Tesina de grado de Lic. en Sistemas

Rita en Red

Una extensión de RITA que facilita las competencias en ROBOCODE

Alumnos:
Maximiliano Rugna
Pablo Vilaltella

Directoras:
Lic. Claudia Queiruga
Lic. Claudia Banchoff Tzancoff

Universidad Nacional de la Plata
Facultad de Informática
La Plata
2015

Índice de contenido

Índice de figuras

1. Objetivos

RITA (Robot Inventor to Teach Algorithms) es un dispositivo educativo destinado a enseñar a programar a estudiantes de escuelas secundarias basado en la creación de videojuegos de robots virtuales. RITA es un desarrollo de la Facultad de Informática iniciado en el año 2012 a partir del trabajo de grado de Licenciatura en Informática de la Lic. Vanessa Aybar Rosales (Aybar Rosales V. et al, 2012) y que ha ido evolucionando a partir de los aportes surgidos de su uso en el aula de la escuela. Desde el punto de vista técnico, RITA es una aplicación Java (Eckel B. et al, 2003) que integra los frameworks de código abierto, **OpenBlocks**[1] y **Robocode**[2] y cuya licencia es libre. Actualmente RITA está disponible en el repositorio GitHub (https://github.com/vaybar/RITA) bajo licencia GNU GPL versión 3 en adelante.

RITA constituye un aporte a una línea de investigación y extensión del LINTI vinculada a acercar la enseñanza de la programación a la escuela secundaria y que se ha puesto en práctica en diversas escuelas secundarias de la región a través de los proyectos de extensión de la Facultad de Informática: *"Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica. Conectar Saberes*" y "Extensión en Vínculo con Escuelas Secundarias".

El objetivo general de nuestra tesina de grado es mejorar la experiencia del usuario con RITA al incorporar la posibilidad de crear un juego de competencias (batallas) de robots en red.

De acuerdo a este objetivo general, se plantean los siguientes objetivos específicos relacionados a aspectos tecnológicos y pedagógicos:

[1] Massachusetts Institute of Technology. Openblocks: Librería Java open-source para crear bloques de programación. Recuperado en Agosto 2014 de http://education.mit.edu/about-old/open-blocks-download-page/

[2] Mathew Nelson, Flemming N. Larsen. Robocode: Juego educativo open-source para la creación de estrategias para controlar robots en un campo de batalla. Recuperado en Marzo 2014 de http://robocode.sourceforge.net/

1. Objetivos tecnológicos:
    a. Extender la funcionalidad de RITA para permitir la ejecución de juegos de robots en la red local o intranet de la escuela.
    b. Incorporar a RITA una interfaz de usuario gráfica simple y amigable que facilite la configuración y ejecución de las competencias de robots en red.

2. Objetivos pedagógicos:
    a. Estimular entre los estudiantes la creación de juegos de robots virtuales en forma colaborativa y el análisis de las estrategias desarrolladas.
    b. Facilitar la tarea docente permitiendo proponer, crear y controlar los juegos que los estudiantes construyen.

2. Agradecimientos

Queremos agradecer a todas las personas que colaboraron en la realización de nuestra tesina. A nuestras directoras Lic. Claudia Queiruga y Lic. Claudia Banchoff Tzancoff que aceptaron dirigir y ayudar en esta tesina; a todo el equipo del proyecto de extensión de la Facultad de Informática: *"Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica. Conectar Saberes*"[3] que participó con muy buena predisposición en la organización de los encuentros con docentes y alumnos de las Escuela Secundaria Media Nº 12 "Manuel B. Gonnet" en la que se evaluó **RITA en Red**, entre ellos a Vanessa Aybar Rosales, Matías Brown Barnetche e Isabel Kimura por su colaboración en las pruebas de campo en la escuela y a todo el personal de la escuela Nº12 que nos ayudó, sobre todo a los alumnos que participaron del taller y nos ayudaron a experimentar con **RITA en Red** con mucha atención y respeto al trabajo realizado.

Por último queremos agradecer en forma particular cada uno, en el caso de Maximiliano Rugna quiero agradecer especialmente a mis viejos Estela y Daniel por el apoyo incondicional en tantos años, a toda mi familia Antonella, Daniel, Pamela e Indiana; en especial a mi compañera de años Estefanía y por último agradezco a la flia. Vilaltella que me ayudó mucho en mi paso por la ciudad de La Plata.

En el caso de Pablo Vilaltella quiero agradecer a mis padres Roberto y Susana que siempre me apoyaron en todos los aspectos de la vida al igual que mis hermanos Paula, Pedro y Laura y mi sobrina Josefina. A mi novia Vanesa que llevamos un largo camino juntos y mucho más por recorrer, a mis compañeros de estudio de la facultad que siempre nos reunimos a estudiar, Fernando, Leandro, Eduardo, Diego, Anselmo y a los padres de Maximiliano que nos apoyaron para terminar la carrera. A todos mis amigos de siempre y también agradezco en particular a Maximiliano que hicimos casi toda la carrera juntos y esta tesina final también.

Para terminar ambos queremos hacer un agradecimiento en conjunto a la Facultad de Informática, UNLP que nos permitió estudiar y formarnos como profesionales y sobre todo como buenos ciudadanos valorando la importancia de la educación pública y gratuita que nos brinda nuestra nación con el aporte del todo el pueblo argentino.

3. Motivación

RITA fue utilizada con muy buenos resultados en diversas escuelas de la región en las que se trabajó con el proyecto de extensión "**Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica. Conectar Saberes"**[3]. Las experiencias con RITA se desarrollaron en los laboratorios de computación de las escuelas y en la Facultad de Informática de la UNLP, utilizando PCs y las netbooks del programa nacional "**Conectar igualdad**"[4]. Cada uno de los alumnos que participaron de las experiencias desarrollaron en forma individual robots virtuales con estrategias de combate que respetan consignas de tiempo y consideraciones propias de cada grupo seleccionado para la experiencia. Luego los robots construidos son puestos a competir en un campo de batalla unificado, conformando un juego de robots virtuales en la computadora del docente instructor. Esta experiencia dejó planteada la problemática de que para poner a competir robots en el aula se requiere la intervención manual del docente instructor de la actividad. No se aprovechan las conexiones disponibles en las aulas ni la posibilidad de recrear en forma automática las batallas construidas.

Teniendo en cuenta que los lineamientos del trabajo de grado de Vanessa Aybar Rosales y los trabajos futuros planteados en el mismo, junto con la recolección de información obtenida de la interacción entre alumnos y docentes en el trabajo de campo, motivó la formulación de esta tesina de grado, **RITA en red.**

Contar con una herramienta didáctica innovadora que permita acercar a los estudiantes y docentes de la escuela secundaria a la "programación" desde un enfoque lúdico y social, nos impulsa a extender RITA incorporándole funcionalidades que permitan que las competencias entre los robots construidos en el aula puedan llevarse a cabo de una manera simple, automática, aprovechando las conexiones de red del aula y que de esta manera las estrategias desarrolladas por los estudiantes puedan ponerse en juego en el campo de batalla.

A través de la implementación de **RITA en red** se logrará disponer de una herramienta didáctica que mediante simples configuraciones permita que las estrategias de los robots construidos por los estudiantes, puedan ponerse en

[3] UNLP - Facultad de Informática - LINTI (2012 - Actualidad). JETs Java en escuelas técnicas. "Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica. Conectar Saberes". La Plata, Buenos Aires, Argentina. Recuperado de: http://jets.linti.unlp.edu.ar

competencia en el campo de batalla, promoviendo el interés, la interacción y colaboración en el grupo de estudiantes.

4. Introducción

Los cambios vertiginosos del mundo actual requieren no sólo de conocimiento sino capacidad y habilidades para poder pensar y actuar creativamente ante problemas inesperados. Actualmente estamos transitando lo que se identifica como la “sociedad de la creatividad”. La creatividad tiene que ver con el desarrollo de la capacidad para: cuestionar, hacer conexiones, innovar, resolver problemas y reflexionar críticamente (Resnick M., 2008).

Los niños y jóvenes actualmente están familiarizados con el uso de la tecnología como el envío de mensajes de textos, juegos en línea, navegar internet, etc. Este uso se realiza como consumidores de la misma, el desafío es que sean ellos mismos los capaces de construir sus propios juegos, animaciones y simulaciones; esto se logra a través de la programación de computadoras (Díaz J. et al, 2014).

Aprender a programar puede ser una ardua tarea para alumnos de la enseñanza inicial y media, ellos además de aprender algún lenguaje de programación con su sintaxis y semántica particular, algunas veces confusa y no intuitiva, deben aprender a estructurar su pensamiento y soluciones para comprender la ejecución de sus programas. Sin embargo, una vez superados estos obstáculos, la programación puede ser una habilidad muy útil y beneficiosa. El inventor de la World Wide Web, Tim Berners-Lee, nos habla de la nueva brecha digital como “una distancia menos conocida es la que separa a aquellos que pueden programar y aquellos cuyas habilidades informáticas se limitan a saber cómo trabajar con aplicaciones estándar, como procesadores de texto y hojas de cálculo” (Tim Berners-Lee, 2013).

Las ventajas de aprender a programar incluyen pero no se limitan a:

- Aprender a resolver problemas de una manera estructurada y lógica.
- La construcción de software personalizado para uso personal.
- Exploración de temas tales como el estudio de las simulaciones biológicas o crear animaciones interactivas.

En la educación secundaria, la programación de computadoras, de acuerdo a Tim Berners-Lee, es "un medio ideal para que los estudiantes expresen la creatividad, desarrollen nuevas ideas, aprendan a comunicar sus ideas, y colaboren con los demás".

Según Jeannette Wing, “El pensamiento computacional es una habilidad fundamental para todos, no sólo para los científicos de la computación. Además de la lectura, escritura y la aritmética, hay que añadir pensamiento computacional a la capacidad analítica de cada niño. Así como la imprenta facilita la propagación de la tres R[4], la informática y las computadoras facilitan la propagación del pensamiento computacional.” (Wing J., 2006).

Existen muchos sistemas de programación gráfica para derribar las barreras en las primeras etapas de la enseñanza de programación de computadoras, ayudando a reducir el umbral a los principiantes para aprender a programar y cosechar los beneficios de la programación. En lugar de trabajar exclusivamente con el texto y recordando la sintaxis del lenguaje y las reglas, muchos sistemas de programación gráfica permiten a los alumnos manipular e interactuar con los objetos visuales o imágenes para construir sus proyectos de programación (Ricarose Vallarta R., 2007).

Por lo tanto, pensamos en una herramienta que motive a los alumnos a introducirlos en el aprendizaje de la programación de una forma visual y sencilla, de esta manera se concentran todas sus energías en la solución del problema.

RITA es una aplicación que permite a los alumnos a través de la manipulación de bloques de programación y de propiedades, programar las estrategias de combate (comportamiento) de unos robots virtuales que competirán en un batalla contra otros robots. De esta forma el alumno ve a su robot en acción enfrentándose a sus rivales e intenta mejorar su comportamiento, es decir, reescribir sus programas, para poder vencerlo. Con estas simples ideas el sistema motiva e incentiva a los estudiantes a autosuperarse para conseguir un comportamiento, que luego será traducido a un lenguaje de programación específico, en este caso particular Java.

El proyecto RITA se encuentra en desarrollo y en crecimiento, nuestra tesina RITA en Red incorpora funcionalidades que lo hacen más adecuado a las nuevas tecnologías, permitiendo que las batallas creadas por los estudiantes sean más sencillas utilizando la comunicación por red para el envío y recepción de los robots.

La aplicación RITA en Red implementa la arquitectura cliente/servidor y por lo tanto puede ser utilizada en dos modos:

[4] Sir William Curtis (1795) Bases de un programa para educación básica dentro de las escuelas en Inglaterra: lectura, escritura y aritmética. Recuperado de: https://en.wikipedia.org/wiki/The_three_Rs

- **Cliente:** en este modo el usuario de la aplicación envía por red el robot con el cual va a competir en la batalla. Usualmente el usuario que utiliza este modo es el alumno.

- **Servidor**: en esta modalidad el usuario se encarga de administrar las batallas recibiendo los robots de los distintos clientes, ejecutar la batalla y devolver a cada cliente el resultado de la misma. Generalmente este modo suele ser utilizado por los docentes para tener un control de los alumnos que participan de la competencia.

Con Rita en Red es posible disponer de un entorno de programación y ejecución sencillo para que cada alumno pueda enviar el robot con su estrategia diseñada para competir contra los demás y el docente administre los robots que van a participar de la batalla y la ejecute para luego enviarle el resultado a cada alumno.

5. Programación en las escuelas secundarias

En este punto mostraremos algunas de las iniciativas que se llevan a cabo en la actualidad en Argentina para fomentar la programación en las escuelas, como así también en distintos niveles de enseñanza educativa a nivel nacional y/o provincial. La decisión del estado nacional de promover iniciativas que intenten reducir la segunda brecha digital de acuerdo a la definición de Tim Berners-Lee, está orientado al crecimiento de la informática en todos los aspectos de la vida cotidiana, desde aplicaciones para celulares, ya sean juegos o que utilicen información de geolocalización como podrían ser la búsqueda de hoteles cercanos a donde nos encontramos, como a desarrollos específicos que mueven las compuertas de una represa.

A su vez, la industria del desarrollo de software es una cadena de producción muy importante para el desarrollo de nuestro país, cada vez son mayores las demandas de informáticos y de personas calificadas en programación para desarrollar sistemas que cumplan con los requerimientos que se desean realizar en las distintas áreas de la industria como demuestra el informe de La Cámara de Empresas de Software y Servicios Informáticos de la República Argentina (CESSI)[5]. El gráfico que se muestra en la figura 1 indica la evolución de los recursos formados en tecnologías de la información y su proyección a 2020 dado el crecimiento experimentado por los mismos entre 2006 y 2014. Como se puede apreciar la cantidad de estudiantes ha ido en aumento en forma pronunciada acompañado por el total de recursos generados para la industria. El número de graduados en carreras de informática como así también el de los técnicos secundarios está tendiendo levemente a crecer aunque esto se irá pronunciando si se mantiene con el mismo crecimiento el de estudiantes en las carreras de sistemas. De acuerdo al aumento del empleo observado en el sector de Software y Servicios Informáticos en estos mismos años, puede inferirse que aproximadamente la mitad de los recursos generados fueron absorbidos por empresas del sector.

[5] Cámara de Empresas de Software y Servicios Informáticos de la República Argentina (CESSI). (2015). Reporte anual del sector de software y servicios informáticos de la República Argentina. CABA, Argentina. Recuperado en Junio del 2015 de http://www.cessi.org.ar/opssi-reportes-949/index.html

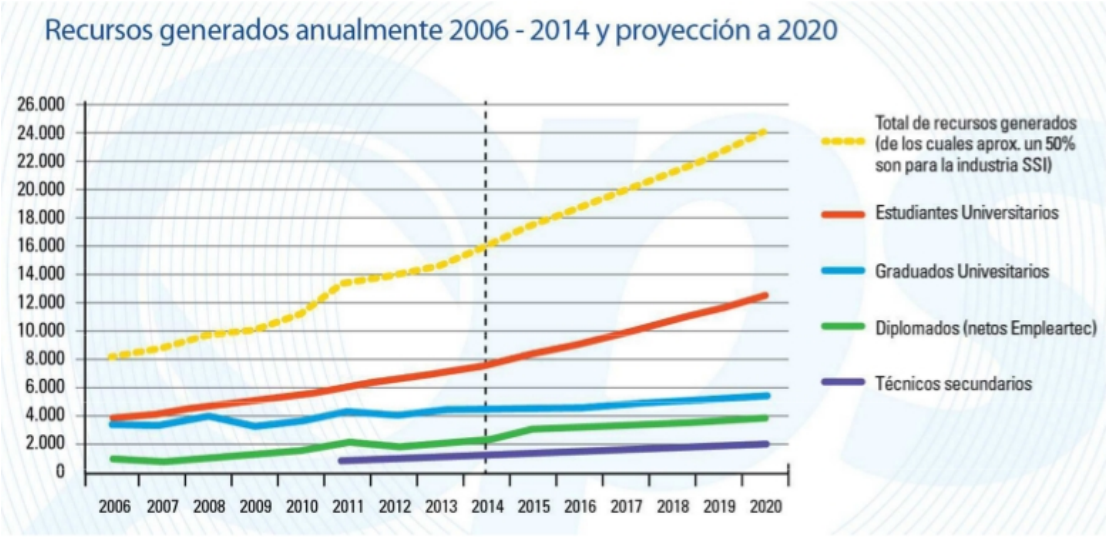


Figura 1 - Informe Anual 2015 del CESSI

5.1 Iniciativas en Argentina

5.1.1 Dale Aceptar

El concurso “Dale Aceptar”[6] es organizado por la Fundación Dr. Manuel Sadosky[7] siendo el objetivo del concurso la promoción de las “Tecnologías de la Información y las Comunicaciones” entre los jóvenes de la República Argentina. Este concurso está destinado a estudiantes de escuelas secundarias de la república Argentina, es libre y gratuito, sólo requiere inscripción previa en cualquiera de las categorías. Es un concurso de animaciones y juegos, no siendo necesario para los alumnos saber programar. El desafío tiene las categorías: Animación, Juego (ambas con nivel principiantes y avanzados) y Coartada. Además, en el nivel de los avanzados pueden participar los mayores de 14 años y menores de 24 años que ya no sean alumnos de escuelas secundarias y tampoco sean estudiantes de carreras terciarias o universitarias relacionadas con sistemas informáticos, computación o diseño.

A continuación pasamos a describir las distintas herramientas propuestas del concurso de “Dale Aceptar” de la convocatoria 2014.

[6] Ministerio de Ciencia, Tecnología e Innovación Productiva - Dale Aceptar (2013 -Actualidad). Argentina. Recuperado de: http://w ww.daleaceptar.gob.ar/

[7] Fundación Dr. Manuel Sadosky (2011 - Actualidad). Argentina. Recuperado de: http://www.fundacionsadosky.org.ar/

5.1.1.1 Chatbot

Uno de los software propuestos por el concurso es el Chatbot que, como se puede ver en la figura 2, fue encomendado por la Fundación Sadosky a un equipo de expertos liderados por la Dra. Luciana Benotti, del Grupo de Procesamiento de Lenguaje Natural[8] de la Facultad de Matemática, Astronomía y Física (FaMAF) de la Universidad Nacional de Córdoba[9] y desarrollado en colaboración con la empresa cordobesa de software LVK[10]. El equipo también está integrado por el Lic. Andrés Pagliano, María Emilia Echeveste y la Dra. Cecilia Martínez.

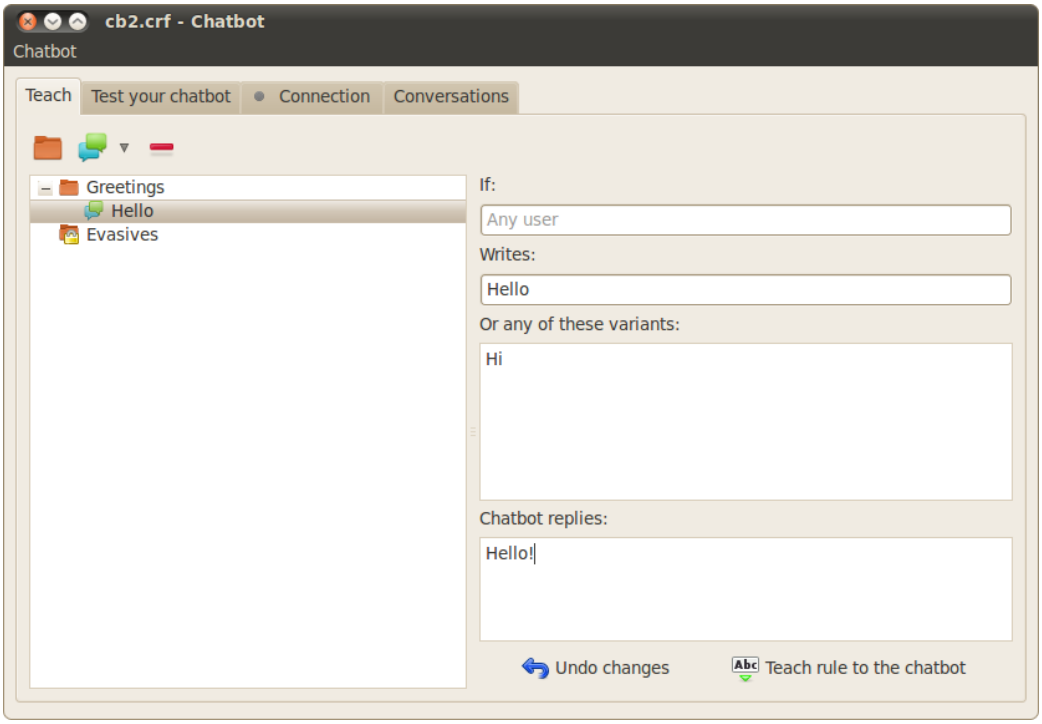


Figura 2 - Chatbot

Esta herramienta es libre y gratuita, se puede descargar[11] desde el portal del concurso y además es la primera herramienta disponible de su tipo que se

[8] Facultad de Matemática, Astronomía y Física (FaMAF) de la Universidad Nacional de Córdoba. Grupo de Procesamiento de Lenguaje Natural. Córdoba Argentina. Recuperado de: http://pln.famaf.unc.edu.ar/es

[9] Facultad de Matemática, Astronomía y Física de la Universidad Nacional de Córdoba. Córdoba, Argentina. Recuperado de: http://www.famaf.unc.edu.ar/

[10] LVK Software - Chatbot, One of a kind chatbot builder. Córdoba, Argentina. Recuperado de: http://www.lvklabs.com/

[11] Benotti Luciana, Lic. Andrés Pagliano, María Emilia Echeveste, Dra. Cecilia Martínez. Chatbot. Recuperado en Noviembre 2014 de http://www.daleaceptar.gob.ar/cms/coartada/descarga-chatbot/

conecta a las redes sociales. El Chatbot está disponible para ser usado y, a su vez, ser modificado. Esto significa que su código será liberado bajo una licencia de código abierto. La plataforma incluye herramientas de procesamiento de lenguaje natural (como un lematizador del idioma español) y un motor de análisis de reglas basado en el lenguaje AIML[12] del área de Inteligencia Artificial.

5.1.1.2 Alice

Alice[13] es un ambiente de programación educativo libre y abierto orientado a objetos con un entorno de desarrollo integrado (IDE). Está programado en Java y utiliza un entorno sencillo basado en «arrastrar y soltar» para crear animaciones mediante modelos 3D. De esta forma permite que los estudiantes aprendan los conceptos fundamentales de programación en el contexto de la creación de películas animadas y videojuegos simples. Este software fue desarrollado por los investigadores de la Universidad Carnegie Mellon[14]**.** La versión actual puede ser ejecutada en ambientes Windows, Mac y Linux.

La figura 3 muestra la interfaz de Alice**,** donde se puede apreciar a la izquierda de la imagen el panel con los distintos objetos con los cuales puede trabajar el programador, al centro, el objeto que diseñó y también podrá ver el comportamiento que le haya dado.

[12] Dr. Richard Wallace. Lenguaje AIML. (s.f.). En *Wikipedia*. Recuperado en Noviembre de 2014 de http://es.wikipedia.org/wiki/AIML

[13] Universidad Carnegie Mellon (1995 - 2000). Alice: A new way to teach programming. Virgina, USA. Recuperado en Noviembre 2014 de http://www.alice.org

[14] Universidad Carnegie Mellon. http://www.cmu.edu/

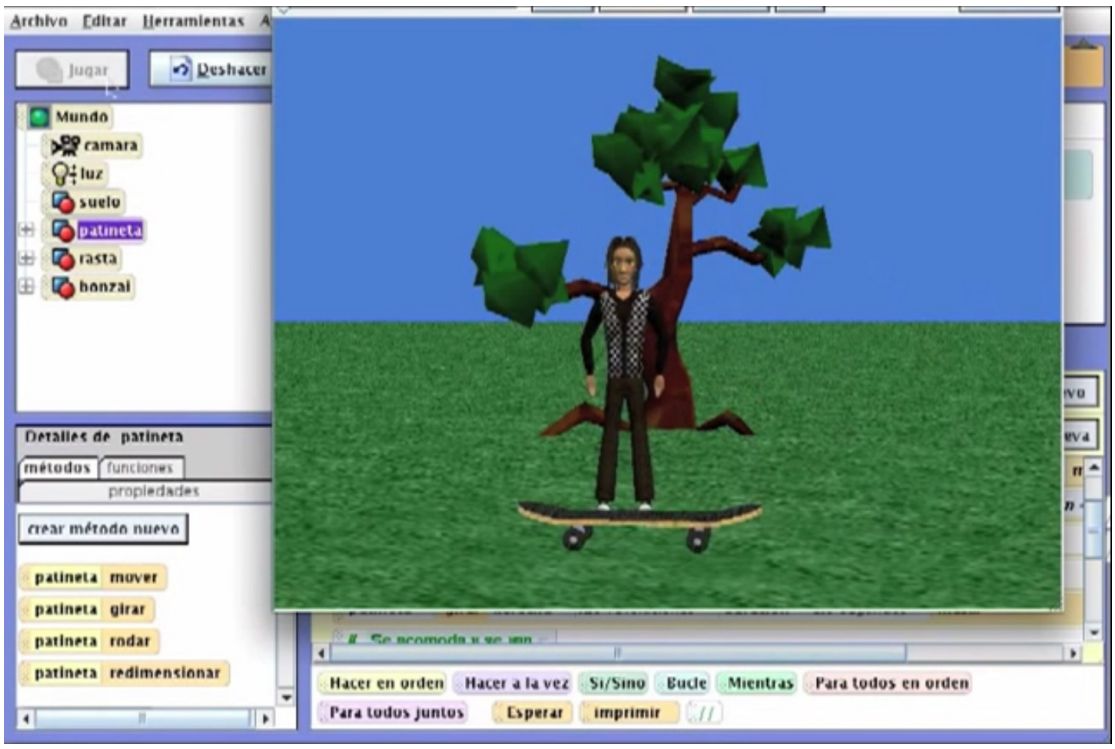


Figura 3 - Interfase de Alice

Alice permite a los estudiantes ver de inmediato cómo se ejecutan sus programas de animación, lo que les permite comprender fácilmente la relación entre las sentencias de programación y el comportamiento de los objetos en su animación. Mediante la manipulación de los objetos en su mundo virtual, los estudiantes adquieren experiencia con todas las construcciones de programación que normalmente se enseñan en un curso de introducción a la programación.

Al no existir en Alice un editor de código se alivian las dificultades del rigor sintáctico en los primeros pasos dentro de la programación. Alice propone introducir a los alumnos a la programación estimulando su creatividad: el estudiante tiene que diseñar previamente en un “Story-board” su mundo virtual antes de iniciar con el proceso de creación de su mundo virtual. Los alumnos se divierten de manera intuitiva a través de una interfaz gráfica amigable, van aprendiendo los conceptos de instancia, atributos, métodos y funciones; incluso el tema de arreglos es soportado por el ambiente de programación Alice.

5.1.2 Program.AR

Program.AR[15] es una iniciativa del Estado Nacional que busca acercar a los jóvenes al aprendizaje de las Ciencias de la Computación y concientizar a la sociedad en general sobre la importancia de conocer estos conceptos.
La iniciativa es llevada a cabo por:

- Jefatura de Gabinete de Ministros.
- Fundación Sadosky (Ministerio de Ciencia, Tecnología e Innovación Productiva).
- Portal educ.ar (Ministerio de Educación).
- Programa Conectar Igualdad (ANSES y Ministerio de Educación).

El principal fin es promover en estudiantes de todas las edades el aprendizaje de la programación informática, con actividades libres y gratuitas. Los proyectos hasta el momento que se han iniciado son:

- **Foros de debate:** eventos abiertos a toda la comunidad que se realizan en distintas regiones del país y que buscan generar una concientización sobre la problemática así como un debate dinámico, democrático e inclusivo. Cada uno de los encuentros está integrado por: talleres, una jornada de debate y un hackatón de desarrollo. Además, se abren espacios virtuales para que todos puedan opinar sobre los tópicos y seguir enriqueciendo el diálogo entre todos.
- **La hora del código:** el plan Program.ar se sumó a la “Hora del Código”, una campaña mundial organizada junto con la plataforma Code.org[16] que tiene como principal fin promover en estudiantes de todas las edades el aprendizaje de la programación informática, con actividades libres y gratuitas. La “Hora del Código” es un movimiento en el que participan más de 180 países y el año pasado formaron parte unos 15 millones de estudiantes de todo el mundo. La iniciativa busca incentivar entre las escuelas la práctica de una serie de ejercicios de programación, para los que no se requiere conocimientos

[15] Fundación Sadosky del Ministerio de Ciencia, Tecnología e Innovación Productiva, el portal Educ.ar del Ministerio de Educación, la Jefatura de Gabinete de Ministros y el Programa Conectar Igualdad a través de ANSES y el Ministerio de Educación. (2013 - Actualidad). Program.ar. Argentina. Recuperado en Noviembre 2014 de http://program.ar/
[16] Hadi, Partovi & Ali Partovi (2013 - Actualidad). Code.org. Recuperado en Noviembre 2014 de: http://code.org/

previos y que cuenta con niveles iniciales y avanzados. La idea es que cualquier escuela puede organizar "Una Hora del Código" en cualquier momento, pero el objetivo de esta campaña popular es que millones de estudiantes prueben "Una Hora del Código" entre el 8 y el 14 de diciembre, en la celebración de la Semana de la Educación de las Ciencias Computacionales. No existe una hora específica para realizarlo, se puede celebrar "Una hora de código" en cualquier momento de esa semana, o cuando se pueda, la idea es realizarlo. Los diferentes tutoriales que se crearon para que los estudiantes se inicien en la programación son variados, desde programar el juego “Angry birds”[17]; “Pilas Engine”[18], que es un motor para realizar videojuegos de manera rápida y sencilla, argentino, con distintos niveles de dificultad; “Mis amigos robóticos”[19], que está diseñado para realizarse sin una computadora;“Khan Academy”[20], para hacer dibujos generados por computadora, etc.

5.1.3 Iniciativas de la Facultad de Informática

La Facultad de Informática de la Universidad Nacional de La Plata continuando con las iniciativas y programas impulsados por el Ministerio de Ciencia y Tecnología y Educación como Program.AR y Conectar-Igualdad, promueve 2 iniciativas que persiguen el mismo objetivo. (Díaz J. et al, 2014).

5.1.3.1 Programando con robots y software libre

La Facultad de Informática inició este proyecto en el año 2008 y consiste en la enseñanza de programación de robots simples en la escuela secundaria usando el lenguaje Python. En una primera etapa se utilizaron robots denominados Scribblers que pueden desplazarse sin estar conectados por medio de ningún cable ya que utilizan la tecnología bluetooth para la comunicación entre la computadora y el robot y poseen sensores que permiten detectar obstáculos, detectar contrastes de colores y tomar fotografías utilizando una cámara web incorporada. Los Scribblers se programan utilizando el lenguaje de programación Python y la API (Application Programming Interface) Myro (Lanfranco F., et al. 2012)

[17] Rovio Entertainment (2009) Angry Birds. (s.f.). En *Wikipedia*. Recuperado en Noviembre 2014 de: http://es.wikipedia.org/wiki/Angry_Birds

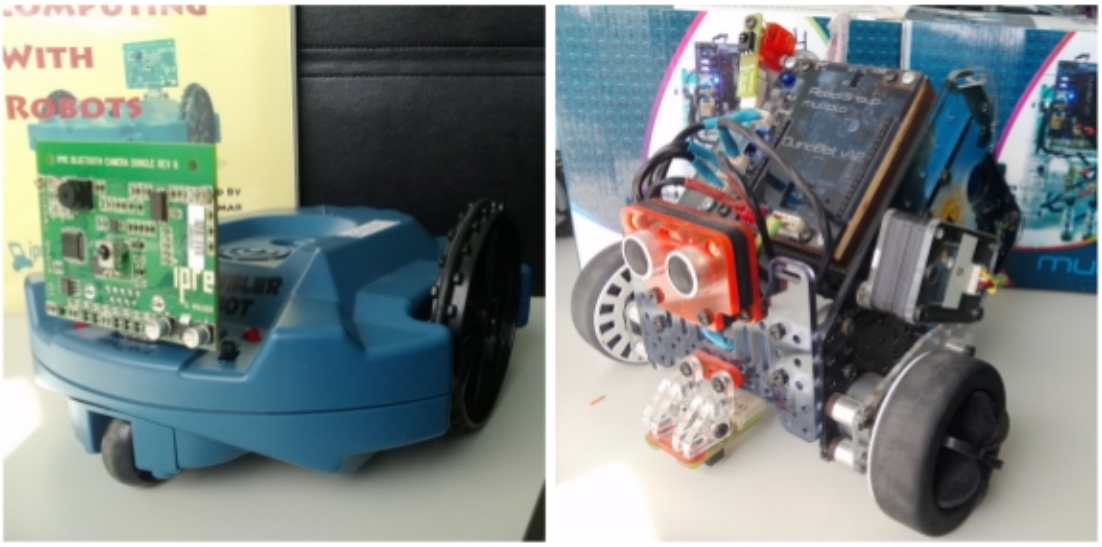
[18] Hugo Ruscitti (2010 - Actualidad). Pilas Engine. Argentina. Recuperado en Noviembre 2014 de: http://pilas-engine.com.ar/

[19] Thinker Smith (2011 - Actualidad). My Robotics Friends. Recuperado en Noviembre 2014 de: http://thinkersmith.org/

[20] Salman Khan (2006 - Actualidad). Khan Academy. Recuperado en Noviembre de 2014 de: https://es.khanacademy.org/

desarrollada en el Georgia Tech Institute. Los robots Scribbler son importados, por este motivo se buscaron alternativas de fabricación nacional que pudieran ser programados en lenguaje Python y tuvieran especificaciones abiertas. De esta manera en el año 2012, se adquirieron robots Múltiplo N6, de fabricación nacional, de la empresa RobotGroup. Tanto el software necesario para controlar estos robots como sus especificaciones se distribuyen bajo una licencia abierta (López M. et al., 2015)

A lo largo del proyecto se utilizaron los dos modelos de robots mencionados anteriormente: el Scribbler y el Multiplo N6. Ambos robots se muestran en la figura 4.



(a) Robot Scribbler de Parallax *(b) Robot Multiplo N6 de RobotGroup*

Figura 4 - Robots del proyecto “Programando con robots y software libre”

Características del robot **Scribbler**:

Los robots Scribbler (versión 1) cuentan con un microcontrolador que viene programado con un intérprete de Basic. Para su locomoción este robot cuenta con dos ruedas conectadas a través de cajas reductoras a sus respectivos motores de DC (corriente continua) y una tercer rueda no conectada a ningún motor que sirve como tercer punto de apoyo para el cuerpo del robot. La alimentación eléctrica del robot es provista por 6 pilas AA.

En cuanto a los sensores, este robot está equipado con:

- Dos emisores infrarrojos a los lados de su parte frontal y un sensor infrarrojo entre ellos que permite sensar (de acuerdo a si se refleja el haz de luz infrarroja de alguno de los laterales) si hay algún obstáculo a la derecha o a la izquierda del robot.

- Tres fotorresistores ubicados dentro de 3 cavidades al frente del robot que permiten sensar la presencia de fuentes de luz e identificar (si la luz no es muy difusa) si las mismas se encuentran directamente enfrente, a la izquierda o a la derecha del robot.

- Dos sensores de línea que consisten cada uno de un emisor y un sensor infrarrojo (IR). Los mismos detectan cambios de contraste en el piso de acuerdo a la cantidad de luz IR reflejada y generalmente se utilizan para que el robot siga un camino demarcado por una línea de un color sobre una superficie de un color contrastante.

Si bien las anteriores son las características generales de estos robots, usándolos de esta manera es necesario programarlos a través de un cable serial usando el lenguaje PBASIC basado en BASIC. Para las actividades planteadas en el marco del proyecto “Programando con robots y software libre” se utilizó una versión que agrega un dispositivo que se conecta al puerto serial del robot y provee la capacidad de controlarlo de forma inalámbrica usando Bluetooth. Este dispositivo también provee una cámara de baja resolución con la que se pueden tomar fotografías, que luego serán transmitidas por Bluetooth. Además agrega un sensor infrarrojo que permite detectar la presencia de obstáculos a la izquierda, al centro o a la derecha (está compuesto de 3 emisores y un receptor). Con esta incorporación el robot queda equipado ahora con sensores en la parte frontal y en la parte trasera, pero más importante aún el módulo Bluetooth permite controlarlo de forma inalámbrica, esto es lo que permite que estos robots puedan ser programados con Python.

Características del robot **Multiplo N6**:

El robot Múltiplo N6, fabricado por la empresa RobotGroup, cuenta con un microcontrolador con Arduino[21] precargado. Para su locomoción cuenta con dos ruedas conectadas a través de reductores de velocidad a sus correspondientes motores de corriente continua y una tercer rueda no conectada a ningún motor que sirve como tercer punto de apoyo. La

[21] Arduino . Recuperado en Noviembre de 2014 en: https://www.arduino.cc/

alimentación eléctrica de estos robots está dada por tres pilas AA en serie (o bien dos packs de tres pilas cada uno en paralelo).

En cuanto a los sensores el Múltiplo N6 está equipado con:

- Un sensor ultrasónico para detectar obstáculos al frente del robot.

- Un sensor infrarrojo que puede ser utilizado en conjunto con un control remoto universal para enviar comandos al robot (siempre que se programe este comportamiento previamente).

- Dos sensores detectores de línea, compuestos de un emisor y un receptor infrarrojo cada uno. Opcionalmente estos sensores pueden desatornillarse y ensamblarse para funcionar como “encoders” de las ruedas y de esta manera es posible calcular la velocidad de giro de las ruedas. Este uso no es recomendable si se controla el robot de forma remota ya que la latencia de las comunicaciones puede hacer que se pierdan datos haciendo que el cálculo de la velocidad sea impreciso.

En esta configuración estos robots son programables usando el entorno de Arduino y desde MiniBloq (entorno de programación que permite programar el robot de forma gráfica usando bloques), pero dado que el objetivo del proyecto es la enseñanza de programación en el lenguaje Python se encargó a la empresa una versión modificada del Múltiplo N6 que permitiera controlarlo de forma inalámbrica, de manera tal que una computadora con un intérprete de Python instalado pudiera enviar instrucciones y recibir los valores de los sensores del robot tal como se hace con los robots Scribbler.

En el marco del proyecto, las actividades que los estudiantes realizan con los robots están orientadas a eventos sociales y lúdicos, promoviendo la creatividad en el desarrollo de las mismas. La primera motivación está dada por querer mover al robot, en donde se plantean carreras, batallas, bailes etc., pero pasada esta primera instancia, se trabaja de manera más compleja que requiere conocimientos más avanzados del lenguaje y se plantean actividades interdisciplinarias, convocando a docentes de otras áreas.

5.1.3.2 Programando con RITA

El LINTI (Laboratorio de Investigación en Nuevas Tecnologías Informáticas) inició el proyecto RITA (Remote Inventor to Teach Algorithms) en 2012 y su objetivo es acercar la programación a la escuela secundaria, a través de la programación de videojuegos de robots virtuales. Las diferentes versiones

de RITA fueron usadas en los proyectos de extensión de la Facultad de Informática “Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica” y “Extensión en vínculo con escuelas secundarias” pudiendo llevar adelante intervenciones inicialmente en escuelas secundarias técnicas de la provincia de Buenos Aires con orientación informática y luego en escuelas de enseñanza media.

Actualmente se está trabajando con RITA con estudiantes y docentes secundarios de diferentes niveles, en actividades que estimulen la curiosidad, generen espacios exploratorios, colaborativos y experimentales. Esto se lleva a cabo mediante la implementación de talleres en las escuelas y/o en la Facultad que promueven la integración con el plan nacional Conectar-Igualdad.

6. RITA (Remote Inventor to Teach Algorithms)

Como se mencionó anteriormente, RITA es un desarrollo de la Facultad de Informática iniciado a partir del trabajo de grado de la Lic. Vanessa Aybar Rosales en el año 2012. RITA es una aplicación hecha en Java que integra los frameworks de código fuente abierto Openblocks y Robocode. Esta integración consiste en adaptar y extender las funcionalidades brindadas por Openblocks para que brinde soporte a las clases Java provistas por Robocode, así como también a las estructuras clásicas de Java de modo que el usuario de RITA pueda mediante Openblocks escribir las estrategias de los robots de Robocode.

6.1 Características de RITA

RITA permite crear robots virtuales simples, poniendo en práctica los conceptos de programación que los alumnos deben aprender, al menos en la primera etapa del aprendizaje. Estos robots mantienen información básica de su estado y además reaccionan a eventos que ocurran sobre ellos mismos o en su entorno.
La programación en RITA se realiza a través de la manipulación de bloques provistos por el framework Openblocks, de la forma de arrastrar y soltar.
RITA provee los siguientes tipos de bloques:

- Estructuras de control (if-then / if-then-else / while / for) (Ejemplo en figura 5)
- Definición de variables, a las cuales se les puede dar un valor o recuperarlo
- Creación de métodos con parámetros y valores de retorno
- Tipos de datos Integer, Boolean y String
- Variables predefinidas
- Métodos predefinidos

Figura 5 - Bloques de estructuras de control en RITA

Con estos elementos el alumno construye estructuras de bloques que corresponden a su estrategia de combate. En general, hay una relación directa entre la complejidad de la estrategia y la complejidad de la estructura de bloques resultante.

Una vez que el alumno programa la estrategia de su robot, RITA permite evaluar la misma en el campo de batalla de Robocode.

Robocode es un framework libre para desarrollar estrategias de robots virtuales, data del inicio de la década del 2000 y fue creado por Mathew Nelson. Robocode está desarrollado en el lenguaje JAVA pero también incorpora la posibilidad de desarrollar los robots en la plataforma .NET aunque su uso no es tan extendido, este se puede descargar desde su sitio web.[22]

6.1.1 OpenBlocks

Existen muchos sistemas de programación gráfica para derribar las barreras de la programación en computadoras, que ayudan a reducir el umbral a los principiantes para aprender a programar y cosechar los beneficios de la programación. En lugar de trabajar exclusivamente con el texto y recordar la sintaxis del lenguaje y las reglas, muchos sistemas de programación gráfica permiten a los usuarios manipular e interactuar con los objetos visuales o imágenes para construir sus proyectos de programación. Openblocks es una librería Java de código fuente abierto que permite crear interfaces de usuario de programación basado en bloques distribuido por el Massachusetts Institute of Technology (MIT) a través de su programa Scheller Teacher Education Program (STEP) que surge como tesis de maestría en Ciencias de Computación e Ingeniería de Ricarose Vallarta Roque realizado en el año 2007.

[22] Robocode. http://sourceforge.net/projects/robocode/files/

Openblocks es un framework libre y ya no se encuentra en desarrollo en el **MIT STEP**, pero se creó un grupo de Google para preguntas y apoyo[23]. Los autores del software son miembros activos del grupo y siguen ayudando a los desarrolladores interesados, lo que aumenta la calidad del apoyo de toda la comunidad.

Openblocks mantiene las ideas de StarLogo TNG[24], mayormente la interfaz visual, es decir la forma de los bloques y conectores. Además se agregaron características en pro de mejorar la vista y usabilidad de los bloques.

Openblocks brinda un espacio de trabajo o workspace donde se crea un programa a través del movimiento de arrastrar y soltar bloques. Los bloques se encuentran definidos en un archivo XML. En el espacio de trabajo a la izquierda se encuentran las imágenes de los bloques disponibles, mientras que del lado derecho está el panel de edición, llamado “block canvas” que contiene los bloques que agrega el programador para componer su programa. El panel de edición además puede estar dividido en sub secciones, que corresponden a páginas. Cada página servirá de contenedor a partes de nuestro código, como se ve en la figura 6 que se extrajo de la tesis de maestría de Ricarose Vallarta Roque que es abierta para todo público.

[23] Eric Klopfer (2009 - actualidad). MIT Scheller Teacher Education Program. Referencia al grupo de google: https://groups.google.com/forum/?fromgroups#!forum/openblocks

[24] Eric Klopfer, Andrew Begel (2006 – actualidad). MIT Scheller Teacher Education Program. Recuperado de: http://education.mit.edu/portfolio_page/starlogo-tng/

Figura 6 - Ambiente de programación de StarLogo TNG

Openblocks implementa muchas características de StarLogo TNG que ayudan a los usuarios a administrar la complejidad de sus proyectos y herramientas para que sean programadores de bloques más eficientes. Por ejemplo, una de las críticas de los entornos gráficos de programación es cómo los objetos visuales, en este caso los bloques, pueden absorber rápidamente espacio en pantalla a medida que crecen los proyectos. StarLogo TNG implementó una función de zoom para ayudar a los usuarios a administrar su espacio de trabajo. También implementó una función de plegado de código, donde una pila de bloques puede "colapsar" u ocultar su conjunto de bloques, dejando sólo un bloque para representar la pila.

Características de Openblocks:

- **Funcionalidades esenciales del entorno de programación:** bloques gráficos, un ambiente para el manejo de los bloques y un “cajón” para guardar esos bloques. También incluye mecanismos para guardar y cargar proyectos y deshacer y rehacer acciones del usuario.

- **Múltiples formas de configurar los elementos específicos de interfaz de usuario:** el sistema proporciona más de una manera de configurar determinados elementos de la interfaz de usuario. Por ejemplo, los desarrolladores deben ser capaces de elegir si sus cajones de bloque (que contienen bloques) flotan por encima del “canvas de bloque” o permanecen en una ubicación estática.

- **Entorno de programación personalizable:** el sistema proporciona a los desarrolladores formas de personalizar el entorno de programación, tales como activar o desactivar ciertos elementos de la interfaz, posicionar elementos en múltiples ubicaciones y cambiar el aspecto y la sensación del entorno en general.

- **Fácil de extender:** los desarrolladores y diseñadores de lenguajes sólo tienen que trabajar con el menor número de componentes que sea posible para crear un entorno de programación de bloques muy simple.

- **Fácil de entender:** las características y componentes del sistema deben ser fáciles de entender para todos los usuarios. En otras palabras, no se requiere de una gran cantidad de documentación y código fuente para empezar a programar. Las características del sistema deben ser comprensibles para los desarrolladores y diseñadores de lenguajes para que puedan trabajar con la totalidad del conjunto de características. Y, por último, los elementos de la interfaz de usuario y las acciones del sistema deben ser intuitivas para los usuarios finales.

- **Documentación completa:** múltiples fuentes de documentación clara y detallada deben ser proporcionados en forma de tutoriales, ejemplos, especificaciones y una API pública para el framework Openblocks.

- **Funcionalidad optimizada:** funcionalidad común, tal como arrastrar y soltar los bloques dentro del ambiente debe ser optimizada y bien probada.

6.1.2 Robocode

Robocode es un juego educativo de código fuente abierto cuyo objetivo es programar la estrategia de un tanque-robot para competir contra otros robots en un campo de batalla. El jugador es el programador del robot, quien no tendrá influencia directa en el juego, sino que mediante el código brindará inteligencia al robot, indicando cómo comportarse y reaccionar frente a eventos ocurridos en el juego.

El juego está diseñado para ayudar a aprender Java de una manera entretenida, donde el desafío es el estímulo para el aprendizaje. Los robots son escritos en el lenguaje de programación Java y el juego Robocode, también escrito en Java, puede ejecutarse en cualquier sistema operativo que posea la plataforma Java, tales como Windows, Mac OS X, Linux etc.

Las batallas de Robocode ocurren en un campo de batalla en tiempo real en pantalla, como se puede observar en la figura 7, donde pequeños robots luchan hasta que sólo uno resulta victorioso en la contienda. Cada batalla consiste en una sucesión de turnos en los cuales cada robot realiza un movimiento u acción de acuerdo al comportamiento que le dieron. Esta característica nos permite realizar un seguimiento de la batalla por turnos con las opciones de parar, retroceder o avanzar un turno.

Figura 7 - Campo de batalla en Robocode

También es posible probar un robot contra otros mediante la descarga de su código fuente Java. Robocode ofrece un entorno seguro en relación a lo que se puede hacer en la máquina donde se ejecuta, y de esta manera la redistribución a través de Internet sea segura.

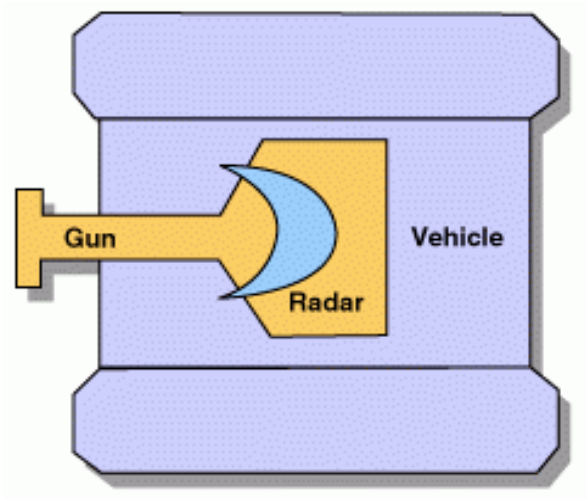


Figura 8 - Tanque de Robocode

6.1.2.1 Partes de un robot y su manejo

Los robots en Robocode son representados como tanques de guerra como se puede observar en la figura 8 y se componen de tres partes: el cuerpo (vehicle), el cañón (gun) y el radar (radar).

El cañón rota sobre el eje central del tanque al igual que el radar y dicha rotación es controlada completamente desde programación. Solamente del cuerpo y del radar podemos obtener información:

- El cuerpo: con getX() y getY() localizamos al tanque en el campo de batalla y con getHeading() obtenemos la dirección hacia donde esté mirando. Este último dato es importante ya que para hacer avanzar el robot, sólo disponemos de las funciones ahead (distancia_a_avanzar) y back(distancia_a_retroceder).

- El radar: lo podremos girar con turnRadarRight(grados) o turnRadarLeft (grados) y esto es importante porque **el radar es direccional** y lo que no esté en su radio de acción no lo detectará.

Existen más comandos disponibles, como fijar el radar al cañón o al cuerpo, para consultar tu estado y disparar. Los más importantes son los que se refieren al escáner del radar. Cada vez que nuestro radar detecta un robot a nuestro alcance se nos hace saber mediante el evento onScannedRobot donde podremos detectar todo sobre el tanque que acabamos de avistar.

6.1.2.2 El entorno de programación

Para programar nuestros robots no necesitaremos ningún IDE externo ni instalaciones complicadas: todo lo necesario está en el mismo Robocode, este provee de editor de código, debugger y compilador como muestra la figura 9.

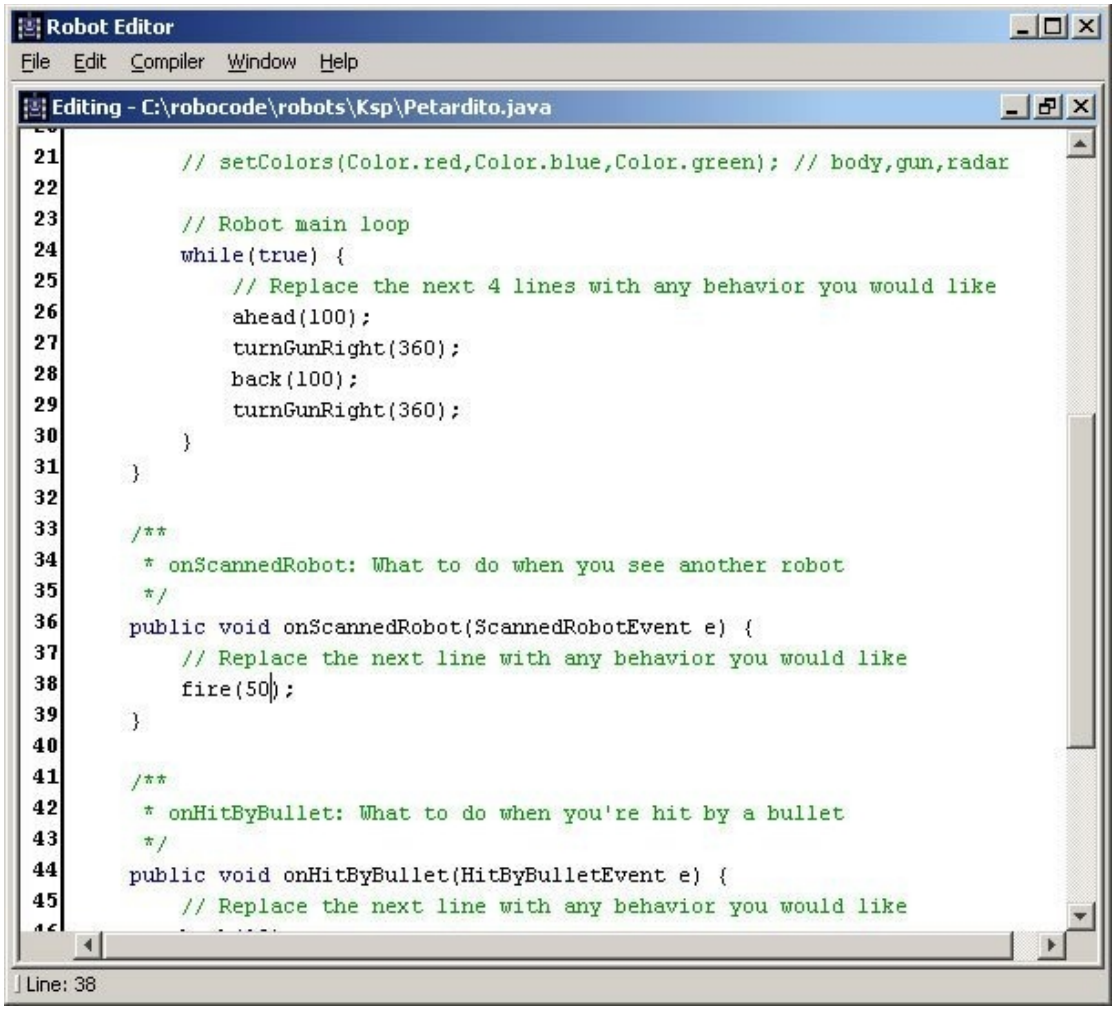


Figura 9 - Entorno de desarrollo de Robocode

6.2 Funcionalidades provistas por RITA

El objetivo de RITA es crear robots para que puedan enfrentarse en batallas contra otros, para cumplir con esto, primero creamos un robot, luego a través de bloques provistos por una interfaz simple arrastrando y soltando le damos comportamiento. RITA convertirá estos bloques en código Java que define los movimientos y acciones de nuestro robot para luego competir en una batalla y probar así nuestra estrategia programada.

A continuación describimos las funcionalidades que permiten realizar lo mencionado anteriormente.

6.2.1 Creación del robot

Cuando el alumno ingresa al sistema se le solicita un nombre para su robot como vemos en la figura 10.

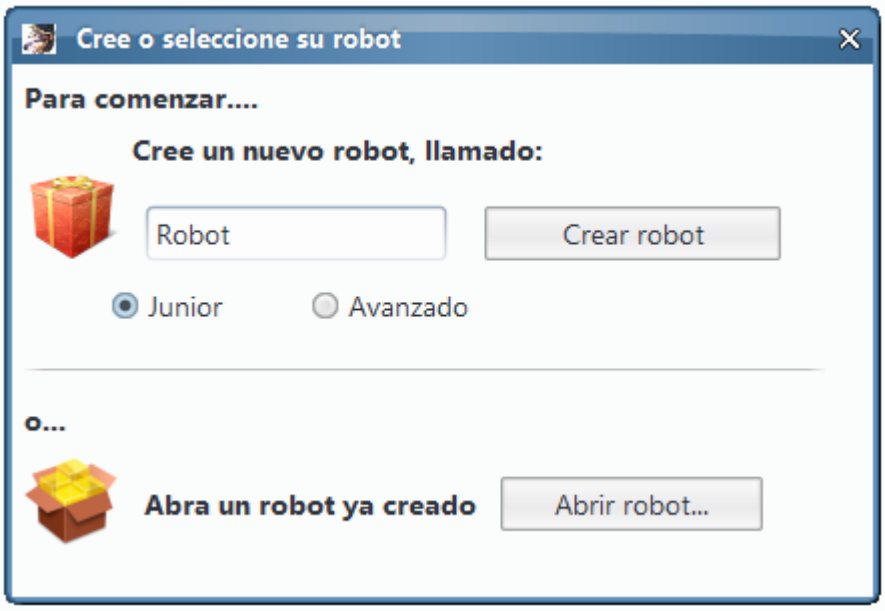


Figura 10 - Creación de Robot

Una vez seleccionado el nombre del robot se puede comenzar a programar el comportamiento y luego probarlo. El nombre elegido es el nombre de la clase Java que se va a compilar y luego ejecutar con los demás robots que participen de la batalla.

6.2.2 Programación del robot

Una vez creado el robot, el alumno le da comportamiento a su robot arrastrando y ubicando los bloques en el área de trabajo que provee Rita, como se muestra en la figura 11.

Los bloques que posee Rita están organizados en distintas categorías como se hizo mención anteriormente.

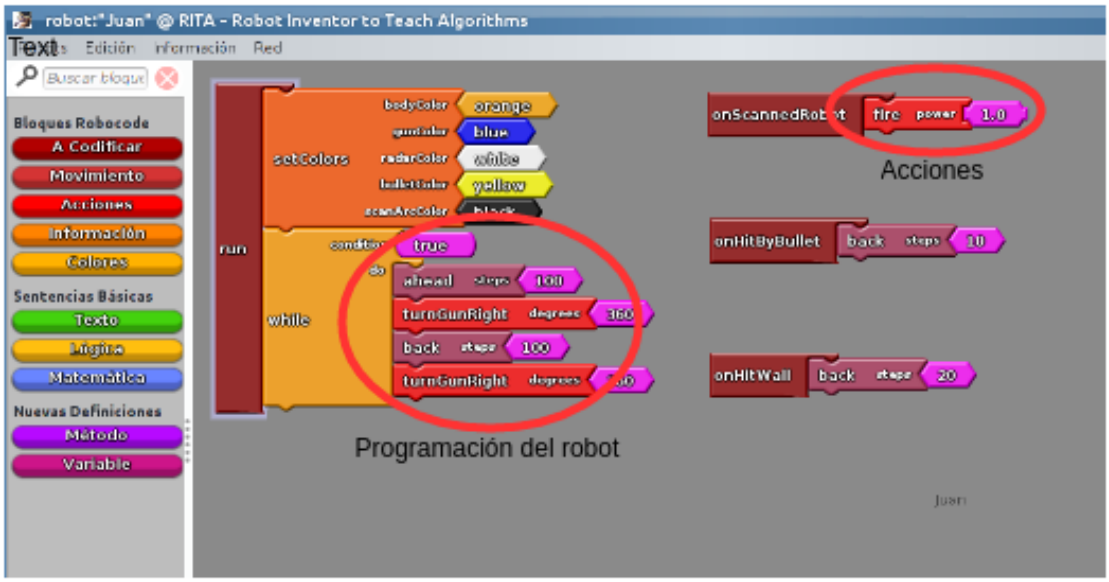


Figura 11 - Programación del robot

6.2.3 Generación de Código fuente Java

El alumno tiene la posibilidad de ver el código fuente Java de su robot generado a partir de la estrategia provista por él mediante la programación con bloques. Esto se puede hacer presionando el botón **“Ver código”**, como se indica en la figura 12.

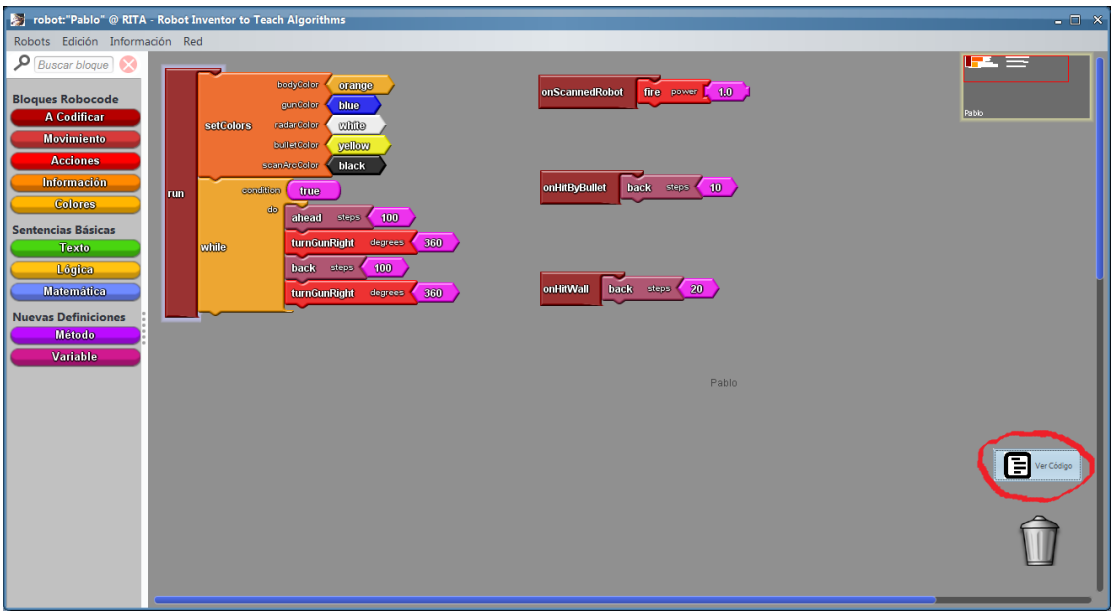


Figura 12 - Ver código del Robot

Tener la posibilidad de ver del código fuente Java es útil para que el alumno vaya familiarizándose con las distintas estructuras del lenguajes y poder relacionar las acciones que se introducen con bloques en **RITA** con el código fuente Java. En la figura 13 se puede ver el código Java generado automáticamente a partir de la programación en bloques.

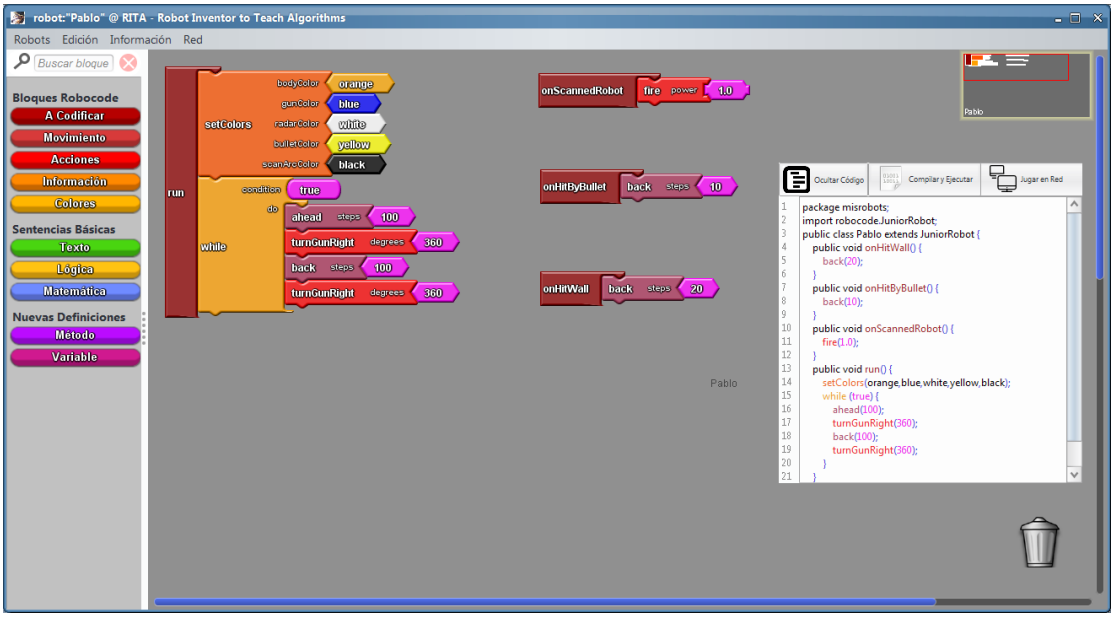


Figura 13 - Vista del código Java

6.2.4 Configuración y ejecución de una batalla

Una vez que el alumno ha definido el comportamiento de su robot programando en bloques, puede ejecutarlo para probar la estrategia programada contra otros robots. Para ello debe seleccionar contra quiénes va a competir, las rondas de la batalla y si los robots arrancan en posiciones fijas; todas estas configuraciones se pueden observar en la figura 14.

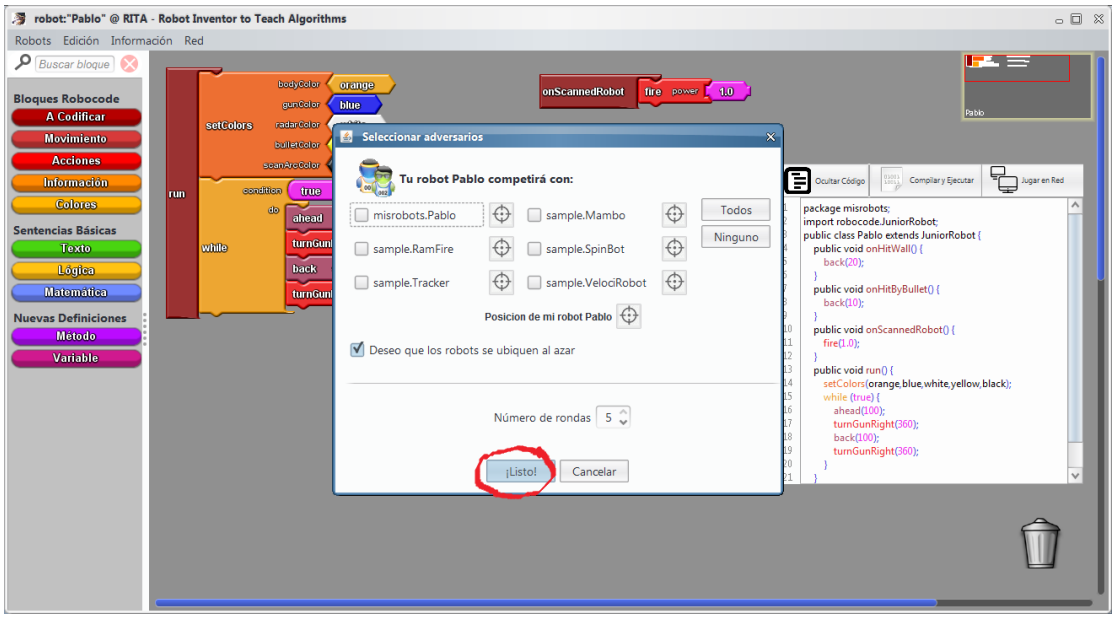


Figura 14 - Confiugración y Ejecución de la Batalla

Para poder realizar la ejecución se debe elegir la opción **“compilar y ejecutar”** como puede observarse en la figura 15 seleccionando los robots con los cuales se va competir. Los mismos pueden ser los que vienen como ejemplo en RITA, uno creado por el mismo alumno u otro creado por otro alumno. Una vez seleccionados los rivales, es posible ejecutar la batalla. Una vez finalizada la batalla se podrá saber qué robot resulta ganador, también es posible observar resultados parciales y conocer la energía que tiene cada robot que participa de la batalla.

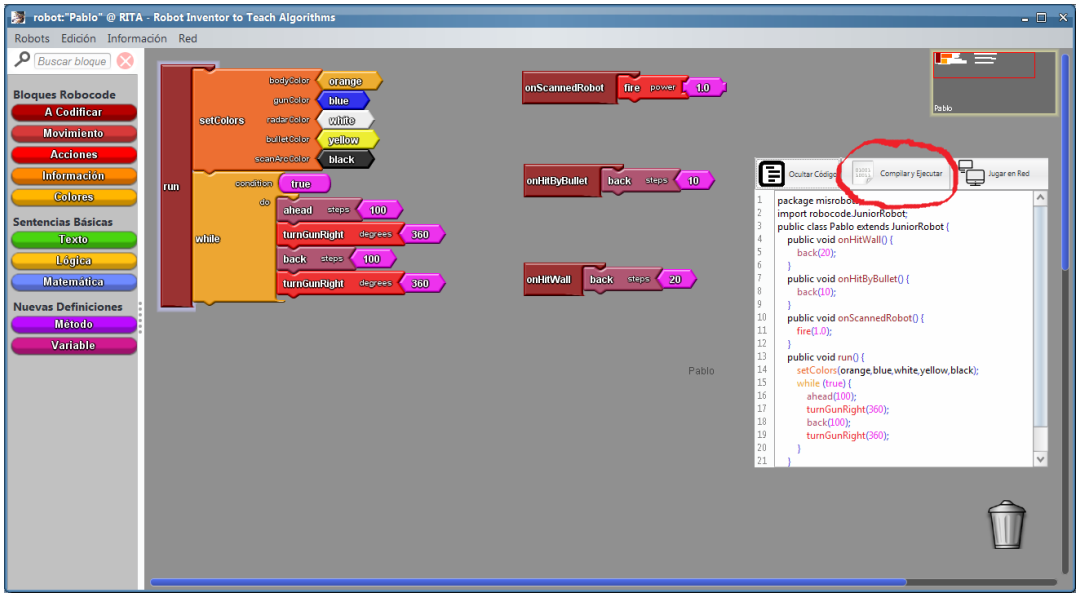


Figura 15 - Preparándose para la batalla

Luego de presionar el botón “¡Listo!” se muestra la batalla de los robots participantes de la misma y se llevan a cabo tantas rondas como se haya seleccionado. La batalla se observa en pantalla como en la figura 16.

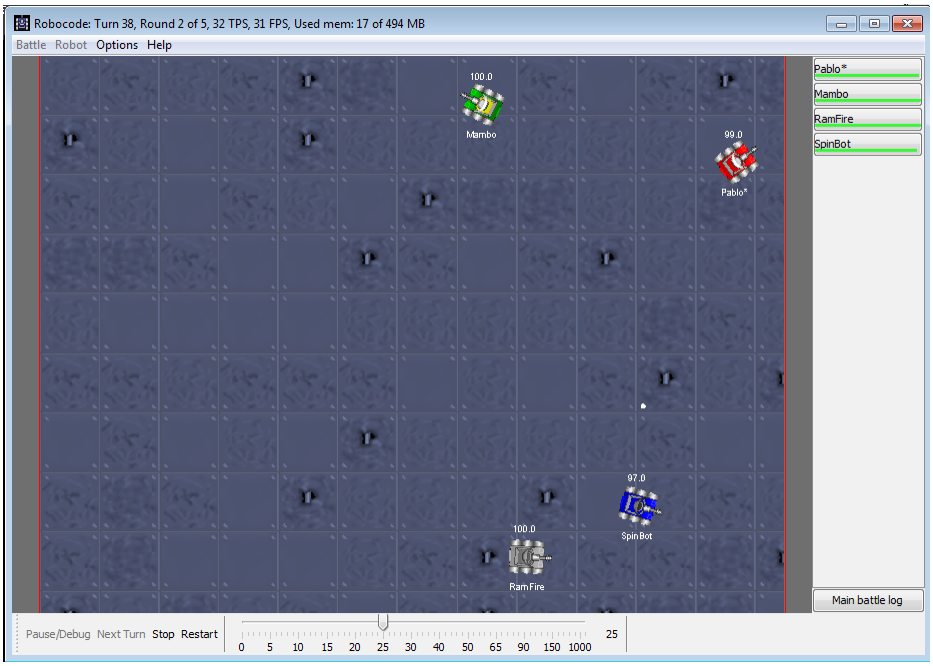


Figura 16 - Vista de la batalla

7. RITA en Red

Como hemos mencionado anteriormente **RITA** es una herramienta que incentiva a los estudiantes a programar, pero además ayuda a los alumnos a interiorizarse en los distintos conceptos de las **fases del desarrollo de software.**

Lo primero que realiza el alumno es pensar la estrategia de su robot (etapa de diseño), luego convierte esa estrategia en un algoritmo utilizando los distintos tipos de bloques (etapa de codificación) para terminar con la competencia del robot en la batalla (etapa de ejecución y testing). Cuando la batalla termina, el alumno observando el resultado de la competencia puede seguir mejorando su estrategia, volviendo a entrar en el ciclo de vida del desarrollo de software.

Obviamente vemos que es una forma simplificada de las etapas del desarrollo de software pero si el estudiante es guiado en el desarrollo de su algoritmo y se le van dando los conocimientos de la **ingeniería del software** está entrando de a poco en las formas en que se desarrollan los sistemas.

La figura 17 muestra la pantalla de selección de robots que participarán de una competencia. Esta acción se realiza luego que el alumno presionó el botón “compilar y ejecutar”.

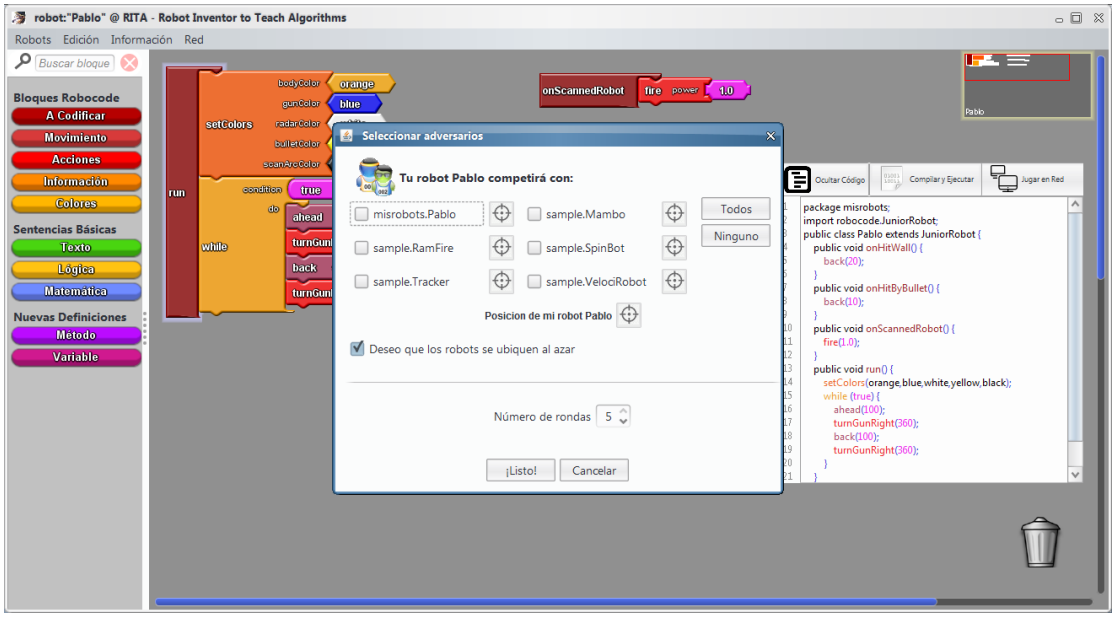


Figura 17 - Selección de robots para una batalla

Existen robots que vienen cargados en forma predeterminada para que se usen de ejemplo y se puedan hacer pruebas. Sin embargo, si se desea competir con un robot que desarrolló un compañero de aula, éste se debe cargar manualmente. RITA no provee ninguna funcionalidad para compartir robots.

Los pasos que se deben realizar para poder poner a competir robots desarrollados por compañeros de aula son:

1. Programar en bloques el comportamiento del robot en RITA en una computadora, llamémosla Computadora A.
2. Guardarlo en un archivo (XML).
3. Guardarlo en algún medio de almacenamiento removible o enviarlo por correo electrónico.
4. Abrir el archivo XML desde una instalación de RITA en otra computadora, llamémosla Computadora B y para cargar el robot creado por el alumno de la Computadora A.
5. El alumno de la Computadora B puede desde la selección de robots elegir el robot programado en la Computadora A

Esta forma de realizar una batalla con uno o más compañeros no es muy práctica, esto fue uno de los puntos fuertes para el desarrollo de **Rita en Red**. La idea de poder conectarse directamente con un servidor, que ejecuta también **Rita**, para enviar y compartir los robot desarrollados por los alumnos y que todo aquel que desea participar de una batalla con esos robots sólo necesite conectarse al mismo servidor, hace que la experiencia de usar RITA en un aula y realizar batallas, sea por un lado más sencillo y por otro acelera el tiempo de armado de competencias de robots de muchos participantes.

Rita en Red tiene dos modos de trabajo: cliente y servidor**.** El modo cliente, es el habitual de RITA con algunas funcionalidades extras para conectarse al servidor. El modo servidor, provee funcionalidades de manejador u administrador de batallas, por lo tanto puede decidir de cuántas rondas será la batalla y quiénes podrán participar.

Como mencionamos anteriormente, **Rita** fue pensado para acercar la enseñanza de la programación en escuelas secundarias, con **Rita en Red** buscamos lograr una mejor interacción entre los alumnos, docentes y la aplicación en el ámbito de un aula con computadoras en una red LAN, motivando la programación del comportamiento de los robots que luego se pondrán a prueba en una batalla en común con todos los participantes. Todo esto se logra sin movernos de nuestra computadora, sin transferir archivos en forma explícita y de forma simple, permitiendo que los docentes administren

las batallas o también que grupos de alumnos puedan generar y administrar su propias batallas.

Rita en Red es una rama (branch) de un proyecto open source administrado a través de GitHub [https://github.com/maxirp9/RITA.git], decidimos administrar el proyecto de esta manera para no entorpecer el proyecto original de **Rita** hasta determinar que la versión de **Rita en Red** sea estable, luego se decidirá si se integra (merging) al proyecto original formando uno solo o se lo deja como un subproyecto de **Rita**. Siguiendo los principios de **Rita,** su código fuente es libre y desarrollado en Java. Por lo tanto, **Rita en Red** es portable, pudiendo ser ejecutado tanto en plataformas Windows como Linux.

7.1 Casos de uso

A continuación describimos las nuevas funcionalidades que se incorporaron a **Rita** para poder realizar la ejecución de las batallas en red.

El sistema posee dos actores principales que son el alumno y el docente, a través de los siguientes casos de uso mostramos las acciones que pueden realizar y que son desarrolladas con mayor detalle en la siguiente sección.
En la figura 18 se pueden observar los casos de uso del alumno.

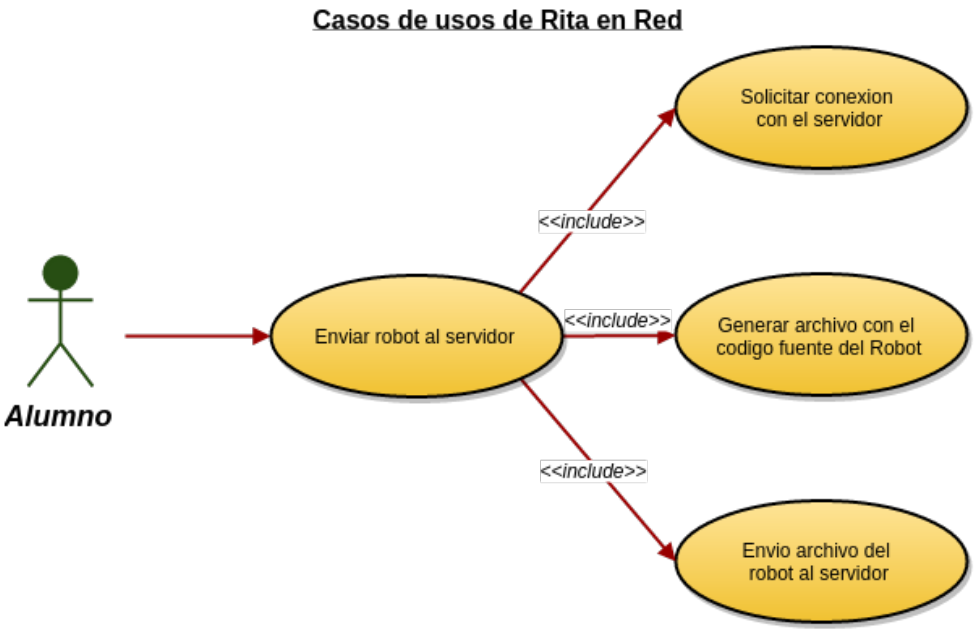


Figura 18 - Casos de uso del alumno

En la figura 19 se muestran los casos de uso del docente.

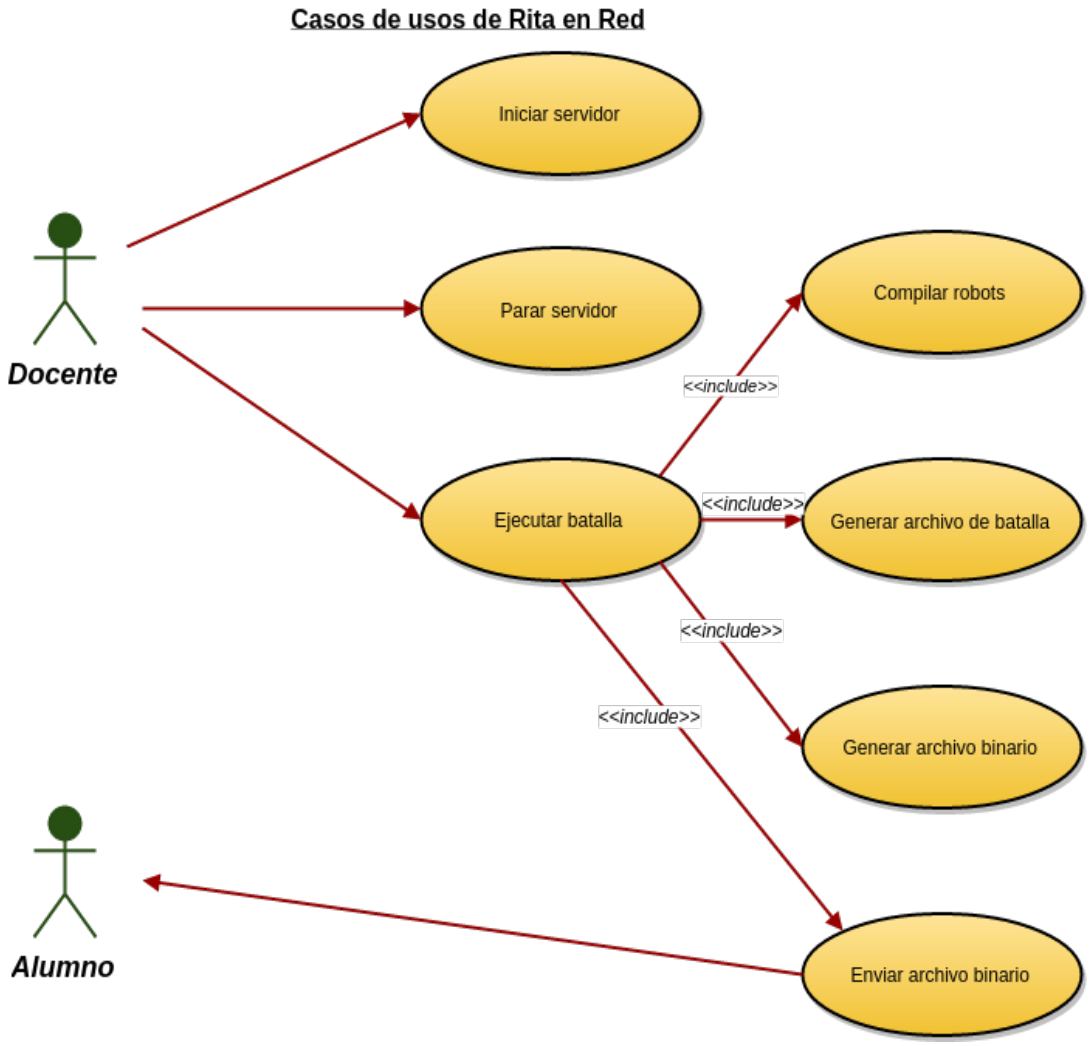


Figura 19 - Casos de uso del Docente

7.2 Diseño de la batalla en red

Cuando iniciamos con el análisis y diseño de la solución para poder extender RITA a una aplicación en la que se puedan compartir los robots y competir en red, estudiamos varias opciones. Para esto tuvimos que investigar sobre la herramienta en la que se ejecutan las batallas que es Robocode.

La primera opción fue investigar sobre una versión de ejecución distribuido[25], este fue un problema por la falta de documentación que hay en la red y la complejidad del mismo. Luego de descartar esta opción seguimos investigando la API de Robocode y encontramos 2 funcionalidades, una que nos permitió guardar la ejecución de cada turno de la batalla en un archivo de formato binario o XML y la otra es la posibilidad de reproducción de la batalla utilizando este archivo. De esta forma pudimos simular la ejecución en red de una batalla transfiriendo y reproduciendo el mismo archivo binario en distintas computadoras al mismo tiempo. Elegimos el formato binario por tener menor tamaño que el XML y de esta manera es más eficiente la transferencia de los archivos desde el servidor hacia los clientes.

Básicamente el diseño **Rita en red** consiste en el envío del un archivo binario generado por el servidor a cada uno de los clientes conectados para que luego éstos puedan reproducir la misma batalla en forma asincrónica en sus respectivas computadoras.

7.3 Arquitectura

Consideramos adecuada la arquitectura **cliente/servidor** para la implementación de **Rita en Red**. El servidor espera conexiones solicitadas por los clientes, cada cliente representa la presencia de un robot que participará de una batalla organizada por el servidor como muestra la figura 20.

La conexión se realiza usando sockets, para ésto se usó la implementación de Java en las clases **ServerSocket** y **Socket**, para el servidor y cliente respectivamente. Los sockets son conexiones que pertenecen a la *capa de transporte* del modelo OSI[26]. La comunicación se da bajo el protocolo orientado a la conexión**,** ya que el cliente pide una conexión al servidor,

[25] Distributed Robocode (s.f.). Recuperado de http://robowiki.net/wiki/Distributed_Robocode

[26] Modelo OSI - Open System Interconnection. (1980). (s.f.). En *Wikipedia*. Recuperado en Noviembre 2014 de: https://es.wikipedia.org/wiki/Modelo_OSI

cuando éste lo atiende permanece conectado y a la espera de las respuestas.

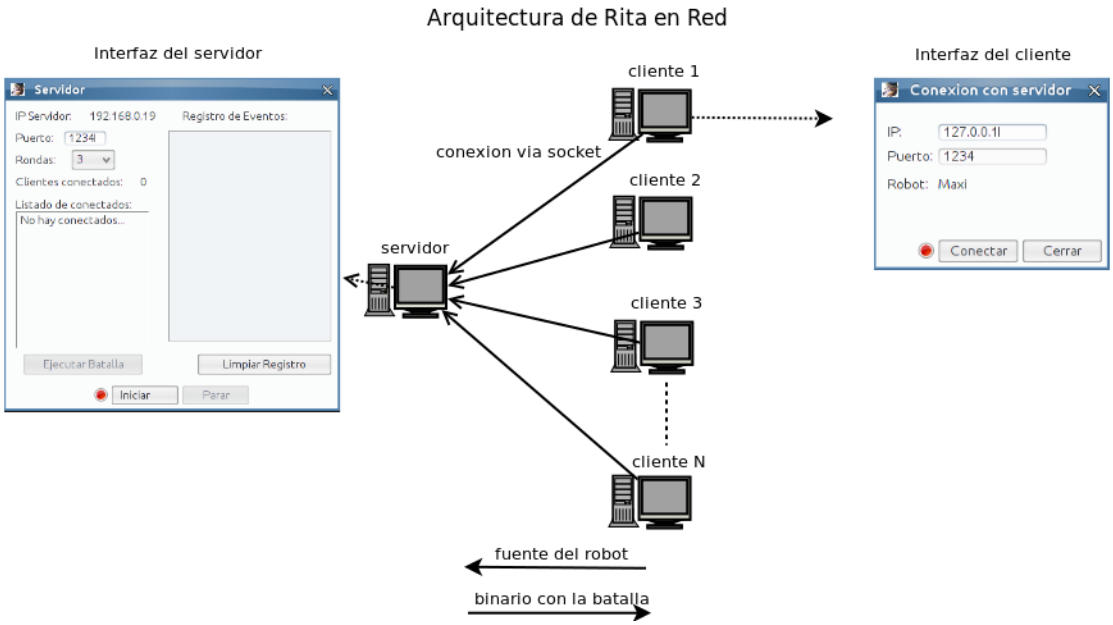


Figura 20 - Arquitectura de Rita en Red

En la figura 20, del lado izquierdo se puede observar la interfaz del servidor. Tanto el servidor como el cliente son hilos de ejecución diferentes implementados con la clase Thread de Java, el servidor crea diferentes hilos 1, 2, …, N para atender a cada cliente que se conecta de manera independiente, estos hilos llamados *worker* realizan toda la comunicación entre el cliente y el servidor. La interfaz de cliente se muestra a la derecha de la misma figura.

La tarea principal que se realiza a través de los sockets es la transferencia de archivos entre el cliente y el servidor, ilustrada en la figura 20 por medio de flechas en ambos sentidos. Estos archivos son de dos tipos diferentes, desde el cliente hacia el servidor viaja el archivo con el código fuente escrito en Java que contiene la lógica del comportamiento del robot y desde el servidor hacia los clientes se envía un archivo de formato binario con la ejecución de la batalla y un archivo con el resultado de la misma.

La figura 21 muestra el diagrama de clases de **Rita en Red** con algunas de las clases agregadas**.**

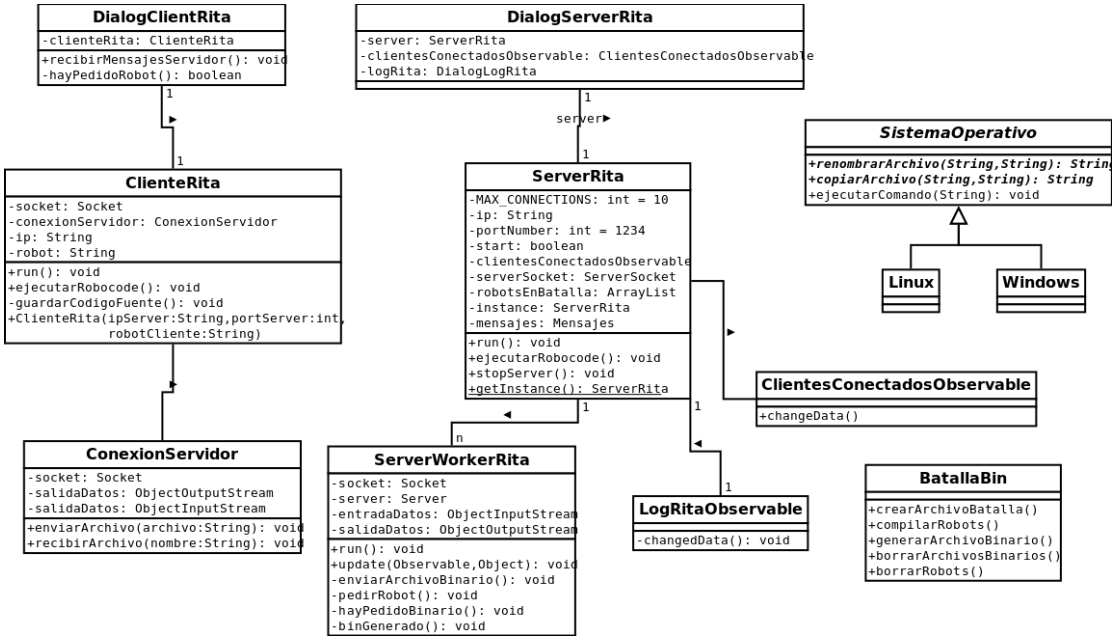


Figura 21 - Diagrama de clases agregadas a Rita en Red

En el diagrama de clases de **Rita en Red** se muestran algunas de las clases más importantes de la implementación. Las clases DialogClientRita y DialogServerRita que son las interfaces gráficas del servidor y cliente respectivamente, las vistas heredan de la clase JDialog e implementan la interfaz Observer y las clases ClienteRita y ServerRita que representan el modelo. Las vistas cambian cuando el modelo cambia según los eventos que sucedan. Las clases ClienteRita y ServerRita son hilos, heredan de la clase Thread y implementan el patrón Singleton para garantizar una única instancia. La clase ServerWorkerRita también hereda de Thread y representa cada conexión con un cliente, estos hilos son creados por la clase ServerRita a medida que se van aceptando conexiones. La clase SistemaOperativo es una clase abstracta que utilizamos para modelar la abstracción de múltiples sistemas operativos, en ésta mantenemos el registro de los métodos que son ejecutados por el SO para que cada una implementen según corresponda, en nuestro caso se crearon las clases Windows y Linux.

7.3.1 Servidor

El servidor es el encargado de gestionar la batalla entre los distintos robots. Las dos funciones principales son:

1. Aceptar las conexiones de cada cliente y recibir un archivo con el código fuente de su robot.

2. Ejecutar la batalla con todos los robots conectados y generar el archivo binario con el resultado de la misma, este archivo será enviado a cada uno de los clientes para que puedan reproducir la batalla.

Figura 22 - Servidor de RITA en Red

En la figura 22 se puede observar la interfaz del servidor donde es posible configurar el puerto al cual se conectarán los clientes y la cantidad de rondas que contará la batalla. También se puede visualizar la cantidad de clientes conectados y los nombres de los mismos.

Para iniciar una partida, el servidor lanza la ejecución con el botón “Iniciar” y se queda a la espera de solicitudes de conexión, cuando se conecten los clientes que quieren participar de una batalla compartida, se inicia la

ejecución de la batalla desde el botón “Ejecutar Batalla”. En la interfaz de usuario se pueden ir viendo los eventos que suceden en el Registro de Eventos, de esta manera es posible conocer cuando finaliza la ejecución de una batalla.

Cuando se inicia el servidor se crea una única instancia de la clase Servidor utilizando el patrón Singleton (Gamma E. et al, 1994), que hereda de la clase Thread de Java, su uso es de Main Thread, es decir, es el hilo principal para el manejo del servidor que luego administra todas las conexiones de los clientes que se conecten. Para instanciar la clase servidor se utilizan los parámetros cargados en la interfaz del servidor, el puerto por donde se va a crear la conexión y el número de rondas que se van a utilizar en la batalla que se ejecutará. Una vez instanciado el servidor éste ingresa en un bucle hasta la notificación de parada del mismo, a su vez utiliza otro bucle para la aceptación de conexiones por parte de los clientes como se observa en la figura 23.

```
String texto = "Servidor a la espera de conexiones.";
guardarLog(texto);
while (!shutDownFlag && true && (activeConnectionCount < MAX_CONNECTIONS)
                && !iniciarBatalla) {
        try {
                socket = serverSocket.accept(); // Aceptando las conexiones
                socketes.add(socket);
                socket.setKeepAlive(true);
        } catch (InterruptedIOException e) {
                log.error("Error: " + e.getMessage());
        } // try

        if (!shutDownFlag && !iniciarBatalla) {
                // SUMO SOLO LOS CONECTADOS
                setActiveConnectionCount(activeConnectionCount + 1);
                // Crea el objeto worker para procesar las conexiones
                ServerWorkerRita serverWorkerRita = new ServerWorkerRita(
                                socket, mensajes, clientesConectadosObservable, this);
                serverWorkerRita.start();
        }

} // end del While
```

Figura 23 - Aceptación de pedido de conexión de los clientes al servidor

Cuando un cliente se conecta, el servidor crea un socket y una instancia del objeto ServerWorkerRita que hereda de Thread de Java y le asigna la conexión. De esta manera se administran los clientes en hilos separados con una conexión sincrónica.

En el momento que se decide ejecutar la batalla presionando el botón “Ejecutar Batalla” el servidor recorre todos los robots que fueron enviados para participar y los compila. Después crea el archivo de configuración de la batalla con los robots, la cantidad de rondas parametrizadas y algunas propiedades básicas de inicio como el tamaño del campo de batalla, etc.

El siguiente diagrama de secuencias de la figura 24 mostramos como se lleva a cabo la ejecución de una batalla iniciada en el servidor.

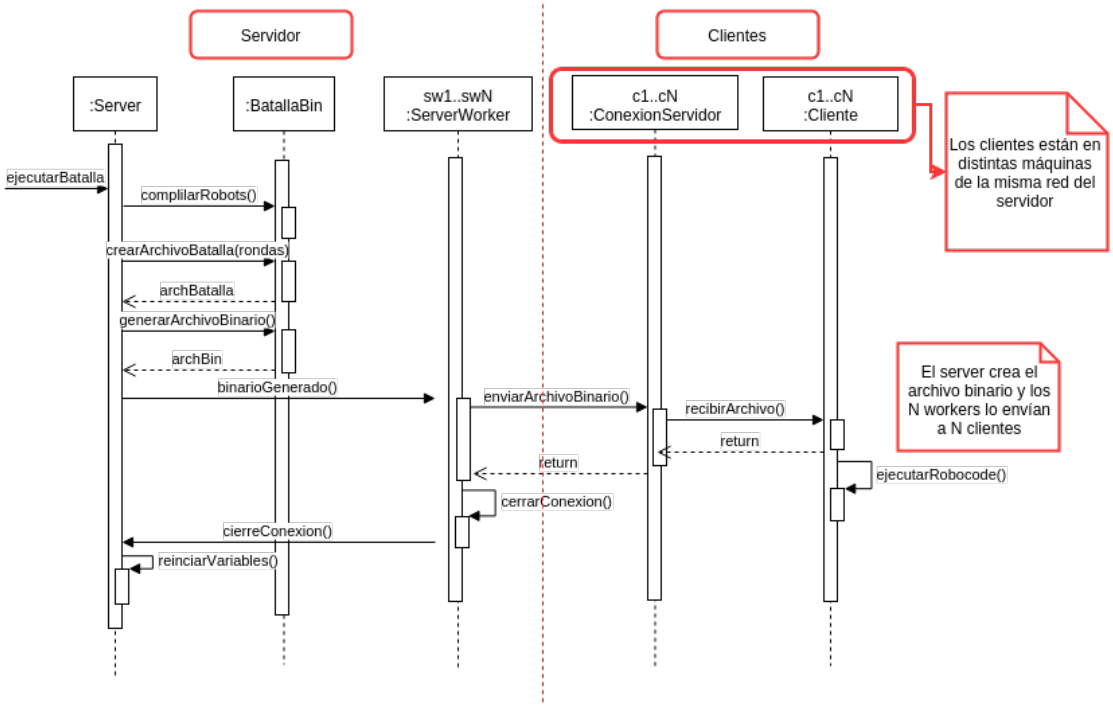


Figura 24 - Mensajes y clases que interactúan en la ejecución de una batalla iniciada por el servidor

7.3.2 Cliente

El cliente tiene como función principal, además de conectarse al servidor, el envío del código fuente con el comportamiento de su robot, esto lo realiza a través de una conexión por sockets instanciando la clase ConexionServidor. A través de esta clase se administran los mensajes que se envían y reciben entre cliente y servidor.

A continuación se muestran los métodos más importantes de la clase ConexionServidor:

- Método para enviar desde el cliente el archivo con el código fuente del robot al servidor:

```
public void enviarArchivo(String nombre){
        try {
                String fichero = Settings.getRobotsPath() + File.separator + Settings.getProperty("defaultpackage") + File.separator + nombre + ".java";
                boolean enviadoUltimo = false;
(1)             fis = new FileInputStream(fichero);

                // Se instancia y rellena un mensaje de envio de fichero
                MensajeTomaFichero mensaje = new MensajeTomaFichero();
                mensaje.nombreFichero = nombre;

                // Se leen los primeros bytes del fichero en un campo del mensaje
                int leidos = fis.read(mensaje.contenidoFichero);
                // Bucle mientras se vayan leyendo datos del fichero
(2)             while (leidos > -1) {
                        // Se rellena el numero de bytes leidos
                        mensaje.bytesValidos = leidos;
                        // Si no se han leido el maximo de bytes, es porque el fichero se ha acabado y este es el ultimo mensaje
                        if (leidos < MensajeTomaFichero.LONGITUD_MAXIMA) {
                                mensaje.ultimoMensaje = true;
                                enviadoUltimo = true;
                        } else {
                                mensaje.ultimoMensaje = false;
                        }
                        // Se envia por el socket
(3)                     salidaDatos.writeObject(mensaje);

                        // Si es el ultimo mensaje, salimos del bucle.
                        if (mensaje.ultimoMensaje) {
                                break;
                        }
                        // Se crea un nuevo mensaje
                        mensaje = new MensajeTomaFichero();
                        mensaje.nombreFichero = nombre;
                        // y se leen sus bytes.
                        leidos = fis.read(mensaje.contenidoFichero);
                }
                if (enviadoUltimo == false) {
                        mensaje.ultimoMensaje = true;
                        mensaje.bytesValidos = 0;
                        salidaDatos.writeObject(mensaje);
                }
                // Se cierra el ObjectOutputStream
                log.info("Termine de enviar el archivo del Robot al Servidor, soy Cliente: " + socket.getLocalAddress().getHostAddress());
        } catch (Exception e) {
                log.error("Error al crear el stream de salida : " + e.getMessage());
                e.printStackTrace();
        }

}
```

Figura 25 - Clase ConexionServidor, método enviarArchivo()

En la figura 25 mostramos el método enviarArchivo, en el punto 1 se observa que creamos una instancia de la clase FileInputStream para leer el código fuente java generado a partir de los bloques de Rita. Utilizamos la clase

MensajeTomaFichero para el envío de los bytes del código fuente a través de sockets por medio de la clase ObjectOutputStream como se ve en el punto 3, el envío de los bytes se realiza dentro un bucle hasta que se llega al final del archivo, marcado con el punto 2.

- Método para recibir en el cliente el archivo binario y el resultado de la batalla en formato de texto enviado por el servidor:

```
public void recibirArchivo(String nombre){
            try {

                    // Se abre un fichero para empezar a copiar lo que se reciba.
                    FileOutputStream fos = new FileOutputStream(nombre);

                    MensajeTomaFichero mensajeRecibido;
                    Object mensajeAux;
                    do {
                            // Se lee el mensaje en una variabla auxiliar
                            mensajeAux = entradaDatos.readObject();

                            // Si es del tipo esperado, se trata
                            if (mensajeAux instanceof MensajeTomaFichero) {
                                    mensajeRecibido = (MensajeTomaFichero) mensajeAux;
                                    // Se escribe en pantalla y en el fichero
                                    fos.write(mensajeRecibido.contenidoFichero, 0,
                                                    mensajeRecibido.bytesValidos);
                            } else {
                                    // Si no es del tipo esperado, se marca error y se termina el bucle
                                    log.error("Mensaje no esperado "
                                                    + mensajeAux.getClass().getName());
                                    break;
                            }
                    } while (!mensajeRecibido.ultimoMensaje);

                    log.info("Ya termino la trasferencia del BIN el cliente: "
                                    + socket.getLocalAddress().getHostAddress());
                    // Se cierra socket y fichero
                    fos.close();

            } catch (Exception e) {
                    log.error("Error al crear el stream de entrada : " + e.getMessage());
                    e.printStackTrace();
            }

}
```

Figura 26 - Clase ConexionServidor, método recibirArchivo()

El método recibirArchivo, que se ve en la figura 26, de la clase ConexionServidor utiliza la clase ObjectOutputStream para recibir los bytes que envía el cliente, punto 1. Recibe los bytes en un bucle hasta que llegue el último mensaje. La diferencia importante con el método enviarArchivo explicado anteriormente es que al recibir el mensaje, como son objetos, se verifica que sea de la instancia MensajeTomaFichero, como se ve en el punto 2.

La clase cliente hereda de la clase Thread, es un hilo que se ejecuta de forma asincrónica que se utiliza para interactuar con el servidor. El primer paso de la clase cliente es guardar en un archivo físico el algoritmo creado

por el alumno para luego poder enviarlo al servidor. Una vez enviado su robot, el cliente se queda a la espera del binario generado o de la detención por parte del servidor, como se ve en la figura 27, punto 1. Si recibe el mensaje de parada, el cliente vuelve al estado inicial para poder enviar de nuevo si lo desea el robot al servidor. Si recibe la notificación de que fue generado el binario, hace el pedido del mismo y se queda a la espera de recibir dos archivos, uno es la batalla en formato binario y el otro archivo que representa el resultado en formato texto, punto 2. Luego, si el cliente debe ejecutar la batalla, la ejecuta y reinicia las variables correspondientes para poder volver a estar listo para un nuevo envío, punto 3.
La imagen de la figura 27 mostramos parte del código de la clase ClienteRita.

```
try {
        Mensaje mensajeRecibido;
        mensajeRecibido = conexionServidor.recibirMensaje();

(1)     while (!mensajeRecibido.accion.equals("ParoServidor") && !mensajeRecibido.accion.equals("BinGenerado") && ventantaAbierta) {
                log.info("El Cliente "
                                + getMiDireccion()
                                + " recibe mensaje:"
                                + mensajeRecibido.accion);
                mensajeRecibido = conexionServidor.recibirMensaje();
        }

        log.info("El Cliente " + getMiDireccion() + " recibe mensaje:" + mensajeRecibido.accion);

        if(!mensajeRecibido.accion.equals("ParoServidor")){
                if (ventantaAbierta){
                        log.info("Ya esta el BinGenerado para el cliente: "
                                        + getMiDireccion());

                        Mensaje mensajeAEnviar = new Mensaje("DameBinFile", getMiDireccion());
                        conexionServidor.enviarMensaje(mensajeAEnviar);
                        log.info("Pide el binario el cliente: "
                                        + getMiDireccion());
                        DialogClientRita.startWait();
(2)                     conexionServidor.recibirArchivo(Settings.getBinaryPath() + File.separator + "batalla.copia.bin");
                        conexionServidor.recibirArchivo(Settings.getInstallPath() + File.separator + "resultado-batalla.txt");
                        log.info("Paso el resultado: "
                                        + getMiDireccion());

                if(this.ejecutarRobocode)
(3)                     ejecutarRobocode();
                        DialogClientRita.resetearBotones();
                        DialogClientRita.stopWait();
                }
        }else
        {
                log.info("PARA EL SERVIDOR");
                DialogClientRita.resetearBotones();
        }
} catch (NullPointerException ex) {
        log.error("El socket no se creo correctamente. ");
}
```

Figura 27 - Código de la clase ClienteRita

En la figura 28 mostramos el diagrama de secuencia del método que envía el código del robot al servidor.

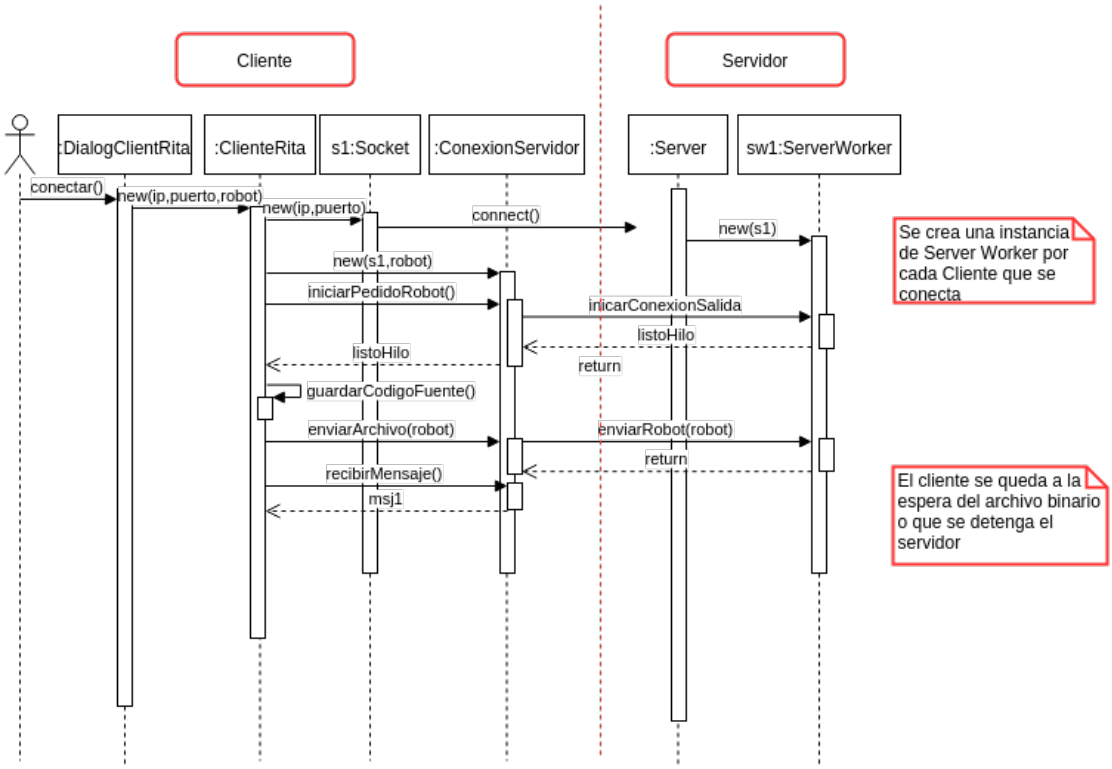


Figura 28 - Envío del robot desde cliente al servidor

El evento es disparado desde la interfaz del cliente a través del botón “Conectar”. El diagrama de secuencia anterior muestra los objetos y los mensajes que intervienen en el envío del código fuente del robot.

En la interfaz, mostrada en la figura 29, el botón “Conectar” permite realizar la conexión con el servidor y al mismo tiempo se envía el código fuente del robot, si la conexión es exitosa el semáforo se pone de color verde.

Figura 29 - Cliente de Rita en Red

7.3.3 Protocolo de comunicación

Para realizar la comunicación entre el cliente y el servidor creamos un protocolo de comunicación propio que consiste en el pasaje de mensajes entre los distintos actores de **Rita en Red**.
A continuación describimos los mensajes utilizados en el protocolo:

- ListoHilo: el servidor instancia una clase de ServerWorkerRita que hereda de Thread y el hilo avisa con este mensaje al cliente que está preparado para atenderlo. El ServerWorkerRita queda a la espera de mensajes.
- IniciarConexionSalida: el cliente acepta la conexión con el hilo ServerWorkerRita asignado por el servidor e inicia la transferencia del archivo con el código fuente del robot.
- IniciarBatalla: desde la interfaz del servidor se envía este mensaje que dispara el proceso de ejecución de una batalla con los robots enviados.
- BinarioGenerado: el servidor notifica a los ServerWorkers que el archivo binario con la batalla ya fue generado y puede ser enviado a los clientes.
- ParoServidor: el servidor con este mensaje es notificado desde la interfaz que se detenga y a su vez avisa a todas las conexiones de los clientes para que se cierren.
- DameArchivoBinario: el cliente solicita al servidor el archivo binario con la batalla generada y queda a la espera del mismo. Cuando termina de recibir el archivo binario ejecuta la batalla en Robocode.
- CerrarWorker: una vez que se envío el archivo binario con la batalla a cada cliente conectado, el worker notifica al servidor que finalizó su tarea.

7.3.4 Algunos aspectos de la implementación

7.3.4.1 Tecnologías de comunicación

Los sockets son un sistema de comunicación entre procesos de diferentes máquinas de una red. Más exactamente, un socket es un punto de comunicación por el cual un proceso puede emitir o recibir información.

Decidimos utilizar sockets para realizar la comunicación entre el servidor y el cliente. La comunicación consiste en el envío de mensajes y archivos entre los procesos para llevar adelante la tarea de **Rita en Red**. Esta decisión la

tomamos por ser la comunicación de sockets TCP de tipo orientada a conexión, lo que nos dio seguridad y estabilidad en la comunicación.

Para realizar la comunicación utilizamos las clase Socket y ServerSocket de Java incluidas en el paquete java.net, la clase ServerSocket es usada en el servidor para esperar las conexiones de cada cliente, este las recibe por medio del método accept(). La clase Socket la utiliza el cliente para establecer una comunicación con el servidor a través de una IP y un puerto donde el servidor esté escuchando. Para el envío de datos utilizamos las clases ObjectInputStream y ObjectOutputStream para la entrada y salida de datos respectivamente.

En **Rita en Red** cada conexión entre el servidor y un cliente es atendida por un hilo de ejecución debido al carácter sincrónico del tipo de conexión. A través de los mensajes enviados (ver protocolo de comunicación) entre sí, se va llevando adelante en forma secuencial el envío y recepción de los datos necesarios para realizar la tarea, tanto mensajes de sincronización como los archivos que representan el código fuente de los robots y el archivo binario con la batalla.

7.3.4.2 Métodos de sincronización

Uno de los problemas que enfrentamos fue la espera de los robots enviados por cada alumno para participar en la batalla. El servidor de **RITA en Red** realiza la espera de cada robot a través de un hilo de ejecución independiente que interactúa con cada alumno. Estos hilos se ejecutan en forma concurrente, por este motivo tuvimos que implementar algunos de los métodos de sincronización en el envío de mensajes (Vallejo Fernández D. , 2012), el código del robot y el resultado de la batalla.

Estas son algunas de las alternativas de sincronización:

- Semáforos
- Monitores
- Paso de mensajes
- Barreras(Barriers)

Para la sincronización se utilizaron dos métodos:

- **Paso de mensajes:** para la comunicación entre el servidor y los clientes por medio de los sockets. El servidor crea un hilo para atender a cada cliente llamado **ServerWorker,** éste se comunica con el cliente por medio de mensajes sincronizando la ejecución y envío de datos.

- **Barreras(Barriers):** se utilizan para sincronizar el envío de datos entre el servidor y los clientes, los datos enviados son los archivos con el código fuente de los robots de cada cliente y el archivo binario con la batalla ejecutada. Como la batalla no se ejecuta hasta recibir todos los robots aquí se creó una barrera para esperar todos los archivos de cada robots. Lo mismo pasa con el envío del archivo binario con la batalla para cada cliente, el servidor debe esperar hasta que todos los clientes reciban el archivo para terminar su ciclo de ejecución. En la figura 30 se muestra la sincronización por barreras utilizada en **Rita en Red.**

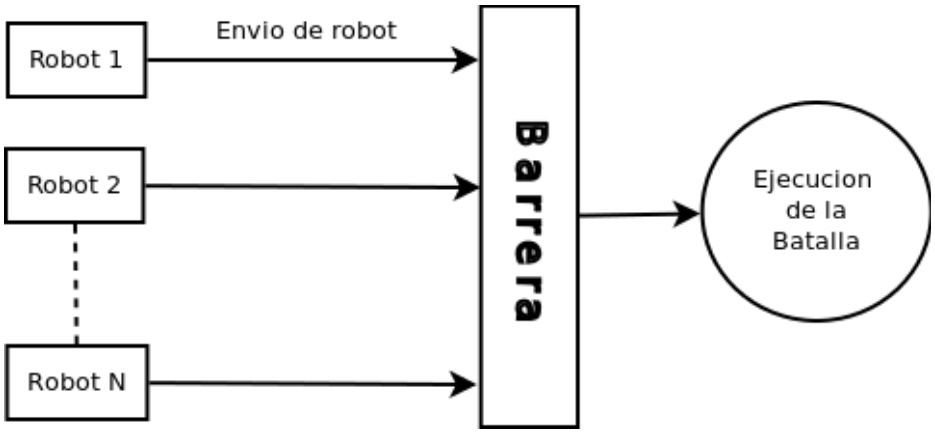


Figura 30 - Sincronización por barreras

7.3.4.3 Actualización de la interfaz gráfica

Para la comunicación entre la interfaz gráfica y el modelo de clases se utilizó el patrón de diseño Observer como se puede observar en la figura 31. El patrón Observer nos permite implementar una estrategia que reaccione a los cambios de estado en el objeto observado.

Utilizamos este patrón para la actualización de la interfaz gráfica del servidor ante los cambios generados con la entrada y salida de clientes conectados. (Grand M. , 2003). La clase DialogServerRita extiende de JDialog e implementa la interfaz Observer, esta clase ejecuta el método update()

cuando es notificado que un objeto que observa cambió su estado, esta notificación es realizada con los métodos setChanged() y notifyObservers(). Cuando un cliente se conecta al servidor, el servidor crea un hilo para atender a ese cliente con una instancia de la clase ServerWorkerRita, éste notifica que se agregó un cliente al listado de clientes conectados con el método addRobotNames(), este método ejecuta un changeData() del objeto observable y a su vez este método ejecuta un setChanged() y notifyObservers() avisando a la interfaz que cambió el objeto observado, de esta manera se actualiza el listado en la interfaz gráfica del servidor.

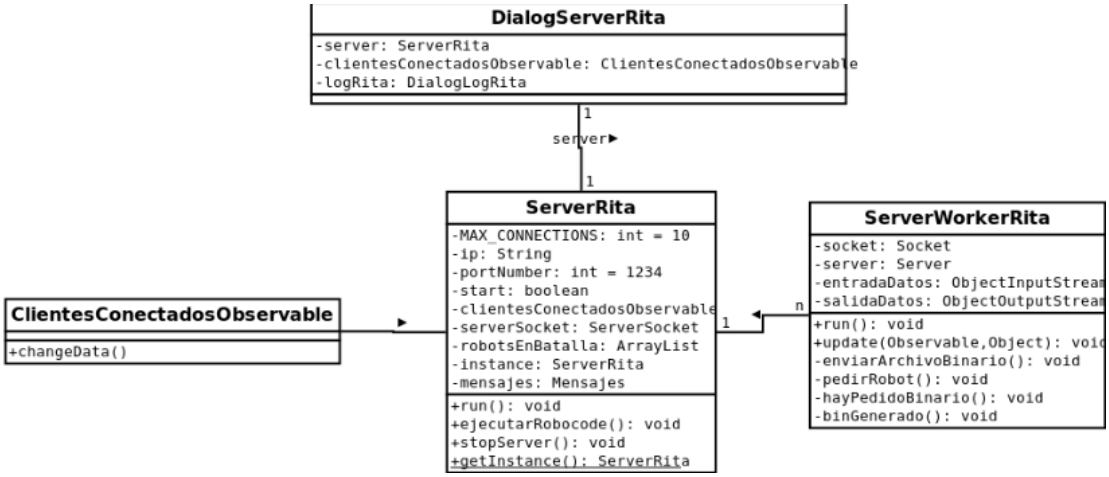


Figura 31 - Clases que intervienen en el patrón observer de rita en red

7.3.5 Tecnologías y herramientas utilizadas para el desarrollo

- Java
  - Sockets (Método de comunicación entre 2 programas en una red)
  - Swing (Librerías gráficas de Java)
- Eclipse - Java Integrated Development Environment (IDE) - https://eclipse.org
- GitHub - Hosting para Git - https://github.com/
- Maven 2 (herramienta de software para la gestión y construcción de proyectos Java) - http://maven.apache.org/
- Izpack (generador de programas instaladores de software de código abierto cuyo objetivo es la plataforma Java) - http://izpack.org/

7.3.5.1 Java Swing

Swing es una biblioteca gráfica para Java. Incluye widgets para interfaz gráfica de usuario tales como cajas de texto, botones, desplegables y tablas.

Es un framework MVC (Model View Controller) para desarrollar interfaces gráficas para Java con independencia de la plataforma. Sigue un simple modelo de programación por hilos y posee las siguientes características principales:

- Independencia de plataforma.

- Extensibilidad: es una arquitectura altamente particionada; los usuarios pueden proveer sus propias implementaciones modificadas para sobrescribir las implementaciones por defecto. Se puede extender clases existentes proveyendo alternativas de implementación para elementos esenciales.

- Personalizable: dado el modelo de representación programático del framework de Swing, el control permite representar diferentes estilos de apariencia "look and feel" (desde apariencia MacOS hasta apariencia Windows XP pasando por apariencia GTK+, IBM UNIX o HP UX entre otros). Además, los usuarios pueden proveer su propia implementación de apariencia, que permitirá cambios uniformes en la apariencia existente en las aplicaciones Swing sin efectuar ningún cambio al código de aplicación.

7.3.5.2 Eclipse

Eclipse es un entorno de desarrollo integrado, llamado también **IDE** (sigla en inglés de *integrated development environment*), está compuesto por un conjunto de herramientas de programación. Puede dedicarse en exclusiva a un sólo lenguaje de programación o bien puede utilizarse para varios. En nuestro caso se utilizo Eclipse Kepler para programar en Java integrando varias herramientas como Maven, GitHub, Izpack y frameworks de java como Swing. En la figura 32 se muestra el entorno de desarrollo de Kepler.

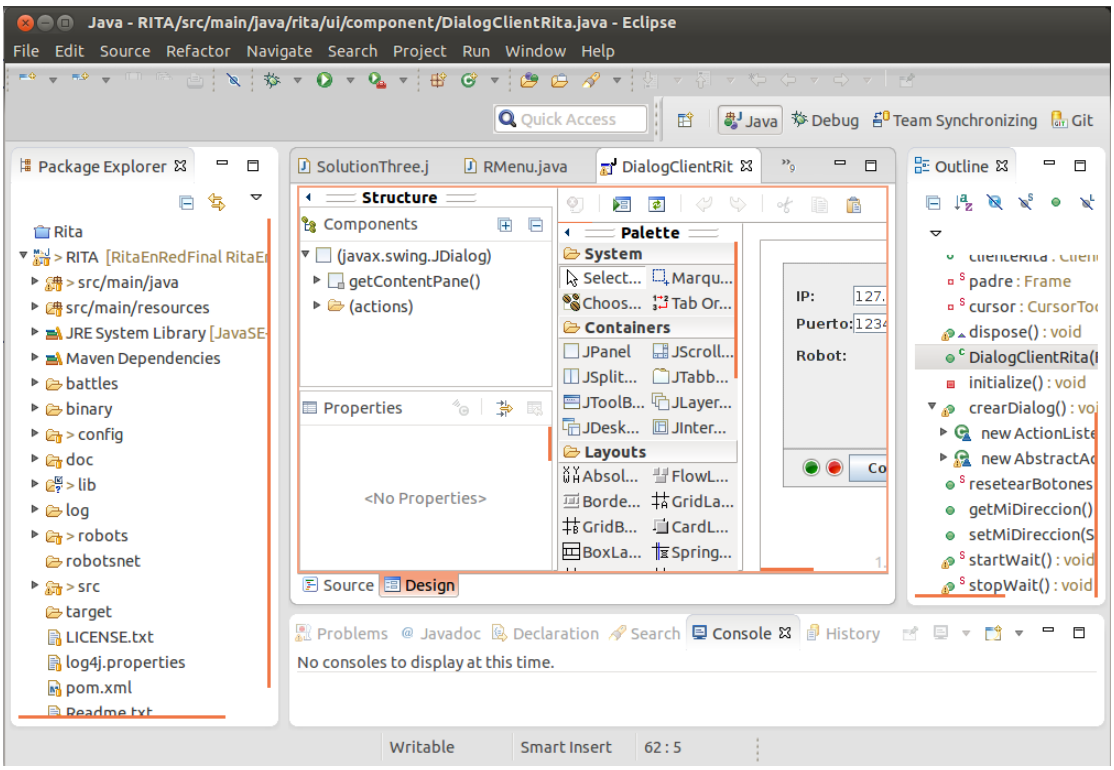


Figura 32 - Eclipse Kepler

7.3.5.3 GitHub

Plataforma de desarrollo colaborativo de software, usado para alojar proyectos, el cual utiliza el sistema de control de versiones **Git**.

Una de las prestaciones que hace especial a **GitHub**, es el hecho de alojar el repositorio de código en el que se trabaja de manera individual o en el equipo, evitando estar intercambiando estos archivos de computadora en computadora.

De las características más resaltables de GitHub para el control de versiones, podemos enumerar las siguientes:

- Tenemos una Wiki para el proyecto, para el intercambio y la documentación del proyecto.

- Contamos con un gráfico detallado, de cómo cada colaborador está trabajando en el proyecto y de las "*bifurcaciones*" que se van creando en el mismo.

- Podemos tener alojada una página web para la presentación del repositorio y/o proyecto, que tengamos alojados.

GitHub tiene dos funcionalidades importantes, fork y pull. El fork es la clonación de un repositorio que permite realizar libremente cambios sin afectar el proyecto original. El pull permite notificarle a otros usuarios los cambios que se realizamos a un repositorio.

7.3.5.4 Maven

Maven es una herramienta de software para la gestión y construcción de proyectos Java. Tiene un modelo de configuración de construcción simple, basado en un formato XML.

Maven utiliza un Project Object Model (POM) para describir el proyecto de software a construir, sus dependencias de otros módulos (figura 33) y componentes externos, y el orden de construcción de los elementos. Viene con objetivos predefinidos para realizar ciertas tareas claramente definidas, como la compilación del código y su empaquetado.

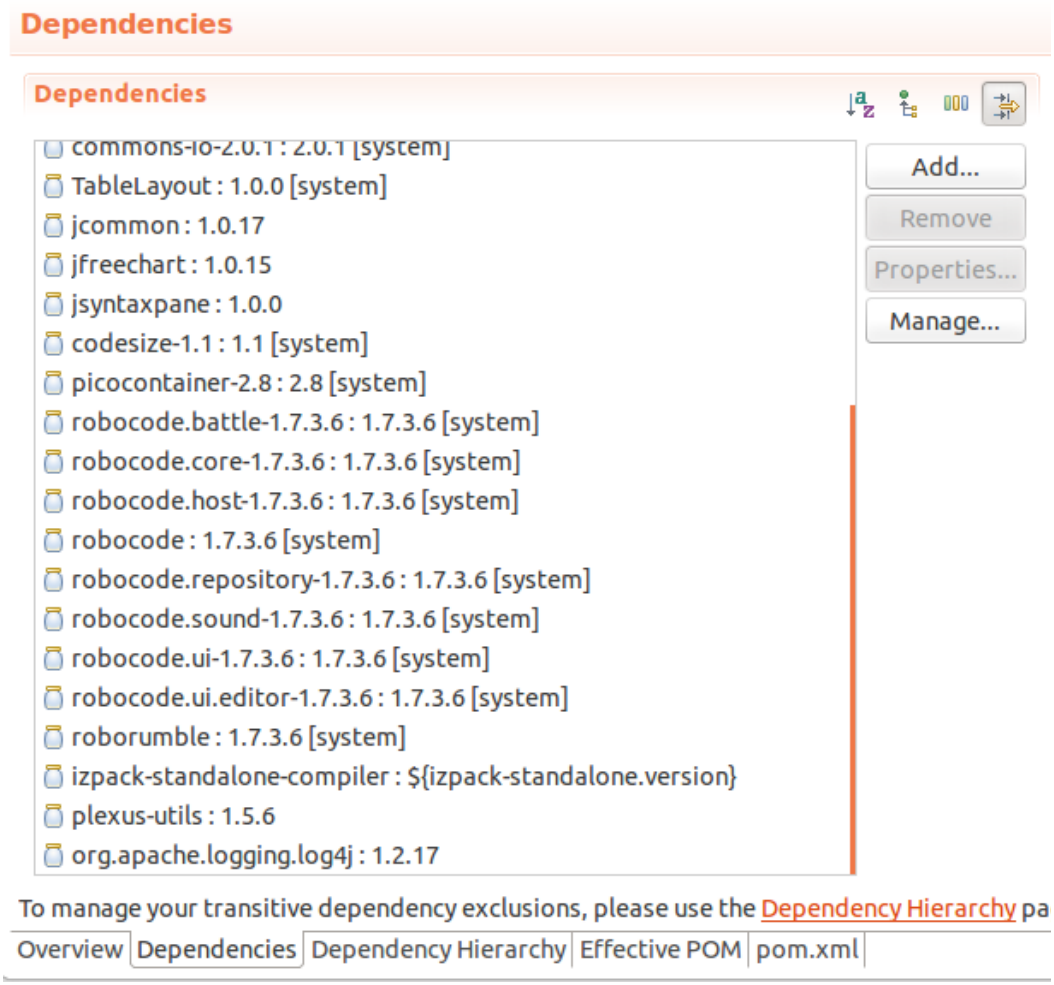


Figura 33 - Dependencias en Maven - Proyecto Rita en Red

Una característica clave de Maven es que está listo para usar en red. El motor incluido en su núcleo puede dinámicamente descargar plugins de un repositorio, el mismo repositorio que provee acceso a muchas versiones de diferentes proyectos Open Source en Java, de Apache y otras organizaciones y desarrolladores.

Maven está construido usando una arquitectura basada en plugins que permite que utilice cualquier aplicación controlable a través de la entrada estándar.

Las partes del ciclo de vida principal del proyecto Maven son:

1. *compile*: genera los ficheros .class compilando los fuentes .java
2. *test*: ejecuta los test automáticos de JUnit existentes, abortando el proceso si alguno de ellos falla.
3. *package*: genera el fichero .jar con los .class compilados
4. *install*: copia el fichero .jar a un directorio de nuestro ordenador donde maven deja todos los .jar. De esta forma esos .jar pueden utilizarse en otros proyectos maven en el mismo ordenador.
5. *deploy*: copia el fichero .jar a un servidor remoto, poniéndolo disponible para cualquier proyecto maven con acceso a ese servidor remoto.

Existen plugins de Maven para crear archivos de configuración del **IDE** a partir de los POMs. Nosotros utilizamos el plugin para **Eclipse** para **Maven 2** en el proyecto.

7.3.5.5 Izpack

Izpack es una amplia herramienta usada para empaquetar aplicaciones sobre plataforma Java. Es de fácil instalación y multiplataforma ya sea Windows, Linux, Solaris o Mac OS. Nosotros lo utilizamos para crear el instalador de Rita en Red, este está disponible para Windows (7, 8) o Linux (Debian, Ubuntu, Huayra).

7.4 Dificultades en el desarrollo

A través de las distintas etapas del desarrollo tuvimos que afrontar varias dificultades que pudimos superar, algunas con respecto a la solución del problema y otras con las tecnologías seleccionadas. Estas son algunas de ellas:

- Robocode distribuido: al comienzo investigamos sobre la existencia de una versión que permita la ejecución en red de batallas entre varios robots, pero la poca documentación sobre el tema nos dificultó la búsqueda. Además la supuesta solución no era la que nosotros necesitábamos ya que era demasiada compleja y nunca se pudo poner en funcionamiento.

- Comunicación de Rita en red: Nos encontramos con la dificultad de tener que enviar mensajes entre las distintas instancias de Rita en Red ejecutadas en diferentes máquinas para sincronizar el envío y recepción de archivos que utiliza la aplicación para la simulación de la ejecución en red de la batalla. Decidimos utilizar el método por **barreras** para resolver esta dificultad.

- El proyecto **Rita** utiliza el versionado de código provisto por GitHub, al comienzo tuvimos inconvenientes por ser una herramienta nueva que no habíamos utilizado. Luego de haber pasado por la curva de aprendizaje normal de una herramienta pudimos aprovechar las ventajas de la misma en el trabajo en equipo con respecto al versionado por SVN.

- Una dificultad que debimos afrontar fue la creación de un paquete para la instalación y ejecución de **Rita en Red** en diferentes sistemas operativos de plataformas Windows y Linux. Esto se resolvió con la herramienta **Izpack** investigando los distintos parámetros de configuración para diferentes sistemas operativos.

7.5 Uso de RITA en Red

7.5.1 Uso del cliente

Si un usuario quiere competir por red con sus compañeros debe presionar el botón de “Jugar en Red” (figura 34 indicado en rojo) o desde el menú “Red” y después “Cliente” (figura 34 indicado en negro)

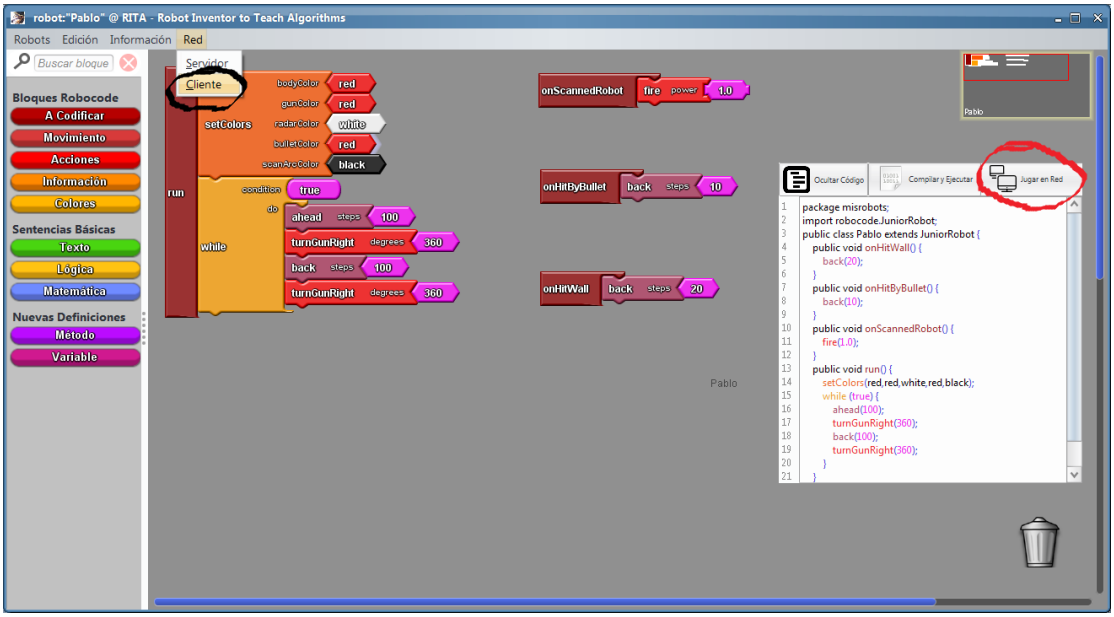


Figura 34 - Área de trabajo de Rita en red

Una vez seleccionado de alguna de las dos formas para jugar en red con uno o más compañeros debe conectarse a través de **RITA en Red** a un servidor por medio de la dirección IP del mismo. Esto lo realiza cuando le aparece la ventana del cliente (figura 35)

Figura 35 - Interfaz del cliente

Se debe ingresar la dirección IP, como así también el puerto, que le indique el usuario que está haciendo de servidor para esta batalla, una vez ingresados debe presionar el botón “Conectar”. Ya conectado se puede ver en la interfaz una imagen en verde de que se encuentra “en línea”. Para competir debe esperar que le llegue automáticamente la batalla con los rivales que participan en la misma. El cliente ve directamente la batalla una vez que es enviada por el servidor**.**

7.5.2 Uso del Servidor

Para poder utilizar **Rita en Red** como servidor se debe presionar en el menú “Red” y luego “Servidor” (figura 36).

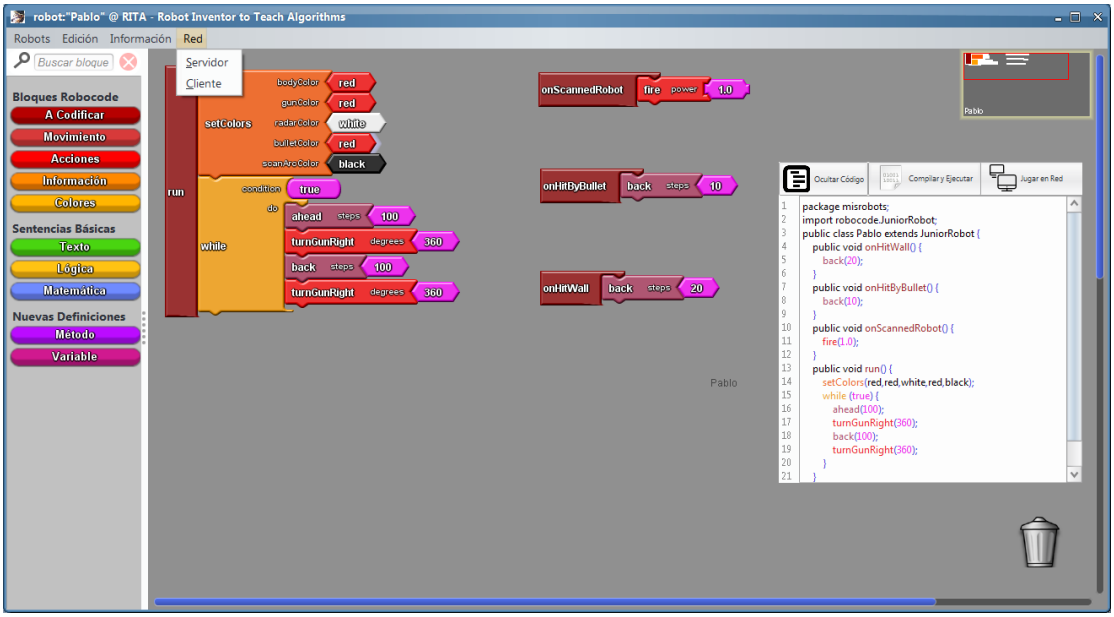


Figura 36 - Área de trabajo de Rita en red

En la interfaz del servidor se puede configurar la opción de la cantidad de rondas que va durar la batalla. También se puede configurar el puerto por el cual se van a conectar los clientes con el servidor (figura 37).

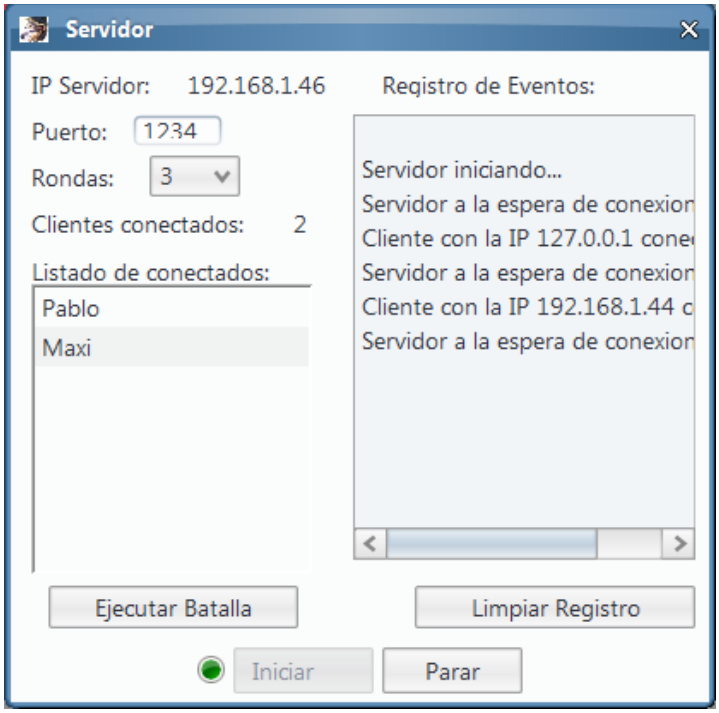


Figura 37 - Interfaz del servidor

Además de las configuraciones anteriores hay una serie de informaciones que resultan útiles para conocer qué es lo que pasa con el servidor. Esta información es: la cantidad de clientes conectados con el servidor, el nombre de los clientes conectados y además un registro con los eventos que se van sucediendo en el servidor. También se puede limpiar el registro si es que ha crecido mucho.

Algunos mensajes que se ven en el registro de eventos son si el servidor está listo, qué cliente se conecta desde qué IP, qué cliente envió el robot, archivos creados para configuración, etc.

Las funciones más importantes de la pantalla del servidor son el “Iniciar”, el cual arranca el servidor para que se puedan conectar los clientes y el “Ejecutar Batalla” que genera un archivo binario con la ejecución de la batalla de acuerdo a la cantidad de robots enviados por los alumnos y a la cantidad de rondas definidas por el docente para luego enviársela a cada cliente, luego cada cliente la reproduce para ver el resultado.

8. Experiencia en el aula

En el marco del proyecto de extensión **“Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica” (JETs)** se planificó un encuentro en la escuela de Enseñanza Secundaria Media N°12 “Manuel B. Gonnet”. El mismo se realizó el viernes 8 de mayo de 2015 con alumnos de cuarto y quinto año con edades que van desde los 15 a los 17 años.

Los alumnos que participaron en la actividad en su mayoría fueron participantes de la experiencia de **RITA** en el año 2014 por lo tanto tenían los conocimientos básicos de la herramienta.

El encuentro estuvo dividido en dos etapas:

1. Una presentación de qué es **Rita**, un repaso de los distintos bloques de movimiento, lógica e información. También de estrategias básicas para programar el robot y por último **Rita en Red.**

2. Un desafío grupal donde los alumnos formaron equipos de a dos para programar la estrategia de su robot. Una vez desarrollada, se unían a una red para competir entre cuatro equipos y luego los ganadores debían enfrentarse en semifinal hasta llegar a la final. Además al terminar esas semifinales se hizo una batalla con todos los robots creados en una única batalla.

En la actividad participaron Vanessa Aybar Rosales, Matías Brown Barnetche, Isabel Kimura y Claudia Queiruga. Nos encargamos de que los alumnos tuvieran asistencia para sacarse cualquier duda que les surgiese para diseñar el comportamiento del robot, como así también del funcionamiento de **Rita en Red**.

Los alumnos que participaron de la actividad demostraron mucho interés en poder realizar distintas pruebas de los movimientos de defensa y ataque con su robot, tratando de construir la mejor estrategia intentando ganarle a los demás robots que participen de la batalla.

El armado de los grupos que compitieron en la actividad se hizo dividiendo a los alumnos en 3 redes diferentes utilizando 2 routers llevados por el equipo de trabajo y la red brindada por el colegio, todos de manera inalámbrica. En cada red se conectaron entre 3 y 4 robots para participar de la batalla que se realizó en 3 rondas. Los ganadores de cada red luego se enfrentaron en una competencia final que se mostró con un proyector. El ganador fue el robot llamado “Desapitodominaargentina”.

Además se realizó una batalla con todos los robots, en esa contienda se utilizó la red del colegio. La batalla también fue proyectada y el ganador fue el mismo robot “Desapitodominaargentina”.

Los equipos que se utilizaron para este trabajo de campo fueron las netbooks del plan **conectar igualdad** de los alumnos y se utilizaron tanto en Sistema Operativo Windows como Linux**.**

La preparación de la actividad consistió en acercarse al establecimiento unos días antes para la preparación del aula multimedia. Esto requirió que el docente le solicite a los alumnos sus netbooks del plan conectar igualdad para instalarles Rita en red en las mismas, tanto en el sistema operativo Windows como en Huayra(Linux).

Una vez instalada la aplicación se hicieron pruebas de la red wifi del aula y determinamos que era conveniente para agilizar la actividad, llevar 2 routers adicionales por la sobrecarga de dispositivos que hacían lenta la transferencia de los datos. De todos modos en el momento de realizada una de las actividades utilizamos únicamente la red del aula y se obtuvieron resultados satisfactorios.

En el caso de la batalla con todos los robots se utilizó una notebook de uno de los compañeros para mostrarla con el proyector.

A continuación mostramos algunas fotos de la actividad realizada.

Figura 38 - Escuela de Enseñanza Secundaria Media N°12 “Manuel B. Gonnet”

Figura 39 - Alumno programando su robot

Figura 40 - Trabajando en equipo con Rita en Red

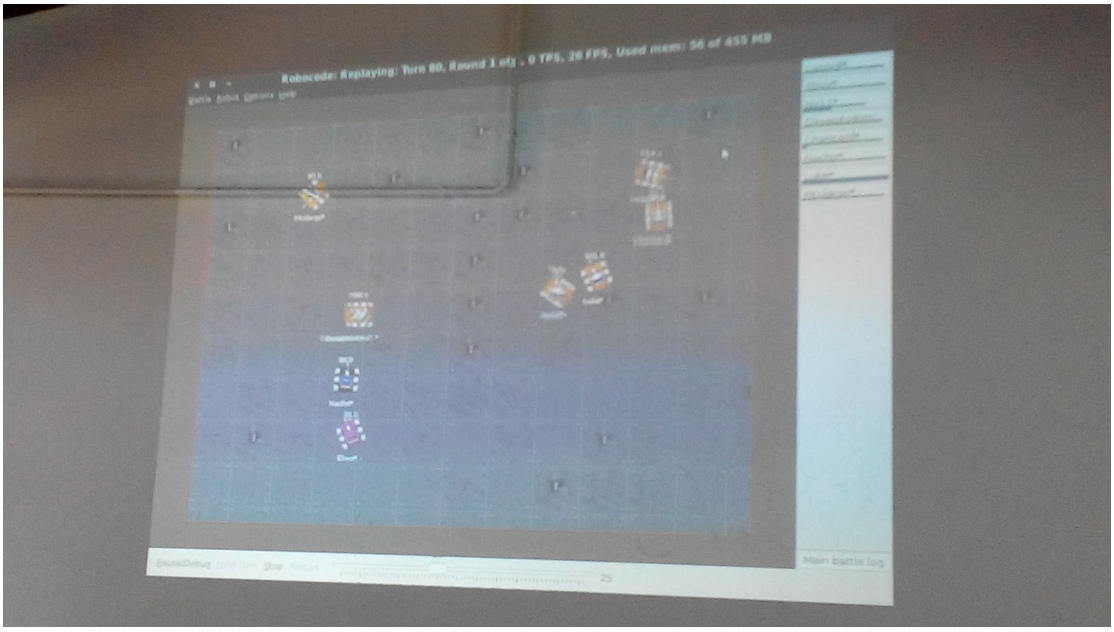


Figura 41 - Batalla final de todos los robots

El robot “Desapitodominaargentina” que fue el ganador de todas las batallas tenía el siguiente código java generado (figura 42):

Desapitodominaargentina.java

```
1   package misrobots;
2   import robocode.JuniorRobot;
3   public class Desapitodominaargentina extends JuniorRobot {
4       public void onHitByBullet() {
5           turnGunTo(scannedAngle);
6           fire();
7           back(20);
8       }
9       public void onHitRobot() {
10          turnTo(hitByBulletAngle);
11          if ( 50 > energy ) {
12              fire();
13          }
14          else {
15              if ( 40 > energy ) {
16                  fire();
17              }
18              else {
19                  if ( 20 > energy ) {
20                      back(200);
21                  }
22              }
23          }
24      }
25      public void onHitWall() {
26          back(150);
27      }
28      public void onScannedRobot() {
29          turnGunTo(scannedAngle);
30          fire(3.0);
31          turnTo( scannedAngle - 90 );
32          ahead(200);
33          fire();
34      }
35      public void run() {
36          setColors(orange,blue,white,yellow,black);
37          while (true) {
38              ahead(100);
39              turnGunRight(360);
40              back(100);
41              fire();
42              turnTo(180);
43              turnGunRight(360);
44              ahead(100);
45          }
46      }
47  }
48
```

Figura 42 - Código del robot ganador Desapitodominaargentina

Una vez finalizada la actividad entregamos a los alumnos una encuesta que tuvo como objetivo recolectar información sobre la experiencia en el aula utilizando **Rita en Red**. La cantidad de alumnos que encuestamos fue de 20,

con una edad promedio de 16 años y realizamos 7 preguntas sobre la experiencia realizada.

La encuesta fue creada con el propósito de poder obtener información sobre las nuevas funcionalidades presentes en **Rita en Red**.

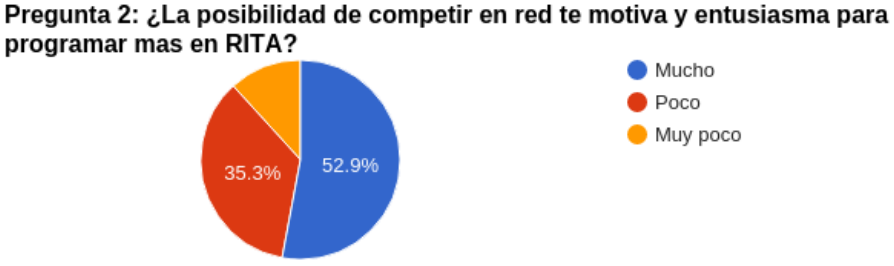
La orientación de las preguntas fueron hechas con distintas perspectivas, una de ellas era saber la complejidad en el uso de **Rita en Red**, teniendo en cuenta que los alumnos debieron incorporar nuevos conceptos como cliente, servidor, dirección IP, puerto, etc.

Otro objetivo de la encuesta fue ver si hubo un aumento en cuanto a la usabilidad, la motivación y entusiasmo en los alumnos utilizando **Rita en Red** con respecto a su versión anterior.

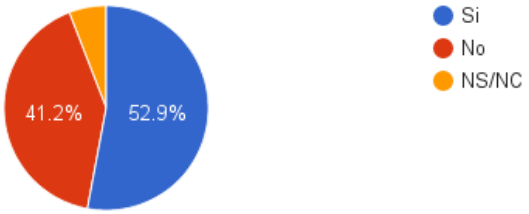
Además hicimos consultas con respecto a la eficiencia del sistema preguntando el nivel de conformidad de los tiempos de respuesta para obtener el resultado de la batalla una vez que había sido enviado el robot para participar.

Como pregunta final se le pidió a los alumnos cómo calificarían su experiencia con **Rita en Red** y a su vez qué cambiarían para mejorar la experiencia.

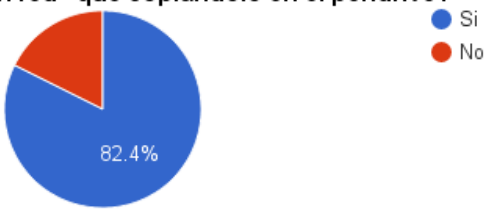
La misma arrojó los siguientes resultados:



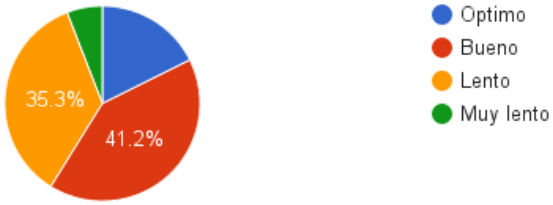
Pregunta 3: ¿Te parece que Rita en Red esta buena para compertir con tu amigos o compañeros fuera del aula?



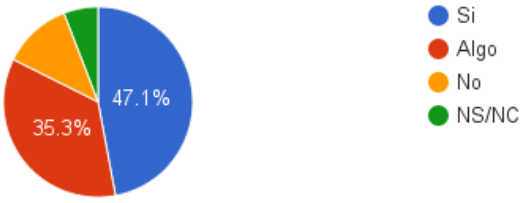
Pregunta 4: ¿Te resultó más cómodo enviar tu robot a competir usando la nueva opción "Jugar en red" que copiándolo en el pendrive?



Pregunta 5: ¿Cómo calificarías el tiempo que esperaste para ver el resultado de la batalla?



Pregunta 6: Rita en Red tiene la opción Cliente y la opción Servidor. ¿Entendiste la diferencia?



Resultados de la encuesta en la escuela secundaria media Nº12 "Manuel B. Gonnet"

Los resultados de la encuesta reflejan distintas conclusiones, con respecto a la complejidad del uso de **Rita en Red. S**e ve claramente que tiene una complejidad de uso medio de 82,4% con respecto al fácil 17,6% y ninguno de los alumnos la calificó de difícil o imposible. Estos resultados demuestran que **Rita en Red** puede ser utilizada sin inconvenientes por alumnos de escuelas secundarias con la ayuda de algún docente que los guíe en el proceso de aprendizaje.

De acuerdo al resultado de la pregunta 2 se aprecia que el 52.9% de los alumnos los motiva y entusiasma mucho programar con la nueva funcionalidad de **Rita en Red**, mientras que el 35.3% le entusiasma poco, y un 11.8% muy poco. Los resultados de esta pregunta nos indica que con la nueva funcionalidad a más alumnos los entusiasma utilizar Rita.

Los resultados de la pregunta 3 nos marcan que el 52.9% de los alumnos competirían con otros amigos o compañeros fuera del aula y un 41.2% no lo haría, mientras que un 5.9% no contestó la pregunta. Estos resultados indican que los alumnos interesados en la programación podrían utilizar **Rita en Red** fuera del aula al ser más fácil competir con sus compañeros.

Con respecto a la pregunta 4 nos indica claramente, con un porcentaje de 82.4%, que a los alumnos les resultó mucho más cómodo enviar el robot a competir usando la nueva funcionalidad de "jugar en red" que utilizar un pendrive, con un porcentaje de 17.6%. La utilización del pendrive para enviar los robots quedaría para el caso de escuelas que no dispongan de red o no estén funcionando en ese momento.

El resultado de la siguiente consulta es sobre el tiempo de espera para ver el resultado de la batalla, la mayoría indicó con un 41.2% que fue bueno, un 35.3% dijo que fue lento, un 17,6% indico que fue óptimo y un 5,9% dijo que fue muy lento. Según estos resultados se puede deducir que los alumnos quedaron conformes con el tiempo de respuesta de la aplicación.

La pregunta número 6 corresponde a si los alumnos pudieron comprender los conceptos de cliente y servidor en el transcurso de la actividad, un 47,1% entendieron los conceptos, mientras que un 35,3% dijo que entendieron a medias, y el resto, un 17,6% no los comprendieron. Estos resultados indican que la mayoría de los alumnos entendieron los conceptos y esto ayudó a realizar las actividades de manera correcta, distribuyéndose bien los roles entre los alumnos. Entendiendo que el alumno que organiza la batalla es el servidor y el que participa es el cliente.

La última pregunta refiere a cómo calificarían la experiencia que tuvieron con **Rita en Red** y qué cambiarían. La misma fue de texto libre y la mayoría apuntó que el sistema debería tener la opción de idioma en español en los bloques que se utilizan para darle comportamiento al robot, esto ayudaría a los alumnos a enfocarse a definir el algoritmo de su robot. Un alto porcentaje dijo que es “bueno y divertido” programar con **Rita en Red** por la posibilidad de poder jugar y competir con sus compañeros en batallas de robots.

Algunos alumnos escribieron que los ayudaría a realizar batallas más rápidamente si tuviesen más opciones simples para programar. De esta respuesta pensamos que es posible crear bloques de comportamiento como “girar en círculos” o “mover en diagonales”, etc. para que los alumnos tengan más opciones para hacer movimientos sencillos.

Otra cuestión que demostraron los alumnos en menor medida en el resultado de esta pregunta fue la disconformidad con la gráfica del sistema. Sería recomendable en un futuro realizar una actualización de la gráfica del sistema.
De los resultados de la encuesta podemos decir en términos generales que los alumnos tuvieron una buena aceptación de **Rita en Red** y nosotros junto a los compañeros, profesores y autoridades del colegio quedamos conformes con el resultado de la experiencia realizada.

9. Conclusión

Consideramos gratificante para nuestra formación como futuros profesionales haber podido contribuir en un proyecto de la Facultad de Informática de la UNLP que tiene como objetivo ayudar a introducir la programación en las escuelas secundarias conociendo la importancia que tiene ésta en el desarrollo de la industria de software nacional y la generación de personal calificado que puedan realizar trabajos con valor agregado.

Otro aspecto que destacamos es haber podido continuar con el trabajo de tesis realizado por **Vanessa del Carmen Aybar Rosales** en el proyecto *"Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica*", sabiendo los resultados que se obtuvieron en su implementación en las escuelas secundarias donde fue probado. Nuestro aporte apunta en la misma línea que el proyecto mencionado, con el objetivo que **Rita en Red** ayude a ser más simple la enseñanza de las programación en los jóvenes.

RITA nació como un proyecto de software libre y **Rita en Red** sigue los mismos principios con el objetivo de dejar para toda la comunidad educativa una aplicación que pueda ser utilizada en la enseñanza y promoción de la programación en alumnos de los niveles iniciales y medios.

Por último queremos comentar sobre el trabajo de campo realizado en las escuelas secundarias, quedamos muy conforme con los resultados y la aceptación de la nueva característica de **Rita en Red** que permite a los alumnos probar sus robots con el de otros compañeros en forma más simple y cómoda. Esto permite detectar con más rapidez que alumnos tienen más facilidad para desarrollar algoritmos que den comportamiento a su robot, y a la vez que alumnos necesitan ayuda y apoyo en áreas de conocimiento como lógica y matemáticas. Otro resultado interesante fue la interacción entre los alumnos aprovechando la independencia que da la aplicación en la gestión de las batallas como también una mejor organización por de parte de los docentes en el aula.

9.1 Mejoras a futuro

- Para fomentar una mayor participación de los alumnos en la competencia e incentivarlos a programar se podría agregar la funcionalidad de la creación de torneos. Los mismos deberían permitir agregar una cantidad de participantes y generar un torneo ya sea de eliminación directa como así también de un todos contra todos, mostrando los partidos y resultados, la tabla de posiciones, estadísticas, etc.

- Aprovechando la nueva funcionalidad proporcionada por **Rita en Red,** un comportamiento que le agregaría valor a la aplicación sería la posibilidad de poder crear equipos de robots, un equipo es una colección de robots que trabajan juntos para derrotar al enemigo. Para lograrlo se debería sumar un nuevo tipo de robot, este extenderá de la clase TeamRobot, para proporcionar la comunicación entre ellos, pues esta clase contiene los métodos para enviar y recibir mensajes entre robots.

- Agregar plantillas para las diferentes áreas de comportamiento del robot para facilitar la programación a los alumnos con nivel inicial permitiendo a partir de las mismas ir agregando mayor complejidad al diseño del algoritmo.

- De acuerdo al resultado de la encuesta se podría realizar la traducción al español de algunas partes de código y agregar la opción de selección de idioma.

- Desde el punto de vista tecnológico, se podrían mejorar temas de implementación, tiempos de respuestas, transferencia.

10. Referencias bibliográficas:

Aybar Rosales Vanessa del Carmen (2012). **Aplicaciones complementarias a ROBOCODE que faciliten el aprendizaje de programación en escuelas secundarias**. (Tesis de grado) Universidad Nacional de La Plata, La Plata.

Berners-Lee Tim . (5 de febrero,2013). ***Pc World Professional España***, Recuperado de: http://www.pcworld.es/business-ti/saber-programacion-es-la-nueva-brecha-digital-segun-bernerslee

Diaz Javier, Banchoff Tzancoff Claudia, Queiruga Claudia, Martín Sofia (2014). ***Experiencias de la Facultad de Informática en la Enseñanza de Programación en Escuelas con Software Libre.*** Publicado en Memorias del Congreso Iberoamericano de Ciencia, Tecnología, Innovación y Educación, Buenos Aires, Argentina, noviembre de 2014. Recuperado de: http://www.oei.es/congreso2014/memoriactei/1426.pdf

Eckel Bruce . (2003). **Thinking in Java.** Prentice Hall Professional. Cuarta Edición

Facultad de Informática de la Universidad Nacional de La Plata. (2012). **Articular universidad-escuela con JAVA para fortalecer la Educación-Técnica (JETs).** http://jets.linti.unlp.edu.ar/

Gamma Eric, Helm Richard, Johnson Ralph y Vlissides John (1994). ***Design Patterns: Elements of Reusable Object-Oriented Software.*** Pearson Education (p.144)

Grand Mark . (2003). **Patterns in Java, Volume 2.** Wiley India Pvt. Limited.

Jeannette Wing. (2006, Mar.). **Computational thinking. Communications of ACM.** Vol 49 Nº 3, (p.33–35).

Klopfer Eric , Wendel Daniel , Roque Ricarose , McCaffrey Corey , Ye Lunduo, Ho Aidan, Warne Brett, Liu Xudan, Nga Hout . (2009). **Openblocks.** Recuperado de: http://education.mit.edu/openblocks

Mathew A. Nelson. (2000). ***Robocode: Build the best - destroy the rest!*** Recuperado de: http://robocode.sourceforge.net/

Queiruga Claudia, Fava Laura, Banchoff Tzancoff Claudia, Aybar Rosales Vanessa, Miyuki Kimura Isabel, Brown Bertneche Matías.(2013, Octubre).

RITA: an innovative didactic-pedagogical high school tool. CLEI 2013, Venezuela. Octubre 2013. En proceso de publicación.

Resnick, Mitch. (2008). "**Sowing the Seeds for a More Creative Society**". Learning & Leading with Technology, *35*(4), 18-22.

Ricarose Vallarta Roque (2007). **Openblocks: An Extendable Framework for Graphical Block Programming Systems.** Massachusetts Institute of Technology, Department of Electrical Engineering and Computer Science.

Vallejo Fernández David, González Morcillo Carlos, Alonso Albusac Jiménez Javier. (2012). **Programación Concurrente y Tiempo Real.** Castilla, España.