el tipo de dato que retorna y, a diferencia de las funciones los métodos tienen que retornar un valor, una variable o una expresión matemática.

```
method ::= 'METHOD' 'id' ',' fm_parameters 'RETURN' data_type[9] ','
'THEN' body 'RETURN' alguito[10] 'END' 'METHOD'
```

6) Cuerpo del programa

La regla *body* o cuerpo del programa se utiliza tanto para la lógica del mismo como para las funciones y métodos definidos. En ella se implementan todas las funciones (matemáticas, de Arduino o propias del lenguaje), ciclos y asignaciones de variables permitidas en Rosy que, en conjunto, conforman la lógica del programa.

```
body ::= digital_write body
        | analog_write body
        | delay body
        | tone body
        | no_tone body
        | rotate body
        | sounding body
        | scoreboard body
        | number_dspy body
        | clear_dspy body
        | dot body
        | row body
        | column body
        | text_matrix body
        | clear_matrix body
        | asignacion body
        | asignacion_variable body
        | const_variable body
        | call_function_method body
        | print body
        | for body
        | while body
        | do_while body
        | if body
        |
```

7) Asignación

Asigna un valor, expresión o resultado de una función a una variable previamente declarada.

```
asignacion ::= 'id' '=' algo[11]
```

[9] Detallada en la **regla 1** de la sección VII. Variables y tipos de datos del capítulo **5. Desarrollo de la solución**
[10] Detallada en la **regla 2** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**
[11] Detallada en la **regla 1** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**

8) Asignación de variable

Define el tipo de dato de una variable para su posterior asignación.

```
asignacion_variable ::= data_type asignacion
```

9) Variable constante

Define una variable constante.

```
const_variable ::= 'CONST' asignacion_variable
```

10) Variable global

Define una variable de alcance global, ya sea de constante o no.

```
global_variable ::= 'GLOBAL' asignacion_variable
                  | 'GLOBAL' const_variable
```

11) Llamada a una función/método

Esta regla se utiliza dentro del cuerpo del programa para llamar a funciones y/o métodos definidos previamente.

```
call_function_method ::= 'CALL' 'id' cfm_arguments[12]
```

12) Print

La regla *print* permite mostrar en pantalla un valor, variable, expresión matemática o secuencia de caracteres. Es posible concatenar diferentes valores mediante el uso de la regla conctenate print.

```
asignacion_variable ::= 'PRINT' alguito concatenate_print
```

13) Concatenar print

Esta regla se utiliza para concatenar uno o varios valores, mediante recursividad, a la regla *print*. Admite la función vacía en caso que no se sea necesario concatenar otro valor.

```
concatenate_print ::= ',' alguito concatenate_print
                    |
```

[12] Detallada en la **regla 5** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**

II. Ciclos condicionales

El lenguaje de Rosy cuenta con reglas para los siguientes ciclos condicionales.

1) For

La regla para el ciclo *for* permite ejecutar n cantidad de veces el código contenido en el *body*. El *token* id representa a la variable a incrementar, el primer número entero será el valor inicial que tomara la variable y el segundo hace referencia al valor máximo que llegará la misma. Por lo tanto, el ciclo se ejecutará una cantidad de veces equivalente a la diferencia entre el segundo entero y el primero. Por ejemplo: para ejecutar un ciclo 5 veces, el primer valor debe ser 0 y el segundo 5. Esta regla permite realizar un decremento, envés de un incremento, mediante el uso de la regla decrement[13].

```
for ::= 'FOR' 'id' 'IN' 'RANGE' 'entero' ',' 'entero' ',' decrement
'THEN' body 'END' 'FOR'
```

2) While

La regla para el ciclo *while* permite ejecutar de manera continua el código contenido en el *body* mientras se cumpla la condición lógica de la regla logic[14].

```
while ::= 'WHILE' logic ',' 'THEN' body 'END' 'WHILE'
```

3) Do while

Esta regla funciona de manera similar a la del ciclo while, con la diferencia de que siempre ejecuta una vez el código contenido en el *body* y luego valida la condición lógica de la regla logic.

```
do_while ::= 'DO' 'WHILE' logic ',' 'THEN' body 'END' 'DO' 'WHILE'
```

4) If

La regla para la sentencia *if* valida la condición lógica de la regla logic y, en caso de ser verdadera, ejecuta el código contenido en el *body*. Puede estar precedida por ninguna o varias reglas else if y solo una o ninguna regla else.

```
if ::= 'IF' logic ',' 'THEN' body 'END' 'IF' elseif else
```

5) Else if

Esta regla permite agrupar múltiples condiciones lógicas luego de haber definido una regla *if*, lo que aporta un mayor control sobre el flujo del código.

```
elseif ::= 'ELIF' logic ',' 'THEN' body 'END' 'ELIF' elseif
         |
```

[13] Detallada en la **regla 6** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**
[14] Detallada en la **regla 7** de la sección VI. Reglas auxiliares del capítulo **5. Desarrollo de la solución**

6) Else

La regla de la cláusula *else* se utiliza después de haber definido una cláusula if o else if con el objetivo de implementar un código para aquellas condiciones lógicas que fueron validadas negativamente por las sentencias anteriores.

```
else ::= 'ELSE' body 'END' 'ELSE'
       |
```

III. Funciones exclusivas de Arduino

Este conjunto de reglas agrupa las funciones exclusivas del lenguaje de Arduino, que permiten realizar tanto lecturas y escrituras en pines analógicos y digitales como medir el tiempo que tarda un pin analógico en cambiar su valor.

1) Read

La regla *read* permite realizar la lectura analógica o digital de un pin definido en el archivo de configuración de pines.

```
read ::= 'id' '.' 'READ'
```

2) Digital write

Esta regla es similar a la función digitalWrite del lenguaje de Arduino, donde se indica un pin analógico y el valor que se desea escribir. Los valores permitidos están definidos por las palabras reservadas HIGH y LOW, que hacen referencia a 1 y 0 respectivamente.

```
digital_write ::= 'id' '.' value[15]
```

3) Analog write

La regla de escritura analógica es similar a la de escritura digital, con la diferencia de que acepta como argumento una expresión matemática, valor numérico o variable. Solo puede ser utilizada en pines analógicos y es similar a la función analogWrite de Arduino.

```
analog_write ::= 'id' '.' 'WRITE' expression
```

4) Delay

Esta regla se utiliza para detener la ejecución del programa actual durante los segundos que se indiquen.

```
delay ::= 'DELAY' number[16]
```

[15] Detallada en la **regla 2** de la sección VII. Variables y tipos de datos del capítulo **5. Desarrollo de la solución**
[16] Detallada en la **regla 3** de la sección VII. Variables y tipos de datos del capítulo **5. Desarrollo de la solución**

5) Tone

Equivalente a la función tone de Arduino. Esta regla permite ingresar un segundo argumento numérico opcional que representa el tiempo en segundos del tono que emite el pin.

```
tone ::= 'id' '.' 'TONE' alguito
       | 'id' '.' 'TONE' alguito ',' number
```

6) No tone

Esta regla detiene la emisión del tono que ejecuta un pin mediante el uso de la regla anterior, en caso de no haber definido un tiempo límite.

```
no_tone ::= 'id' '.' 'NO' 'TONE'
```

7) Pulse

La regla pulse devuelve el tiempo que tarda un pin digital en cambiar su valor, acepta un valor numérico opcional que indica el tiempo de espera máximo en caso que no cambie de valor el pin.

```
pulse ::= 'id' '.' 'PULSE' value
        | 'id' '.' 'PULSE' value ',' number
```

IV. Funciones propias de Rosy

Las reglas para las funciones propias de Rosy fueron diseñadas con el objetivo de facilitar la implementación y el uso de los distintos sensores y actuadores definidos en el alcance del proyecto.

1) Distancia

Obtiene la distancia (en centímetros) que existe entre el sensor ultrasónico y un objeto situado a la misma altura.

```
distance ::= 'id' '.' 'DISTANCE'
```

2) Eje x

Devuelve la posición en el eje x del stick analógico de un módulo joystick.

```
x_axis ::= 'id' '.' 'X_AXIS'
```

3) Eje y

Devuelve la posición en el eje y del stick analógico de un módulo joystick.

```
y_axis ::= 'id' '.' 'Y_AXIS'
```

4) Push

Detecta cuando se presiona el stick analógico de un módulo joystick.

```
push ::= 'id' '.' 'PUSH'
```

5) Ángulo

Obtiene el ángulo en el que se encuentra posicionado el eje de un servomotor.

```
angle ::= 'id' '.' 'ANGLE'
```

6) Rotar

Esta regla se utiliza para rotar el eje de un servomotor al ángulo indicado mediante el calor entero, el cual debe ser entre 0 y 180.

```
rotate ::= 'id' '.' 'ROTATE' 'entero'
```

7) Sondeo

La función de sondeo fue descrita previamente en la sección 5.1.2. Requiere como parámetros el ángulo mínimo, el ángulo máximo y la velocidad de giro respectivamente. Además, permite agregar de manera opcional la palabra reservada SOFT, la cual indica que el giro de vuelta al ángulo mínimo se realizara con la misma velocidad de giro definida.

```
sounding ::= 'id' '.' 'SOUNDING' 'entero' ',' 'entero' ',' 'entero'
           | 'id' '.' 'SOUNDING' 'entero' ',' 'entero' ',' 'entero'
     ',' 'SOFT'
```

8) Tablero

Esta regla se utiliza en el módulo display 4 dígitos como tablero de puntajes. Si uno de los números enteros ingresados es mayor a 9 se deberá especificar el atributo PLUS, el cual permite ingresar valores enteros de 0 hasta 99.

```
scoreboard ::= 'id' '.' 'SCOREBOARD' 'entero' ',' 'entero'
             | 'id' '.' 'SCOREBOARD' 'PLUS' 'entero' ',' 'entero'
```

9) Número display

Muestra un número entero en el módulo display 4 dígitos. El número debe estar comprendido entre 0 y 9999.

```
number_dspy ::= 'id' '.' 'NUMBER' 'entreo'
```

10) Limpiar display

Esta regla se utiliza para limpiar los números existentes en el módulo display de 4 dígitos, no requiere parámetros adicionales.

```
clear_dspy ::= 'id' '.' 'CLEAR'
```

11) Punto

Enciende un punto en una matriz LED 8x8 situado en la posición indicada mediante fila y columna, correspondientes a los dos primeros valores enteros respectivamente. También es necesario indicar el número de la matriz en la que se mostrará el punto a través del tercer valor entero ingresado.

```
dot ::= 'id' '.' 'DOT' 'entero' ',' 'entero' ',' IN 'entero'
```

12) Fila

Esta regla se utiliza en una matriz LED 8x8 para especificar los puntos de una columna que se iluminan mediante un código binario de la forma BXXXXXXXX dónde X puede ser un 0 o un 1 y éstos establecen si el led se prende (1) o se apaga (0). Es necesario indicar el número de columna que se iluminará por medio del primer valor entero y el número de la matriz en la que se mostrará el punto a través del último valor entero ingresado.

```
row ::= 'id' '.' 'ROW' 'entero' ',' 'binary' ',' IN 'entero'
```

13) Columna

Funciona de manera similar a la regla de fila descrita anteriormente, con la diferencia de que esta ilumina por columna.

```
column ::= 'id' '.' 'COLUMN' 'entero' ',' 'binary' ',' IN 'entero'
```

14) Texto matriz

Muestra un texto ingresado n cantidad de veces en una matriz LED 8x8. Las veces que se va a mostrar el texto debe ser indicada por el segundo número entero antes de la palabra reservada TIMES, mientras que el primer número entero hace referencia al número de la matriz en la que se mostrará el texto.

```
text_matrix ::= ‘id’ ‘.’ ‘TEXT’ ‘cadena’ ‘,’ ‘entero’ ‘,’ ‘entero’
‘TIMES’
```

15) Limpiar matriz

Esta regla se utiliza para limpiar todos los puntos de una matriz LED 8x8. Permite indicar el número de matriz que se desea limpiar, o bien, limpiar todas las matrices conectadas a la placa mediante el uso de la palabra reservada ALL.

```
clear_matrix ::= ‘id’ ‘.’ ‘CLEAR’ ‘entero’
               | ‘id’ ‘.’ ‘CLEAR’ ‘ALL’
```

V. Funciones matemáticas

El lenguaje de Rosy cuenta con las siguientes funciones matemáticas predeterminadas.

1) Absoluto

Calcula el valor absoluto de un número, valor o expresión matemática.

```
abs ::= ‘ABS’ expression
```

2) Máximo

Devuelve el mayor de dos números, valores o expresiones matemáticas.

```
max ::= ‘MAX’ expression ‘,’ expression
```

3) Mínimo

Devuelve el menor de dos números, valores o expresiones matemáticas.

```
min ::= ‘MIN’ expression ‘,’ expression
```

4) Raíz cuadrada

Calcula la raíz cuadrada de un número, valor o expresión matemática.

```
sqrt ::= ‘SQRT’ expression
```

5) Coseno

Calcula la función coseno de un número, valor o expresión matemática.

```
cos ::= ‘COS’ expression
```

6) Seno

Calcula la función seno de un número, valor o expresión matemática.

```
sin ::= 'SIN' expression
```

7) Tangente

Calcula la función tangente de un número, valor o expresión matemática.

```
tan ::= 'TAN' expression
```

8) Mapeo

Esta regla mapea un número de un rango a otro, es decir, el valor de la primera expresión se asignará a la tercera expresión y el valor de la segunda expresión se asignará a la cuarta expresión, los valores internos comprendidos en el rango de la primera y segunda expresión se calculan de manera similar.

```
map ::= 'MAP' expression 'FROM' expression ',' expression 'TO'
expression ',' expression
```

9) Número aleatorio

Esta regla permite generar un número pseudo-alearorio comprendido entre dos valores enteros. Si se omite el primer valor entero, el generador de números generará una secuencia entre 0 y el valor ingresado.

```
random ::= 'RANDOM' 'entero'
         | 'RANDOM' 'IN' 'RANGE' 'entero' ',' 'entero'
```

VI. Reglas auxiliares

Las reglas auxiliares se utilizan durante toda la gramática del lenguaje de Rosy para agrupar posibles valores como variables, expresiones matemáticas, constantes, valores de pines, identificadores, entre otros. También incorporan reglas para concatenar argumentos o parámetros en funciones y métodos y establecer condiciones lógicas en ciclos condicionales.

1) Regla "algo"

La regla "algo" es una de las más importantes de la gramática, ya es utilizada por regla de asignación, que permite declarar variables de todos los tipos permitidos por el lenguaje. Está compuesta por todas las funciones matemáticas que retornan un valor numérico y la regla para llamar funciones o métodos definidos.

```
algo ::= alguito
       | pulse
       | abs
       | max
       | min
       | sqrt
       | cos
       | sin
       | tan
       | map
       | random
       | call_function_method
```

2) Regla “alguito”

Esta regla forma parte de la regla “algo” y también es utilizada en gramáticas que requieren una cadena de caracteres, expresión matemática, identificadores u otra variable; como la regla print por ejemplo.

Originalmente esta regla formaba parte de la regla “algo”, pero surgió la necesidad de tener ciertas reglas como las funciones de Arduino (exclusivas y propias de Rosy) que retornan valores numéricos separadas de las reglas de funciones matemáticas y llamadas a funciones y métodos definidos. Por esta causa se desagregó la regla “algo” para conformar la regla “alguito” que internamente trabajan de la misma forma, pero hace uso de diferentes reglas.

```
alguito ::= ‘cadena’
          | read
          | distance
          | x_axis
          | y_axis
          | push
          | angle
          | boolean[17]
          | expression
```

3) Expresión matemática

La regla para expresiones matemáticas permite formar expresiones únicamente con números e identificadores de variables. Esto se debe a que al otorgar la posibilidad de agregar funciones o métodos que retornen valores la gramática del lenguaje se volvía compleja y podía generar ambigüedades. Como el objetivo del proyecto es desarrollar un IDE que facilite el uso a las personas se optó por realizar la gramática de la forma más sencilla, pero complete posible, evitando generar complejidad extra en su uso.

Mediante el uso de esta regla se pueden concatenar varias expresiones con diferentes operadores matemáticos y separarlas con paréntesis. Esto se logra a través de la recursividad.

[17] Detallada en la **regla 4** de la sección VII. Variables y tipos de datos del capítulo **5. Desarrollo de la solución**

```
expression ::= expression ‘-’ expression
             | expression ‘+’ expression
             | expression ‘/’ expression
             | expression ‘*’ expression
             | expression ‘%’ expression
             | expression ‘^’ expression
             | ‘(’ expression ‘)’
             | number
             | ‘id’
```

4) Parámetros para funciones y métodos

Estas reglas permiten agregar parámetros a los funciones y métodos declarados mediante la regla functions_methods. Mediante el uso de la recursividad es posible agregar la cantidad de parámetros que sean necesarios o ninguno a través de la función vacía. Se debe indicar el nombre del parámetro y su tipo de dato.

```
fm_parameters ::= ‘WITH’ data_type ‘id’ ‘,’ fm_parameters2
                |

fm_parameters2 ::= data_type ‘id’ ‘,’ fm_parameters2
                 |
```

5) Argumentos para funciones y métodos

Las reglas de argumentos para funciones y métodos se utilizan en la regla call_function_method con el fin de ingresar valores que representan los argumentos de las funciones y/o métodos que se invoquen durante el programa. Al igual que la regla de los parámetros, permite agregar varios o ningún argumento a la función o método.

```
cfm_arguments ::= ‘WITH’ alguito cfm_arguments2
                |

cfm_arguments2 ::= ‘,’ alguito cfm_arguments2
                 |
```

6) Decremento

Regla utilizada en un ciclo for para establecer que el mismo realice un decremento de la variable envés de un incremento.

```
decrement ::= ‘DECREMENT’ ‘,’
            |
```

7) Lógica booleana

La regla de lógica se utiliza en los ciclos condicionales para generar lógicas booleanas que retornan un valor *TRUE* o *FALSE*. De manera similar que las expresiones matemáticas es posible agrupar dichas reglas mediante el uso de paréntesis y recursividad.

```
logic ::= logic ‘AND’ logic
        | logic ‘OR’ logic
        | ‘(’ logic ‘)’
        | ‘NOT’ logic
        | condition
```

8) Condición lógica

Esta regla es utilizada por la regla de lógica booleana para generar condiciones lógicas mediante el uso de los operadores lógicos definidos en los símbolos de Rosy. Al igual que la regla de expresiones matemáticas permite agrupar las condiciones en paréntesis para lograr un mejor control y generar múltiples condiciones.

```
condition ::= val ‘>’ val
            | val ‘<’ val
            | val ‘>=’ val
            | val ‘<=’ val
            | val ‘<>’ val
            | val ‘==’ val
            | val
```

9) Valor

La regla de valor se utiliza únicamente en la regla de condiciones lógicas. Es similar a la regla “alguito”, con la diferencia de que esta no permite expresiones matemáticas, sino números e identificadores. Esto se debe a que la regla de expresiones matemáticas permite agrupar otras expresiones entre paréntesis, lo cual generaría una ambigüedad al compilador a la hora de interpretar una línea, ya que la regla de lógica booleana también permite agrupar lógicas mediante el uso de paréntesis y por ende el compilador no podría diferenciar si es un paréntesis perteneciente a la regla de expresiones matemáticas o a la regla de lógica booleana.

```
val ::= ‘cadena’
      | read
      | distance
      | x_axis
      | y_axis
      | push
      | angle
      | boolean
      | number
      | ‘id’
```

VII. Variables y tipos de datos

Las reglas para variables y tipos de datos, como su nombre indica, establecen los tipos de datos y valores que pueden tomar ciertas variables de la gramática de Rosy.

1) Tipos de datos

Regla terminal para los tipos de datos admitidos por el lenguaje de Rosy.

```
data_type ::= 'STRING'
            | 'INT'
            | 'LONG'
            | 'BYTE'
            | 'CHAR'
            | 'FLOAT'
            | 'BOOLEAN'
```

2) Valores

Regla terminal para los valores admitidos por los pines digitales.

```
value ::= 'HIGH'
        | 'LOW'
```

3) Número

La regla número solo admite números enteros y decimales.

```
number ::= 'entero'
         | 'decimal'
```

4) Booleano

Regla terminal para los valores booleanos.

```
boolean ::= 'TRUE'
          | 'FALSE'
```

5.4. Analizador léxico y sintáctico

Los analizadores léxicos y sintácticos del archivo de configuración de pines y de Rosy fueron desarrollados en el lenguaje Python mediante el uso de la librería PLY (Python Lex-Yacc). Esta librería requiere dos archivos, uno donde se declaren los *tokens* (archivo Lex) del lenguaje y otro que importe los mismos y los utilice en las reglas de la gramática definida (archivo Yacc).

Para generar los *tokens* una vez definidos, se utilizó la siguiente función de la librería de PLY en los archivos Lex que contienen los *tokens*, símbolos y palabras reservadas de Pini y Rosy descritas en la sección anterior

```
lexer = lex.lex()
```

donde se asigna el resultado de la función a una variable llamada lexer, esto permite que sea importada en los archivos Yacc de la gramática.

En los archivos que contienen las reglas gramaticales del lenguaje y las precedencias, se utiliza la siguiente función perteneciente a la librería de PLY

```
parser = yacc.yacc()
```

la cual genera un archivo parser.out y parsetab.py que contienen las tablas y arboles generados a partir de las reglas de los archivos Yacc. Se asigna el resultado de la función a una variable con el fin de compilar un código escrito en el lenguaje desarrollado mediante el uso de la función *parse* de PLY, la cual genera el código objeto del programa.

5.4.1. Diferenciación de tipos de pines

La regla "read" descrita en la sección III. Funciones exclusivas de Arduino del **capítulo 5. Desarrollo de la solución** es gramaticalmente igual tanto para pines digitales como analógicos (Ilustración 15). Esto generaría un conflicto en la traducción de la línea ya que el lenguaje de Arduino basado en C++ utiliza funciones diferentes para ambos pines.

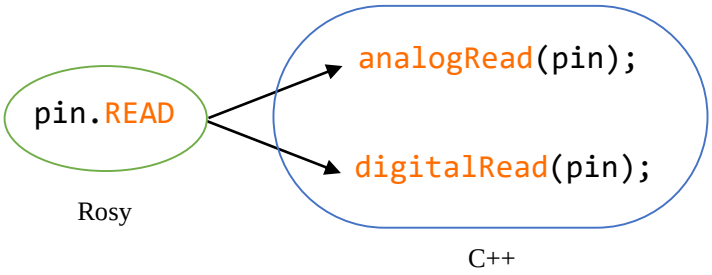


Ilustración 15: Funcionamiento de la regla read

Con el objetivo de evitar crear una regla para cada tipo de pin y mantener la simplicidad del lenguaje, se utilizaron dos vectores globales que almacenan el nombre de los pines definidos en la configuración de Pini: uno para los pines digitales y otro para los analógicos. Ambos vectores se encuentran en el archivo Yacc de Pini y son importados al archivo Yacc de Rosy cuando se compila el código. De esta manera se logra que cada vez que la regla read se ejecute, se recorran los vectores en busca del nombre de la variable y, según donde sea encontrado el nombre se utilice la función analogRead o digitalRead correspondiente al tipo de pin definido.

5.4.2. Librerías de terceros

Tres de los componentes incluidos en la configuración de pines utilizan librerías de terceros para su correcto funcionamiento en la placa Arduino. Ellos son:

1. Servomotor SG90: librería **Servo.h** (incluida en el lenguaje de Arduino)
2. Módulo display de 4 dígitos: librería **TM1637.h**
3. Matriz LED 8x8: librería **LedControlMS.h**

Para evitar el uso innecesario de estas librerías, al incluirlas de manera genérica en cada código compilado, se implementó un vector global (en el archivo Yacc de Pini) donde se van almacenando cada vez que se ejecuta una regla de configuración de pines que contenga alguno de los componentes mencionados.

Luego, mediante el uso de la función "*dict.fromkeys"* de Python se eliminan las librerías repetidas y se agregan al inicio del código fuente generado por el compilador de Pini. De esta manera solo se importan las librerías necesarias y se optimiza el código fuente.

5.4.3. Manejo de variables globales, funciones y métodos

Las variables globales en el lenguaje de Arduino se declaran al inicio del programa, mientras que los métodos al final del mismo, pero luego de la declaración de variables globales se define el ciclo *setup* para la configuración de pines y el *loop* para la lógica del programa, como se muestra en la Ilustración 16.

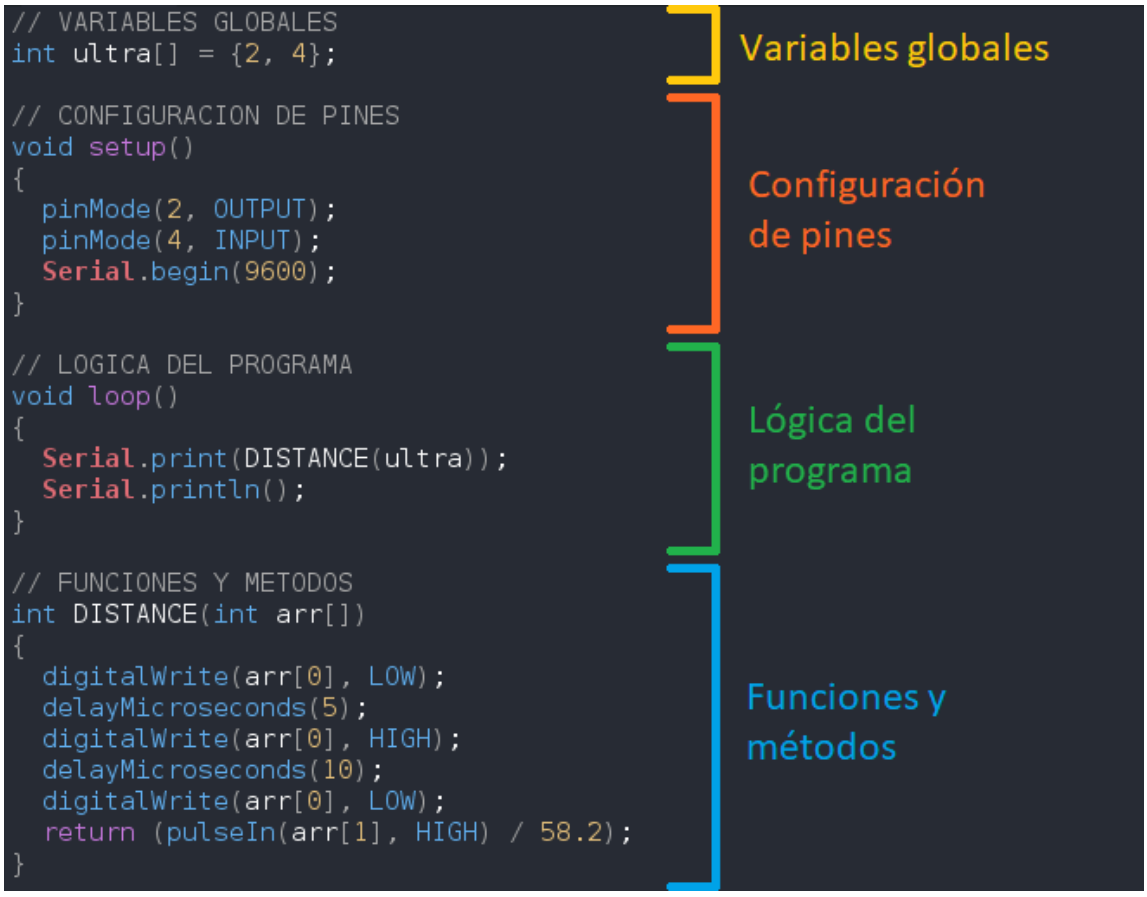


Ilustración 16: Estructura del lenguaje de Arduino basado en C++

Pini se encarga de generar el ciclo *setup*, mientras que Rosy genera el ciclo *loop* pero las variables globales se definen en Rosy, por lo tanto deberían ser incorporadas antes del ciclo *setup* generado por Pini. Para lograr esto, se utilizó una variable global que almacena todas las variables globales definidas durante la escritura del programa y luego las inserta antes del ciclo generado por el compilador de Pini.

De manera similar se trabajó con las funciones y métodos, una variable global los almacena hasta que se compila el código y son insertados después del ciclo *setup* generado por Pini, pero antes del ciclo *loop* que contiene la lógica del programa, respetando la estructura del lenguaje de Arduino.

En el caso de la función para el servomotor y el método del sensor ultrasónico, definidos en la sección 5.1, se declararon dos variables booleanas que indican si la función o método ya fue agregado al vector global de funciones y métodos. De esta manera se evita agregar más de una vez el mismo método o función al vector si es invocado en distintas líneas a lo largo del programa, generando así un código más limpio y optimizado.

5.4.4. Precedencia

El archivo Yacc de Pini no utiliza precedencias para evitar ambigüedades en la gramática ya que la misma es simple y posee pocas reglas que nunca llegarán a generar un conflicto de reducción gramatical.

En cambio, Rosy además de contar con 68 (sesenta y ocho) reglas gramaticales, la regla de expresiones matemáticas y la de lógica booleana generan conflictos de ambigüedad al hacer uso de los símbolos de paréntesis y la recursividad. Para evitar conflictos de reducción se declararon las siguientes precedencias:

a. Primera precedencia por izquierda para los símbolos **+** y **-**
b. Segunda precedencia por izquierda para los símbolos ***** y **/**
c. Tercera precedencia por izquierda para los símbolos **%** y **^**
d. Cuarta precedencia por izquierda para las palabras reservadas **AND** y **OR**
e. Quinta precedencia por izquierda para la palabra reservada **NOT**

De esta manera el compilador sabe cuál es la regla por la que debe reducir en caso de encontrar una ambigüedad gramatical.

El orden de las precedencias se declaró de esta forma para respetar el orden de los operadores matemáticos, ya que el compilador ordena los *tokens* desde la mayor precedencia hasta la menor. Por lo tanto, primero realiza la operación de lógica booleana NOT, luego AND y OR (que se encuentran en el mismo nivel de precedencia) y, por último, las operaciones matemáticas, de las cuales el módulo y la potencia tienen mayor precedencia que la multiplicación y división, por ende, las realiza primero, dejando como últimas a las operaciones se suma y resta.

5.4.5. Depurador de errores

Tanto los archivos Lex y Yacc de Pini como los de Rosy cuentan con un depurador de errores, el cual marca el tipo de error e indica la línea donde se encuentra el mismo.

Los depuradores de Lex se encargan de los errores de tipo “carácter ilegal”, este tiene origen cuando se ingresa un carácter, palabra o símbolo no definido en los *tokens* y palabras reservadas de Pini/Rosy. En la Ilustración 17, se muestra el código de la función que controla los errores en los archivos Lex, siendo la misma para el depurador de Pini y Rosy.

```
def t_error(token):
    showwarning(
        title='Advertencia',
        message=("Caracter ilegal '%s' en la línea %s"
                 % (token.value[0], token.lexer.lineno))
    )
    token.lexer.skip(1)
```

Ilustración 17: Depurador de archivos Lex de Pini y Rosy

La función muestra un dialogo de texto con una advertencia donde indica el carácter ilegal que se detectó, ubicado en el vector *token.value[0]*, y la línea donde se encuentra, almacenada en una variable de *lexer* (de la librería PLY) llamada *lineno*.

En los archivos Yacc se implementó otro depurador, el cual se encarga de controlar los errores de sintaxis y los finales de línea inesperados. Un error de sintaxis ocurre cuando el compilador detecta una línea que no coincide con ninguna de las reglas gramaticales definidas, o que fue mal escrita. Por ejemplo: una asignación de variable que no tenga el símbolo =.

Los errores de final de línea inesperados tienen lugar cuando el compilador detecto que una línea pertenece a una regla gramatical pero no llega a completarse. Por ejemplo, al escribir solo la palabra PRINT sin ningún valor a continuación se genera este tipo de error, ya que el compilador sabe que la línea debe interpretarse con la regla print pero faltan valores para que esta sea correcta.

Las funciones del depurador de archivos Yacc son similares para el compilador de Pini y Rosy, en la Ilustración 18, se muestra el código de la función de Rosy.

```
def p_error(p):
    global rosy_error
    rosy_error = True
    message = ""
    if p != None:
        message = "Error de sintaxis en la línea %s" % p.lineno
    else:
        message = "Unexpected end of input"

    showerror(title='Fallo al compilar', message=message)
```

Ilustración 18: Depurador del archivo Yacc de Rosy

La variable global de error se utiliza antes de compilar el código fuente para evitar que el programa genere algún fallo al compilar el código con errores. Si el parámetro *p* es igual a "None", es decir, un valor nulo, significa que el compilador no llego a completar la regla, por lo tanto, corresponde a un error de final de línea inesperado. Caso contrario, el error es de sintaxis y se muestra la línea donde ocurrió el mismo mediante el valor de *lineno*.

5.4.6. Proceso de compilación

El proceso de compilación requiere los códigos fuente de Pini y Rosy al mismo tiempo. Esto se debe a que al compilar el archivo de Pini se genera el vector con los nombres de las variables de los pines definidos, el cual es requerido por el compilador de Rosy, además, el compilador de Rosy también requiere el código objeto del ciclo *setup* generado por Pini para incorporarlo en el código objeto final (Ilustración 19).

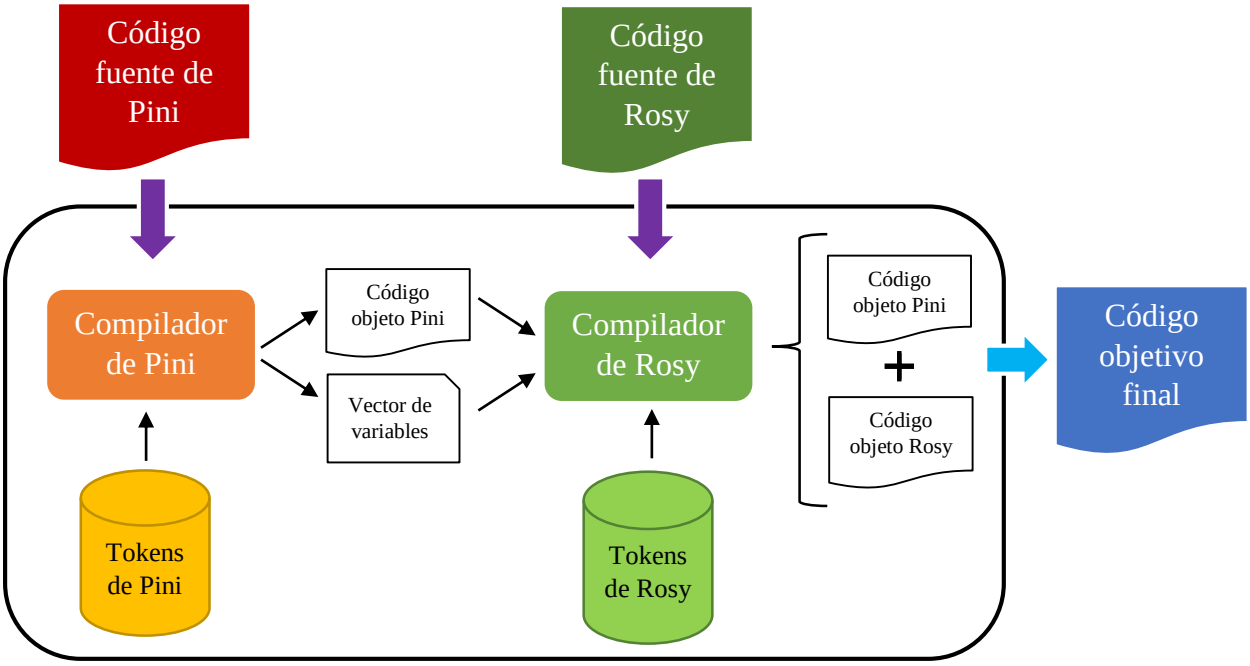


Ilustración 19: Diagrama de proceso de compilación

Durante el proceso de compilación de Pini, se agrega el código objeto resultante del mismo como cuerpo del ciclo *setup*, luego se añaden las librerías utilizadas al inicio del código objeto, pero fuera del ciclo (Ilustración 20).

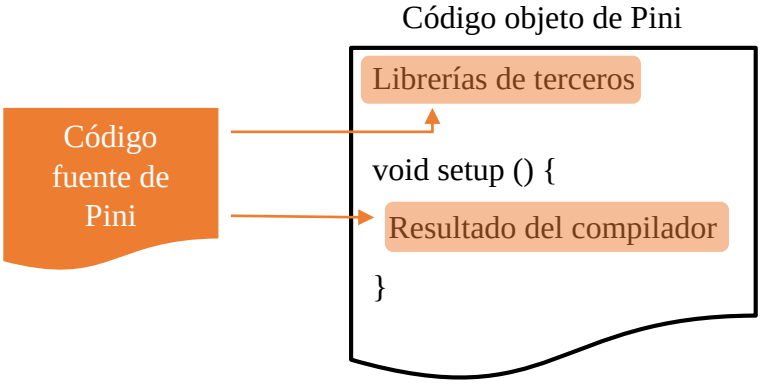


Ilustración 20: Diagrama del proceso de compilación de Pini

El compilador de Rosy toma el código objeto generado por Pini junto al vector de nombre de variables y, si no existen errores en la sintaxis en el código fuente del archivo Rosy, compila el mismo. En el proceso de compilación, inserta las variables globales (existentes en el vector de variables globales) al inicio del código objeto, luego agrega el código objeto generado por Pini y a continuación, dentro del ciclo *loop* incorpora el resultado de la compilación. Por último, añade las funciones y métodos (almacenadas en el vector de funciones y métodos al final) del código objetivo (Ilustración 21).

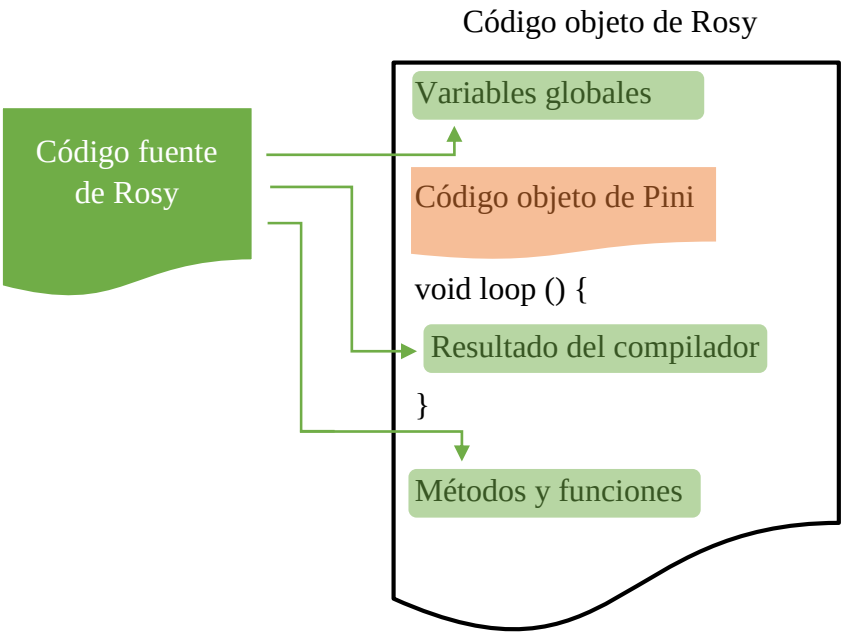


Ilustración 21: Diagrama de proceso de compilación de Rosy

Como resultado final se obtiene el código objetivo en C++ listo para ser cargado a una placa Arduino UNO a través del IDE de Arduino.

5.5. Interfaz gráfica de usuario (GUI)

La interfaz gráfica de Rosy IDE se desarrolló con tkinter, una librería grafica para Python basada en TCL. En esta sección se detallan las pantallas y funcionalidades más importantes de la GUI.

5.5.1. Interfaz principal de la aplicación

La interfaz principal de la GUI cuenta con tres sectores. En el primero, ubicado en la parte superior, se encuentran los menús de la aplicación, agrupados según la funcionalidad que cumple cada uno. En el lateral izquierdo se ubica el editor de texto para la lógica del programa, es decir, los archivos Rosy. Por último, en la parte derecha de la aplicación está ubicado un editor de texto más pequeño que muestra la sintaxis de los pines que se van agregando y, en la parte inferior se encuentra una imagen de referencia de la placa Arduino UNO para facilitar visualmente la configuración de pines (Ilustración 22).

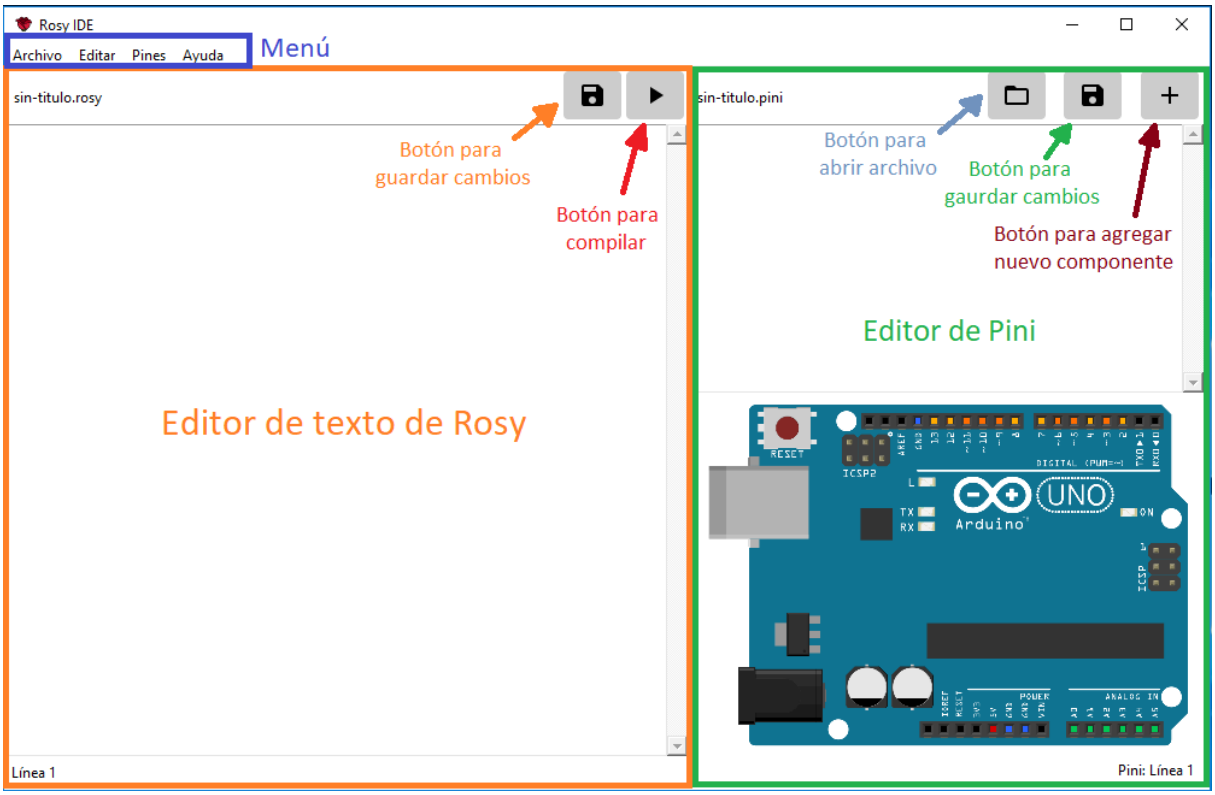


Ilustración 22: Interfaz principal de Rosy IDE

5. 5.1.1. Menú

El menú de la GUI se divide en cuatro categorías:

a) Archivo

Esta categoría agrupa funciones generales y opciones para el editor de texto de Rosy, tales como guardar los cambios, abrir un archivo o crear uno nuevo (Ilustración 23). También permite compilar el archivo actual y seleccionar una carpeta para almacenar los códigos objetivos generados al compilar. Además, se agregaron configuraciones de personalización como la de cambiar el tema (color de fondo) del IDE.

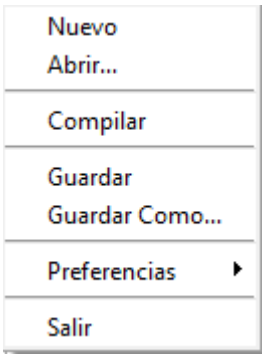


Ilustración 23: Opciones del menú Archivo

b) Editar

En esta categoría se encuentran las opciones de edición para el editor de textos de Rosy, tales como deshacer o rehacer una acción y la opción de dirigirse a una línea en particular del código (Ilustración 24). Esta última resulta útil cuando el código desarrollado presenta muchas lianas y ocurre un error de sintaxis, para ubicar de manera rápida el cursor en la línea donde se encuentra el error.

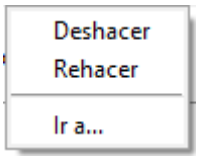


Ilustración 24: Opciones del menú Editar

c) Pines

La categoría de pines agrupa funcionalidades para la configuración de pines. Permite abrir, guardar o crear una nueva configuración (Ilustración 25). También es posible añadir un nuevo componente mediante la ventana de selección y configuración de componentes.

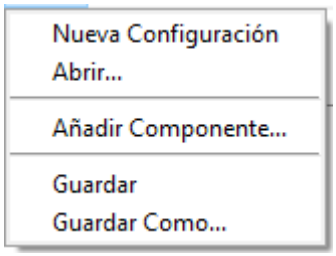


Ilustración 25: Opciones del menú Pines

d) Ayuda

Esta última categoría del menú brinda soporte al usuario con la opción de redirigirlo al sitio web de la documentación del proyecto, donde puede encontrar más información sobre el uso y manejo del lenguaje, junto a ejemplos prácticos de la gramática y funciones del mismo (Ilustración 26). También incorpora información acerca de la versión actual de Rosy IDE y la licencia de código libre GNU GPLv3 (GNU General Public License version 3)

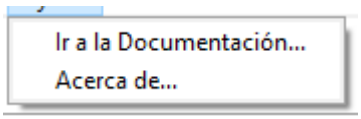


Ilustración 26: Opciones del menú Ayuda

5.5.1.2. Editor de texto (Rosy)

En la parte superior del editor de texto de Rosy se encuentran dos botones. El primero es un atajo de la opción de menú **Archivo > Guardar** para guardar los cambios en el archivo

Rosy abierto actualmente. Y el segundo, es el botón para compilar, es decir, generar el código objeto a partir de los códigos fuente de Rosy y Pini.

Para compilar correctamente el código es necesario tener abierto ambos archivos al mismo tiempo, el archivo Pini de configuración de pines y el archivo Rosy de la lógica del programa. Esto se debe que el compilador requiere los dos códigos fuentes al momento de compilar como se detalló en la sección 5.4.6.

Si el código presenta errores de sintaxis, se mostrará una alerta que indica el tipo de error, la línea donde ocurrió y el archivo al que pertenece (Ilustraciones 27 y 28).

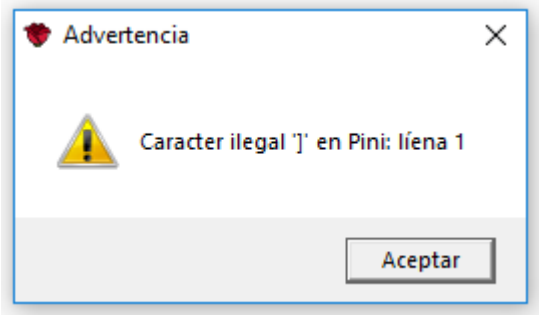


Ilustración 28: Mensaje de carácter ilegal en Pini

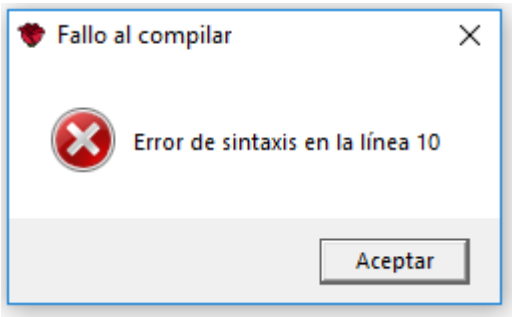


Ilustración 27: Mensaje de error de sintaxis de Rosy

Por lo general no deberían ocurrir errores de sintaxis o advertencias de caracteres ilegales en la configuración de pines, siempre que se utilice la interfaz gráfica de selección para configurar los mismos, ya que el código objeto de Pini es generado internamente por el ID. Pero si el usuario prefiere configurar los pines manualmente mediante el editor de texto de Pini existe la posibilidad del error humano al escribir la sentencia.

5.5.1.3. Editor de texto (Pini)

El editor de texto de Pini brinda al usuario la posibilidad de escribir manualmente la configuración de los pines. Al mismo tiempo, se utiliza como referencia para mostrar el nombre de los pines y el código de la configuración generada mediante el uso de la interfaz gráfica de selección de pines.

5.5.2. Interfaz de selección de componentes

Para abrir la interfaz de selección de componentes, la cual permite configurar los pines, se debe hacer click en el botón representado con signo más (+), ubicado en la esquina superior derecha del editor de texto de Pini, o bien, mediante la opción del menú **Pines > Añadir Componente**. Esta interfaz muestra botones con las imágenes y nombre de los componentes admitidos por Rosy (Ilustración 29).

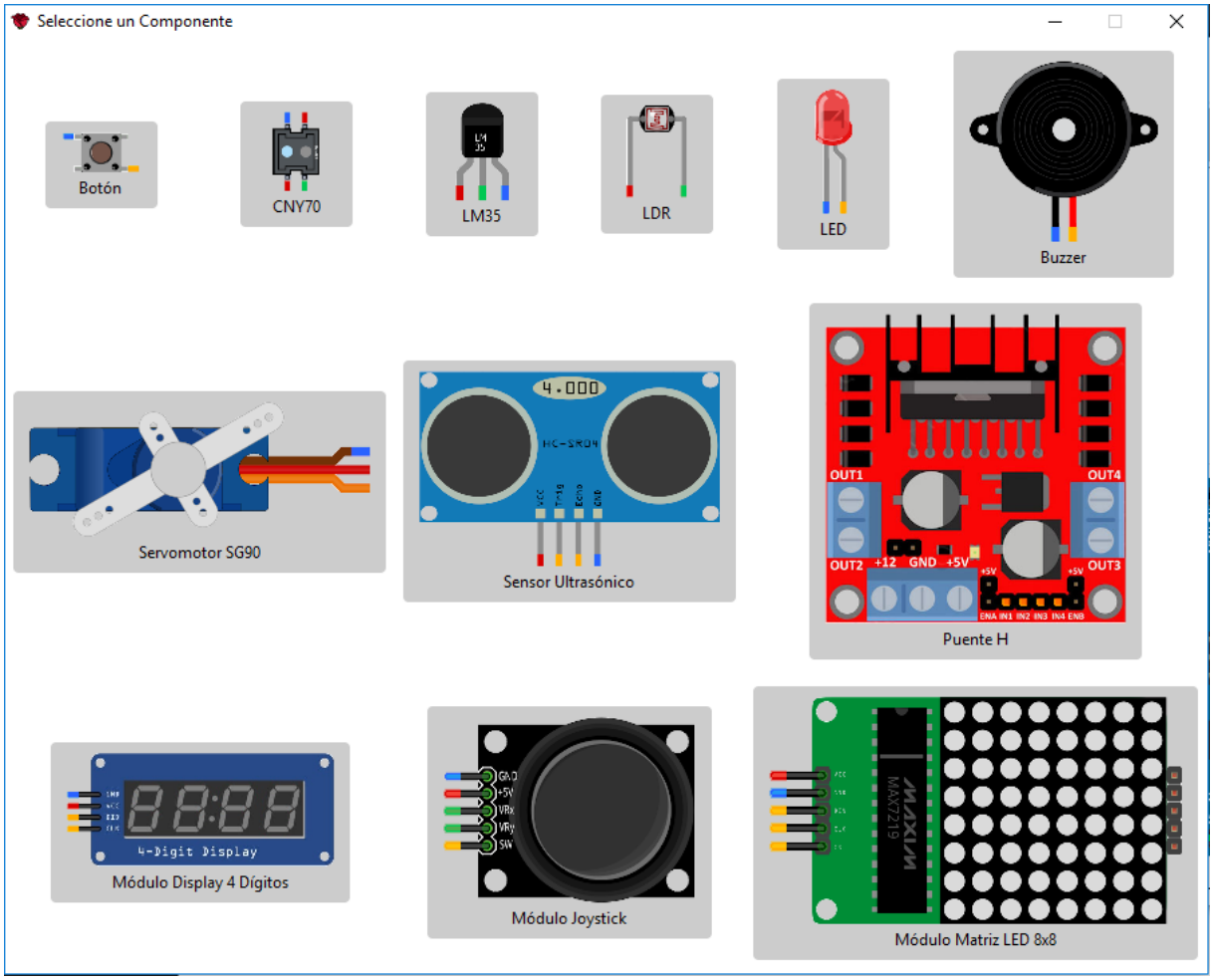


Ilustración 29: Interfaz de selección de componentes

Al hacer click sobre un componente, se despliega una ventana para configurar el nombre y los pines del mismo. Los pines se deben seleccionar de una lista desplegable con los tipos de pines admitidos por cada componente. Por ejemplo, el servomotor SG90 solo permite seleccionar pines digitales de tipo pwm.

A cada tipo de pin se le asignó un color en particular para diferenciarlos entre si y agilizar el ensamblado en la placa Arduino, ya que la imagen la misma, ubicada en la esquina inferior derecha de la aplicación, presenta el mismo código de colores en las ubicaciones correspondientes a cada tipo de pin (Ilustración 30).

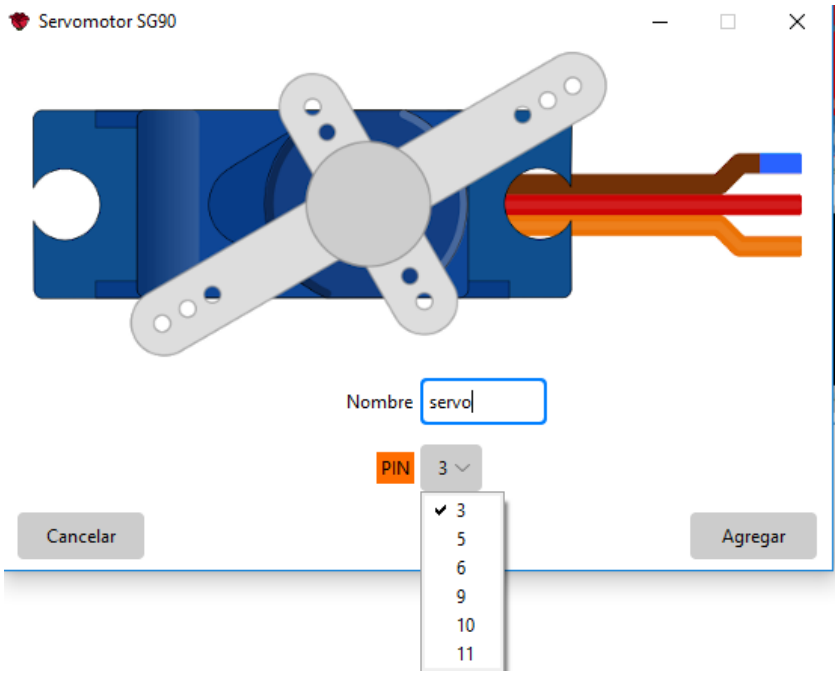


Ilustración 30: Ventana para configurar un componente

Al finalizar la configuración del componente se debe hacer click sobre el botón "Agregar" situado en la esquina inferior derecha de la venta. Esta acción genera un código con la configuración ingresada y lo agrega al editor de texto de Pini (Ilustración 31).

Ilustración 31: Código generado por la interfaz de selección de componentes

5.5.3. Resaltador de sintaxis

Los editores de texto de Rosy y Pini cuentan con un resaltador de sintaxis que diferencia las palabras reservadas de los nombres de variables y otros *tokens*, agilizando de esta manera la lectura y escritura del código.

Para diferenciar los *tokens* de las palabras reservadas, se importan las mismas en el código principal de la interfaz gráfica y se hace uso de una función que observa constantemente el estado del texto ingresado y si se detecta una palabra reservada o expresión regular, se crea una etiqueta que indica el color, la línea y columna que corresponden a esta palabra o expresión en el editor.

En los comentarios y cadenas de caracteres, que también son remarcadas de otro color, se utilizaron las expresiones regulares definidas de las mismas. En este caso, la función detecta donde inicia y finaliza la expresión regular para marcar de color solo los caracteres que la comprenden.

5.5.4. Archivo de preferencias

Se implementó un archivo de configuración, llamado config.ini, con el objetivo de almacenar las preferencias del usuario y que las mismas no se pierdan al cerrar el IDE. Las preferencias que guarda este archivo son: la carpeta donde se almacenan los códigos objetos compilados y el color del tema del IDE.

Cuando se inicia por primera vez el IDE se crea el archivo con valores por defecto. El valor predeterminado de la carpeta donde se guardan los archivos compilados es en una carpeta llamada "ino" en la misma ruta donde se instaló Rosy IDE, mientras que el tema por defecto es el de color claro. Estas preferencias pueden ser modificadas en el menú accediendo a la opción de **Archivo > Preferencias**.

5.6. Sitio web y documentación

La última etapa del desarrollo de la solución propuesta consiste en la implementación y despliegue de un sitio web que alberge los archivos descargables de Rosy IDE para los sistemas operativos compatibles y contenga toda la documentación necesaria para que los usuarios tengan una referencia del uso de la aplicación y todas sus funciones y características. El enlace para acceder al sitio web se encuentra en los Anexos.

La página web se desarrolló mediante el uso de un framework de JavaScript llamado Vue.js, en su versión 2, y se alojó en un servidor de Firebase proporcionado por Google. El sitio cuenta con cuatro secciones:

1. Página de inicio

La página de inicio del sitio muestra información acerca del IDE y las funcionalidades que ofrece el mismo, mostrando las ventajas de su uso y dando a conocer su simplicidad (Ilustración 32).

Ilustración 32: Página de inicio del sitio web de Rosy

Al inicio de esta página se agregaron dos botones de rápido acceso (Ilustración 33), uno que descarga la versión más reciente de Rosy IDE de acuerdo al sistema operativo de donde se está accediendo al sitio web, y otro que redirige a la documentación del IDE.

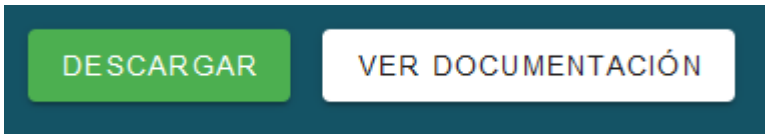


Ilustración 33: Botones de acceso rápido de la página principal

2. Descargas

La página de descargas contiene las dos alternativas para descargar Rosy IDE en su última versión disponible, una para sistemas GNU/Linux basados en Debian/Ubuntu y otra para sistemas Windows (Ilustración 34). Los archivos comprimidos que contienen el ejecutable de Rosy se almacenaron en el Storage de Firebase. El enlace para acceder al código fuente del programa se encuentra en los Anexos.

Ilustración 34: Página de descarga de Rosy IDE

Los botones de "¿Cómo instalar?" redirigen al usuario a la página de la documentación correspondiente en donde se indica paso a paso como realizar la instalación del programa en cada sistema operativo.

3. Documentación

En la página de documentación se encuentra toda la información necesaria para que los usuarios comprendan el funcionamiento del lenguaje de Rosy y el uso de las interfaces del IDE para aprovechar todas funcionalidades y opciones que brinda el mismo. El enlace para acceder a esta sección se incluye en los Anexos. La documentación de Rosy IDE se divide en tres secciones:

I. Empezar

Contiene información sobre el IDE, para qué sirve, cuáles son sus ventajas de uso y los pasos para su instalación en los sistemas operativos compatibles (Ilustración 35).

Ilustración 35: Documentación de Rosy IDE

Este apartado también incluye un ejemplo del código *blink*, popular en el desarrollo de Arduino, el cual permite encender y apagar un led conectado al pin 13. Este ejemplo es utilizado como base en la mayoría de los editores e IDE para Arduino. En la Ilustración 36, se muestra dicho ejemplo, comparándolo con el código necesario en el lenguaje de Arduno, lo que demuestra la reducción de líneas de código para la misma lógica.

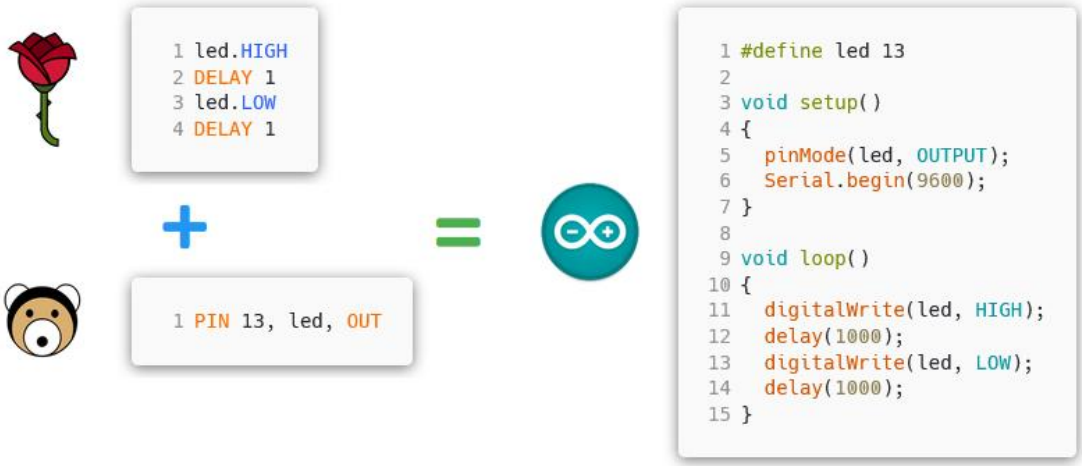


Ilustración 36: Comparación de códigos

II. Lenguaje

Esta sección es la más amplia, en ella se indican las buenas prácticas con el objetivo de mejorar la experiencia de los usuarios (Ilustración 37). Se muestra una lista de imágenes con los sensores y actuadores disponibles y el código correspondiente que se genera al

seleccionarlos desde la interfaz gráfica. Por último, se detallan todas las funciones, ciclos condicionales y estructura del lenguaje de Rosy con ejemplos del uso de su sintaxis.

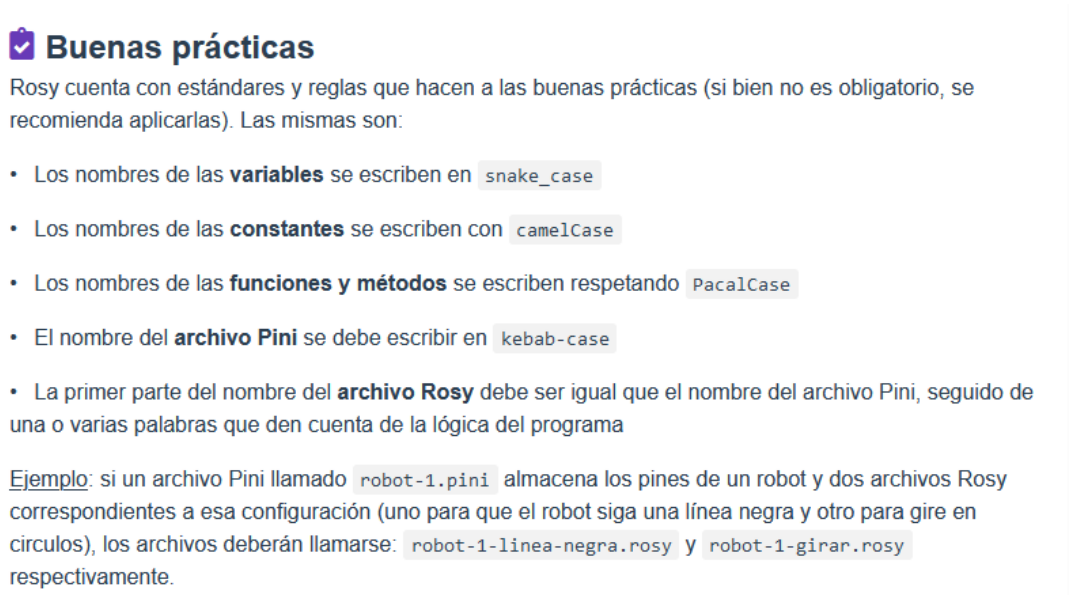


Ilustración 37: Buenas practicas del lenguaje

Las buenas practicas del lenguaje aplican sobre la denominación de los elementos que interactúan en el IDE. Se incorporaron con fin de diferenciar entre las variables, las constantes y las funciones y métodos. La forma de nombrar los archivos Pini y Rosy se enfoca en mejorar su búsqueda en el explorador de archivos, ya que, al nombrarlos de la manera recomendada, se ordenan de forma alfabética y facilita su visualización.

Se incorporaron imágenes de los actuadores y sensores descritos en el alcance del proyecto, junto a la sintaxis que se genera cuando son seleccionados mediante la interfaz gráfica (Ilustraciones 38 y 39).

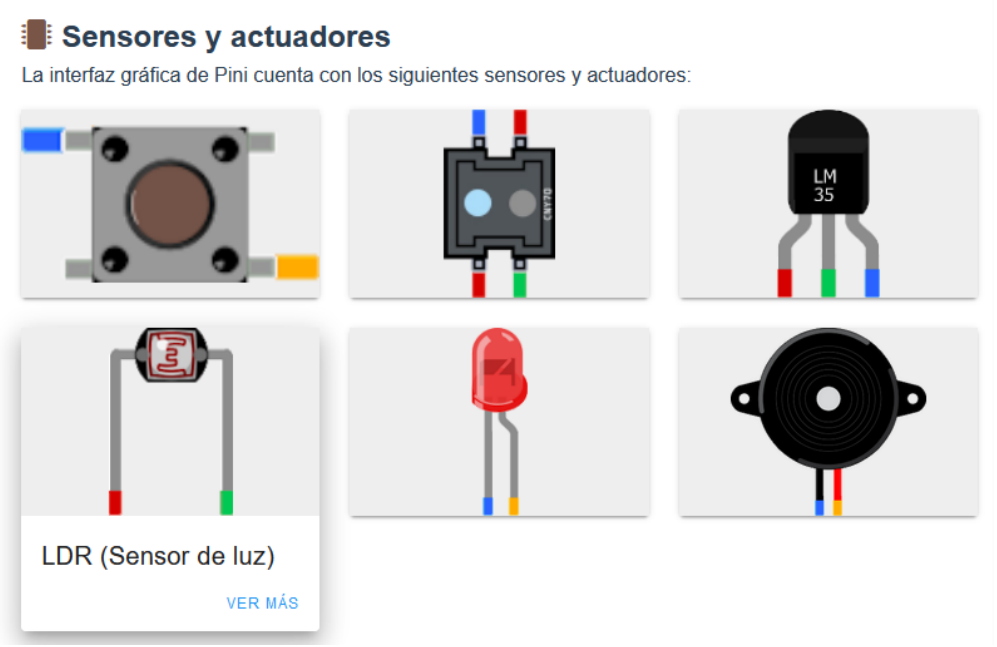


Ilustración 38: Imágenes de los sensores y actuadores compatibles con Rosy IDE

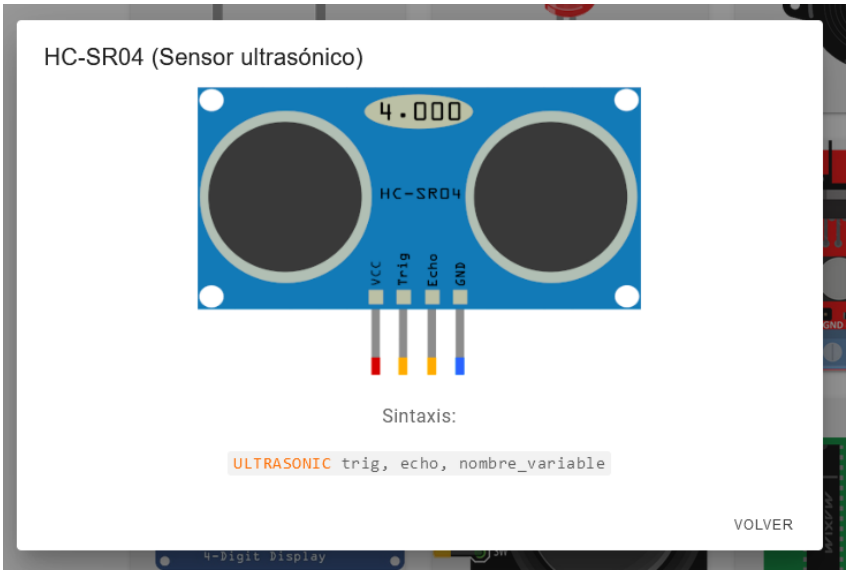


Ilustración 39: Sintaxis para el sensor ultrasónico

En la documentación, también se detalla la sintaxis de las principales reglas gramaticales del lenguaje. Para simplificar la lectura a los usuarios, se separaron en las siguientes categorías:

i. Variables y Tipos de datos

Se explica la sintaxis de todas las reglas definidas en la sección VII. Variables y tipos de datos, y las reglas 2, 7, 8, 9 y 10 de la sección I. Generales del capítulo **5. Desarrollo de la solución**.

ii. Ciclos

En esta categoría se muestran el uso y un ejemplo en seudocódigo (Ilustración 40) de todos los ciclos condicionales detallados en la sección II. Ciclos condicionales del capítulo **5. Desarrollo de la solución**.

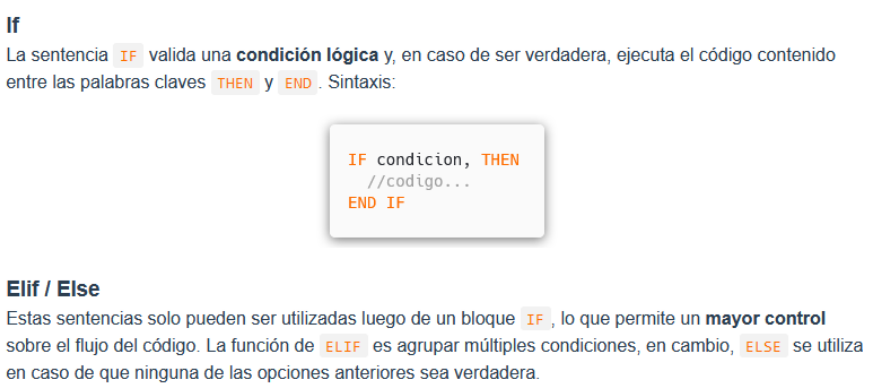


Ilustración 40: Explicación del ciclo condicional *if*

iii. Funciones y métodos

Se detalla la sintaxis para declarar y llamar funciones y/o métodos según las reglas 3, 4, 5 y 11 definidas en la sección I. Generales del capítulo **5. Desarrollo de la solución**. En la Ilustración 41, se muestran los ejemplos incluidos en esta categoría de la documentación.

```
// funcion sin variables de entrada
FUNCTION Funcion1, THEN
  //codigo...
END FUNCTION

// funcion con dos variables de entrada
FUNCTION Funcion2, WITH tipo_dato_1 variable_1, tipo_dato_2 variable_2, THEN
  //codigo...
END FUNCTION
```

```
// metodo sin variables de entrada
METHOD Metodo1, RETURN tipo_dato, THEN
  //codigo...
  RETURN valor
END METHOD

// metodo con una variable de entrada
METHOD Metodo2, WITH tipo_dato_1 variable_1, RETURN tipo_dato, THEN
  //codigo...
  RETURN valor
END METHOD
```

Ilustración 41: Ejemplos para definir funciones y métodos

iv. Estructura del lenguaje

Esta categoría explica la forma de estructurar el código escrito en Rosy (Ilustración 42), el cual se trata de la representación de la regla 1 de la sección I. Generales del capítulo **5. Desarrollo de la solución**.

```
// variables globales
GLOBAL tipo_dato variable = valor_inicial
//...

// funciones y metodos
FUNCTION Funcion, THEN
  //codigo...
END FUNCTION
METHOD Metodo, RETURN tipo_dato, THEN
  //codigo...
  RETURN valor
END METHOD
//...

// logica del programa
PRINT "hola mundo"
CALL Funcion
variable = CALL Metodo
PRINT variable
//...
```

Ilustración 42: Estructura general del código en Rosy

v. Funciones generales

Las funciones descritas en esta categoría de la documentación, hacen referencia a la sintaxis de las reglas definidas en la sección V. Funciones matemáticas del capítulo **5. Desarrollo de la solución**.

vi. Funciones de Arduino y Funciones específicas

Esta categoría engloba la descripción de la sintaxis de las reglas definidas en las secciones III. Funciones exclusivas de Arduino, y la sección IV. Funciones propias de Rosy, del capítulo **5. Desarrollo de la solución**. En cada regla, además de explicar su funcionamiento y sintaxis, se referencia a un ejemplo práctico en el cual se muestra el uso de la misma (Ilustraciones 43 y 44).

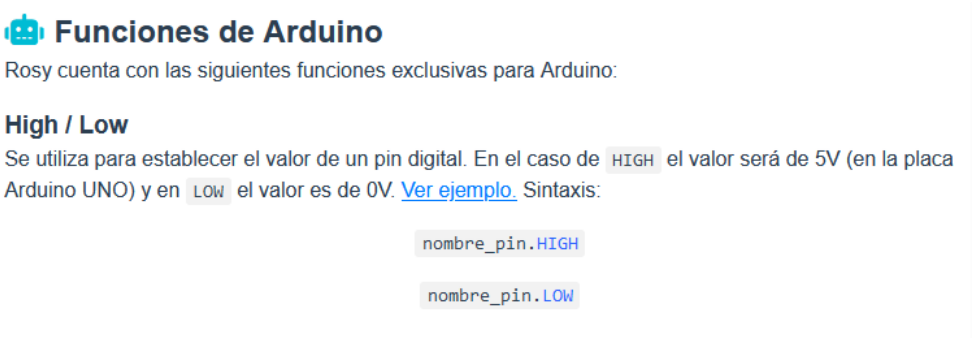


Ilustración 44: Explicación de las funciones de

Ilustración 43: Arduino Explicación de las funciones específicas de Rosy

III. Ejemplos

En esta sección se muestra un ejemplo con su correspondiente código fuente para cada sensor y actuador disponible, de esta manera se brinda al usuario una base para comprender las funcionalidades de los mismos (Ilustración 45).

Ilustración 45: Ejemplo de la función READ

4. Información adicional

La última sección del sitio web brinda información adicional sobre el IDE, su desarrollador, las tecnologías que usa y la licencia de código libre (Ilustración 46).

Ilustración 46: página de información adicional de Rosy IDE

6. Resultados

Rosy IDE es capaz de reducir de manera significativa el código necesario para realizar ciertas acciones con los sensores y actuadores utilizados. El claro ejemplo que lo demuestra es el uso de la función DISTANCE del sensor ultrasónico, donde en Rosy solo es necesario escribir un par de líneas, mientras que en el lenguaje de Arduino, esta acción requiere casi 30 líneas de código (Ilustraciones 47 y 48).

Ilustración 47: Código necesario en Rosy y Pini para medir la distancia con un sensor ultrasónico

```
#define TRIG 2
#define ECHO 4

long tiempo;
int distancia;

void setup()
{
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
  Serial.begin(9600);
}

void loop()
{
  digitalWrite(TRIG, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);
  tiempo = pulseIn(ECHO, HIGH);
  distancia = tiempo / 58.2;
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println();
  delay(1000);
}
```

Ilustración 48: Código necesario en C++ para medir la distancia con un sensor ultrasónico

El IDE reduce notablemente la cantidad de código necesario para la misma función y, además, disminuye la complejidad al configurar los sensores. A continuación, se mencionan los pasos necesarios en cada lenguaje para realizar mediciones con el sensor ultrasónico.

Utilizando el lenguaje de Arduino

Cuando se inicia un código nuevo en el lenguaje de Arduino, mediante el uso del programa Arduino IDE, descrito en la sección 2.2. Soluciones Similares del capítulo **2. Estado de la cuestión**, se obtiene el siguiente código, dónde se encuentran los ciclos *setup* y *void* para la configuración de pines y lógica del programa respectivamente (Ilustración 49).

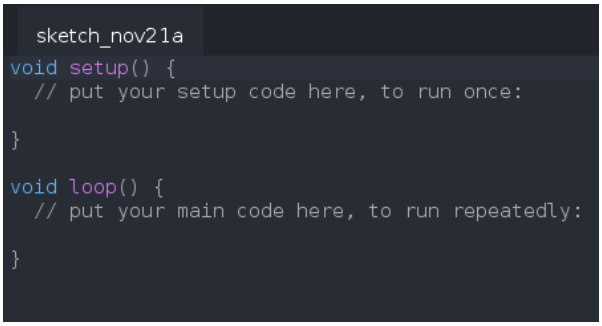


Ilustración 49: Código generado por Arduino IDE [captura de pantalla tomada por Macoritto Torcivia]

a. Lo primero que se debe realizar es declarar las variables para los pines del sensor ultrasónico y definir el pin al cual será conectado (Ilustración 50).

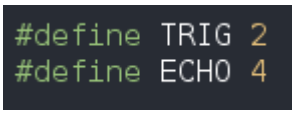


Ilustración 50: Declaración de pines en Arduino

b. Luego, se declaran las variables globales para calcular el tiempo y la distancia correspondientes a las variables de la formula descrita en la sección 5.1.1 del capítulo **5. Desarrollo de la solución** (Ilustración 51).

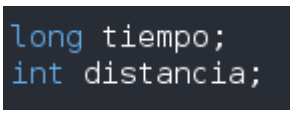


Ilustración 51: Declaración de variables en Arduino

c. El siguiente paso es configurar los pines del sensor ultrasónico (previamente declarados) en el ciclo *setup*. El pin correspondiente al *trigger* del sensor es de salida, mientras que el *echo* es de entrada. También se debe inicializar un *Serial* para mostrar los resultados en pantalla (Ilustración 52).

```
void setup() {
  pinMode(TRIG, OUTPUT);
  pinMode(ECHO, INPUT);
  Serial.begin(9600);
}
```

Ilustración 52: Ciclo *void* en Arduino

d. Por último, se escribe el código que calcula la distancia mediante el uso de las variables globales definidas en el paso b (Ilustración 53).

```
void loop() {
  digitalWrite(TRIG, LOW);
  delayMicroseconds(5);
  digitalWrite(TRIG, HIGH);
  delayMicroseconds(10);
  digitalWrite(TRIG, LOW);
  tiempo = pulseIn(ECHO, HIGH);
  distancia = tiempo / 58.2;
  Serial.print("Distancia: ");
  Serial.print(distancia);
  Serial.println();
  delay(1000);
}
```

Ilustración 53: Ciclo *setup* en Arduino

Al seguir los cuatro pasos descritos anteriormente, se obtiene el código que figura en la Ilustración 48. El proceso resulta complejo para un usuario sin experiencia en Arduino, ya que debe buscar información adicional sobre los pines del sensor ultrasónico para saber dónde conectarlos en la placa y, además, debe tener en cuenta que un pin se configura como entrada y el otro como salida.

Utilizando Rosy IDE

Para implementar el código que calcule la distancia con un sensor ultrasónico mediante el uso de Rosy IDE se deben seguir los pasos que se describen a continuación.

a. Abrir Rosy IDE y seleccionar el botón + situado en la esquina superior derecha de la pantalla. Se abrirá una ventana con las imágenes de los sensores y actuadores disponibles (Ilustración 29) y se debe hacer click sobre la imagen del sensor ultrasónico.

b. El siguiente paso es elegir un nombre descriptivo para el sensor y seleccionar los pines de la placa donde irá conectado el mismo. La lista desplegable para indicar los pines solo contiene los pines de la placa válidos para el sensor, así se evita que el usuario sepa de antemano cuáles son los pines dónde puede conectar el sensor (Ilustración 54). No es necesario definir si el pin es de entrada o salida, ya que esta diferenciación la realiza internamente el compilador de Rosy.

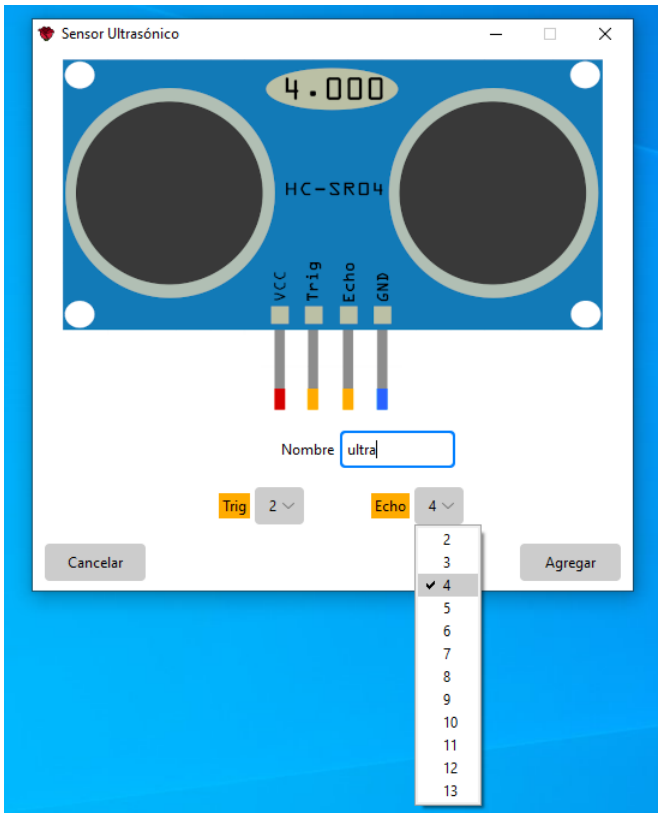


Ilustración 54: Ventana de configuración para el sensor ultrasónico

c. Por último, se debe escribir el código para obtener la distancia con el sensor ultrasónico previamente configurado y hacer click sobre el botón compilar ubicado en la esquina superior derecha del editor de texto (Ilustración 55).

Ilustración 55: Código en Rosy para obtener la distancia mediante el uso de un sensor ultrasónico

De esta forma se obtiene el código fuente en C++ listo para ser implementado en Arduino IDE. El código generado se guarda en la carpeta definida en el menú **Archivo -> Preferencias -> Carpeta de Archivos …**

Análisis de uso

Luego de comparar ambas formar de implementar la misma funcionalidad para un sensor ultrasónico, se puede observar que al utilizar el lenguaje de Arduino se requieren más conocimientos sobre los sensores, como por ejemplo el tipo de pines que requiere (entrada/salida y analógico/digital) y la distribución de los mismos en la placa.

En cambio, con el uso de Rosy IDE, estos conocimientos no son estrictamente necesarios ya que el mismo programa diferencia los tipos de pines y le muestra al usuario de manera sencilla todos los pines validos donde puede conectar el componente. Además, la interfaz gráfica de sensores y actuadores ayuda a la rápida distinción de cada componente de la placa.

Al utilizar el lenguaje de configuración de pines Pini se reduce de manera significativa el tiempo necesario para implementar un robot que requiera la misma configuración de pines, ya que Pini permite la reutilización de configuraciones para emplearlos con diferentes archivos Rosy, mientras que en Arduino IDE es necesario escribir el mismo ciclo *setup* en cada implementación.

Además de la facilidad de reconocer los sensores y actuadores gracias a sus figuras tamaño escala, otra ventaja de la interfaz gráfica de Pini, es la del sistema de colores que incorpora. Dicho sistema permite identificar rápidamente donde puede ir conectado un pin en la placa, lo que reduce el tiempo de ensamblado del robot.

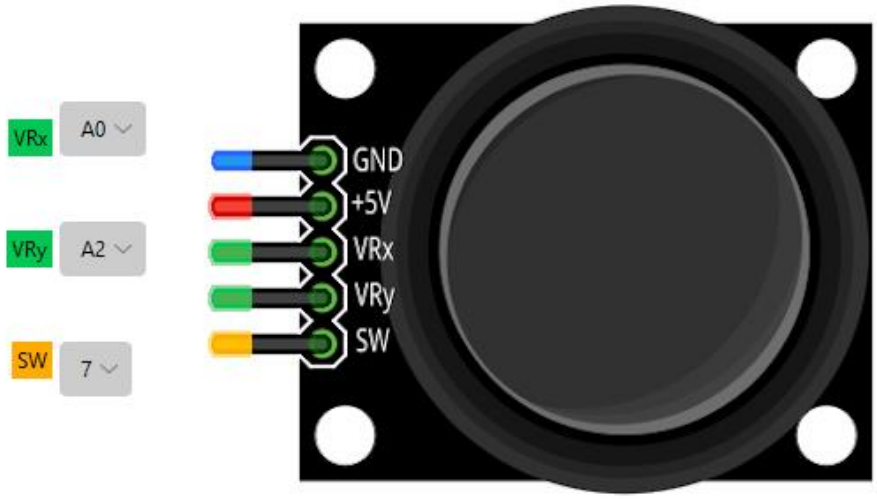


Ilustración 56: Interfaz de configuración de pines para el módulo joystick

La Ilustración 56 muestra la interfaz de configuración de pines para un módulo joystick. Como se puede observar, los pines VRx y VRy correspondientes a los ejes x e y del stick se marcan de color verde, lo que indica que son pines analógicos, mientras que el pin SW que detecta las pulsaciones del stick es un pin digital, por lo tanto, se marca en color naranja claro. De esta manera el ensamblado del módulo con la placa Arduino UNO resulta más sencillo y fácil de realizar ya que en la misma también se marcan los pines con los mismos colores según corresponda (Ilustración 57).

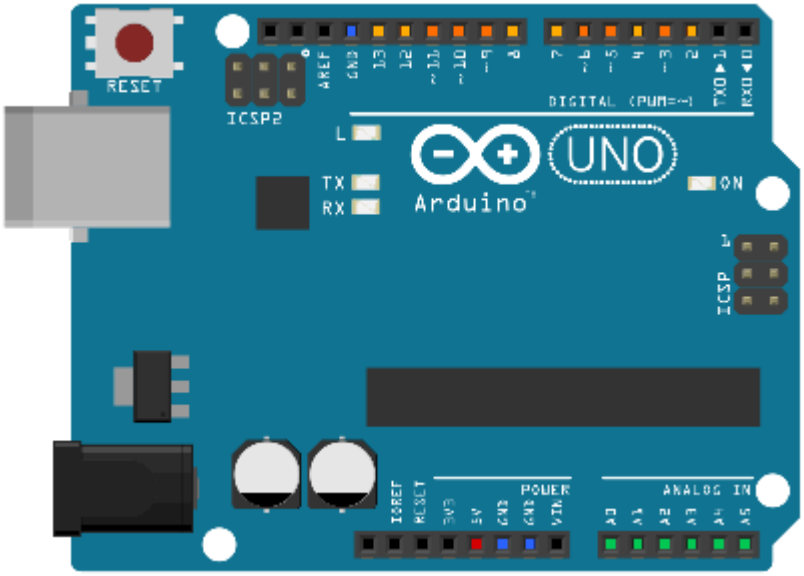


Ilustración 57: Modelo de placa Arduino UNO con colores de referencia en los pines utilizada en el IDE

En el modelo de la placa utilizado también se marcaron de colores los pines correspondientes a GND o tierra/negativo (en azul) y VCC o 5V (en rojo). Estos conectores se utilizan en la mayoría de los sensores y actuadores incorporados. Los pines digitales pwm se diferencias de los digitales normales al ser de un naranja más oscuro.

En base a lo expuesto anteriormente, se evidencia que el IDE resultante del trabajo simplifica la codificación y estructuración de un proyecto en Arduino, al facilitar la configuración de los componentes en la placa a través de la interfaz gráfica. La misma también ayuda en la conexión de los pines mediante su sistema de colores y las imágenes a escala.

Estas funciones reducen el esfuerzo requerido para ensamblar los componentes en la placa a los usuarios que no están familiarizados con la robótica o electrónica, mientras que la sencillez de la sintaxis de Rosy simplifica la escritura de código a los usuarios con poca experiencia en programación.

7. Conclusiones y futuras líneas de investigación

El desarrollo y despliegue final de la solución propuesta en este proyecto de grado dio como resultado un IDE innovador, capaz de facilitar y simplificar el ensamblado de robots junto a la escritura del código fuente de éstos, la cual contiene sus funcionalidades. Se implementó un lenguaje de programación propio con una interfaz gráfica moderna que ayuda en el proceso de configuración de pines y, al mismo tiempo, los almacena para futuros usos, de esta manera se logra reducir el código necesario y evitar la redundancia.

Los criterios de éxito descritos al inicio del proyecto, en el capítulo 1, se cumplieron satisfactoriamente, ya que se logró llevar a cabo el desarrollo de la solución según lo planteado en el capítulo 4 y al mismo tiempo alcanzar todos los objetivos planteados en el capítulo 3.

Con lo que respecta a las futuras líneas de investigación, este proyecto deja abiertas varias y diversas posibilidades. Algunas de ellas podrían ser:

- Mejorar las funcionalidades de los editores de texto añadiendo más opciones tales como reemplazar palabras, agregar un cursor multilínea, autocompletado de palabras reservadas y predicción de nombres de variables entre otras.

- Agregar más componentes, sensores y actuadores, como pantallas LCD, módulos de Bluetooth y WiFi, reconocimiento de tarjetas RFID, sensores de humedad, controles para leds RGB, etc.

- Incorporar compatibilidad con placas Arduino NANO y Arduino MEGA y permitir que desde el IDE se pueda subir el código ya compilado a las placas, sin la necesidad de utilizar el software de Arduino, para lograr una independencia total con aplicaciones de terceros.

- Implementar nuevas reglas gramaticales en Rosy que permitan el uso estructuras de datos complejas como vectores, matrices y punteros.

En conclusión, Rosy IDE abre las puertas a múltiples proyectos, tanto de desarrollo como de investigación, con el fin de perfeccionar el lenguaje y generar nuevas reglas gramaticales que optimicen el código objetivo.

Como resultado de este proyecto, no solo se logró cumplir los objetivos y criterios definidos, sino que se dio origen a una sólida solución ante los problemas que otros IDE o lenguajes similares no pudieron cubrir en su totalidad. Rosy IDE reúne características y funcionalidades innovadoras que simplifican el desarrollo e implementación de proyectos de robótica, programación o electrónica que, junto a la licencia de código libre, otorga a sus usuarios las libertades y posibilidades que la mayoría los softwares de desarrollo deberían incluir.

8. Bibliografía

Alternative Arduino Interfaces. https://learn.sparkfun.com/tutorials/alternative-arduino-interfaces/all. Página vigente al 16/05/20

Aprendiendo Arduino. Microcontrolador. https://aprendiendoarduino.wordpress.com/category/microcontrolador/. Página vigente al 14/05/20

Arduino Forum. Visual Framework GUI for Arduino, WIP. https://forum.arduino.cc/index.php?topic=439846.0. Página vigente al 16/05/20

Arduino. Introducción. https://www.arduino.cc/en/Guide/Introduction. Página vigente al 14/05/20

COPAIPA. Honorarios Mínimos Indicativos de Informática. http://www.copaipa.org.ar/informatica/. Página vigente al 28/07/22

GNU. ¿Qué es el software libre? https://www.gnu.org/philosophy/free-sw.es.html. Página vigente al 16/05/20

Herrador, R. E. (2009). Guía de Usuario de Arduino. http://electroship.com/documentos/Arduino_user_manual_es.pdf. Página vigente al 14/05/20

Louden, K. C. (2005). Construcción de compiladores. Principios y práctica. Editorial Thomson. ISBN 970-686-299-4.

Luis Llamas, 2015. Medir distancias con Arduino y sensor de ultrasonidos HC-SR04. https://www.luisllamas.es/medir-distancia-con-arduino-y-sensor-de-ultrasonidos-hc-sr04/. Página vigente al 09/08/22

Meza Ortíz, C. O.; Rivera López, R.; Ceballos Mejía, J. de J. & Rodríguez León, A. (2010). Una Arquitectura Modular para el desarrollo de un IDE que apoye a la Enseñanza de los Fundamentos de la Programación Orientada a Objetos. Revista Internacional de Educación en Ingeniería. Volumen 3. Páginas 23-32.

miniBloq. Blog. http://blog.minibloq.org/. Página vigente al 16/05/20

Ministerio de Educación de la Nación Argentina. (2017). Escuelas técnicas. Características institucionales y desempeños. https://www.argentina.gob.ar/sites/default/files/escuelas_tenicas_caracteristicas_institucionales_y_desempenos_web_a4_simple.pdf. Página vigente al 09/08/22

Morris Mano, M. (1998). Arquitectura de computadoras. Editorial Prentice-Hall Hispanoamericana. ISBN 968-880-361-8.

Ocaña Moratilla, Alberto. 2015. ByteCode: ¿Sabes lo que realmente programas en Java? https://www.adictosaltrabajo.com/2015/04/16/byte-code/. Página vigente al 18/05/20

PMI. (2013). Guía de los fundamentos para la dirección de proyectos (Guía del PMBOK). Project Management Institute, Inc. Quinta edición. ISBN 978-1-62825-009-1.

Pressman. R. S. (2010). Ingeniería del software. Un enfoque práctico. Editorial McGraw-Hill. ISBN 978-607-15-0314-5.

Quiroz, P.; Muñoz, R. & Noël, R. (2012). Desarrollo de un lenguaje de programación y entorno de desarrollo que facilite la programación de robots LEGO mindstorms. Memorias del XVII Congreso Internacional de Informática Educativa, TISE. (http://www.tise.cl/volumen8/TISE2012/68.pdf). Chile.

Red Hat. El concepto de IDE. https://www.redhat.com/es/topics/middleware/what-is-ide. Página vigente al 14/05/20

Ruiz Catalán, J. (2010). Compiladores. Teoría e implementación. Editorial Alfaomega. ISBN 978-607-7854-68-5.

Russell, S. J. & Norvig, P. (2004). Inteligencia Artificial. Un enfoque moderno. Editorial Pearson. ISBN 978-84-205-4003-0.

S4A. Home page. http://s4a.cat/index_es.html. Página vigente al 16/05/20

Sindicato de Investigadores y Docentes de la UTN. Salarios Básicos Docentes Nivel Universitario Anido 2022. https://www.sidut.org.ar/images/Documentos/Salarios_2022.pdf. Página vigente al 09/08/22

Stallman, Richard. Por qué el «código abierto» pierde de vista lo esencial del software libre. https://www.gnu.org/philosophy/open-source-misses-the-point.es.html. Página vigente al 16/05/20

Visuino. Home page. https://www.visuino.com/. Página vigente al 16/05/20

Anexos

Diagrama de Gantt detallado

Rosy IDE: un entorno de desarrollo integrado para proyectos Arduino	**121 días**
1. Análisis de sensores y actuadores utilizados	**14 días**
1.1. Análisis del funcionamiento de los sensores y actuadores	2 días
1.2. Diseño de funciones y métodos	3 días
1.3. Codificación de funciones y métodos	5 días
1.4. Pruebas de funciones y métodos	3 días
1.5. Implementación de funciones y métodos	1 día
2. Desarrollo del archivo de configuración de pines (Pini)	**25 días**
2.1. Análisis inicial	2 días
2.2. Desarrollo del analizador léxico	8 días
2.2.1. Diseño del analizador léxico	2 días
2.2.2. Codificación del analizador léxico	4 días
2.2.3. Pruebas del analizador léxico	2 días
2.3. Desarrollo del analizador sintáctico	12 días
2.3.1. Diseño del analizador sintáctico	3 días
2.3.2. Codificación del analizador sintáctico	6 días
2.3.3. Pruebas del analizador sintáctico	3 días
2.4. Pruebas integrales	2 días
2.5. Implementación de Pini	1 día
3. Desarrollo del lenguaje de programación para el IDE (Rosy)	**45 días**
3.1. Análisis inicial	2 días
3.2. Desarrollo del analizado léxico	9 días
3.2.1. Diseño del analizador léxico	2 días
3.2.2. Codificación del analizador léxico	5 días
3.2.3. Pruebas del analizador léxico	2 días
3.3. Desarrollo del analizador sintáctico	31 días
3.3.1. Diseño del analizador sintáctico	6 días
3.3.2. Codificación del analizador sintáctico	20 días
3.3.2.1. Ciclos condicionales	5 días
3.3.2.2. Funciones exclusivas de Arduino	4 días
3.3.2.3. Funciones propias de Rosy	6 días
3.3.2.4. Otras funciones auxiliares	5 días
3.3.3. Pruebas del analizador sintáctico	5 días
3.4. Pruebas integrales	4 días
3.5. Implementación de Rosy	1 día
4. Desarrollo del IDE	**36 días**
4.1. Análisis de librería gráfica	2 días
4.2. Diseño del IDE	8 días
4.2.1. Diseño de la GUI para Pini	5 días
4.2.2. Diseño del editor de texto	3 días
4.3. Desarrollo del IDE	15 días
4.3.1. Codificación de la GUI para Pini	15 días
4.3.1.1. Ventanas y vistas	6 días
4.3.1.2. Botones y enlaces	5 días
4.3.1.3. Integración con Pini	4 días
4.3.2. Codificación del editor de texto	10 días
4.3.2.1. Resaltador de sintaxis	4 días
4.3.2.2. Funciones de edición	2 días
4.3.2.3. Funciones de guardado	4 días
4.4. Pruebas del IDE	12 días
4.4.1. Pruebas de la GUI para Pini	4 días
4.4.2. Pruebas del editor de texto	3 días
4.4.3. Pruebas integrales	5 días
4.5. Implementación del IDE	2 días
5. Documentación y despliegue	**20 días**
5.1. Análisis inicial	2 días
5.2. Documentación del IDE	6 días
5.2.1. Documentación de Pini	2 días
5.2.2. Documentación de Rosy	4 días
5.3. Diseño de la página web	3 días
5.4. Codificación de la página web	7 días
5.5. Publicación de la página web	2 días

Código fuente del IDE

https://github.com/martins36/Rosy-IDE

Documentación de Rosy

https://rosy.com.ar/documentacion

Sitio web de Rosy IDE

https://rosy.com.ar/