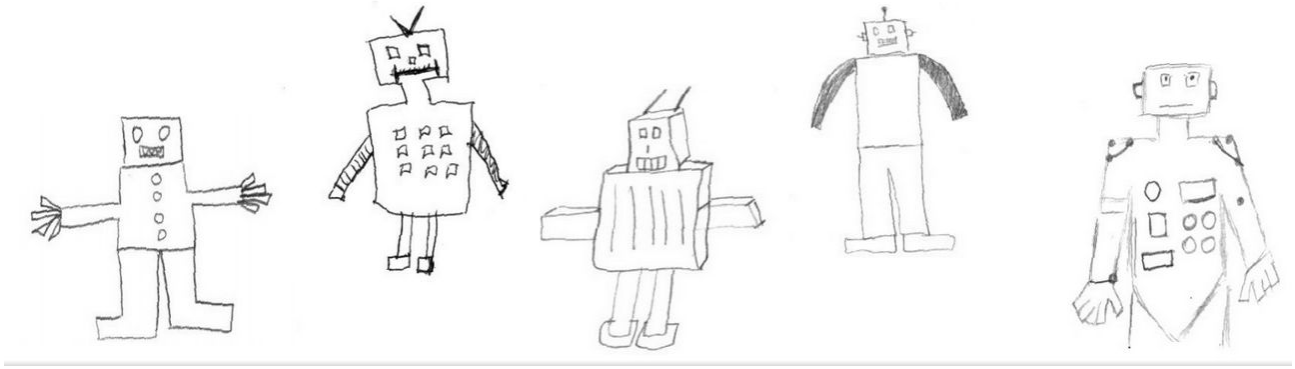
Introducción a la robótica educativa con un enfoque desde la didáctica de la informática.

Colección Alfabetización Digital y Proyectos Educativos

Plan Nacional de Alfabetización Digital

Centros MEC

Julio 2019



Autores
Andrés Aguirre
Bruno Michetti

Corrección
Ivonne Dos Santos
Leticia Rocca
Cecilia Ducos

Colección alfabetización digital y proyectos educativos de Centros MEC, ISSN 2393-6746.

Índice

Recurso educativo abierto

Las ***licencias libres*** son especialmente adecuadas para materiales educativos, ya que permiten acceder, copiar, modificar y distribuir libremente, de manera total o parcial, con cualquier propósito y por cualquier persona o institución, los materiales de estudio. Dichas posibilidades facilitan el acceso y reutilización de los materiales educativos, aportando así a democratizar la educación. El derecho a la educación gratuita es un pilar fundamental de las sociedades democráticas y el acceso libre a los materiales educativos es parte esencial de este derecho.

Pero además, la licencia de este manual exige que todas las ***obras derivadas de él*** (adaptaciones, traducciones, remixes, etc.) otorguen las mismas libertades a sus usuarios. De esta manera, se garantiza que las obras derivadas no serán privatizadas, promoviendo así un ecosistema de obras educativas libres.

En la práctica, los recursos educativos abiertos permiten que estudiantes puedan hacer y compartir copias de manera legal, y que otros docentes puedan reutilizar los materiales en sus propias clases sin pedir permiso. Bajo el modelo de ***Copyright***, estas actividades, tan habituales y necesarias en el ámbito educativo, son generalmente realizadas de manera ilegal, con los peligros y restricciones que ello conlleva. Con la ***producción y el uso de recursos educativos abiertos***, dicho problema se soluciona, dado que estas prácticas pasan a ser legales.

Pero la producción de recursos educativos abiertos no sólo es importante desde el punto de vista de la justicia social. También es ***ventajosa desde el ángulo económico***, ya que aumenta la eficiencia de la inversión pública en educación, al facilitar la actualización de los materiales y eliminar los pagos de costosos derechos de uso.

La ***cooperación*** entre las personas es una pieza fundamental para la construcción social del conocimiento. La promoción de un conocimiento libre y compartido hace a una ciudadanía más consciente y solidaria.

Creative Commons Uruguay

Dedicado a quienes busquen disfrutar de la Ciencia de la Computación.

Prólogo

Este documento constituye un valioso aporte tanto a la didáctica de la ciencia de la computación como a la divulgación de esta disciplina.

En este trabajo, los autores ponen de manifiesto de manera muy clara que la didáctica de la ciencia de la computación (o informática) es una disciplina científica basada en sólidos fundamentos teóricos que sustentan una práctica docente seria, a diferencia de aquella que se basa en cuestiones de opinión. Mediante su participación en nuestro grupo de didáctica de la informática de la Facultad de Ingeniería de la UdelaR, ellos han tomado contacto con un amplio campo de investigación y han aplicado principios teóricos al diseño de pautas para la práctica docente, que constituye gran parte del material de este documento. Nos muestran cómo conceptos que habitualmente son introducidos en la educación superior, son trabajados exitosamente por estudiantes de educación media de una manera que establece el vínculo entre lo que los estudiantes conocen informalmente sobre los mismos y el conocimiento formal que el sistema educativo debe ayudarles a construir.

Por otro lado, con este documento los autores contribuyen al esclarecimiento de una de las confusiones que más daño hace a la educación en informática; nos referimos a la equivocación que significa considerar que la educación en informática se logra con el desarrollo de destrezas en el uso y manipulación de tecnología (alfabetización digital), o que la educación en informática significa formar a los docentes en el uso de tecnología en las aulas (informática educativa). Muy por el contrario, la educación en informática apunta a que los jóvenes conceptualicen y logren formalizar el conocimiento que hace posible la existencia de tecnología. Ello supone un cambio cultural profundo, accesible por medio del estudio serio de las distintas áreas de la informática, donde la didáctica es una de ellas, como lo muestran claramente los autores en este documento.

Por último, creo que vale la pena señalar el gran aporte que los autores brindan a la comunidad educativa, en tanto arrojan luz sobre cuestiones oscuras (como la confusión de que hablamos en el párrafo anterior), y muestran que lograr una educación en informática de calidad no solo es posible sino que es necesario y beneficioso para la formación de los jóvenes.

Sylvia da Rosa,
Setiembre del 2019

Prefacio

Desde el año 2006 hemos estado involucrados en actividades enmarcadas en el uso de robots con fines educativos (popularmente conocido como robótica educativa), principalmente en el Espacio de Formación Integral Butiá (EFI Butiá)[1] de la Facultad de Ingeniería (FING) de la Universidad de la República (UdelaR), donde se realizan talleres de sensibilización en robótica para estudiantes y la asignatura: Formación en robótica educativa para educadores[2]. Han participado en esta asignatura educadores de diversas formaciones previas: maestros, maestros especiales, profesores de enseñanza secundaria de diferentes asignaturas –principalmente informática, matemática y física- ingenieros en computación, estudiantes de ingeniería en computación, psicopedagogos, educadores sociales, docentes universitarios (Otegui. X. 2015).

En el marco de esas actividades, notamos inicialmente un interés muy grande, principalmente de parte de los docentes de educación media, en utilizar los robots como una herramienta en sus clases. Lo que les permitía trabajar los temas de su asignatura de una forma más cercana y motivante para los estudiantes. Pero para lograrlo es inevitable dominar la programación de computadoras y lo común era que tanto estudiantes como docentes no lo hicieran, siendo ésta su primera experiencia en la mayoría de los casos.

Dada esta situación, el uso de los robots en las actividades llevadas adelante por el EFI Butiá se transformó en enseñar a programar utilizando robots. Esto muchas veces fue objeto de reflexión con diferentes compañeros del Instituto de Computación (InCo) de la UdelaR, generando la interrogante acerca de si la robótica es una buena herramienta para aprender a programar, o, por el contrario, la dificultad de manejar la complejidad física de los sensores y actuadores, sumado a aprender un lenguaje de programación y su teoría; es una opción que atenta contra el proceso de generación de conocimiento del estudiante sobre los conceptos de programación.

Nosotros estábamos seguros del poder de los robots, lo veíamos en cada taller; pero teníamos claro que era una "diversión dura", como

[1] Espacio de Formación Integral Butiá: https://www.fing.edu.uy/inco/proyectos/butia/, visitada en mayo del 2019.

[2] Programa de la asignatura: Formación en robótica educativa para educadores: https://www.fing.edu.uy/sites/default/files/cursos/2019/anexos/36686/Formaci%C3%B3n%20de%20formadores%20robotica.odt-1.odt.pdf, visitada en julio del 2019.

apuntaba Seymour Papert haciendo referencia al aprendizaje de este tipo de conceptos: "*es un trabajo duro pero que genera mucha pasión"*, en contraposición a otros enfoques de "hacerlo divertido, hacerlo fácil".(Papert, S. 2002) Asimismo, compartíamos la preocupación del equipo de InCo, lo que nos llevó a buscar soluciones a estos problemas en el área de la Didáctica de la Informática, concretamente: cómo enseñar estos conceptos a personas que no tenían conocimientos previos de informática o robótica.

Esperamos que esta publicación sea de ayuda para educadores, principalmente quienes trabajan en el proyecto de robótica educativa de Centros MEC, y que quieran aprender a programar robots y enseñar a hacerlo, buscando el placer en entender conceptos "duros", como decía Seymour Papert (Papert, S. 2002).

En esta publicación buscamos sintetizar los conceptos más importantes necesarios para programar autómatas utilizando robots educativos, junto con algunos casos de estudio donde se trabaja con una metodología fundada en un modelo didáctico. Dicho modelo puede ser utilizado para elaborar ejemplos de cómo utilizar la robótica educativa, aprovechando la motivación en el uso de elementos tangibles, como los robots.

Entendemos que no es posible utilizar la robótica educativa como una herramienta (enfoque TIC) si previamente no se dominan los conceptos fundamentales de las ciencias de la computación que tratamos en esta publicación, nuestra visión de lo que debería ser la robótica educativa incluye entender estos conceptos.

Más allá de la licencia seleccionada para este trabajo, fomentamos que este material pueda ser accedido por quien lo desee en un formato editable y abierto, que permita generar obras derivadas a partir del mismo. Esta idea que tratamos de fomentar, va más allá de solamente acceder y modificar el texto del documento e involucra a las diferentes actividades y desafíos planteados, que pueden ser modificados y utilizados libremente.

Los ejemplos presentados en esta publicación pueden aplicarse en diferentes kits de robótica educativos presentes en Uruguay, como ser el Robot Butiá, Lego NXT, Lego WeDo, Fischer y Microbit, entre otros.

Andrés Aguirre y Bruno Michetti
Julio del 2019

Agradecimientos

Los autores quieren agradecer a los compañeros de Centros MEC, por el valioso apoyo en diferentes actividades que sirvieron de insumo para este trabajo: Facundo Bermúdez, Cecilia Ducos, Ivonne Dos Santos, Ximena Tauré, Mónica Benitez, Carla Margenat, Lucía Rodriguez, Valentina Lasalvia, Virginia Loustau, Dayana Aguiar, María Noel Araujo, Denisse Laporta, Pedro Montaña, Irene Pereira, Guillermo Artola, Aimará Mendieta, Sadiel Hormaza, Hernán Camejo, Gustavo Castiñeira, Rosina Espiga, Juan Pablo Ancelotti y Rodrigo Ortiz.

A Sylvia Da Rosa que con su curso de Didáctica de la Informática, nos abrió la puerta a un área que nos marcó y supimos disfrutar mucho.

A Walter Bender por la pasión, generosidad y el apoyo al Proyecto Butiá durante estos 10 años.

A Ximena Otegui por sus aportes en las formaciones de robótica educativa para educadores.

A Carla Margenat por la idea de hacer este material; la pasión y energía para que el proyecto de robótica educativa de Centros MEC naciera.

A Agustín Guerra por el trabajo en equipo y haber sido una fuente diaria de enseñanzas.

A toda la comunidad Butiá, educadores, desarrolladores y entusiastas de la robótica, por haber creado un proyecto hermoso.

A Néstor Larroca y a sus alumnos del liceo nro 2 de La Paz, por su apoyo y participación en las actividades didácticas fundamentales para este trabajo.

A las autoridades del MEC por el apoyo al programa de robótica educativa.

"Nosotros no le damos contenido sino herramientas, ellos son los creadores. Si sólo tienes la oportunidad de ser usuario, no te podrás convertir en un maestro. En cambio, si eres desarrollador, sí lo puedes lograr".

Walter Bender

1 Introducción

Desde inicios del año 2014, el proyecto de Robótica Educativa de Centros MEC busca sensibilizar a sus participantes en el proceso de creación de tecnología, utilizando al robot como herramienta, entendiendo la tecnología como una actividad cultural. La propuesta de Robótica Educativa de Centros MEC persigue como objetivo que quienes participen de las actividades puedan trascender la frontera que divide ser usuario de tecnología de ser creador. Por lo general el público objetivo de la propuesta domina la tecnología en el plano de la acción, la utilizan a diario para consumir contenidos culturales, pero no identifican la posibilidad de crear tecnología como una opción cultural por sí misma, que pueda ser a futuro objeto de estudio y desarrollo profesional, o como una actividad que genere placer, a diferencia de lo que ocurre con otras actividades culturales más populares como ser el teatro o la música, entre otras, donde se identifica claramente el disfrute asociado a su práctica. Compartimos lo que Guzmán Trinidad junto con otros autores postulan: "El objetivo no es el de lograr aumentar los ingresos a las escuelas de ingeniería. Pocos niños elegirán carreras de ciencias de la computación. Pero la mayoría de los niños van a vivir en un mundo que estará en gran parte impreso por la computación." (Trinidad, G. et al. 2015).

Se espera que esta publicación sirva como guía docente para diseñar actividades utilizando la robótica como herramienta didáctica, trabajando desafíos que permitan al estudiante reflexionar sobre nociones generales de robótica y conceptos asociados a las ciencias de la computación, como la programación y el diseño de autómatas.

En esta publicación se presenta una propuesta de robótica educativa que surgió a partir de la experiencia desarrollada por los autores en el EFI Butiá y en Centros MEC junto con la búsqueda de su formalización aplicando una instancia del modelo didáctico desarrollado por el grupo de didáctica de la informática del InCo (Da Rosa, S. 2015).

1.1 ¿Para qué sirve la robótica educativa?

Los robots pueden ser una herramienta didáctica poderosa y flexible que permita a los estudiantes reflexionar sobre el funcionamiento de los objetos tecnológicos a partir del trabajo con propuestas que, mediante la construcción y programación de robots, generen el modelado de problemas significativos. Estos problemas, que utilizaremos a modo de desafíos robóticos; son conocidos por el estudiante en el plano de la acción y forman parte de su día a día, como por ejemplo moverse al son de unos tambores, barrer el piso de una habitación o hacer dibujos.

A partir de estos desafíos, los estudiantes pueden reflexionar acerca de las acciones que ellos mismos utilizan para resolver la misma dificultad y las repercusiones que obtienen. Esto involucra elaboraciones mentales de orden superior que permiten conceptualizar el problema hasta lograr su formalización, lo que a su vez hace posible modelarlos en un lenguaje de programación que lleve al robot a resolver el desafío planteado. Resolver estos problemas significativos, como moverse al son de unos tambores, en el que sabemos a qué se refiere y podemos experimentarlo nosotros mismos, permite a los programadores novatos reflexionar sobre cómo ellos resuelven el problema, bailando en el caso del ejemplo; para poder luego "enseñarle" a una máquina a hacer lo mismo.

El proceso que transitan los estudiantes para lograr la conceptualización de un problema y finalmente su formalización mediante la implementación de un agente, les permite generar conocimiento acerca de conceptos asociados con las ciencias de la computación y la robótica a partir de elementos que les resultan cercanos y motivantes, como son los robots aplicados a la resolución de problemas de su interés (Aguirre, A. Da Rosa S. 2017), los que llamamos problemas significativos. Con el rol de acompañar a los estudiantes en su reflexión y acción, proponiendo y guiando, los docentes también forman parte del proceso. En la Sección 4 de esta publicación veremos un ejemplo en el que el educador guía la reflexión del estudiante para lograr el diseño de un autómata que juega de forma automática a un videojuego.

1.2 Nuestra propuesta de robótica educativa

La propuesta de Robótica Educativa de Centros MEC ha sido acompañada en su consolidación inicial por el EFI Butiá; tanto el proyecto Butiá como el de Alfabetización Digital de Centros MEC tienen como objetivo facilitar el acceso a bienes y servicios culturales como la tecnología y el conocimiento para crearla. El Proyecto Butiá persiguió este objetivo mediante la creación de un robot educativo basado en software y hardware libre, que puede ser replicado fácilmente a bajo costo, utilizando incluso materiales reciclados, junto con instancias de formación para facilitar ese proceso (Benavides, F. et al. 2013). Dichas formaciones fueron tomadas por los docentes de Centros MEC, quienes oficiaron como replicadores, ofreciendo actividades a lo largo del país. Ambos proyectos se complementaron y enriquecieron, y se llegó a una propuesta que va más allá de la utilización de un kit robótico: propicia su autoconstrucción y modificación en lo relativo al hardware e incluye una metodología para aprender a programarlo.

Lograr transformar el hardware en un robot puede ser una actividad altamente enriquecedora y motivante que requiere conocimientos de programación y diseño de agentes. Si bien existe un grado de complejidad en este proceso, tiene como contrapartida el ser muy disfrutable. Este material busca ser un aporte para docentes que se proponen acompañar y guiar a sus estudiantes en la creación de autómatas. Proponemos comenzar por resolver tareas que a los los estudiantes les son cercanas (problemas significativos), situándolos en la posición de reflexionar acerca de cómo hacen ellos mismos para resolver dichas tareas. Esto es, éstas ya son dominadas por los estudiantes en el plano de la acción, pero aún no conocen su formalización para que una máquina pueda ser programada para resolver las tarea por sí misma (de forma autónoma). En este sentido el proceso de reflexión implica conceptualizar y formalizar sus ideas en un lenguaje de programación, como fue mencionado en la Sección 1.1. Esta metodología didáctica para la enseñanza de la disciplina de la robótica educativa, está basada en la Epistemología Genética de Jean Piaget. (Piaget, J. 1977) y en la propuesta de aplicación a la computación de Sylvia da Rosa[3] (Da Rosa, S. 2015).

Por otro lado, nuestra propuesta también busca mediante la programación de robots, eliminar ciertos preconceptos comunes acerca de las computadoras y los robots, mostrando que se trata de máquinas muy simples, donde la percepción de complejidad en las acciones que realizan está dada por la programación que podemos darles.

En estos seis años de trabajo en el proyecto de robótica educativa de Centros MEC hemos encontrado una oportunidad, desde la educación no formal, para la enseñanza de la informática como ciencia, que estimula y favorece una actitud proactiva, crítica e inventiva de quienes participan de los talleres y actividades de Robótica Educativa.

1.3 Objetivo de la publicación

Esta publicación persigue como objetivo responder a las preguntas: ¿Qué enseñar? ¿Cómo enseñar? y ¿Para qué enseñar? robótica educativa, de manera que sirva para la formación de los educadores en informática y como insumo en la elaboración de propuestas didácticas.

A partir de una introducción a los fundamentos generales de programación y las herramientas para poder diseñar el comportamiento de autómatas, y su posterior programación, tratamos de dar respuestas a la pregunta: **¿qué enseñar en robótica educativa?**; presentamos un modelo didáctico y aplicaciones a los conceptos presentados, que nos

[3] Ver Sección Fundamentos de la propuesta de Silvia Da Rosa, 2015

permite encontrar respuestas a la pregunta: **¿cómo enseñar robótica educativa?**; también son presentadas en este trabajo las razones por las cuales entendemos es importante enseñar robótica educativa, tratando de aportar en la búsqueda de respuestas para la pregunta: **¿para qué enseñar robótica educativa?.**

1.4 Estructura del documento

En la Sección 2 es una breve introducción a la Epistemología Genética, necesaria para comprender el modelo didáctico específico de las ciencias de la computación utilizado para el desarrollo de la propuesta de robótica educativa que se presenta en el resto de la publicación; en la Sección 3 se aborda el concepto de autómata y robot, se analizan sus diferencias y se presenta el formalismo de máquina de estados; la Sección 4 auspicia de nexo entre la Sección 2 y la 3, a través de una propuesta para trabajar con los estudiantes; en la Sección 5 se presentan las estructuras de datos y de control básicas del paradigma de programación imperativo. Finalmente, la Sección 6 está enfocada a un caso de estudio que trata de sintetizar todos los conceptos trabajados en este texto en un problema concreto a resolver con el Robot Butiá.

2 Fundamentos teóricos

La construcción de conocimiento acerca de algoritmos y estructuras de datos es un proceso que puede modelarse mediante la ley general de la cognición de Jean Piaget (Piaget, J. 1977). Nuestra propuesta didáctica está basada en la aplicación de dicha ley, la cual presentaremos en esta sección y aplicaremos en la Sección 4 para la enseñanza de los conceptos de autómata que presentaremos en la Sección 3.

Esta sección ha sido elaborada tomando como referencia el material y bibliografía de la asignatura: Didáctica de Algoritmos y Estructuras de Datos, del InCo[4].

2.1 Epistemología Genética

La Epistemología Genética de Jean Piaget, es una teoría que explica la construcción del conocimiento como un proceso y estudia la transición en el ser humano de un nivel menor de conocimiento, a otro que puede ser juzgado como mayor.

Se trata de un modelo que puede ser usado en todos los dominios y en todos los niveles de desarrollo de las personas. La teoría fue elaborada sobre dos fuentes importantes de información:

- Estudios empíricos realizados por Piaget sobre el proceso de construcción del conocimiento, en sujetos desde el nacimiento hasta la adolescencia.
- Un análisis crítico de la historia de las ciencias, elaborado por Jean Piaget y Rolando García para investigar el origen y desarrollo histórico de las ideas científicas, los conceptos y las teorías (Castorina, A. 2001).

Nombradas las dos fuentes principales, se puede decir que Piaget investigó la relación entre el desarrollo psicogenético en la evolución de la inteligencia de los niños y el desarrollo sociogenético de las ideas y teorías en las ciencias. La noción más importante de este paralelismo se centra en la construcción del conocimiento de un ser humano a través del pasaje por tres etapas:

1. La etapa ***intra***: El sujeto se centra en elementos vistos de forma aislada, persiguiendo un objetivo y viendo los resultados de sus acciones.
2. La etapa ***inter***: El sujeto toma en cuenta la relación entre los elementos y sus acciones sobre los mismos.
3. La etapa ***trans***: El sujeto construye internamente las estructuras generales de los elementos y de las acciones sobre los mismos, e

[4]Sitio web de la asignatura Didáctica de Algoritmos y Estructuras de Datos: https://eva.fing.edu.uy/course/view.php?id=874, visitada en julio del 2019.

integra las construcciones de las etapas anteriores como casos particulares.

2.1.1 Ley General de la Cognición

El problema más importante para este abordaje del desarrollo epistémico se encuentra en determinar el rol de la experiencia y de las estructuras operacionales del individuo en el desarrollo del conocimiento, y en estudiar los instrumentos por los cuales se construye dicho conocimiento antes de ser formalizado. Piaget atacó profundamente este problema en sus estudios sobre la psicología genética y elaboró la Ley General de la Cognición (también llamada Ley de la Toma de Conciencia) que rige a la relación entre el ***"saber hacer"*** (una persona sabe realizar acciones, por más de que pueda no saber cómo las hace) y la ***conceptualización***. Dicha ley explica la ***toma de conciencia***, en la construcción de conceptos, a través de la relación nombrada anteriormente, que surge en la interacción del sujeto con objetos en un entorno para resolver un problema determinado y/o realizar una tarea.

La ley se representa de la siguiente forma:

$$C \leftarrow P \rightarrow C'$$

- ***P* (Periferia)** refiere a la reacción más inmediata del sujeto operando sobre el objeto para resolver un problema y/o realizar una tarea. El sujeto persigue un objetivo y ve los resultados pero no es consciente ni de las acciones ni de las razones de su éxito (o fracaso).
- En ***C*** el sujeto se vuelve consciente de la coordinación de acciones sobre el objeto.
- En ***C'*** el sujeto se vuelve consciente de las modificaciones generadas en el objeto por las acciones, y de las propiedades intrínsecas del mismo.

Las flechas representan el mecanismo interno del proceso del pensamiento. En la construcción de conceptos el sujeto parte desde la Periferia *P* y se desplaza hacia *C* y *C'*.

El desplazamiento desde *P* se da gracias a herramientas cognitivas (que también se construyen), explicadas en la teoría de Piaget.

2.1.2 Herramientas Cognitivas

La Epistemología Genética describe los instrumentos cognitivos que hacen posibles la toma de conciencia y la construcción de conocimiento. A estas herramientas Piaget las llama ***Abstracción*** y ***Generalización***.

Los dos tipos de abstracción son la ***empírica*** y la ***reflexiva.*** La primera supone abstraerse del objeto físico y de la acción concreta,

permitiendo la formación de explicaciones causales (p. e.: como empujé el vaso, el vaso se movió). La segunda implica abstraerse de la coordinación concreta entre las acciones, y actúa de dos maneras: permite la proyección de la coordinación de operaciones en el plano de la acción a la coordinación de inferencias en el plano conceptual (p. e.: si empujo el vaso entonces el vaso se mueve) y también permite la reorganización de las estructuras mentales para integrar lo nuevo. El "motor" de la abstracción es la motivación del sujeto y su búsqueda de las razones de su éxito o fracaso.

Los dos tipos de generalización son la ***inductiva*** y la ***constructiva.*** Cuando el sujeto enfrenta nuevas situaciones que presentan variaciones y similitudes con respecto a situaciones anteriores, se da un desequilibrio en las estructuras cognitivas, que deben ser transformadas nuevamentes para volver al equilibrio, haciendo posible la construcción del conocimiento apropiado para resolver la nueva situación. Una vez que un método particular ha sido comprendido, el razonamiento del sujeto intenta generalizar dicho método aplicándolo a todas las situaciones, haciendo deducciones a partir de observaciones sobre nuevos objetos; a esto se le llama generalización inductiva. Cuando el sujeto realiza internamente un conjunto de reflexiones e inferencias sobre sus acciones dando lugar a nuevos métodos, a esto se le llama generalización constructiva.

3 Robots y Autómatas

En esta Sección analizaremos desde un punto comparativo los conceptos de robot y autómata y presentaremos la herramienta de *máquinas de estado,* que nos permitirá modelar el comportamiento de autómatas. Los conceptos de autómatas y robots a menudo se confunden en la bibliografía, donde de hecho no existe una definición de consenso al respecto. En esta sección tratamos de aportar algunos elementos que ayudarán a identificar las diferencias entre ambos conceptos, según bibliografía considerada de referencia en el área de robótica.

3.1 Agentes

Un agente se encuentra inmerso en un medio con el cual interactúa a través de sus sensores y actuadores, los sensores le permiten percibir el medio y en función de ello el agente puede responder, utilizando sus actuadores para producir cambios en el mismo medio (Russell, S. J.; Norving, P. 2004), ver Figura 3.1.A. Tanto los robots como los autómatas, a quienes veremos a continuación, pueden ser modelados como un agente, ya que podemos describirlos como algo capaz de percibir el medio, también llamado entorno, y actuar en consecuencia. La percepción del entorno se realiza mediante los sensores y la actuación mediante los actuadores, según reglas que son especificadas en el control del agente, ver Figura 3.1.A.

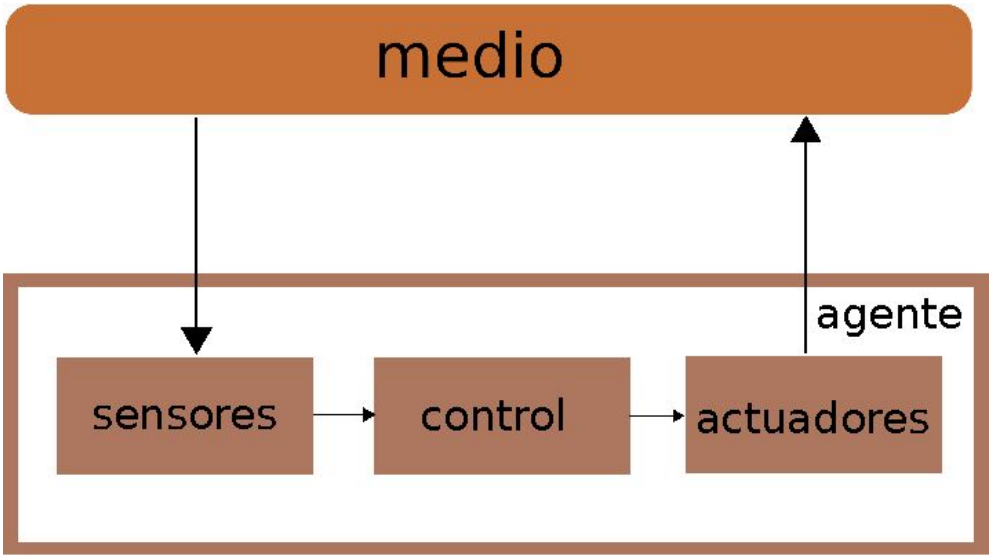


Figura 3.1.A: Modelo de agente.

3.1.1 Autómatas

Los automatismos que utilizamos a diario pueden ser modelados como agentes, algunos ejemplos son: el ascensor, el microondas, el calefón, el tele-peaje, entre otros. Los agentes que modelan a estos autómatas son simples y cercanos a nuestra vida diaria, por lo que resultan un excelente caso inicial de estudio a la hora de diseñar actividades con los participantes del programa de robótica educativa. Un buen ejercicio que muchas veces se utiliza en las actividades que realizamos es tratar de identificar autómatas en nuestra vida y reflexionar si pueden clasificarse como un robot.

3.1.2 Robots

El concepto de robot es muy cercano al de autómata, ambos perciben el entorno con sus sensores y actúan en el mismo mediante sus actuadores, como veíamos en el modelo de un agente de la Figura 3.1.A. La diferencia fundamental está en la capacidad de adaptación del agente al entorno cambiante: cuando hablamos de robots esperamos que estos "aprendan" de sus experiencias anteriores y puedan adaptarse al entorno. En cambio un autómata es un agente más simple, que siempre realiza una misma tarea de una misma forma, independientemente de lo que ocurra en el entorno. Como ejemplo podemos imaginarnos un agente encargado de poner tornillos en piezas de una línea de producción, si en el lugar de la pieza colocamos otro objeto, como una mochila, el autómata va a intentar colocar el tornillo en la mochila sin adaptarse al cambio que ocurrió en el entorno.

En la Tabla 3.1.2.A, podemos ver un resumen de las diferencias entre un autómata y un robot.

Autómata	**Robot**
Percibe el entorno mediante sensores.	Percibe el entorno mediante sensores
Actúa modificando el entorno mediante actuadores.	Actúa modificando el entorno mediante actuadores.
Se caracteriza por realizar tareas repetitivas, siempre de la misma forma, sin tener en cuenta los cambios del entorno.	**Se caracteriza por adaptarse a los cambios del entorno**

Tabla 3.1.2.A: Comparación entre un Autómata y un Robot

3.2 Caso de estudio

Para analizar el concepto de autómata y agente trabajaremos el diseño de un dispositivo hipotético, similar al de la Figura 3.2.A, que permita a una persona ciega cruzar la calle.

Figura 3.2.A: Dispositivo hipotético que ayuda a una persona ciega a cruzar la calle[5]

El dispositivo posee los siguiente sensores: un bastón con sensor de contacto y un par de cámaras que permitirán al dispositivo reconocer el semáforo y su color, y un par de auriculares para informar de eventos al usuario.

La persona va a circular hasta que llega a un cruce con semáforo, si el semáforo está rojo o amarillo el dispositivo debe de alertar al usuario, avisándole que no debe cruzar mediante la reproducción de un audio con el mensaje "atención! no cruzar, debe esperar por luz verde". La persona espera por la luz verde, la cual es indicada por el dispositivo reproduciendo el audio con el mensaje "puede cruzar" cuando el semáforo se pone en verde. La persona comienza a cruzar y el dispositivo mientras el semáforo está verde continúa informando el mensaje; si el semáforo se pone amarillo el dispositivo informa a la persona mediante la reproducción del mensaje "atención, luz amarilla!" para que pueda estar alerta a la situación de peligro. Cuando la persona llega al cordón de la vereda el impacto del bastón con el cordón de la misma provoca que se active el sensor de contacto, por lo que el

[5] imagen extraída de: How scientists are helping blind people see with their ears, https://www.vox.com/2014/11/7/7171119/blind-sonar-echolocation, visitada en mayo del 2019.

dispositivo asume que la persona terminó de cruzar y reproduce el audio "fin de cruce", quedando listo para un nuevo cruce.

El programa que controla al dispositivo será el encargado de implementar la etapa de control del agente (Figura 3.1.A); utilizando para ello la información del entorno proporcionada por los sensores y la posibilidad de producir cambios en el entorno mediante órdenes a los actuadores.

Este dispositivo corresponde a un agente del tipo autómata, ya que en los requerimientos no especificamos que pueda adaptarse a nuevas situaciones, si la persona se para frente a una luz que no sea la del semáforo se va a comportar como si lo fuese, carece de la capacidad de aprender por sí sólo; además realiza un trabajo repetitivo, otra de las características de los autómatas.

¿Cuáles son las órdenes que tenemos que programar en el control del agente para ayudar a la persona a cruzar la calle? ¿De qué van a depender?

Esas órdenes, o salidas, van a depender del semáforo (luz verde, luz roja), que podemos ver como una entrada para el autómata obtenida a partir de sus sensores, y la situación de la persona respecto de la calle (cruzando, esperando por luz verde, circulando), a estas situaciones las llamamos estados. En la Tabla 3.2.A podemos ver una representación de cuáles son las órdenes a ejecutar por el autómata y cuándo hacerlo. Podemos imaginar al comportamiento del autómata como una función de las entradas *i* y los estados *s*, *f(i,s),* donde la salida depende del estado del autómata y de las entradas; por ejemplo, la salida que el autómata dará cuando la luz está amarilla va a depender si la persona está circulando o si está cruzando; si la persona está circulando no puede comenzar a cruzar con la luz amarilla, por lo que el dispositivo debe reproducir el audio "atención! no cruzar, debe esperar por luz verde", por el contrario, si la persona ya se encuentra cruzando, necesitamos advertirle para que pueda apurarse a terminar de cruzar y por consiguiente el dispositivo debe de reproducir el audio "atención, luz amarilla!", ver filas 2 y 6 de la Tabla 3.2.A.

estados	**entradas**	**salidas**
circulando	luz roja	reproducir "atención! no cruzar, debe esperar por luz verde"
circulando	luz amarilla	reproducir "atención! no cruzar, debe esperar por luz verde"

esperando por la luz verde	luz verde	reproducir "puede cruzar"
cruzando	el bastón no toca el cordón	reproducir "puede cruzar"
cruzando	el bastón toca el cordón	reproducir "fin de cruce"
cruzando	luz amarilla	reproducir "atención, luz amarilla!"

Tabla 3.2.A: Estados, entradas y salidas.

3.3 Máquinas de estado

Para ayudarnos a diseñar el control del autómata existen herramientas que nos permiten modelar su diseño, una de esas herramientas son los diagramas de máquinas de estado. Las máquinas de estado nos permiten formalizar, mediante diagramas, el comportamiento del autómata. Pudiendo especificar qué debe hacer el control del autómata ante determinada situación.

Como vimos en el ejemplo del dispositivo para la persona ciega, existen situaciones (estados) que el autómata debe determinar y recordar para poder tomar la decisión de qué orden enviar a los actuadores ante igual situación percibida del entorno. Un ejemplo es el caso de la luz amarilla, donde el autómata debe comportarse de diferente manera en función de si la persona está cruzando la calle o si está circulando, ante la misma situación del entorno (luz amarilla). Mediante el uso de los diagramas de máquinas de estado, esas situaciones quedan más claras.

Esas situaciones a recordar, las cuales determinan la actuación ante determinado evento ocurrido en el entorno, son los ***estados*** de la máquina de estado y los vamos a representar mediante un óvalo. Un estado posee un ***conjunto de entradas***, que se corresponden con los valores obtenidos por los sensores del autómata, y un ***conjunto de salidas***, que se corresponden con las órdenes enviadas a los actuadores. Existen transiciones entre los estados, definidas por ***funciones de transición***, que toman como argumento los valores de entrada y el estado actual y determinan cuál es el próximo estado de la máquina.

Para el ejemplo del dispositivo que asiste a la persona ciega visto en la Sección 3.2, tenemos la entrada *color* que representa el color del

semáforo, la cual toma valores en el conjunto {verde, amarillo, rojo} según el color del semáforo; y la entrada *bastón*, la cual toma el valor 1 cuando el bastón choca contra un objeto y 0 cuando está libre, como puede verse en la Tabla 3.3.A. La salida *parlantes*, la cual puede tomar valores del conjunto: {0,1,2, 3, 4} se describe en la Tabla 3.3.B.

entrada	**valores que toma**	**descripción**
bastón	1, 0	1 corresponde al botón presionado, 0 suelto
color	verde, amarillo, rojo	representa el color del semáforo

Tabla 3.3.A: Entradas del autómata

salida	**valores que toma**	**descripción**
parlante	0, 1, 2, 3, 4	0 reproduce por los parlantes el audio "atención! no cruzar, debe esperar por luz verde". 1 reproduce por los parlantes el audio "puede cruzar". 2 reproduce por los parlantes el audio "fin de cruce". 3 reproduce por los parlantes el audio "atención, luz amarilla!" 4 no reproduce sonido

Tabla 3.3.B: Salidas del autómata

Como mencionamos anteriormente el comportamiento del autómata cambia dependiendo del estado y de la entrada. La entrada es algo externo al autómata, depende del entorno, pero el estado es interno y este debe ser actualizado por el autómata en función de la entrada y el estado actual, esta función la llamaremos de *transición;* a partir de la información de la Tabla 3.3.C queda determinada la función de transición para el ejemplo, junto con la salida de cada estado a partir del estado actual y la entrada.

Por ejemplo, si el estado es '*circulando'* y la luz está roja, *color = rojo*, se da la salida '0' y se actualiza el estado a '*esperando por la luz verde'*. Por otro lado cuando el estado es '*esperando la luz verde'* y la luz cambia de roja a verde, *color = verde*, la salida es '1' y el se actualiza el estado a '*cruzando';* finalmente cuando el estado es '*cruzando'* y el bastón toca el cordón (*bastón = 1*) la salida es '2' y se actualiza el estado a '*circulando'*. En este momento el autómata está listo para volver a enfrentarnos a un cruce. En la Tabla 3.3.C podemos ver para cada par *(estado, entrada)* cuál es próximo estado, determinando de esta manera la *función de transición*.

estados	**entradas**	**salidas**	**próximo estado**
circulando	color = rojo	0	esperando por la luz verde
circulando	color = amarillo	0	esperando por la luz verde
circulando	color = verde	1	cruzando
esperando por la luz verde	color = verde	1	cruzando
esperando por la luz verde	color = rojo	4	esperando por la luz verde
esperando por la luz verde	color = amarillo	4	esperando por la luz verde
cruzando	bastón = 0	1	cruzando
cruzando	bastón = 1	2	circulando
cruzando	color = amarillo	3	cruzando

Tabla 3.3.C: Estados, entradas, salidas y función de transición.

La misma información de la Tabla 3.3.C podemos representarla gráficamente utilizando diagramas de máquina de estados; en la Figura 3.3.A podemos ver un diagrama que representa la máquina de estados del problema planteado.

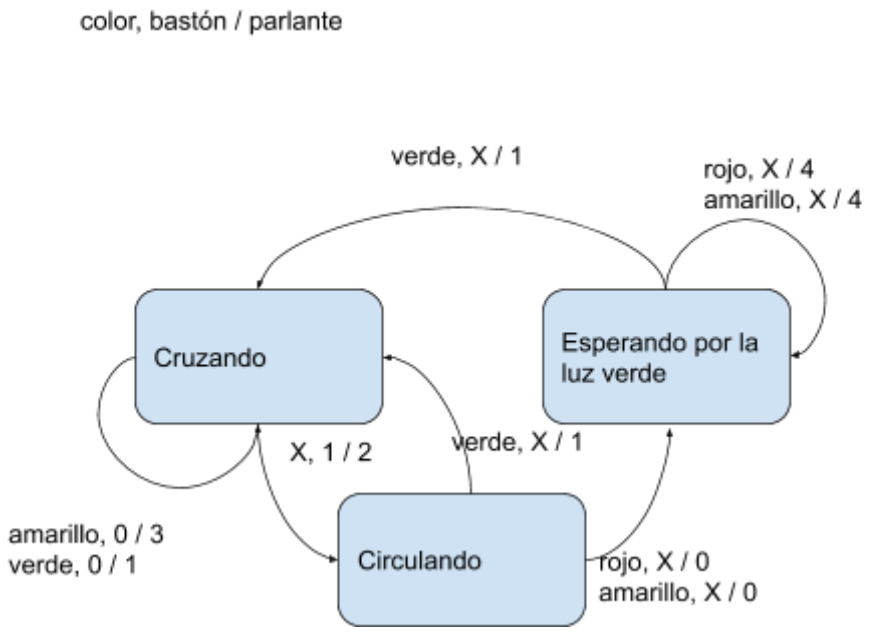


Figura 3.3.A: Diagrama de máquina de estados para el autómata que ayuda a cruzar a una persona ciega.

Notar que utilizamos la notación: entrada1, entrada2, … entradaN / salida1, salida2, … salidaN en cada una de las transiciones, en el ejemplo de la Figura 3.3.A corresponden con color, bastón/ parlante; siendo 'color' y 'bastón' las entradas y 'parlante' la salida. Para cada transición se especifican los valores que toma tanto las entradas como la salidas, si no importa alguno de los valores que puede llegar a tomar la entrada o salida lo especificamos mediante el uso del valor 'X'; el sentido de la flecha en la transición determina cual es el próximo estado. Por ejemplo, cuando el autómata se encuentra en el estado: "Esperando por la luz verde" y el semáforo está de color rojo o amarillo, entonces se mantiene en dicho estado, con salida: '4', pero si la luz cambia a verde el próximo estado será: "Cruzando" y la salida: '1', notar que en ninguno de los casos fue tenido en cuenta el valor del bastón, ya que no es relevante para esos casos.

En la Sección 6 veremos cómo a partir del diseño de un autómata utilizando máquinas de estados se puede programar dicho autómata en una máquina.

4 Didáctica de autómatas

En las Sección 3 vimos los conceptos fundamentales para el diseño de autómatas, en la Sección 2 se presentó el modelo de Piaget acerca de cómo las personas generamos el conocimiento; en esta sección veremos una instancia de dicho modelo, como ejemplo de cómo aplicar la metodología didáctica propuesta en la enseñanza de los conceptos de autómata. Esta sección está basada en el trabajo: "Students teach a computer how to play a game" (da Rosa, S. Aguirre, A. 2018), donde se utiliza un videojuego sencillo, como es el juego de LumberJack[6] para trabajar el concepto de autómata y su diseño, junto con la propuesta se presenta una serie de clases, a modo de implementación de la propuesta didáctica.

Dicha propuesta fue validada con estudiantes de enseñanza media, en el liceo número 2 de la ciudad de La Paz, en las clases de informática a cargo del profesor Nestor Larroca, en el marco de las actividades llevadas adelante por el grupo de Didáctica de la Informática del InCo. El grupo de estudiantes estaba conformada por 25 alumnos: 13 de 13 años, 8 de 14 años, 1 de 15 y 3 de 16 años. Cada clase consta de dos partes de 45 minutos cada una, separadas por un recreo de 5 minutos.

4.1 Propuesta

Un videojuego es un elemento cercano a los estudiantes. Por lo general se sienten motivados y rápidamente logran visualizar qué es lo que hay que hacer para ganar y qué es lo que los lleva a perder, es decir, lo dominan ampliamente en el plano de la acción. La propuesta educativa consiste en diseñar un autómata que pueda jugar al mismo videojuego sin asistencia de ninguna persona (de forma autónoma): enseñarle a una computadora a jugar al videojuego.

Mediante un conjunto de preguntas se guía al grupo de estudiantes en la reflexión acerca de las reglas que utilizaron para jugar al juego, de forma de que tomen conciencia de las acciones que realizan y el efecto que estas generan en el videojuego. A partir de estas reglas se propone su uso como insumo para el diseño de un autómata.

Las clases fueron diseñadas para ayudar a los estudiantes a tomar conciencia de la relación entre: por un lado, el algoritmo y las estructuras de datos —que corresponden con el texto de un programa,

[6] El videojuego LumberJack se encuentra disponible en: http://tbot.xyz/lumber/

especificado en un lenguaje de programación— y, por el otro, los elementos que forman parte de la ejecución por una máquina. Estos elementos relativos a la ejecución, tienen una contraparte en el texto del programa, donde son modelados como estructuras de datos, pero físicamente corresponden con elementos de hardware, como los periféricos de entrada (mouse, teclado, touch screen, entre otros) y salida (pantalla, entre otros).

Utilizando la entrada, el texto del programa va a evaluar las reglas codificadas y en función del resultado de la evaluación genera acciones sobre las estructuras de datos, especificadas en el texto, lo que produce una salida para el autómata a diseñar. En el caso concreto del videojuego de LumberJack, esas salidas generadas por el autómata simulan eventos de teclado o mouse (presionar una tecla o mover y hacer click con el mouse) que producen una entrada en el videojuego, y por consecuencia un cambio en el modelo del mundo, o entorno, como fue presentado en el modelo del agente, ver Figura 3.1.A. En la pantalla del computador se mapea el estado del mundo, por lo que el autómata, al igual de como hace una persona, la utilizará como una entrada, permitiéndole detectar los cambios producidos en el modelo del mundo del videojuego para volver a evaluar las reglas codificadas y de esta forma seguir jugando.

Se proponen una serie de clases para que los estudiantes puedan enseñarle a una computadora a jugar a un videojuego, utilizando el modelo de toma de conciencia de Piaget. A fin de obtener el objetivo planteado, en las clases se va a trabajar un conjunto de actividades que se ordenan de la siguiente manera:

1. Jugar al juego.
2. Reflexionar sobre las reglas empleadas y expresarlas como reglas de inferencia en idioma español.
3. Expresar las reglas del punto anterior como una máquina de estados (un autómata).
4. Programar el autómata en TurtleBots[7] y hacer que el programa juegue al juego.

4.2 El juego

El videojuego LumberJack tiene como personaje principal al leñador Jack, quien debe talar un árbol con su hacha, sin que ninguna rama golpee su cabeza, mientras la altura de dicho árbol decrece y las ramas van bajando. Tanto el personaje como las ramas, pueden encontrarse a la izquierda o a la derecha del árbol; y cada vez que el

[7] Programa Turtle Bots, https://www.fing.edu.uy/inco/proyectos/butia/files/package/, visitada en enero del 2019.

leñador golpea con el hacha, el tronco se acorta y las ramas bajan, generando el peligro de golpear en la cabeza al personaje. A medida que se golpea con el hacha, se suman puntos, y el juego consiste en sumar la mayor cantidad posible, sin que ninguna rama golpee a Jack. Además hay una cuenta regresiva, a la cual se le suman segundos a medida que el jugador avanza en el juego, y esto exige talar el árbol aumentando la velocidad, poniendo a prueba los reflejos de la persona que controla al leñador, que debe entonces, además de esquivar el golpe de las ramas, evitar quedarse sin tiempo.

El jugador sólo tiene dos botones para presionar, el izquierdo y el derecho, que al apretarlos hacen que Jack golpee al árbol del lado correspondiente al botón, es decir que presionar el botón izquierdo, genera que el personaje se coloque a la izquierda del árbol y golpee con el hacha. De forma análoga sucede con el botón derecho.

4.3 Instanciando la ley general de la cognición

Sylvia da Rosa y Andrés Aguirre implementan una instancia de la ley general de la toma de conciencia de Piaget para describir casos donde el estudiante debe enseñarle una acción a una computadora en base al estado del entorno. La consigna que se propone a los estudiantes en el referido trabajo, y que es desarrollada durante tres clases, consiste en enseñar a jugar al juego LumberJack a una computadora, especificando las reglas que ellos mismos emplean al jugar como un algoritmo en lenguaje natural (en idioma español). Por otro lado, se propone también diseñar un autómata que juega al juego, y finalmente escribir y ejecutar el programa que implementa el autómata para que juegue al juego (da Rosa, S. Aguirre, A. 2018).

En la Sección 2 se introduce la Epistemología genética de Piaget, y su Ley General de la Cognición, que rige la relación entre el sujeto que sabe hacer algo, y la conceptualización de lo que hace y lo que sucede con lo que hace. Recordemos que la ley dice que el sujeto interactuando con un objeto parte de una Periferia, en la que sabe hacer cosas pero aún no es consciente de lo que hace, ni de las acciones que produce; y luego se traslada a C, su conciencia sobre la coordinación de acciones realizadas, y a C', su conciencia sobre el resultado de dichas acciones en el objeto.

Para clarificar dicha ley, vamos a suponer que el sujeto es un niño o una niña en un entorno, que interactúa con un conjunto de piedritas, que serían el objeto. Supongamos también que ese sujeto ya sabe contar (se encuentra en P), y juega con las piedritas contándolas, cambiándolas de lugar y volviendo a contarlas, llegando a que sin importar el orden de las mismas, al contarlas siempre obtiene la misma

cantidad. En este ejemplo podemos afirmar que C es el conjunto de acciones que el sujeto ejecuta sobre las piedritas (acomodarlas, reordenarlas, contarlas) y C' es la constatación del sujeto sobre cómo van quedando las piedras y sobre el resultado de los diferentes conteos, que siempre es el mismo. El niño o la niña, ¿Sobre qué estaba tomando conciencia? ¡Lo estaba haciendo sobre la conmutatividad de la suma! Aprovechando la Ley mencionada, un educador podría conducirle a que tome conciencia de que para todo par de números a y b, se cumple que a + b = b + a, de una forma más cercana que explicándoselo con letras que representan a los números de forma abstracta.

De un tiempo a esta parte los videojuegos se han convertido en algo cercano, donde este ejemplo de jugar con piedritas, no es tan habitual como lo es jugar con una tablet o un celular a un videojuego. En este tipo de ejemplos aparece algo más, y es el aparato con el que juegan, que puede ser una consola, una tablet, un celular o lo que sea, pero que en última instancia termina siendo una computadora. Por lo tanto se puede decir que tenemos dos niveles de toma de conciencia: en un nivel el sujeto toma conciencia sobre lo que quiere que la computadora haga, y en el otro nivel el sujeto debe lograr que la computadora comprenda lo que quiere que haga; ¡y que efectivamente lo haga!

La actividad que se presenta en esta sección, trata estos dos niveles. El sujeto debe enfrentar el desafío de conceptualizar las reglas del videojuego LumberJack, construir de forma abstracta un algoritmo para resolver dicho problema y luego enfrentar el segundo desafío: lograr que la computadora pueda jugar al videojuego.

Para resolver el primer desafío, el estudiante debe identificar la relación causal que existe entre la actuación que realiza el autómata y el estado del mundo del videojuego, permitiéndole identificar las reglas especificadas en el control del agente; una vez que el estudiante ha conceptualizado estas reglas ha construido una especificación del autómata para el problema planteado. El estudiante ya no se encuentra más en "la periferia", ha conseguido la toma de conciencia planteada en el primer nivel.

Como se mencionaba al describir la propuesta en el comienzo de esta sección, la metodología a emplear es la de proponer a los estudiantes reflexionar sobre las acciones que ellos mismos realizan en función del estado del mundo del videojuego. Este acercamiento ya era utilizado en 1980 por Seymour Papert, refiriéndose a la programación de una tortuga autómata, donde plantea: "programar la tortuga comienza por reflexionar acerca de cómo uno mismo lo hace y qué desearía uno que la tortuga hiciera". En este caso, programar un autómata que juega un juego comienza por hacer reflexionar al estudiante acerca de lo que él/ella hace al jugar y qué le gustaría que hiciese el autómata. (Papert, S. 1980: 28)

4.4 Diseño de las clases

En esta sección se presenta la implementación, en formato de clase, de la propuesta didáctica presentada en la Sección 4.1, buscando sea de aporte para los docentes en la enseñanza del concepto de autómata, utilizando un elemento cercano y motivante para los estudiantes, como lo es un videojuego.

Para las clases se utilizaron algunas slides con el fin de introducir algunos conceptos generales y disparar la actividad de taller con los estudiantes, las mismas pueden descargarse de: https://gitlab.fing.edu.uy/aaguirre/lumber-jack

Para esta actividad no se tuvo en cuenta la barra de tiempo del videojuego, este puede ser un elemento interesante para trabajar con estudiantes más avanzados el concepto de eficiencia de un programa.

4.4.1 Primera clase

En la primer primer clase se le propone a los estudiantes jugar al juego durante aproximadamente veinte minutos, para que puedan descubrir las reglas del mismo desde la acción. Inmediatamente después se propusieron una serie de preguntas para lograr la toma de conciencia por parte de los estudiantes de las reglas que aplican en el plano de la acción.

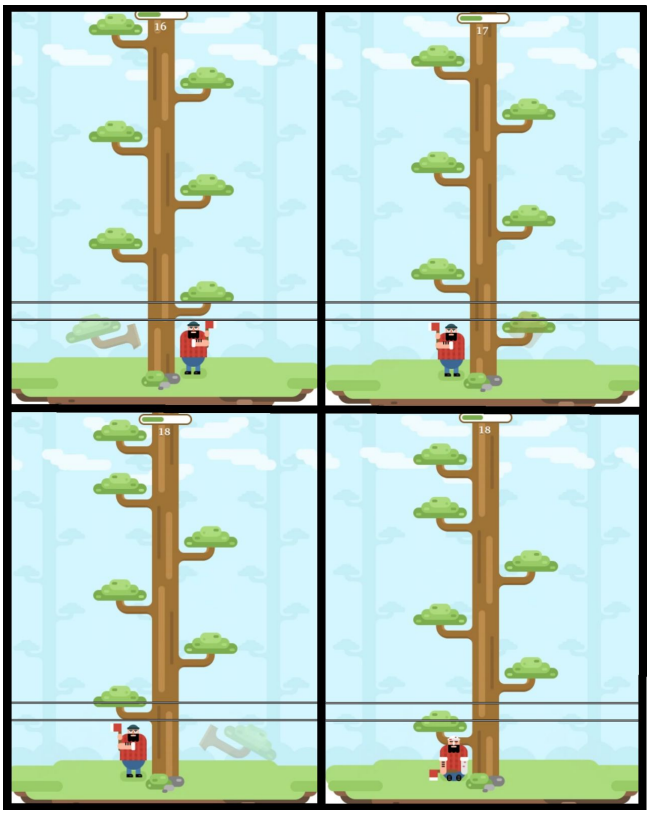


Figura 4.4.1.A: Secuencia de jugadas en el videojuego con un área de la pantalla resaltada para hacer reflexionar a los estudiantes sobre los cambios en el modelo del mundo del videojuego.

La consigna para los estudiantes decía: "si imaginamos que hay filas o renglones en la pantalla, decimos que Jack está siempre en la fila 1 y que las ramas están en filas más altas y van bajando de fila a medida que Jack corta el árbol. La fila crítica es la fila encima de la que está Jack (fila 2, ver área resaltada en la Figura 4.4.1.A), ya que si hay rama y Jack está del mismo lado que la rama, será abatido; mientras que si está del otro lado, cortará la rama y el juego seguirá. Apretando un botón tú decides de qué lado estará Jack cuando la rama llegue a la fila encima de Jack (fila 2).

1. ¿Cuáles son las posibles posiciones en que puede estar Jack, con respecto al árbol?
2. ¿Cómo decides cuál botón apretar?
3. ¿Puedes completar las celdas que tienen el símbolo '?'de la tabla?
4. ¿Puedes resumir abajo cuando se da el éxito y cuándo el fracaso, con tus propias palabras?"

La tabla de la tercera pregunta es la que se muestra en la Tabla 4.4.1.A. Cada fila indica el estado de Jack (Derecha/Izquierda) y el árbol en la situación crítica (fila anterior con rama sobre su cabeza del mismo lado

-Si- o no -No-). En base a la información en esas dos columnas, el jugador decidió qué botón presionar, que es lo indicado en las filas de la columna "Botón" (el de la Derecha o el de la Izquierda).

Jack	**Rama**	**Botón**
Izquierda	No	Izquierda
Izquierda	Si	Derecha
Derecha	No	Izquierda
Izquierda	No	?
Izquierda	Si	Derecha
Derecha	Si	Izquierda
Izquierda	No	Izquierda
?	Si	Derecha
Derecha	No	Derecha
Derecha	Si	?
Abatido	X	X
Derecha	No	Izquierda
Izquierda	Si	Derecha
Derecha	No	Izquierda
Izquierda	Si	Izquierda
Abatido	X	X
Izquierda	No	Derecha
Derecha	Si	Derecha
?	X	X

Tabla 4.4.1.A: Tabla correspondiente a la pregunta 3.

4.4.2 Segunda clase

En la segunda clase se presentan los conceptos de autómata y de robot, y se trabaja con los estudiantes tratando de construir en conjunto una definición de los mismos para proponer una categorización de diferentes artefactos de nuestra vida cotidiana (electrodomésticos, ascensor, semáforo, entre otros) como autómatas o como robots. Se presenta el modelo de agente que vimos en la Figura 3.1.A y se trabaja con el ejemplo de autómata que asiste a una persona ciega a cruzar la calle que fue presentado en la Sección 3.

El objetivo de esta clase es que los estudiantes identifiquen los componentes más importantes de un autómata y puedan trabajar sobre un ejemplo concreto de agente, especificando las reglas como acciones que se ejecutan según el estado del mundo del videojuego. Seguidamente se presenta el concepto de máquinas de estado como una herramienta para diseñar los autómatas y se especifican las reglas del autómata como una máquina de estados.

Finalmente se propone como actividad para realizar de tarea domiciliaria el diseñar la máquina de estados del autómata que juega al lumberjack, como la de la Figura 4.4.2.A.

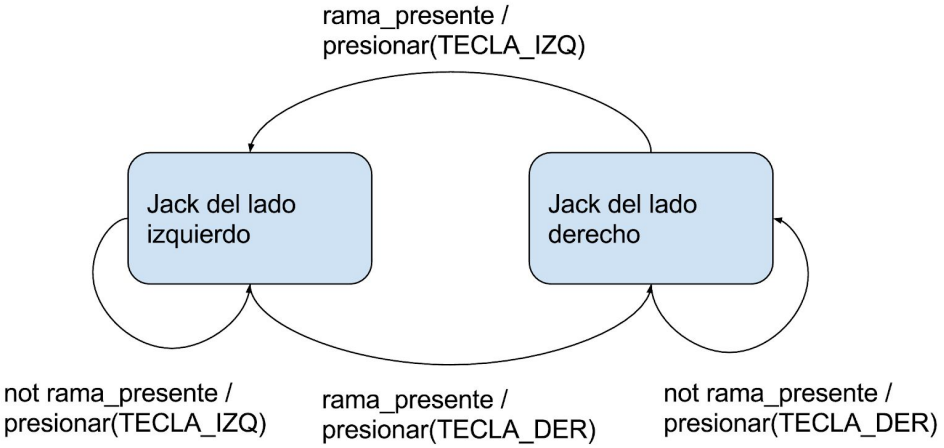


Figura 4.4.2.A: Autómata que modela el comportamiento de Jack.

4.4.3 Tercera clase

En la tercera clase tiene por objetivo que el estudiante logre generar conocimiento acerca de cómo el control del agente, definido en una máquina de estados, se codifica como un programa, lo cual implica el uso de una computadora que instancia nuestro autómata y sitúa al estudiante ante una nueva periferia.

Para atacar el objetivo, primeramente se entrega a cada estudiante la devolución del autómata que crearon, en la experiencia realizada en el liceo de La Paz un dato interesante es que la inmensa mayoría de las entregas estuvieron totalmente correctas; el resto tenía pequeñas correcciones para hacer (da Rosa, S. Aguirre, A. 2018) .

La primera actividad consiste en repasar en conjunto, el autómata del algoritmo para jugar a LumberJack, de modo de refrescar lo hecho, e identificar los errores comunes que surgen. A esta altura de la actividad, los estudiantes deberían estar "en el primer nivel" de toma de conciencia mencionado previamente, es decir, basándonos en la Ley General de la Cognición y los resultados obtenidos en las entregas, podemos ver que los estudiantes comienzan a comprender la relación causal que existe entre las acciones que realizan sobre los objetos(C) y los cambios que generan gracias a dichas acciones(C'), logrando conceptualizar las reglas del algoritmo que permite jugar al videojuego.

A esta altura de la actividad, lo que resta es que el estudiante pueda "enseñar a la computadora a jugar a LumberJack", es decir, construir el conocimiento de cómo lograr que la computadora ejecute el algoritmo del autómata construido.

Como paso intermedio para lograr esto, se propone una actividad más cercana a la computadora, en la cual se presenta un nuevo problema. Dado el personaje de la tortuga del programa TurtleBots, y un sensor botón, se debe lograr que, al presionar el botón, la tortuga avance si está quieta y se detenga si está avanzando. Podemos observar que esto corresponde a un problema que se puede resolver con máquinas de estado, ya que interesa saber en qué estado está la tortuga a la hora de presionar el botón.

En la actividad realizada en el liceo de La Paz, se formaron grupos de estudiantes, y a cada grupo se le entregó un programa que implementa el algoritmo de la máquina de estados, el cual estaba incompleto y ellos debían completarlo correctamente. Dado que en TurtleBots se crean programas donde las instrucciones son bloques que se encastran, un programa incompleto puede entenderse como un programa con bloques desencastrados. Dichos bloques estaban flotando desordenados y el trabajo de los grupos fue encastrar cada uno correctamente, basándose en el autómata que previamente habían construído para enfrentar el problema de la tortuga.

Para ilustrar la lógica de la planificación de esta clase, podríamos decir que el grupo de estudiantes se encontraba en una nueva periferia en la que sabían hacer autómatas para resolver un problema. Ahora debían desplazar la toma de conciencia a un nuevo C y C' para implementar el autómata en una computadora, y que ésta última lo pueda ejecutar. Este nuevo C es tomar conciencia de las acciones que se pueden ejecutar en la computadora, y C' tomar conciencia de qué sucede en ella cuando se ejecutan dichas acciones. Por eso la actividad de la tortuga, para practicar el pasaje del autómata diseñado en papel a la ejecución en computadora. Luego de que la mayoría resolvió correctamente el problema, y otros lo hicieron también con un poco de ayuda, probaron con entusiasmo lo que habían programado, y jugaron una y otra vez con la tortuga que obedecía a las órdenes del botón.

Análogamente al programa de la tortuga, se les otorgó luego un programa incompleto que implementa el autómata que habían construído en la clase anterior. Es decir, el autómata del algoritmo para jugar a LumberJack. Dados los tiempos y recursos, no se esperaba que programen desde cero los autómatas. Las cuestiones principales eran tomar conciencia de que resolver un problema de ciencias de la computación, tiene además de la construcción de una solución, el trabajo no menor de "enseñar dicha solución a la computadora". En todas las actividades los alumnos se sintieron motivados, y se considera que el enfoque utilizado, es decir el enfoque de comenzar desde algo que ellos saben hacer y les gusta, fue muy importante para lograr dicha motivación.

5 Programación

"La programación presenta una oportunidad, para participar en dar respuesta a problemas que presentan más de una forma correcta de ser resueltos, generando poder en un ambiente seguro: no hay nada que romper aparte de su programa, y cuando se rompe se genera la oportunidad para encontrar y solucionar el problema. La naturaleza misericordiosa de la programación ofrece un lugar seguro para tomar riesgos intelectuales: no hay una sola manera correcta de hacer las cosas, por lo cual proporciona autonomía al usuario." (Trinidad, G. et al. 2015)

Esta sección busca profundizar en el aprendizaje de la programación más allá de las tecnologías involucradas en los ejemplos que presentamos, como ser el robot a utilizar o el lenguaje de programación para programarlo. Los conceptos principales a trabajar en esta sección son: la noción de algoritmo y programa, variables y estructuras de control, los cuales son los pilares de la programación imperativa.

5.1 TurtleBots

La programación en computación es la acción de crear un conjunto de órdenes para una computadora en un lenguaje específico, en pos de resolver un problema determinado. El presente documento trabaja con el paradigma de programación imperativa, cabe aclarar que existen otros, pero escapan a los objetivos de este material. La programación imperativa tiene tres componentes fundamentales: la secuencia, la selección y la repetición que en siguientes subsecciones serán explicadas; además se enumerarán conceptos básicos de la programación, como lo son las variables y las estructuras de control.

Para introducir conceptos de programación se utilizará TurtleBots [8] , un software de fácil aprendizaje, lo que comúnmente se conoce como "piso bajo", que permite construir programas encastrando bloques que representan instrucciones para el computador, TurtleBots es una distribución de TurtleBlocks orientada a robótica educativa; TurtleBlocks está inspirado en el lenguaje Logo[9]. TurtleBots además de libre, es gratuito, sencillo de utilizar y compatible con muchos de los kits robóticos presentes en Uruguay, por ejemplo Butiá o Lego NXT.

[8] Programa Turtle Bots, https://www.fing.edu.uy/inco/proyectos/butia/files/package/, visitada en enero del 2019.

[9] Logo Programming Language; Logo Foundation http://el.media.mit.edu/logo-foundation/what_is_logo/logo_programming.html, visitada en junio del 2019.

En TurtleBots/TurtleBlocks programar es encastrar bloques (es decir encadenar instrucciones) de forma correcta y ordenada para que la computadora comprenda cómo queremos resolver un problema y ejecute las acciones que le indiquemos.

"El "piso bajo" de Turtle Blocks proporciona un punto de entrada fácil para los principiantes. También cuenta con funciones de programación de "techo alto" que desafían a los estudiantes más aventureros. Al igual que en la mayoría de los ambientes Logo, en Turtle Blocks, la tortuga existe en tres formas: (1) como un robot que comparten el mismo espacio físico como el niño; (2) como un objeto computacional que se mueve en la pantalla; y (3) como una entidad matemática abstracta." (Trinidad, G. et al. 2015)

A continuación comenzaremos explorando la forma 2, para luego ir avanzando en las otras, podemos programar una pequeña tortuga que se encuentra inicialmente en el centro de la pantalla y al moverse deja su camino marcado, cuyo color y grosor son configurables, lo que permite crear una infinidad de figuras y dibujos a través de la ejecución de un programa. Se puede apreciar la tortuga y algunas posibles figuras creadas por su movimiento en la siguiente figura:

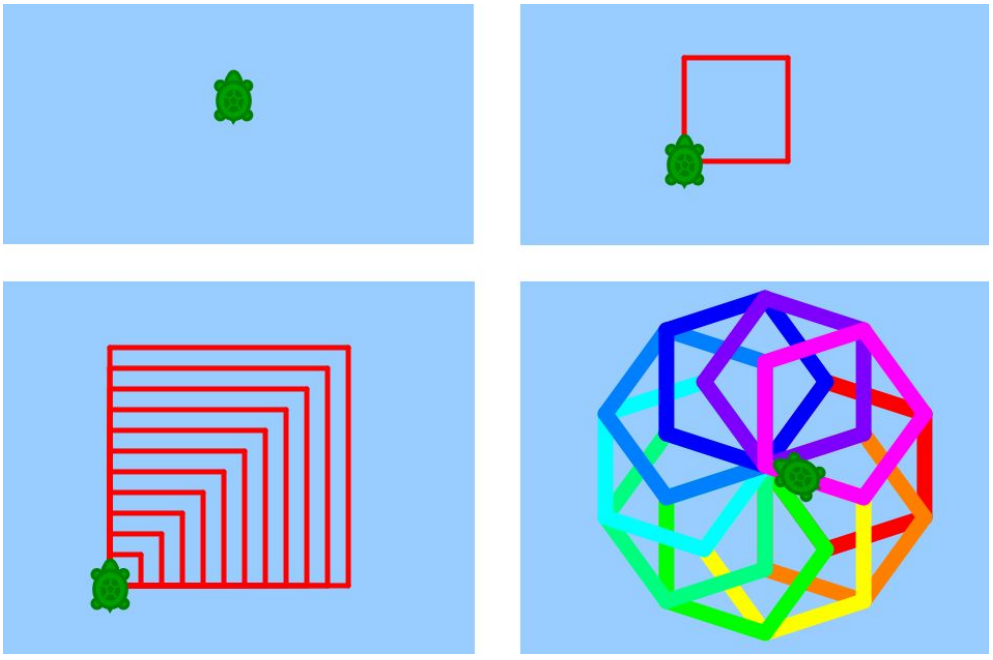


Figura 5.1.A: Ejemplos tortuga.

El usuario de TurtleBots puede crear programas para que la tortuga realice dibujos y figuras muy fácilmente con esta herramienta, y además de estos programas, puede crear otros con diferentes propósitos, por ejemplo programar kits robóticos.

En secciones posteriores se introducirán conceptos de programación usando como lenguaje de ejemplo TurtleBots. Para saber cómo instalarlo, ejecutar programas y comprender sobre las paletas y funcionalidades de TurtleBots, se puede acceder al **Manual de uso básico de TurtleBots y robot Butiá 2.0**[10]

5.2 Algoritmo y Programa Imperativo

A partir de la noción de programación descrita, ¿Cuál es la definición de programa? A grandes rasgos podemos definir un programa como un conjunto ordenado de instrucciones que indican a una computadora cómo resolver un determinado problema. Esta definición corresponde a la de un programa construido bajo el paradigma imperativo, mencionado anteriormente.

La noción de algoritmo es similar a la de programa, ya que un algoritmo consiste también en un conjunto ordenado de instrucciones, pero el programa, a diferencia del algoritmo, es además comprendido y ejecutado por una máquina. Podemos ver al algoritmo como un concepto más general, que refiere a la solución elaborada por una persona para resolver determinado problema, por ejemplo: repartir n elementos entre m personas, la solución a este problema es el algoritmo de la división entera.

A partir de este momento nos concentramos en los programas, ya que las máquinas que ejecutarán nuestros algoritmos serán, en su mayoría, autómatas.

Un programa entonces es fácilmente comparable con una receta de cocina, donde se le indica a una persona qué debe hacer y en qué orden en pos de resolver el problema en cuestión, que en este caso es realizar la comida correspondiente. Una diferencia importante entre una receta y un programa es que este último se ejecuta dentro de una computadora y se encuentra sujeto a las restricciones tecnológicas de la misma, como ser la memoria y la capacidad de procesamiento. La programación tiene también un componente físico que refiere al hecho de que mientras se ejecuta un programa, éste efectúa cambios en la memoria de la computadora, por lo que terminan siendo cambios en su hardware, por ejemplo en la información que se encuentra almacenada en sus circuitos.

A continuación se muestra un programa muy sencillo en TurtleBots para el cual se explicará qué sucede en cada instrucción:

[10] Manual de uso básico de TurtleBots y Butiá 2.0, https://www.centrosmec.gub.uy/innovaportal/file/823/1/manual-turtlebots---butia-2.0_05-11-18-de-a-una-pag.pdf, visitada en mayo del 2019.

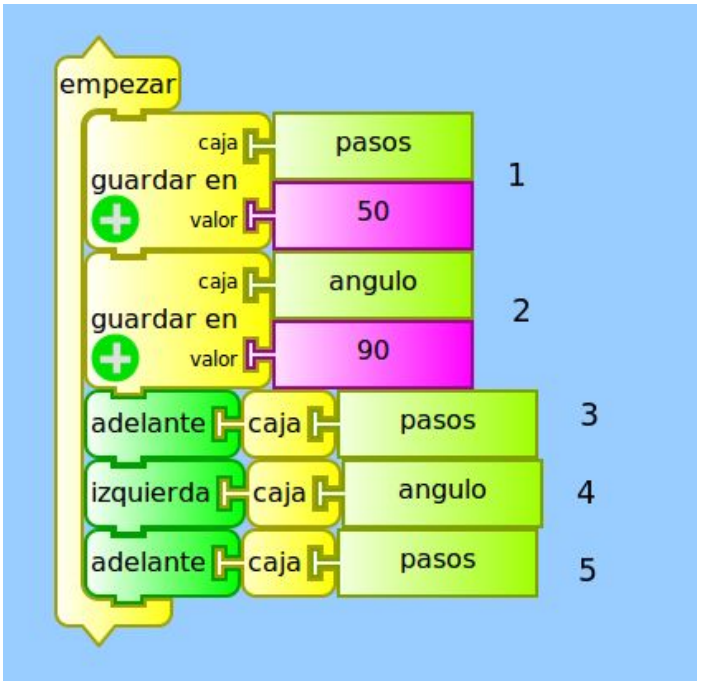


Figura 5.2.A: Ejemplo de programa 1.

Como ya fue mencionado, tenemos un conjunto ordenado de instrucciones, es decir un programa imperativo. Los números a la derecha indican a qué instrucción refiere cada bloque:

Instrucción 1

La primer instrucción corresponde a la creación de una variable que se llama **pasos** y en ella se almacena el valor **50**. Las variables se explican mejor en próximas secciones. Por lo pronto lo importante es entender que al ejecutar esta instrucción, la computadora sabe que debe asociar el valor 50 a la variable pasos, y cada vez que se consulta qué valor está asociado a dicha variable, debe saber que es 50. En resumidas cuentas, la computadora guarda en su memoria la palabra **pasos** y la asocia al valor **50**.

Instrucción 2

La segunda instrucción es análoga a la primera, guardando en la variable **ángulo**, el valor **90**.

Instrucción 3

Se da la orden a la tortuga de ir hacia **adelante**. Dicha instrucción necesita conocer cuántos pasos debe avanzar la tortuga, para ello se le encastra el bloque correspondiente a la variable **pasos**, representada mediante el concepto de caja, es decir que debe dar **50** pasos hacia adelante, dado que previamente se le almacenó el valor 50 a la variable mencionada.

Instrucción 4

La instrucción **izquierda**, le indica a la tortuga que debe girar hacia la izquierda (sentido antihorario). Dicha instrucción necesita conocer el ángulo de giro, es decir cuántos grados debe girar la tortuga, por lo que se le encastra la variable **ángulo**, previamente asociada al valor **90**. En conclusión: gira hacia la izquierda 90 grados.

Instrucción 5

Es igual a la instrucción 3, es decir que la tortuga avanza 50 pasos.

¿Cuál es el resultado de ejecutar el programa? El programa guarda valores en variables, y le ordena movimientos a la tortuga. Ejecutar el mismo genera el siguiente efecto:

La tortuga avanza 50 pasos.
La tortuga gira 90 grados.
La tortuga avanza nuevamente 50 pasos.

Como se mencionó en la introducción de TurtleBots, la tortuga deja su rastro al moverse, por lo que la ejecución del programa de ejemplo genera el siguiente dibujo:

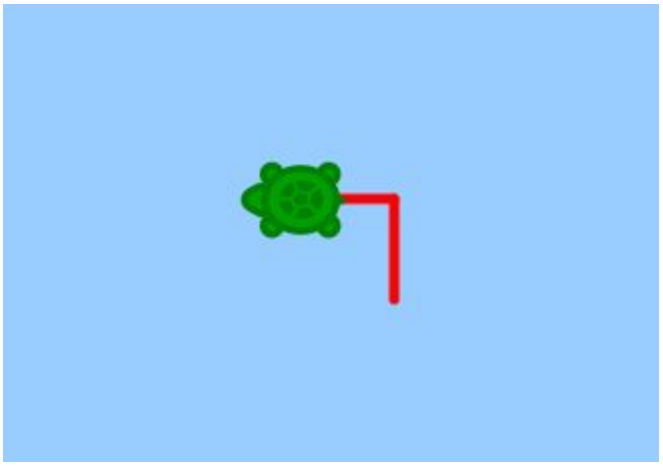


Figura 5.2.B: Ejemplo de programa 1 ejecutado.

En definitiva, se puede definir la ejecución de un programa, como la computadora realizando las órdenes que componen al mismo y en el orden indicado. Las computadoras ejecutan las instrucciones de forma muy rápida, por lo que al ejecutar el programa de ejemplo, no seremos capaces de ver a la tortuga moviéndose, sino que veríamos de forma inmediata el resultado que se muestra en la foto previa.

Lo que acabamos de ver corresponde a la **secuencia**; una de las tres componentes fundamentales mencionadas al principio, y refiere a la naturaleza de la ejecución secuencial y ordenada de instrucciones; es decir que se ejecuta una instrucción y luego de finalizada se ejecuta la siguiente, y así sucesivamente hasta completar la secuencia.

5.3 Variables

Cuando resolvemos (o intentamos resolver) un determinado problema, es decir cuando ejecutamos un conjunto de acciones, por lo general recordamos datos que usamos a lo largo de dicha resolución. Volviendo al ejemplo de cocinar, si sabemos que hay que usar 200 gramos de manteca en una receta, nos acordamos de ese dato y lo usamos cuando sea necesario a lo largo de la ejecución de acciones.

Lo mismo pasa al programar: cuando la computadora ejecuta un programa, guarda datos en su memoria que le son necesarios para dicha ejecución. Si miramos el ejemplo de programa de la sección anterior (Figura X), podemos apreciar que se tienen los datos de cuántos pasos debe avanzar la tortuga, y cuántos grados debe girar. Dichos datos son útiles para la ejecución de ese programa, y la computadora los guarda en cierta región accesible de su memoria, para utilizarlos cada vez que se precisen. ¿Dónde guarda dichos datos la computadora? Para lo que nos concierne, podemos afirmar que los guarda en variables.

La variable en un programa es un espacio en la memoria de la computadora, el cual tiene un nombre, y se le almacena un valor determinado. Los valores almacenados en las variables, pueden ser accedidos y actualizados a lo largo de una ejecución: accedidos para que la computadora utilice dicho valor cada vez que sea necesario, y actualizados por si la ejecución necesita que se almacene un nuevo valor y se olvide del anterior.

Dado que las variables tienen nombre, podemos representar la memoria de la computadora como una tabla donde en cada fila se tiene el nombre de las variables definidas, y su correspondiente valor. Manteniéndonos en el programa de ejemplo, podemos decir que definimos dos variables cuyos nombres son: **pasos** y **grados**, por lo tanto la tabla para ese programa luego de la ejecución de la instrucción 2 sería:

Nombre de variable	**Valor**
pasos	50
grados	90

A lo largo de la ejecución de un programa, una vez que se le asigne un valor a una variable, ésta última lo mantendrá al menos que sea actualizado; por eso es que en las instrucciones 3 y 5 del programa de ejemplo, la instrucción **adelante** se ejecuta las dos veces con el valor 50, valor que almacenamos previamente (en la variable **pasos**) y la computadora mantiene recordado durante la ejecución.

Cabe preguntarse: ¿Qué tipos de valores puede almacenar una variable en computación? Eso va a depender del lenguaje de programación que estemos utilizando, pero los tipos básicos son:

- Números reales.
- Números enteros.
- Caracteres (letras y símbolos).
- Strings (cadenas de caracteres).
- Booleanos (Verdadero o Falso).

A veces los lenguajes nos piden que indiquemos explícitamente qué tipo de valores van a almacenar las variables que creamos, y nos permiten crear nuevos tipos de datos, por ejemplos compuestos a partir de los tipos básicos. En TurtleBots no es necesario indicar el tipo de valor que vamos a almacenar, alcanza con simplemente asignarle ese valor a la variable.

En definitiva una variable en programación: tiene un nombre que la identifica dentro del programa, un valor que se le asigna y que puede ser accedido y modificado dentro del programa, y tiene una región de la memoria de la computadora donde es guardada con su correspondiente valor. En la próxima subsección trabajaremos con variables en TurtleBots para comprender mejor estos puntos nombrados.

5.3.1 Variables en TurtleBots

Para crear una variable en TurtleBots, nos dirigimos a la paleta de bloques de variables, y seleccionamos el bloque:

Figura 5.3.1.A: Bloque de variable.

Esto genera en la pantalla de TurtleBots el siguiente bloque:

Figura 5.3.1.B: Bloque de variable seleccionado.

En donde dice caja se le conecta el bloque con el nombre de la variable que se va a crear, y donde dice valor, se le conecta justamente el valor que queremos almacenar en ella. ¿Por qué TurtleBots le llama caja a las variables? Esto se debe a que conceptualmente podemos ver a una variable como a una caja: en ella guardo algo y puedo abrirla (acceder a la variable) para ver lo que guardé (ver el valor que tiene

almacenado). Además puedo sacar lo que guardé dentro de la caja, y ponerle algo nuevo (que corresponde a actualizar la variable).

¿Qué nombre se le pone a una variable? En principio se le puede llamar como se desee. Una buena práctica es llamarla con un nombre que se asocie a su utilidad: por eso en el primer ejemplo de programa en TurtleBots, a la variable que se le guarda el valor de los pasos que da la tortuga, se le puso el nombre **pasos**, y a la que guarda los grados de giro se la llama **grados**. Si bien se las puede llamar como se desee, es aconsejable utilizar la regla de darle nombres apropiados para lo que va a ser usada, ya que ayuda a comprender mejor un programa al leerlo. Los nombres de dos variables diferentes dentro de un programa no pueden repetirse porque sino la computadora no distinguiría una variable de la otra.

5.3.2 Las variables y la memoria

En matemática, las variables tienen un valor de un tipo, por ejemplo al definir $x = 6$, el nombre x queda asociado a ese valor y es equivalente (en ese contexto) decir $8 > 6$ y $8 > x$. En computación, las variables tienen algo más: un nombre, un tipo, y un lugar en la memoria. En nuestros programas podemos entonces definir distintas variables, de distintos tipos, y con distintos nombres, y luego podemos manipularlas, guardando, consultando y/o actualizando el valor que tienen.

Veamos el siguiente programa que contiene 4 instrucciones (al igual que en el primer ejemplo de programa, los números a la derecha son únicamente con fines explicativos):

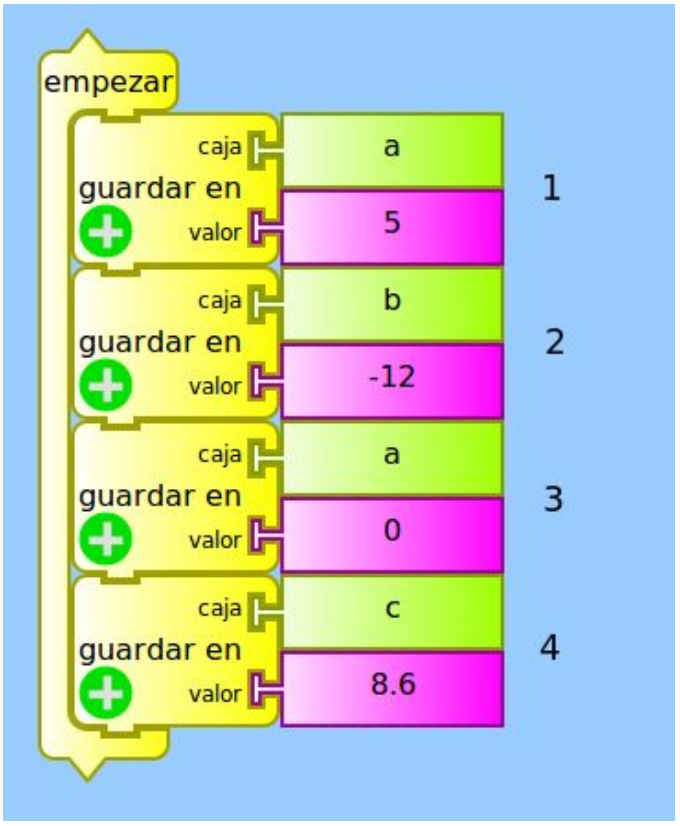


Figura 5.3.2.A: Ejemplo de programa con variables.

Vamos a explicar instrucción por instrucción, mostrando el estado de la memoria con respecto a las variables. Para ello utilizaremos la tabla conceptual que pusimos como ejemplo en la sección anterior (la tabla con los nombres de las variables y sus respectivos valores). Veamos cada instrucción:

Instrucción 1

Al ejecutarse esta instrucción, se crea una variable de nombre **a**, y se le guarda el valor **5**. Por lo tanto en la memoria de la computadora, tenemos la tabla así (por el momento):

Nombre de variable	**Valor**
a	5

Instrucción 2

Ahora se crea otra variable de nombre **b** y se le asigna el valor **-12**. Ahora a partir de esta instrucción la tabla queda:

Nombre de variable	Valor
a	5
b	-12

Instrucción 3

En esta instrucción se **actualiza** el valor de **a**, ya que se le asigna el valor **0** cuando antes tenía **5**. Por lo tanto ahora la tabla que representa a la memoria del programa ejecutando es la siguiente:

Nombre de variable	Valor
a	0
b	-12

¿Qué pasó con el valor **5**? Dado que una variable puede almacenar un valor a la vez, el valor asociado anteriormente fue olvidado por la computadora y se le asignó el nuevo. Luego de esta instrucción queda asociado el **0** a la variable en cuestión y la computadora mantendrá esa asociación hasta que termine la ejecución del programa.

Instrucción 4

Finalmente, se crea la variable de nombre **c** y se le asigna el valor **8,6**. Por lo que la memoria en este momento tiene el siguiente estado:

Nombre de variable	Valor
a	0
b	-12
c	8,6

Podemos decir que **a** y **b** almacenan valores de tipo entero, y **c** almacena valores de tipo real.

Acabamos de ver cómo asignarle valores a las variables, y cómo actualizarlos (en definitiva actualizar es guardar otro nuevo valor). Ahora veremos cómo acceder a los valores que tienen las mismas: si trabajamos con variables, tiene sentido querer acceder a sus valores, es decir, ver y utilizar lo último que se le guardó. Una vez que creamos una y le asignamos un valor, en TurtleBots tenemos un bloque (también en

la paleta de bloques de variables) que nos permite acceder a la misma y utilizar el valor que almacena, todas las veces que sea necesario. Dicho bloque en la paleta se ve de la siguiente forma (veamos el bloque de acceso de la variable **a**):

Figura 5.3.2.B: Acceder a variable 1.

Que si lo seleccionamos, lo vemos en la pantalla de TurtleBots de la siguiente forma:

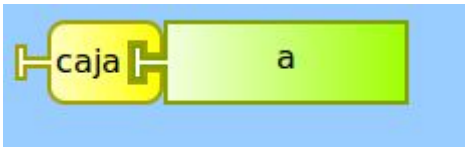


Figura 5.3.2.C: Acceder a variable 2.

Cada vez que se ejecute este último bloque, se devolverá el valor que **a** tenga almacenado en ese momento. Veamos el siguiente programa:

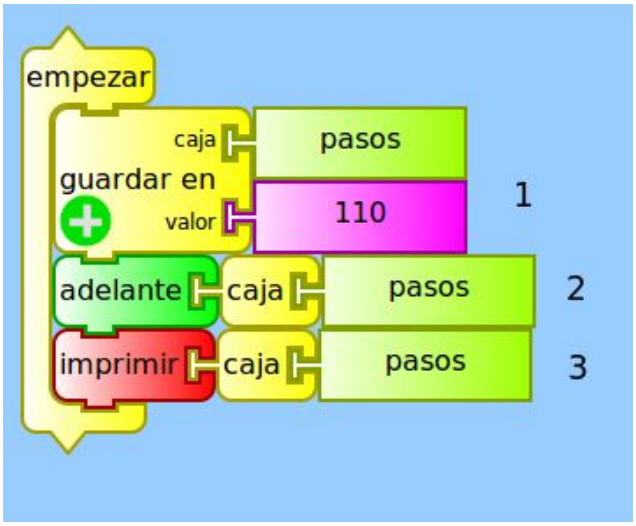


Figura 5.3.2.D: Ejemplo de programa de acceso a variables.

La instrucción 1 ya la conocemos, se está guardando en la variable **pasos** el valor **110**. Ahora veamos las otras dos instrucciones:

Instrucción 2

Como ya sabemos, la función **adelante** hace que la tortuga avance la cantidad de pasos indicada. Ahora bien, ya que se le conecta el bloque de acceso de la variable **pasos**, el valor que recibe la función

es **110**, que es el último valor que se le asignó a **pasos**. ¿Esto significa que al acceder al valor de **pasos**, dicho valor se remueve de la variable? No, porque la operación de acceso obtiene el valor que la variable almacena, pero no lo borra ni lo cambia. Veamos lo que sucede en la siguiente instrucción.

Instrucción 3

En esta instrucción también se accede a la variable, pero con otra función: la de **imprimir**. Esta función muestra en pantalla aquel valor que se le pase. Dado que la variable sigue almacenando el valor **110**, éste número es el que se imprimirá en pantalla. El resultado final luego de la ejecución de este programa de ejemplo, es que la tortuga avanza 110 pasos, y se imprime en pantalla el número 110.

Luego de la explicación de las instrucciones, podemos afirmar que dentro de la ejecución de un programa, todas las veces que sea necesario, se puede acceder a los valores que las variables almacenan.

5.3.3 Asignación de valores a las variables

Como ya sabemos, a una variable se le asigna un valor (ya sea al crearla, o al actualizarla). Veamos ejemplos de asignación de valores a las variables, además de los que ya se vieron en ejemplos anteriores.

Ejemplo 1

Supongamos que tenemos dos variables llamadas **X** e **Y** que guardan valores numéricos, y queremos calcular el promedio de las mismas y guardarlo en otra, que llamaremos justamente **promedio**. El programa que realiza la tarea es el siguiente:

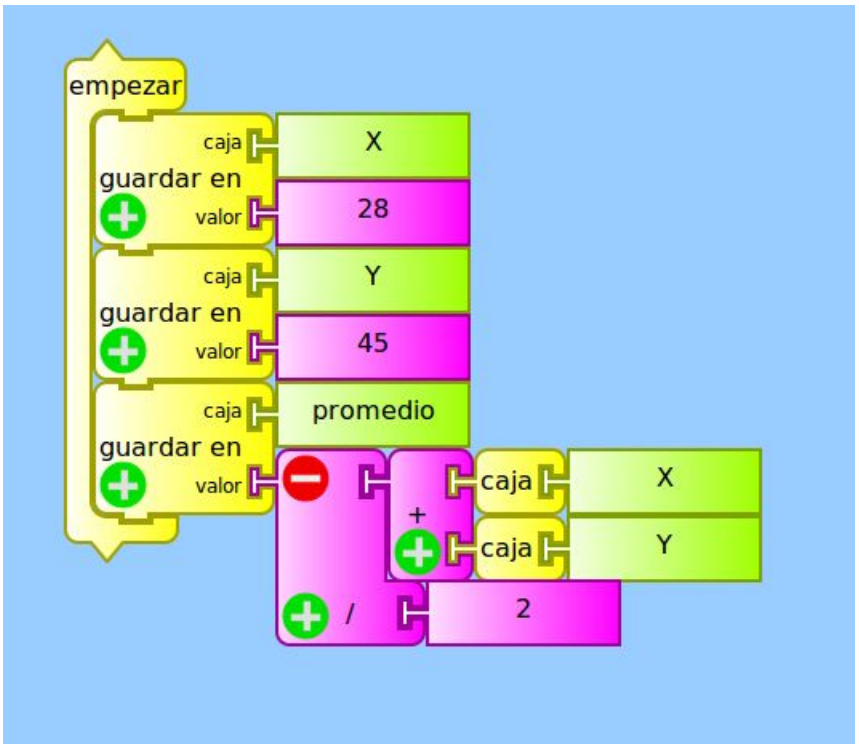


Figura 5.3.3.A: Ejemplo de programa con promedio.

Las dos primeras instrucciones son conocidas, se está guardando en la variable **X** e **Y**, los valores **28** y **45** respectivamente. ¿Y en **promedio**? En dicha variable se almacena el resultado de una operación aritmética, la cual se debería leer de derecha a izquierda. Primero se tiene una suma, que toma los valores almacenados en **X** e **Y**, es decir se calcula **28 + 45**. Dicho resultado (**73**), entra en el bloque de división y se divide entre **2**. En definitiva se le está asignando a **promedio** el resultado de la siguiente operación:

(28 + 45) / 2

Es decir se almacena en **promedio** el valor **36,5** que es justamente el promedio entre **X** e **Y**.

¿Qué se puede concluir de este ejemplo? Que podemos acceder a valores de variables para realizar operaciones, y que podemos asignarle a una variable un valor que se calcula en función de los valores que almacenan otras variables (en este caso calculamos el valor que se almacena en **promedio**, en función de **X** e **Y**).

¿Y podemos almacenar en una variable un valor en función de lo que ella misma almacena? En el siguiente ejemplo se responde la pregunta.

Ejemplo 2

Supongamos que tenemos una variable que se llama **aumentar**, y se le asigna un valor aleatorio entre **20** y **80**. Luego se desea aumentar el valor almacenado en **1** y guardar ese resultado en la misma variable **aumentar**. A continuación tenemos el programa de ejemplo que realiza estas acciones:

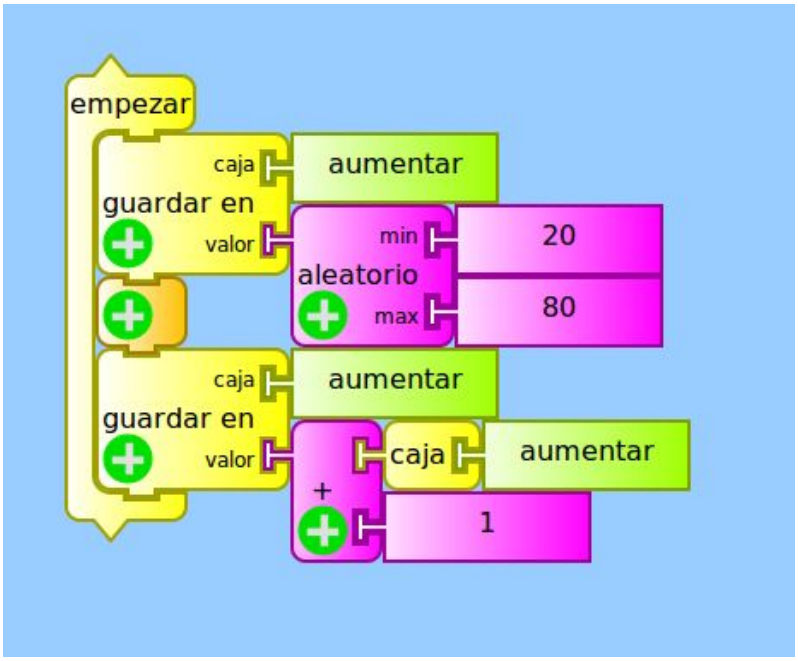


Figura 5.3.3.B: Ejemplo de programa con aumentar.

Nota: Los bloques de valor aleatorio y separador, se explican en el **Manual de uso básico de TurtleBots y robot Butiá 2.0**.

En la primera instrucción, el bloque de valor aleatorio devuelve un valor cualquiera entre **20** y **80** y lo guarda en **aumentar**. Luego de la ejecución de la primer instrucción ¿Con qué valor se actualiza la variable **aumentar**? Veamos con atención: en ella se guarda el resultado de una suma, que accede al valor almacenado en **aumentar** y le suma **1**. En definitiva se está guardando en **aumentar**, el valor que tiene, más **1**. Por más paradójico que suene, esta es una práctica muy habitual en programación, sobretodo cuando se quieren resolver problemas que requieren contar: se usa una variable en la que se le suma **1** a ella misma cada vez que aumente el contador. Veremos un ejemplo en secciones siguientes. Ahora mismo puede resultar que las variables no sean muy útiles en un programa, pero en realidad son fundamentales; en la siguiente sección se explican las estructuras de control, y se puede ver mejor su utilidad.

Nota: De aquí en más, en todo pseudocódigo presentado, la asignación de valores a las variables, se hará a través del operador ***:=***, de la forma ***variable := valor***.

5.4 Estructuras de control

En el presente documento se trata el paradigma de programación imperativo, el cual se centra en solucionar un problema especificando una secuencia de acciones. Recordando que en TurtleBots la tortuga se mueve dejando un rastro, mostraremos a continuación un programa que hace que la tortuga se mueva describiendo la figura de un cuadrado:

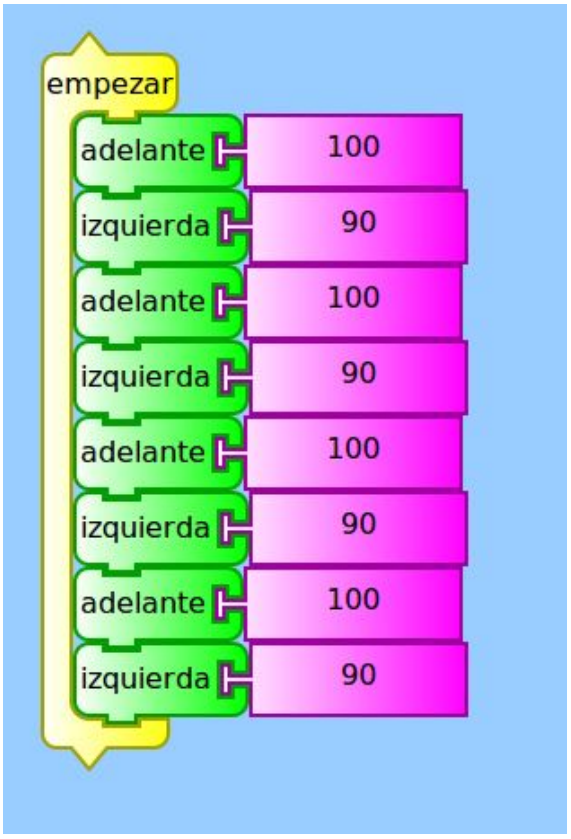


Figura 5.4.A: Ejemplo de programa de cuadrado.

En resumidas cuentas, se le indica a la tortuga que avance y gire cuatro veces. Estas instrucciones dejan como resultado el siguiente cuadrado y a la tortuga de vuelta en su posición inicial:

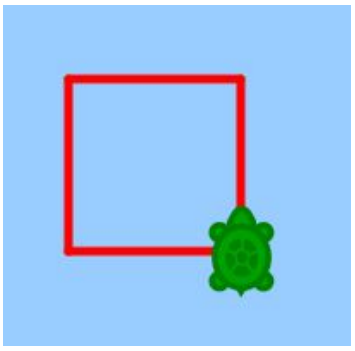


Figura 5.4.B: Tortuga describiendo un cuadrado.

Si observamos el programa, podemos darnos cuenta que repite muchas instrucciones. Cada instrucción de avanzar y girar está repetida tres veces más. ¿Hay alguna forma de indicarle a la computadora que hay instrucciones que queremos que se repitan, en vez de ingresarlas nosotros mismos una y otra vez?

Sí que la hay, para este caso tenemos el bloque **repetir** que nos permite justamente ejecutar de forma repetida instrucciones en vez de escribirlas una por una. Dado que para este ejemplo, se repite cuatro veces la acción de avanzar y girar, podemos escribir un programa que hace lo mismo pero con el bloque mencionado:

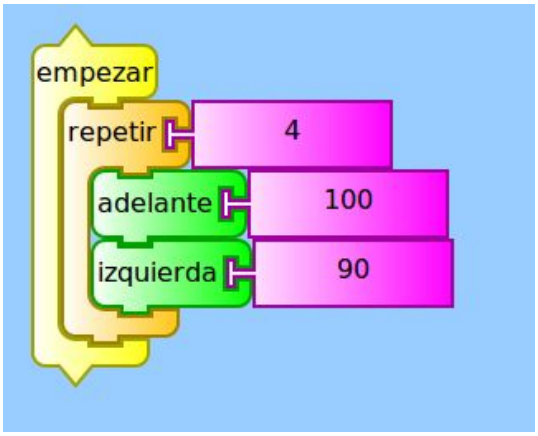


Figura 5.4.C: Ejemplo de programa de cuadrado con repetir.

Se puede afirmar que la segunda forma de resolver el mismo problema, es más prolija y más fácil de entender para las personas, además de que no se repiten instrucciones de forma innecesaria.

El bloque repetir recibe el número de veces que se desea que se ejecuten las instrucciones que se encuentran en él (en este caso 4 veces). Por lo que la tortuga hará lo siguiente: va a avanzar 100 pasos y girar 90 grados a la izquierda, 4 veces.

Lo que acabamos de ver es el uso de una **estructura de control**, ¿Qué son estas estructuras? Son herramientas de programación que nos permiten controlar la ejecución de nuestros

programas, por ejemplo realizar acciones repetitivas, tomar decisiones según cierto contexto, seleccionar entre diferentes caminos a tomar, ejecutar un número variado de instrucciones, etc. En próximas subsecciones se tratarán diferentes estructuras de control a través de TurtleBots.

5.4.1 Estructura "repetir"

Como acabamos de ver en la subsección previa, el bloque repetir es una estructura de control que nos permite repetir una instrucción o un conjunto de instrucciones una cantidad de veces determinada. A veces nos enfrentamos a problemas (como el problema de la tortuga describiendo el cuadrado) para los cuales sabemos cuántas veces se debe ejecutar la o las instrucciones necesarias para resolver el problema en cuestión, y en esos casos es importante saber usar el bloque **repetir**.

Si en vez de hacer un cuadrado se desea hacer un hexágono con la tortuga, ¿Cómo utilizamos el bloque repetir? Realizando el razonamiento análogo: para el cuadrado la tortuga gira y avanza 4 veces, o sea 1 vez por lado. Con el hexágono entonces la tortuga debe girar y avanzar 6 veces; y la cantidad de grados en vez de ser 90 es 60 (ya que 360 / 6 = 60). Por lo tanto tenemos un programa análogo:

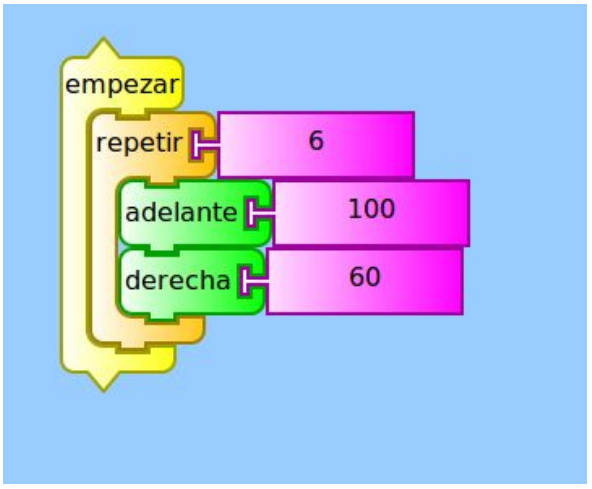


Figura 5.4.1.A: Ejemplo de hexágono con repetir.

Que al ejecutarlo da como resultado a la tortuga describiendo el hexágono:

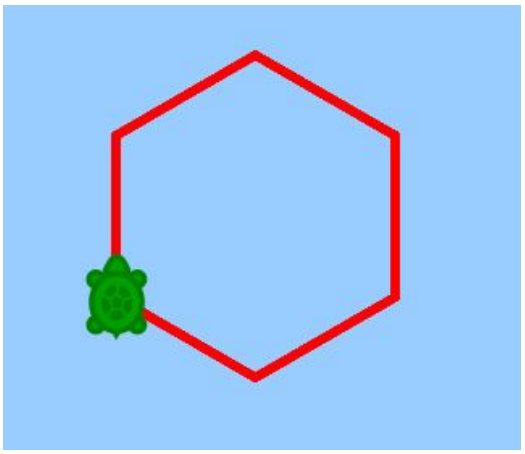


Figura 5.4.1.B: Tortuga describiendo hexágono.

Es importante notar lo siguiente: primero, el 100 en el programa corresponde a la cantidad de pasos que avanza la tortuga, es decir al largo de cada lado de la figura, y segundo, se puede generalizar esta resolución para que la tortuga haga figuras geométricas de más o menos lados (triángulos, pentágonos, heptágonos, octógonos, etc).

El bloque en cuestión tiene la siguiente forma:

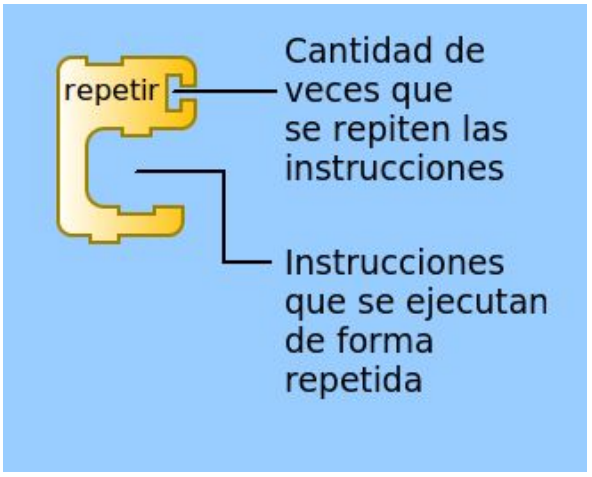


Figura 5.4.1.C: Bloque repetir.

En conclusión, el bloque **repetir** se utiliza cuando la resolución del problema que enfrentamos requiere de la repetición de una ó un conjunto de instrucciones, y que además sabemos cuántas veces repetirlas. Más adelante veremos estructuras de control que nos permitirán repetir instrucciones una cantidad de veces que no conocemos a priori ya que, por ejemplo, dependen de una entrada del usuario.

5.4.2 Estructura "si-entonces" y "si-entonces-sino"

Al resolver problemas (no sólo en programación) es necesario ejecutar acciones que dependen de las circunstancias en las que se está. Si una persona va a salir de su casa y no quiere mojarse, puede elegir llevar el paraguas o no y se puede expresar un pseudocódigo de la siguiente manera:

si afuera llueve o probablemente lloverá:
    llevar paraguas
sino:
    no llevar paraguas

La estructura de control **si-entonces-sino** (en inglés conocida como **if-then-else**) sirve para tomar decisiones de acciones a ejecutar, en función de los datos con los que trabaja el programa. Por ejemplo supongamos que tenemos un programa que toma un número aleatorio entre -50 y 50, y determina si es positivo o negativo. El pseudocódigo sería de la siguiente manera:

```
numero := obtenerAleatorio(-50,50)
si (numero es menor que 0) entonces:
      imprimir(es negativo)
sino:
      imprimir(es positivo)
```

La estructura **si-entonces-sino** permite entonces llevar a cabo una acción u otra dependiendo de lo que pase al momento de su ejecución. ¿Qué es necesario para utilizar esta estructura de control? Lo crucial es una **expresión booleana**, es decir una expresión de la cual siempre se pueda determinar si es verdadera o falsa. Si la condición es verdadera se ejecuta una instrucción (o serie de instrucciones) y si es falsa se ejecuta otra (u otra serie de instrucciones). Veamos el programa de ejemplo en código TurtleBots:

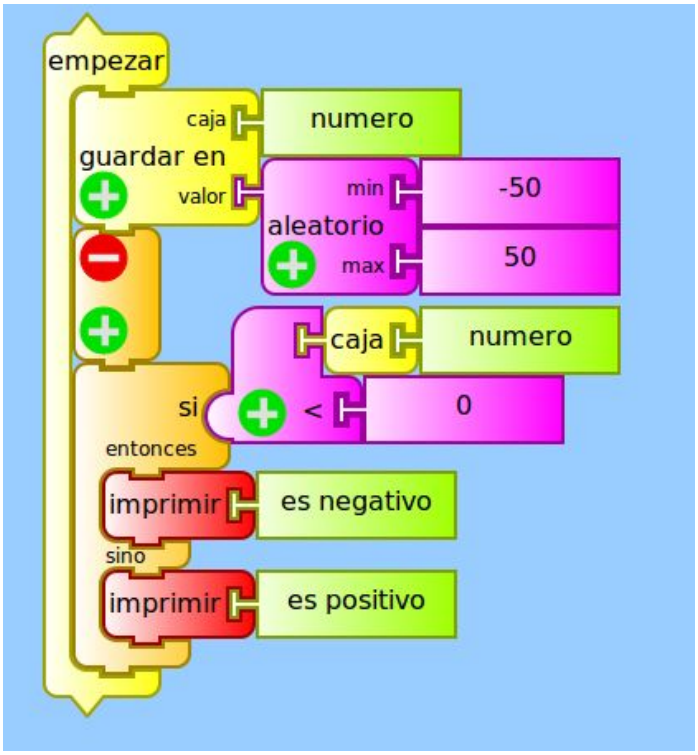


Figura 5.4.2.A: Ejemplo de programa con si-entonces-sino.

En la primera instrucción se guarda un número que va entre -50 y 50, en la variable **numero**. Luego en el programa se pregunta si dicho número guardado es menor que 0: si lo es, se imprime "**es negativo**", de lo contrario se imprime "**es positivo**". ¿Cuál es la expresión booleana del **si-entonces-sino**? O mejor dicho ¿Cuál es la pregunta dentro del programa que le permite tomar la decisión? En TurtleBots es el bloque violeta que tiene un semicírculo que se encastra en el hueco del **si-entonces-sino**:

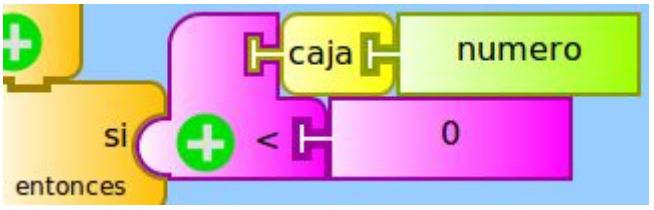


Figura 5.4.2.B: Ejemplo de expresión booleana.

Este bloque al ejecutarse en un programa, está consultando si (**numero < 0**) y devuelve True o False, dependendiendo del valor que se le haya asignado a la variable en cuestión. En caso de devolver True la estructura de control **si-entonces-sino** ejecuta las instrucciones indicadas dentro del primer hueco, y si devuelve False ejecuta las instrucciones del segundo hueco. A continuación se muestra el bloque en forma en TurtleBots:

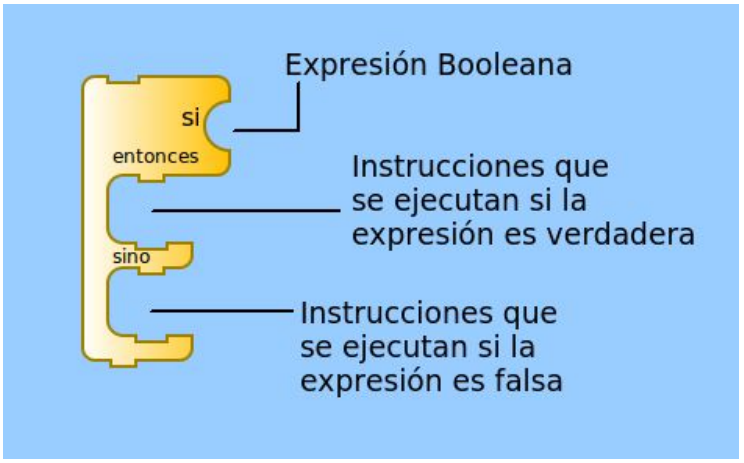


Figura 5.4.2.C: Bloque si-entonces-sino.

El funcionamiento de la estructura **si-entonces-sino** es el siguiente:

1. Se evalúa la expresión booleana.
2. Si el resultado de la expresión booleana es verdadero se ejecuta el conjunto de instrucciones que se encuentra primero, y se saltea el segundo bloque de instrucciones. Por otro lado, si el resultado es falso, se saltea el primer bloque de instrucciones, y se ejecuta el segundo.
3. Se continúan ejecutando las instrucciones que puedan existir siguientes a la estructura.

Para el siguiente ejemplo de programa, a veces es necesario tomar una decisión pero no entre dos caminos posibles, sino que interesa tomar un solo camino, o no tomarlo: se desea un programa que obtiene un número aleatorio entre -80 y 45 y muestra su valor absoluto, por lo tanto se tiene el siguiente pseudocódigo que implementa una posible solución:

```
numero := obtenerAleatorio(-80,45)
si (numero es menor que 0) entonces:
      multiplicar numero por -1
      guardar el resultado en numero
imprimir(numero)
```

Se tiene que el programa invierte el signo del número obtenido sólo en el caso de que sea un número negativo. Si es un número positivo no se realiza la multiplicación. Esto resuelve el problema dado que el valor absoluto de un número es el mismo número pero siempre positivo. Acabamos de ver un programa que usa **si-entonces** (en inglés **if-then**) en lugar de **si-entonces-sino** puesto que hay ocasiones en los que si una expresión es verdadera entonces se ejecutan acciones, sino se saltean y se continúa con el programa. Veamos dicho pseudocódigo construído en TurtleBots:

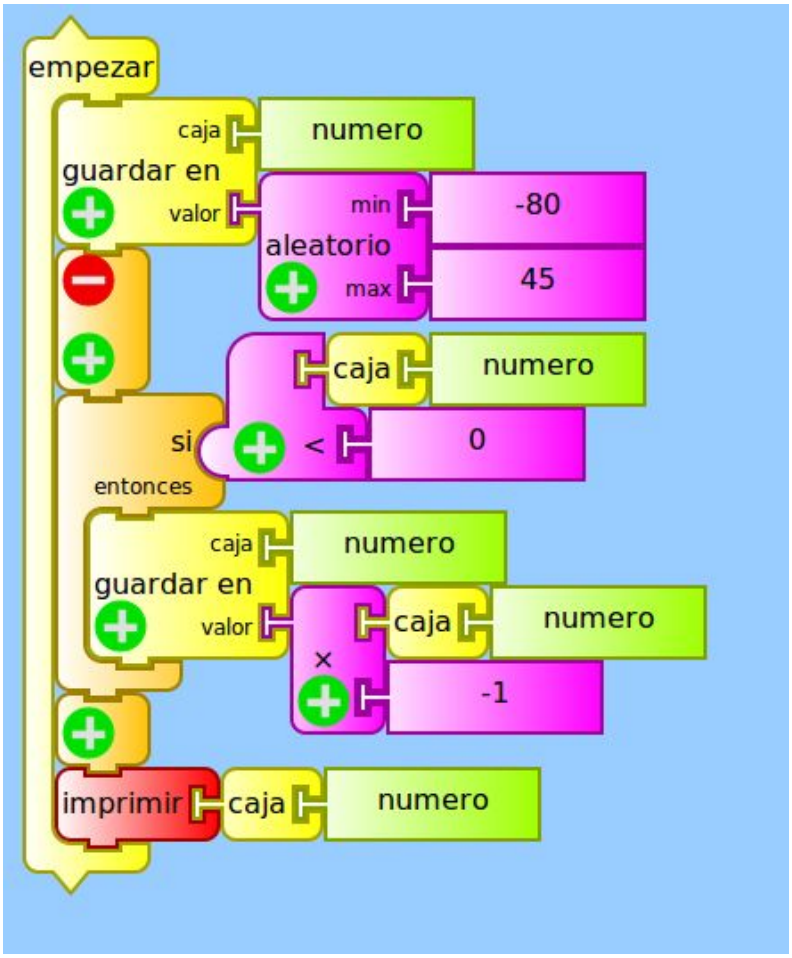


Figura 5.4.2.D: Ejemplo de programa con valor absoluto.

El funcionamiento de la estructura **si-entonces** es el siguiente:

1. Se evalúa la expresión booleana.
2. Si el resultado de la expresión booleana es verdadero se ejecuta el conjunto de instrucciones. Por otro lado, si el resultado es falso se saltea dicho conjunto.
3. Se continúan ejecutando las instrucciones que puedan existir siguientes a la estructura.

Las estructuras **si-entonces** y **si-entonces-sino** corresponden a otra de las componentes de la programación: **la selección**. Al estudiar estas estructuras, se puede ver que ahora nuestros programas pueden **seleccionar** diferentes caminos de ejecución, es decir que podemos construir algoritmos que resuelven los problemas que atacamos, tomando decisiones en función de los datos que se tienen a la hora de

su ejecución, ampliando de forma considerable la cantidad de problemas que se pueden resolver a través de la programación.

5.4.3 Estructura "mientras"

Como vimos en la subsección del bloque repetir, es posible en programación utilizar estructuras que repiten acciones. Hay veces que la resolución del problema nos permite saber de antemano cuántas veces vamos a repetir dichas acciones (y en ese caso se puede usar **repetir**), pero hay otras veces en la que no es posible, ya que dependen de datos de entrada. Veamos el siguiente problema:

Se quiere obtener una y otra vez números aleatorios entre **-10** y **45**, y se quiere contar cuántos números positivos (mayores o iguales a 0) se obtienen antes del primer negativo. Por ejemplo, si se obtiene:

- 2, 0, 15, -2 entonces la cantidad es 3 ya que fueron 3 positivos antes del primer negativo.
- -7 entonces la cantidad es 0 ya que no hubo ningún positivo antes del primer negativo.
- 40, -6 entonces la cantidad es 1.

Ya que queremos contar números que se obtienen aleatoriamente, vamos a tener una variable que llamaremos **cantidad**, que se irá incrementando en 1 cada vez que se obtenga un número positivo. También tendremos una variable que llamaremos **numero**, en la que se irá guardando dicho número obtenido. ¿En qué casos aumentará la variable que cuenta los números positivos? Aumentará **mientras** el **numero** obtenido aleatoriamente sea mayor que -1. Por lo tanto el pseudocódigo queda:

```
1) numero := obtenerAleatorio(-10,45)
2) cantidad := 0
3) mientras (numero es mayor que -1) entonces:
4)     aumentar cantidad en 1
5)     numero := obtenerAleatorio(-10,45)
6) imprimir(cantidad)
```

En el pseudocódigo presentado las instrucciones tienen números para explicarlas una a una.

Instrucción 1

Obtenemos el número que se genera aleatoriamente entre -10 y 45, y lo guardamos en la variable **numero**. Es decir, obtenemos el primer número. Si ejecutamos varias veces el programa, en algunas dicho número será negativo y en otras positivo.

Instrucción 2

A la variable **cantidad** se le asigna el valor **0**. ¿Por qué? Porque estamos contando los positivos antes del primer negativo, y si vamos a aumentar en **1** dicha variable, debe inicialmente tener un valor. Parece correcto asignarle **0** ya que es el caso en que el primer número obtenido es negativo (la cantidad daría 0 en ese caso).

Instrucción 3

Aparece la estructura **mientras**. Dicha estructura al igual que el **si-entonces** y **si-entonces-sino** necesita una expresión **booleana** (una expresión de la que pueda determinarse si es verdadera o falsa) que en este caso es: (**si el valor almacenado en numero es mayor que -1**). Esta estructura ejecuta las instrucciones que se le indiquen **mientras** la expresión sea verdadera. Cuando sea falsa se terminan de ejecutar y se continúa con el programa. En este ejemplo, las instrucciones dentro del **mientras** son la 4 y 5 (que están corridas un poco hacia la derecha, análogo al ejemplo del **si-entonces-sino**). Cuando la expresión sea falsa, se salta a la instrucción 6.

Instrucción 4

En este caso se incrementa en **1** la cantidad de números positivos contados. Esta instrucción se ejecuta cada vez que la expresión booleana se evalúa y da verdadero.

Instrucción 5

Se actualiza la variable **numero** con un nuevo número aleatorio entre -10 y 45. Recordemos que queremos obtener números hasta encontrar el primero negativo. Ésta es la última instrucción dentro del **mientras**. ¿Qué sucede cuándo se termina de ejecutar esta última instrucción? El programa no avanza hacia la instrucción 6 directamente, sino que vuelve a evaluar la expresión booleana, sólo que ahora tiene un nuevo valor en la variable **numero**. Entonces al evaluar nuevamente la expresión, si tiene como resultado verdadero ejecuta de nuevo las instrucciones 4 y 5 (ejecuta de nuevo las instrucciones del **mientras**) y sino salta a la instrucción 6, que es la primera luego del fin de la estructura.

Instrucción 6

El bloque de **mientras** se terminó la primera vez que se obtuvo un número negativo, es decir, la primera vez que la expresión **(numero es mayor que -1)** tiene como resultado el valor falso al ser evaluada. Entonces al terminar el bloque **mientras**, se sigue con la siguiente instrucción, es decir la 6. La misma muestra en pantalla el valor almacenado en **cantidad**. Esto significa que al finalizar el programa muestra la cantidad de números positivos obtenidos aleatoriamente.

Al ejecutar el programa varias veces: ¿Siempre va a imprimir la misma cantidad? La respuesta es que no, ya que el bloque aleatorio funciona justamente de forma aleatoria, es decir, cuando consultemos una y otra vez va a ir devolviendo diferentes valores entre -10 y 45. A continuación se muestra el pseudocódigo traducido a programa en TurtleBots:

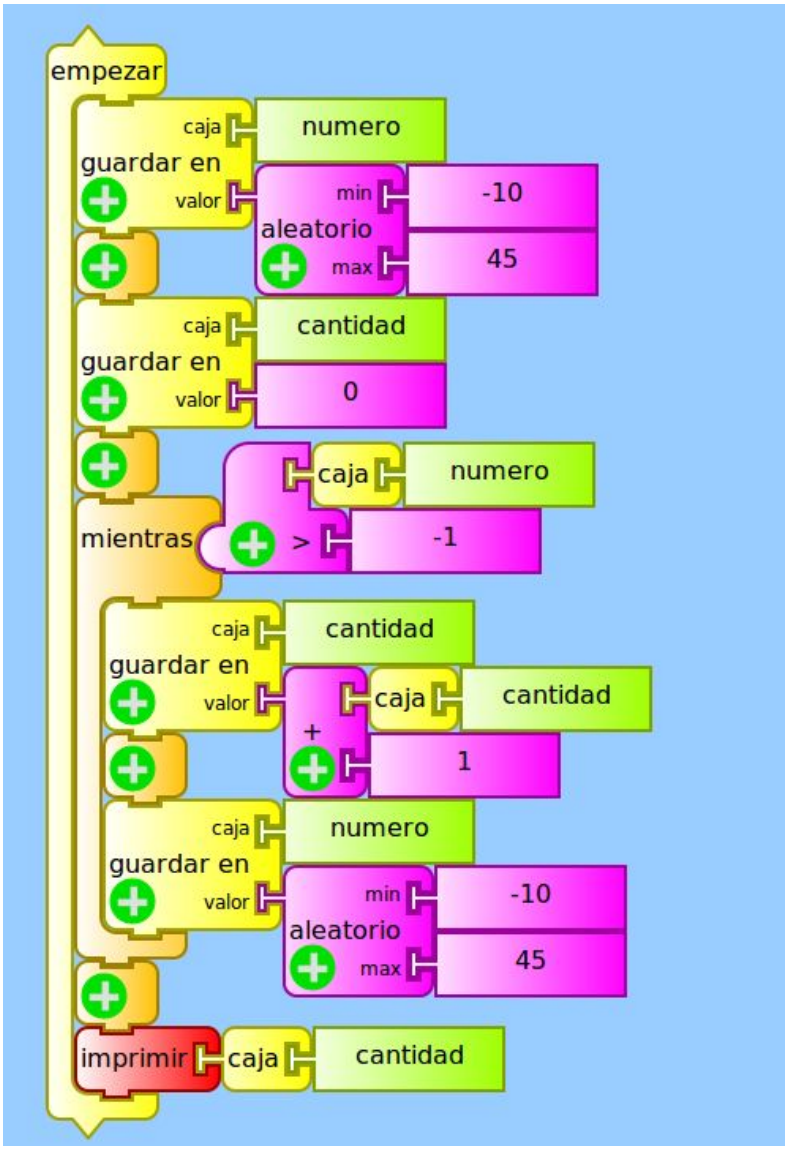


Figura 5.4.3.A: Ejemplo de programa con mientras.

El funcionamiento de la estructura **mientras** es el siguiente:

1. Se evalúa la expresión booleana.
2. Si el resultado de la expresión booleana es verdadero se ejecuta el conjunto de instrucciones y se vuelve al paso 1. Por otro lado, si el resultado es falsa no se ejecuta el bloque de instrucciones.
3. Se continúan ejecutando las instrucciones que puedan existir siguientes a la estructura.

Observar que si la primera vez que se llega a la ejecución de un bloque **mientras**, su expresión booleana es falsa, entonces nunca se entraría a su bloque de instrucciones. Veamos el bloque mientras en TurtleBots:

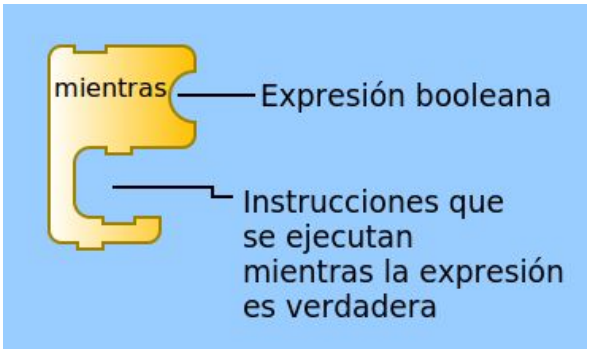


Figura 5.4.3.B: Bloque mientras.

La estructura **mientras** (que en inglés se le llama **while**) debe tener siempre una variable, que hace que la expresión booleana sea falsa en algún momento de la ejecución del programa, es decir, una variable que asegure que no se queda ejecutando en un bucle infinito. A esta variable se le llama generalmente **variable de control**; que en nuestro ejemplo era la variable **numero**; ya que de ella dependía la expresión booleana del bloque mientras. Dicha variable de control, debe cambiar dentro del bloque de instrucciones de la estructura **mientras**, para asegurar que en algún momento, la expresión booleana (que depende de la variable) se haga falsa y se termine el bucle de instrucciones. Veamos un ejemplo en TurtleBots que no toma en cuenta la variable de control:

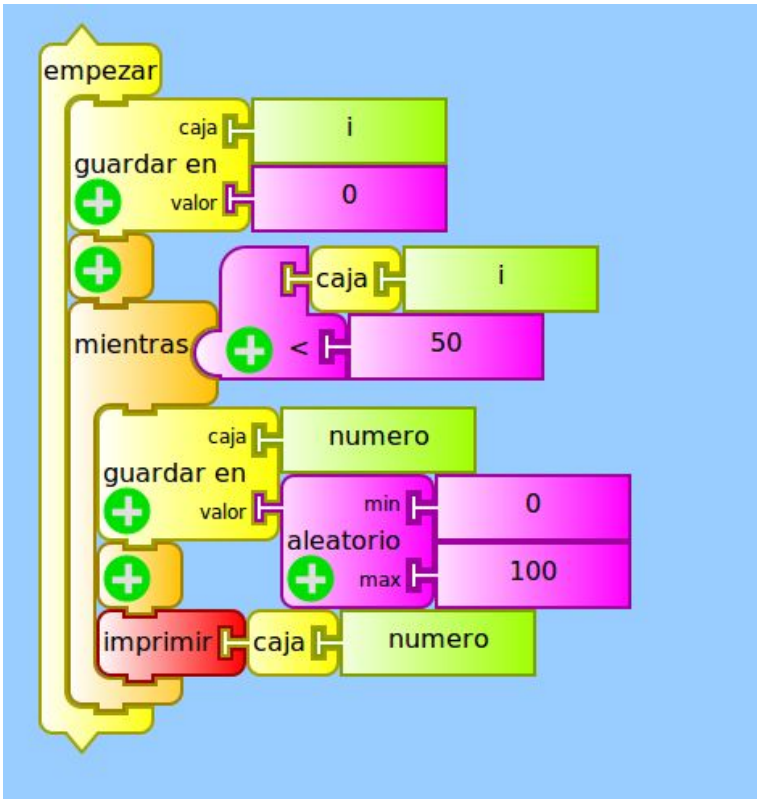


Figura 5.4.3.C: Mientras con error de variable de control.

Este programa de ejemplo, tiene una estructura **mientras** que nunca termina de ejecutarse (se queda en un bucle infinito). Lo primero que hace es asignarle a la variable **i** el valor **0**. Lo siguiente es la estructura **mientras** cuya expresión booleana es: (**i < 50**). Es decir que las instrucciones del bloque **mientras** se ejecutan mientras **i** sea menor que **50**. Podemos afirmar que **i** es la variable de control del bloque, y dentro de sus instrucciones su valor nunca es modificado. Por lo tanto **siempre** será verdadera la expresión dado que **i** no cambia y siempre tendrá el valor **0**. Por lo tanto si la expresión booleana del **mientras** nunca se hace falsa, nunca se terminan de ejecutar sus instrucciones. En conclusión, si queremos que nuestros programas terminen, debemos trabajar con cuidado con las variables de control.

5.4.4 Estructura "hasta"

La estructura **hasta** es muy similar a **mientras**, pero semánticamente opuesta, es decir, la estructura **mientras** ejecuta un bloque de instrucciones mientras la expresión booleana sea verdadera, y la estructura **hasta** ejecuta un bloque de instrucciones hasta que su expresión booleana sea verdadera; veamos un ejemplo para aclarar esta cuestión. Supongamos que queremos hacer el mismo programa que en la subsección anterior (el de obtener números aleatorios entre -10 y 45 y contar la cantidad de positivos obtenidos) pero en vez de utilizar la

estructura **mientras**, utilizando la estructura **hasta**. El pseudocódigo sería el siguiente:

```
1) numero := obtenerAleatorio(-10,45)
2) cantidad := 0
3) ejecutar:
4)     aumentar cantidad en 1
5)     numero := obtenerAleatorio(-10,45)
6) hasta que (numero < 0)
7) imprimir(cantidad)
```

Podemos ver que todas las instrucciones son iguales excepto la **3** y **6.** En ellas se cambia el **mientras** por el **hasta**, y lo más importante: cambia la expresión booleana. En el ejemplo con bloque mientras, la expresión booleana es **(numero > -1)** es decir, se ejecuta el bloque de instrucciones mientras que el número obtenido sea mayor que **-1**. Y en el ejemplo con bloque **hasta** la expresión booleana es **(numero < 0)** es decir, que se ejecuta el bloque hasta que el número obtenido sea menor que **0**, que equivale a decir: hasta que el número obtenido sea menor o igual que **-1**, ya que siempre se obtienen números enteros del bloque de números aleatorios. En conclusión, las condiciones booleanas son opuestas, por cómo funcionan las estructuras: la estructura **mientras** ejecuta mientras la expresión booleana es verdadera, y la estructura **hasta** ejecuta mientras la expresión booleana es falsa (es decir, ejecuta hasta que sea verdadera).

El funcionamiento de la estructura **hasta** es el siguiente:

1. Se ejecuta su bloque de instrucciones.
2. Se evalúa su expresión booleana. Si el resultado es verdadero se termina. Por otro lado, si es falso se vuelve al paso 1.
3. Se continúan ejecutando las instrucciones que puedan existir siguientes a la estructura.

Observar que el bloque **hasta** siempre ejecuta al menos una vez su bloque de instrucciones, porque a diferencia de **mientras**, evalúa la expresión booleana al final. Veamos el bloque **hasta** cómo es en TurtleBots:

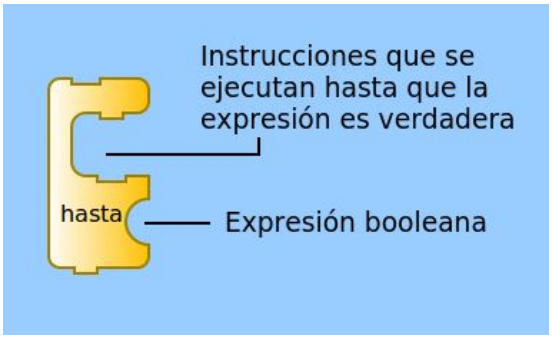


Figura 5.4.4.A: Bloque hasta.

Volviendo al ejemplo, el resultado del programa con las dos estructuras es el mismo: se obtienen números aleatorios entre -10 y 45 y se van contando los positivos, esto se hace hasta que aparece uno negativo. Pero hay una pequeña diferencia: ¿Qué pasa en el caso borde en el que el primer número obtenido es negativo? Para el ejemplo con bloque **mientras** no hay problema, porque como ya sabemos, la expresión booleana de primera da falsa y no se ingresa a su bloque de instrucciones, por lo tanto imprimirá la cantidad **0** (porque se obtuvieron **0** números positivos). Por otro lado, el bloque **hasta, siempre ejecuta su bloque de instrucciones al menos una vez porque evalúa la expresión booleana al final**, por lo que en el caso de que el primer número aleatorio obtenido sea negativo, igual se entrará una vez (al menos) a su bloque de instrucciones. Esto es un problema porque si tenemos por ejemplo la siguiente secuencia de números aleatorios:

-3 14 2 6 -9 entonces ese programa no imprimirá **0**, sino que imprimirá **4**, que es incorrecto ya que no resuelve el problema que se está atacando. El programa mostrado anteriormente utilizando el bloque **hasta,** tal como está funciona bien sólo cuando el primer número obtenido no es negativo. Dado que es importante elaborar programas que funcionen bien para todos los casos, es necesario corregirlo. Lo que haremos simplemente es consultar si el primer número obtenido es negativo, en caso de que no lo sea ejecutaremos el bloque **hasta**. El pseudocódigo corregido es el siguiente:

```
numero := obtenerAleatorio(-10,45)
cantidad := 0
si (numero > -1) entonces
      ejecutar:
            aumentar cantidad en 1
            numero := obtenerAleatorio(-10,45)
      hasta que (numero < 0)
imprimir(cantidad)
```

Podemos ver que el programa luego de obtener el primer número verifica si es mayor que **-1** , si lo es, se ejecuta el bloque **hasta**, de lo contrario saltea dicho bloque e imprime **0**, solucionando el problema que

se generaba si el primer número aleatorio obtenido era negativo. Veamos el pseudocódigo anterior escrito en TurtleBots.

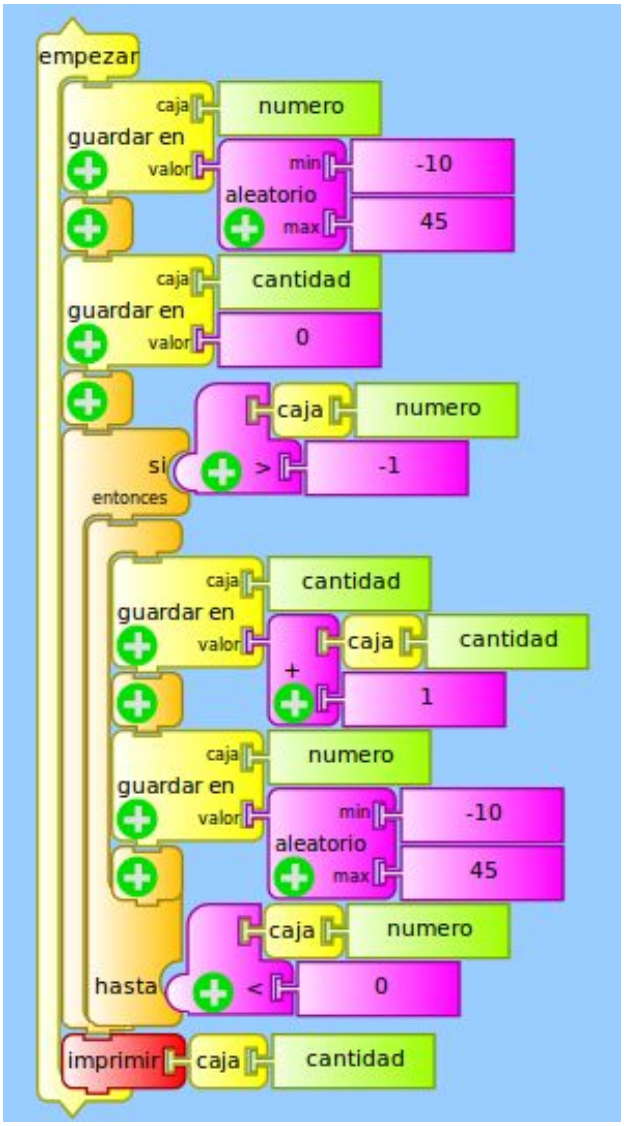


Figura 5.4.4.B: Ejemplo de programa con bloque hasta.

De las estructuras vistas en esta subsección y en anteriores, podemos afirmar que **repetir**, **mientras**, y **hasta** corresponden a la tercera componente de programación nombradas al principio: **la repetición**. Esta componente refiere al hecho de que en programación podemos indicar que existe un conjunto de instrucciones que se repite una y otra vez según ciertas condiciones (expresiones booleanas para el caso de **mientras** y **hasta**, y el número indicado para el caso de **repetir**). En conclusión, la **secuencia, selección** y **repetición** se pueden combinar en programación para construir algoritmos que resuelven los diferentes problemas que enfrentamos.

5.4.5 Estructura “por siempre”

En TurtleBots existe otra estructura que se llama **por siempre** que tiene la siguiente forma:

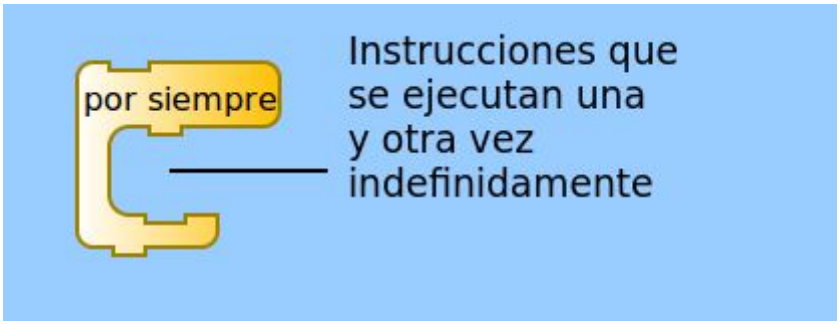


Figura 5.4.5.A: Bloque por siempre.

Los bloques que se coloquen dentro se ejecutarán en orden una y otra vez hasta que se cancele el programa o se apague la computadora. Se comporta como un bloque **mientras** cuya expresión booleana siempre es verdadera, o de bloque **hasta** cuya expresión booleana es siempre falsa, es decir que es un bloque que no termina. Por eso su nombre. Si colocamos instrucciones abajo de un bloque **por siempre**, las mismas nunca se ejecutarán, porque el programa se queda “por siempre” ejecutando las instrucciones que contiene.

La pregunta es, ¿Tiene sentido un programa que “nunca termina”? De hecho sí, varios programas que utilizamos como usuarios están dentro de un bloque por siempre; y en el caso de TurtleBots su uso más importante es al programar robots. Muchas veces, los problemas que queremos resolver con robots requieren que el robot se encuentre constantemente sensando (consultando los valores que devuelven sus sensores) para tomar decisiones y actuar en función de los mismos, y eso se puede hacer utilizando un bloque **por siempre**.

Un ejemplo de uso del robot Butiá 2.0 con el bloque por siempre, es el seguidor de líneas, es decir resolver el problema de que el robot siga un camino negro sobre una superficie blanca, sin salirse del mismo y sin retroceder. Veamos para el caso en que la pista tiene forma ovalada:

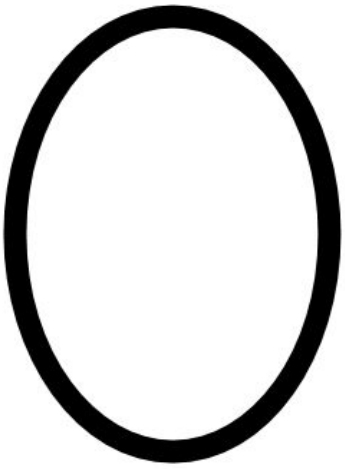


Figura 5.4.5.B: Pista ovalada.

Para esta pista, una posible solución es la siguiente:

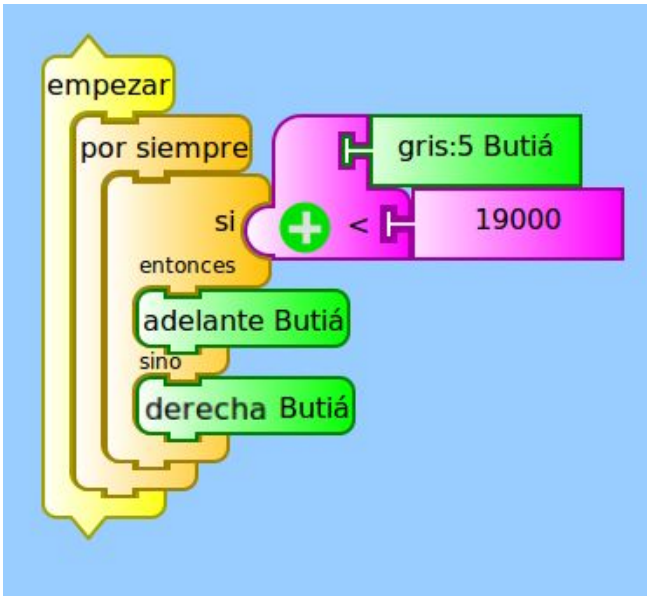


Figura 5.4.5.C: Ejemplo de solución de pista ovalada.

Como se puede apreciar, todo el programa está dentro de un bloque **por siempre**. Esto se debe a que el robot debe estar sensando constantemente y tomando una decisión de avanzar o girar a la derecha, es decir que: sensa, toma la decisión y lo hace nuevamente. Este programa hace que el robot siga la línea indefinidamente.

Nota: El valor 19000 es a modo de ejemplo, y corresponde al resultado de calibrar el sensor. Dicho valor y el algoritmo se explican más

detalladamente en el Manual de uso de TurtleBots y Butiá 2.0[11] (Aguirre, A. Michetti,B. 2019)

6 Aplicación de los conceptos presentados

En esta sección se trabajará con un desafío del evento sumo.uy[12] del año 2013. Dicho evento se lleva a cabo todos los años en la FING, en el cual se realizan competencias de robots, exposiciones, charlas y talleres; todos teniendo como eje central a la robótica autónoma. Se presentará el desafío escolar del año mencionado, que es un problema planteado para ser resuelto con el robot Butiá 2.0 y su kit básico de sensores. Luego se hará un análisis del problema y se construirá una solución aplicando los conceptos estudiados en este librillo.

6.1 Desafío planteado

El desafío con el que se va a trabajar, pertenece a la categorìa escolar de la edición de sumo.uy del año 2013. Dicha categoría está pensada para que participen niños de primaria, acompañados por tutores que los guíen en la resolución, utilizando al robot Butiá 2.0. La letra del desafío se encuentra disponible en la página de sumo.uy, acompañada de muchas otras de otros años, con interesantes problemas para resolver con robots autónomos. La realidad del planteo es la siguiente:

"En la pizzería "Pizza&Chips" están muy preocupados por el incremento de pedidos en el último mes. Para optimizar los tiempos de entrega de los mismos han decidido tener un robot autónomo móvil que sea capaz de cumplir con la tarea. El deliverybot Junior tiene capacidad para entregar un pedido por vez, y debe hacerlo lo más rápido posible para evitar que la pizza se enfríe. El pedido recibido por deliverybot indica que debe partir de la pizzería con un pedido de pizza margarita y entregarla en la segunda casa en el camino. Al momento de la entrega descubre que se trata de la casa de su amigo Artoo, por lo que decide quedarse a comer con él." (InCo, 2013)

Ahora veamos las dos primeras reglas del desafío:

[11] Manual de uso básico de TurtleBots y Butiá 2.0, https://www.centrosmec.gub.uy/innovaportal/file/823/1/manual-turtlebots---butia-2.0_05-11-18-de-a-una-pag.pdf, visitada en mayo del 2019.

[12] Evento sumo.uy, https://sumo.uy, visitada en mayo del 2019.

"1. El robot deberá cumplir con la entrega de una pizza, la misma deberá ser entregada en la segunda casa contando desde la pizzería. Es decir, deberá pasar por alto la primera casa.
2. La entrega de un pedido se cuenta como válida si el robot sale de la pizzería, pasa visiblemente frente a la casa (el frente del robot cruza la línea imaginaria, perpendicular a la calle, que contiene el lado de la casa que se encuentra más alejado de la pizzería, (...) y se detiene. Es decir, deberá ir desde la pizzería hasta la segunda casa, "dejar el pedido" y quedarse a comer la pizza con su amigo."

La figura que refleja la entrega de una pizza es la siguiente:

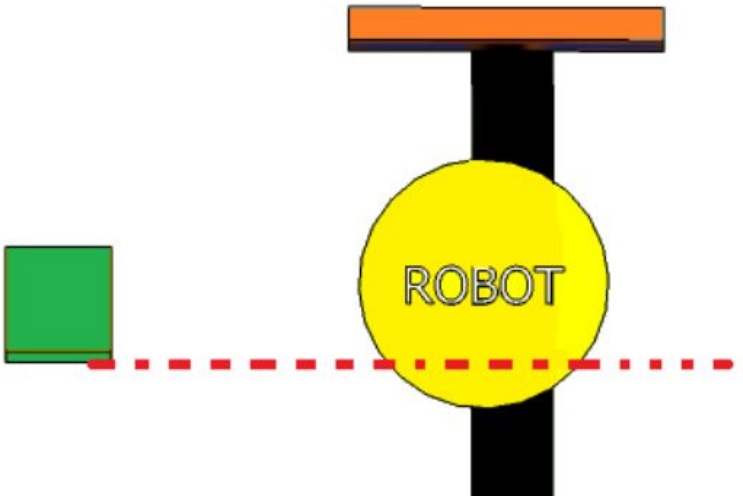


Figura 6.1.A: Entrega de pizza.

A continuación, un ejemplo de un escenario posible para el desafío:

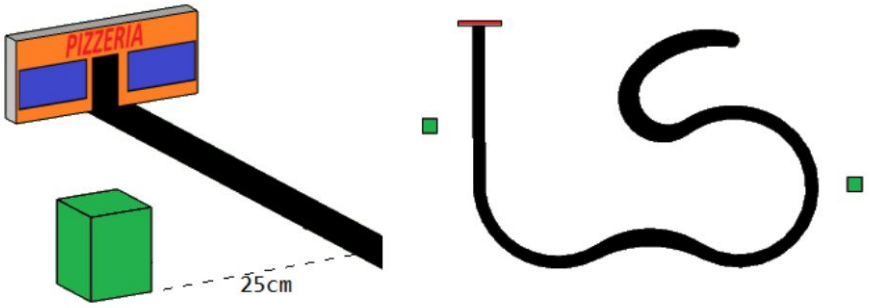


Figura 6.1.B: Ejemplo de escenario desafío escolar.

Si leemos toda la letra, y nos abstraemos de la realidad planteada enfocándonos en los aspectos importantes para resolver el

problema, podemos concluir que lo que debe hacer el robot es: seguir una línea negra, contar cubos que estarán a 25 centímetros hacia la derecha de la línea, y detenerse al pasar el segundo cubo, como se puede apreciar en la figura. En dicho desafío, es importante hacerlo rápido ya que se define por tiempo; pero dado que nos interesa trabajar los conceptos aprendidos, no interesa en este caso la velocidad con la que actuará el robot.

6.2 Solución propuesta

Para atacar el problema planteado, vamos a utilizar el principio de "divide y vencerás" (en inglés: divide and conquer). Dicho principio hace referencia al refrán que implica resolver un problema difícil, dividiéndolo en partes más simples hasta que la resolución de las partes se torna sencilla. Luego, la solución del problema principal se construye con la combinación de las soluciones encontradas.
En este caso queremos lograr que el robot se mueva siguiendo una línea negra (la calle de la ciudad) y vaya contando cubos (las casas) que van a estar a su derecha, por lo que a grandes rasgos podemos resolver nuestro problema principal en dos problemas más sencillos que serían:

- Seguir la línea negra.
- Contar cubos.

Luego de analizar y llegar a las soluciones de los problemas mencionados, vamos a combinarlas para poder construir la solución al problema general.

6.2.1 Seguidor de líneas

El problema del robot seguidor de líneas está planteado y resuelto en el **Manual de uso básico de TurtleBots y Butiá 2.0** (Aguirre, A. Michetti,B. 2019), donde se muestra como colocar los sensores, como calibrarlos, y cómo utilizar las medidas obtenidas para resolver el problema. Queda a cargo del lector revisar dicho planteo y solución. Como podemos apreciar en el manual, la solución propone colocar dos sensores de grises en el robot, apuntando hacia el piso de manera de que quede la línea en el medio, para poder reconocer el color negro y el color blanco. A continuación una imagen de lo propuesto:

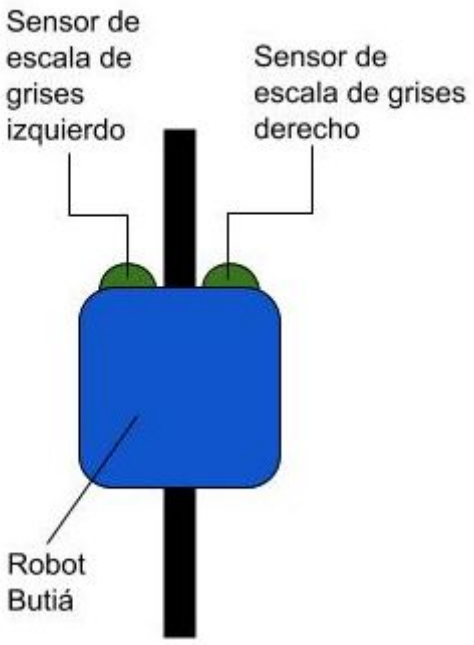


Figura 6.2.1.A: Configuración de sensores para seguidor de líneas.

A partir de la disposición de los sensores señalada, se hace en el manual un análisis de la situación y se llega al siguiente algoritmo:

```
Por siempre:
     Si ambos sensores ven blanco:
          Avanzar
     Sino:
          Si el sensor derecho ve negro:
               Girar hacia la derecha
          Sino:
               Girar hacia la izquierda
```

Con este algoritmo planteado y con un ejemplo de calibración de sensores, se llega al siguiente programa en TurtleBots:

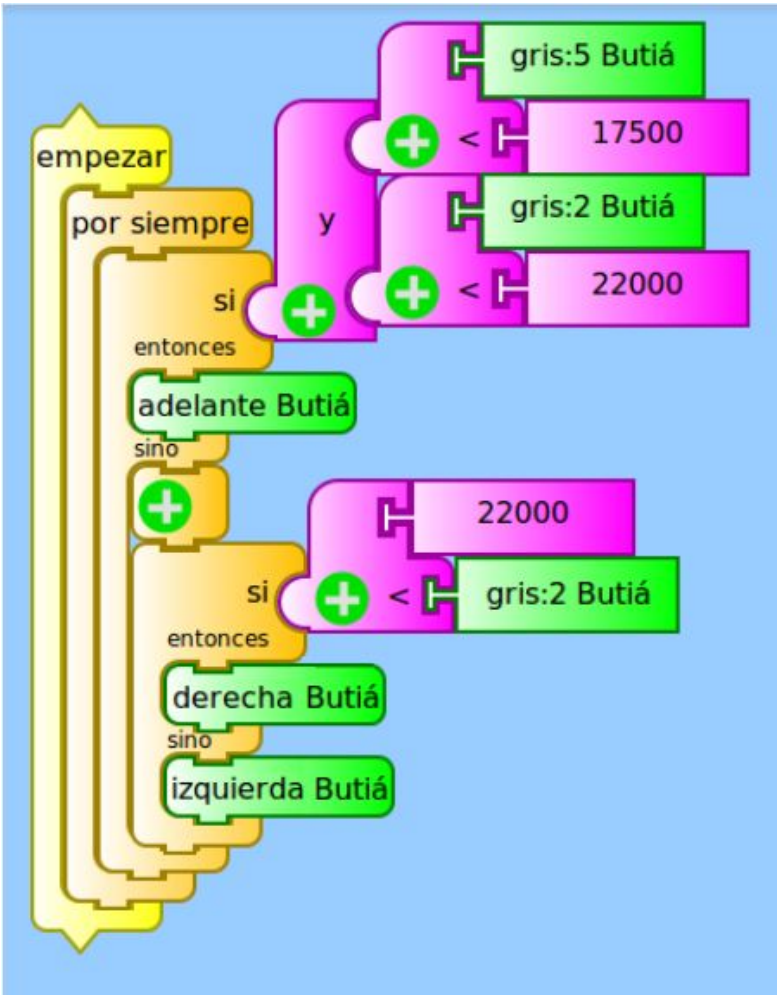


Figura 6.2.1.B: Ejemplo de solución de seguidor de línea.

En la solución mostrada los números **17500** y **22000** son a modo de ejemplo, que surgen de la siguiente suposición:

- El sensor colocado a la **izquierda** está en el **puerto 5**, y al hallar su umbral obtuvimos el valor **17500**.
- El sensor colocado a la **derecha** está en el puerto **2**, y al hallar su umbral obtuvimos el valor **22000**.

Podemos ver que el programa tiene la estructura **por siempre** dado que siempre queremos que se tome una decisión en función de lo que reciben los sensores, y luego una estructura **si-entonces-sino** dentro de otra, en la que primero se va hacia adelante si el robot sensa blanco con ambos sensores, y sino consulta cuál de los dos sensores detectó negro para ver hacia donde doblar y no perder el camino. Tomemos la solución para aplicar algo más trabajado en esta publicación, que son las variables: los umbrales hallados para cada

sensor son almacenados en dos variables y les pondremos los siguientes nombres mnemotécnicos:

- **umbral_derecho**: variable en la que almacenaremos el umbral que hallamos para el sensor derecho luego de calibrarlo.
- **umbral_izquierdo**: análogo a la variable previamente mencionada pero para el sensor izquierdo.

Con la definición de estas variables, el programa que soluciona el problema de seguir la línea tendría el siguiente pseudocódigo:

```
umbral_izquierdo := 17500
umbral_derecho := 22000
Por siempre:
      Si (sensor_gris(PUERTO_5) < umbral_izquierdo) y
           (sensor_gris(PUERTO_2) < umbral_derecho):
            Avanzar
      Sino:
            Si (umbral_derecho < sensor_gris(PUERTO_2)):
                  Girar hacia la derecha
            Sino:
                  Girar hacia la izquierda
```

Implementando este pseudocódigo en TurtleBots se tiene el siguiente programa:

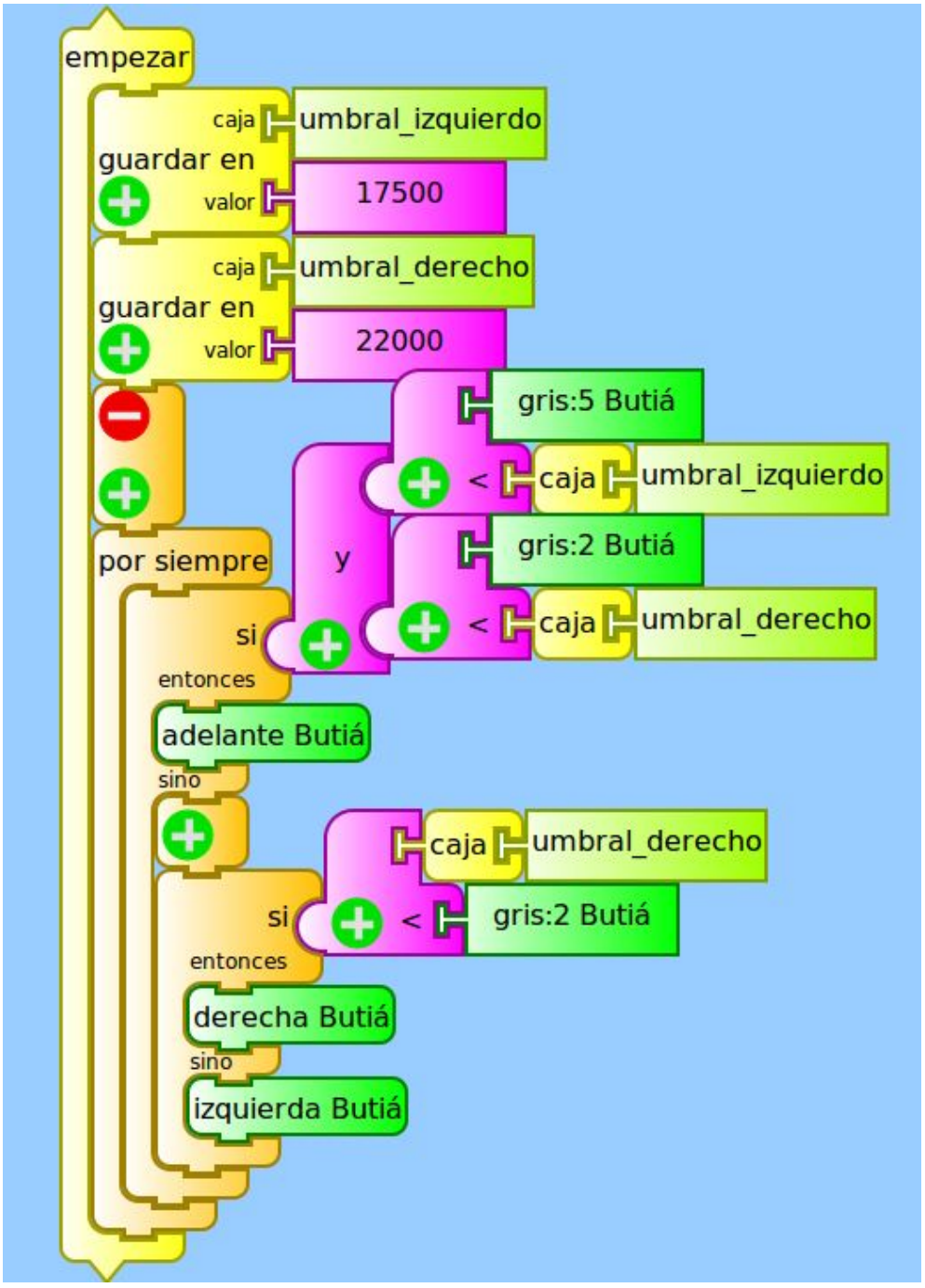


Figura 6.2.1.C: Ejemplo de solución de seguidor de línea utilizando variables.

¿En qué ayuda utilizar variables? Supongamos que nuestro programa crece y se utilizan los valores de umbral de los sensores en varios lugares del mismo. Si por alguna razón es necesario recalibrar los sensores (por ejemplo se mueve de lugar la pista donde se encuentra el

robot, o cambia la luz del entorno), va a ocurrir que los valores de los umbrales cambiarán; y si no se usa variables se debería mirar con atención el programa y buscar y modificar en cada ocurrencia del mismo valor del umbral. Sin embargo, con las variables, el cambio genera menos re-trabajo ya que alcanza con modificar el valor que se le asigna inicialmente a la respectiva variable, es decir, modificar una instrucción del programa. Utilizar las variables en este caso hace nuestro programa más mantenible, es decir, más fácil de arreglar o cambiar.

A esta altura ya disponemos la solución del subproblema de seguir la línea. Por lo que en la subsección siguiente se atacará el problema de contar cubos.

6.2.2 Contador de cubos

Dado que tenemos cubierto el problema de seguir la línea, debemos enfocarnos en la resolución del problema de contar cubos. Las preguntas que surgen son: ¿Qué sensor usamos para contar cubos? ¿Dónde lo colocamos en el robot? Dado que se quiere detectar un objeto a 25 cm del robot, el sensor adecuado sería el de distancia ya que podemos colocarlo apuntando hacia donde eventualmente podría aparecer un cubo, y nuestro programa contará el cubo cuando la distancia percibida cambie. Una posible respuesta a la segunda pregunta consiste en colocar en la parte delantera del robot apuntando hacia su derecha, puesto que el desafío establece que el cubo estará a la derecha del camino. El sensor debe ponerse adelante porque el robot debe detenerse al pasar el cubo desde su borde delantero, como podemos ver en la Figura 5.1.A. Por lo tanto el sensor de distancia quedará como se puede ver en la siguiente figura (mirando al robot desde arriba):

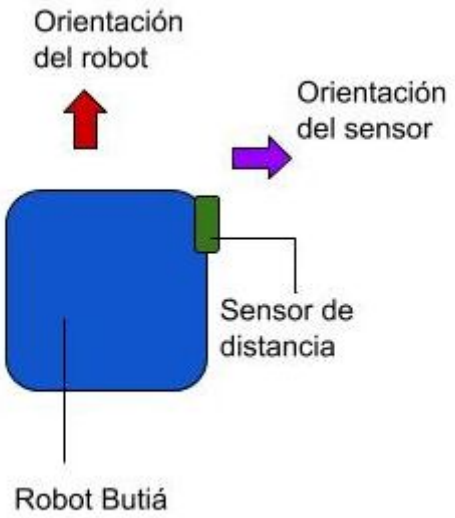


Figura 6.2.2.A: Configuración de sensor de distancia.

Si colocamos un sensor de distancia "apuntando hacia la nada"; es decir, apuntando hacia un lugar donde no hay ningún objeto lo suficientemente cerca como para que el sensor lo pueda percibir, entonces el mismo retornará un valor cercano a **65535** (que es el mayor valor que podría devolver el sensor). Si mientras "apunta hacia la nada" le colocamos cerca un cubo (en este caso a 25 cm), dicha distancia retornada bajará cercana a un valor **X** que le llamaremos umbral. En el **Manual de uso básico de TurtleBots y Butiá 2.0** (Aguirre, A. Michetti,B. 2019) se explica cómo hallar ese umbral y queda para el lector revisar dicho análisis. Veamos con imágenes los ejemplos mencionados:

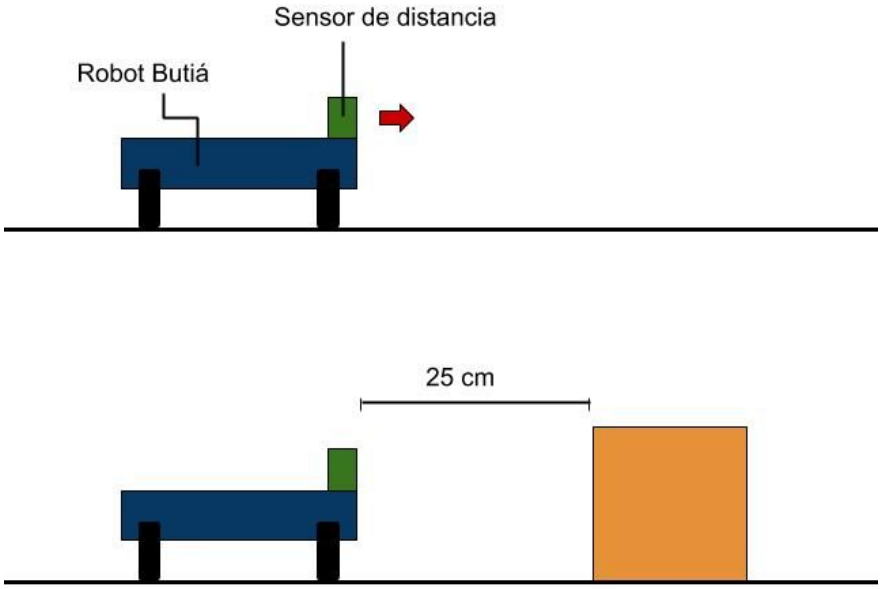


Figura 6.2.2.B: Ejemplo de detección con sensor distancia.

En la imagen de arriba el sensor apunta hacia donde indica la flecha roja, y no tiene ningún objeto lo suficientemente cerca para percibirlo, por lo que los valores que retorna son los más altos posibles. Luego, en la imagen de abajo, al aparecer un cubo a 25 cm, va a retornar un valor entorno al umbral, por lo que podemos afirmar que:

- Si el sensor retorna un valor mayor al umbral, entonces no está detectando un cubo.
- Si el sensor retorna un valor menor al umbral, entonces sí lo está detectando.

Ahora sabemos cuando podemos afirmar que el robot está detectando cubo y cuando no. Lo que queda responder es, ¿Cuándo contamos? o mejor dicho, ¿Cuando aumentamos el contador?. A simple

vista esta pregunta tiene una respuesta sencilla: ¡Aumentamos el contador cuando el sensor detecta la presencia de un cubo!

Sin embargo esa respuesta es incorrecta, el problema no es tan sencillo de resolver como parece a simple vista. Veamos el problema de aumentar el contador cuando el robot detecta la presencia de un cubo, mirando la situación desde arriba:

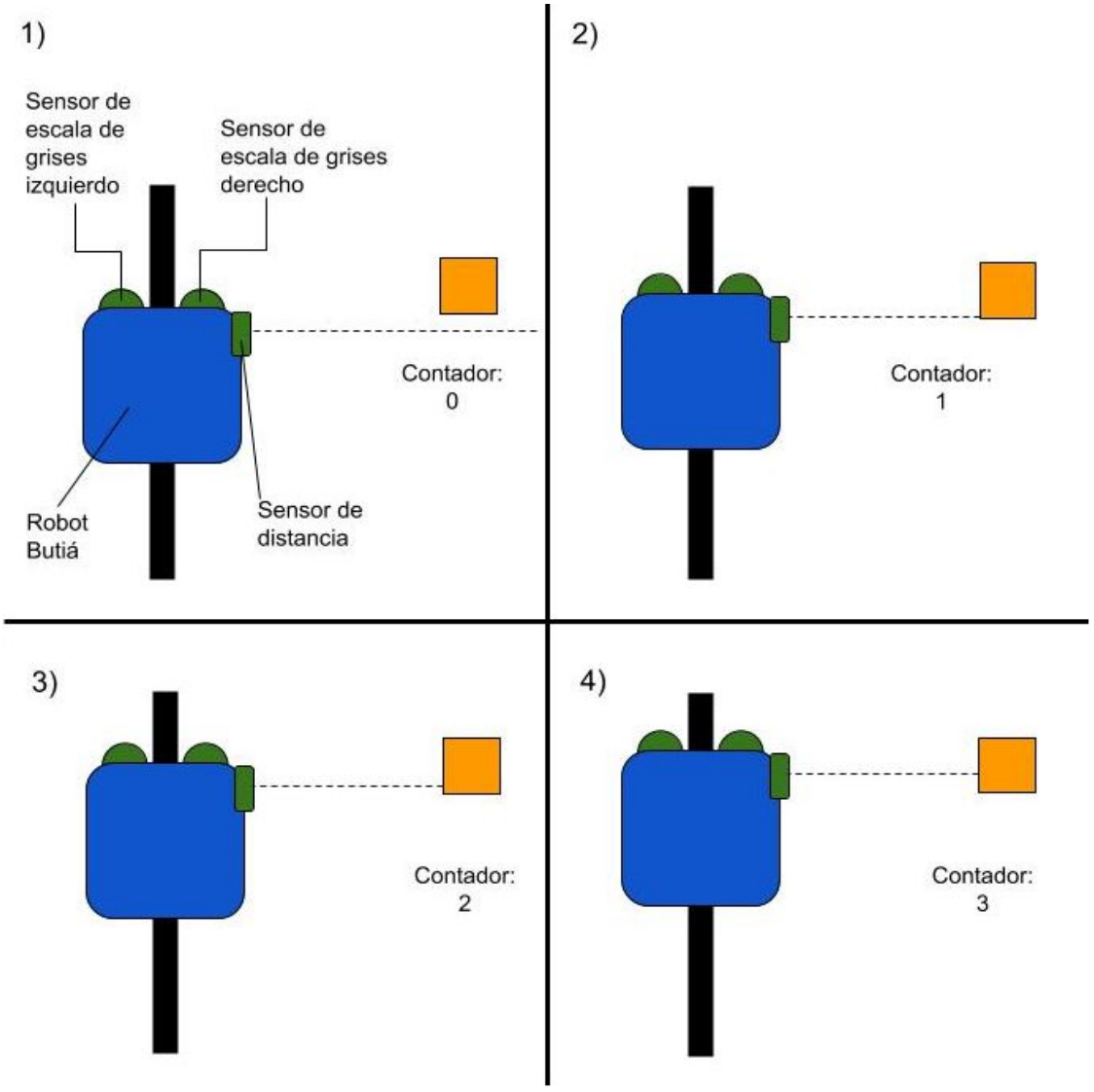


Figura 6.2.2.C: Robot detectando cubo.

Se puede apreciar que en el momento 1) el contador vale 0, ya que el robot aún no ha detectado la presencia de ningún cubo. Inmediatamente, en el momento 2) el robot detecta un cubo por primera vez y el contador aumenta, por lo que vale 1. El problema es que en el momento 3) sigue el robot detectando presencia del cubo, por lo que, al programarlo para que aumente cada vez que se detecte la

presencia de un cubo, el contador aumenta de nuevo, y en el momento 4) pasa lo mismo. Esto quiere decir que luego de que el robot pasó el cubo, el contador tendrá un valor bastante más alto que 1, cuando en realidad debería valer 1! Con esto concluímos que el problema de contar cubos no es tan sencillo como parecía. Vamos a enfrentar este problema aplicando otro concepto estudiado en este librillo: la máquina de estado.

6.2.3 Contador de cubos con máquina de estado

Para resolver el problema de contar cubos con máquina de estados, debemos plantearnos y responder las siguientes preguntas: ¿Cuál es la entrada? ¿Cuál es la salida? ¿Cuáles son los estados?

Para responder la primera pregunta es importante tener en cuenta qué es lo que le da la información del entorno, y en este caso es el sensor de distancia; que nos permite saber si se detecta o no la presencia de un cubo. Para la segunda, podemos afirmar que la salida es aumentar el contador, ya que es el problema que queremos resolver. ¿Y los estados? Los estados corresponden con lo que quiero recordar, y es importante analizar el inconveniente visto en la figura 5.2.2.C, en el que vemos que no hay que aumentar el contador al detectar un cubo. ¿Entonces cuándo?

Vamos a indagar en lo que debe suceder para poder deducir los estados y responder esa pregunta: inicialmente el robot comienza el desafío sin ver cubos, por lo que comienza a recorrer el camino sin detectar presencia de nada ni aumentar el contador. Luego en determinado momento detecta presencia de un cubo, recordar que aún no hay que aumentar el contador, la presencia del cubo es detectada muchas veces mientras pasa el sensor frente al mismo. Una vez que deja de detectarlo significa que acabamos de pasar por el cubo, es decir que es en ese momento cuando hay que aumentar el contador! Cuando en primera instancia lo detectó y luego dejó de detectarlo! Por lo que los estados son: **detectando_cubo** y **no_detectando_cubo,** ya que debemos recordar que detectamos la presencia de un cubo en la lectura anterior a la que indica la ausencia del cubo. La máquina sería la siguiente:

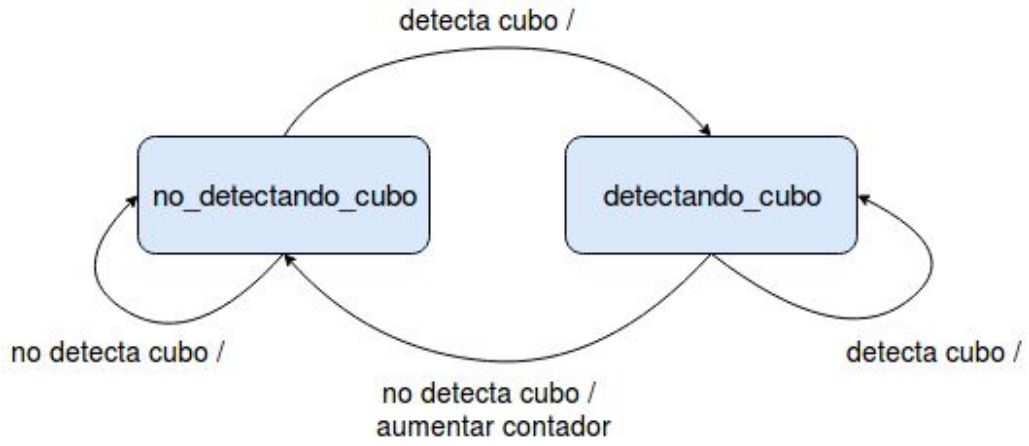


Figura 6.2.3.A: Máquina de estado para contar cubos.

Inicialmente la máquina se encuentra en el estado **no_detectando_cubo**, y mientras no se detecte presencia de un cubo, el estado no cambia, y la máquina no produce ninguna salida, es decir, no incrementa el contador. Una vez que se comienza a detectar la presencia de un cubo, se pasa al estado **detectando_cubo**, también sin aumentar el contador. Mientras detecte presencia de un cubo se mantiene la máquina en ese estado. Cuando deja de detectarlo, justo en ese momento se incrementa el contador, porque significa que el robot pasó el cubo y recién en este momento podemos decir que se ha detectado un cubo. A continuación se presenta un pseudocódigo de la máquina:

```
contador := 0
estado := no_detectando_cubo
por siempre:
      si estado es:
      no_detectando_cubo:
            si no detecta cubo:
                  no hacer nada
            si detecta cubo:
                  estado := detectando_cubo
      detectando_cubo:
            si no detecta cubo:
                  incrementar contador
                  estado := no_detectando_cubo
            si detecta cubo:
                  no hacer nada
```

Suponiendo que el umbral de detectar un cubo es 35400, y el sensor está en el puerto 6, un ejemplo de pasaje de este pseudocódigo a TurtleBots quedaría:

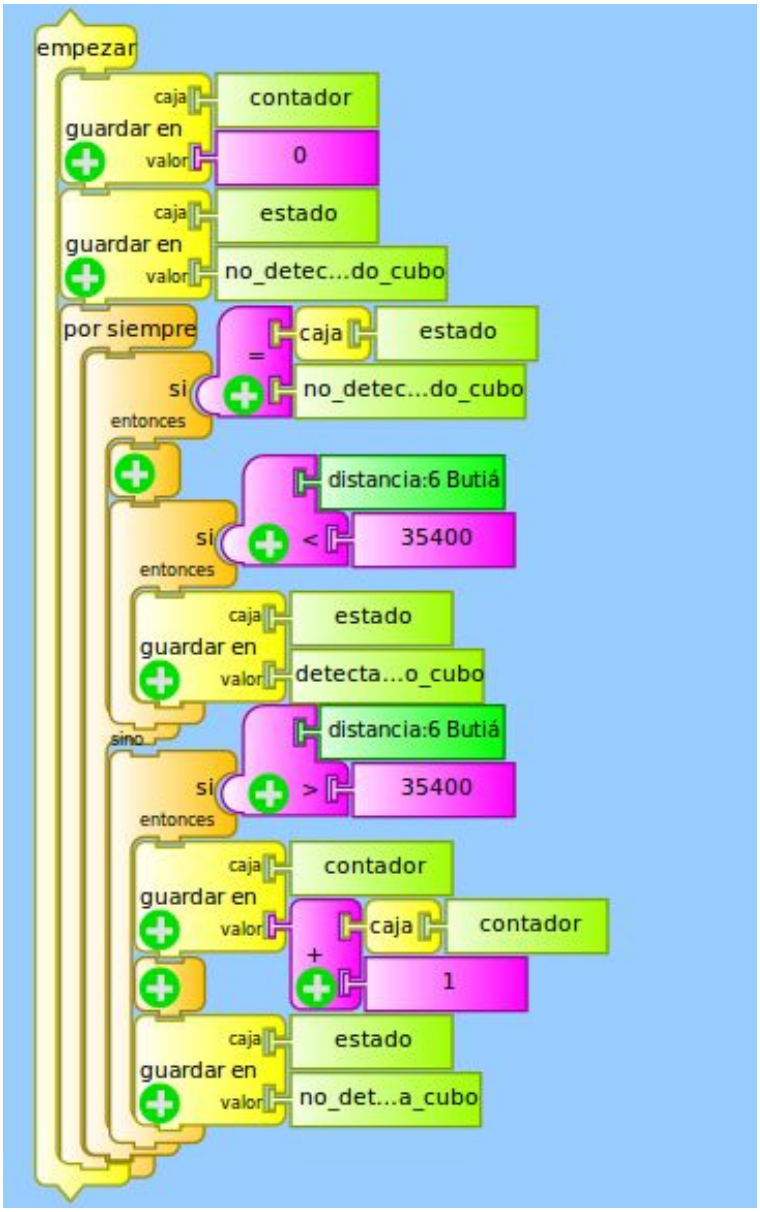


Figura 6.2.3.B: Máquina de estados de contar cubos en TurtleBots.

Esta es una implementación de la máquina de estado. Podemos ver que en las estructuras **si-entonces-sino**, no implementamos los casos del pseudocódigo en los que:

- se está en el estado **no_detectando_cubo** y no se detecta cubo, y
- está en el estado **detectando_cubo** y se detecta cubo

Dichos casos no se implementaron sencillamente porque en ellos no se hace nada, es decir no se produce una salida ni se cambia de

estado (ver el dibujo de la máquina). Recordemos que la misma resuelve el problema de contar cubos, y en la siguiente sección veremos cómo combinar ésta solución y la de seguir líneas para resolver el desafío.

6.2.4 Combinando las soluciones

Volviendo al comienzo del planteo del problema, recordamos que utilizamos el principio de divide y vencerás, en el que se divide un problema grande, en este caso el problema de resolver el desafío, en dos problemas más pequeños: seguir la línea negra, y contar cubos. Ahora que tenemos las soluciones a los problemas más pequeños, vamos a combinarlas para resolver el problema mayor.

6.2.4.1 Subprogramas

Ya tratamos la definición de programa, y ahora vamos a ver la definición de subprograma.Un subprograma es un programa contenido dentro de otro, y éste último lo usa cuando sea necesario para la atacar el problema que está resolviendo.

Más formalmente podemos decir que un subprograma S es un programa independiente dentro de otro programa P, y P utiliza a S en su ejecución, para resolver un aspecto particular de su algoritmo. A partir de esta definición, podemos definir al programa P como el programa que implementa la solución del desafío planteado, y que contiene a los subprogramas S_1 y S_2, donde S_1 es quien implementa la solución de seguir líneas, y S_2 la solución de contar cubos. Podemos llamar a P como "programa principal".

Siguiendo al ejemplo, vamos a pensar cómo combinar a S_1 y S_2 para escribir el algoritmo de P: sabemos que el robot debe seguir la línea contando cubos, y frenar al pasar el segundo cubo, esto nos lleva a escribir el siguiente pseudocódigo con la estructura **mientras** (while) en una combinación muy sencilla:

```
estado := no_detectando_cubo
contador := 0
mientras contador < 2:
      seguir_línea
      contar_cubos
```

Aplicando entonces el principio de divide y vencerás, y teniendo las soluciones de **seguir_linea** y **contar_cubos** obtenemos el algoritmo de la solución del desafío de manera muy fácil. En principio se desearía que los dos subprogramas se ejecuten en paralelo, es decir que siga la línea al mismo tiempo que cuenta cubos, pero TurtleBots no lo permite, otros lenguajes de programación como Java o C sí lo permiten. Por lo tanto vamos a apelar a la velocidad de ejecución de la computadora, esperando que dentro del bucle **mientras** se ejecuten

rápido una y otra vez los subprogramas mencionados. Al ejecutarse rápido se simula ese paralelismo, dado que la computadora ejecuta las órdenes a una velocidad tan alta que se torna imperceptible para la percepción humana.
Las variables **estado** y **contador** se inician en el programa principal, y se modifica su valor dentro del subprograma de **contar_cubos**, de lo contrario la estructura **mientras** nunca terminaría, o los estados serían inconsistentes.

6.2.4.2 Definiendo acciones en TurtleBots

El lenguaje TurtleBots permite definir acciones, es decir permite agrupar bloques para que se ejecuten cuando sea necesario. Esto se hace para dejar el código más entendible y fácil de corregir, y para no repetir instrucciones innecesariamente. Podemos entonces, definir una acción por cada subprograma a utilizar.
¿Cómo podemos definir acciones? Lo primero que hay que hacer es dirigirse a la paleta de variables y seleccionar el siguiente bloque encuadrado en azul:

Figura 6.2.4.2.A: Bloque de acción.

Dicho bloque lo arrastramos al área celeste y cambiamos la palabra acción por el nombre que se desee. Dentro del mismo colocamos los bloques de la acción en cuestión.

Veamos la definición de la acción **contar_cubos**, recordando el ejemplo de que el sensor de distancia está en el puerto **6** y el umbral hallado fue **35400**:

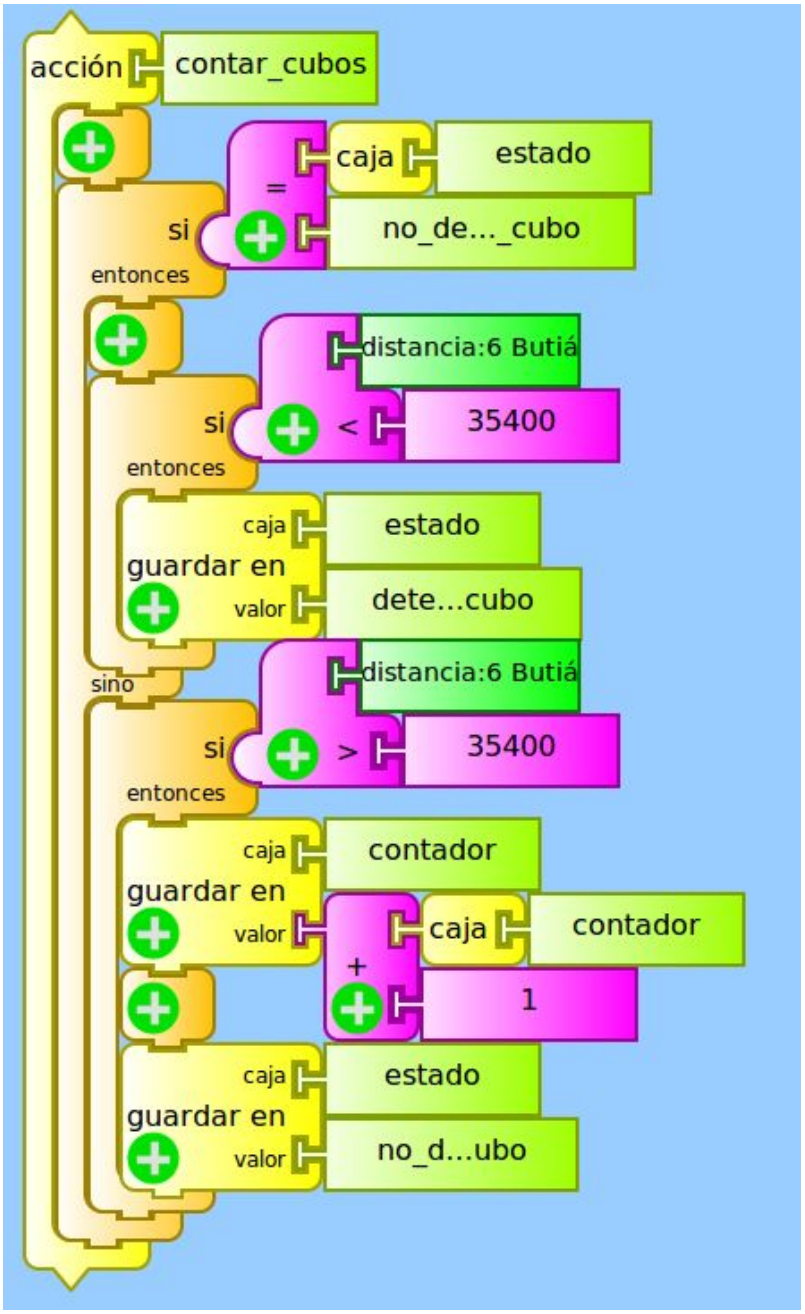


Figura 6.2.4.2.B: Acción de contar cubos.

Podemos ver que está igual que la solución presentada, con las siguientes diferencias:

- Se quitó la inicialización de las variables **estado** y **contador**, ya que se hacen afuera de esta acción, en el programa principal.
- Se quitó el bucle **por siempre**, de lo contrario cuando el programa principal ejecute esta acción, nunca saldría de la misma.

Ahora vamos a ver la acción de **seguir_linea**, que toma las suposiciones presentadas en el **Manual de uso básico de TurtleBots y Butiá 2.0** [12], en las que:

- El sensor colocado a la **izquierda** está en el **puerto 5**, y al hallar su umbral se obtuvo el valor **17500**.
- El sensor colocado a la **derecha** está en el puerto **2**, y al hallar su umbral se obtuvo el valor **22000**.

Por lo tanto la acción quedaría:

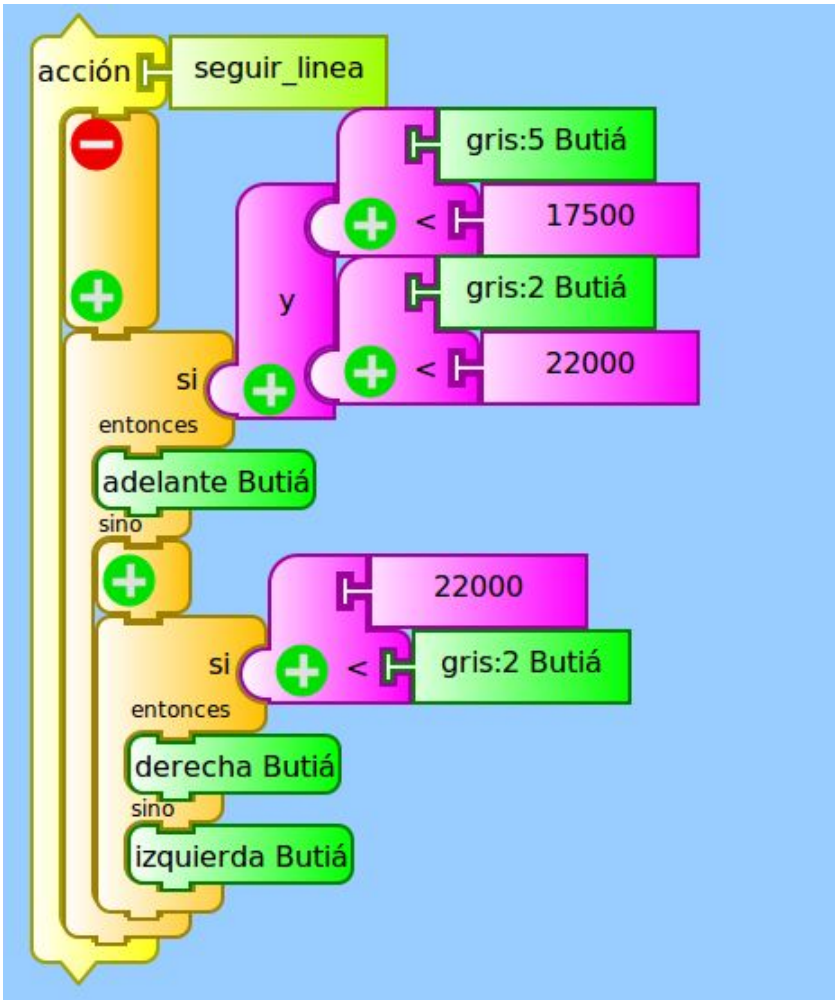


Figura 6.2.4.2.C: Acción de seguir línea.

Notar que al igual que en la acción anterior, se quitó el bucle **por siempre** para no cometer el mismo error.
Finalmente tenemos dos acciones que vamos a combinar en el programa principal para resolver el desafío. ¿Cómo usamos dichas acciones definidas en TurtleBots? Nos dirigimos a la paleta de variables, y luego de crear las acciones mencionadas, aparecerá en dicha paleta los

bloques enmarcados en azul correspondiente a las invocaciones de acciones, como podemos apreciar:

Figura 6.2.4.2.D: Acciones definidas en paleta de variables.

Si las arrastramos al área celeste, podremos utilizarlas cuando la solución la requiera. Entonces veamos cómo queda el programa principal:

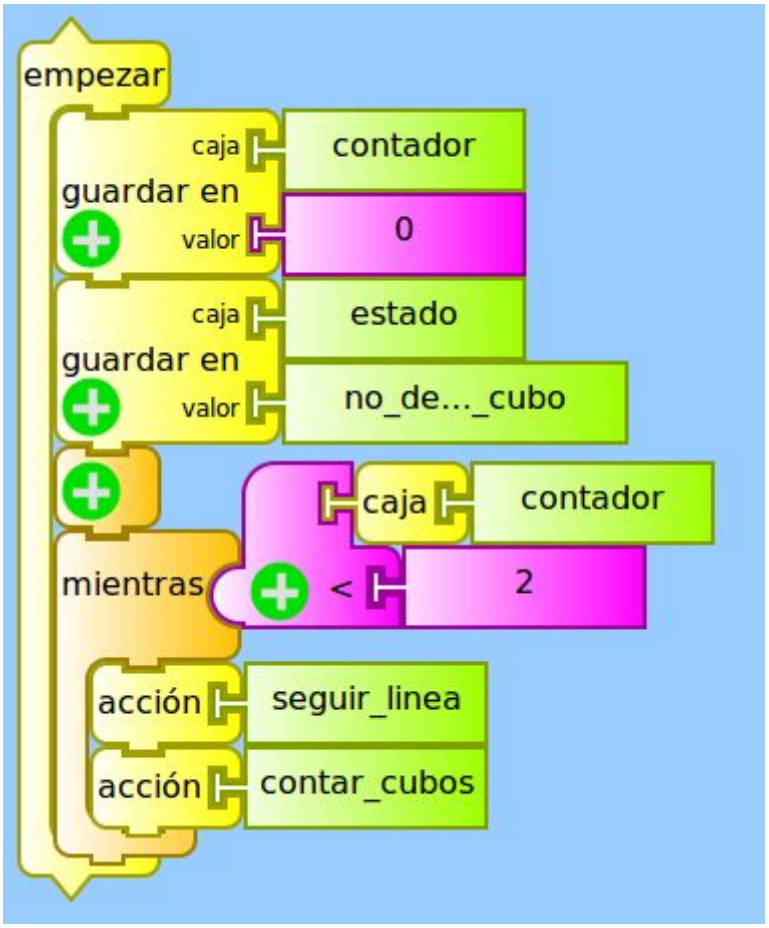


Figura 6.2.4.2.E: Programa principal que resuelve el desafío.

El programa que soluciona el desafío sería entonces un archivo **.tb** con este programa principal, y las dos acciones presentadas. Notar que que podríamos no definir ninguna acción y colocar directamente todos los bloques de **seguir_linea** y **contar_cubos** dentro del bloque **mientras**. Dicho programa funciona pero su codificación es demasiado grande, difícil de mantener y encontrar errores. Queda a cargo del lector probar este programa con un robot Butiá 2.0, hallando los umbrales correspondientes para los sensores utilizados en las condiciones ambiente correspondientes.

8 Bibliografía

Aguirre, Andrés. da Rosa, Sylvia. 2017. *Studying novice learners knowledge about program execution*. En: XLIII Latin American Computer Conference (CLEI), August, 2017.

Aguirre, Andrés. Michetti, Bruno. 2019. *Manual de uso básico de TutleBots y robot Butiá 2.0*. Colección Alfabetización Digital y Proyectos Educativos de Centros MEC. Ediciones IMPO. Montevideo 2019.

Benavides, Facundo. Otegui, Ximena. Aguirre, Andrés. Andrade Federico. 2013. *Robótica educativa en Uruguay: de la mano del robot Butiá*. En: XV Congreso Internacional de Informática en la Educación. Habana. Cuba. 2013.

Castorina, A. (2001). Piaget, las ciencias y la dialéctica. Entrevista a Rolando García. *Herramienta*, *6*(19), 157-182.

da Rosa, S. 2015. *The construction of knowledge of basic algorithms and data structures by novice learners*. En: Proceedings of the 26th Annual Psychology of Programming Interest Group Workshop, Bournemouth, UK, 2015.

da Rosa, Sylvia. Aguirre, Andrés. 2018. *Students teach a computer how to play a game*. En: 11th International Conference on Informatics in Schools: Situation, Evolution, and Perspectives, ISSEP 2018, St. Petersburg, Russia, October 10-12, 2018, Proceedings.

Instituto de Computación. 2013. *Desafío Escolar v0.1. Delivery-bot Junior.* En: Campeonato Uruguayo de Robótica. sumo.uy. Instituto de Computación, Facultad de Ingeniería, Universidad de la República. https://www.fing.edu.uy/inco/eventos/sumo.uy/Documentos/Desafío Escolar 2013 v0.1.pdf, visitada en noviembre del 2018.

Otegui, Ximena. Giachino, Martín. Aguirre, Andrés. Guerra, Agustín. Andrade, Federico. Margenat, Pablo. 2015. Formación de formadores en robótica educativa con Butiá: http://udelar.edu.uy/eduper/wp-content/uploads/sites/29/2015/10/Formaci%C3%B3n-de-formadores-en-rob%C3%B3tica-educativa-con-Buti

%C3%A1.pdf, Jornada sobre Modalidades de Trabajo en Educación Permanente. Visitada en julio del 2019.

Papert, Seymour. 1980. *Mindstorms: children, Computers, and Powerful Ideas*. Basic Books. Inc. Publishers.

Papert, Seymour. 2002. Article for the Bangor Daily News (Bangor, Maine). En: http://www.papert.org/articles/HardFun.html, visitada en abril del 2019.

Piaget, J. 1977. *Genetic Epistemology, a series of lectures delivered by Piaget at Columbia University*, translated by Eleanor Duckworth. Columbia University Press, 1977.

Russell, S. J. Norving, P. 2004. *Inteligencia Artificial, Un Enfoque Moderno*. Segunda edición. Pearson Educación. Upper Saddle River, NJ.

Trinidad, Guzmán. Bender, Walter. Aguirre, Andrés. Aguiar, Alan. Benavides, Facundo. Andrade, Federico. *Sensores Tortuga 2.0: Cómo el hardware y software abiertos pueden empoderar a las comunidades de aprendizaje*. En: RED: Revista de Educación a Distancia, ISSN-e 1578-7680, Nº. 46, 2015 (Ejemplar dedicado a: Pensamiento computacional y competencias para la codificación), 39 págs.