



TECNOLÓGICO DE COSTA RICA

ESCUELA DE INGENIERÍA EN COMPUTACIÓN  
ÁREA ACADÉMICA DE INGENIERÍA EN COMPUTADORES  
LENGUAJES, COMPILADORES E INTÉRPRETES (CE-3104)

Tarea 2  
DrWazeLog

Realizado por:  
OLMAN CASTRO Hernández - 2016003915  
PABLO MORA Rivas - 2017114091  
EDUARDO MOYA BELLO - 2015096664

Profesor:  
Ing. Marco Rivera

Cartago, 28 de mayo de 2019

# Índice

Manual de usuario	3
Introducción	3
<b>Pasos previos antes de la ejecución</b>	<b>3</b>
Cómo ejecutar el programa	5
Funciones utilizadas	5
Estructuras de datos utilizadas	5
Algoritmos detallados	5
Problemas conocidos	5
Actividades realizadas por estudiante	6
Problemas encontrados	7
Conclusiones y Recomendaciones	8
Conclusiones	8
Recomendaciones	8
Referencias bibliográficas	8
Anexos	8
Anexo 1: Bitácora digital	8
Bitácora 1	8
Bitácora 2	9
Fecha: 18/05/2019	9
Bitácora 3	9
Bitácora 4	9
Bitácora 5	10
Bitácora 6	10
Bitácora 7	10
Bitácora 8	10

# Manual de usuario

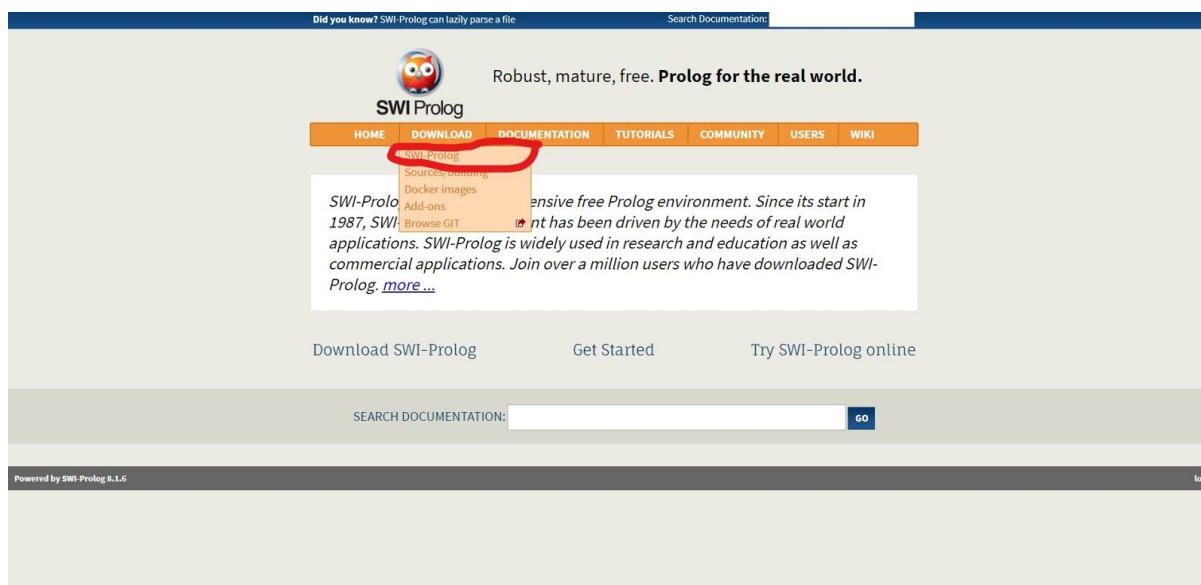
## Introducción

Los sistemas expertos (SE) son aplicaciones de cómputo que involucran experiencia no algorítmica, para resolver cierto tipo de problema. Por ejemplo, los sistemas expertos se usan para el diagnóstico al servicio de humanos y máquinas. Existen SE que juegan ajedrez, que plantean decisiones financieras, que configuran computadoras, que supervisan sistemas de tiempo real, que deciden políticas de seguros, y llevan a cabo demás tareas que requieren de experiencia humana.


La presente tarea tiene como objetivo desarrollar un SE para un sistema de ubicación y guía del tránsito, la interfaz debe ser completamente natural utilizando el lenguaje español. El Usuario ingresa e informa al WazeLog donde se encuentra y a dónde desea llegar.

## Pasos previos antes de la ejecución

Para poder ejecutar el programa en su computador, primeramente debe de cerciorarse de que SWI-Prolog y JAVA se encuentren instalados en la misma, de no ser así, se debe proceder a descargar desde la página oficial (<http://www.swi-prolog.org/>) y seguir los pasos de instalación propuestos por la misma.



News: Ludum Dare Game Jam 44 Search Documentation:

 **SWI-Prolog downloads**

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY USERS WIKI

**Available versions**

The **stable** release is infrequently updated. It is fine for running basic Prolog code without surprises. The **development** version is released roughly every two to four weeks. This is the recommended version for developers and users of applications such as **SWISH** or **ClioPatris**. Finally, the **GIT** and **daily** versions are for developers that want to contribute or have immediate access to patches. These versions are generally fine, but occasionally suffer from regression.

**Stable release**

- [Download the stable release](#)
- [Daily builds for Windows](#)
- [Browse GIT repository](#)

**Read more about**

- [Available SWI-Prolog versions](#)
- [Information on Linux packages and building on Linux](#)

Tags: [Linux](#) [MacOSX](#) [Windows](#) Tag Wiki page "SWI-Prolog downloads"

login to add a new annotation post.


Powered by SWI-Prolog 8.1.6 login


News: Ludum Dare Game Jam 44 Search Documentation:


 **Download SWI-Prolog stable versions**

HOME DOWNLOAD DOCUMENTATION TUTORIALS COMMUNITY USERS WIKI

 Linux versions are often available as a package for your distribution. We collect information about available packages and issues for building on specific distros [here](#). We provide a [PPA for Ubuntu](#).

 Please check the [windows release notes](#) (also in the SWI-Prolog startup menu of your installed version) for details.

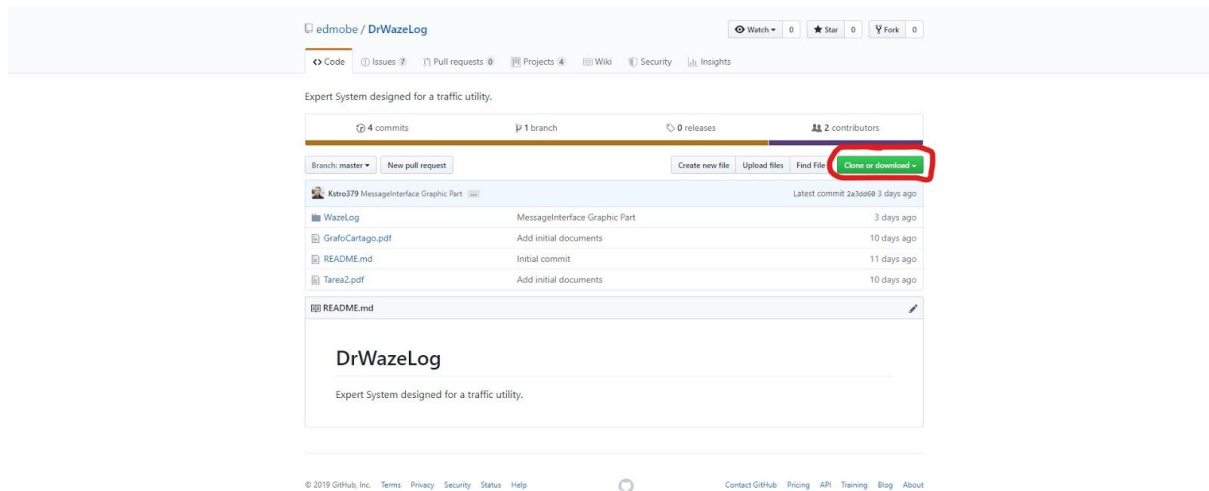
 Examine the [ChangeLog](#).

Binaries	
 12,196,573 bytes	<b>SWI-Prolog 8.0.2-1 for Microsoft Windows (64 bit)</b> Self-installing executable for Microsoft's Windows 64-bit editions. Requires at least Windows 7. See the <a href="#">reference manual</a> for deciding on whether to use the 32- or 64-bits version. This binary is linked against GMP 6.1.1 which is covered by the LGPL license. SHA256: 72a344087370206e051a89e30f07c509393b418c1f6a1e20b81812738756
 11,901,166 bytes	<b>SWI-Prolog 8.0.2-1 for Microsoft Windows (32 bit)</b> Self-installing executable for MS-Windows. Requires at least Windows 7. Installs <b>swipl-win.exe</b> and <b>swipl.exe</b> . This binary is linked against GMP 6.1.1 which is covered by the LGPL license. SHA256: 72a344087370206e051a89e30f07c509393b418c1f6a1e20b81812738756
 34,241,303 bytes	<b>SWI-Prolog 8.0.2-1 for MacOSX 10.12 (Sierra) and later on intel</b> Installer with binaries created using <a href="#">MacPorts</a> . Installs <code>/opt/local/bin/swipl</code> . Needs <a href="#">xquartz</a> (X11) and the Developer Tools (Xcode) installed for running the <a href="#">development tools</a> . SHA256: 34133c086cc08486c08fcb07d5ac70871c9f08a70510b401c0d9edc85af2687
Sources	
 10,413,446 bytes	<b>SWI-Prolog source for 8.0.2</b> Sources in <code>.tar.gz</code> format, including packages and generated documentation files. See <a href="#">build instructions</a> . SHA256: a0081255ac542c90997c0005b1f15c74e0b46638b64e784840a139f0210ba735
Documentation	
 2,505,385 bytes	<b>SWI-Prolog 8.0.2 reference manual in PDF</b> SWI-Prolog reference manual as PDF file. This does not include the <a href="#">package</a> documentation.

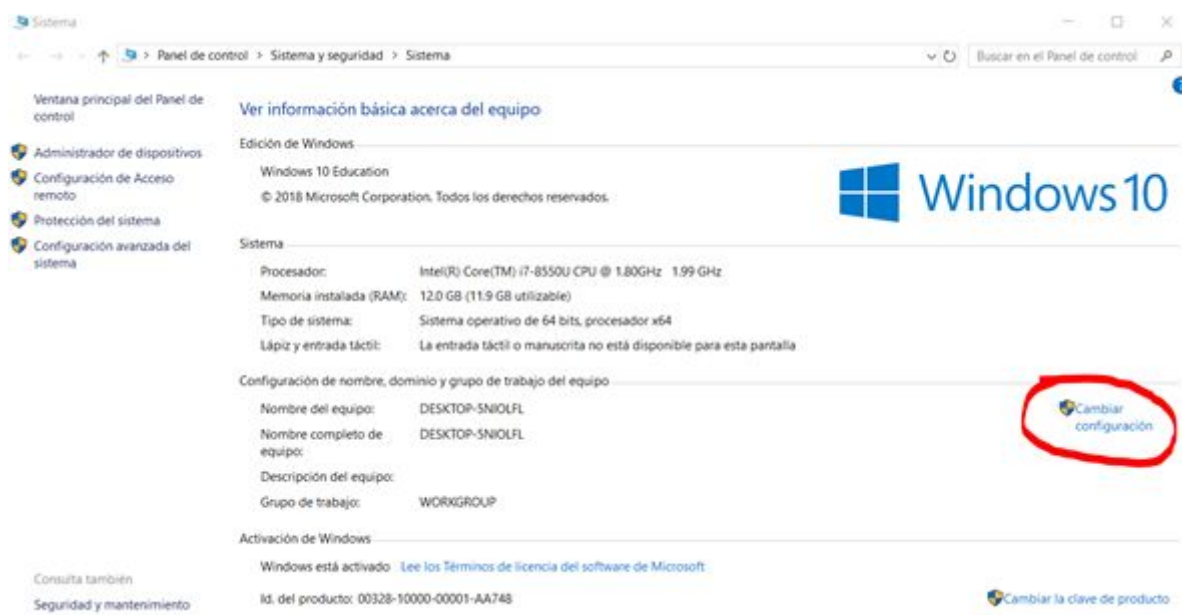
[Show all files](#)

Install scripts may download the SHA256 checksum by appending `.sha256` to the file name. Scripts can download the latest version by the version of the file with `latest`. This causes the cursor to land with the location of the latest version unless an `@@@` is used.

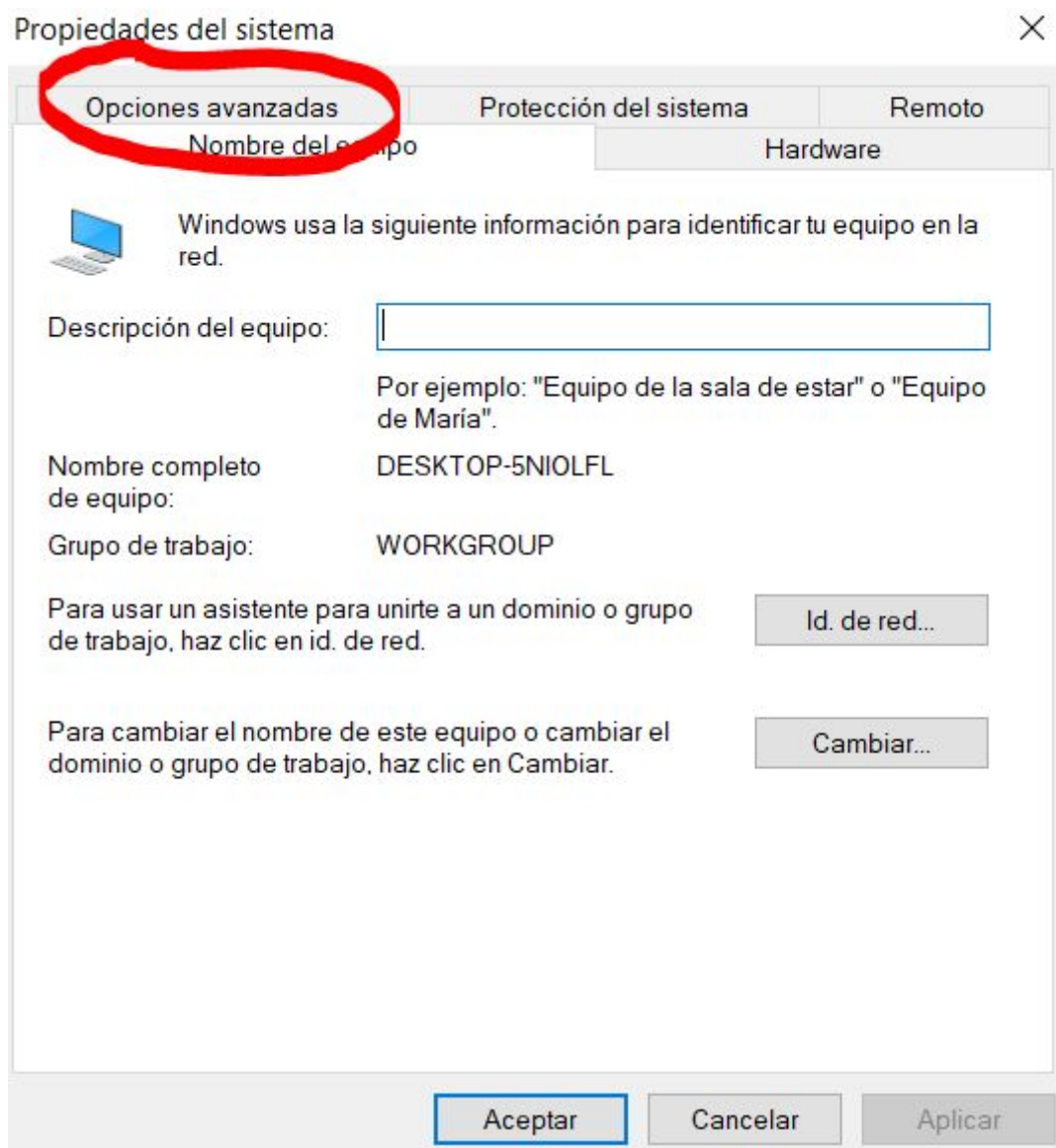
Seguidamente, se debe descargar el repositorio de Github (<https://github.com/edmobee/DrWazeLog>) donde se encuentran almacenados los diferentes archivos ejecutables.



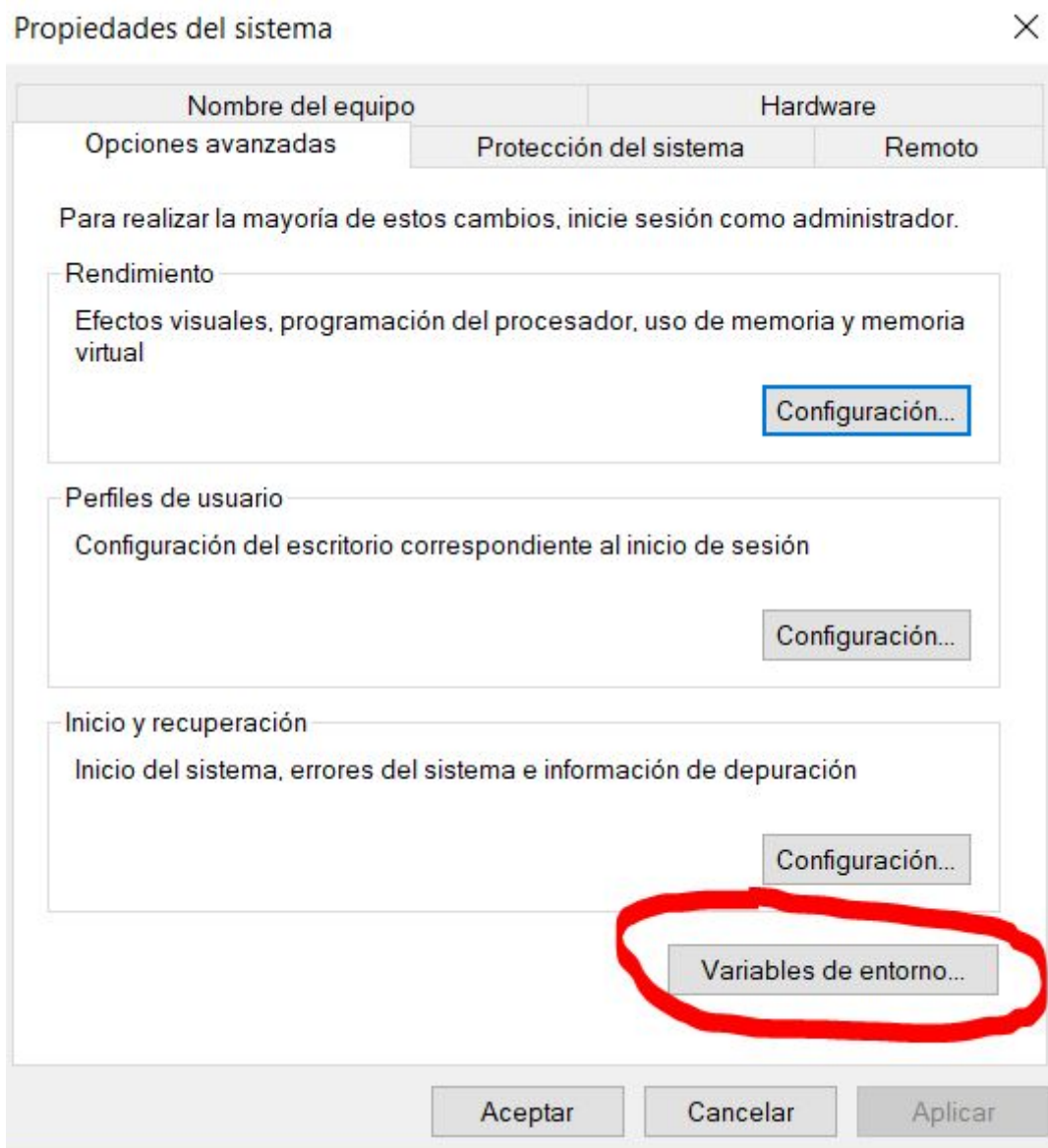
Antes de ejecutar el programa tenemos que hacer unos ajustes en el ordenador. Primero nos tenemos que dirigir al Panel de Control del ordenador, ya en Panel de Control dirigirse a Sistema y Seguridad y seleccionar Sistema.



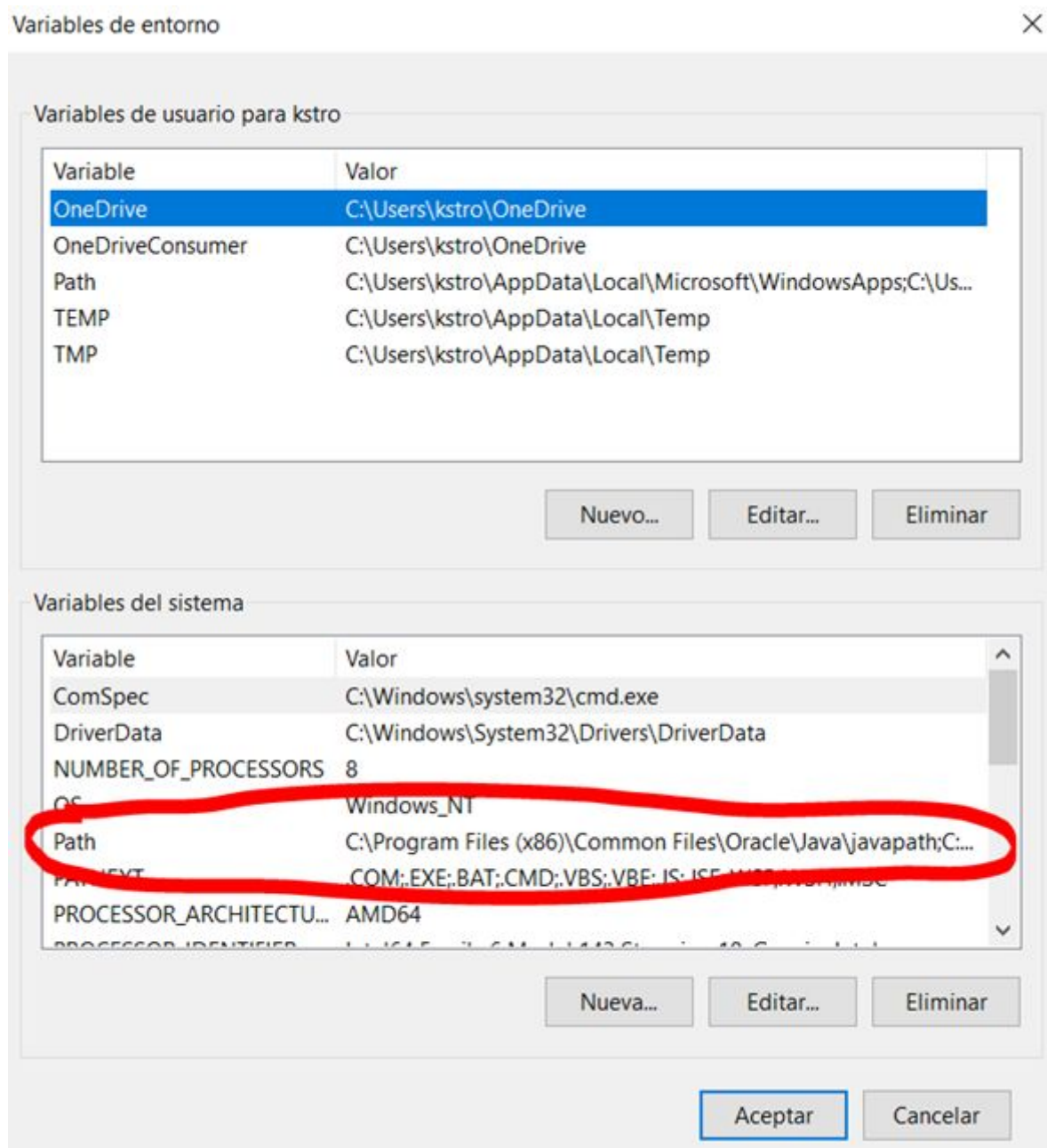
Una vez en Sistema encontrará en el sector derecho de la pantalla la opción Cambiar configuración, daremos click sobre esa opción y se abrirá la siguiente ventana:



En esa ventana, vamos a seleccionar Opciones avanzadas, lo que nos mostrara la siguiente ventana:

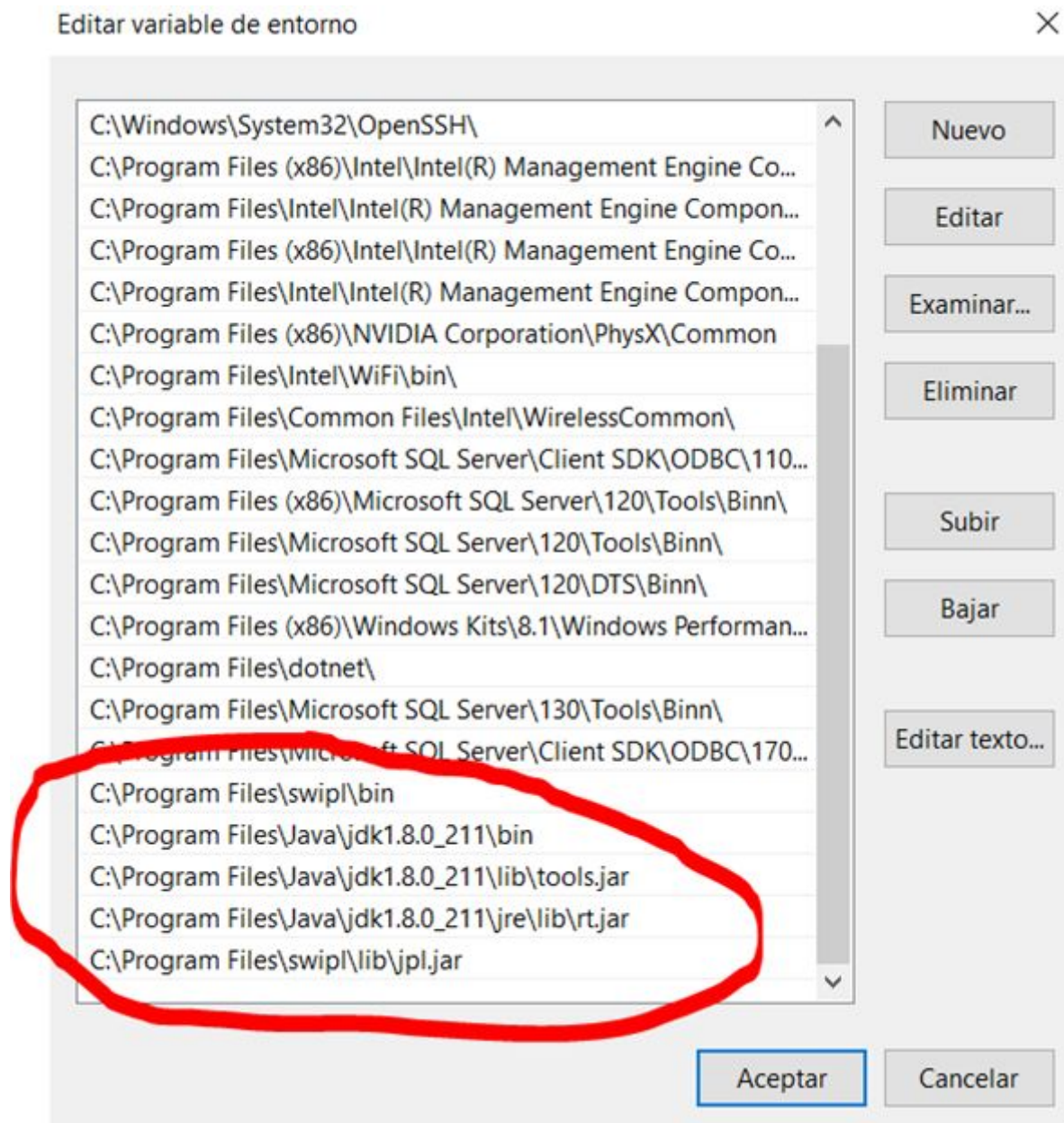


Y vamos a dirigirnos a Variables de entorno que se encuentra en la parte inferior derecha de la ventana. Esto va a desplegar lo siguiente:



En Variables de entorno vamos a seleccionar la variable que se muestra en la imagen, y posteriormente vamos a editar dicha variable haciendo click en el botón editar teniendo la variable seleccionada. Al hacer esto entraremos la siguiente ventana:





En esta parte daremos click en Nuevo, y añadiremos una por una las direcciones que aparecen en rojo. (IMPORTANTE: estas direcciones pueden variar segun la ubicacion en la que se instalo Java en el ordenador, cualquier duda con respecto a estos pasos puede hacerla al correo [kstro379@gmail.com](mailto:kstro379@gmail.com) ).

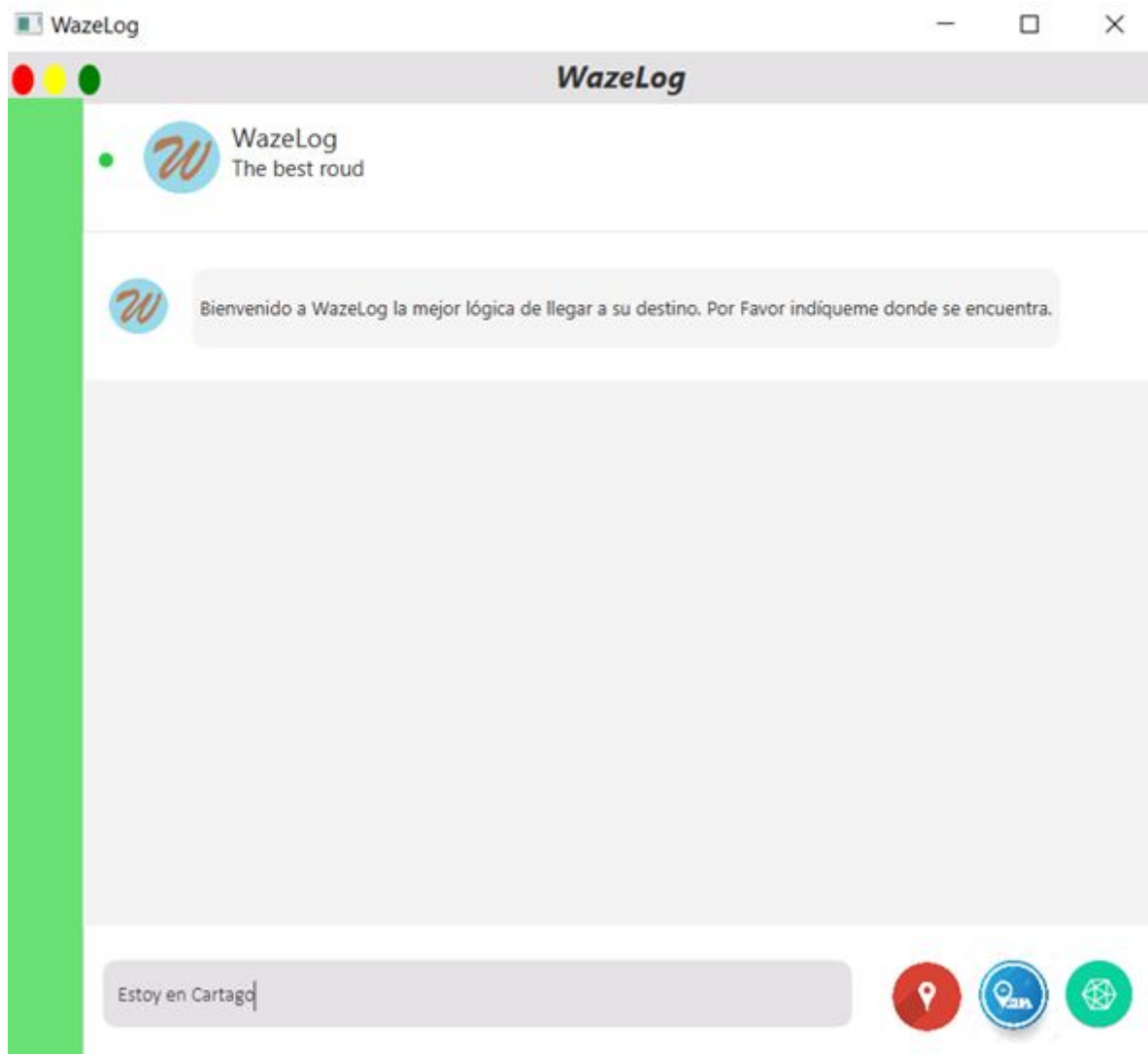
### Cómo ejecutar el programa

Una vez realizados los pasos previos a la ejecución podemos pasar a ejecutar el programa. Para esto diríjase a su IDE Java de su preferencia, en nuestro caso usaremos Eclipse, y habrá el proyecto que descargo de GitHub en la parte de previos a la ejecución. Una vez allí verá un icono verde de play que le permite correr

el programa. (Este icono puede variar según el IDE de Java).

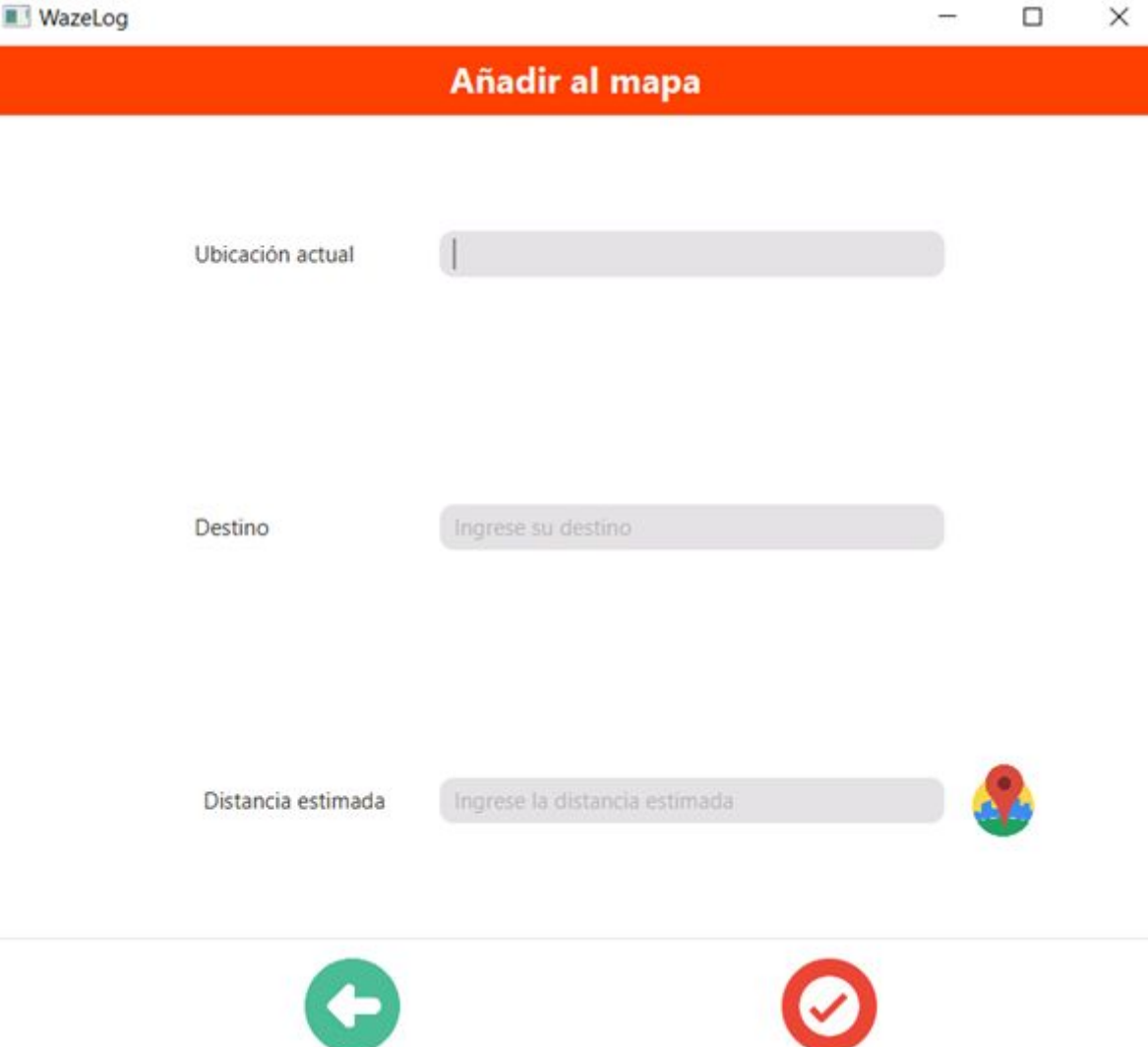


Al ejecutar el programa le aparecerá la siguiente interfaz:



A partir de aquí es cuestión de mantener una conversación con WazeLog, por lo tanto hay que escribir donde se encuentra para iniciar la conversación. Una vez escrito el lugar de partida hay que hacer click en el botón verde que nos permite enviar el mensaje. Al mantener una conversación con WazeLog este nos permitirá encontrar la ruta de un lugar a otro, con la opción de pasar por un lugar intermedio. Es importante aclarar que Wazelog

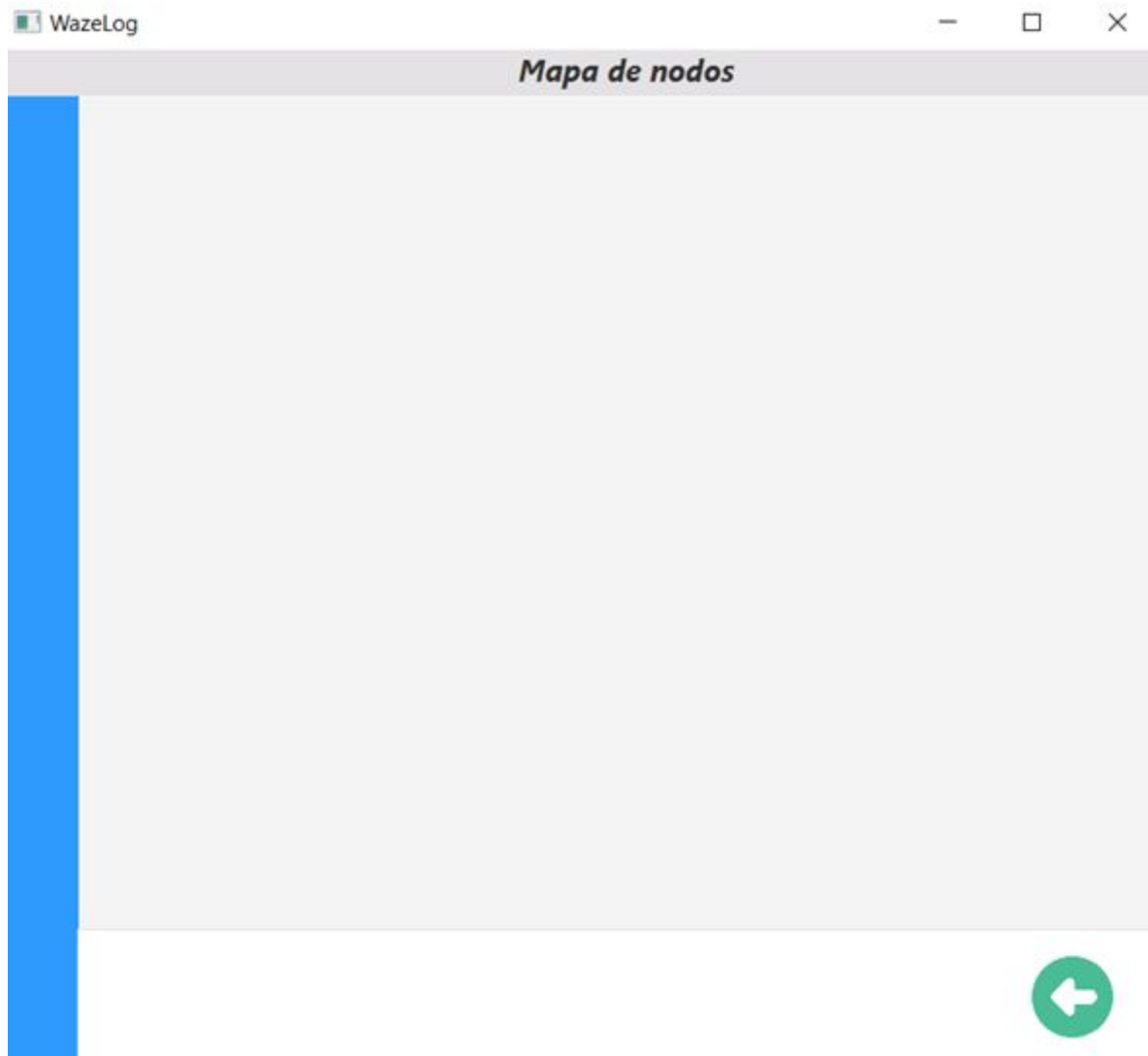
Si desea agregar una nueva ruta, es cuestión de hacer click en el botón rojo, esto nos llevará a la siguiente ventana:



The image shows a web browser window titled 'WazeLog'. The main heading is 'Añadir al mapa' (Add to map) in white text on an orange background. Below this, there are three input fields: 'Ubicación actual' (Current location) with a light gray input box, 'Destino' (Destination) with a light gray input box containing the placeholder text 'Ingrese su destino', and 'Distancia estimada' (Estimated distance) with a light gray input box containing the placeholder text 'Ingrese la distancia estimada'. To the right of the 'Distancia estimada' input box is a Google Maps location pin icon. At the bottom of the form, there are two circular buttons: a green one with a white left-pointing arrow and a red one with a white checkmark.

Aquí hay que llenar la ubicación actual y el destino y tiene la opción para consultar la distancia entre ambos lugares con la API de Google, sin embargo si usted desea agregar manualmente la distancia entre ambos puntos, está en toda la capacidad de hacerlo.(IMPORTANTE: no dejar ningún campo vacío para agregar nuevos lugares de forma exitosa).

Por último, si hace click en el botón azul en la interfaz principal de la conversación con WazeLog, este le mostrará una ventana con una representación gráfica del grafo que se tiene en la base de datos de los lugares. Esta ventana es la siguiente:



## Hechos y reglas implementadas.

- Se implementan los hechos `camino([Origen, Destino, Peso, Id_origen, Id_destino], [])`. Prolog brinda una manera más fácil de visualizar este hecho, de la siguiente manera: `camino --> [Origen, Destino, Peso, Id_origen, Id_destino]`. Estos son utilizados para la elaboración del grafo en Prolog, es decir cada hecho `camino` permite agregar nodos al grafo que en realidad es la totalidad de los hechos `camino`, la variable `Origen` dice el punto de salida, la variable `Destino` da el punto de llegada, la variable `Peso` indica la arista entre ambos nodos del grafo y los `IDs` son identificadores únicos que maneja Google para hacer referencia a un lugar. Con este ID, es posible .
  - Ejemplo: `camino --> [[cartago], [san, jose], 25, ["ChIJISmvIRDFol8RUpyjWIVU5bU"]]`,

`["ChIJxRUNxULjol8RgrgRn2pqdOY"]`. → esto significa que existe una ruta entre Cartago y san José y que entre ambos puntos hay 25 km.

- Se define la regla `camino1(Inicio, Destino, Costo,[Inicio, Destino], _)`. Esta regla nos permite identificar el camino entre dos nodos del grafo y calcular la distancia que hay entre ellos. En caso de que no encuentre un camino directo entre Inicio y Destino, la regla verifica si existe `camino1(Inicio, Destino, Costo, [Inicio|P], V)`. Esta segunda parte de la regla nos permite hacer recursividad para verificar si existe alguna ruta entre Inicio y Destino que pase por algunos nodos intermedios. Asimismo verifica durante la recursividad que no se creen bucles entre los nodos con ayuda de `\+member(Inicio, V)`, y al igual que el primer caso de `camino1` va sumando el costo de todo el camino entre ambos nodos.
- Se define una segunda regla que llamamos `mejorcamino`, esta es una regla dinámica por lo tanto se incluye `:-dynamic(solution/2)`. Decimos que la regla es dinámica pues contiene una variable solución que es una variable dinámica que se irá actualizando durante su ejecución. La regla `mejorcamino` busca realizar un recorrido por todas las posibles rutas entre dos nodos y encontrar el camino que contiene el menor peso, la variable solución es dinámica pues guarda el peso y la ruta del mejor camino, en caso de encontrar un mejor camino la variable solución se actualiza.

## Estructuras de datos utilizadas

- Se utilizaron listas en Prolog para el desarrollo de la regla para encontrar el mejor camino, pues todo el recorrido se almacena en una lista. En caso de encontrar un mejor camino la lista se actualiza, se planificó solo para guardar la mejor ruta pues sino se tendría que guardar una lista de listas de todas las rutas y luego comparar en cada cada ruta cual es el camino más corto, siendo esta segunda opción menos eficiente, decidimos almacenar sola la mejor ruta en una sola lista. Asimismo se usaron listas en Java para almacenar puntos intermedios.

- También se utilizaron grafos para el desarrollo de la aplicación, en este caso se crea en grafo en Prolog, en este paradigma el grafo se representa con una base de datos de hechos.

## Algoritmos detallados

- Concatenar(L1,L2,L): Este pequeño algoritmo nos permite unir dos listas en Prolog.
- Camino: Este algoritmo nos permite encontrar la primera ruta entre un punto y otro, retorna el primer camino que encuentra entre ambos puntos con los intermedios entre ellos en caso de que existan y el costo de ese camino.
- Mejorcamino: Este algoritmo hace uso del algoritmo Camino, para recorrer todos los posibles caminos entre dos puntos y encontrar el camino con el menor costo. Igualmente devuelve la ruta en una lista y el costo de esa ruta.
- Remove\_elements(El,List,Output): Este algoritmo elimina un elemento de una lista.
- Gramatica\_válida: Este algoritmo nos permite identificar la gramática de una frase, es decir puede identificar orígenes o destinos, asimismo al identificar eso, analiza si la frase tiene sentido para Prolog.
- Pregunta(Contexto,Pregunta): Este algoritmo brinda una Pregunta según sea el Contexto de la frase que identifica.
- Respuesta({contexto},Oracion,{respuesta}): Este algoritmo genera una respuesta según el contexto de lo que escribe el usuario.
- Añadir: Este algoritmo nos permite añadir nuevos hechos a la base de datos. Para esto se lee el archivo que contiene todos los hechos, y inserta todos los hechos de ese archivo a una lista, posteriormente añade a la lista el nuevo hecho, y por ultimo borra todo dato en el archivo de la base de datos y escribe todo lo de la lista en el archivo.
- API: Este algoritmo nos permite extraer información de la API de Google Maps para buscar la distancia que existe entre dos lugares y usar esa información al añadir un nuevo nodo.

## Problemas conocidos

- Se intentó como extra la inclusión de una representación del grafo con rutas en la interfaz, sin embargo decidimos pulir algunos detalles de otras partes del proyecto y no tuvimos tiempo de terminar este extra.

## Actividades realizadas por estudiante

Fecha	Horas	Descripción de actividad	Estudiante
18/05/2019	3 horas	Investigar sobre elaboración de bases de datos en Prolog y el manejo de grafos.	Pablo Mora
18/05/2019	10 horas	Investigación sobre comunicación entre Java y Prolog.	Olman Castro
18/05/2019	3 horas	Investigación sobre gramáticas libres de contexto.	Eduardo Moya
20/05/2019	3 horas	Elaboración de la base de datos en Prolog, así como la creación del grafo.	Pablo Mora.
22/05/2019 24/05/2019	20 horas	Definición de las reglas para poder identificar las oraciones, sintagmas nominales y sintagmas verbales para lograr descomponerlos	Eduardo Moya

		en palabras e identificar las palabras claves	
21/05/2019	10 horas	Creación de interfaz gráfica utilizando Java que realice una liga con el código de Prolog el cual será el motor de la aplicación.	Olman Castro
22/05/2019	6 horas	Crear las funciones en Prolog que nos permiten encontrar caminos de un punto a otro, así como la creación de una función que nos permite encontrar la mejor ruta entre dos puntos.	Pablo Mora
24/05/2019	3 hora y 30 min	Unión de las dos primeras partes del proyecto	Pablo Mora Eduardo Moya
26/05/2019	3 horas	Unión de Prolog con Java	Eduardo Moya Olman castro
27/05/2019	3 horas	Agregar nodos nuevos desde Java a la base de datos en Prolog	Pablo Mora Eduardo Moya Olman Castro



# Problemas encontrados

- Después de investigar sobre la estructura de las oraciones para el análisis léxico, no sé exactamente cómo manejar las preguntas que hará el usuario, ya que puede tener una amplia variedad de estructuras. Este problema se resolverá con la ayuda del profesor.
- Se encontraron algunas dificultades para la elaboración de la mejor ruta de un punto a otro, pues la lista de rutas no se actualizaba de la manera correcta. Este problema se soluciona con ayuda de la opción `dynamic` de Prolog, la misma nos permite crear variables dinámicas que se pueden actualizar durante la ejecución del programa. Se debió preguntar al profesor si se podía utilizar esta función.

# Conclusiones y Recomendaciones

## Conclusiones

- La aplicación se creó exitosamente, con un buen trabajo en equipo y buena planificación. Se hizo una buena distribución de trabajo, lo que permitió reafirmar y aumentar nuestro conocimiento tanto en Prolog como en Java, haciendo uso de ellos por separado y unidos para el buen funcionamiento del programa.
- Se puede considerar que el sistema es experto en la búsqueda de la mejor ruta y tiene la capacidad de aumentar su base de conocimiento para ser más robusto y completo.
- Se hizo uso de Prolog lo que nos permitió aplicar conceptos básicos y avanzados de programación lógica.
- El uso de listas tanto con Java como en Prolog nos permitió manejar datos eficientemente.

## Recomendaciones

A los desarrolladores

- Conocer las necesidades de sus compañeros de equipo para garantizar un correcto diseño de los algoritmos. Por ejemplo, al unir Java con Prolog, que desde Java solo tenga que solicitar lo que realmente se necesita, para garantizar una mejor organización del código.
- Comunicar mejor sus problemas a los compañeros, ya que especialmente por la naturaleza del paradigma de programación orientado a objetos, existe una abstracción al realizar el código, la cual puede llevar a distintas maneras de diseñar una solución.

## Referencias bibliográficas

- <https://personal.us.es/fsoler/papers/05capgram.pdf>
- <http://dit.upm.es/~gfer/ssii/rcsi/rcsisu69.html#x125-144000A.2.4>
- <https://programacionlogica.blogspot.com/2006/03/>
- <https://www.inmsol.com/es/gramatica-espanola/verbos-transitivos-e-intransitivos/>
- <http://www.rae.es/>
- <https://www.cs.us.es/cursos/ia2-2004/temas/tema-03.pdf>
- <https://sintaxis.org/analizador/>
- <https://www.youtube.com/playlist?list=PLHNkID2PAnJmoXIM0MtgMmNVpkarf2Cd4>

## Anexos

### Anexo 1: Bitácora digital

#### Bitácora 1

Fecha: 17/05/2019

Participante(s): Olman Castro, Pablo Mora, Eduardo Moya

Asunto: Planificación del proyecto

Descripción detallada: Nos reunimos el viernes 17 de mayo por la tarde con el fin de planificar el proyecto y hacer una división de tarea equitativa. Se aprovecha para generar un repositorio en GitHub y se da acceso a todos los miembros del grupo. Asimismo especificamos las bases con las que iríamos a trabajar para evitar confusiones o un mal funcionamiento al unir las partes de cada integrante.

## Bitácora 2

**Fecha: 18/05/2019**

**Participante(s):** Olman Castro, Pablo Mora, Eduardo Moya

**Asunto:** Investigación sobre los temas asignados

**Descripción detallada:** Cada integrante del grupo realiza por su cuenta la investigación pertinente al trabajo que se le asignó el día anterior en la división de trabajo. Pablo investiga sobre bases de datos y grafos en Prolog, Olman sobre conexión entre Java y Prolog, y Eduardo sobre reglas para identificar oraciones.

## Bitácora 3

**Fecha: 20/05/2019**

**Participante(s):** Pablo Mora

**Asunto:** Implementación de la base de datos y grafo en Prolog

**Descripción detallada:** Se inicia por volver a ver los videos que se clasificaron como útiles para la elaboración de la base de datos y el manejo de grafos en Prolog. Se procede a crear la base de datos y se verifican enlaces para completar dicha base de datos. Se realizan pruebas para verificar el funcionamiento de la misma.

## Bitácora 4

**Fecha: 22/05/2019**

**Participante(s):** Eduardo Moya

**Asunto:** Analisis lexico

**Descripción detallada:** Comenzar con la elaboración del análisis léxico, nos permite que Prolog puede asignar un género y un número. Todo el código fue adaptado a la nueva sintaxis encontrada en los links utilizados. También empecé desarrollar para hacer nuevas gramáticas libres de contexto para casos específicos.

## Bitácora 5

**Fecha: 22/05/2019**

**Participante(s):** Olman Castro

**Asunto:** Implementar GUI en Java

Descripción detallada: Se procede a hacer el link entre Java y Prolog, así como la interfaz que permite agregar un nuevo destino y mantener una conversación entre DrWazeLog y el usuario.

## **Bitácora 6**

Fecha: 24/05/2019

Participante(s): Pablo Mora, Eduardo Moya

Asunto: Unificar la base de datos con el análisis lexico

Descripción detallada: Se realiza una reunión, con el fin unir la base de hechos de Prolog con el análisis léxico y así mantener una conversación. Se encuentran algunas cosas a corregir en ambas partes para el buen funcionamiento del programa, por lo tanto se corrige lo que se puede durante la reunión y lo faltante se resuelve por la noche manteniendo una comunicación.

## **Bitácora 7**

Fecha: 26/05/2019

Participante(s): Olman Castro, Eduardo Moya

Asunto: Unificar Prolog con Java

Descripción detallada: Se realiza una video llamada, con el fin de unificar la parte de prolog anteriormente unificada y el GUI de Java. Se encuentran detalles a modificar y se procede a explicar para corregir y unir luego.

## **Bitácora 8**

Fecha: 28/05/2019

Participante(s): Olman Castro, Eduardo Moya

Asunto: Unificar Prolog con Java

Descripción detallada: Ya con ambas partes resueltas, se decide unificar las dos partes exitosamente. Y se habla de la inclusión de la API de Google Maps para extraer información sobre distancia entre dos puntos y añadirlo al proyecto. También se habla de la elaboración de un grafo en pantalla para seleccionar dos puntos y mostrar la ruta.

