**UNIVERSIDAD DE TARAPACÁ**



**ESCUELA UNIVERSITARIA DE INGENIERÍA INDUSTRIAL, INFORMÁTICA Y DE SISTEMAS**



Área de Ingeniería en Computación e Informática



**Informe Final  
C.A.M**

**Integrantes: Nicolas Jorquera  
José-Ignacio Le-Blanc  
Matias Sandoval  
Luis Soto**

**Asignatura: Proyecto I**

**Profesor(es): Diego Aracena  
Ricardo Valdivia**

ARICA, 28 noviembre del 2018

# Historial de Cambios

|  |  |  |  |
| --- | --- | --- | --- |
| **Fecha** | **Versión** | **Descripción** | **Autor(es)** |
| 06/09/2018 | 1.0 | Primera formulación de Proyecto | Matias Sandoval |
| 07/07/2018 | 1.1 | Segunda formulación de Proyecto | Nicolás Jorquera  José-Ignacio Le-Blanc  Matias Sandoval  Luis Soto |
| 22/11/2018 | 1.2 | Corrección Informe de Avance 2 | Nicolás Jorquera  José-Ignacio Le-Blanc  Matias Sandoval  Luis Soto |
| 28/11/2018 | 1.3 | Implementación de nuevos puntos para Informe Final | Nicolas Jorquera |

# Tabla de Contenidos

1. Panorama General
   1. Introducción (contexto)
   2. Objetivo General
   3. Objetivos Específicos
   4. Restricciones
   5. Entregables
2. Organización del Personal

2.1. Descripción de Roles

2.2. Personal que cumplirá los Roles

2.3. Mecanismos de Comunicación

1. Planificación del Proyecto

3.1. Actividades (nombre, descripción, responsable, producto)

3.2. Asignación de tiempo (carta Gantt Redmine)

3.3. Personal-rol asignado

3.3. Gestión de Riesgos (ver plantilla para el Tratamiento de los Riesgos)

1. Planificación de los Recursos

4.1. Recursos Hardware-Software requeridos

4.2. Estimación de Costos (Hardware, Software, Recursos Humanos)

1. Análisis – Diseño

5.1 Especificación de Requerimientos (incluyendo método/algoritmos considerados para resolver el cubo Rubik)

5.2 Arquitectura Propuesta (incluyendo aspectos de comunicación)

5.3 Diseño de la Interfaz Usuario

1. Implementación

6.1 Descripción de los programas implementados

(entradas, salidas, procesos)

6.2 Diagrama de interacción entre programas

1. Pruebas

7.1 Descripción de las pruebas realizadas

7.2 Resultados de las pruebas

1. Resultados

8.1 Estado Final del Proyecto

8.2 Conclusiones

8.3 Trabajo Futuro

1. Referencias (estándar IEEE)
2. **Panorama General**  
     
   **1.1 Introducción**

En 1974 se creó un famoso juego interactivo llamado cubo de Rubik y desde entonces ha formado parte de la vida de muchas personas, llegando incluso al punto de haber competencias. Además de aflorar la imaginación de las personas explorando diversas combinaciones de giros, permitiéndoles formar patrones.

Estamos acostumbrados a realizar dichos patrones con algoritmos efectuados con nuestras manos, pero el reto de este proyecto es hacer que un robot sea capaz de hacerlos, recibiendo las instrucciones desde una aplicación.

**1.2 Objetivo General**

Construir un robot con el set Lego EV3, capaz de resolver un cubo de Rubik. Desarrollar programas con el lenguaje Python, en los cuales estén los patrones para que el robot los realice. Además de desarrollar una aplicación para interactuar con dicho robot, enviándole las instrucciones pertinentes para ejecutar los patrones.

**1.3 Objetivos Específicos**

* Construir un robot que sea capaz de resolver un cubo rubik 3x3.
* Crear programas en Python con los algoritmos para el armado del cubo.
* Desarrollar una interfaz en Python para enviar los patrones al robot.
* Investigar e implementar un mecanismo de comunicación remota entre la interfaz y el robot.

**1.4 Restricciones**

Para llevar a cabo el proyecto se cuenta con un set de Lego EV3, un cubo Rubik 3x3. La conexión remota se realiza mediante Rpyc, la cual es utilizada a través de Python.   
Posteriormente se buscará mejorar la estética de la aplicación.   
El tiempo para trabajar en el proyecto durante la semana de manera presencial es de 9 horas, y el tiempo de duración del mismo es hasta que finalice el segundo semestre del presente año.  
Para comenzar a ocupar el producto, los brazos del robot deben estar encima del Cubo Rubik.  
Una vez que comiencen los movimientos del robot, no se podrá manipular el cubo con las manos.  
Para armar el Cubo, debe ser visto el robot de frente, ya que los patrones están visto desde ese punto de referencia del usuario.

**1.5 Entregables**

-Bitácoras.

-Robot “EV3” (Patrones ya implementados).

-Aplicación C.A.M.

-Manual de usuario.

-Formulación de proyecto.

-Wiki.

1. **Organización del Personal**  
     
   **2.1 Descripción de Roles**

**Programador:** Escribe, depura y mantiene el código fuente del programa actualizado, libre de errores. Con el cual trabajará el robot “EV3”.

**Ensamblador:** Organizador y montador de la estructura del robot “EV3”,

**Planificador:** Ordenar, guardar y enviar los documentos para llevar un orden en las horas de trabajo grupal.

**Diseñador:** Encargado de la estética de la aplicación.

**Creador Conexión Remota:** Encargado de conectarse al robot “EV3”, por medio de una conexión remota.

**Creador de Algoritmos:** Adapta los algoritmos para ser realizados por el robot “EV3”.

**Desarrollador Aplicación:** Escribe y desarrolla la aplicación para la comunicación con el robot “EV3”.

**2.2 Personal que cumplirá los Roles**

**Programador:** José-Ignacio Leblanc.

**Ensamblador:** Luis Soto, José-Ignacio Leblanc.

**Planificador:** Matias Sandoval.

**Diseñador:** Luis Soto.

**Creador Conexión Remota:** Nicolas Jorquera.

**Creador de Algoritmos:** Matias Sandoval, José-Ignacio Leblanc.

**Desarrollador Aplicación:** Nicolas Jorquera, Luis Soto, Matias Sandoval.

**2.3 Mecanismos de Comunicación**

**Discord:** Programa por el cual se puede comunicar mediante videollamada y chat de voz, el cual fue ocupado para organizar las reuniones y algunos trabajos pendientes en clases.

**WhatsApp:** Aplicación para comunicarse mediante mensajes o audios de voz. Se utilizó para comunicarse de maneras más rápida y compartir detalles breves.

**Dropbox:** Software para almacenar datos en la nube. Se utilizó para subir la documentación de los avances del proyecto.

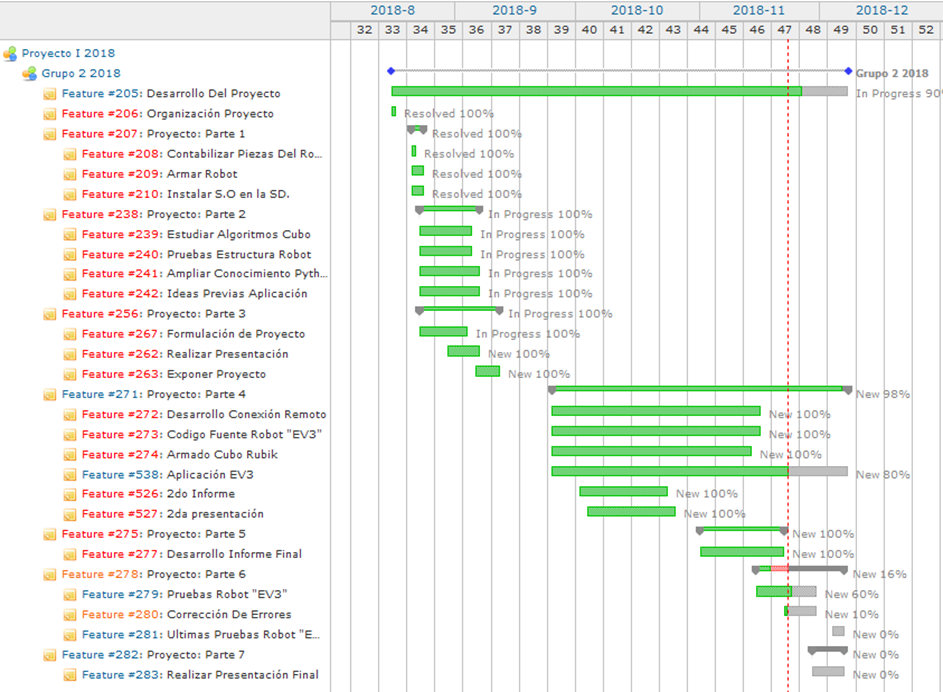
**Correo Electrónico:** Software para enviar archivos. Se utilizó de apoyo al momento que Dropbox no nos permitía subir algunos archivos.

1. **Planificación del Proyecto**

**3.1 Actividades (nombre, descripción, responsable, producto)**

|  |  |  |  |
| --- | --- | --- | --- |
| **NOMBRE** | **DESCRIPCION** | **RESPONSABLE** | **PRODUCTO** |
| Organización Proyecto | Distribución de tareas a los integrantes del grupo, donde cada uno dedicara su mayor tiempo. | Equipo de Trabajo | Realizar las actividades asignadas de forma organizada. |
| Contabilizar Piezas del Robot | Revisión de piezas contenidas dentro de la caja del Robot EV3, para lograr el objetivo de armado. | Luis Soto  Jose Le-Blanc | Disponer de las piezas correctas de forma ordenada. |
| Armar Robot | Armado del Robot, donde se realizará la estructura dependiendo de las dimensiones del cubo. | Luis Soto  Jose Le-Blanc | Robot Ev3. |
| Instalar S.O en la SD | Ingresar el sistema operativo al “brick” para lograr controlar el robot. | Nicolas Jorquera | Contar con el S.O en brick. |
| Estudiar Algoritmos del Cubo | Aprendizaje de los algoritmos del cubo Rubik, para lograr llegar a los patrones deseados. | Matias Sandoval  Jose Le-Blanc | Patrones del Cubo Rubik. |
| Pruebas Estructura Robot | Realizar movimientos con algoritmos básicos en Python, para comprobar que la estructura del robot responde de buena manera a dichos movimientos. | Jose Le-Blanc | Estructura optima del Robot, sin ningún detalle. |
| Estudiar Python | Aprendizaje de Python en la realización del programa fuente. | Jose Le-Blanc  Nicola Jorquera | Conocimiento de las librerías de Python. |
| Desarrollo conexión Remota | Establecer conexión entre el robot y la computadora, desde la cual se emitirán los patrones de armado. | Nicolas Jorquera | Conexión entre robot y la computadora. |
| Aplicación EV3 | Donde se realizará una aplicación, usando la conexión remota. | Luis Soto  Nicolas  Jorquera | Aplicación de la conexión remota. |
| Armado del cubo de Rubik | Armar el cubo Rubik, adaptándose los movimientos del robot. | Matias Sandoval | Traducción de los patrones “normales” a forma del robot. |
| Realizar Código Fuente EV3 | Programar los patrones del cubo Rubik previamente adaptados a los movimientos del robot. “Código fuente”. | José Le-Blanc  Matias Sandoval | Programa que contiene los algoritmos de patrones. |
| Pruebas Robot EV3 | Realizar las pruebas del robot, pero ahora con todos los patrones ya programados. | Jose Le-Blanc  Nicolas Jorquera  Matias Sandoval | Respuesta óptima del robot al ejecutarse los patrones en éste. |
| Realización de Documentos. | Se realizarán las bitácoras semanales, informes, etc. | Matias Sandoval | Un orden al realizar el proyecto. |

* 1. **Asignación de tiempo (carta Gantt Redmine)**



**3.3 Gestión de Riesgos (Ver plantilla para el tratamiento de los riesgos)**

|  |  |  |  |
| --- | --- | --- | --- |
| **RIESGOS** | **PROBABILIDAD DE OCURRENCIA** | **NIVEL DE IMPACTO** | **ACCIÓN REMEDIAL** |
| Problemas estructurales del robot en general. | 80% | 1 | Solucionar dichos problemas de la mejor manera. |
| El personal no cumple con las tareas en el tiempo estimado. | 40% | 2 | Intercambiar roles entre los mismos integrantes del grupo. |
| Un miembro del personal no asiste por problemas de salud o motivos personales. | 50% | 2 | Que otro miembro cubra la actividad asignada al personal ausente. |

1. **Planificación de los Recursos**

**4.1 Recursos Hardware-Software requeridos**

* **Hardware:**

-Robot “EV3” compañía de lego.

-Cubo Rubik 3x3.

-Notebook.

-Tarjeta SD.

-Dongle WIFI.

* **Software:**

-Visual Studio Code.

-Discord.

-Dropbox.

-Correo Electrónico.

-WhatsApp.

-Vi.

-Python.

-S.O ev3dev.

-Wi-fi

**4.2 Estimación de Costos (Hardware, Software, Recursos Humanos)**

|  |  |
| --- | --- |
| **Hardware/Software/RH** | **Costos ($ CLP)** |
| Robot “EV3” | $490.209 |
| Cubo Rubik 3x3 | $5.000 |
| Notebook | $600.000 |
| Tarjeta SD | $5.000 |
| Dongle WIFI | $8.000 |
| HH Desarrollador Aplicación | $8.000 ($936.000) |
| HH Programador | $7.000 ($819.000) |
| HH Ensamblador | $4.000 ($468.000) |
| HH Planificador | $8.000 ($936.000) |
| HH Creador de Algoritmo | $4.000 ($468.000) |
| HH Diseñador | $2.000 ($234.000) |
| HH Creador Conexión Remota | $7.000 ($819.000) |
| Software de Apoyo | Libre De Costo |
| **TOTAL** | **$5.938.209** |

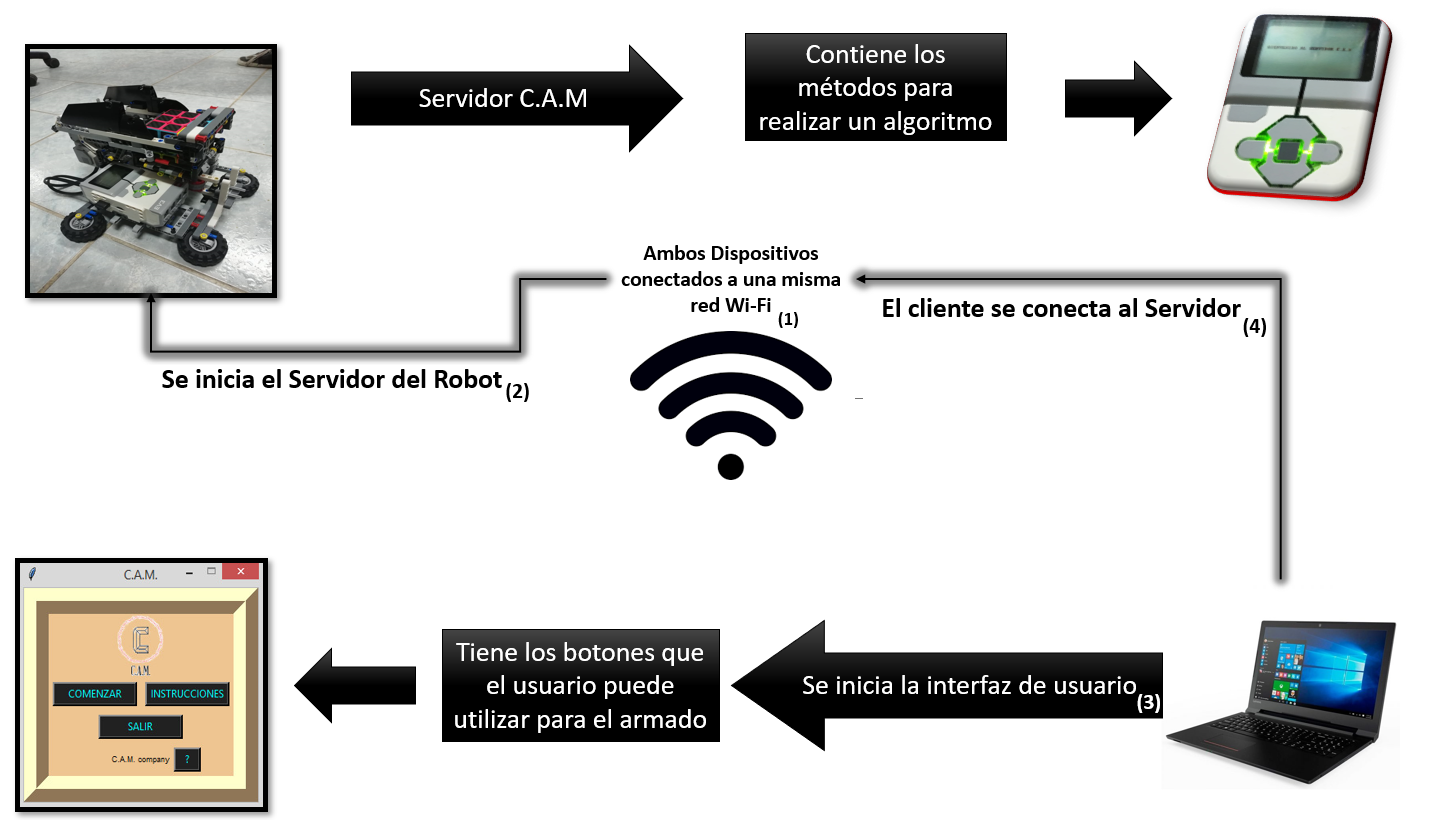
|  |  |
| --- | --- |
| **Fecha de inicio/Fecha de término** | **Fecha vacaciones Septiembre** |
| 16-08🡪 27-11 | 11-09 🡪 25-09 |

**\*SE CONSIDERARON 9 HORAS SEMANALES DE TRABAJO Y TRABAJAMOS 13 SEMANAS DANDO UN TOTAL DE 117 HORAS\*.**

1. **Análisis – Diseño**

**5.1 Especificación de Requerimientos**

|  |  |
| --- | --- |
| **Funcionales** | **No funcionales** |
| -Un robot que sea capaz de armar un cubo Rubik 3x3.  -Contar con los movimientos que van desde básicos hasta complejos.  -Una interfaz a través de la cual ordenarle al robot qué algoritmo realizar. | -El robot debe estar construido a partir del set Lego MindStorms EV3.  -Los algoritmos deben estar codificados en el lenguaje de programación Python. |

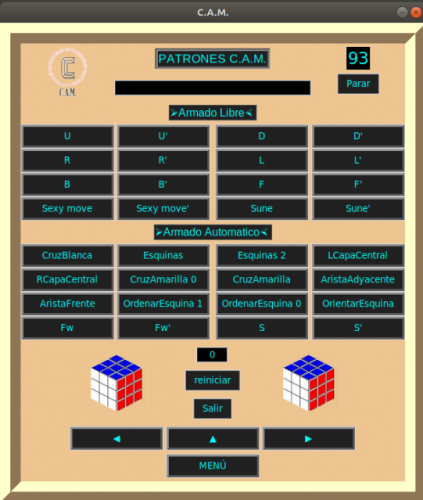
**5.2 Arquitectura**

(1) Ambos dispositivos (robot y notebook) deben estar conectados a la misma red Wi-Fi.  
(2) Iniciar el archivo **Servidor\_C.A.M.py** para tener la conexión remota con el notebook.  
(3) El usuario debe iniciar la interfaz de usuario para realizar los movimientos requeridos.  
(4) La interfaz se conectará al servidor del robot y el usuario podrá empezar a usar la aplicación C.A.M.

**5.3 Diseño de la interfaz Usuario**



Esta es la ***pantalla principal*** de la interfaz, que dispone de un Menú, donde se encuentran 3 botones principales de opciones. El primer botón: **COMENZAR**, al momento de pulsarlo abrirá la ***pantalla de patrones,*** donde el usuario podrá comenzar a interactuar con los botones que controlan al robot**.** El segundo botón: **INSTRUCCIONES**,permite visualizar el manual de usuario, donde se explican los modos de armado que tendrá el usuario a su disposición, además de cómo usar el producto de manera óptima. Y el tercer botón:  **SALIR**, se podrán cerrar todas las ventanas de la interfaz. Además, se tendrá un botón adicional: **?,** da la opción al usuario de saber más de nuestro equipo de trabajo y quiénes conforman el mismo.



Esta es la ***pantalla de Patrones***, donde se encontrarán una variedad de botones de armado, que serán divididos en dos categorías: El modo de **Armado Libre** y el modo de **Armado** **Automático**. Además, cuenta de un cronómetro y un contador de movimientos los cuales le darán una forma de medir sus tiempos de armado y eficacia, a modo de competir con los usuarios, viendo quién arma el cubo en menos movimientos, o en la menor cantidad de tiempo. Esta pantalla también cuenta con música de fondo, con el fin de motivar al usuario al momento de estar usando nuestro producto.

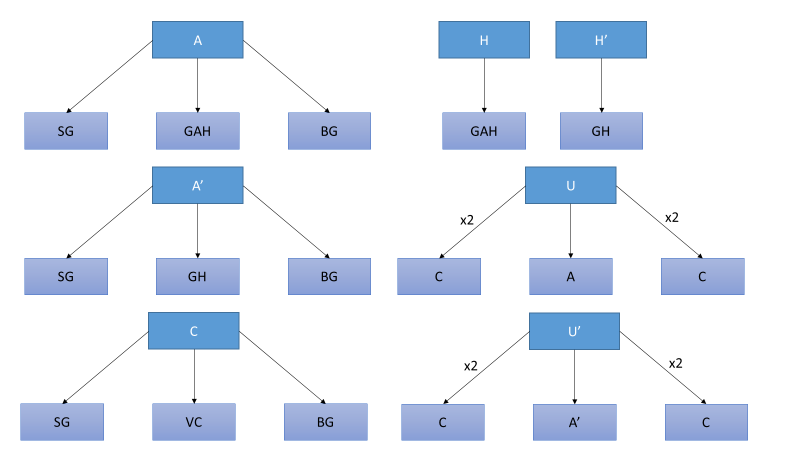
1. **Implementación**

Interacción entre programas

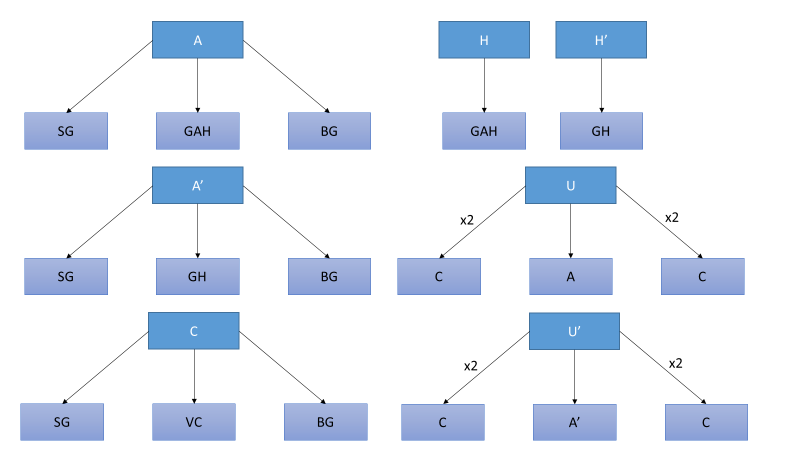
* GH: Función del motor para mover el cubo en sentido horario.
* GAH: Función del motor para mover el cubo en sentido antihorario.
* BG: Función del motor para bajar la garra.
* SG: Función del motor para subir la garra.
* VC: Función de la garra para voltear el cubo Rubik.

\*El procedimiento se lee de izquierda a derecha.

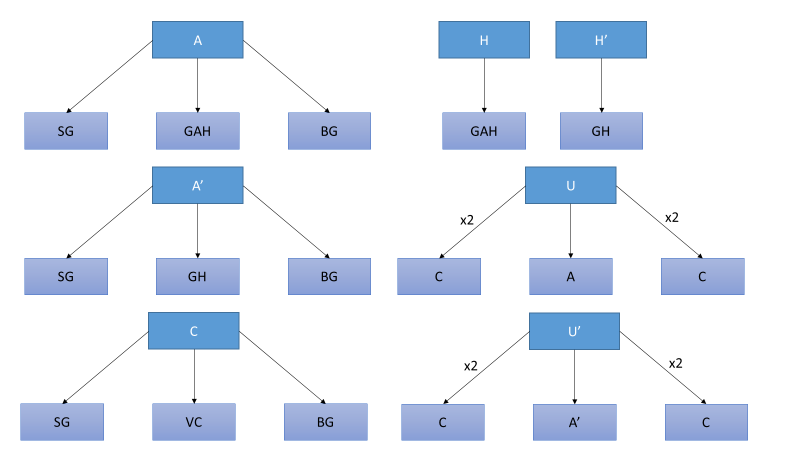
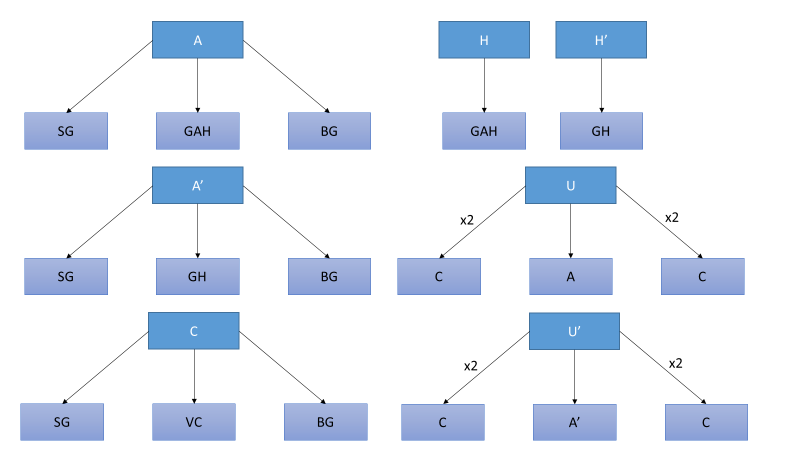
Función A:



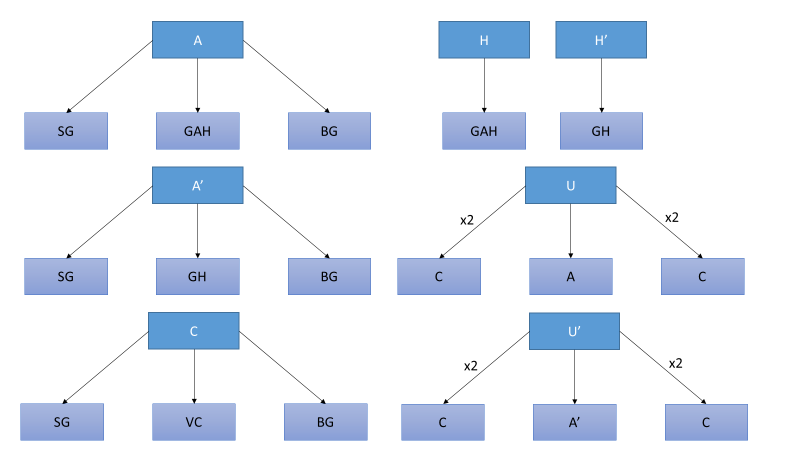
Función A’:



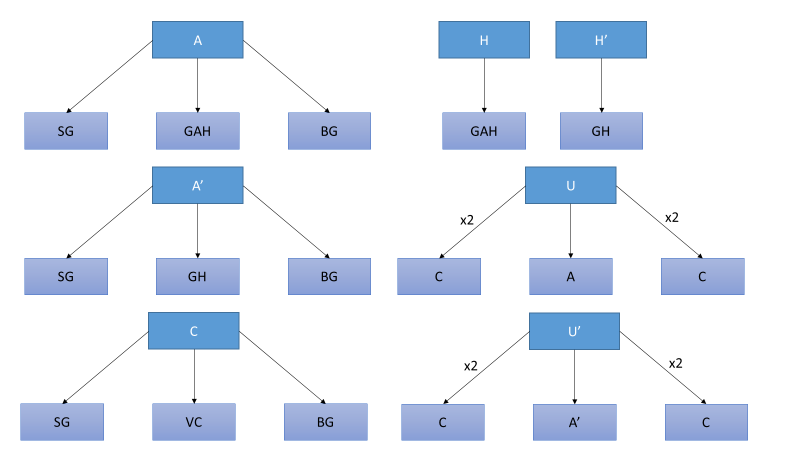
Función H: Función H’:

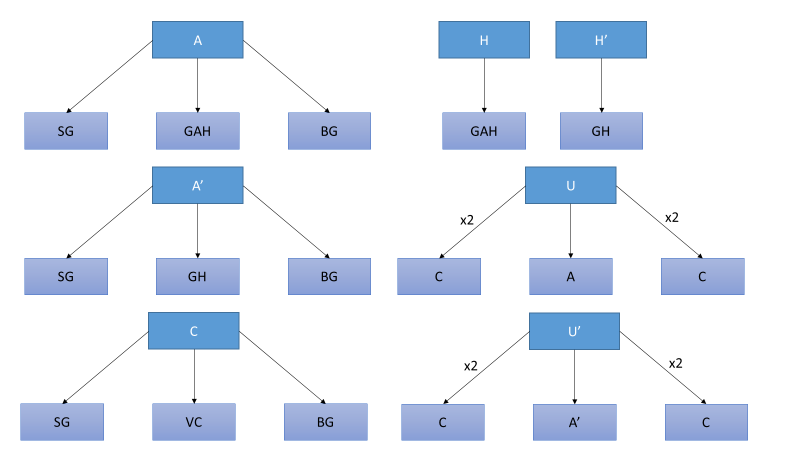
Función C:



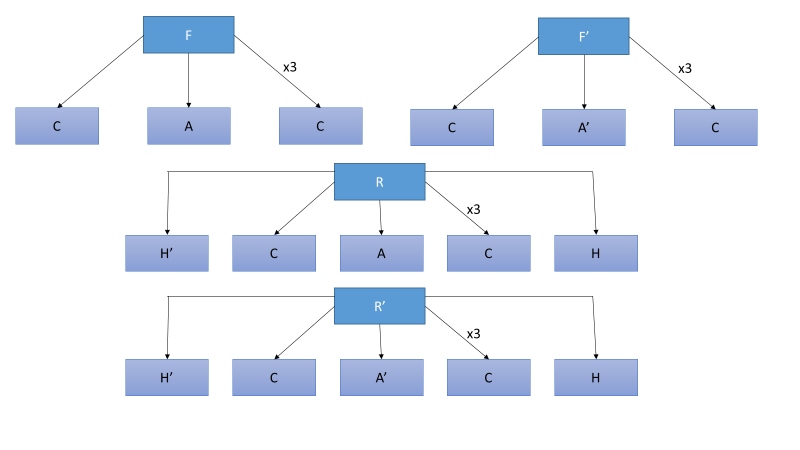
Función U:



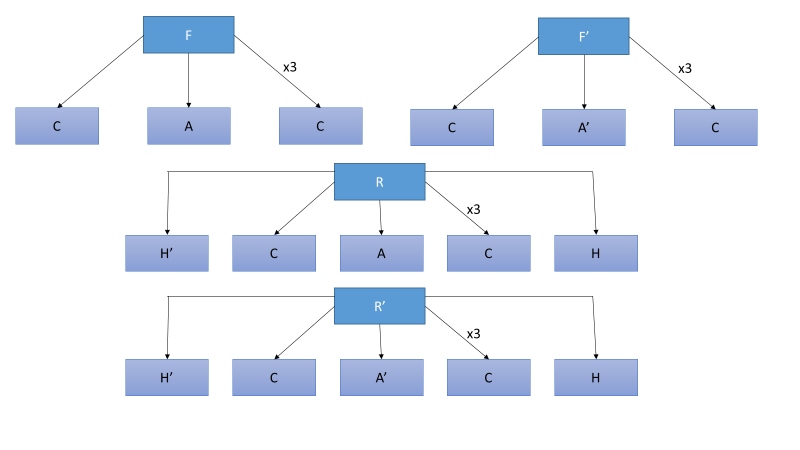
Función U’:



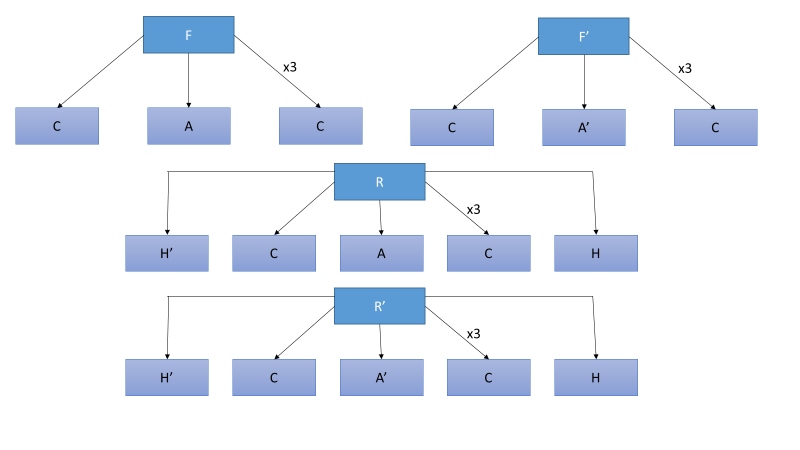
Función F:



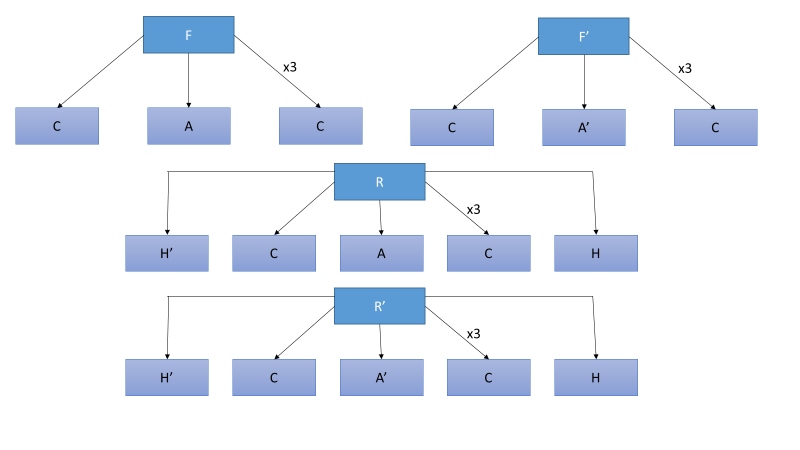
Función F’:



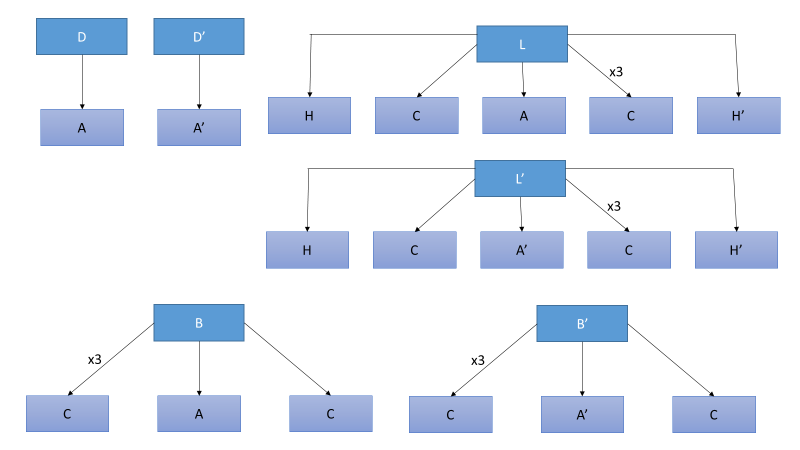
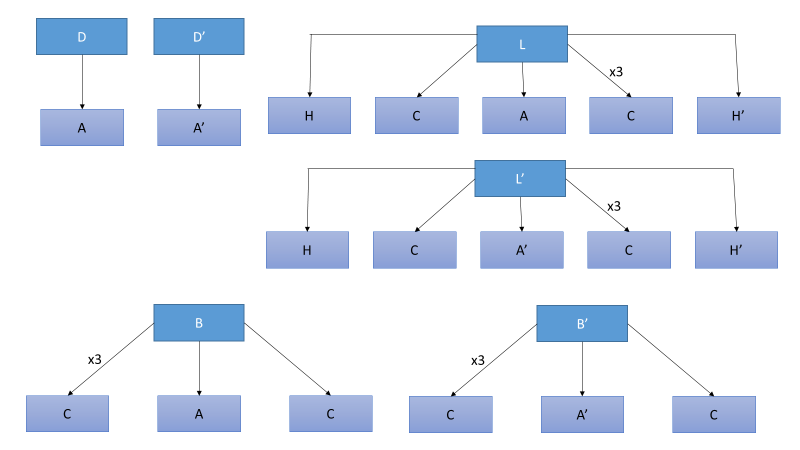
Función R:



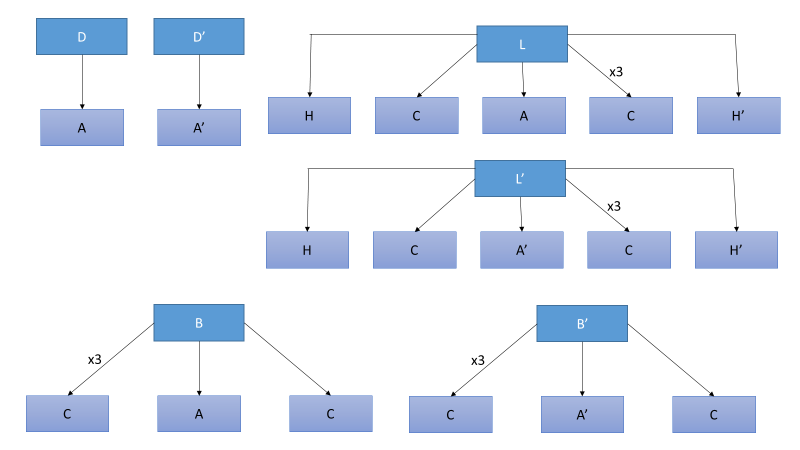
Función R’:



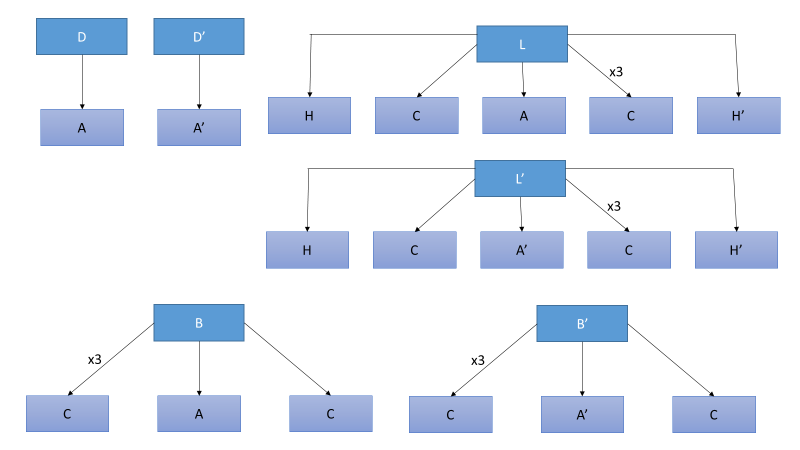
Función D: Función D’:

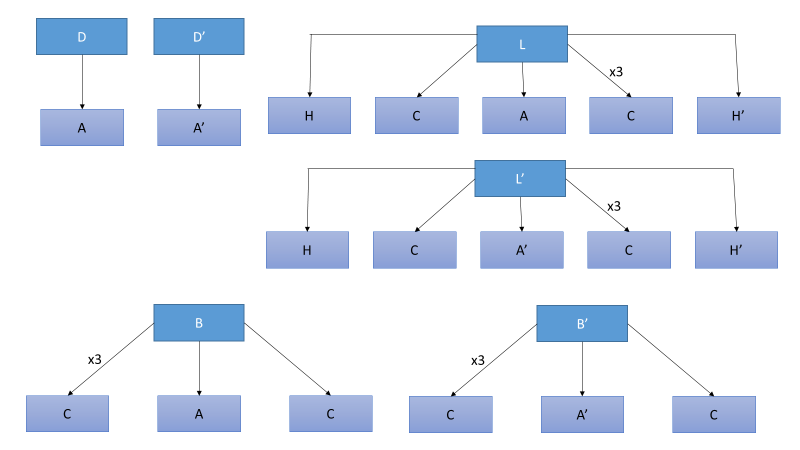
Función L:



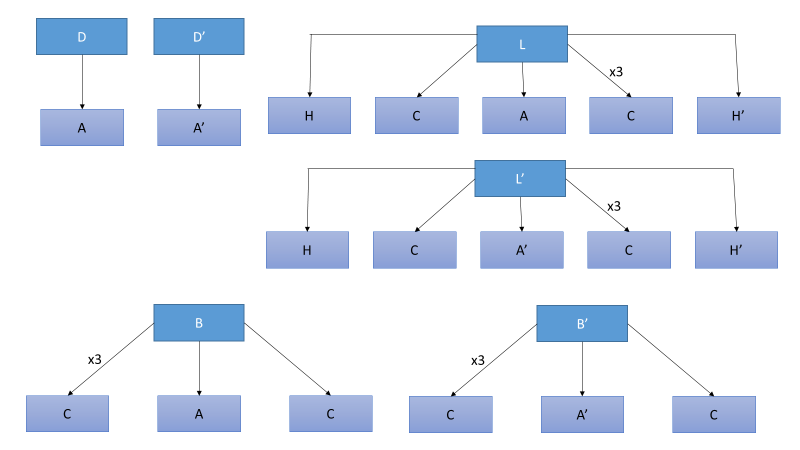
Función L’:



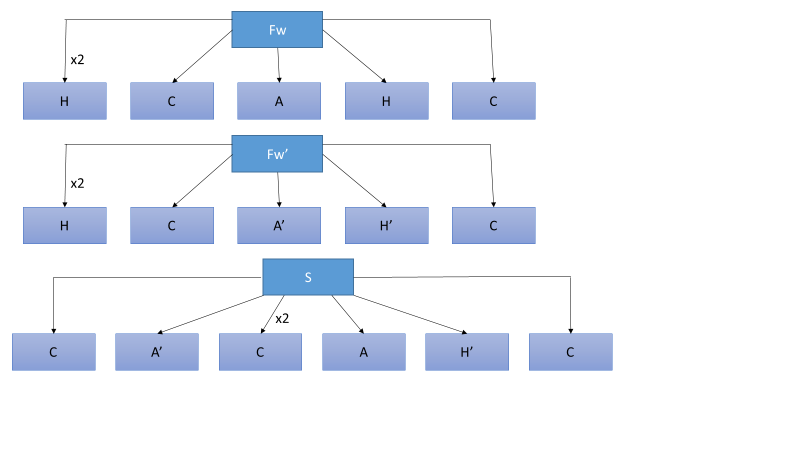
Función B:



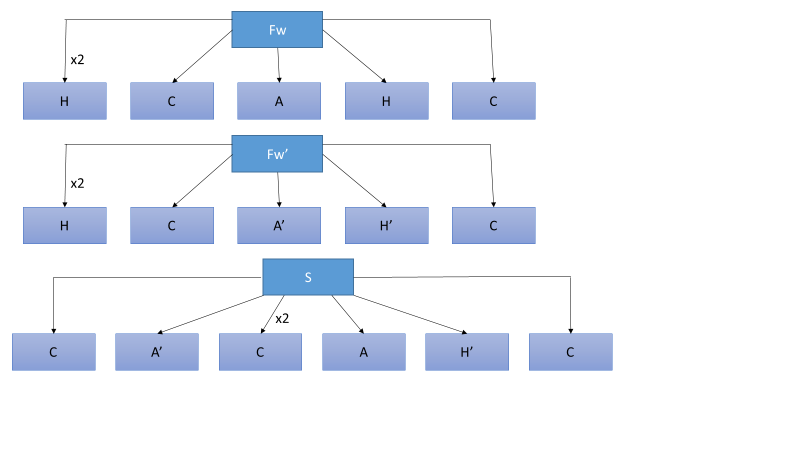
Función B’:



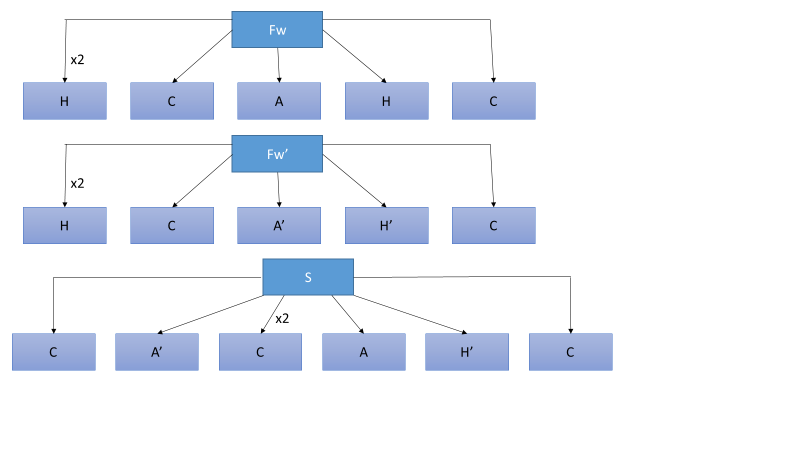
Función Fw:



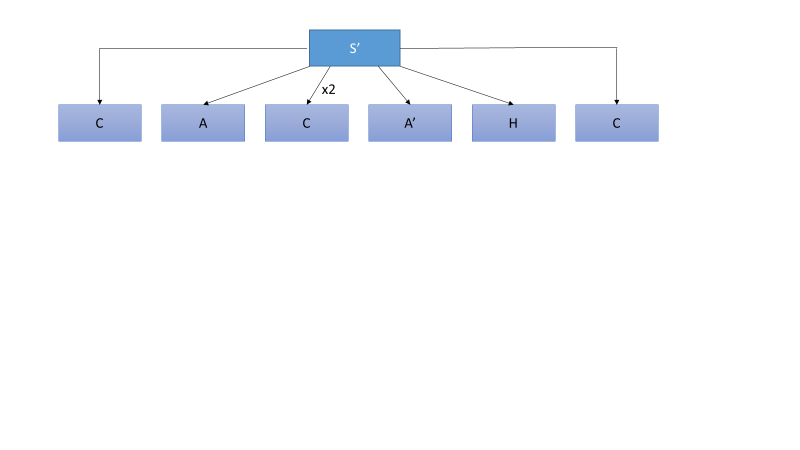
Función Fw’:



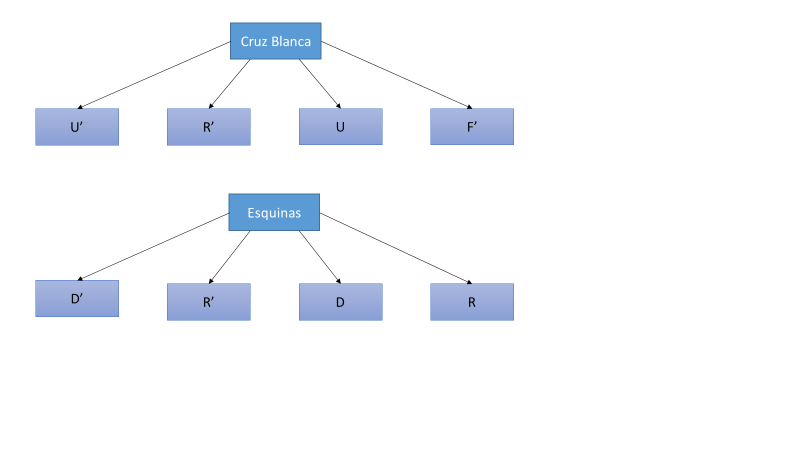
Función S:



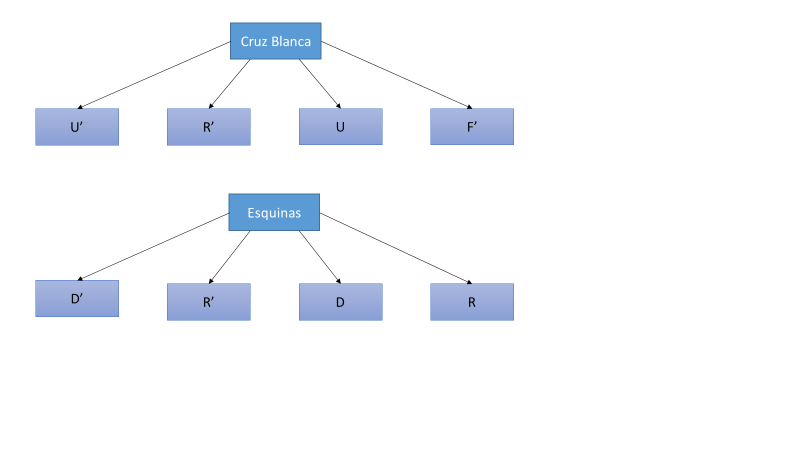
Función S’:



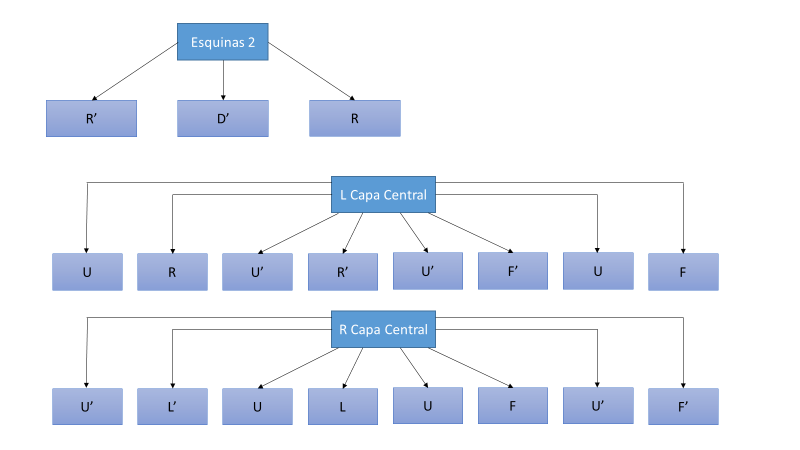
Función Cruz Blanca:



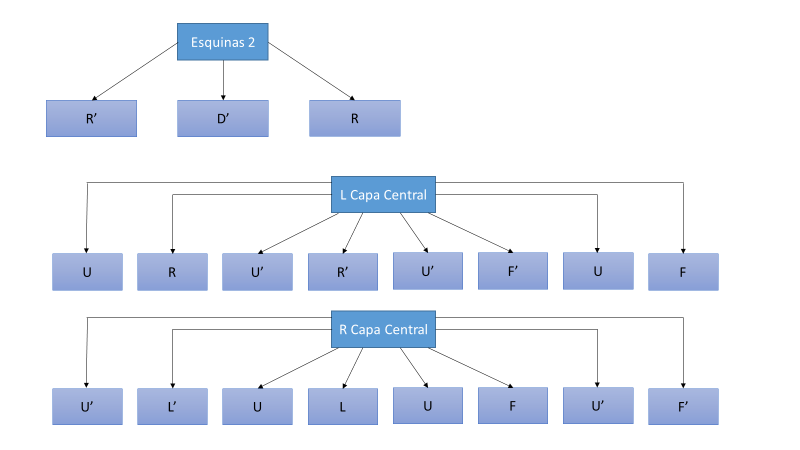
Función Esquinas:



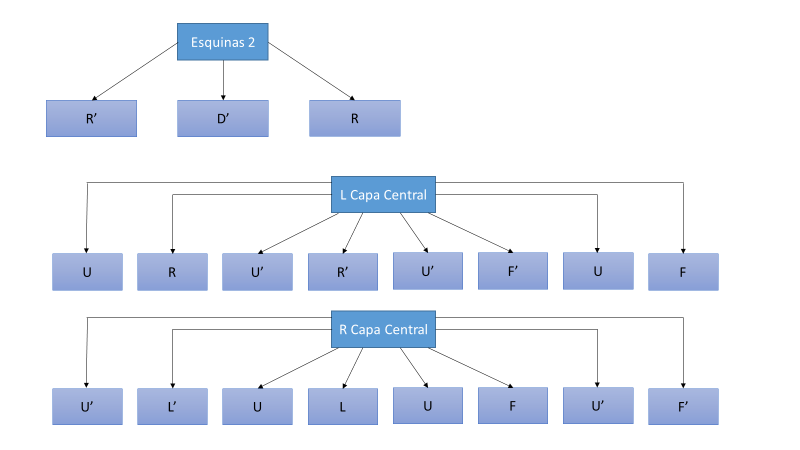
Función Esquinas 2:



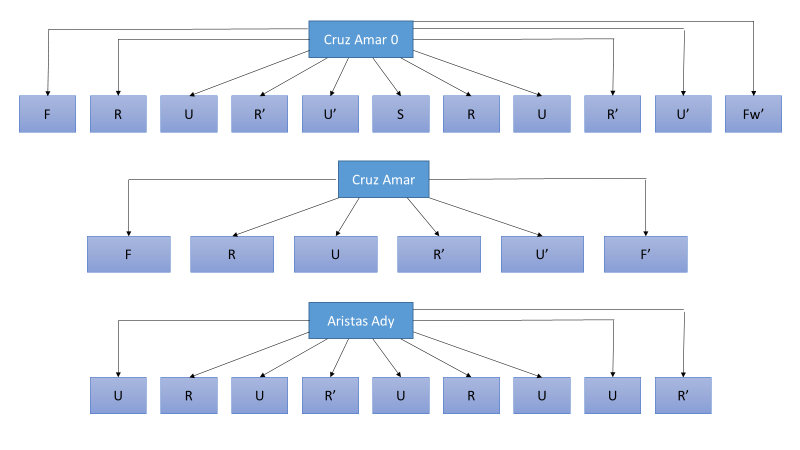
Función L Capa Central:



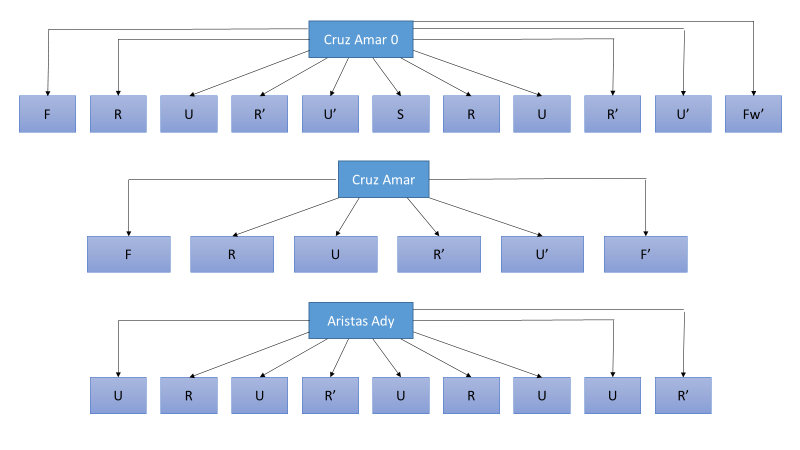
Función R Capa Central:



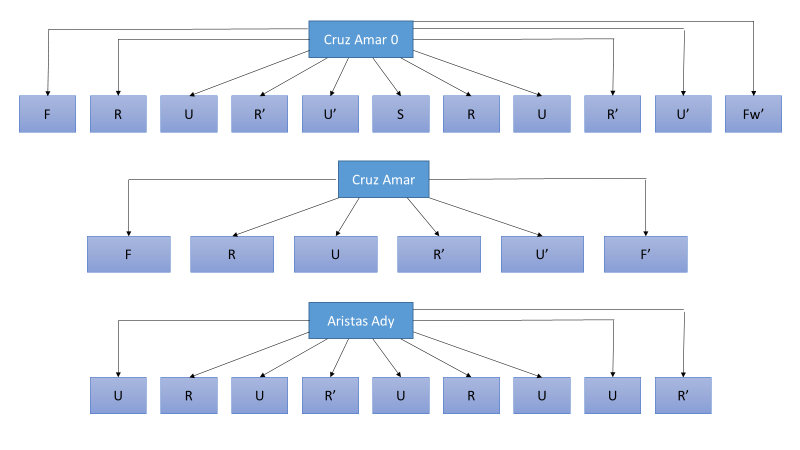
Función Cruz Amar Cero:



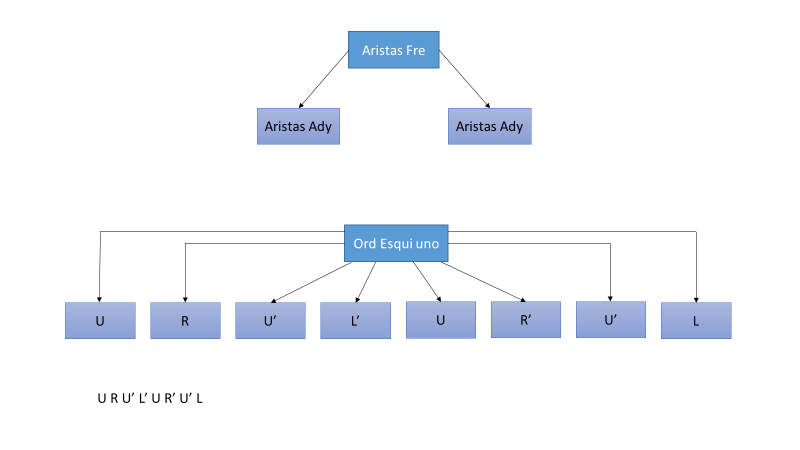
Función Cruz Amar:



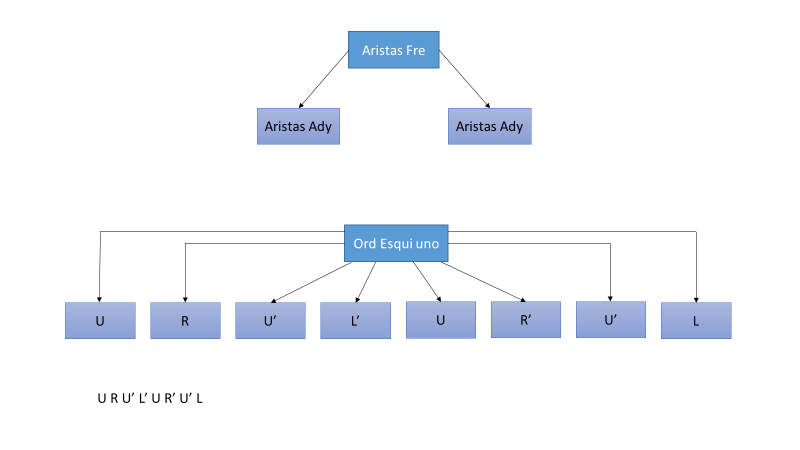
Función Aristas Ady:



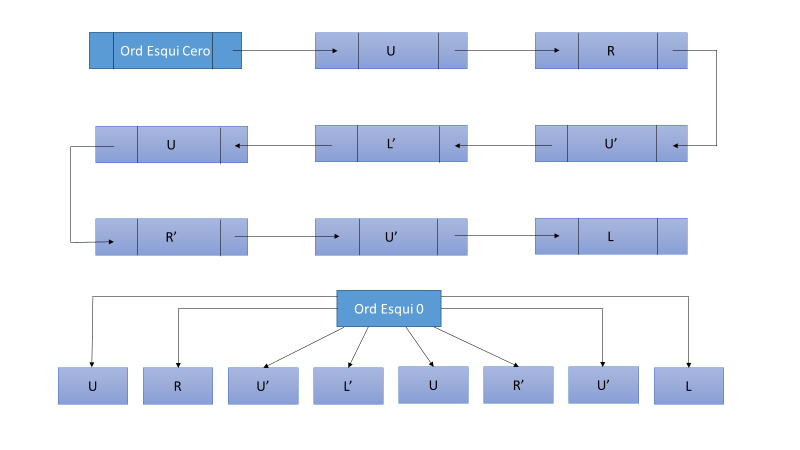
Función Aristas Fre:



Función Ord Esqui Uno:

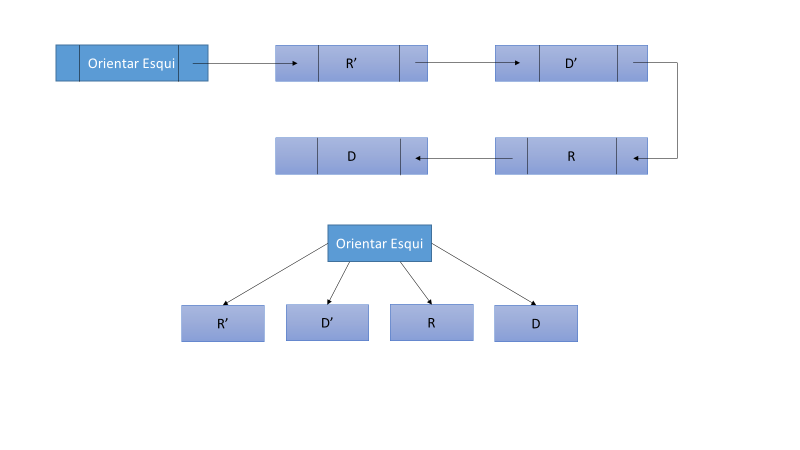


Función Ord Esqui Cero:



\*Nota: Esta Función se repite hasta que se presente la condición específica.

Función Orientar Esqui:



1. **Pruebas**

**7.1 Descripción de las pruebas realizadas**

|  |  |  |
| --- | --- | --- |
| **Versión de Prueba** | **Tipo de Pruebas** | **Descripción Detallada** |
| 1.0 | Aplicación C.A.M | Realizar pruebas de la aplicación C.A.M con todos sus botones totalmente operativos sin presentar algún tipo de error de ejecución. |
| 2.0 | Robot | Realizar pruebas de movimientos de los motores ‘A’ y ‘B’ del robot EV3 donde se realizan los patrones creados los que deben realizarse sin problemas. |
| 3.0 | Conexión Remota | Realizar pruebas donde se ejecuta el archivo **Servidor\_C.A.M.py** que esta almacenado en el robot EV3 y ejecutar el archivo **C.A.M.py** almacenado en un computador y realizar las pruebas que funcione la comunicación entre el **Servidor\_C.A.M** y **C.A.M.py.** |

**7.2 Resultados de las pruebas**

|  |  |  |
| --- | --- | --- |
| **Versión de Prueba** | **Tipo de Pruebas** | **Descripción del Resultado** |
| 1.0 | Aplicación C.A.M | Al realizar todas las pruebas correspondientes no se presentaron problemas con la ejecución del programa. |
| 2.0 | Robot | Se realiza una revisión completa de cada movimiento creado donde no se presentó ningún problema con la ejecución de los patrones. |
| 3.0 | Conexión Remota | Se realizo la prueba de conexión entre el Servidor y el Cliente donde al realizar los llamados de funciones al Servidor no se crea ningún problema de comunicación entre ambos programas. |

1. **Resultados**

**8.1 Estado Final del Proyecto**

El estado del proyecto en el que realizamos la entrega del producto, se encuentra finalizado con los 3 puntos vitales que considera el funcionamiento del robot (movimientos de los motores, Interfaz Gráfica, Conexión Remota) el cual entregamos en la fecha estimada sin ningún inconveniente en su funcionamiento.  
 **8.2 Conclusión**

Con respecto al tiempo transcurrido, se ha logrado cumplir con las actividades que a la fecha están establecidas según la carta Gantt, tales como entablar conexión remota vía rpyc hacia el brick del robot EV3, dejándonos tener control de los motores, permitiendo mover el brazo y la base que sostiene el cubo.No obstante, durante el transcurso del proyecto se presentaron diversos inconvenientes que dificultó nuestro avance. Entre esos problemas se destaca el conflicto que se tuvo al momento de realizar la conexión remota, teniendo que investigar de diversas fuentes para tener una idea más clara.  
Pero a pesar de los tropiezos, se pudo completar la interfaz. La cual cuenta con todas las funciones originalmente pensadas para ésta y viéndose estéticamente atractiva.  
 **8.3 Trabajo Futuro**

Planeamos seguir implementando mejoras tanto visuales, como de funciones/algoritmos que hagan de la experiencia del usuario algo mucho más grato y pueda dejar volar su mente. Buscando así más caminos posibles para completar el armado del cubo Rubik. Además, buscar otros métodos de conexión remota, debido a los problemas que se tuvo implementando rpyc. También diseñar un sistema de puntuación para medir los tiempos y la cantidad de movimientos que les tome a los usuarios armar el cubo, además de poder almacenarlos en una base de datos permitiéndoles competir con las demás personas.

Anexos

Anexo A: Código de los programas implementados  
**Funciones Iniciales**

*def* Cero():

n.run\_to\_rel\_pos(*position\_sp* = 0, *speed\_sp* = 0)

m.run\_to\_rel\_pos(*position\_sp* = 0, *speed\_sp* = 0)

*def* Garra\_Prin():

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(3)

Cero()

**Cero:** Se usa en la finalización de cada programa. Poder mover los motores a voluntad.  
**Garra\_Prin:** Se posiciona la garra para empezar su armado.

*def* A():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

m.run\_to\_rel\_pos(*position\_sp* = -270, *speed\_sp* =500, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

Cero()

*def* Minus\_A():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

m.run\_to\_rel\_pos(*position\_sp* = 270, *speed\_sp*=500, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

Cero()

**A, Minus\_A:** Mover cara inferior del cubo hacia la derecha (A) o izquierda (minus\_A) en 90°.

*def* Double\_A():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

m.run\_to\_rel\_pos(*position\_sp* = -540, *speed\_sp*=500, *stop\_action*="hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

Cero()

*def* Double\_Minus\_A():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

m.run\_to\_rel\_pos(*position\_sp* = 540, *speed\_sp*=500, *stop\_action*="hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

Cero()

**Double\_A, Double\_Minus\_A:** Funciona de igual forma a lo anterior, pero moviéndose en 180°.

*def* H():

m.run\_to\_rel\_pos(*position\_sp* = -270, *speed\_sp*=500, *stop\_action* = "hold")

sleep(2)

Cero()

*def* Minus\_H():

m.run\_to\_rel\_pos(*position\_sp* = 270, *speed\_sp*=500, *stop\_action* = "hold")

sleep(2)

Cero()

**H, Minus\_H:** Mueve el cubo hacia la derecha (H) o izquierda (minus\_H) en 90°.

*def* Double\_H():

m.run\_to\_rel\_pos(*position\_sp* = -540, *speed\_sp*=500, *stop\_action* = "hold")

sleep(2)

Cero()

*def* Double\_Minus\_H():

m.run\_to\_rel\_pos(*position\_sp* = 540, *speed\_sp*=500, *stop\_action* = "hold")

sleep(2)

Cero()

**Double\_H, Double\_Minus\_H:** Similar a lo anterior, pero en 180°.

*def* C():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

Cero()

*def* Double\_C():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(0.5)

Cero()

**C, Double\_C:** Voltear el cubo una (C) y dos (Double\_C) veces

*def* Triple\_C():

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = 85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

n.run\_to\_rel\_pos(*position\_sp* = -85, *speed\_sp* = 300, *stop\_action* = "hold")

sleep(2)

Cero()

**Triple\_C:** Voltear el cubo tres veces.

**Algoritmos Básicos en Robot EV3:**

*def* exposed\_U(*self*):

Funciones.Double\_C()

Funciones.A()

Funciones.Double\_C()

Funciones.Cero()

*def* exposed\_Minus\_U(*self*):

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.Double\_C()

Funciones.Cero()

**U, Minus\_U:** Girar cara superiror del cubo hacia la izquierda (U) y derecha (Minus\_U).

U  U’ 

*def* exposed\_Minus\_F(*self*):

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.Cero()

*def* exposed\_F(*self*):

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.Cero()

**F, Minus\_F:** Girar cara frontal del cubo hacia la izquierda (Minus\_F) y derecha (F).

F  F’ 

*def* exposed\_Minus\_R(*self*):

Funciones.Minus\_H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Cero()

*def* exposed\_R(*self*):

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Cero()

**R, Minus\_R:** Girar cara derecha del cubo hacia arriba (R) y abajo (Minus\_R).

R  R’ 

*def* exposed\_Minus\_L(*self*):

Funciones.H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.Minus\_H()

Funciones.Cero()

*def* exposed\_L(*self*):

Funciones.H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.Minus\_H()

Funciones.Cero()

**L, Minus\_L:** Girar cara izquierda del cubo hacia arriba (Minus\_L) y abajo (L).

L  L’ 

*def* exposed\_Minus\_D(*self*):

Funciones.Minus\_A()

Funciones.Cero()

*def* exposed\_D(*self*):

Funciones.A()

Funciones.Cero()

**D, Minus\_D:** Girar cubo hacia la izquierda (Minus\_D) y derecha (D).

D  D’ 

*def* exposed\_Minus\_B(*self*):

Funciones.Triple\_C()

Funciones.Minus\_A()

Funciones.C()

Funciones.Cero()

*def* exposed\_B(*self*):

Funciones.Triple\_C()

Funciones.A()

Funciones.C()

Funciones.Cero()

**B, Minus\_B:** Girar cara previa del cubo hacia la izquierda (B) y derecha (Minus\_B).

B  B’ 

*def* exposed\_Minus\_Fw(*self*):

Funciones.Double\_H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Minus\_H()

Funciones.C()

Funciones.Cero()

*def* exposed\_Fw(*self*):

Funciones.Double\_H()

Funciones.C()

Funciones.A()

Funciones.H()

Funciones.C()

Funciones.Cero()

**Fw, Minus\_Fw:** Gira la cara frontal y central hacia la izquierda (Minus\_Fw) y derecha (Fw).

*def* exposed\_S(*self*):

Funciones.C()

Funciones.Minus\_A()

Funciones.Double\_C()

Funciones.A()

Funciones.Minus\_H()

Funciones.C()

Funciones.Cero()

*def* exposed\_Minus\_S(*self*):

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.H()

Funciones.C()

Funciones.Cero()

**S, Minus\_S:** Gira la cara vertical central hacia la izquierda (Minus\_S) y derecha (S).

S  S’

**Algoritmos Complejos:**

*def* exposed\_CruzBlanca(*self*):

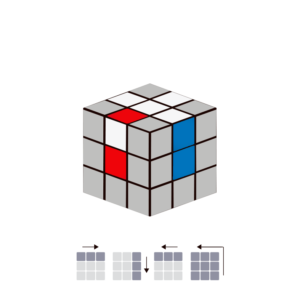
Funciones.Minus\_U()

Funciones.Minus\_R()

Funciones.U()

Funciones.Minus\_F()

**CruzBlanca:** Orientar arista para completar cruz blanca.

****

*def* exposed\_Esquinas(*self*):

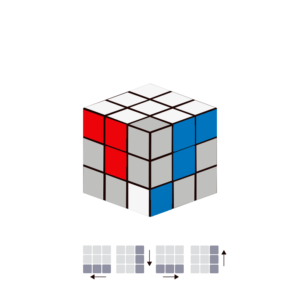
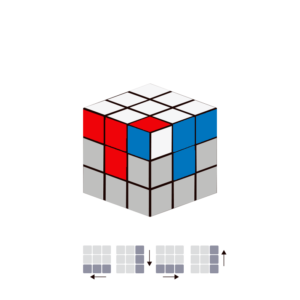
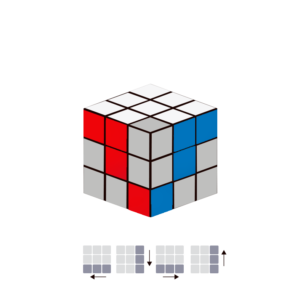
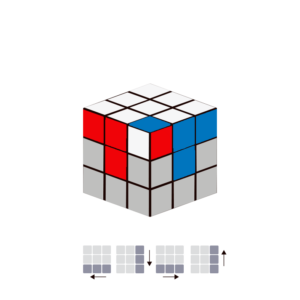
Funciones.Minus\_D()

Funciones.Minus\_R()

Funciones.D()

Funciones.R()

**Esquinas:** Posicionar y/u orientar esquinas (se puede repetir hasta 4 veces según el caso).

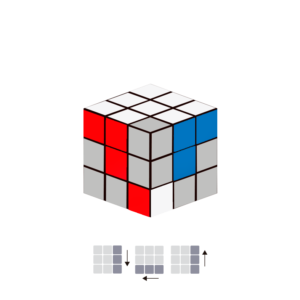
*def* exposed\_Esquinas\_2(*self*):

Funciones.Minus\_R()

Funciones.Minus\_D()

Funciones.R()

**Esquinas 2:** Caso especifico de posicionamiento de esquina.



*def* exposed\_L\_capa\_central(*self*):

Funciones.U()

Funciones.R()

Funciones.Minus\_U()

Funciones.Minus\_R()

Funciones.Minus\_U()

Funciones.Minus\_F()

Funciones.U()

Funciones.F()

*def* exposed\_R\_capa\_central(*self*):

Funciones.Minus\_U()

Funciones.Minus\_L()

Funciones.U()

Funciones.L()

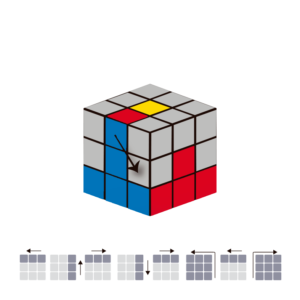
Funciones.U()

Funciones.F()

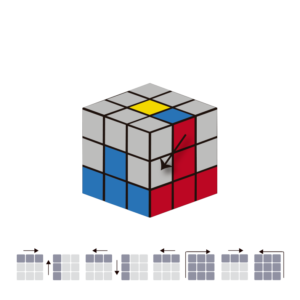
Funciones.Minus\_U()

Funciones.Minus\_F()

**Capa Central:** Para completar la segunda capa, mueve arista del sector amarillo hacia la posicion indicada.



**L:**

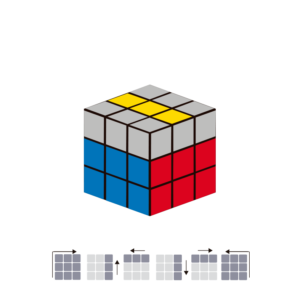
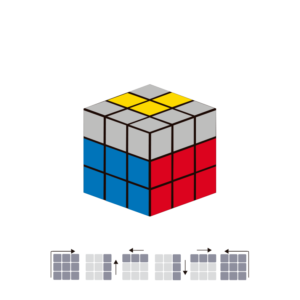


**R:**

*def* exposed\_Cruz\_Amar(*self*):

Funciones.F()  
 Funciones.R()  
 Funciones.U()  
 Funciones.Minus\_R()  
 Funciones.Minus\_U()  
 Funciones.Minus\_F()

**Cruz Amar:** Completa la cruz amarilla según pida lo necesario (1, 2 veces).

1 vez 2 veces

*def* exposed\_Cruz\_Amar\_Cero(*self*):

Funciones.F()

Funciones.R()

Funciones.U()

Funciones.Minus\_R()

Funciones.Minus\_U()

Funciones.S()

Funciones.R()

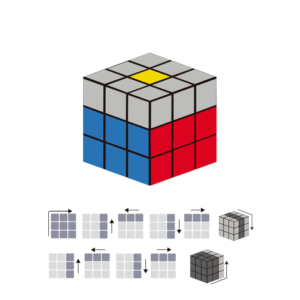
Funciones.U()

Funciones.Minus\_R()

Funciones.Minus\_U()

Funciones.minus\_Fw()

**Cruz Amar Cero:** Caso especial de armar la cruz amarilla cuando solo esté el centro.



*def* exposed\_Aristas\_Ady(*self*):

Funciones.U()

Funciones.R()

Funciones.U()

Funciones.Minus\_R()

Funciones.U()

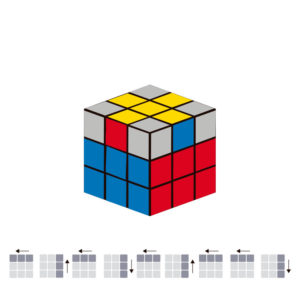
Funciones.R()

Funciones.U()

Funciones.U()

Funciones.Minus\_R()

**Aristas Ady:** Ordena las aristas adyacentes de forma que queden en la posición deseada.

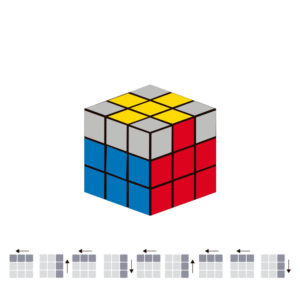


*def* exposed\_Aristas\_Fre(*self*):

Funciones.Aristas\_Ady()

Funciones.Aristas\_Ady()

**Aristas Fre:** Caso especial en que las aristas de en frente están en buena posición.



*def* exposed\_Ord\_Esqui\_uno(*self*):

Funciones.U()

Funciones.R()

Funciones.Minus\_U()

Funciones.Minus\_L()

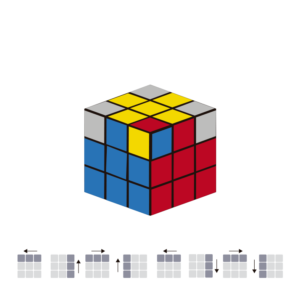
Funciones.U()

Funciones.Minus\_R()

Funciones.Minus\_U()

Funciones.L()

**Ord Esqui uno:** Ordena esquinas de la ultima capa desde la esquina que esté bien ordenada (repetir hasta que todas las esquinas estén ordenadas)



*def* exposed\_Ord\_Esqui\_cero(*self*):

U()

R()

Minus\_U()

Minus\_L()

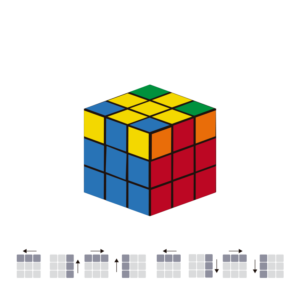
U()

Minus\_R()

Minus\_U()

L()

**Ord Esqui cero:** Cuando ninguna esquina esté ordenada, se usa este algoritmo hasta que salga alguna esquina ordenada.



*def* Orientar\_Esqui(*self*):

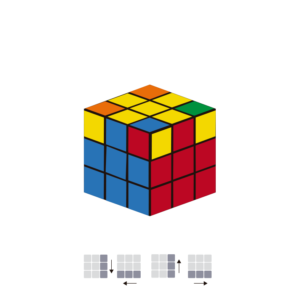
Funciones.Minus\_R()

Funciones.Minus\_D()

Funciones.R()

Funciones.D()

**Orientar Esqui:** Orienta la esquina deseada hasta que esté bien colocada.



*def* exposed\_Sexy\_Move(*self*):

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Double\_C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.Double\_C()

*def* exposed\_Minus\_Sexy\_Move(*self*):

Funciones.H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.Minus\_H()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.Double\_C()

Funciones.H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.Minus\_H()

Funciones.Double\_C()

Funciones.A()

Funciones.Double\_C()

*def* exposed\_Sune(*self*):

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Double\_C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Double\_C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Triple\_C()

Funciones.H()

Funciones.Double\_C()

Funciones.Double\_A()

Funciones.Double\_C()

Funciones.Minus\_H()

Funciones.C()

Funciones.Minus\_A()

Funciones.Triple\_C()

Funciones.H()

*def* exposed\_Seis\_Lunares(*self*):

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.H()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_A()

Funciones.H()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.Minus\_A()

*def* exposed\_Diagonales(*self*):

Funciones.A()

Funciones.Double\_C()

Funciones.A()

Funciones.H()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.A()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.A()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.A()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.A()

Funciones.C()

Funciones.A()

Funciones.Double\_C()

Funciones.A()

*def* exposed\_Dos\_Esquinas\_Cambiadas(*self*):

Funciones.A()

Funciones.C()

Funciones.Double\_A()

Funciones.Triple\_C()

Funciones.Minus\_A()

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Minus\_H()

Funciones.C()

Funciones.Double\_A()

Funciones.Triple\_C()

Funciones.Minus\_A()

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Minus\_H()

Funciones.C()

Funciones.Double\_A()

Funciones.Triple\_C()

Funciones.Minus\_A()

Funciones.Minus\_H()

Funciones.C()

Funciones.A()

Funciones.Minus\_H()

Funciones.C()

Funciones.Double\_A()

Funciones.Triple\_C()

Funciones.Minus\_A()

*def* exposed\_Cubo\_X(*self*):

Funciones.Double\_A()

Funciones.Double\_C()

Funciones.Double\_A()

Funciones.H()

Funciones.C()

Funciones.Double\_A()

Funciones.Double\_C()

Funciones.Double\_A()

Funciones.H()

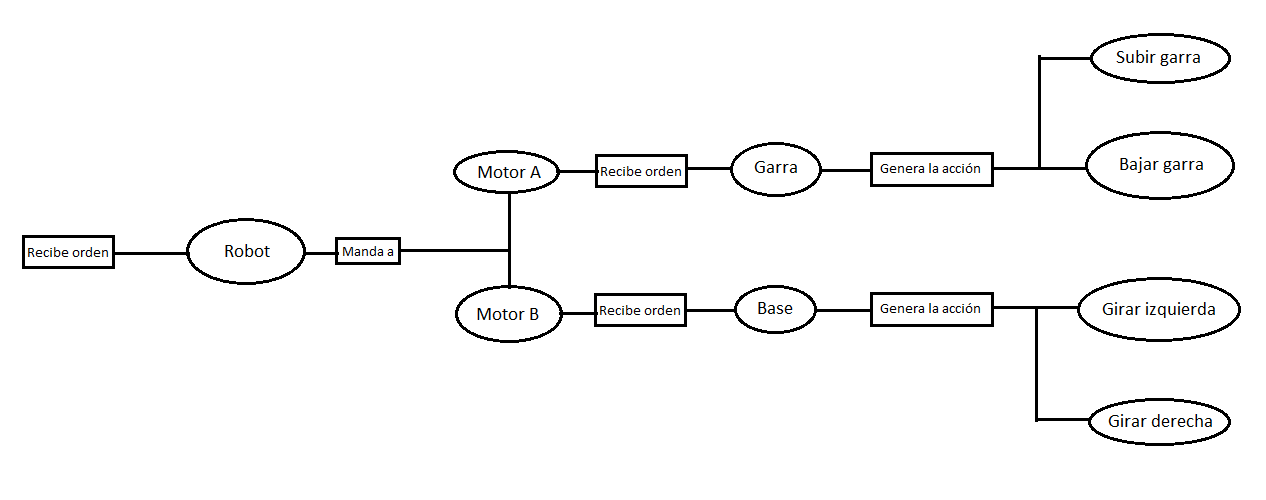
Funciones.C()

Funciones.Double\_A()

Funciones.Double\_C()

Funciones.Double\_A()

Anexo B: Robot (diagrama de construcción, componentes principales)

 Diagrama Construcción