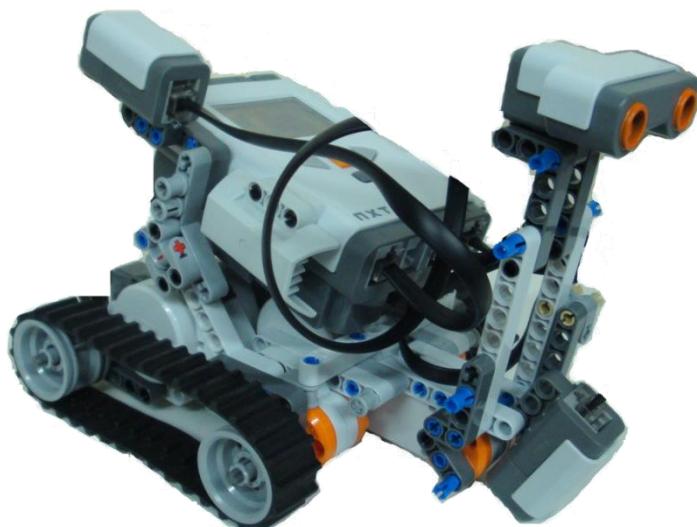


Proyecto Final de Carrera

“Entorno de Simulación MINDSTORMS”



Griselda Inés Arias

Sebastián Eduardo Mangioni

Directora: Dra. Milagros Gutiérrez

Co-Directora: Dra. Georgina Stegmayer

Año: 2013



Índice de contenido

1	Introducción.....	3
1.1	Temática del proyecto	3
1.2	Problema a resolver.....	3
1.3	Objetivos y alcance	4
1.4	Programas similares	5
1.5	Organización del informe	5
2	Marco teórico	7
2.1	LEGO® MINDSTORMS® NXT 2.0	7
2.2	LeJOS	10
2.3	LeJOS NXJ.....	10
2.4	Librería de juegos JGameGrid	11
2.5	Sistema de movimiento del robot.....	11
2.6	Cinemática del robot	13
3	Metodología de desarrollo.....	15
3.1	Descripción general de un proyecto utilizando Scrum.....	16
3.1.1	Definiciones.....	17
3.2	Porque utilizamos Scrum.....	20
3.3	Desarrollo del proyecto utilizando Scrum.....	21
3.3.1	Product backlog	21
3.3.2	Sprints.....	23
3.4	Validación y pruebas	31
3.4.1	Validaciones	31
3.4.2	Pruebas	34
4	Caso de estudio.....	36
5	Conclusiones	50
5.1	Aportes	50
5.2	Trabajos futuros.....	52
	Anexo I.....	53
	Bibliografía.....	58



1 Introducción

Este proyecto final de carrera de investigación y desarrollo fue pensado para dar soporte a la cátedra *Inteligencia Artificial* dictada en la carrera Ingeniería en Sistemas de información (ISI) de la Universidad Tecnológica Nacional, Facultad Regional Santa Fe (FRSF), Argentina, suministrando a los alumnos una nueva herramienta para facilitar el entendimiento de los conceptos impartidos en la cátedra, usando para esto una aplicación que permita *simular* el comportamiento previamente concebido de un *agente inteligente*.

1.1 Temática del proyecto

La *Inteligencia Artificial* (IA) es un área de las ciencias de la computación, dedicada a estudiar la creación de máquinas inteligentes [1]. El enfoque de agentes inteligentes en esta área ha tenido un constante crecimiento. Un *agente* se define como todo aquello que puede considerarse que percibe su ambiente mediante sensores y actúa en tal ambiente por medio de efectores [2]. Así un agente en función de la información que recibe de su ambiente y la definición de la meta a alcanzar, debe implementar un mecanismo de toma de decisión que le permita seleccionar la mejor acción, la cual modifica el ambiente de alguna manera.

La *simulación* es una técnica muy poderosa y ampliamente usada en las ciencias para analizar y estudiar sistemas complejos [3]. Simular, es reproducir artificialmente un fenómeno o las relaciones entrada-salida de un sistema. Esto ocurre siempre cuando la operación de un sistema o la experimentación en él son imposibles, costosas, peligrosas o poco prácticas.

Un *entorno de simulación* es un laboratorio virtual, el cual brinda herramientas para el análisis, comprensión y experimentación de modelos de sistemas reales.

1.2 Problema a resolver

En el año 2010, se ha incorporado a la cátedra una unidad de LEGO® MINDSTORMS® NXT 2.0 (MINDSTORMS) [4] con el objetivo de que el alumno pueda plasmar en un ambiente real un agente inteligente. De esta manera, al incorporar un programa de agente en un dispositivo MINDSTORMS, se logra que el mismo, se comporte como un agente



inteligente en un mundo real y no simulado. La experiencia de los alumnos al trabajar con sensores reales y aspectos físicos reales permite adquirir conocimientos sobre un campo aún no explorado en la enseñanza de inteligencia artificial en nuestra facultad, donde generalmente se trabaja con entornos simulados y donde las condiciones externas están controladas. Esto permite ofrecer una alternativa académica a la enseñanza de agentes inteligentes.

Sin embargo el problema que se plantea para lograr este objetivo de la cátedra es no contar con suficientes unidades MINDSTORMS como para que los alumnos puedan probar sus programas antes de presentar el resultado final a los docentes. En promedio en esta cátedra cursan por año unos 40 alumnos, existiendo aproximadamente unos 15 grupos de alumnos para el desarrollo de trabajos prácticos.

1.3 Objetivos y alcance

Ante esta problemática, se considera necesario poder contar con una herramienta de software¹ que a partir de recibir un programa² de agente, imite el comportamiento del dispositivo físico controlado por dicho programa. De esta forma, los alumnos podrán probar sus programas de agentes utilizando un simulador del dispositivo físico, y una vez probados y depurados, se podrá ejecutar la versión definitiva en el robot MINDSTORMS.



Debido a que el robot permite una infinidad de formas posibles de dispositivos físicos (vehículos, humanoides, animaloides, máquinas tales como brazos de robot, etc.) se prevé comenzar en la cátedra con una estructura predefinida de robot, en este caso un vehículo con motores en ruedas del tipo orugas y con sensores localizados en distintas posiciones en donde el tipo y la localización de los mismos puede variar.

¹Herramienta de software: es un tipo de programa informático diseñado como herramienta para permitir a un usuario realizar uno o diversos tipos de trabajos.

²Programa informático: conjunto de instrucciones que una vez ejecutadas realizarán una o varias tareas en una computadora



1.4 Programas similares

Al momento de comenzar con este proyecto, se investigó la existencia de programas que cumplan con los requisitos solicitados por la cátedra. Si bien es una necesidad extendida sobre todo para distintas cátedras donde usan dicho robot para la enseñanza, y existen distintos foros y sitios especializados donde se plantea la necesidad de un simulador, hasta este momento sólo existen aplicaciones básicas que no son capaces de simular el comportamiento total de un robot MINDSTORMS en sus distintas configuraciones ni que permitan ejecutar programas creados en lenguaje LeJOS (firmware³ que contiene el código para manejar el hardware⁴ del robot, se describirá en detalle en el capítulo 2).

Sin embargo en dicha búsqueda hemos encontrado una librería⁵ llamada “JGameGrid” que da soporte para la creación de juegos simples en dos dimensiones dentro de una matriz de píxeles⁶ que es utilizada por un simulador básico implementado con una librería llamada “NxtJLib” teniendo una configuración de un vehículo parecida a la propuesta en este proyecto, y pudiendo agregar y utilizar distintos sensores pero bastante acotada respecto a las funcionalidades de LeJOS que implementa, y planteando interfaces de movimiento en lugar de implementar las clases LeJOS por lo cual no es viable a las necesidades de la cátedra, pero la librería de base “JGameGrid” nos permitió crear una matriz personalizada para la creación del prototipo del simulador creado en este proyecto.

1.5 Organización del informe

Debajo se realiza una síntesis de los siguientes capítulos que comprenden este informe.

Capítulo 2: En este capítulo se brinda el marco teórico de los conceptos más relevantes que abarca este proyecto, los cuales son:

- Kit de piezas LEGO® MINDSTORMS® NXT 2.0
- Maquina virtual de java LeJOS

³*Firmware*: es un bloque de instrucciones de máquina para propósitos específicos, grabado en una memoria, normalmente de lectura / escritura (ROM, EEPROM, flash, etc), que establece la lógica de más bajo nivel que controla los circuitos electrónicos de un dispositivo de cualquier tipo.

⁴*Hardware*: se refiere a todas las partes tangibles de un sistema informático; sus componentes son: eléctricos, electrónicos, electromecánicos y mecánicos.

⁵*Liberaría obiblioteca* (del inglés *library*) es un conjunto de implementaciones de comportamiento, escritas para un lenguaje de programación, que tienen una interfaz bien definida para el comportamiento que se invoca.

⁶Un *píxel*, plural *píxeles*, (acrónimo del inglés picturelement, ‘elemento de imagen’) es la menor unidad homogénea en color que forma parte de una imagen digital, ya sea esta una fotografía, un fotograma de video o un gráfico.



- Librería LeJOS NXJ
- Librería para la creación de juegos en 2 dimensiones JGameGrid
- Sistema de movimiento diferencial para vehículos

Capítulo 3: Se explica la metodología de desarrollo utilizada durante la construcción del simulador, así como también su aplicación.

Capítulo 4: Se presenta un caso de estudio, con la explicación paso a paso de cómo realizar una simulación, exponiendo las principales características del prototipo.

Capítulo 5: Se detallan los principales aportes y características logradas con este proyecto y la posibilidad de extensión del mismo.



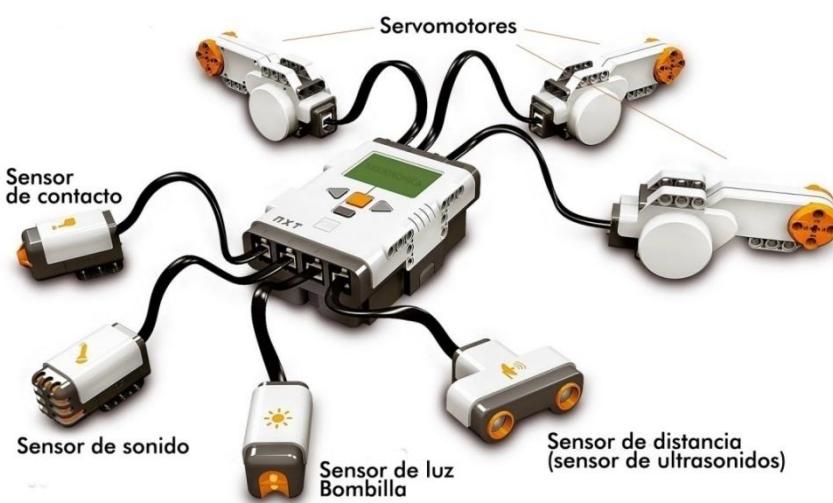
2 Marco teórico

2.1 LEGO® MINDSTORMS® NXT 2.0



LEGO Mindstorms NXT es un kit de piezas lanzado por LEGO⁷ en Julio de 2006, este kit es comercializado en distintas versiones variando en cada una en su cantidad y tipo de piezas. El kit adquirido por la cátedra está formado por:

- 612 piezas para armar diferentes formas de robot;
- un ladrillo inteligente con un microprocesador ARM7 de 32-bits, cuenta con cuatro puertos para conectar sensores y tres puertos donde conectar actuadores;
- tres servo motores;
- cuatro sensores, los cuales son: un sensor de luz y color, un sensor ultrasónico y dos sensores de tacto;
- software para programar el ladrillo inteligente.



⁷LEGO Group es una compañía comprometida con el desarrollo de la creatividad de los niños a través del juego y del aprendizaje. (<http://www.lego.com/es-ar/>)



El ladrillo inteligente que es corazón del kit, y a partir del cual se crean las distintas formas y configuraciones. Es un microprocesador ARM7 de 32-bits que cuenta con:

- 3 puertos de entrada enumerados con letras A, B y C;
- 4 puertos de salida enumerados del 1 al 4;
- 1 puerto USB8;
- 1 pantalla LCD9 monocromática de 100x60 pixeles y
- 4 botones para interactuar con la interfaz de usuario:
dos botones de navegación (color gris claro) uno de mover a la izquierda y otro a la derecha, un botón de selección y encendido (color naranja) y un botón de volver (color gris oscuro).



Puede ejecutar instrucciones desde archivos cargados en su memoria o bien recibir instrucciones a través de conexión bluetooth¹⁰. Es alimentado por 6 pilas AA.

El sensor de tacto, es un pequeño pulsador con dos estados posibles 1 para cuando se encuentra presionado o 0 cuando se encuentra sin presionar, es el sensor más simple del kit.



El sensor de ultrasonido, mide distancias desde el sensor hasta obstáculos que están frente a él. Este sensor puede configurarse para devolver la medición en centímetros (cm) o pulgadas. Su alcance máximo es de 233 cm con una precisión de +3 cm. El principio de funcionamiento de este sensor es enviar ondas de ultrasonido y medir el tiempo que toma a dichas ondas rebotar contra una superficie y regresar hacia el sensor. Este sensor funciona dentro de las especificaciones siempre que las superficies que detecta sean planas.



⁸Universal Serial Bus (USB) (*bus universal en serie BUS*): es un estándar industrial desarrollado a mediados de los años 1990 que define los cables, conectores y protocolos usados en un bus para conectar, comunicar y proveer de alimentación eléctrica entre ordenadores y periféricos y dispositivos electrónicos.

⁹ Una pantalla de cristal líquido o LCD (sigla del inglés liquid crystal display) es una pantalla delgada y plana formada por un número de píxeles en color o monocromos colocados delante de una fuente de luz o reflectora. A menudo se utiliza en dispositivos electrónicos de pilas, ya que utiliza cantidades muy pequeñas de energía eléctrica.

¹⁰Bluetooth es una especificación industrial para Redes Inalámbricas de Área Personal (WPAN) que posibilita la transmisión de voz y datos entre diferentes dispositivos mediante un enlace por radiofrecuencia en la banda ISM de los 2,4 GHz.



El sensor de luz y color, este sensor detecta el nivel de luz y también incluye un LED para iluminación de objetos. El sensor debe ser calibrado y es capaz de distinguir entre los colores blanco, negro, rojo, verde, amarillo y azul.



Servo motor, estos son los actuadores del robot y a partir de los cuales se realiza el movimiento del mismo. Los servomotores no son simples motores, ya que tienen una doble función: En primer lugar, podemos hacer que se muevan a una determinada velocidad o con determinados ángulos, pero además, también pueden actuar como sensores de rotación. Es decir, podríamos, por ejemplo, conectarlo a un volante, y además de poder girarlo con el motor, si alguien moviera el volante, también nos permitiría saber cuánto lo ha girado.



Con dichos elementos es posible implementar un agente que perciba a través de los sensores conectados y que actúe a través de sus actuadores. Este tipo de agente se desenvuelve en un ambiente real.

El ladrillo incluye un lenguaje de programación residente llamado LabVIEW (acrónimo de Laboratory Virtual Instrumentation Engineering Workbench) [5] es una plataforma y entorno de desarrollo para diseñar sistemas, con un lenguaje de programación visual gráfico que es propiedad de National Instrument¹¹. Este lenguaje es útil en versiones simples del robot pero para obtener comportamiento más sofisticado este lenguaje es muy acotado por lo cual se decidió instalar LeJOS.

¹¹National Instruments es una empresa pionera y líder en el desarrollo gráfico de sistemas, un concepto revolucionario que ha cambiado la forma en que ingenieros y científicos resuelven aplicaciones de medición, pruebas y control. Al proveer una plataforma unificada desde el diseño hasta la implementación, el enfoque de GraphicalSystemDesign aumenta la productividad y facilita la innovación, al integrar hardware y software modular basado en tecnología comercial, reduciendo costos y mejorando la competitividad. (<http://latam.ni.com/compania>).



2.2 LeJOS

LeJOS es un firmware para ladrillos programables Lego Mindstorms, que incluye una Maquina Virtual Java (JVM)¹², dándole a estos la capacidad de ejecutar código Java¹³ pre compilado. [6]

2.3 LeJOS NXJ

Es una librería que contiene el código para manejar el hardware del ladrillo, la librería contiene una API¹⁴ bastante completa con la cual el desarrollador puede trabajar con instrucciones de alto nivel, concentrándose así en el comportamiento que se desea obtener del robot sin tener que preocuparse y ocuparse de cómo lograr tal comportamiento. Además permite una conexión entre clases Java de los usuarios con classes.jar para crear un archivo binario que puede cargarse y ejecutarse en el bloque NXT. Incluye también un API para la PC, para escribir programas que se comunican con programas de LeJOS NXJ vía USB o Bluetooth, o bien usando el Protocolo de Comunicaciones de LEGO.

Si bien el proyecto LeJOS nace en el año 2006 para la versión RCX de Lego, con lo cual al momento de esta presentación han pasado más de 7 años, aún no se cuenta con una versión suficientemente estable de la librería (a la fecha la última versión es la 0.9.1.beta), teniendo en cuenta esto, en cada nueva actualización se tienen muchos cambios en agregados, modificaciones y eliminación de funciones.

Al momento de comenzar con este proyecto la cátedra ya contaba con LeJOS 0.8.5beta instalado en el robot en lugar del software grafico que viene con el kit NXT de fábrica.

¹²Una máquina virtual Java (en inglés Java Virtual Machine, JVM) es una máquina virtual de proceso nativo, es decir, ejecutable en una plataforma específica, capaz de interpretar y ejecutar instrucciones expresadas en un código binario especial (el bytecode Java), el cual es generado por el compilador del lenguaje Java.

¹³Java es un lenguaje de programación de propósito general, concurrente, orientado a objetos y basado en clases que fue diseñado específicamente para tener tan pocas dependencias de implementación como fuera posible. Su intención es permitir que los desarrolladores de aplicaciones escriban el programa una vez y lo ejecuten en cualquier dispositivo (conocido en inglés como WORA, o "write once, run anywhere"), lo que quiere decir que el código que es ejecutado en una plataforma no tiene que ser recompilado para correr en otra.

¹⁴Interfaz de programación de aplicaciones (IPA) o API (del inglés Application Programming Interface) representa la capacidad de comunicación entre componentes de software. Se trata del conjunto de llamadas a ciertas bibliotecas que ofrecen acceso a ciertos servicios desde los procesos y representa un método para conseguir abstracción en la programación, generalmente (aunque no necesariamente) entre los niveles o capas inferiores y los superiores del software. Uno de los principales propósitos de una API consiste en proporcionar un conjunto de funciones de uso general, por ejemplo, para dibujar ventanas o iconos en la pantalla. De esta forma, los programadores se benefician de las ventajas de la API haciendo uso de su funcionalidad, evitándose el trabajo de programar todo desde el principio.



2.4 Librería de juegos JGameGrid

Para la realización del simulador es fundamental contar con un ambiente gráfico, el cual permita agregar diferentes componentes con sus distintas características, poder manipular dinámicamente el entorno de tal forma de acomodar los componentes tanto fijos como móviles dentro del ambiente, y por último definir las reglas del entorno, las cuales dictan el comportamiento de dichos componentes.

JGameGrid [7] es una librería gratis creada con fines académicos que contiene numerosas clases¹⁵ para la creación de juegos simples en dos dimensiones. Las principales clases de dicha librería y que utilizamos para la creación del simulador son:

GameGrid: clase para crear una cuadrícula de píxeles, la cual puede contener actores que es otra clase que se define a continuación. En nuestro caso creamos una cuadrícula de 1.000 píxeles por 600 píxeles que se corresponde con un ambiente a escala de 2 por 1,2 metros.

Actor: clase que permite crear componentes de la cuadrícula representados por una imagen y con un comportamiento definido dentro de la misma.

En nuestro caso creamos distintas clases que heredan de *Actor*, representadas con diferentes imágenes a las cuales se les incorporó el comportamiento deseado dependiendo de si es un componente estático o dinámico y como es la interacción entre dichos componentes (actores). Todas las imágenes que se utilizaron para la creación del simulador fueron transformadas según la escala de que 1 píxel representa 2 milímetros, tal como se utiliza para la cuadrícula.

2.5 Sistema de movimiento del robot

El sistema de movimiento que utiliza el robot para moverse es llamado **control diferencial de dirección** que consiste en dos ruedas principales montadas sobre un mismo eje, cada una de las cuales está conectada a su propio motor logrando proveer conjuntamente las funciones de tracción y de dirección a partir de la diferencia de velocidad de cada rueda motora.

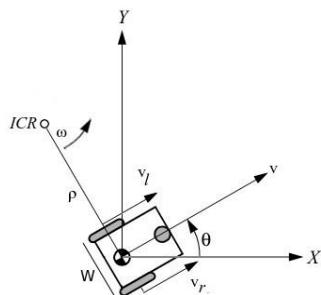
Los movimientos posibles mediante este sistema de movimientos son, movimiento rectilíneo manteniendo las velocidades de los motores igualadas o bien rotaciones alrededor del punto

¹⁵En la programación orientada a objetos (o *POO*) es un paradigma de programación que usa los objetos en sus interacciones, para diseñar aplicaciones y programas informáticos), una *clase* es una construcción que se utiliza como un modelo (o plantilla) para crear objetos de ese tipo. El modelo describe el estado y contiene el comportamiento que todos los objetos creados a partir de esa clase tendrán. Un objeto creado a partir de una determinada clase se denomina una instancia de esa clase.



llamado Centro de Rotación Instantáneo (ICR por sus singlars en Inglés “Instant Centre of Rotation”). El ICR está ubicado sobre el eje imaginario que coincide con el eje de ruedas del robot, y su posición varía en base a la diferencia de las velocidades y sentido de movimiento de ambos motores en el mismo instante.

La figura 1 presenta un esquema de los parámetros de un robot con control diferencial de dirección:



Consideremos como la variación de las velocidades v_l y v_r determinan el movimiento del robot, la velocidad angular ω es una medida de la velocidad de rotación. [8]

Se define como el ángulo girado por una unidad de tiempo y se designa mediante la letra griega ω (omega) (ecuación 1).

Figura 1: Robot con control diferencial

$$\omega = \frac{2\pi}{T} = 2\pi f \quad (1)$$

Su unidad en el Sistema Internacional es el radián por segundo (rad/s).

En cada instante, se debe cumplir que la rueda izquierda y derecha sigan un trayecto que se mueve alrededor del ICR a la misma velocidad angular ω .

De ésta manera, podemos escribir las ecuaciones 2 y 3 para las velocidades v_r y v_l :

$$\omega * (\rho + w/2) = v_r \quad (2)$$

$$\omega * (\rho - w/2) = v_l \quad (3)$$

Donde w es la distancia del eje que une las dos ruedas, v_r y v_l son las velocidades de las ruedas derecha e izquierda, respectivamente, y ρ (ro) es la distancia desde el ICR al punto medio entre las ruedas. En cualquier instante, a partir de (2) y (3) se puede resolver para ρ , ω y v (*ecuaciones 4,5 y 6*):

$$\rho = \frac{w v_r + v_l}{2 v_r - v_l} \quad (4)$$

$$\omega = \frac{v_r - v_l}{w} \quad (5)$$



$$\boldsymbol{v} = \omega \boldsymbol{\rho} = \frac{1}{2}(\mathbf{v}_r + \mathbf{v}_l) \quad (6)$$

De (4), (5) y (6) se deduce que, si $\mathbf{v}_l = \mathbf{v}_r$, entonces el radio ρ es infinito y el robot se mueve en línea recta. Si $\mathbf{v}_l = -\mathbf{v}_r$, entonces el radio es cero y el robot rota sobre su eje de rotación vertical. Para otros valores de \mathbf{v}_l y \mathbf{v}_r , el robot no se mueve en línea recta, sino que sigue una trayectoria curva alrededor de un punto a distancia ρ del centro del robot, cambiando tanto la posición como la orientación del robot. La figura 2 ilustra las distintas combinaciones de velocidades y sus resultados en el movimiento del robot.

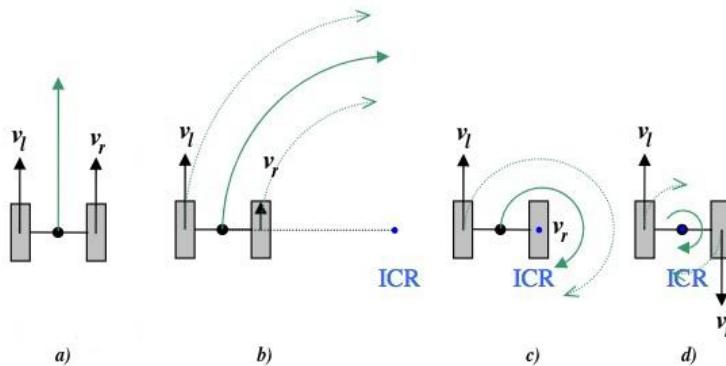


Figura 2: a) $\mathbf{v}_l = \mathbf{v}_r > 0$, b) $\mathbf{v}_l > \mathbf{v}_r > 0$, c) $\mathbf{v}_l > 0, \mathbf{v}_r = 0$, d) $\mathbf{v}_l = -\mathbf{v}_r$

2.6 Cinemática del robot

La cinemática (del griego κινεω, kineo, movimiento) es una rama de la física que estudia las leyes del movimiento (cambios de posición) de los cuerpos, sin tomar en cuenta las causas (fuerzas) que lo producen y se limita, esencialmente, al estudio de la trayectoria en función del tiempo.

Para poder simular el movimiento del robot se necesita un modelo matemático que muestre cómo a partir de las entradas, en nuestro caso un punto inicial (x_0, y_0), una dirección inicial dada por el ángulo inicial θ_0 y velocidades de cada rueda v_r y v_l , el robot se mueve por el plano xy .



El modelo matemático que se adapta a nuestro caso es el siguiente:

$$\left\{ \begin{array}{l} \theta = \theta_0 + c * (v_r - v_l) \\ x = x_0 + ((v_r - v_l)/2) * \cos(\theta) \\ y = y_0 + ((v_r - v_l)/2) * \sin(\theta) \end{array} \right.$$

donde:

c = radio de ruedas / distancia entre ruedas como se muestra en la figura 3



Figura 3: (a) radio de la rueda. (b) distancia entre ruedas



3 Metodología de desarrollo

La metodología que se utilizó para el desarrollo de este proyecto fue **Scrum**, la cual es una metodología ágil de desarrollo¹⁶ de proyectos que toma su nombre y principios de los estudios realizados sobre nuevas prácticas de producción por Hirotaka Takeuchi e Ikujiro Nonaka a mediados de los 80.

Aunque surgió como modelo para el desarrollo de productos tecnológicos, también se emplea en entornos que trabajan con requisitos inestables y que requieren rapidez y flexibilidad; condiciones que se repiten con mucha frecuencia en el desarrollo de sistemas de software.

Un principio clave de Scrum es el reconocimiento de que durante un proyecto los clientes pueden cambiar de idea sobre lo que quieren y necesitan, y que los desafíos impredecibles no pueden ser fácilmente enfrentados de una forma predictiva y planificada. Por lo tanto, Scrum adopta una aproximación pragmática¹⁷, aceptando que el problema no puede ser completamente entendido o definido, y centrándose en maximizar la capacidad del equipo de entregar rápidamente y responder a requisitos emergentes.

Scrum es una metodología de desarrollo muy simple, la cual propone un alto intercambio de información entre los principales interesados del sistema que se está desarrollando y el equipo de desarrollo que lo lleva a cabo a fin de identificar rápidamente desvíos entre lo que se quiere y lo que se está haciendo y realizar las correcciones en etapas tempranas del desarrollo. Esta metodología no se basa en el seguimiento de un plan, sino en la adaptación continua de la evolución/realidad del proyecto.

Un lema recurrente en Scrum es “inspección y adaptación”, dado que el desarrollo conlleva de forma inevitable aprendizaje, innovación y sorpresas. Scrum enfatiza dar pequeños pasos en el desarrollo, inspeccionando tanto el producto resultante como la eficacia de las prácticas

¹⁶ El *desarrollo ágil de software* son métodos de ingeniería del software basados en el desarrollo iterativo e incremental, donde los requerimientos y soluciones evolucionan mediante la colaboración de grupos auto organizados y multidisciplinarios.

¹⁷ La *pragmática* o *pragmalingüística* es un subcampo de la lingüística, también estudiado por la filosofía del lenguaje, la filosofía de la comunicación y la psicolingüística o psicología del lenguaje, que se interesa por el modo en que el contexto influye en la interpretación del significado. El contexto debe entenderse como situación, ya que puede incluir cualquier aspecto extralingüístico: situación comunicativa, conocimiento compartido por los hablantes, relaciones interpersonales, etc. La pragmática toma en consideración los factores extralingüísticos que condicionan el uso del lenguaje, esto es, todos aquellos factores a los que no se hace referencia en un estudio puramente formal.



actuales, y a continuación adaptar los objetivos respecto al producto y las prácticas de los procesos.

3.1 Descripción general de un proyecto utilizando Scrum

Se comienza con la visión general del producto, especificando y dando detalle a las funcionalidades o partes que tienen mayor prioridad de desarrollo y que pueden llevarse a cabo en un periodo de tiempo breve (normalmente un período de tiempo de entre 1 y 4 semanas).

Cada uno de estos períodos de desarrollo es una iteración que finaliza con la producción de un incremento operativo del producto. Estas iteraciones son la base del desarrollo ágil, y Scrum gestiona su evolución a través de reuniones breves diarias en las que todo el equipo revisa el trabajo realizado el día anterior y el previsto para el día en curso.

Scrum controla de forma empírica y adaptable la evolución del proyecto, empleando las siguientes prácticas de la gestión ágil [9]:

- **Revisión de las Iteraciones**

Al finalizar cada iteración se lleva a cabo una revisión con todas las personas implicadas en el proyecto. Este es el periodo máximo que se tarda en reconducir una desviación en el proyecto o en las circunstancias del producto.

- **Desarrollo incremental**

Durante el proyecto, las personas implicadas no trabajan con diseños o abstracciones. El desarrollo incremental implica que al final de cada iteración se dispone de una parte del producto operativa que se puede inspeccionar y evaluar.

- **Desarrollo evolutivo**

Los modelos de gestión ágil se emplean para trabajar en entornos de incertidumbre e inestabilidad de requisitos. Intentar predecir en las fases iniciales cómo será el producto final, y sobre dicha predicción desarrollar el diseño y la arquitectura del producto no es realista, porque las circunstancias obligarán a remodelarlo muchas veces. Para qué predecir los estados finales de la arquitectura o del diseño si van a estar cambiando. En Scrum se toma a la inestabilidad como una premisa, y se adoptan técnicas de trabajo para permitir esa evolución sin degradar la calidad de la arquitectura que se irá generando durante el desarrollo.



El desarrollo Scrum va generando el diseño y la arquitectura final de forma evolutiva durante todo el proyecto, no los considera como productos que deban realizarse en la primera “fase” del proyecto (el desarrollo ágil no es un desarrollo en fases).

- **Auto-organización**

Durante el desarrollo de un proyecto son muchos los factores impredecibles que surgen en todas las áreas y niveles. La gestión predictiva confía la responsabilidad de su resolución al gestor de proyectos.

En Scrum los equipos son auto-organizados (no auto-dirigidos), con margen de decisión suficiente para tomar las decisiones que consideren oportunas.

- **Colaboración**

Las prácticas y el entorno de trabajo ágiles facilitan la colaboración del equipo. Ésta es necesaria, porque para que funcione la auto-organización como un control eficaz cada miembro del equipo debe colaborar de forma abierta con los demás, según sus capacidades y no según su rol o su puesto.

3.1.1 Definiciones

Sprint: Scrum denomina “sprint” a cada iteración de desarrollo y recomienda realizarlas con duraciones entre 1 y 4 semanas. El sprint es por tanto el núcleo central que proporciona la base de desarrollo iterativo e incremental.

User Story (US): Las historias de usuario son sentencias escritas en lenguaje natural o de negocios de un usuario final o un usuario del sistema que se quiere crear, que describe lo que el usuario quiere o necesita para hacer parte de su trabajo. Estas sentencias expresan “quien”, “que” y el “porque” de un requerimiento de un modo simple y conciso, generalmente careciendo de detalles de cómo se va a implementar.

Task: Cada US es detallada en distintas tareas a ser desarrolladas para la culminación de la misma. La tarea debe cumplir con ciertos requisitos, debe ser clara al momento de explicitar los resultados esperados, debe ser simple de tal forma de que pueda ser estimada correctamente y debe ser de corta/media duración de tal forma de respetar un límite máximo de tiempo para su resolución (en nuestro caso tomamos como base que si la tarea requiere más de 16 hs entonces se debe dividir en tareas más simples).



Principales Roles en Scrum:

- **Scrum master:** es el entrenador del equipo de desarrollo, ayuda a cada miembro del equipo a alcanzar su mejor performance, es el indicado para quitar las distracciones o impedimentos fuera de las tareas principales en las que se focaliza cada integrante del equipo de desarrollo.
- **Product owner:** es la persona que mejor conoce las necesidades del producto que se va a desarrollar, tiene una visión abstracta y general de los requerimientos del producto. Es la persona encargada de transmitir y guiar al equipo de desarrollo a los objetivos primarios del producto que se va a desarrollar.
- **Scrum team:** el equipo de desarrollo propiamente dicho, el equipo está integrado por personas con diferentes capacidades, habilidades y aptitudes, donde cada miembro aporta algo único y particular al proceso y que impacta en el producto que se está desarrollando. Todo el esfuerzo de cada miembro esta focalizado en contribuir a la finalización de todas las tareas del sprint. Se espera que los miembros del equipo concentren la mayoría de su tiempo en aplicar sus habilidades para la terminación exitosa de cada sprint, unido a esto se espera que cada miembro trabaje en sus disciplinas preferidas o de acuerdo con sus intereses.

Artefactos:

La metodología Scrum se nutre de distintos artefactos para llevar adelante el proceso de desarrollo y mejora continua tanto del producto como del proceso mismo. [10] A continuación se detallan los principales artefactos del proceso:

- **The final product:** el principal artefacto que se genera de un proceso Scrum es el producto de software o de cualquier otra clase que da origen a todo el proceso.
- **Product backlog:** es una lista completa de funcionalidades que están faltando agregar modificar o corregir en el producto. Este artefacto es muy dinámico y está continuamente actualizándose. Es responsabilidad del *product owner* mantener este artefacto priorizado y actualizado de manera tal que el *scrum team* haga uso de él y trabaje sobre los requerimientos que generen más valor a los stakeholders¹⁸ en cada

¹⁸Stakeholders (Clientes, Proveedores, Vendedores, etc): Se refiere a la gente que hace posible el proyecto y para quienes el proyecto producirá el beneficio acordado que justifica su producción.



sprint. La forma más común de expresar las funcionalidades en el product backlog es mediante las [user stories](#).

El product owner es el encargado y responsable de registrar los requerimientos en el artefacto product backlog, el cual es un artefacto fundamental como documento y herramienta de trabajo del scrum team.

- **Sprint backlog:** consiste en una lista de tareas que el scrum team debe realizar para la culminación exitosa del sprint. Esta lista se realiza durante las [print planning meeting](#), donde se va tomando cada US (desde el product backlog) a ser desarrollada durante el sprint que se está planificando, se intercambian ideas y opiniones entre los miembros del scrum team acerca de las tareas por hacer para la realización de dicha US. Al final de las print planning meeting, cada tarea termina siendo estimada en cantidad de horas o minutos y asignada a algún miembro del scrum team. Se toman tantas US como sea posible para el tiempo del que se dispone en el sprint.

Proceso de desarrollo

El proceso comienza con la definición de User Stories en el artefacto Proyect Backlog, este artefacto es muy dinámico y nutre al resto del proceso siendo la fuente de requerimientos de cada nueva iteración o sprint.

El comienzo de cada iteración se inicia con la ***sprint planning meeting*** la cual está integrada generalmente por el product owner, el scrum master y el scrum team, en esta reunión se seleccionan las US desde el product backlog que el equipo se compromete a terminar en la sprint que se está planificando y que tienen mayor importancia en el incremento de funcionalidad entregable a los Stakeholders, parte del trabajo realizado en esta reunión es desglosar cada US en Tasks y estimar cada Task individual. Al finalizar la iteración se realiza la llamada ***Sprint Review*** donde se concluye los puntos positivos, negativos y se proponen mejoras en el proceso para las próximas iteraciones. También de acuerdo al avance que se haya logrado es una práctica habitual concluir con una demo para mostrar las nuevas funcionalidades agregadas a los Stakeholders.



3.2 Porque utilizamos Scrum

Basamos la elección de la metodología de desarrollo en ciertos criterios que consideramos que iban a allanarnos el camino en el desarrollo de este proyecto, los cuales serán detallados a continuación:

► *Requerimientos inconclusos*

Al momento de comenzar con este proyecto no contamos con la información/dominio suficiente de las diferentes áreas involucradas (comportamiento y arquitectura del robot, simulación, modelados matemáticos de movimientos de robots, etc.), es por esto que esperamos que los requerimientos expuestos vayan completándose y emergan nuevos a medida que se avance en la construcción del simulador.

► *Retroalimentación (feedback)*

El feedback es fundamental, para que las partes involucradas tengan siempre la misma visión de lo que se está haciendo y si además es realizado entre períodos breves posibilita la corrección temprana de desviaciones en las interpretaciones de los requerimientos.

Si bien ninguna metodología de desarrollo prohíbe el feedback entre los stakeholders y el equipo de desarrollo, Scrum no solo brinda reuniones periódicas donde se obtiene feedback, sino que promueve la comunicación continua entre todos los involucrados.

► *Generación de prototipos*

Contar con prototipos incompletos del simulador nos brinda la posibilidad de poder ir probando funcionalidades ya implementadas y compararlas con el funcionamiento real del robot NXT. En una metodología tradicional por etapas, los primeros resultados de la aplicación que se desarrolla se empiezan a ver pasado un largo tiempo, dado que la etapa de desarrollo queda postergada a tener terminadas las etapas previas de adquisición de requisitos, armado de los modelos y arquitectura.

Puesto que Scrum permiten tener versiones del producto/aplicación previas a la versión final, el cliente puede disponer de forma rápida de alguna versión intermedia.



► *Desarrollo incremental*

Dado que el robot es modular y permite diferentes configuraciones y su complejidad puede ir desde muy simple a muy compleja, nos habilita a comenzar por la construcción de una versión de robot simple (un sensor y dos motores) e ir implementando funcionalidades de mayor complejidad en sucesivas iteraciones.

► *Escalabilidad*

Consideramos que este proyecto puede ir creciendo en nuevos requerimientos y funcionalidades dado que el robot en el que basa su funcionalidad va mejorando, actualizándose y agregando nuevas piezas con el tiempo y creemos que una metodología como Scrum permite que esto sea posible con mínimo esfuerzo.

3.3 Desarrollo del proyecto utilizando Scrum

Tal como se definió en el capítulo anterior, el proceso de Scrum comienza definiendo las User Stories (USs) dentro del product backlog, la sección siguiente muestra todas las USs que se definieron desde el inicio del proceso hasta la presentación de este informe.

3.3.1 Product backlog

User Stories con estado “Ejecutada”:

- **US1** - Como programador LeJOS (PL), quiero utilizar la librería LeJOS en un ambiente distinto al robot Lego NXT.
- **US2** - Como PL, quiero conocer las diferentes herramientas de simulación para un robot lego NXT existentes que se ejecuten a partir de código LeJOS.
- **US3** - Como PL, quiero conocer cuál es el comportamiento del robot a partir de diferentes programas LeJOS de entrada.
- **US4** - Como PL, quiero tener una herramienta que me sirva para determinar si mi programa LeJOS está haciendo lo que yo quiero cuando no tengo el robot físico para probarlo.



- ☛ **US5** - Como PL, quiero poder cambiar algunos aspectos de la configuración del robot (ubicación de los sensores).
- ☛ **US6** - Como PL, quiero poder simular programas LeJOS, que contenga la clase Motor para mover el robot como hace el real.
- ☛ **US7** - Como PL, quiero poder ejecutar un programa simple de código LeJOS en el simulador sin tener que agregar código JAVA extra.
- ☛ **US8** - Como PL, quiero poder simular programas LeJOS, que contenga la clase ColorSensor para detectar colores tal como el robot real lo hace.
- ☛ **US9** - Como PL, quiero poder simular programas LeJOS, que contenga la clase TouchSensor para detectar obstáculos tal como el robot real lo hace.
- ☛ **US10** - Como PL, quiero poder simular programas LeJOS, que contenga la clase UltrasonicSensor para detectar obstáculos tal como el robot real lo hace.
- ☛ **US11** - Como PL, quiero actualizar la versión LeJOS de tal forma de trabajar con la misma versión del robot, la cual se actualizó recientemente (versión 0.9.1beta).
- ☛ **US12** - Como PL, quiero poder ubicar el robot en cualquier posición del espacio de simulación antes de comenzar a simular.
- ☛ **US13** - Como PL, quiero poder realizar modificaciones en el ambiente y/o posición del robot en cualquier momento siempre que la simulación se encuentre detenida.
- ☛ **US14** - Como PL, quiero poder crear una réplica de ambientes reales sencillos donde probar los programas creados para el robot.
- ☛ **US15** - Como PL, quiero poder pintar cualquier posición dentro del área de simulación en los colores que detecta el sensor de color del robot (blanco, negro, rojo, verde, amarillo y azul).
- ☛ **US16** - Como PL, quiero poder limpiar el ambiente de simulación.
- ☛ **US17** - Como PL, quiero poder Guardar/Modificar/Borrar un ambiente.



User Stories con estado “Pendiente”:

- ☛ **US18** - Como PL, quiero poder disponer de un plug-in¹⁹ para Eclipse²⁰ que al hacer clic con botón derecho en un código LeJOS, ejecute dicho código en el simulador.
- ☛ **US19** - Como PL, quiero poder simular programas LeJOS, que contenga la clase InfraredSensor para detectar obstáculos tal como el robot real lo hace.
- ☛ **US20** - Como PL, quiero poder utilizar todas las clases de la librería LeJOS en los programas que simule.
- ☛ **US21** - Como PL, quiero poder visualizar la lectura de los sensores en una ventana.
- ☛ **US22** - Como PL, quiero tener una pantalla de las mismas características que las del robot, de tal manera de poder visualizar el mismo contenido que se muestra en la pantalla del robot.

3.3.2 Sprints

Para las sucesivas iteraciones (sprints) no nos ajustamos estrictamente a fechas, debido a que cada miembro del scrum team tiene comprometida gran parte de su disponibilidad horaria en cargas laborales y otras responsabilidades.

Pautas de generación y ejecución de los Sprints:

- Duración del sprint: 4 semanas.
- Duración máxima de una tarea: 16 hs (si la tarea requiere más horas entonces se debe dividir en tareas más simples).
- Cantidad de integrantes del scrum team: 2 personas.
- Cantidad de hs por miembro del scrum team: 2 hs por día de lunes a viernes.
- Reuniones del scrum team: una reunión semanal.
- Duración del sprint en hs del scrum team: 4 semanas x 5 días/semanas x 4 hs/día = 80hs

¹⁹ Un *complemento* es una aplicación que se relaciona con otra para aportarle una función nueva y generalmente muy específica. Esta aplicación adicional es ejecutada por la aplicación principal e interactúan por medio de la API. También se conoce como *plug-in* (del inglés «[un] enchufable o inserción»), add-on («añadido»), conector o extensión.

²⁰ El *Eclipse* es un entorno de desarrollo integrado (IDE, Integrated Development Environment) que facilita las tareas de edición, compilación y ejecución de programas durante la fase de desarrollo.



A continuación se detallarán algunos de los sprints más relevantes del proceso:

Sprint 3:

User Stories:

US3 - Como programador LeJOS (PL), quiero conocer cuál es el comportamiento del robot a partir de diferentes programas LeJOS de entrada.

Tasks:

Task 3-1: Investigar el Kit de piezas LEGO® MINDSTORMS® NXT 2.0. 16 hs.

Task 3-2: Determinar la configuración del robot que se utilizará. 8 hs.

Task 3-3: Configurar las PCs de desarrollo para poder comunicarse con el robot. 16 hs.

Task 3-4: Realizar programas LeJOS para testear los actuadores. 4 hs.

Task 3-5: Hacer las pruebas correspondientes a los actuadores en el robot, y tomar nota de los resultados. 12 hs

Task 3-4: Realizar programas LeJOS para testear los sensores. 10 hs.

Task 3-6: Hacer las pruebas correspondientes a los sensores en el robot, y tomar nota de los resultados. 16hs.

Valor generado de este sprint:

Configuración del robot:

Se eligió la configuración del robot mostrada en la figura 4.

Dicha configuración se corresponde con un vehículo explorador, que cuenta con dos ruedas tipo oruga y la ubicación de hasta 4 sensores (tacto, luz y ultrasonido) que puede ser modificada de acuerdo a lo que el vehículo requiere para manejarse dentro del ambiente planteado.



Figura 4: configuración del robot

Cada rueda tipo oruga es manejada por un motor, los dos motores utilizados se corresponden a los puertos de salida 1 y 2, el primero para la rueda izquierda y el segundo para la rueda derecha del vehículo.



En el caso de los sensores, dependiendo el ejemplo que se esté simulando, se usan 1 o más sensores. Existen 4 posiciones posibles para agregar un sensor en el vehículo, estas son al frente del vehículo izquierda, central y derecha dichas posiciones se corresponden con los puertos de entrada 1, 2 y 3 respectivamente, y una posición central en la parte trasera del vehículo que se corresponde con el puerto de entrada 4.

Resultados obtenidos de los actuadores:

Las clases más importantes de los actuadores son (figura 5): `lejos.nxt.Motor` y `lejos.nxt.MotorPort`.

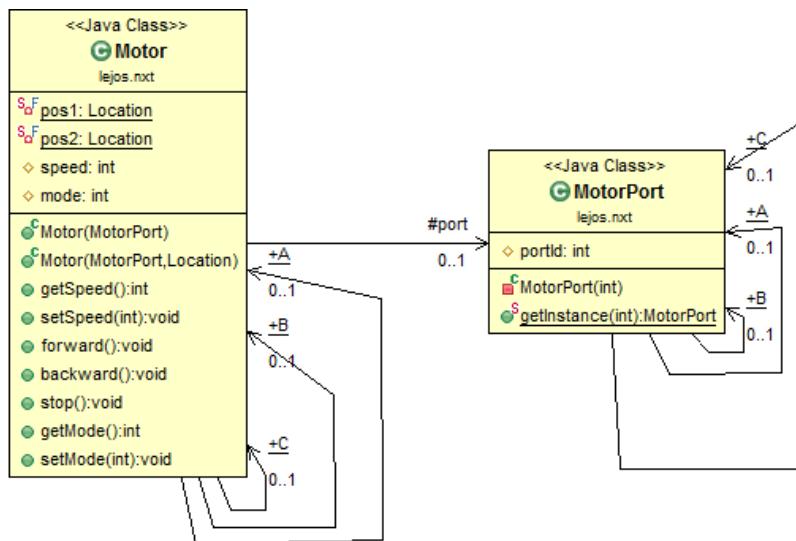


Figura 5: Clases para representar actuadores

Resultados obtenidos de los sensores:

Las clases más importantes de los sensores son (figura 6): `lejos.nxt.Sensor`, `lejos.nxt.SensorPort`, `lejos.nxt.TouchSensor`, `lejos.nxt.ColorSensor` y `lejos.nxt.UltrasonicSensor`.

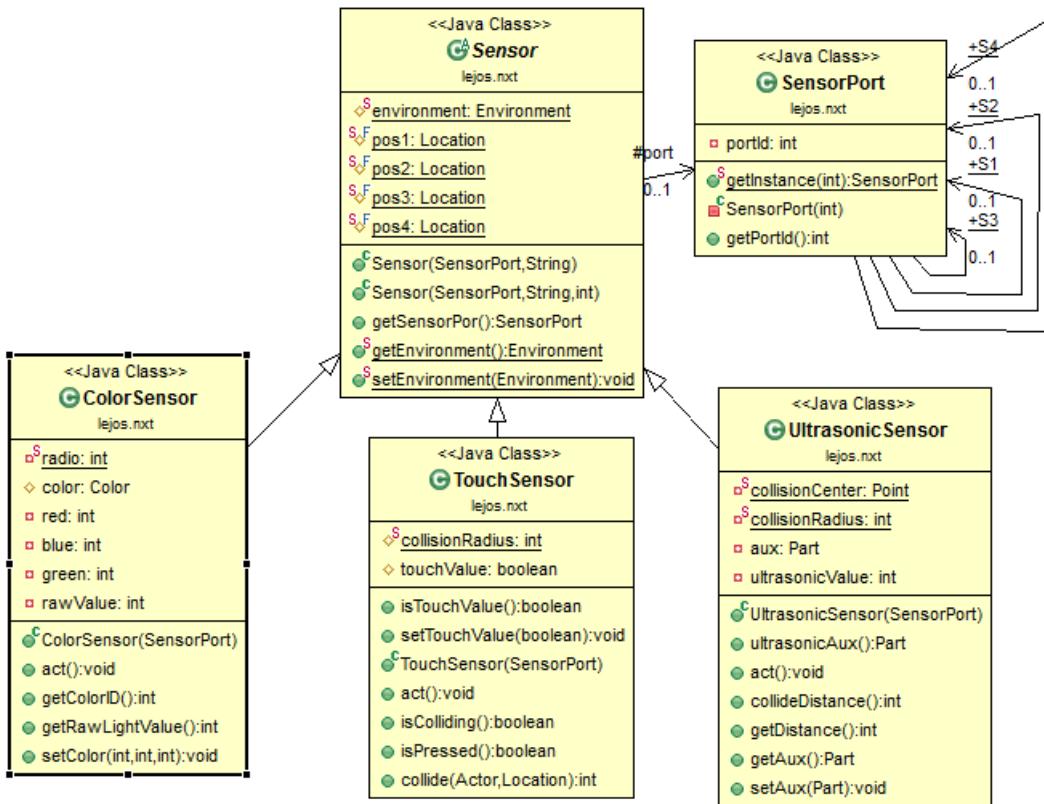


Figura 6: clases para representar los sensores

Sprint 4:

User Stories:

US4 - Como programador LeJOS (PL), quiero tener una herramienta que me sirva para determinar si mi programa LeJOS está haciendo lo que yo quiero cuando no tengo el robot físico para probarlo.

Tasks:

Task 4-1: Investigar las librerías gráficas existentes que permitan representar un robot y un ambiente definido. 16 hs.

Task 4-2: Crear un proyecto JAVA Swing. 2 hs.

Task 4-3: Agregar librería LeJOS al proyecto y extender o sobreescibir todas las clases nativas de dicha librería. 16 hs.

Task 4-4: Crear una cuadrícula que represente un ambiente de 2 metros por 1.20 metros. 16 hs.



Task 4-5: Tomar fotos del robot y sus componentes. 2 hs.

Task 4-6: Transparentar el fondo de las imágenes y llevar a escala las mismas. 16 hs.

Task 4-7: Agregar la imagen del ladrillo inteligente en una posición dada de la cuadrícula. 8 hs.

Task 4-8: Agregar las imágenes de las ruedas oruga a los laterales del ladrillo inteligente. 5 hs.

Valor generado de este sprint:

En este sprint se tomó la decisión de implementar el simulador mediante el uso de la librería JGameGrid, la cual brinda un marco para la creación de aplicaciones similares a la que estamos desarrollando.

También fue creada la estructura básica de nuestro proyecto java respetando la estructura de paquetes brindada por la librería LeJOS, de tal forma que los programas puedan ejecutarse tanto en el simulador como en el robot real sin tener que modificar, agregar o quitar código.

Por último se decidió implementar la escala de 1px = 2mm, y se llevaron a dicha escala las fotografías tomadas del robot de la cátedra IA.

La figura 7 muestra una imagen del avance logrado en este sprint:

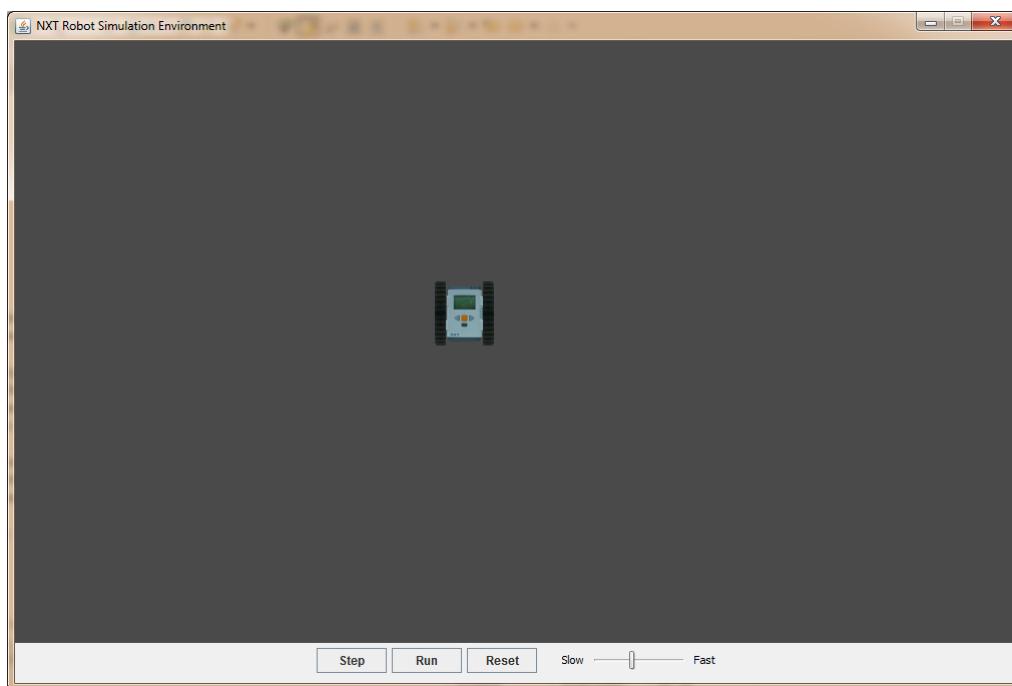


Figura 7: interfaz gráfica del simulador



Sprint 14:

User Stories:

US14 - Como PL, quiero poder crear una réplica de ambientes reales sencillos donde probar los programas creados para el robot.

US15 - Como PL, quiero poder pintar cualquier posición dentro del área de simulación en los colores que detecta el sensor de color del robot (blanco, negro, rojo, verde, amarillo y azul).

Tasks:

Task 14-1: Cambiar la clase “`nxt.simulator.Obstacle`” para que el simulador permita agregar ladrillos cuadrados pequeños de tamaño fijo en lugar de las imágenes pre cargadas que con las cuales se venía trabajando. 16 hs.

Task 14-2: Crear la funcionalidad de agregar obstáculos presionando el botón izquierdo del mouse en una posición determinada del ambiente. 16 hs.

Task 14-3: Crear la funcionalidad de borrar obstáculos presionando el botón izquierdo del mouse sobre un obstáculo del ambiente. 8 hs

Task 14-4: Agregar en la interfaz radio buttons para poder seleccionar agregar o borrar obstáculos del ambiente. 5 hs.

Task 15-1: Agregar en la interfaz un radio button para poder seleccionar pintar el ambiente. 2 hs.

Task 15-2: Agregar en la interfaz un combo box donde se pueda seleccionar el color que se desee usar para pintar el fondo. 5hs.

Task 15-3: Crear la funcionalidad de pintar una superficie cuadrada de un color previamente seleccionado presionando el botón izquierdo del mouse en una posición determinada del ambiente. 16 hs

Task 15-4: Crear la funcionalidad de borrar una superficie pintada presionando el botón izquierdo del mouse en una posición determinada del ambiente. 8 hs.

Valor generado de este sprint:

La figura 8 muestra la imagen obtenida del simulador antes de comenzar con el sprint:

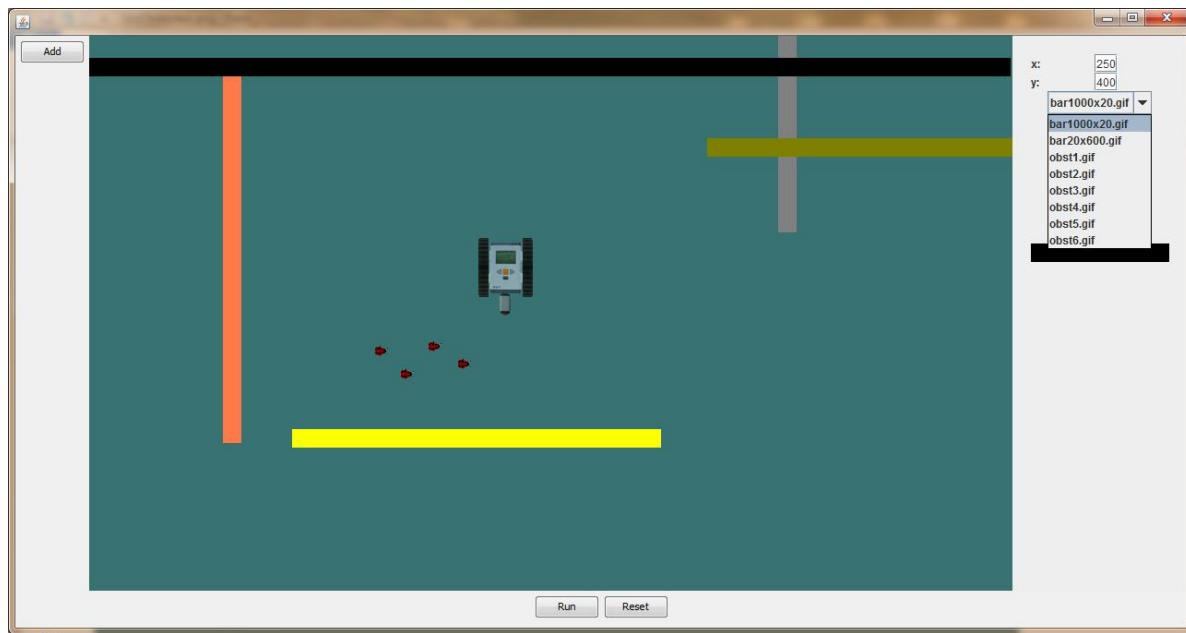


Figura 8: Imagen del simulador antes de comenzar el sprint

La figura 9 muestra la imagen obtenida del simulador al finalizar el sprint:

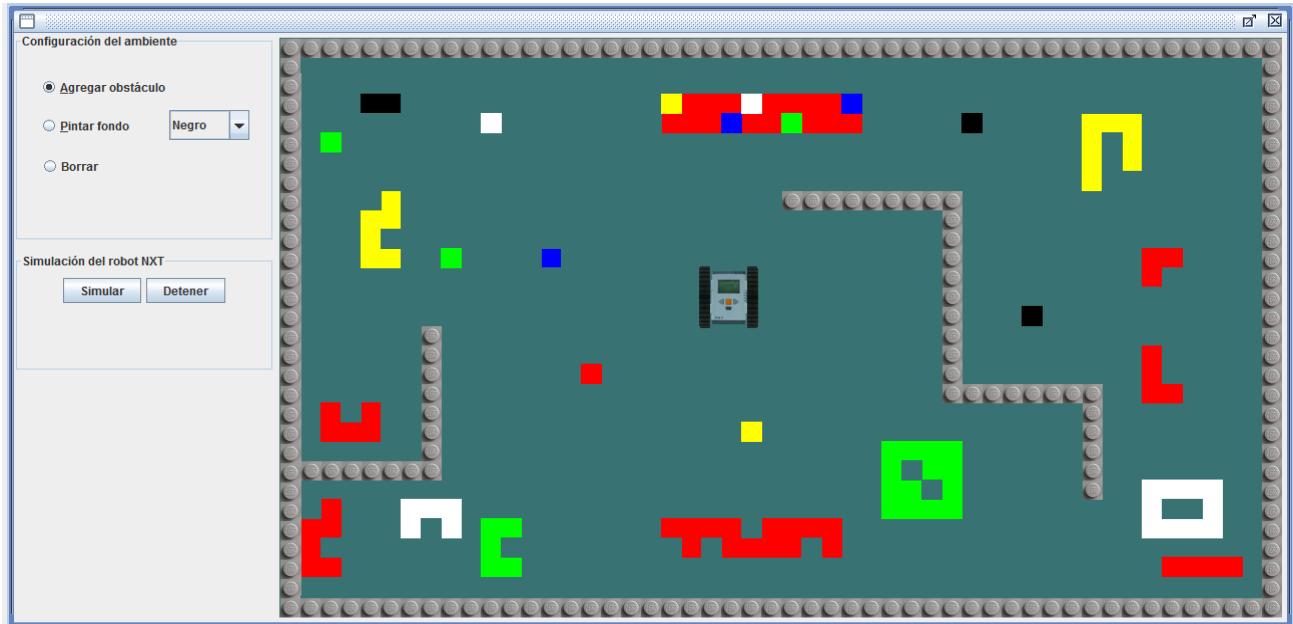


Figura 9: Imagen del simulador después de terminado el sprint

Sprint 15:

User Stories:



US12 - Como PL, quiero poder ubicar el robot en cualquier posición del espacio de simulación antes de comenzar a simular.

US16 - Como PL, quiero poder limpiar el ambiente de simulación.

US17 - Como PL, quiero poder Guardar/Modificar/Borrar un ambiente.

Tasks:

Task 12-1: Agregar en la interfaz dos botones para rotar el robot. 8 hs.

Task 12-2: Agregar la funcionalidad de rotar el robot en sentido de las agujas del reloj y contrario a las agujas del reloj. 5 hs.

Task 12-3: Agregar la funcionalidad de mover el robot dentro del ambiente (drag and drop²¹). 16 hs.

Task 16-1: Agregar en la interfaz un botón “Limpiar”. 4 hs.

Task 16-2: Agregar la funcionalidad para limpiar el ambiente, de tal forma que se eliminen todos los cambios se fueron realizando en el mismo. 4 hs.

Task 17-1: Crear una clase EnvironmentConfigurationDAO que permita guardar los cambios que se generaron en el ambiente. 16 hs.

Task 17-2: Agregar en la interfaz los botones para guardar, cargar un ambiente. 6 hs.

Task 17-3: Crear la funcionalidad de persistir un objeto environmentConfigurationDAO como un archivo binario. 10 hs.

Task 17-4: Crear la funcionalidad de cargar un ambiente desde un archivo binario previamente guardado. 16 hs.

Task 17-5: Agregar la funcionalidad de guardar la dirección y posición actual del robot en el ambiente en el archivo binario. 6 hs.

Task 17-6: Agregar la funcionalidad de cargar la dirección y posición del robot en el ambiente desde el archivo binario guardado. 6 hs.

Valor generado de este sprint:

²¹Arrastrar y soltar (drag and drop) es una expresión informática que se refiere a la acción de mover con el ratón objetos de una ventana a otra o entre partes de una misma ventana.



La figura 10 muestra una imagen con las modificaciones realizadas y las funcionalidades respectivas que se lograron en el sprint:

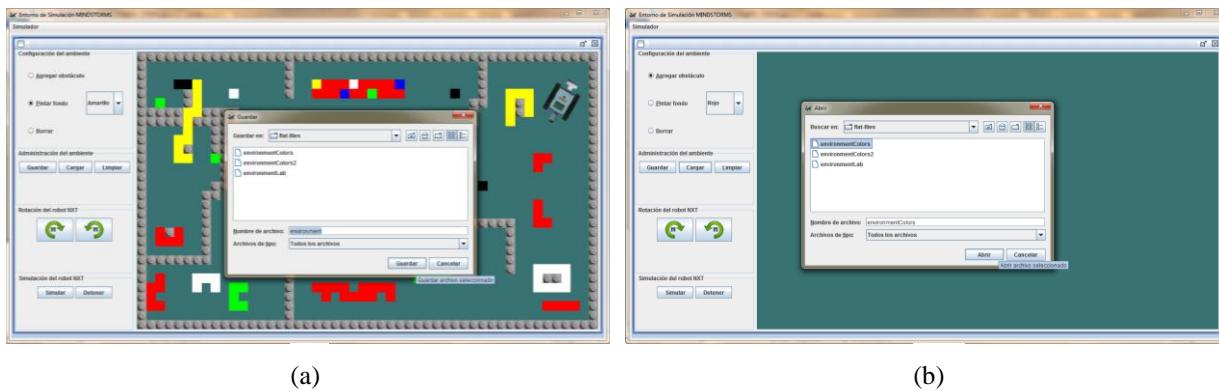


Figura 10: Interfaz gráfica del simulador: nuevas funcionalidades (a) guardar y (b) abrir.

3.4 Validación y pruebas

3.4.1 Validaciones

En esta subsección se detallarán algunos de los programas escritos en código LeJOS que fueron utilizados para realizar diferentes validaciones entre el simulador y el robot real, probando así todos los sensores y actuadores del robot.

El primer programa llamado “LineaRoja” utiliza el sensor de color *ColorSensor* y hace que el robot se mueva hacia adelante y cuando detecta el color rojo detiene una de sus motores (B) con lo cual el robot gira hacia la derecha. Este código puede verse en la figura 11.

El segundo programa llamado “Chocador” utiliza los sensores de tacto *TouchSensor* y de ultra sonido *UltrasonicSensor* teniendo dos ciclos diferentes, el primero funciona de manera que mientras el *UltrasonicSensor* detecte una distancia superior a 15 cm va a avanzar hacia adelante, y de ser menor se detiene y el segundo ciclo funciona de manera que mientras el *TouchSensor* no esté presionado el robot va a retroceder. La figura 12 muestra el código de este programa LeJOS.

El tercer programa llamado “Movimientos” muestra diferentes movimientos del robot al avanzar, retroceder a iguales y diferentes velocidades y con alguna de sus ruedas detenida. Este programa permitió validar los motores. La figura 13 muestra el código de este programa LeJOS.



```
import lejos.nxt.Button;
import lejos.nxt.ColorSensor;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.robotics.Color;

public class LineaRoja {

    public LineaRoja() throws InterruptedException {
        ColorSensor cs = new ColorSensor(SensorPort.S2);
        Motor.A.setSpeed(500);
        Motor.B.setSpeed(500);
        while (!Button.ENTER.isPressed()) {
            Motor.A.forward();
            Motor.B.forward();
            if (Color.RED == cs.getColorID()) {
                Motor.B.stop();
                Thread.sleep(7000);
            }
            Thread.sleep(0);
            Motor.B.setSpeed(500);
            Motor.B.forward();
        }
    }

    public static void main(String[] args) throws InterruptedException {
        new LineaRoja();
    }
}
```

Figura 11: código del programa LeJOS “LineaRoja”

```
import lejos.nxt.Button;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.nxt.TouchSensor;
import lejos.nxt.UltrasonicSensor;

public class Chocador {
    public Chocador() throws InterruptedException {
        UltrasonicSensor us = new UltrasonicSensor(SensorPort.S2);
        TouchSensor ts = new TouchSensor(SensorPort.S4);
        Motor.A.setSpeed(500);
        Motor.B.setSpeed(500);
        while (!Button.ENTER.isPressed()) {
            while (us.getDistance() > 15) {
                Motor.A.forward();
                Motor.B.forward();
                Thread.sleep(1);
            }
            Thread.sleep(1);
            Motor.A.stop();
            Motor.B.stop();
            while (!ts.isPressed()) {
                Motor.A.backward();
                Motor.B.backward();
                Thread.sleep(1);
            }
        }
    }

    public static void main(String[] args) throws InterruptedException {
        new Chocador();
    }
}
```

Figura 12: Código del programa LeJOS “Chocador”



```
import lejos.nxt.Motor;

/**
 * Ejemplo de los distintos movimientos del robot, variando las velocidades de
 * cada motor de movimiento.
 */
public class Movimientos {

    public Movimientos() throws InterruptedException {
        // Ir hacia adelante, rueda derecha mas rápido
        Motor.A.setSpeed(500);
        Motor.B.setSpeed(250);
        Motor.A.forward();
        Motor.B.forward();
        Thread.sleep(8000);
        Motor.A.stop();
        Motor.B.stop();
        // Ir hacia adelante, rueda izquierda mas rápido
        Motor.A.setSpeed(250);
        Motor.B.setSpeed(500);
        Motor.A.forward();
        Motor.B.forward();
        Thread.sleep(8000);
        Motor.A.stop();
        Motor.B.stop();
        // Ir hacia adelante con rueda derecha detenida
        Motor.B.setSpeed(500);
        Motor.A.stop();
        Motor.B.forward();
        Thread.sleep(8000);
        Motor.B.stop();
        // Ir hacia adelante con rueda izquierda detenida
        Motor.A.setSpeed(500);
        Motor.A.forward();
        Motor.B.stop();
        Thread.sleep(8000);
        Motor.A.stop();
        // Ir hacia atrás a la misma velocidad
        Motor.A.setSpeed(400);
        Motor.B.setSpeed(400);
        Motor.A.backward();
        Motor.B.backward();
        Thread.sleep(4000);
        Motor.A.stop();
        Motor.B.stop();
        // Ir hacia adelante, rueda derecha mas rápido
        Motor.A.setSpeed(300);
        Motor.B.setSpeed(300);
        Motor.A.forward();
        Motor.B.forward();
        Thread.sleep(4000);
        Motor.A.stop();
        Motor.B.stop();
    }

    public static void main(String[] args) throws InterruptedException {
        new Movimientos();
    }
}
```

Figura 13: Código del programa LeJOS “Movimientos”



3.4.2 Pruebas

Durante todo el proceso de desarrollo de la aplicación se fueron realizando pruebas²², las cuales sirvieron para descubrir y resolver diferentes bugs²³.

A continuación se listan los bugs que quedaron pendientes de resolución:

- *En los límites del ambiente parte del robot deja de verse, ya que queda debajo de otras secciones de la ventana (figura 14).*

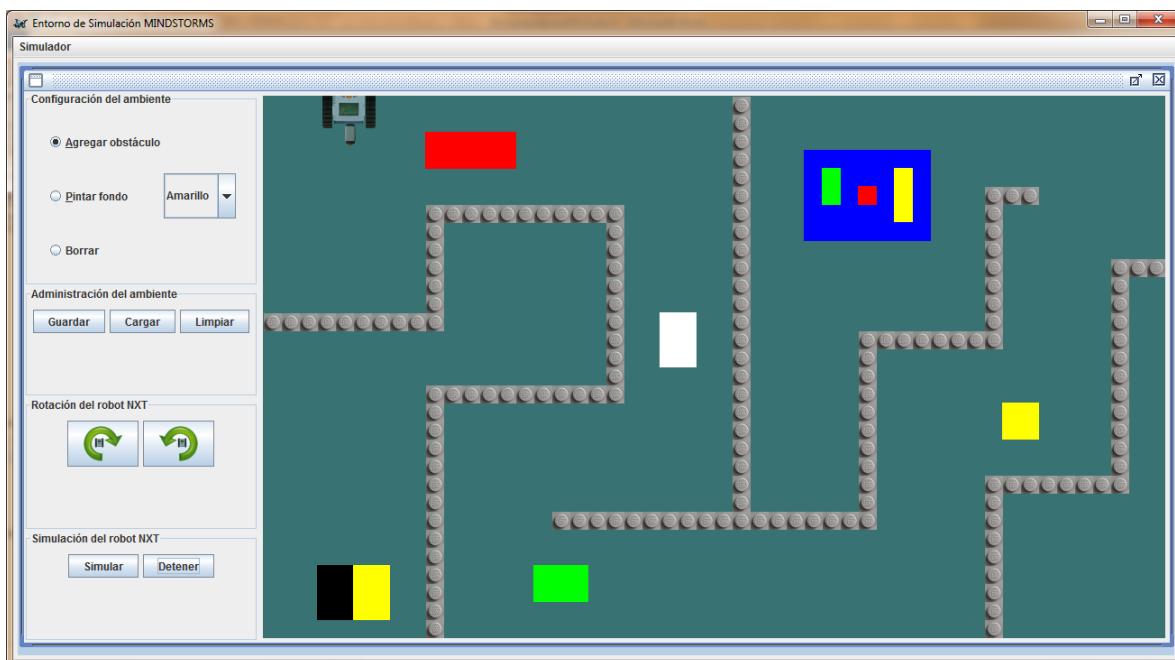


Figura 14: Bug 1 – Detección de límites del ambiente en la interfaz gráfica

²²Las *pruebas de software* (en inglés software testing) son las investigaciones empíricas y técnicas cuyo objetivo es proporcionar información objetiva e independiente sobre la calidad del producto a la parte interesada o stakeholder. Es una actividad más en el proceso de control de calidad.

²³Un *error de software*, comúnmente conocido como *bug* (bicho), es un error o fallo en un programa de computador o sistema de software que desencadena un resultado indeseado.



- Cuando los sensores (*TouchSensor*, *UltrasonicSensor* y *ColorSensor*) chocan contra los obstáculos el sensor no se detiene y se superponen las imágenes de los sensores con la de los obstáculos (figura 15).

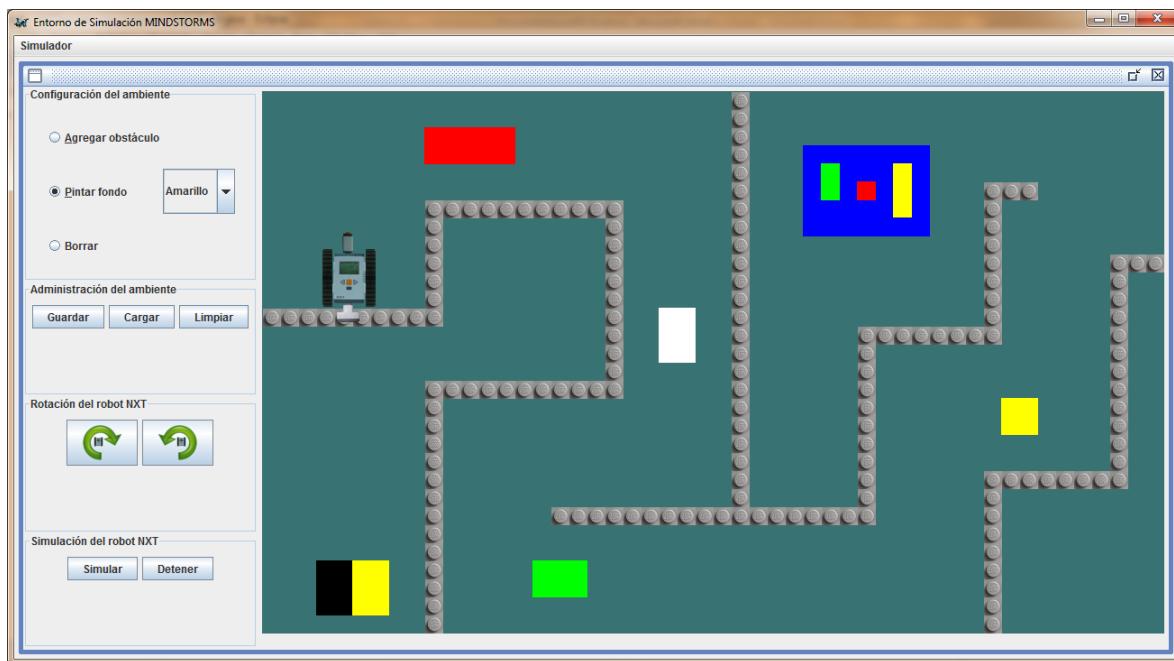


Figura 15: Bug 2 – Superposición de sensores con obstáculos en la interfaz gráfica

- Cuando el robot tiene agregado el sensor “*UltrasonicSensor*” y no se agrega un obstáculo en el ambiente, entonces la simulación no comienza.
- Se ha observado que luego de estar en simulación por varios minutos utilizando el sensor “*UltrasonicSensor*”, el crecimiento en la utilización de recursos es muy elevado (memoria y procesamiento).



4 Caso de estudio

Un entorno de desarrollo integrado, llamado también IDE (sigla en inglés de integrated development environment), es un programa informático compuesto por un conjunto de herramientas de programación.

El uso IDEs está altamente divulgado entre la comunidad de desarrolladores ya que integra distintas herramientas para la escritura, organización, depuración y ejecución programas escritos en diferentes lenguajes de programación.

En nuestro caso elegimos Eclipse, el cual es un IDE de código abierto ampliamente utilizado y con soporte para distintos lenguajes de programación, entre los cuales se encuentra JAVA y el cual cuenta con un plug-in para LeJOS que luego de ser configurado permite compilar y enviar los programas al robot.

A continuación se detallarán los diferentes pasos a seguir para ejecutar una clase con código LeJOS en el simulador.

1. Contar con un IDE Eclipse en la PC que desea ejecutar el simulador, el cual puede descargarse desde <http://www.eclipse.org/downloads/>.
2. Descargar la librería “NXTSimulator.jar” desde nuestro repositorio GitHub²⁴: <https://github.com/sedugri/proyecto/blob/master/lib/NXTSimulator.jar> y la librería “JGameGrid.jar” desde su sitio oficial <http://www.aplu.ch/home/apluhomex.jsp?site=45>.
3. Dentro del Eclipse crear un nuevo proyecto JAVA (figura 16).

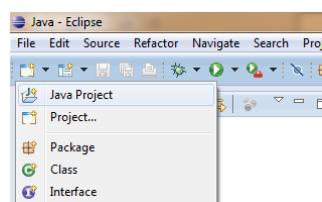


Figura 16: crear un proyecto Java en Eclipse

²⁴ GitHub es un hosting online para nuestros repositorios que utiliza Git (sistema de control de versiones distribuido) para el mantenimiento y versionado del código fuente, añadiendo una serie de servicios extras para la gestión del proyecto y el código fuente. La parte gratuita de este hosting permite alojar nuestro código en repositorios públicos, si queremos repositorios privados entramos en la parte “premium” del servicio.



4. En la pantalla de creación de proyecto asignarle el nombre que deseé, para nuestro caso lo llamaremos “ProyectoCliente”, y presionar el botón “Next”.
5. Se abrirá una pantalla como la que se observa en la figura 17.

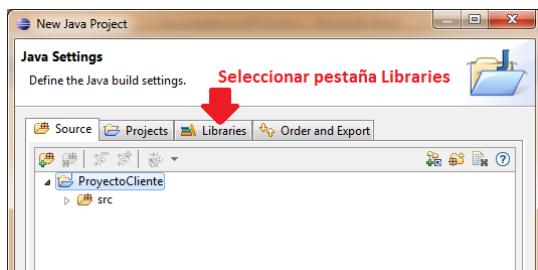


Figura 17: Agregar librerías a un proyecto en Eclipse

En dicha pantalla deberá seleccionar la pestaña llamada “Libraries” para poder importar las librerías necesarias para ejecutar el simulador (figura 18).

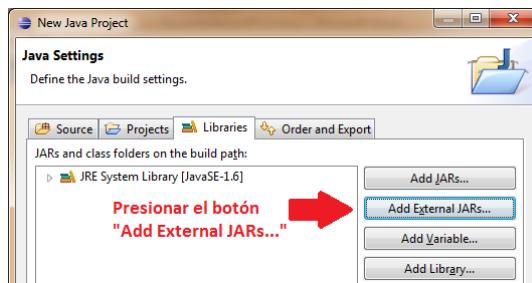


Figura 18: Agregar archivos JAR en Eclipse

6. Presionar el botón llamado “AddExternalJARs” con lo cual se abrirá una ventana que le permitirá buscar y seleccionar las librerías “NXTSimulator.jar” y “JGameGrid.jar” que fueron descargadas y guardadas en el paso 2.
7. Una vez importadas las librerías se observará una pantalla como la siguiente (figura 19):

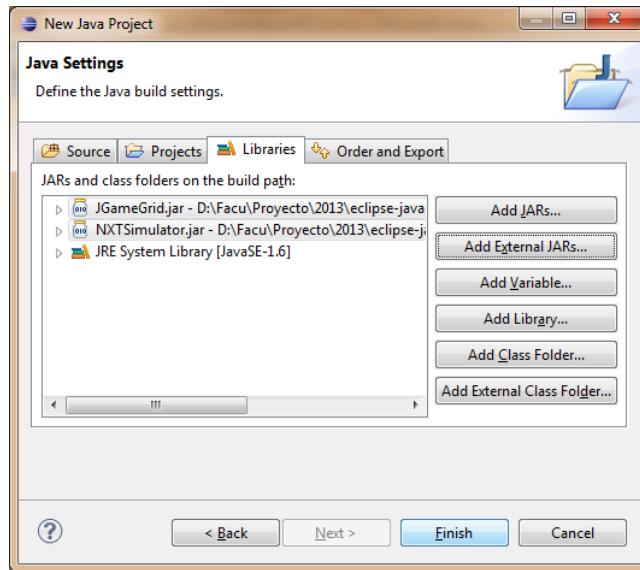


Figura 19: Listado de librerías de un proyecto en Eclipse

Presionar el botón “Finish” para finalizar la creación del proyecto.

8. Dentro del proyecto creado, se deberá crear una Clase JAVA con el nombre y la estructura que se desee para poder escribir el código LeJOS en ella.

En nuestro caso tomaremos como nombre de clase “LejosCode.java” y la crearemos dentro de un paquete llamado “programs”. En la figura 20 se observa la pantalla de creación de dicha clase:

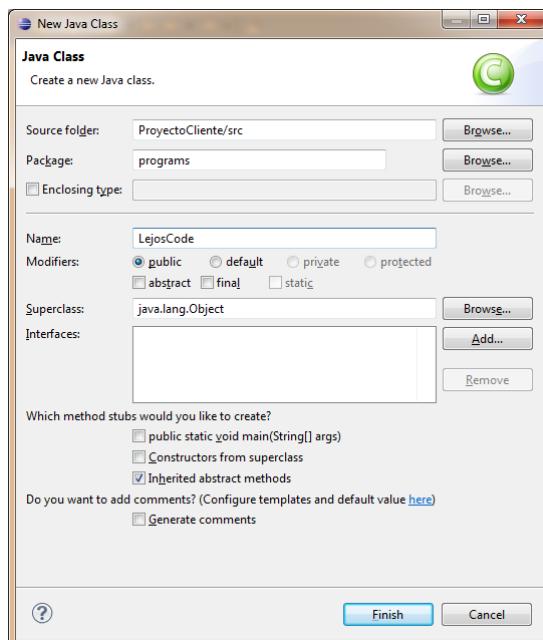


Figura 20: crear una clase en Eclipse

9. Seleccionar la opción “Run Configurations...” del menú “Run” y con lo cual se abrirá una pantalla como la que se observa en la figura 21:

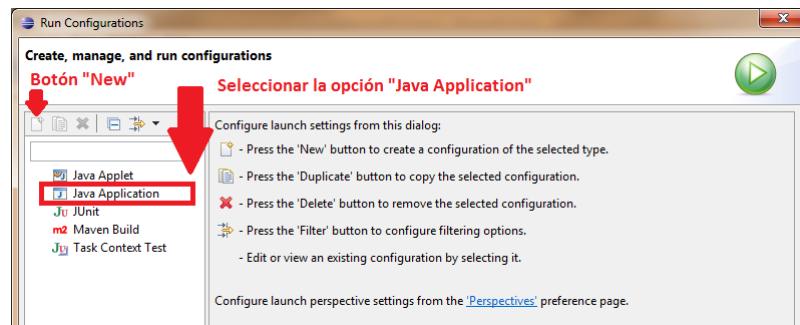


Figura 21: Configuraciones de ejecución en Eclipse

10. Seleccionar la opción “Java Application” y presionar el botón “New” situado en la esquina superior izquierda de la pantalla.
11. En el campo “Name” de la nueva pantalla, escribir el nombre que desee darle a la configuración de la ejecución de la clase LeJOS; en nuestro caso la llamaremos “LejosCodeRun”. En el campo “Main class:” escribir el nombre de la clase principal del simulador NXT “`nxt.simulator.UI.MainFrame`”, con lo cual la pantalla se observará como la figura 22:

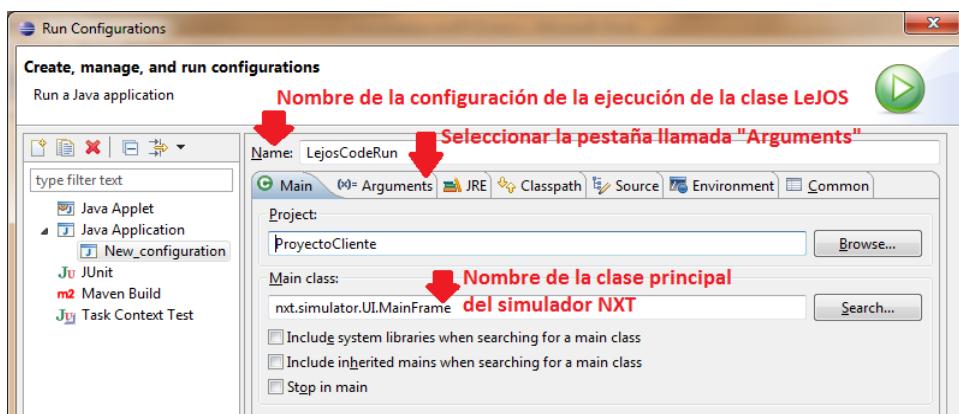


Figura 22: Configuración de ejecución de la clase LeJOS para el simulador - Principal

12. Seleccionar la pestaña llamada “Arguments” y dentro de “Program arguments” escribir el nombre de la clase que contiene el código LeJOS junto con la ruta que la compone, para nuestro caso debemos escribir “`programs.LejosCode`”, con lo cual la pantalla se observará como la figura 23:

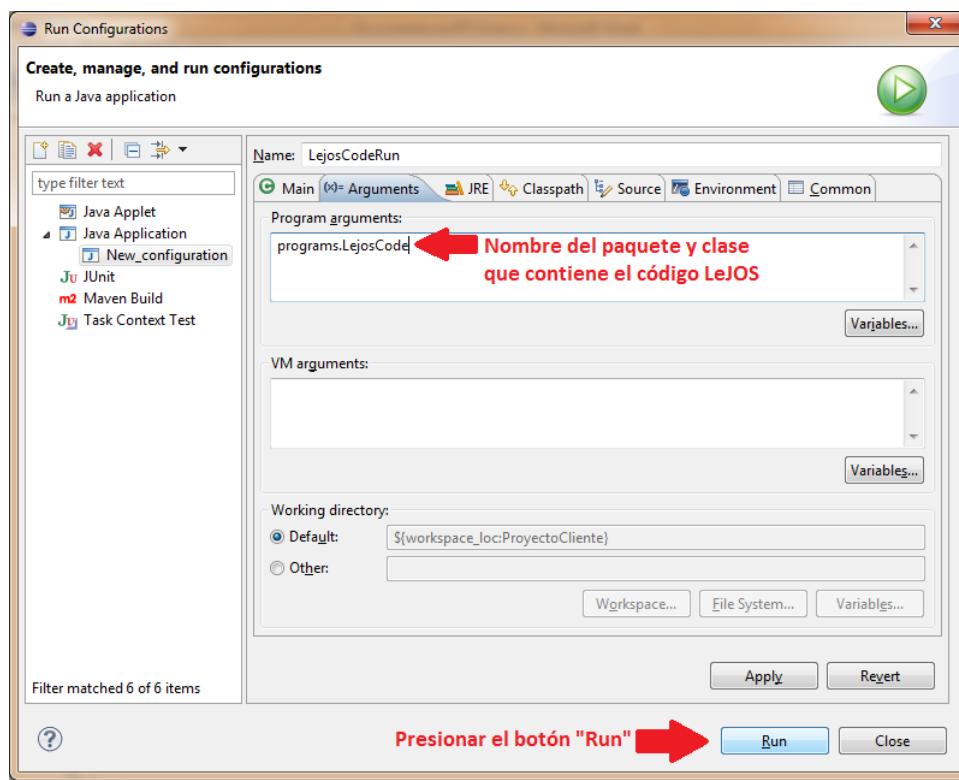


Figura 23: Configuración de ejecución de la clase LeJOS para el simulador - Argumentos

13. Presionar el botón “Run” con lo cual se guardará la configuración para iniciar la aplicación y se iniciará el “Entorno de Simulación MINDSTORMS”.

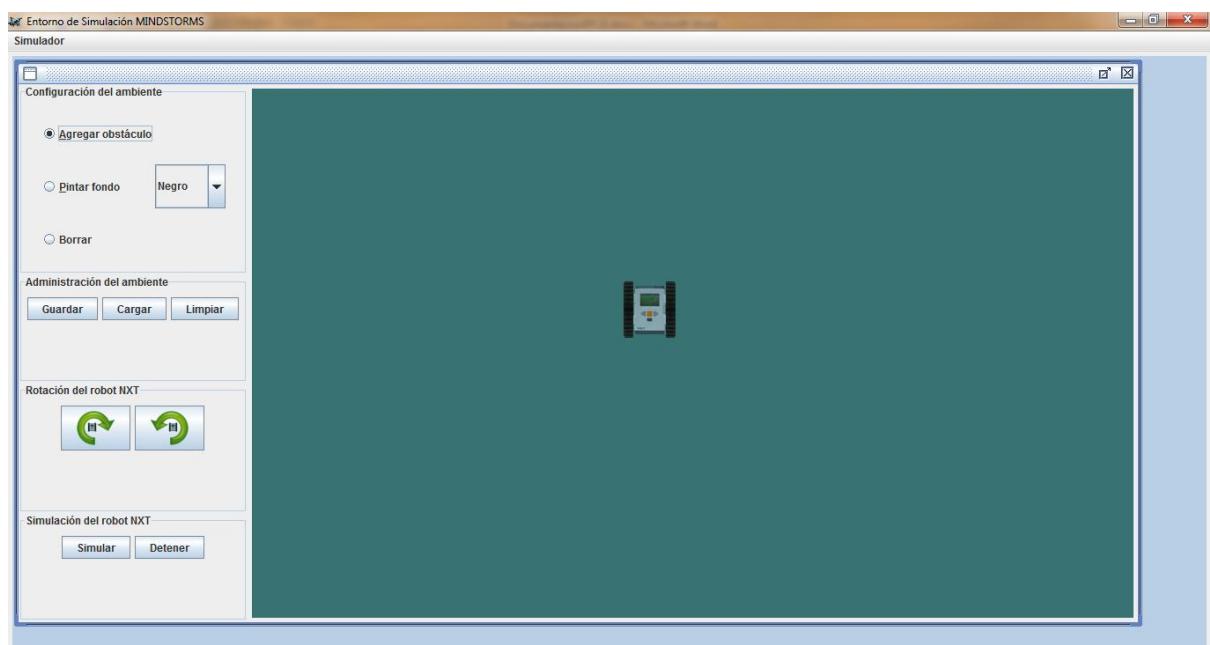


Figura 24: Pantalla principal del Entorno de Simulación MINDSTORMS



14. Una vez dentro del simulador se podrá “*crear*” el ambiente deseado donde se desenvolverá el robot.

Para nuestro caso de estudio deseamos que el robot avance a través de un laberinto, para lo cual crearemos un circuito con diferentes colores de fondo de forma que el robot tenga un comportamiento determinado de acuerdo al color que detecte.

Por lo tanto, el comportamiento esperado del robot será el siguiente:

- Si el sensor detecta el color blanco ○ deberá avanzar hacia adelante;
- Si el sensor detecta el color amarillo ● deberá girar aproximadamente 90° a la derecha.
- Si el sensor detecta el color negro ● deberá girar aproximadamente 90° a la izquierda.
- Si el sensor detecta el color rojo ● el robot deberá detenerse.
- Los colores azul ● y verde ● se utilizarán como límites del laberinto trazado sirviendo estos para corregir el rumbo si el robot sale de dicho circuito.

15. Para crear el ambiente deseado, en primera instancia definiremos un perímetro de obstáculos para que el robot no pueda traspasarlos, para lo cual dentro del panel de configuración del ambiente, deberá seleccionar la opción “Aregar obstáculo”, tal como se observa en la figura 25.

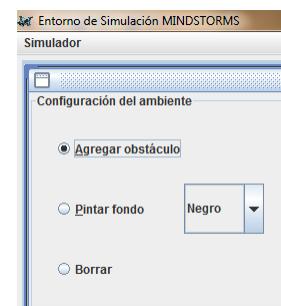


Figura 25: Agregar obstáculo

16. Una vez seleccionada la opción, deberá hacer clic con el botón izquierdo del mouse en los límites del ambiente para ir creando el perímetro. Al hacer un clic en una posición deseada del ambiente, se creará un obstáculo que el robot no podrá sobreponer. Si desea crear un grupo de obstáculos a la vez deberá presionar el botón izquierdo del mouse en la posición inicial, arrastrarlo hasta la posición final y soltar dicho botón, con lo cual se creará una línea de obstáculos que es la misma línea que se siguió con el mouse.

Si por equivocación se agregó un obstáculo en una posición no deseada, deberá seleccionar la opción “Borrar” del panel de configuración del ambiente y hacer clic en el obstáculo que desea quitar.



En la figura 26 se observa como quedó el ambiente luego de crear el perímetro:

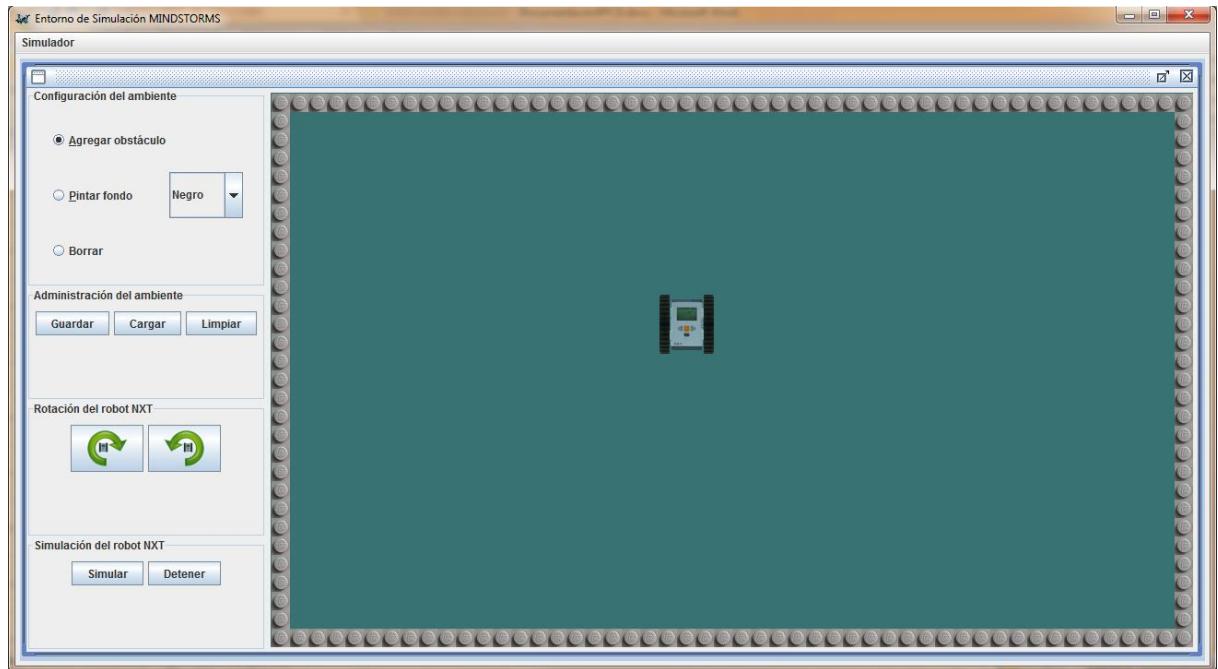


Figura 26: Simulador – Ambiente creado con perímetro de obstáculos

17. Una vez delimitado el ambiente, comenzaremos con la creación del circuito, para lo cual, dentro del panel de configuración del ambiente, seleccionamos la opción “Pintar fondo” y luego dentro del combo de colores elegimos el color deseado para pintar el fondo del ambiente (Figura 27). Comenzaremos seleccionando del combo el color blanco para poder crear el circuito que deberá seguir el robot.

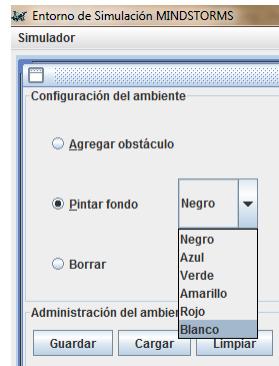


Figura 27: Pintar fondo – Selección de color

18. Al hacer un clic en la posición deseada del ambiente, se pintará un pequeño cuadrado (del tamaño de un obstáculo) de color blanco, de la misma forma, haciendo clic en diferentes posiciones del ambiente podremos ir creando el circuito. También podrá pintar un sector al presionar el botón izquierdo del mouse en la posición inicial, arrastrarlo hasta la posición final y soltar dicho botón, con lo cual se creará una línea pintada que es la misma línea que se siguió con el mouse (Figura 28).

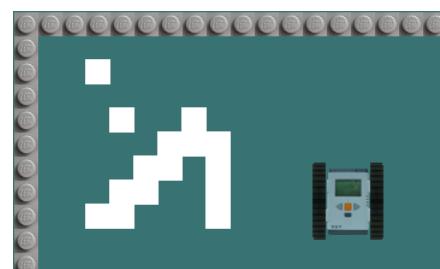


Figura 28: Ejemplo de fondo pintado de color blanco



Si al crear el ambiente, el robot se encuentra en una posición en la que se desea pintar o agregar un obstáculo, podrá mover el robot manteniendo presionado el botón izquierdo del mouse sobre el mismo y llevándolo a la posición deseada donde deberá dejar de presionarlo (función “drag and drop”) para que se le asigne dicha posición al robot. Esta acción podrá repetirse en cuanto desee, teniendo en cuenta que sólo puede moverse mientras la simulación se encuentre detenida.

19. Se deberán repetir los dos pasos anteriores (17 y 18) para cada color que desee agregarse en el ambiente. Para nuestro caso agregaremos los colores amarillo y negro en las esquinas del circuito para que el robot gire cuando detecte dichos colores. Luego bordearemos el circuito internamente con color azul y externamente con color verde y pintaremos el final del circuito de color rojo para que al llegar al mismo el robot se detenga.
20. Una vez creado el circuito debemos posicionar el robot en la posición y dirección deseada (inicio del circuito), para lo cual utilizaremos la función “drag and drop” para mover el robot a la posición deseada y con los botones del panel de rotación del robot NXT rotaremos el robot a la izquierda o derecha según la dirección que debe tener al iniciar la simulación (Figura 29). En nuestro caso debemos girarlo 90° para que quede al inicio del circuito por lo cual presionaremos repetidamente el botón de rotar el robot a la derecha, hasta llegar a la dirección deseada.

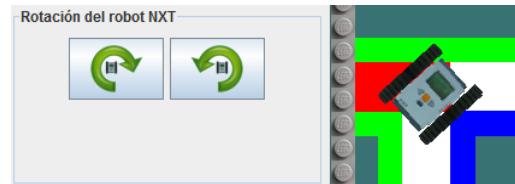


Figura 29: Botones de rotación del robot

En la figura 30 puede observarse el ambiente logrado:

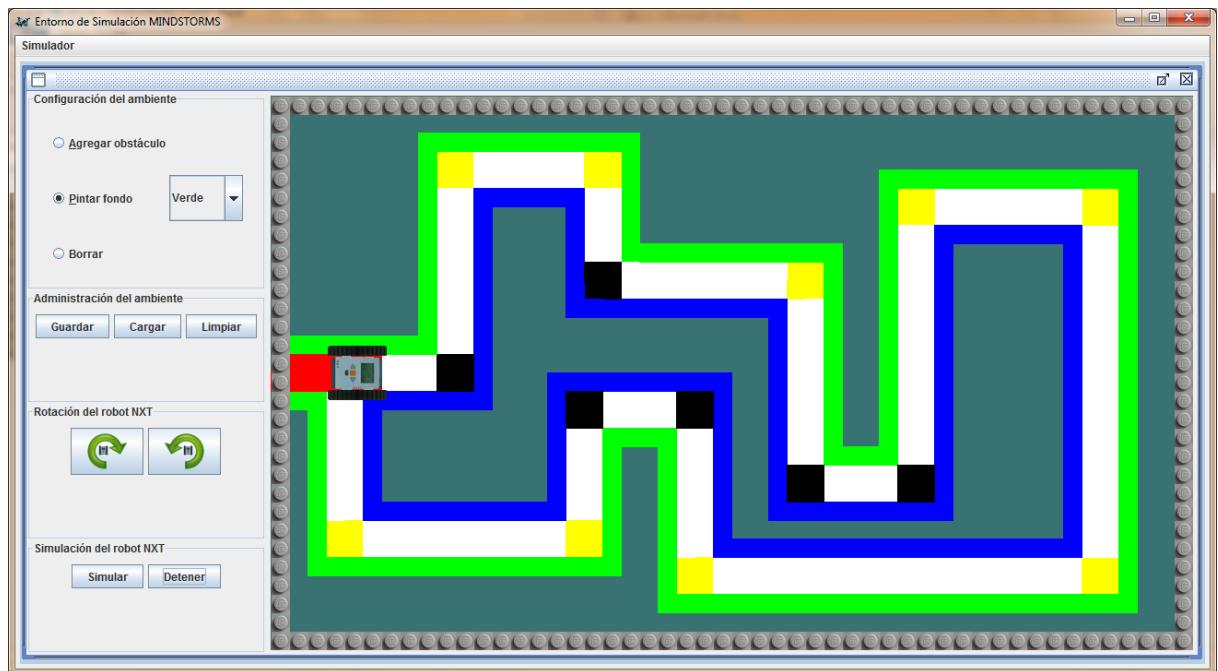


Figura 30: Ambiente del circuito creado

21. Una vez finalizada la creación del ambiente es posible comenzar la simulación presionando el botón “Simular” del panel de Simulación del robot NXT, pero como la clase que contiene el código LeJOS (creada en el paso 8) está vacía el robot no se moverá ya que no tiene ninguna instrucción para hacerlo, por lo tanto guardamos el ambiente presionando el botón “Guardar” del panel de Administración del ambiente, lo cual abrirá un explorador que nos permite guardar el ambiente en la carpeta deseada (el simulador por defecto intenta abrir la carpeta llamada “flat-files” dentro del proyecto, la cual puede observarse en la figura 31). Una vez guardado cerramos el entorno de simulación.
22. Escribir dentro de la Clase `programs.LejosCode` creada en el paso 8 el código LeJOS que guiará al robot en el simulador (Figura 32).

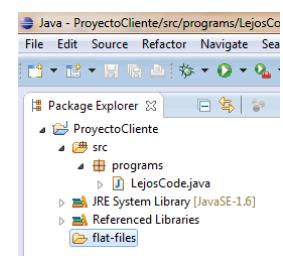


Figura 31: ubicación de la carpeta por defecto del simulador para guardar y/o cargar ambientes



```
import lejos.nxt.Button;
import lejos.nxt.ColorSensor;
import lejos.nxt.Motor;
import lejos.nxt.SensorPort;
import lejos.robotics.Color;

public class LejosCode {
    public LejosCode() throws InterruptedException {
        ColorSensor cs = new ColorSensor(SensorPort.S2);
        Motor.A.stop();
        Motor.B.stop();
        Motor.A.setSpeed(500);
        Motor.B.setSpeed(500);
        while (!Button.ENTER.isPressed()) {
            switch (cs.getColorID()) {
                case Color.WHITE:
                    Thread.sleep(1);
                    Motor.A.forward();
                    Motor.B.forward();
                    break;
                case Color.GREEN:
                    Motor.A.stop();
                    Thread.sleep(500);
                    break;
                case Color.BLUE:
                    Motor.B.stop();
                    Thread.sleep(500);
                    break;
                case Color.YELLOW:
                    Thread.sleep(300);
                    Motor.A.stop();
                    Motor.B.forward();
                    Thread.sleep(3000);
                    Motor.B.forward();
                    Motor.A.forward();
                    Thread.sleep(1000);
                    break;
                case Color.BLACK:
                    Thread.sleep(300);
                    Motor.A.forward();
                    Motor.B.stop();
                    Thread.sleep(3000);
                    Motor.B.forward();
                    Motor.A.forward();
                    Thread.sleep(1000);
                    break;
                case Color.RED:
                    Thread.sleep(500);
                    Motor.A.stop();
                    Motor.B.stop();
                default:
                    Motor.A.stop();
                    Motor.B.stop();
                    break;
            }
            Thread.sleep(1);
        }
    }
    public static void main(String[] args) throws InterruptedException {
        new LejosCode();
    }
}
```

Figura 32: Código del programa LeJOS “LejosCode”



23. Iniciamos el simulador nuevamente seleccionando en la opción “Run History” del menú “Run” la configuración creada en el paso 11 llamada “LejosCodeRun”.
24. Dentro del simulador, presionar el botón “Cargar” del panel de Administración del ambiente y buscar dentro del explorador el ambiente previamente guardado, y una vez cargado el ambiente estaremos en condiciones de iniciar la simulación y probar el comportamiento programado.
25. Al presionar el botón “Simular” se observará como el robot sigue el circuito hasta llegar al final donde el fondo se encuentra pintado de rojo y se detiene.



Figura 33: Búsqueda y selección del ambiente a cargar

Debajo se observan diferentes capturas de pantallas de la simulación realizada:

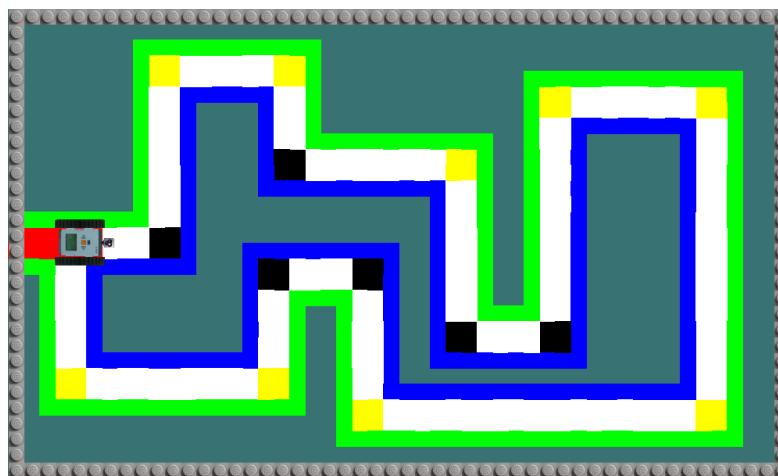


Figura 34: Captura de pantalla 1 de la simulación – Robot detectando color blanco

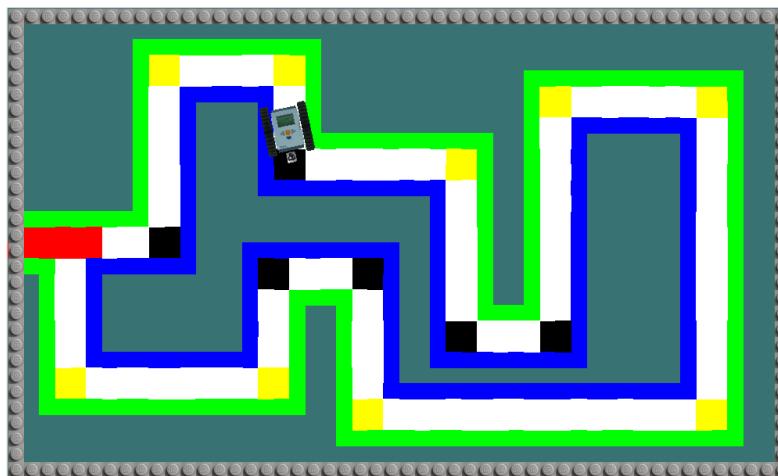


Figura 35: Captura de pantalla 2 de la simulación – Robot detectando color negro

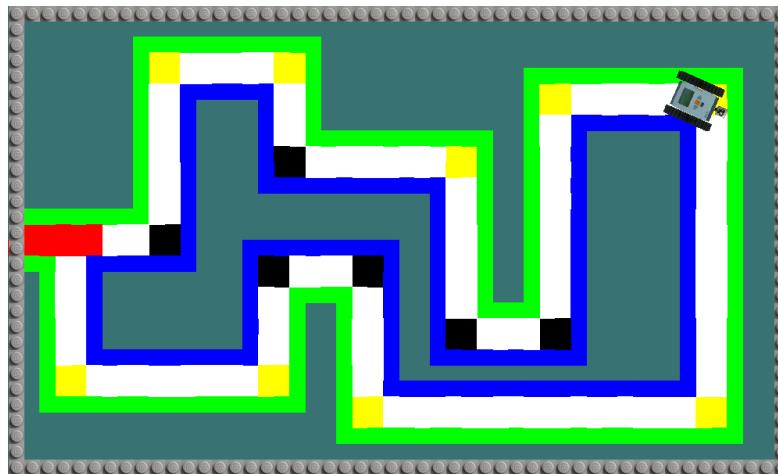


Figura 36: Captura de pantalla 3 de la simulación – Robot detectando color amarillo

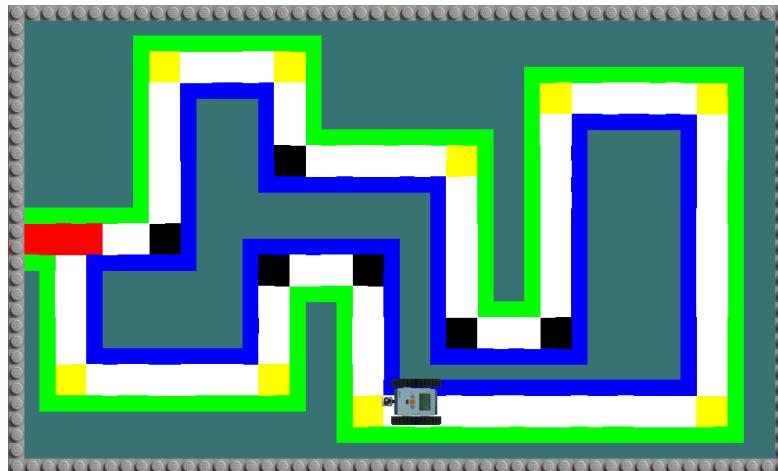


Figura 37: Captura de pantalla 4 de la simulación – Robot detectando color blanco

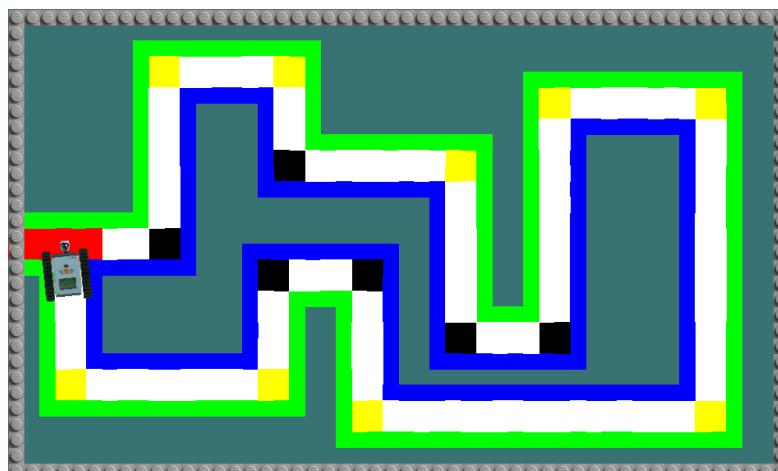


Figura 38: Captura de pantalla 5 de la simulación – Robot detectando color rojo



Una vez realizada la simulación y observando que la funcionalidad es la correcta, puede transferirse la clase “LejosCode” al robot, para lo cual se detallan los diferentes pasos a seguir:

1. Instalar el plug-in LeJOS para Eclipse (el instructivo para la instalación se obtiene en el link: <http://www.lejos.org/nxt/nxj/tutorial/Preliminaries/UsingEclipse.htm>).
2. Crear un nuevo proyecto LeJOS, como se observa en la figura 39.

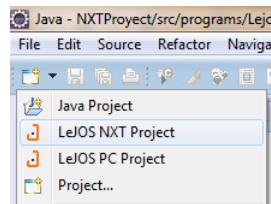


Figura 39: Creación de nuevo proyecto LeJOS

3. Copiar la clase “LejosCode.java” en el nuevo proyecto LeJOS.
4. Conectar el robot NXT a la PC mediante el cable USB.
5. Encender el robot.
6. Dentro del “Package Explorer” del Eclipse, hacer clic con botón derecho en la clase “LejosCode.java” y seleccionar la opción “LeJOS NXT Program” del menú “Run As”, como se observa en la figura 40.

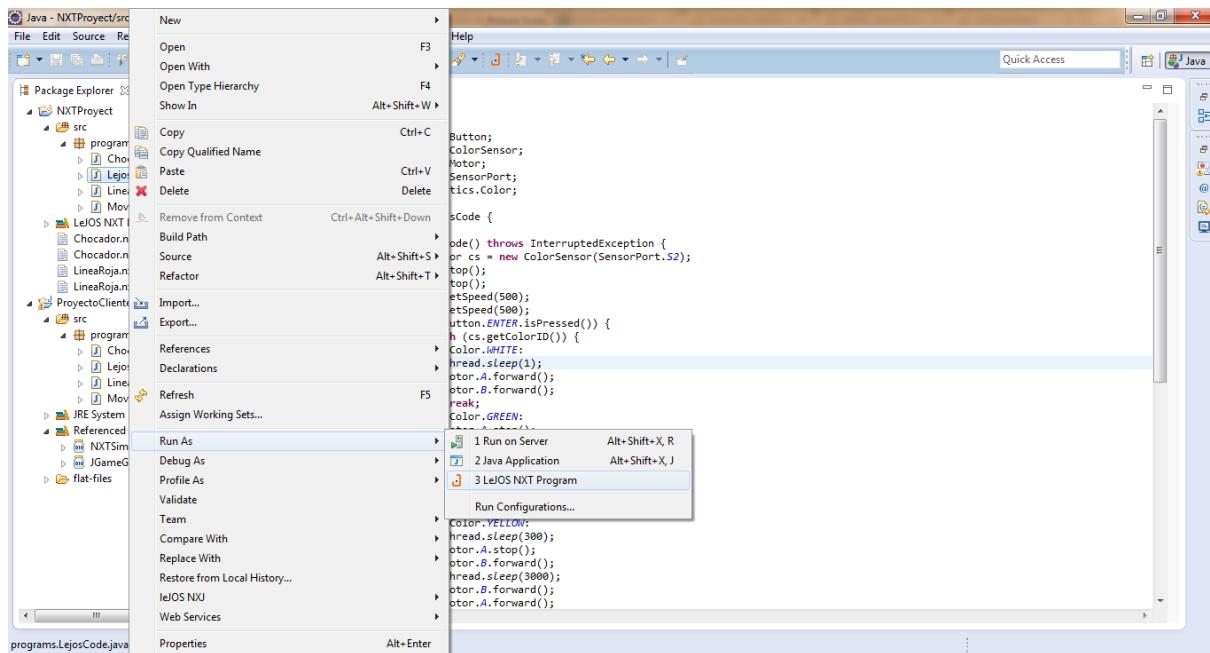


Figura 40: Selección para compilar y transferir la clase seleccionada al robot NXT.



Con lo cual se compilará la clase y se subirá al robot NXT para ser ejecutada cuando se desee.

En la figura 41 se observa la consola del Eclipse luego de una ejecución exitosa:

```
Problems @ Javadoc Declaration Console 
leJOS NXJ
Linking ...
Program has been linked successfully
Uploading ...
Found NXT: NXT 00165313A907
leJOS NXJ> Connected to NXT
leJOS NXJ> Upload successful in 1628 milliseconds
program has been uploaded
```

Figura 41: Consola del Eclipse que muestra que la clase se transfirió correctamente al robot.

Luego podrá seleccionar el programa llamado “LejosCode” desde la pantalla del robot para comenzar la ejecución y para terminarla bastará con presionar el botón ENTER (de color naranja).

Tener en cuenta que para que funcione de la misma forma que el simulador los sensores deben conectarse en los mismos puertos, es decir que para este ejemplo el sensor *ColorSensor* debe conectarse en el puerto 2.



5 Conclusiones

En este último capítulo se presentan las conclusiones a las que se arribaron al ultimar el Proyecto Final de Carrera, los aportes que se obtienen del “Entorno de Simulación MINDSTORMS” y los trabajos futuros.

El entorno de simulación resultante, no sólo permite imitar el comportamiento de un robot Lego MINDSTORMS 2.0, sino que también permite escribir programas en lenguaje LeJOS y ejecutarlos en el robot simulado. El entorno brinda herramientas básicas para definir el ambiente en donde el robot se desenvuelve, pudiendo definir posiciones de obstáculos, colores del suelo, como así también posición y orientación inicial del robot. De esta manera, el entorno se convierte en una herramienta didáctica, fácil de usar y con todas las funcionalidades necesarias para experimentar en el diseño y ejecución de agentes inteligentes que se desempeñan en un determinado ambiente.

Si bien el entorno permite simular un robot Lego con forma de oruga, es flexible la ubicación de los sensores en el mismo. De esta forma, es posible experimentar con distintos comportamientos de acuerdo a los sensores que se le provean.

El entorno de simulación está integrado a la IDE Eclipse de manera de permitir crear programas para el robot físico y analizar su comportamiento en el simulador para realizar las correcciones que sean necesarias antes de su ejecución real, evitando pérdidas de tiempo en la construcción física de escenarios y configuraciones del robot físico.

5.1 Aportes

Dentro de los aportes más relevantes de este Proyecto Final de Carrera se encuentra la utilización del “Entorno de Simulación MINDSTORMS” para la promoción, difusión e incentivación de la carrera Ingeniería en Sistemas de Información en alumnos de escuelas secundarias dentro del marco del proyecto de voluntariado “Conectar Escuela y Universidad: RobotLab2.0”.

Gracias a la colaboración de docentes, estudiantes del Departamento Ingeniería en Sistemas de Información y el CIDISI (Centro de Investigación y Desarrollo de Ingeniería en Sistemas de Información) [11], el martes 8 de octubre de 2013 se llevó a cabo el primer curso de

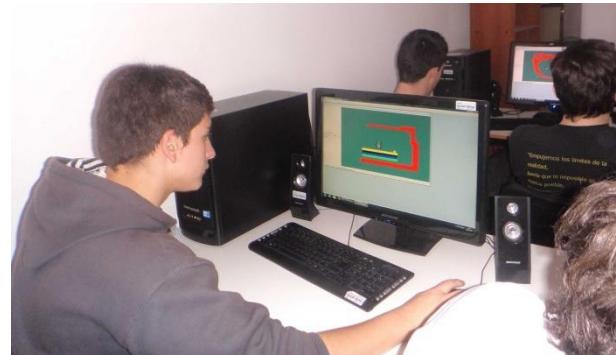
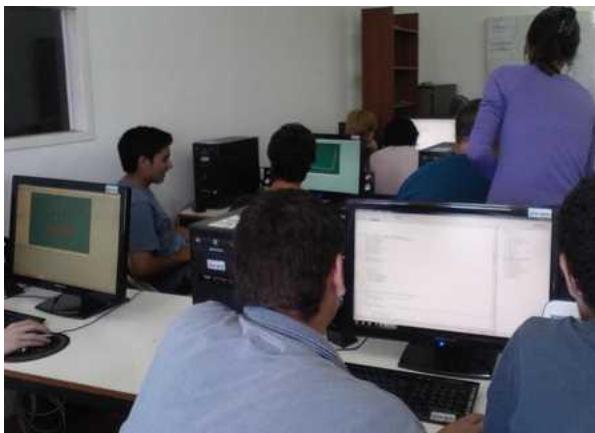


capacitación en programación JAVA empleando la biblioteca LEJOS del robot LEGO NXT 2.0, para los alumnos de 5^{to} año de la especialidad informática de la Escuela de Educación Técnica N° 480 Manuel Belgrano.

En dicho taller se utilizó por primera vez el “Entorno de Simulación MINDSTORMS” a lo cual los alumnos se mostraron muy interesados e incentivados gracias a poder contar con un ambiente gráfico que les demuestre con movimientos del robot lo que ellos programaron. Cada alumno a partir del código escrito con la ayuda de los docentes pudo definir su propio ambiente para poder comprobar cómo actuaba el robot ante los diferentes estímulos, lo cual les proporcionó una visión real de lo que puede crearse mediante la programación.

Las transparencias utilizadas para la exposición del curso fueron creadas a partir del “Entorno de Simulación MINDSTORMS”, las cuales pueden consultarse en el anexo I del corriente informe.

A continuación se muestran fotos tomadas durante el taller, en donde se observa a los alumnos utilizando la herramienta:





Dentro de este proyecto se dictaron 4 cursos más a diferentes alumnos de 4to y 5to año de escuelas técnicas y no técnicas.

Por otro lado no es menor considerar que esta herramienta será usada a partir del año próximo en la cátedra Inteligencia Artificial para la experimentación con agentes inteligentes.

5.2 Trabajos futuros

Este proyecto final de carrera se presenta como base fundamental de un entorno de simulación del robot Lego NXT que, gracias a su diseño modular y flexible, es posible agregar nuevas funcionalidades como ser la implementación de nuevos sensores independientes.

También es posible definir diferentes posiciones a las existentes (tres delanteras y una trasera) para conectar dichos sensores, siempre con la limitación actualmente definida de 4 sensores como máximo dado que es el límite de lo que pude captar el ladrillo inteligente del robot.



Anexo I

Se detallan las transparencias utilizadas para la exposición del taller de programación “Lego NXT 2.0” dictado el martes 8 de octubre de 2013:

Taller de Programación
Lego NXT 2.0

Voluntariado Universitario: RobotLab 2.0

Taller de Programación
Aprendiendo a Programar con un robot
Lego NXT

Voluntariado Universitario: RobotLab 2.0

UTN ■ SANTA FE

El taller...

- 1 Introducción
- 2 Máquinas e Instrucciones
- 3 Estructuras de Programación
- 4 Procedimientos y Lenguaje

Voluntariado Universitario: RobotLab 2.0

UTN ■ SANTA FE

Introducción

- ✓ Máquinas e instrucciones
- ✓ Programas y algoritmos

Voluntariado Universitario: RobotLab 2.0

UTN ■ SANTA FE

Máquinas

- guardar archivo
- entrar a Facebook
- imprimir archivo
- enviar mail
- acelerar
- trenar
- enviar msn
- sacar foto
- llamar

Voluntariado Universitario: RobotLab 2.0

UTN ■ SANTA FE

Instrucciones

- ✓ Cuales serían las instrucciones para el robot?
- ✓ Cada máquina tiene un conjunto de instrucciones básicas: acciones que puede realizar directamente.

detener adelante
velocidad atrás
Leer sensor

Voluntariado Universitario: RobotLab 2.0

UTN ■ SANTA FE



Mover los motores
Cómo logramos efectos en el robot

Girar a la derecha:

```
Motor.A.setSpeed(200);
Motor.B.setSpeed(80);
Motor.A.forward();
Motor.B.backward();
```

Voluntariado Universitario: RobotLab 2.0 UTN SANTA FE

Mover los motores
Cómo logramos efectos en el robot

Girar a la derecha:

```
Motor.A.setSpeed(200);
Motor.B.stop();
Motor.A.forward();
```

Voluntariado Universitario: RobotLab 2.0 UTN SANTA FE

Mover los motores
Cómo logramos efectos en el robot

Girar a la izquierda:

```
Motor.A.stop();
Motor.B.setSpeed(200);
Motor.B.forward();
```

Voluntariado Universitario: RobotLab 2.0 UTN SANTA FE

Instrucciones Básicas

✓ Al robot le podemos instalar diferentes sensores:

- Existen 4 lugares donde puede tener sensores el robot llamados puertos: S1, S2, S3 y S4
- S1 se encuentra delante a la izquierda,

Instrucciones Básicas

✓ Al robot le podemos instalar diferentes sensores:

- Existen 4 lugares donde puede tener sensores el robot llamados puertos: S1, S2, S3 y S4
- S2 se encuentra delante en el centro,

Instrucciones Básicas

✓ Al robot le podemos instalar diferentes sensores:

- Existen 4 lugares donde puede tener sensores el robot llamados puertos: S1, S2, S3 y S4
- S3 se encuentra delante a la derecha,



ALGORITMOS Y PROGRAMA

Pasos para crear un PROGRAMA

4) Escribir el conjunto de instrucciones para el programa "adelante"

```
private void adelante() {  
    while (true) {  
        Motor.A.setSpeed(500);  
        Motor.B.setSpeed(500);  
        Motor.A.forward();  
        Motor.B.forward();  
    }  
}
```

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE

Escribiendo PROGRAMAS

Pasos para crear un PROGRAMA

5) Ejecutar

Botón ejecutar

```
public void adelante() {  
    while (true) {  
        Motor.A.setSpeed(500);  
        Motor.B.setSpeed(500);  
        Motor.A.forward();  
        Motor.B.forward();  
    }  
}  
  
public void lineaRoja() {  
    try {  
        LineaRoja();  
    } catch (Exception e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}  
  
private void LineaRoja() {  
    while (true) {  
        Motor.A.setSpeed(500);  
        Motor.B.setSpeed(500);  
        Motor.A.forward();  
        Motor.B.backward();  
    }  
}
```

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE

Escribiendo PROGRAMAS

Simulador

* Agregar obstáculo
○ Color fondo Negro
○ Borrar

Guardar Cargar Limpiar Simular Detener

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE

Escribiendo PROGRAMAS

6) Crear un ambiente para el robot se puede pintar o agregar obstáculos:

* Agregar obstáculo
○ Color fondo Negro
○ Borrar

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE

Escribiendo PROGRAMAS

6) Seleccionar el botón simular para ver cómo funciona el programa.

* Agregar obstáculo
○ Color fondo Negro
○ Borrar

Guardar Cargar Limpiar **Simular** Detener

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE

Ejemplo 1 - lineaRoja

Ir hacia delante hasta detectar una línea roja, cuando se detecte girar sobre el eje y regresar hacia delante

1) Crear un programa que se llame lineaRoja

```
public void run() {  
    try {  
        lineaRoja();  
    } catch (Exception e) {  
        // TODO Auto-generated catch block  
        e.printStackTrace();  
    }  
}
```

Voluntario Universitario: RobotLab 2.0 UTN SANTA FE



Ejemplo 1 - lineaRoja

Ir hacia delante hasta detectar una línea roja, cuando se detecte girar sobre el eje y regresar hacia delante

2) Escribir el programa *lineaRoja*:

```
private void lineaRoja() throws InterruptedException {
```

Para poder detectar un color necesitamos un sensor de color para agregarlo escribimos lo siguiente:

```
ColorSensor cs = new ColorSensor(SensorPort.S2);
```

El sensor se encuentra delante y en el centro por estar en el puerto S2.

Voluntariado Universitario: RoboLab 2.0 UTN ■ SANTA FE

Ejemplo 1 - lineaRoja

Ir hacia delante hasta detectar una línea roja, cuando se detecte girar sobre el eje y regresar hacia delante

3) Establecemos la velocidad de los motores:

```
Motor.A.setSpeed(500);  
Motor.B.setSpeed(500);
```

Voluntariado Universitario: RoboLab 2.0 UTN ■ SANTA FE

Ejemplo 1 - lineaRoja

Ir hacia delante hasta detectar una línea roja, cuando se detecte girar sobre el eje y regresar hacia delante

4) Repetir siempre este conjunto de instrucciones:

```
while (true) {
```

4.1) Andar hacia delante:

```
    Motor.A.forward();  
    Motor.B.forward();
```

4.2) Si detecta ROJO (RED) entonces doble:

```
    if (AdministratorConstants.RED == cs.getColor()) {  
        Motor.B.stop();  
        Thread.sleep(7000);  
    }
```

Voluntariado Universitario: RoboLab 2.0 UTN ■ SANTA FE

Ejemplo 1 - lineaRoja

Ir hacia delante hasta detectar una línea roja, cuando se detecte girar sobre el eje y regresar hacia delante

5) Cuando termine de doblar entonces siga hacia delante:

```
    Motor.B.setSpeed(500);  
    Motor.B.forward();
```

Voluntariado Universitario: RoboLab 2.0 UTN ■ SANTA FE

Ejemplo 1 - lineaRoja

```
private void lineaRoja() throws InterruptedException {  
    ColorSensor cs = new ColorSensor(SensorPort.S2);  
    Motor.A.setSpeed(500);  
    Motor.B.setSpeed(500);  
  
    while (true) {  
        Motor.A.forward();  
        Motor.B.forward();  
        if (AdministratorConstants.RED == cs.getColor()) {  
            Motor.B.stop();  
            Thread.sleep(7000);  
        }  
        Motor.B.setSpeed(500);  
        Motor.B.forward();  
    }  
}
```

Voluntariado Universitario: RoboLab 2.0 UTN ■ SANTA FE

Fin?

- ✓ Sí!, este es el final de la presentación!
- ✓ Ahora te toca a vos seguir con la Guía de ejercicios....



Voluntariado Universitario: RoboLab 2.0
Ingreso 2013 UTN ■ SANTA FE



¿Dudas?

Lego NXT 2.0

Voluntariado Universitario: RobotLab 2.0



Bibliografía

- [1] Massachusetts Institute of Technology (MIT): Rethinking Artificial intelligence, 2013. <http://web.mit.edu/newsoffice/2009/ai-overview-1207.html>.
- [2] Russell and Norvig (2003). Artificial Intelligence. A modern approach third edition. Prentice Hall.
- [3] SHANON, Robert E. – Simulación de Sistemas. México. Editorial Trillas (1988).
- [4] LEGO: LEGO® MINDSTORMS® NXT 2.0. <http://mindstorms.lego.com/en-us/Default.aspx> (2011); <http://www.lego.com/es-ar/mindstorms?icmp=COARFR15Mindstorms> (2013).
- [5] National Instruments: LabVIEW, 2013. <http://www.ni.com/labview/whatis/esa/>; <http://en.wikipedia.org/wiki/LabVIEW>.
- [6] LeJOS: Java for LEGO Mindstorms, 2013. <http://lejos.sourceforge.net/nxj.php>.
- [7] Plüss, Aegidius: The JGameGrid Framework, An Education Oriented Java Package For Developing Computer Games, 2013. <http://www.aplu.ch/home/apluhomex.jsp?site=45>.
- [8] Sánchez, Guido M. (2011). Navegación autónoma de robots utilizando herramientas de inteligencia computacional. Ingeniería Informática, Facultad de Ingeniería y Ciencias Hídricas, UNL.
- [9] Palacio, Juan. (2006): El modelo Scrum. http://www.navegapolis.net/files/s/NST-010_01.pdf.
- [10] Mountain Goat Software: Scrum, 2013. <http://www.mountaingoatsoftware.com/agile/scrum>.
- [11] UTN-FRSF: Centro de Investigación y Desarrollo de Ingeniería en Sistemas de Información (CIDISI), 2013. <http://extranet.frsf.utn.edu.ar/CIDISI>.