Un entorno visual, dinámico y flexible para evaluar el desempeño de agentes inteligentes

por

Sergio Gastón Burdisso

Tesis de Licenciatura en Ciencias de la Computación
Facultad de Ciencias Físico, Matemáticas y Naturales
Universidad Nacional de San Luis
San Luis, Argentina
2015

Guillermo Aguirre, Director
Marcelo Errecalde, Co-Director

"La ciencia es la expresión de una necesidad inherente al ser humano y, en todo caso, está ligada a la función superior de su naturaleza inteligente: la capacidad de crear"

—Dr. René Gerónimo Favaloro

A mi madre —por mis alas;

A mi padre —por mis pies.

AGRADECIMIENTOS

Pocas veces en la vida uno tiene la oportunidad de agradecer y devolver un poco de lo mucho que le han dado en la vida. Por este motivo, he decidido destinar este pequeño espacio para ello.

Este humilde trabajo está dedicado a mis padres, uno de los pilares fundamentales en mi vida. Su tenacidad y lucha insaciable han hecho de ellos un gran ejemplo a seguir; personas humildes, trabajadoras, con el tesón heredado de esos abuelos italianos, inmigrantes que vinieron a estas tierras a construir un nuevo futuro en base al trabajo y el sacrificio diario. A ustedes les digo: gracias, muchas gracias de corazón y, como bien lo dicen, "¡a no aflojarle la cola a la vaca!".

También agradezco a todos mis formadores. A lo largo de mi vida he tenido la suerte de conocer un gran número de excelentes personas que han contribuido a mi formación profesional. Ellas no sólo se han limitado a brindarme sus conocimientos, sino también a formarme como persona, y en muchos casos hasta entablando una relación de amistad, del tipo padre e hijo con sus alumnos. A todos ustedes, muchas gracias por todo lo que brindan y han brindado, no sólo a mí, sino a la sociedad entera, ya que la educación es uno de los pilares fundamentales de la misma. En particular me gustaría agradecer a Luis Savid, Cecilia Brizuela y a todo el personal del trayecto técnico profesional del colegio N^o 20 de Villa Mercedes. De igual modo agradezco a todos los profesores de la Universidad Nacional de San Luis, quienes diariamente realizan su labor con mucho empeño y pasión, en especial a mis asesores Guillermo Aguirre y Marcelo Errecalde.

Agradezco a todos mis amigos, familiares y seres queridos, ¿Qué cosa más grande que tener a alguien con quien uno se atreva a hablar como consigo mismo? Son muchas las personas especiales a las que me gustaría agradecerles por su amistad, apoyo, ánimo y compañía en las diferentes etapas de mi vida. Algunas están aquí conmigo y otras en mis recuerdos, en el corazón. Especialmente a mis hermanos Mariano (Titín), Carolina (Carito), Joaquín y Juan Eduardo Burdisso; a mi novia Denise Varela; y a mis amigos, Maximiliano Agüero, Daniel Godoy, Alejandro Garcia, Roberto Palma, Gastón Aguilera, Alejandro Fantini, Javier Quevedo, Gastón Simirgiotis, Mario Simirgiotis, Rodrigo y Lautaro Molina. A todos ustedes muchísimas gracias por todo y aprovecho la ocasión para disculparme por todas las horas de ausencia en las que no he podido estar a su lado a causa de mi formación profesional.

Agradezco a mi país, en particular a mi universidad, por permitirme estudiar y obtener un título de grado de forma gratuita. Con país hago referencia a todos los ciudadanos que con su trabajo diario, honestidad, y sacrificio hacen que ésto sea posible; también a todos los que entregaron su vida a la patria para que hoy podamos formar parte de un país libre y soberano, con educación pública y gratuita.

Finalmente, agradezco humildemente a todos aquellas personas que si bien ya no están aquí, siguen formando parte de nuestras vidas, enseñándonos con su ejemplo de vida. En especial al Dr. Favaloro quien, tras 15 años de su muerte, sigue formando parte de todos nosotros, enseñándonos con su entrega, dedicación, humildad y ejemplo. Como bien él dijo una vez: “Debe entenderse que todos somos educadores. Cada acto de nuestra vida cotidiana tiene implicancias, a veces significativas. Procuremos entonces enseñar con el ejemplo”.

Eternamente agradecido,

Sergio.

TABLA DE CONTENIDOS

LISTA DE FIGURAS

Figura

LISTA DE APÉNDICES

Apéndice

LISTA DE ABREVIACIONES

IA Inteligencia Artificial

AIMA Artificial Intelligence: A Modern Approach

HTTP Hypertext Transfer Protocol

JS JavaScript

JIT Just-In-Time

JSON JavaScript Object Notation

XML Extensible Markup Language

IETF Internet Engineering Task Force

WebGL Web Graphics Library

GPU Unidad de Procesamiento Gráfico

CSS Cascading Style Sheets

MVC Modelo-Vista-Controlador

AGPL Affero General Public License

CAPÍTULO I

INTRODUCIÓN

Este capítulo sirve como una introducción general donde se realiza la presentación del problema abordado, se consideran los principales antecedentes relacionados con el tema que han sido encontrados en la bibliografía consultada, se enuncian los objetivos perseguidos y finalmente se presenta la organización del informe.

§1.1 Presentación del Problema

Antes de comenzar a tratar el problema central de este trabajo final, es necesario establecer de manera general algunos conceptos comenzando por lo que es un *agente*. La definición que se adopta en este punto del informe es la siguiente:

"un sistema (o entidad) físico o virtual, situado en algún ambiente, que es capaz de actuar de manera autónoma y flexible, cuyo comportamiento está orientado por un conjunto de tendencias que pueden estar dadas por la satisfacción de ciertos objetivos individuales o la optimización de alguna función de satisfacción/supervivencia". [22]
En esta definición se ve claramente la importancia que tiene el *ambiente* cuando se quieren establecer las características de los agentes. De manera general se dice que el ambiente es todo aquello que puede ser percibido por el agente y sobre lo cual también puede actuar.

Más específicamente, para hacer una mejor caracterización del tipo de agente que se considera aquí, además del ambiente, es necesario incluir otros términos como la *medida de performance* y los *actuadores* y *sensores* que posee el agente. Todo esto constituye lo que se denomina *entorno de trabajo* y es habitual nombrarlo con las siglas "P.A.E.S" como abreviatura de *Performance*, *Ambiente*, *Efectores*[1] y *Sensores*. Para introducir un ejemplo de agente muy simple, es habitual recurrir a contextos conocidos como pueden ser los sistemas de control, concretamente los *termostatos*. Los termostatos tienen un sensor, generalmente construído con dos metales con diferente dilatación, para percibir la temperatura de una habitación. Este sensor está inmerso en el ambiente (una habitación) y produce dos señales: una que indica que la temperatura es demasiado baja y otra que indica que la temperatura es la adecuada. La forma de actuar que tiene el termostato es elevar la temperatura de la habitación y dejar de hacerlo; de esa manera cumple con su tarea: mantener una temperatura confortable en la habitación. Para el caso del termostato, el entorno de trabajo está constituído por:

Performance	Ambiente	Efectores	Sensor
confort	habitación	elevar temperatura y dejar de elevar temperatura	bi-metal

Existen muchos otros ejemplos, como ser: un sistema de diagnostico médico o un robot para la selección de componentes; en estos ejemplos se asocian diferentes elementos básicos de PAES con diferentes clases de agentes. Existen también muchos ejemplos de agentes que son programas de computadoras, los cuales operan en un entorno totalmente artificial donde es posible que las entradas provengan de un mouse o de un teclado y que la salida solamente se pueda percibir en una pantalla. En estos

[1]En esta informe los términos "efectores" y "actuadores" se usarán de forma indistinta.

casos, vale la pregunta: "¿No estamos en un ambiente real, verdad?". A los fines prácticos no importa si el medio es "real" o "artificial", sino la complejidad de la relación entre el comportamiento del agente, la secuencia de percepción generada por el medio y la medida de performance. Existen algunos *agentes de software* que habitan ambientes muy ricos y prácticamente ilimitados. Este es el caso de un robot diseñado para revisar fuentes de información en Internet y para que muestre aquellas que sean interesantes a sus clientes. Para lograrlo debe poseer alguna capacidad de interpretación de lenguaje natural, tendrá que aprender qué es lo que le interesa a cada cliente y tendrá que ser capaz de cambiar dinámicamente sus planes cuando se interrumpa la conexión con una fuente de información o cuando aparezca una nueva. Las variantes posibles de los entornos han obligado a considerar una cantidad limitada de categorías o *dimensiones* de los entornos.

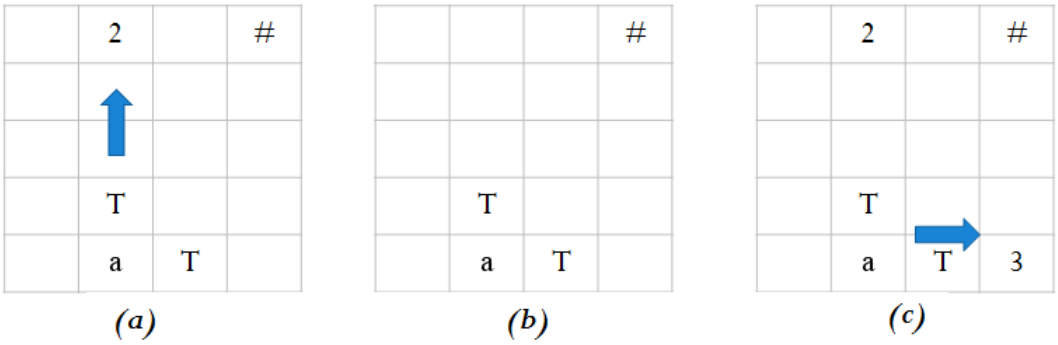
Al considerar ambientes complejos se hace necesario distinguir el problema a resolver del escenario propiamente dicho. Los *campos de prueba* (o *test beds* en inglés) son ambientes desafiantes en los cuales los programas de Inteligencia Artificial (IA) como los agentes, pueden ser estudiados. Los sistemas de IA están pensados para ser aplicados en ambientes extremadamente complejos y los campos de pruebas sirven como versiones simuladas, simplificadas de estos ambientes, en los que el usuario puede establecer ciertos aspectos particulares del mismo, pero algunos otros aspectos se permite que cambien aleatoriamente [39]. El proceso de experimentación consiste en que el investigador pueda variar las características del campo de pruebas o bien el código del propio programa y entonces proceder a medir los efectos que estos cambios produjeron en el desempeño del sistema.

Este trabajo final tiene por objetivo construir una plataforma que permita crear *entornos de trabajo* en los cuales el usuario pueda establecer, mediante parámetros, los valores para determinadas dimensiones del ambiente y evaluar el comportamiento del agente en ese ambiente. Esta plataforma se compone, en parte, por un campo de

pruebas creado a partir de numerosas extensiones y mejoras del Tileworld, descrito a continuación.

§1.2 Antecedentes

El *Tileworld* (en castellano *mundo de baldosas*) [56] se propuso inicialmente como un ambiente experimental para la evaluación de la arquitectura de los agentes. Consiste en un entorno simulado en forma de grilla de dos dimensiones, en el cual existen agentes, huecos, baldosas y obstáculos [49]. Un agente (o robot, como se lo suele denominar aquí) puede moverse en cuatro direcciones: arriba, abajo, derecha e izquierda y si está junto a una baldosa la puede empujar. Un obstáculo es un grupo inamovible de celdas de la grilla, las cuales no pueden ser atravesadas por los agentes. Los huecos deben ser llenados con baldosas por los agentes, el objetivo es llenar tantos huecos como sea posible. El Tileworld es un ejemplo de entorno dinámico; inicialmente, de manera aleatoria, se encuentra en uno de los estados posibles y luego, dependiendo de una serie de parámetros establecidos por el usuario, va cambiando en el tiempo de manera discreta, ya que los huecos aparecen y desaparecen aleatoriamente. El usuario puede establecer un número de parámetros del Tileworld, incluyendo la frecuencia con que aparecen y desaparecen tanto las baldosas como los obstáculos o bien los huecos; también es posible elegir la manera en que desaparecen los huecos: brusca (completamente) o suave (decremento gradual del tamaño). En el Tileworld los huecos aparecen de manera aleatoria y se mantienen de acuerdo a su expectativa de vida, a menos que desaparezcan por el accionar del usuario. El intervalo entre la aparición de huecos sucesivos es denominado el tiempo de gestación de los huecos. El desempeño de un agente en el Tileworld se mide haciendo correr este banco de pruebas durante un número predeterminado de pasos en el tiempo y midiendo la cantidad de huecos que



a = *agente*, # = *obstáculo*, T = *baldosa*, < dígito > = *hueco*

Figura 1.1: Oportunidades en Tileworld.

el agente llenó de manera exitosa. El desempeño (utilidad u) de un agente en una ejecución particular r se define de la siguiente manera:

$$u(r) = \frac{\textit{número de huecos llenados durante } r}{\textit{número de huecos que aparecieron en } r}$$

Esto brinda una medida de desempeño normalizada en el rango de 0 (el agente no tuvo éxito en llenar ningún hueco) a 1 (el agente tuvo éxito en llenar cada hueco que apareció). El error experimental se elimina realizando numerosas ejecuciones y considerando el desempeño promedio. Más allá de su simplicidad, el Tileworld permite examinar importantes capacidades de los agentes. Quizás la más importante sea la habilidad del agente para reaccionar a cambios en el ambiente y de explotar las oportunidades que surgen. Como un ejemplo supongamos que un agente está llevando una baldosa hacia un hueco (Figura 1.1(a)), cuando de pronto el hueco desaparece (Figura 1.1(b)). En ese momento el objetivo pierde sentido y lo mejor que podría hacer el agente, si notó el cambio, es repensar su objetivo original. Para ilustrar lo que significa reconocer oportunidades, supongamos que en la misma situación original aparece un hueco a la derecha del agente (Figura 1.1(c)). Es más probable que el agente sea capaz de llenar el segundo hueco por la sencilla razón que tiene que empujar la baldosa un solo lugar, en lugar de tres. Si todo lo demás se mantiene constante, la probabilidad de que el hueco permanezca allí cuando llegue el agente, es mayor para el que está a la derecha.

Existen numerosos campos de pruebas conocidos, tales como Phoenix [41, 31], Truckworld [24, 52], Ars Magna [20], o RoboCup (fútbol de robots) [12]. De entre todos ellos se eligió a Tileworld principalmente debido a que los demás están fuertemente ligados a un dominio de aplicación en particular y, en algunos casos, no es tan sencillo adaptarlos a ciertos tipos de entornos de trabajo; posiblemente se deba a que ofrecen simulaciones en dominios muy específicos —Fútbol, lucha contra incendios, transporte, etc. En contraste, resultó ser relativamente fácil diseñar modificaciones de Tileworld para adaptarlo a los diferentes tipos de entornos de trabajo, posiblemente debido a su capacidad de ser simple.

Tanto Tileworld como los mencionados campos de pruebas son ejemplos de algunos de los ambientes para agentes más conocidos; se han escrito numerosos artículos científicos en relación a ellos, pero presentan algunos inconvenientes que dificultan su utilización tanto en investigación como en docencia. Entre estos inconvenientes están: limitaciones inherentes al entorno de trabajo del campo de pruebas, dificultades para portarlos a diferentes plataformas, restricciones sobre el lenguaje de programación de los agentes y una interfaz poco atractiva. Considerando este tipo de inconvenientes es que este trabajo final propone un nuevo entorno de trabajo gráfico, flexible y portable.

§1.3 Objetivos

- General:
  - Crear una plataforma de trabajo visual, portable y flexible para evaluar diversas características de los agentes.
- Particulares:
  - Diseñar distintas clases de entornos para evaluar las habilidades de los

agentes.

- Crear los motores de razonamiento para distintos tipos de agentes.
- Evaluar el desempeño de diferentes clases de agentes en distintos tipos de ambientes.
- Proveer una interfaz gráfica similar a un juego.
- Posibilitar que el motor de razonamiento de los agentes sea codificado en una variedad de lenguajes de programación.
- Dotar de flexibilidad al entorno para ser accedido a través de Internet.
- Incorporar las facilidades necesarias para generar las estadísticas del desempeño de los agentes.
- Adoptar para el informe y el sistema una terminología universalmente aceptada por investigadores y docentes.

§1.4 Organización del Informe

Este informe está compuesto de 5 capítulos incluyendo el actual que es de introducción. El próximo capítulo está dedicado a los agentes y su ambiente. Esta es una parte importante del informe y su principal objetivo es destacar la fuerte relación que existe entre el diseño del agente y su evaluación con el ambiente en el que se desenvuelve. El tercer capítulo se concentra en los campos de prueba, se consideran las características generales y deseables de los mismos como herramientas en la experimentación controlada; se realiza un análisis profundo de Tileworld, destacando sus virtudes y defectos, y exponiendo las principales razones que fundamentan su uso como un punto de partida en la creación del campo de prueba de la plataforma. El cuarto capítulo

está dedicado a la descripción de la plataforma desarrollada: el T-World. Los principales aspectos abordados son: a) la interfaz gráfica, los recursos empleados en su construcción; b) el acceso web a la aplicación junto con los elementos tecnológicos involucrados; c) el *servidor proxy* construido para permitir que el programa agente se pueda codificar en diferentes lenguajes de programación; d) descripción general de T-World, cómo aprovechar las distintas facilidades que presenta y algunos ejemplos de programas agentes desarrollados como verificación del funcionamiento correcto del sistema; e) generalización de los resultados obtenidos.

El capítulo final contiene las conclusiones más importantes de este trabajo final, la consideración de las principales dificultades que se debieron superar durante el desarrollo del mismo y también algunas ideas de cómo se puede continuar investigando a partir de los logros alcanzados hasta el momento.

□

CAPÍTULO II

LOS AGENTES Y SU AMBIENTE

En este capítulo se presentan diversos conceptos introductorios vinculados al libro *Artificial Intelligence: A Modern Approach (AIMA)*[1] [59]. Esto se debe a que AIMA es el marco conceptual del trabajo realizado. Así, se comienza presentando la terminología utilizada en el informe: se definen los conceptos de agente, ambiente, percepción, función agente y programa agente. En la sección 2.2 se analiza con más detalle la interacción entre el agente y su ambiente por medio de las percepciones y las acciones. La sección 2.3 trata con el comportamiento racional de los agentes. Luego, en la sección 2.4 se ve cómo este comportamiento racional puede implementarse en un programa agente, los cuales pueden ser clasificados en cuatro tipos diferentes: reflejo simple, reflejo basados en modelo, basados en objetivos, y basados en utilidad. Además, en la sección 2.5 se categorizan los distintos tipos de entornos de trabajo a los que un agente puede enfrentarse. Finalmente, en la sección 2.6 se expone cómo se pueden evaluar y comparar, en base a su comportamiento, diferentes agentes.

[1] Libro de referencia obligado en cualquier curso de Inteligencia Artificial. Utilizado en más de 100 países y más de 1200 universidades, incluidas universidades como Berkeley y MIT.

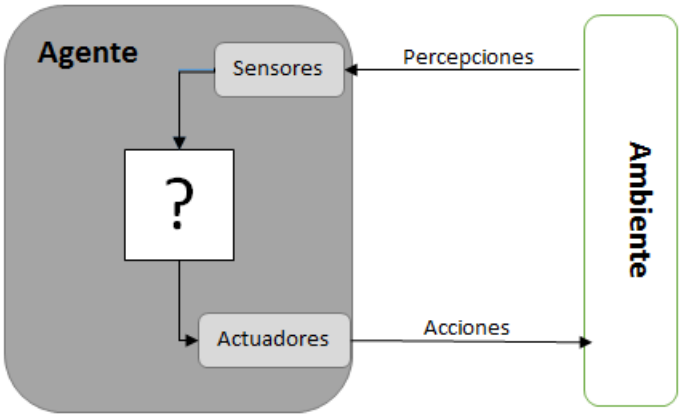


Figura 2.1: Los agentes interactúan con el ambiente por medio de sus sensores y actuadores

§2.1 Agentes Inteligentes Artificiales

Si bien en la literatura existen muchas definiciones para el término agente, como la introducida al comienzo en §1.1, en este capítulo se utilizará la visión de agente dada en AIMA[59]. Así, se comienza definiendo:

Definición 2.1. Un *agente* es todo aquello que se pueda ver como una entidad que percibe su *ambiente* a través de *sensores* y actúa sobre ese ambiente a través de *actuadores*.

Esta idea se ilustra en la Figura 2.1. A modo de ejemplo, un agente humano tiene ojos, oídos, y otros órganos como sensores y manos, piernas, cuerdas vocales y otros como actuadores. Un agente robótico puede tener cámaras y visores de rango infrarrojo como sensores y numerosos motores como actuadores. Un agente de software puede recibir caracteres desde un teclado, contenidos de archivos, y paquetes de red como sensores de entrada y actuar en el ambiente mostrando resultados en la pantalla, escribiendo archivos, y enviando paquetes de red.

Se usa el término *percepción* para referirse a la entrada perceptual del agente en cualquier instante dado. Una *secuencia de percepciones* del agente es el historial

completo de todo lo que el agente haya percibido.

Más formalmente, se dice que el comportamiento de un agente está descripto por una *función agente* que mapea cualquier secuencia de percepciones dada a una acción. Esta función es, por supuesto, una caracterización externa del agente. Internamente, para un agente artificial esta función estará implementada por un *programa agente*. Es importante diferenciar estas dos ideas; la función agente es una descripción matemática; el programa agente es una implementación en concreto, corriendo dentro de algún sistema físico.

Antes de continuar con los aspectos más complejos de los agentes y sus entornos de trabajo, es necesario reforzar las nociones de percepción y acción como interfaz entre el agente y su ambiente, así como también resaltar la fuerte relación que existe entre un agente y su ambiente.

§2.2 Percepción y Acciones de los Agentes

Como se ha mencionado previamente, los agentes perciben el ambiente en el que se encuentran por medio de *sensores*. En la práctica, lo que realmente constituye una *percepción* para un agente varía ampliamente según la tarea particular que éste tenga que realizar. Así, por ejemplo, si se intenta diseñar un agente para categorizar mails entre spam y no spam, la percepción del agente posiblemente esté formada por todos los datos necesarios para poder categorizar un e-mail dado —el correo electrónico emisor, el título del mensaje y el cuerpo del mismo. Mientras que un agente que se encargue de analizar imágenes satelitales tendrá como percepción un arreglo de píxeles de color.

Formalmente, se podría describir a los sensores como una función que recibe toda la información relevante desde el ambiente y aplica una transformación sobre la mis-

ma. Sea E el conjunto de todos los estados en los que el ambiente puede estar, sea P el conjunto de percepciones que el agente puede percibir, luego los sensores pueden describirse matemáticamente como una función $F : E \to P$ que haga corresponder cada estado $e \in E$ a una percepción $p \in P$. Si F es la función identidad se dice que el entorno de trabajo del agente es *completamente observable*, ya que el agente tiene acceso al estado completo del ambiente, es decir, a todos los aspectos que son relevantes para la elección de una acción. Si la función F introduce una distorsión y/o pérdida de información[2] en los elementos $e \in E$, entonces ciertos aspectos que son relevantes para la toma de decisiones del agente simplemente no estarán disponibles en cada momento. Para solucionar este inconveniente el agente deberá incorporar, por ejemplo, algún tipo de memoria que le permita inferir el estado del ambiente actual en base a lo percibido en el momento y el contenido de su memoria que, de cierta forma, sintetiza parte de la información no disponible en la percepción. Esto es una muestra simple de que el ambiente y sus respectivos agentes están fuertemente acoplados; de hecho un "agente" sin un ambiente en el cual actuar y percibir, según definición 2.1, no sería un agente. La interfaz entre el agente y su ambiente son las percepciones y las acciones, así la cantidad de información provista por las percepciones tiene un impacto directo en el comportamiento del agente y en las técnicas que serán utilizadas por el diseñador para construir su agente.

Algo similar sucede con las acciones; el conjunto de acciones disponibles para el agente conforman la interfaz por medio de la cual el agente actúa y realiza cambios sobre el ambiente. Conceptualmente las acciones son pasadas a los actuadores, quienes finalmente las ejecutan y producen los cambios en el ambiente. En ciertos tipos de entornos de trabajo, los actuadores no siempre producen el resultado esperado, esto genera incertidumbre y hace que la tarea a resolver sea mucho más difícil de tratar y

[2]Notar que este es el caso habitual, ya que rara vez los agentes pueden percibir el ambiente en su totalidad. Por ejemplo, pensar en los agentes humanos que, entre otras cosas, poseen un campo visual y auditivo muy limitado.

que las técnicas que se tengan que utilizar para diseñar agentes sean más complejas —por ejemplo, se hace necesario cuantificar la incertidumbre de las acciones por medio de probabilidades. El conjunto de acciones disponibles tiene una influencia directa en el comportamiento del agente y marca límites entre lo que un agente puede y no puede realizar en un ambiente particular, esto es, marca límites con respecto a qué tan bien puede un agente comportarse en un ambiente[3].

§2.3 El Comportamiento de los Agentes

Previamente se mencionó que el comportamiento de un agente está descripto por la correspondencia de secuencias de percepciones a acciones. En esta sección se trata el siguiente interrogante: ¿Cuál es la forma correcta de hacer corresponder esas secuencias de percepciones con las acciones? En otras palabras ¿Qué hace que un agente se comporte de forma adecuada?

Se dice que un agente es *racional* si hace lo *correcto* —conceptualmente hablando, cada correspondencia entre secuencia de percepciones y acciones se realiza correctamente. Esta idea informal del agente "haciendo lo correcto" se refiere a considerar las consecuencias del comportamiento del agente. Cuando un agente es puesto en un ambiente, éste genera una secuencia de acciones de acuerdo a las percepciones que recibe. Esta secuencia de acciones causa que el ambiente pase por una sucesión de estados. Si la secuencia es la deseada, entonces el agente se habrá comportado correctamente. Esta noción de deseabilidad es capturada por una *medida de performance* que evalúa cualquier secuencia de estados del ambiente. Obviamente, no hay una me-

[3]Estrictamente hablando, los limites están impuestos por los actuadores, ya que un agente podría tener un conjunto de acciones muy ricas, pero si sus actuadores sólo pueden llevar a cabo un subconjunto de esas acciones, sólo éstas serán las que efectivamente produzcan cambios reales sobre el ambiente.

dida de performance fija para todas las tareas y agentes; típicamente, el diseñador creará una medida apropiada para cada circunstancia.

Se considera que un agente es inteligente, es decir, se comporta de forma correcta si es racional. Que un agente sea racional depende de cuatro factores:

- La *medida de performance* que define el criterio de éxito.
- El *conocimiento previo* del agente sobre el ambiente.
- Las *acciones* que el agente puede realizar.
- La *secuencia de percepciones* del agente hasta la fecha.

En base a estos 4 factores se define a un agente racional de la siguiente manera:

Definición 2.2. *Por cada secuencia de percepciones posibles, un agente racional debe seleccionar una acción que se espera que maximice su medida de performance, dada la evidencia provista por la secuencia de percepciones y cualquier conocimiento previo que el agente tenga.*

En la práctica, este comportamiento racional en base a la maximización de la medida de performance deberá ser implementado por medio de un programa agente, tema que se trata en la siguiente sección.

§2.4 Programa Agente y Tipos de Agentes

Hasta ahora se ha hablado de agentes por medio de la descripción de su comportamiento —la acción que es realizada para cualquier secuencia dada de percepciones. Ahora se hablará de cómo funciona el interior. Una de las tareas en la Inteligencia Artificial es el de diseñar un programa agente que implemente la función agente —que

haga corresponder percepciones a acciones. Se asume que este programa ejecutará sobre algún tipo de dispositivo de computación físico o virtual, lo que se denomina como la *arquitectura*. Obviamente, el programa que se selecciona tiene que ser el apropiado para la arquitectura. La *arquitectura* puede ser, por ejemplo, simplemente una computadora común, un automóvil robótico con muchas computadoras a bordo, cámaras y otros sensores, ó incluso una pieza de software sobre la cual un agente de software podría ejecutar. Por lo general, la arquitectura dispone las percepciones al programa desde los sensores, corre el programa, y entrega las acciones elegidas por el programa a los actuadores a medida que éstas son generadas.

Si bien hay muchas formas de definir un esqueleto para los programas agentes, se utilizará la forma descrita en AIMA, en donde los programas agentes toman la percepción actual como entrada desde los sensores y devuelven una acción a los actuadores. En lugar de tomar como entrada el historial completo de percepciones, el programa agente toma sólo la percepción actual como entrada porque es lo único a lo que se tiene acceso desde el ambiente; Si las acciones del agente dependen de la secuencia completa de percepciones, el agente tendrá que recordar las percepciones previas. Uno de los desafíos en la Inteligencia Artificial es descubrir cómo escribir programas que, en la medida de lo posible, produzcan un comportamiento racional desde un algoritmo.

En el resto de esta sección se describen cuatro tipos de programas agente que incorporan los principios que existen detrás de casi todos los sistemas inteligentes:

- Agentes reflejo simple;
- Agentes reflejo basados en modelo;
- Agentes basados en objetivos; y
- Agentes basados en utilidad.

Cada tipo de programa agente combina componentes específicos de maneras particulares para generar acciones. A cualquiera de estos cuatro tipos de programas agente se los puede convertir en agentes que aprenden, los cuales pueden mejorar el rendimiento de sus componentes, en base a la experiencia, de manera que genere un mejor comportamiento en el futuro.

§2.4.1 Agentes Reflejo Simple

El tipo más simple de programas agente es el agente reflejo simple. Estos agentes seleccionan acciones en base a la percepción actual, ignorando el resto del historial de percepción, haciendo que el agente tenga un comportamiento completamente reactivo. El comportamiento reflejo-simple ocurre incluso en ambientes complejos. Por ejemplo, en la conducción autónoma de un automóvil, si el auto de adelante frena y su luz de freno se enciende, algún procesamiento se hace sobre la entrada visual para establecer la condición "el auto de adelante está frenando". Luego, esto dispara alguna conexión establecida dentro del programa agente para la acción "comenzar a frenar". A tal conexión se le denomina como regla condición-acción, escrita como *Si auto-de-adelante-está-frenando entonces comenzar-a-frenar.* Notar que la descripción en términos de "reglas" es puramente conceptual; las implementaciones reales pueden ser tan simples como un conjunto de compuertas lógicas combinadas como un circuito.

Los agentes reflejo simple tienen la admirable propiedad de ser simples, pero resultan tener una inteligencia limitada: funcionan *únicamente si la decisión actual puede ser tomada sólo en base a la percepción actual* —lo cual solamente es posible si la percepción brinda toda la información relevante para la elección de una acción.

§2.4.2 Agentes Reflejo Basados en Modelo

La forma más efectiva de manejar el hecho de que algunos aspectos relevantes no sean percibidos, es que el agente lleve un registro de las partes del mundo que él no puede ver en un determinado momento. Esto es, el agente debe mantener algún tipo de estado interno que dependa del historial de percepción y por lo tanto refleje al menos algunos de los aspectos no observables del estado del mundo actual. La actualización de la información de este estado interno a medida que pasa el tiempo requiere la codificación de dos tipos de conocimiento en el programa agente:

- Información sobre cómo evoluciona el mundo de forma independiente al agente.
- Información sobre cómo las propias acciones del agente afectan al mundo.

Esta información sobre "cómo funciona el mundo" —ya sea que esté implementada en simples circuitos lógicos o en teorías científicas completas—es lo que se conoce como un *modelo del mundo*. A un agente que usa tal modelo se lo denomina *agente basado en modelo*.

Sin importar el tipo de representación utilizada, raramente es posible que el agente determine exactamente el estado actual del mundo en un ambiente en el que no se pueda percibir el estado completo del ambiente. En su lugar, el estado interno del agente representa la "mejor suposición" del agente (o a veces mejores suposiciones). Por lo tanto, la incertidumbre sobre el estado actual muchas veces es inevitable, pero aun así el agente tiene que tomar una decisión.

§2.4.3 Agentes Basados en Objetivos

El hecho de conocer el estado actual del ambiente no siempre es suficiente para decidir qué hacer. Por ejemplo, en un cruce de calles, un vehículo autónomo puede doblar a

la izquierda, derecha o seguir derecho. La decisión correcta depende del lugar al que quiere llegar el vehículo. En otras palabras, además de una descripción del estado actual, el agente necesita algún tipo de información sobre el *objetivo* que describe las situaciones que son deseables —por ejemplo, llegar a un determinado lugar. El programa agente puede combinar el *objetivo* con el *modelo* (la misma información que fue usada en los agentes reflejo basados en modelo) para elegir las acciones que alcancen dicho objetivo.

La búsqueda y la planificación (planning) son subcampos de la IA dedicados a encontrar secuencias de acciones que logren los objetivos de los agentes. Notar que este tipo de toma de decisiones implican la consideración del *futuro*. El agente reflejo frena cuando ve la luz de freno. Un agente basado en objetivo, en principio, podría razonar que si el auto de adelante tiene las luces de freno encendidas, va a disminuir la velocidad. Dada la forma con la que usualmente el mundo evoluciona, la única acción que logrará el objetivo de "no chocar otros autos" es frenar.

Aunque el agente basado en objetivo parece menos eficiente, es más flexible porque el conocimiento que apoya sus decisiones está representado explícitamente y puede ser modificado sobre la marcha durante el tiempo de vida del agente —sin necesidad de reescribir partes del código del programa agente.

§2.4.4 Agentes Basados en Utilidad

Los objetivos por sí solos no son suficientes para generar un comportamiento de alta calidad en la mayoría de los ambientes. Los objetivos solamente brindan una distinción cruda y binaria entre estados que cumplen los objetivos y los que no. Una forma de evaluación más general debe permitir una comparación entre diferentes estados del mundo de acuerdo a exactamente *qué tan deseable* es cada uno de ellos para el agente. Para expresar formalmente esta noción de deseabilidad se utiliza una *función de*

utilidad. La función de utilidad de un agente es esencialmente una internalización de la medida de performance. Si la función de utilidad interna y la medida de performance externa concuerdan, entonces un agente que elige acciones para maximizar su utilidad será racional de acuerdo a la medida de performance externa.

Como los agentes basados en objetivos, los agentes basados en utilidad tienen muchas ventajas en términos de flexibilidad y aprendizaje. Además, hay dos casos particulares en los que los objetivos son inadecuados pero en que los agentes basados en utilidad aún pueden tomar decisiones racionales:

- cuando hay objetivos en conflicto y solamente algunos de ellos pueden ser alcanzados.
- cuando hay muchos objetivos a los cuales el agente puede apuntar y ninguno de estos puede alcanzarse con certeza

La visibilidad parcial y la aleatoriedad están omnipresentes en el mundo real, así que por lo tanto, también lo es la toma de decisiones bajo incertidumbre. Técnicamente hablando, un agente basado en utilidad elige la acción que maximiza la utilidad esperada del resultado de la acción [4].

§2.5 Tipos de Entornos de Trabajo

En las secciones anteriores se han tratado, entre otros, los conceptos de agente, ambiente, programa agente y comportamiento racional. Esta sección trata con los distintos tipos de *entornos de trabajo*, que en esencia son los "problemas" para los cuales los agentes racionales son la "solución". El rango de *entornos de trabajo* que pueden surgir en la Inteligencia Artificial es obviamente vasto. Se puede, sin embargo,

[4]Esto es, la utilidad a la que el agente espera derivar, en promedio, dadas las probabilidades y utilidades de cada resultado.

identificar un número pequeño de dimensiones bajo las cuales los entornos de trabajo pueden ser categorizados. Estas dimensiones determinan, en gran medida, el diseño apropiado del agente y la aplicabilidad de cada una de las principales familias de técnicas que se utilizan para la implementación de agentes. Estas dimensiones se listan a continuación:

- **Completamente observable vs. Parcialmente observable:** Si los sensores de un agente le dan acceso al estado completo del ambiente en cada instante de tiempo, entonces se dice que el entorno de tarea es completamente observable. Un entorno de tarea es efectivamente *completamente observable* si los sensores detectan todos los aspectos que son relevantes para la elección de una acción; la relevancia, a su vez, depende de la medida de performance. Los entornos de tarea totalmente observables son convenientes ya que el agente no necesita mantener ningún estado interno para hacer un seguimiento del mundo. Un ambiente puede ser parcialmente observable a causa de ruido o sensores imprecisos o porque partes del estado simplemente no son capturadas como datos del sensor.

- **Único agente vs. Multiagente:** la distinción entre entornos de tarea con único agente y multiagente puede parecer lo suficientemente clara. Por ejemplo, un agente resolviendo un crucigrama por él mismo está claramente en un ambiente de tarea de un único agente, mientras que un agente jugando ajedrez está en un ambiente de dos agentes. Existen, sin embargo, algunos inconvenientes sutiles. Primero, si bien se ha descrito cómo una entidad puede ser vista como un agente, no se ha explicado qué entidades necesariamente tienen que ser vistas como agentes. *Un agente tiene que tratar a una entidad como otro agente si el comportamiento de esta entidad se describe mejor como la maximización de una medida de performance cuyo valor depende del comportamiento del agente.* Si la maximización de la medida de performance de un agente produce un impacto negativo en la medida de performance de los otros agentes, entonces el

ambiente de tarea es **competitivo**, en otro caso es **cooperativo**. Los problemas de diseño de agentes en ambientes multiagentes son a menudo diferentes de los ambientes de un único agente; por ejemplo, la comunicación por lo general emerge como un comportamiento racional en ambientes multiagente.

- **Determinístico vs. Estocástico:** Si el próximo estado del ambiente está completamente determinado por el estado actual y la acción ejecutada por el agente, entonces se dice que el entorno de tarea es determinístico; en otro caso, es estocástico. En principio, un agente no tiene que preocuparse por la incertidumbre en un ambiente completamente observable y determinístico. Si un entorno de tarea es parcialmente observable, sin embargo, entonces podría parecer que es estocástico. Se dice que un entorno es incierto si no es completamente observable o no es determinístico.

- **Episodico vs. Secuencial:** En un entorno de tarea episódico, la experiencia del agente está dividida en episodios atómicos. En cada episodio el agente recibe una percepción y luego realiza una única acción. El próximo episodio no depende de la acción tomada en el episodio anterior. En un entorno de tarea secuencial, por otro lado, la decisión actual podría afectar todas las decisiones futuras, es decir, las acciones a corto plazo pueden tener consecuencias a largo plazo. Los entornos de tarea episódicos son mucho más simple que los secuenciales porque el agente no tiene que pensar a futuro.

- **Estático vs. Dinámico:** Si el ambiente cambia mientras un agente está deliberando, entonces se dice que es dinámico para ese agente; de otra forma, es estático. Los entornos de tarea estáticos son simples de trabajar porque el agente no necesita mantenerse observando el mundo mientras está decidiendo sobre una acción, no necesita preocuparse por el paso del tiempo. Los ambientes de tarea dinámicos, por otro lado, están continuamente preguntándole al agente

qué es lo que él desea hacer; si aún no lo ha decidido, eso cuenta como que ha decidido no hacer nada. Si el ambiente en sí mismo no cambia con el paso del tiempo pero la puntuación de performance del agente sí, entonces se dice que el ambiente de tarea es **semidinámico**.

- **Discreto vs. Continuo:** La distinción entre continuo y discreto se aplica al estado del ambiente, a la forma con la que se maneja el tiempo, y a las percepciones y acciones del agente. Por ejemplo, en el entorno de tarea para jugar Ajedrez se tiene un número finito de estados distintos y también tiene un conjunto discreto de percepciones y acciones. La conducción de vehículos es un problema de estados y tiempo continuo.

- **Conocido vs. Desconocido:** Estrictamente hablando, esta distinción no se refiere al ambiente en sí mismo sino al estado de conocimiento del agente (o diseñador) sobre las "leyes de la física" del ambiente. En un ambiente conocido, los resultados (o las probabilidades de los mismo, si el ambiente es estocástico) para todas las acciones son dados. Obviamente, si el ambiente es desconocido, el agente tendrá que aprender cómo funciona para poder tomar buenas decisiones.

§2.6 Evaluación de los Agentes

En la sección 2.3 se habló sobre qué hace que un agente se comporte de forma inteligente. Se dijo que un agente se considera inteligente si es racional. Se desprende de la definición 2.2 (ρ. 14) de agente racional que, cuando se desean evaluar diferentes programas agentes sobre un mismo entorno de tarea, se debe hacer uso de la medida de performance para poder comparar el comportamiento de cada agente y así decidir, por ejemplo, cuál es el más eficaz —aquel que logre una mayor maximización de su

medida de performance. De esta forma, la medida de performance permite cuantificar qué tan bien cada agente se desempeña en un ambiente determinado. Por esa razón, es común que en la investigación científica se obtengan resultados empíricos cuantificando la eficacia de diferentes arquitecturas, técnicas o políticas de diseño de agentes y/o midiendo el impacto que ciertas características de los ambientes producen en el desempeño de los agentes.

Por ejemplo, en el capítulo anterior (§1.2) se mencionó a Tileworld. La medida de performance en este caso está representada por el score (puntuación) acumulado que el agente obtiene al rellenar huecos durante la simulación. Mayor es el score acumulado, mejor es su comportamiento —e.d. más racional. En Tileworld el score acumulado se puede utilizar para calcular la eficacia de un agente, de forma normalizada, dividiéndolo por el score total completo de todos los huecos que aparecieron durante la simulación. Utilizando esta medida normalizada se han realizado numerosos estudios empíricos en base a qué factores influyen en el comportamiento del agente (e.d. afectan la eficacia) y de qué modo —[19], [54], [56], [13] entre otros.

De especial interés es el estudio y la evaluación de agentes sobre los ambientes de tarea dinámicos y estocásticos, ya que en el mundo real el dinamismo y la incertidumbre están omnipresentes. El estudio del comportamiento y ciertas características de los agentes en este tipo de ambientes puede contribuir de forma positiva al desarrollo de sistemas con la suficiente complejidad como los que se pueden presentar en el mundo real.

Para poder realizar este tipo de estudios, a menudo los investigadores hacen uso de la *experimentación controlada* por medio de *campos de pruebas*. T-World además de ser una plataforma web para la investigación y la educación, es en esencia un *campo de pruebas* totalmente compatible con el Tileworld. Esta compatibilidad es producto de que el ambiente tridimensional altamente configurable de T-World fue diseñado para ser compatible con el Tileworld original y sus respectivas variantes[49]. Es por

esto que el siguiente capítulo está dedicado a los campos de prueba en general, y a Tileworld en particular.

□

CAPÍTULO III

CAMPOS DE PRUEBAS

Un gran número de proyectos de investigación de Inteligencia Artificial hacen uso de la *experimentación controlada*, en la cual el investigador varía las características del programa agente o del ambiente y mide los efectos de estas variaciones sobre el desempeño del agente. A su vez, las herramientas de investigación más utilizadas para realizar *experimentación controlada* son las *tareas de referencia* y los *campos de pruebas*. En la siguiente sección se los describe detalladamente, resaltando su importante rol en la experimentación controlada. Luego se definen un conjunto de criterios para considerar a un campo de pruebas útil y atractivo, no solo como una herramienta de experimentación, sino también como una pieza de software. Estos criterios serán posteriormente empleados para analizar campos de pruebas, en particular se analiza en detalle a Tileworld. Finalmente, se exponen las principales razones que fundamentan el uso de Tileworld como un punto de partida en el desarrollo del campo de prueba de T-World.

§3.1 Herramientas para la Experimentación

De forma general, las *tareas de referencia (benchmarks, en Inglés)* son tareas estandarizadas definidas de una forma muy precisa, mientras que los *campos de pruebas*

son ambientes desafiantes en los cuales los agentes pueden ser puestos a prueba y estudiados, típicamente de manera controlada. Estas herramientas sirven al menos a dos propósitos: proporcionar métricas para comparar diferentes sistemas, y ser empleados en investigaciones científicas. El valor científico de las mismas, cuando están bien elaboradas, es su poder para resaltar ciertos aspectos de interés para el experimentador sobre el comportamiento del agente —siempre y cuando se pueda explicar adecuadamente por qué el agente se comporta como lo hace.

Las tareas de referencias son herramientas muy comúnes en ciencias de la computación. Por ejemplo, en el diseño de Unidades Centrales de Procesamientos (CPUs), la multiplicación de matrices es una buena tarea de referencia porque representa una clase más grande de problemas de cálculo. Un buen desempeño en el problema de multiplicación de matrices predice un buen desempeño en la clase más grande de tareas numéricas para las cuales el procesador está siendo diseñado.

Una tarea de referencia clásica para algoritmos de planificación (planning) en IA es la anomalía de Sussman (el problema de los tres bloques)[66]. La anomalía de Sussman ayudó a muchos investigadores a entender cómo trabajaban sus planificadores. Se hizo popular porque al igual que la multiplicación de matrices, *es un representante de una clase importante de problemas*, aquellos que involucran interacciones entre sub-objetivos conjuntivos; y fundamentalmente porque es fácil de describir.

En el contexto del diseño de agentes inteligentes, normalmente los investigadores necesitan entender los principios detrás de un comportamiento en particular y para lograrlo necesitan un modelo formal de la tarea llevada a cabo. Es por esto que, en la práctica científica las tareas de referencia más útiles suelen ser aquellas que pueden entenderse lo suficientemente bien como para poder modelarse detalladamente. Ya que de esta forma, su uso no sólo podría mostrar qué decisiones de diseño llevan a un buen desempeño, sino también por qué lo hacen. Así, *las tareas de referencia idealmente son problemas que pueden ser sometidos a un análisis preciso, a la vez que representan una*

realidad más compleja y sofisticada. Por otro lado, el control experimental que se logra con los *campos de pruebas* también puede ayudar a explicar los principios detrás de un comportamiento en particular. Ya que en la vida real, se pretende que los agentes inteligentes se desempeñen en ambientes enormes y extremadamente complejos (tan complejos que rara vez pueden ser modelados); los campos de pruebas sirven como versiones simplificadas y simuladas de estos ambientes. Usándolos el investigador tiene control sobre las características del ambiente que le son de interés, pudiendo estudiar y medir cómo afectan al comportamiento del agente.

Aunque algunos *campos de prueba* no son más que una interfaz para especificar los parámetros de una *tarea de referencia* junto a instrumentación para medir el desempeño; el campo de prueba brindado por T-World proporciona un ambiente lo suficientemente flexible y rico como para poder diseñar una gran cantidad de tareas para agentes. Esto permite el estudio de numerosos fenómenos, técnicas y/o problemas tales como umbral de aceptabilidad en tiempo real (real-time satisficing), degradación graciosa (graceful degradation) bajo restricción de recursos, planificación de trayectoria y navegación, aprendizaje, mantenimiento, búsqueda, razonamiento bajo incertidumbre, coordinación multiagente, etc.

Para concluir esta sección, es importante mencionar que la dificultad en el uso de ***campos de pruebas*** radica en lo siguiente: *las tareas que se creen utilizándolos, idealmente deben satisfacer el criterio de las* ***tareas de referencia*** *—ser representantes de problemas más grandes e interesantes*. De esta forma los resultados que se obtengan podrán trasladarse, en cierto grado, hasta el problema más grande representado por la tarea.

§3.2 Requerimientos para los Campos de Pruebas

Existen dos tipos diferentes de campos de pruebas, los que brindan un ambiente real para estudiar agentes y los que brindan uno simulado. El campo de pruebas construido como parte de este trabajo final brinda un ambiente simulado. Por este motivo, es necesario estudiar qué elementos hacen que un mundo simulado sea considerado útil y atractivo, en el estudio de agentes inteligentes.

De forma general, se necesita que el ambiente sea desafiante al mismo tiempo que adecuado para un uso en la experimentación. Sin embargo, esta no es una tarea sencilla ya que ambos criterios son propensos a oponerse mutuamente: un dominio desafiante tiende a ser complejo, irregular y desordenado; mientras que un buen dominio para la experimentación controlada tiende a ser limpio, uniforme y controlado. No obstante esto, de acuerdo a la capacidad del ambiente para representar problemas interesantes de investigación y las características de ingeniería de software deseables, se pueden detectar una serie de requerimientos para un campo de prueba.

§3.2.1 Requerimientos de Investigación

En base a los distintos tipos de entornos de tarea introducidos en §2.5 y las principales áreas de investigación dentro de la IA, es posible identificar un conjunto de requerimientos que hacen a un campo de prueba útil en la medida que aumentan la capacidad de su ambiente para representar problemas interesantes de investigación:

- **Eventos exógenos:** En un entorno de trabajo dinámico los eventos externos al agente (no planificados) introducen incertidumbre en el ambiente. Esto hace que para el agente, el proceso de predicción de los efectos reales de sus planes sea considerablemente más difícil [37]. También introduce la necesidad de reaccionar

a eventos no planificados ya que ocurren en tiempo de ejecución [2][23]. El costo del tiempo que el agente utiliza para razonar se vuelve importante en un mundo que permite cambios no planificados: mayor es el tiempo que el agente se toma para planificar, mayor es la probabilidad de que el mundo haya cambiado significativamente entre el tiempo en el que el plan fue generado y el tiempo en el que éste es ejecutado [11][58][15]. Un campo de pruebas debería, por lo tanto, *permitir la ocurrencia de eventos exógenos* —además, éstos deben generar problemas interesantes, tanto en predicción como en ejecución reactiva.

- **Costos del tiempo de razonamiento:** Como se mencionó en el ítem anterior, el mundo cambia con el paso del tiempo, y la construcción de un plan lleva tiempo. Esto lleva a una línea de investigación que se centra en tomar en cuenta el tiempo de razonamiento a la hora de decidir cuándo se planifica y cuándo se actúa [60][10]. Las implicación para un campo de pruebas es que *el paso del tiempo debería ser manejado de una forma realista: el mundo y los agentes deben ser capaces de operar de forma simultánea.*

- **Complejidad del mundo:** Un mundo realista tiene muchas características aunque muchas de éstas puedan ser irrelevantes para un problema en particular. Además, un mundo realista tiene una estructura causal compleja: los cambios en un aspecto del mundo causarán cambios en muchos otros aspectos, incluso aunque la mayoría de éstos pueden ser, nuevamente, irrelevantes para un problema en particular. El razonamiento sobre modelos más realistas del mundo requiere la necesidad de representar y hacer predicciones sobre mecanismos complejos [70], así como también la habilidad de reconocer y poner atención a esos aspectos del mundo relevantes para el problema que se resuelve [16]. Un campo de pruebas para explorar problemas con complejidad realista debería proporcionar una compleja diversidad de características.

- **Sensores y actuadores:** En un mundo real, los entornos de trabajo rara vez son completamente observables. Esto normalmente se debe a que los sensores y los actuadores *no* son perfectos ni libres de costo. En estos casos un agente debe, por lo tanto, incorporar percepciones provenientes de sensores incorrectos y ruidosos en el modelo predictivo del mundo [38] y debe planificar cuándo percibir para mejorar su estado interno del mundo, teniendo en cuenta ambos, el beneficio de la información adquirida al percibir y el costo de adquirirla [14]. Así, un campo de pruebas debería *permitir la existencia de sensores y actuadores defectuosos*, así como también asignarle un costo a éstos. Desde un punto de vista de implementación, se debe hacer una distinción clara entre el agente y el mundo simulado, en donde las capacidades de los sensores (percepciones) y actuadores (acciones) del agente *definan la única interfaz entre ellos.*

- **Medida de éxito:** A los agentes basados en objetivo se les da un estado a alcanzar, y se detienen cuando sus planes alcanzan dicho estado. Pero el logro de un estado objetivo es una medida inadecuada de éxito: no toma en cuenta el costo de lograr el objetivo, ni admite la posibilidad de una satisfacción parcial del mismo [35][36][71]. Idealmente, un campo de pruebas debería permitirle al diseñador plantear problemas que involucren una satisfacción parcial de los estados deseables, obligando al agente a intercambiar los beneficios de lograr el objetivo por el costo de lograrlo. Además, ya que el agente debe ser capaz de actuar bien bajo diversas circunstancias, *el mundo en sí mismo no debería dictar si un curso de acciones fue exitoso o no. El éxito es una propiedad del problema planteado al agente, no una propiedad del mundo en el que éste opera.*

- **Múltiples agentes:** El permitir que múltiples agentes actúen en el mundo introduce nuevos problemas: Cómo coordinar los comportamientos, cómo deberían comunicarse los agentes, cómo los efectos de las acciones simultáneas

difieren de aquellas realizadas de manera serial. Existe una gran área de investigación dentro de la Inteligencia Artificial encargada del estudio de sistemas multi-agente, en donde los sistemas están compuestos por múltiples agentes que interactúan entre sí dentro de un mismo ambiente. Un campo de pruebas para estudiar este tipo de sistemas debería poder *soportar la comunicación y el comportamiento coordinado (en paralelo) entre los agentes que habitan el mundo simulado.*

§3.2.2 Requerimientos de Ingeniería

Si el campo de pruebas se va a utilizar ampliamente para fines de validación, por ejemplo, validando una arquitectura de agente comparándola contra otras o probando diferentes decisiones de diseño; ó si simplemente se quiere que sea más útil para potenciales usuarios, surgen algunas inquietudes de implementación desde un punto de vista de ingeniería. Estas inquietudes tienen que ver principalmente con la interfaz entre el agente y el mundo, ésta tiene que ser clara y bien definida. Además, la necesidad de una experimentación controlada requiere que el investigador sea capaz de caracterizar y sistemáticamente variar aspectos del mundo y medir el rendimiento del agente, ambos de una forma bien definida y objetiva. Todo esto sugiere algunos requerimientos:

- **Una interfaz limpia:** Es importante mantener una distinción clara entre el agente y el mundo en el que éste está operando. La separación natural es a través de las acciones y las percepciones, así que la interfaz debería ser limpia, estar bien definida y documentada. Un diseñador debe ser capaz de determinar fácilmente qué acciones están disponibles para el agente, cómo son ejecutadas las acciones por el campo de pruebas, y cómo la información sobre el mundo es

comunicada hacia el agente —es decir, cómo es percibida por el agente.

- **Un modelo bien definido del tiempo:** Los campos de pruebas tienen que presentar un modelo razonable del paso del tiempo para simular eventos exógenos y acciones simultáneas; y para definir claramente el costo del tiempo de razonamiento y de las acciones. Además tiene que, de alguna manera, ser capaz de comunicar cuánto tiempo de simulación ha transcurrido. Este problema es crucial, tiene un efecto tanto en el agente (ya que razona sobre el costo del tiempo de sus acciones) como en los resultados experimentales —el tiempo probablemente figure en cualquier evaluación del performance de una arquitectura. Cabe aclarar que se requiere de una forma de conciliar la medida de tiempo del campo de prueba con la usada por el agente para que los resultados experimentales tengan sentido.

- **Soporte para la experimentación:** Usar el campo de pruebas para evaluar el comportamiento de los agentes requiere ponerlos a prueba bajo una gran variedad de circunstancias y problemas, esto implica la adecuada configuración del mismo. Además, la experimentación controlada necesita que los problemas y las condiciones ambientales sean variadas *de una manera controlada*. Es por esto que el campo de pruebas debe proporcionar un mecanismo para que el investigador pueda variar el comportamiento del mundo simulado, quizás por medio de parámetros que puedan ser establecidos por el investigador. Sin embargo, proporcionar estos parámetros puede ser complicado: por un lado si hay muy pocos el investigador no será capaz de variar el ambiente lo suficientemente bien como para producir resultados significativos. Si hay demasiados, se vuelve difícil llevar a cabo un número pequeño de experimentos e interpretar los resultados. La forma en la que el campo de pruebas permite la variación de su comportamiento es crucial en el uso del mismo para experimentar con arquitec-

turas de agentes. No obstante eso, el investigador también tiene que ser capaz de monitorear el comportamiento del agente en el mundo simulado, tanto en tiempo real como a través de la recolección de estadísticas.

Además de estos requerimientos de ingeniería relacionados al campo de pruebas como herramienta de investigación, es posible identificar un conjunto de propiedades deseables en relación a su cualidad de ser una pieza software. En la bibliografía consultada se detectaron algunos problemas en la manera con la que típicamente se distribuyen las implementaciones de los campos de prueba. Esto normalmente se realiza por medio de la disposición del código fuente, lo que implica, entre otras cosas, que el experimentador tenga que estar familiarizado con el lenguaje de programación utilizado en el código fuente; deba comprenderlo lo suficientemente bien como para poder acoplar su programa agente al mismo; pueda ser capaz de ejecutarlo, instalando el compilador, los paquetes que se requieran para hacerlo o, en caso de ser un lenguaje interpretado, la máquina virtual o intérprete necesario. Llevando a que muchas veces los investigadores empleen tiempo implementando sus propias versiones del campo de prueba necesitado, lo que a su vez dificulta la comparación relativa (bajo las mismas condiciones de implementación) de resultados experimentales con las otras versiones disponibles del mismo. Esto, sumado al estado del arte y las necesidades informáticas actuales, sugieren las siguientes propiedades deseables:

- **Existencia propia:** Idealmente, el campo de pruebas debería ser una pieza de software *independiente, autocontenida y ejecutable*. De esta forma el experimentador no necesitaría estar familiarizado con los detalles de implementación o el código fuente del mismo para utilizarlo; él simplemente ejecutaría el campo de pruebas y conectaría, de algún modo, sus programas agentes a la simulación. Al ser una pieza ejecutable, naturalmente se necesitaría algún tipo de *interfaz experimentador/campo de pruebas* para configurarlo y realizar las acciones

que normalmente se harían desde el código fuente —tal vez por medio de una Interfaz Gráfica de Usuario (GUI).

- **Accesibilidad:** El campo de prueba debería ser fácilmente accesible para ser el preferentemente utilizado por los investigadores, evitando que tengan que implementar sus propias versiones a causa de no haber encontrado una disponible.

Habiendo resaltado los requerimientos deseables en un campo de pruebas, es posible realizar una crítica más precisa de los mismo. Esto será útil en la siguiente sección donde se analiza a Tileworld y además, también serán de gran utilidad en el capítulo IV, cuando se hable del campo de prueba desarrollado como parte de este trabajo de fin de carrera —podremos hacer un análisis más profundo sobre qué se necesitó hacer y por qué.

§3.3 Tileworld

Tileworld fue el campo de pruebas elegido como base para el desarrollo de la plataforma. Esta sección está completamente dedicada al él; se comienza describiéndolo en detalle, luego se analizan sus falencias, virtudes y las modificaciones que ha sufrido; también el rol que cumple en la investigación. Finalmente, todo esto se combina para exponer las razones de su uso como parte de este trabajo.

§3.3.1 Descripción

Tileworld es un famoso campo de pruebas para agentes; introducido por primera vez en 1990 por Martha Pollack y Marc Riguette en su artículo científico titulado *"Introducing the Tileworld: Experimentally Evaluating Agent Architectures"*[56]. Consta de

un agente robot y un ambiente simulado que es tanto dinámico como impredecible. Ambos, el agente y el ambiente, están altamente parametrizados, permitiendo controlar ciertas características de cada uno. Por esta razón, fue utilizado por primera vez para investigar experimentalmente el comportamiento de diversas estrategias de razonamiento de meta-nivel y evaluarlas en diferentes ambientes variando los parámetros ambientales[56].

Tileworld consiste en una cuadrícula rectangular sobre la que se sitúan obstáculos, huecos, baldosas (tiles), y un agente robot; como se ilustra en la Figura 3.1. Exceptuando los huecos, cada uno de estos objetos ocupa una celda de la cuadrícula. El agente puede moverse, de a una celda por vez, hacia arriba, abajo, izquierda o derecha —siempre y cuando esto no implique terminar sobre un obstáculo o fuera de los límites de la cuadricula. Cuando una baldosa está en una celda adyacente al agente, este último puede empujarla moviéndose en dirección a ella —haciendo que dicha baldosa se deslice. El agente puede empujar múltiples baldosas al mismo tiempo (filas de baldosas), pero los obstáculos son celdas inamovibles. Un hueco está formado por un grupo de celdas; el agente puede "rellenar" estas celdas empujando una baldosa encima de cada una de ellas. Cuando el agente rellena una celda del hueco, dicha celda desaparece junto con la baldosa utilizada para hacerlo —dejando una celda en blanco. Una vez el agente ha rellenado *todas* las celdas de un hueco, obtiene los puntos asociados al mismo. El agente conoce de antemano qué tan valioso es cada hueco; su objetivo global es el de obtener tantos puntos como sea posible rellenando huecos.

La simulación en Tileworld ocurre de manera dinámica: se comienza con un ambiente generado aleatoriamente por el simulador de acuerdo a un conjunto de parámetros; y luego cambia continuamente a lo largo del tiempo. Al mismo tiempo que el agente se mueve y empuja baldosas adentro de los huecos, los objetos (huecos, baldosas, y obstáculos) aparecen y desaparecen con una frecuencia determinada por ciertos parámetros —fijados por el experimentador. Cada simulación tiene un límite de tiem-

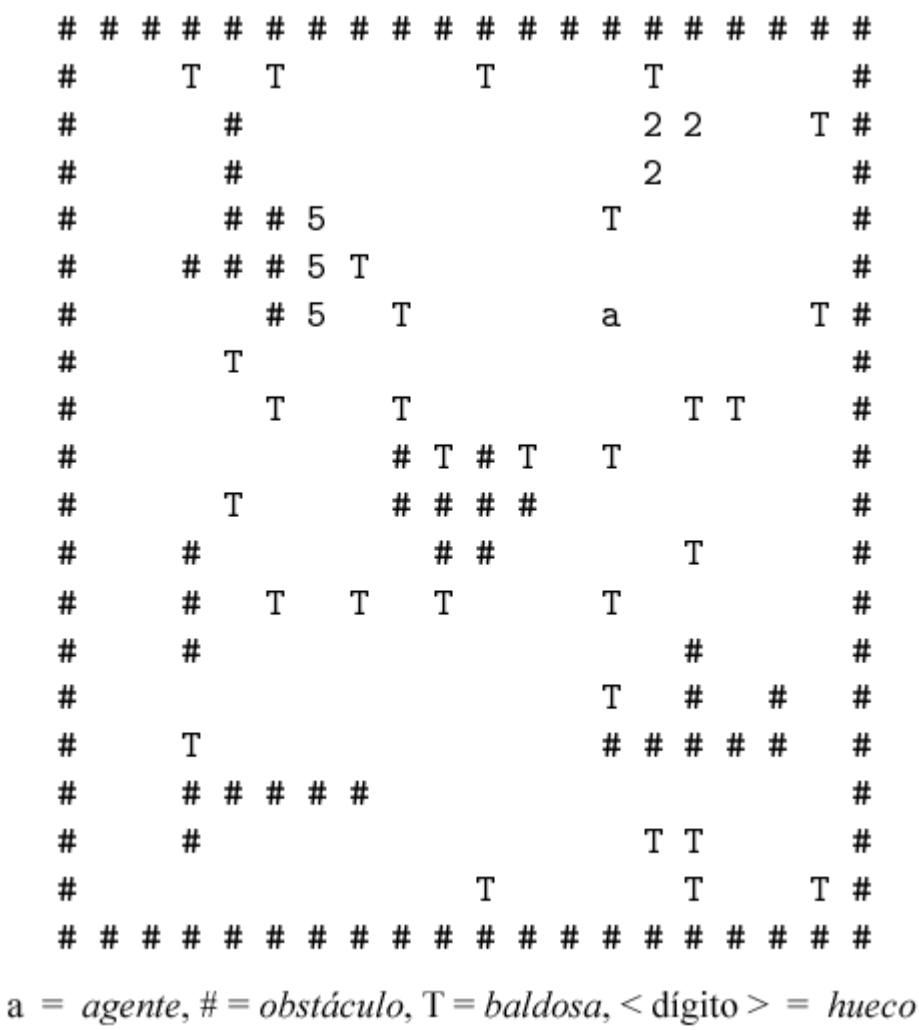


Figura 3.1: Instantánea de un posible estado inicial de Tileworld.

po y el desempeño del agente se mide por la puntuación (score) obtenida al final de la simulación.

Tileworld proporciona una serie de "perillas" (knobs) que pueden ser ajustadas para fijar los parámetros que controlan la simulación, estos son:

- *Dinamismo (Dynamism)*: Controla la frecuencia con la que nuevos huecos aparecen;
- *Hostilidad (Hostility)*: Controla la frecuencia con la que aparecen los obstaculos;
- *Variabilidad de la utilidad*: Controla la diferencia en la puntuación recibida por cada hueco;
- *Variabilidad de la dificultad*: Controla la diferencia en el tamaño de los huecos y la distancia de las baldosas con respecto a los huecos;
- *Límites Suaves/Rígidos (Hard/Soft Bounds)*: Controla si los huecos tienen un tiempo de vida fijo o si decae gradualmente con el tiempo.

Por cada conjunto de parámetros de configuración, el agente puede ser probado en docenas o cientos de simulaciones (trials) generadas aleatoriamente de forma automática. Se pueden comparar diferentes agentes ejecutándolos sobre el mismo conjunto de mundos pseudo-aleatorios.

§3.3.2 Análisis

El campo de pruebas de este trabajo final está creado a partir de numerosas extensiones y mejoras de Tileworld, por este motivo es necesario analizar cuáles son algunos de sus defectos si se pretende mejorarlo.

En 1991, Al-Badr y Hanks [3], investigaron en profundidad el sistema Tileworld y mencionaron que de hecho era demasiado simple; resaltaron que había muchas

características del mundo real (interesantes para los investigadores) que no podían ser estudiadas usándolo:

Desafío para el Agente Debido a su simplicidad, podría no representa un desafío para un agente. El agente sólo puede realizar cuatro acciones y sus efectos son muy limitados; hay pocas características asociadas con cada objeto (posición, capacidad y puntuación). Un agente no tiene la necesidad de ser selectivo sobre qué razonar en la construcción de sus planes: tiene pocas opciones, y todo lo que percibe sobre el mundo es probable que sea relevante en la toma de decisiones.

El mundo en sí mismo tiene una estructura casual demasiada simple: las baldosas y los huecos aparecen dinámicamente de acuerdo a una distribución de probabilidad uniforme, de una manera no sistemática y sin seguir ningún patrón en particular. Por ejemplo, uno podría pensar en construir un agente que sepa, o aprenda, que las baldosas aparecen con mayor probabilidad en la parte superior de la cuadrícula; este tipo de relaciones sistemáticas son las que podrían hacer interesante el problema de predecir el futuro en un ambiente dinámico.

Los sensores y actuadores del agente son perfectos: el agente percibe el mundo tal cual es y sus acciones siempre son determinísticas —siempre producen el efecto esperado.

El éxito del agente se mide por su puntuación (score) lograda en una simulación de duración fija. Si bien esto representa un avance sobre medir el éxito en base a si se llegó o no a un estado objetivo, el inconveniente está en que la puntuación es la *única* medida de éxito —esto no le permite a los investigadores explorar temas como la satisfacción parcial de objetivos y el balance costo/beneficio. El problema real radica en que es el mundo en sí mismo el que mide el éxito del agente.

Facilidad de uso Uno de los elementos de ingeniería más importantes que un campo de pruebas puede proveer es una interfaz limpia entre el agente y el mundo. Sin embargo, en esta implementación de Tileworld, el agente accede directamente a la estructura de datos que el simulador usa internamente para simular el mundo —de esta forma el modelo del mundo del agente y el mundo simulado son exactamente los mismos. El otro elemento de ingeniería importante es el manejo del paso del tiempo. En Tileworld, el agente controla completamente el paso del tiempo; de acuerdo a cuánto tiempo utiliza para razonar y actuar, el agente actualiza el tiempo de simulación. Este aspecto podría ser malo desde un punto de vista experimental: si el tiempo va a ser usado para medir la eficiencia de un agente, se podría considerar una mala idea que él mismo tenga el control del paso del tiempo.

El primer punto sobre considerar la simplicidad de Tileworld como un defecto es relativo ya que, en realidad, hace referencia a dos enfoques diferentes sobre cómo llevar a cabo la experimentación controlada ([39],*ρρ*. 32-38). La postura a favor del uso de ambientes simples sostiene que para que la experimentación tenga éxito en alcanzar su objetivo, tienen que hacerse dos tipos de simplificaciones. La primera es inherente a la noción del diseño experimental. En síntesis, la experimentación necesariamente involucra la atención selectiva y la manipulación de ciertas características del fenómeno siendo investigado, dicha selectividad y control constituyen un tipo de simplificación del fenómeno. El segundo tipo de simplificación que se necesita actualmente surge de las habilidades existentes para construir sistemas de IA complejos. Los sistemas grandes, complejos, que abordan problemas interesantes generalmente no son lo suficientemente íntegros como para permitirle, al experimentador, significativamente probar las decisiones de diseño subyacentes. Más aún, estos sistemas están

diseñados para ambientes en los que puede ser dificultoso o imposible aislar, manipular, y medir características particulares. En contraste, el tipo de sistema simplificado brindado por campos de pruebas tales como el Tileworld están específicamente diseñados para proveer el control requerido por el experimentador. Por esta razón, en el desarrollo original de Tileworld, se adoptó una filosofía minimalista: *la política fue mantener el ambiente tan abstracto y simple como fuese posible, con el fin de brindarle al experimentador un máximo control sobre el ambiente y asegurar que el desempeño del agente no estuviese atado a particularidades de ningún dominio específico.* Cada variable en el Tileworld original se introdujo porque representa una abstracción de una característica ambiental interesante e importante:

- El grado de dinamismo: ¿Qué tan rápido cambia el ambiente?¿Los eventos suceden extremadamente rápido, o el ambiente es bastante tranquilo?
- La uniformidad de la tarea presentada al agente. Esta tiene dos partes:
  - ¿Son las tareas relativamente uniformes en importancia?¿Es probable que el agente reciba una recompensa más o menos equivalente por completar cualquier tarea, o algunas tareas tienen una recompensa mucho mayor que otras?
  - ¿Son las tareas relativamente uniformes en dificultad?¿Es relativamente igual de probable que el agente tenga éxito en cualquier tarea que se le presente, o algunas tareas son mucho más difíciles que otras?

Dentro del Tileworld original, estas características ambientales fueron operacionalizadas y puestas bajo el control del experimentador por medio de las ya mencionadas "perillas" (knobs).

Más importante aún es que la simplicidad de Tileworld da origen a una de sus mayores fortalezas; su flexibilidad conceptual: *es relativamente fácil diseñar modificaciones de Tileworld que soporten experimentos que investiguen cuestiones ambientales*

o de diseño de agentes para los cuales éste no fue originalmente diseñado. Esto se puede observar en los siguientes ejemplos:

- En [53, 54] se presenta una versión extendida de Tileworld. En base a la necesidad de investigar con más detalle la estrategia de razonamiento de meta-nivel "filtering"[1] se realizaron algunas modificaciones para agregar más elementos sobre los que el agente tuviese que deliberar. El sistema se expandió de la siguiente manera: (a) Al agente se le agregó un nivel de combustible que es responsable de mantener. Esto permite el estudio del mantenimiento. (b) Para permitir que el agente mantuviese su propio nivel de combustible, al ambiente se le agregó una "estación de combustible"; esto es, una celda especial a la que el agente tiene que llegar para restablecer su nivel de combustible. Además se agregó otro objetivo de alto nivel, construir pilas de baldosas en ubicaciones estratégicas de la cuadrícula. (c) Se le asignaron figuras a los huecos y a las baldosas. Se cambió la manera en la que se premia al agente por rellenar huecos. Un agente puede rellenar un hueco utilizando cualquier baldosa; sin embargo, obtiene más puntos si utiliza las baldosas cuyas figuras se correspondan con la asignada al hueco. Adicionalmente, el agente es capaz de cargar más de una baldosa —haciendo que consuma más combustible.

- En [19] Pollack, Ephrati y Ur desarrollan una versión multi-agente del sistema denominada MA-Tileworld. Cada uno de los agentes se comporta de la misma forma que el agente de Tileworld. Esta modificación se realizó para soportar la investigación de estrategias de filtrado con un sistema multi-agente.

- En [47] se presenta una versión simplificada de Tileworld; Las simplificaciones del sistema son: los huecos ocupan sólo una celda y se omiten las baldosas; el agente rellena un hueco simplemente ubicándose encima de él. Estas modifica-

[1] En la que un agente se compromete con los objetivos que ya ha adoptado, y luego tiende a filtrar las nuevas opciones que puedan entrar en conflicto con el logro de los objetivos existentes [11].

ciones se realizaron porque los experimentadores consideraron que Tileworld era demasiado complejo como para estudiar el fenómeno de interés.

§3.3.3 ¿Por qué Tileworld?

Si bien es cierto que Tileworld presenta varios defectos, en el siguiente capítulo (§4.1) se verá cómo éstos pudieron ser tratados gracias a su flexibilidad. En esta sección se exponen algunas de las principales razones que fundamentan su uso como base en el desarrollo del campo de prueba de T-World.

Desde su primera aparición en 1990[56] , Tileworld ha sido utilizado y mencionado en un gran número de investigaciones científicas —algunos ejemplos son [32, 73, 1, 50, 46, 61, 4, 30, 27, 51, 63, 64, 62, 9, 55, 48, 8, 68, 57, 45, 43]. Su éxito probablemente se deba a las ventajas que ofrece como campo de prueba, resumiendo:

- Es esencialmente simple, pero *lo suficientemente interesante como para extraer conclusiones de un experimento*. El tener un ambiente simple en el que probar los agentes tiene ventajas:
  - Hace al problema más pequeño y fácil de evaluar.
  - Brinda un máximo control sobre el ambiente y asegura que el desempeño del agente no esté atado a particularidades de un dominio en particular.
  - Un ambiente simple siempre es bueno como punto de partida para una experimentación inicial.
- *Ambiente altamente parametrizado:* cada parámetro de Tileworld representa una abstracción de una característica ambiental interesante e importante.
- *Es conocido y se lo comprende bien:* esto hace posible comparar resultados obtenidos con experimentos similares (usando el mismo ambiente). Importante

cuando la experimentación está en una etapa inicial —es posible obtener un indicador de qué tan factible es lo que uno intenta hacer.

- *Es genérico:* no está fuertemente acoplado a ningún dominio de aplicación, permitiendo que un investigador estudie características que no dependan de un dominio de aplicación en particular. Por ejemplo, el experimentador puede concentrarse en dominios en los que la característica central es una amplia distribución en la valoraciones de las tareas (simulado en Tileworld por la puntuación de los huecos), o en la dificultad de la tarea (simulado por el tamaño de los huecos).

y fundamentalmente:

- *Es flexible:* la capacidad de ser genérico, sumado a su simplicidad, hacen que sea relativamente sencillo diseñar modificaciones de Tileworld para brindar soporte a los distintos tipos de entornos de trabajo—según la categorización dada en §2.5.

Uno de los principales objetivos planteado desde el comienzo de este trabajo fue que la plataforma tendría que permitir estudiar agentes de forma tal que, sin importar cuáles sean los fenómenos que se quieran investigar, una vez detectadas las *condiciones*[2] bajo las cuales deberían ser estudiados, éstas pudieran ser simuladas en el campo de pruebas. Por consiguiente, *el campo de pruebas tendría que ser lo* ***suficientemente flexible*** como para poder abarcar la mayoría de los tipos de *entornos de trabajos.*

Esto hizo que Tileworld fuese un candidato ideal, ya que de entre todos los campos de pruebas consultados en la bibliografía fue el que satisfacía, de manera más directa

[2]Condiciones de entorno de trabajo, tales como ¿El ambiente cambia conforme pasa el tiempo o es estático?¿los sensores son perfectos?¿los actuadores?¿se necesitan multiples agentes?¿éstos van a cooperar entre sí o a competir?, etc.

y sencilla[3], este objetivo —quizas debido a su simpleza. Otros campos de pruebas conocidos, tales como Phoenix [41, 31], Truckworld [24, 52], Ars Magna [20], o RoboCup (fútbol de robots) [12], ofrecen simulaciones en dominios de aplicación específicos y, en algunos casos, no resultó ser tan sencillo adaptarlos a ciertos tipos de entornos de trabajo para los que no fueron creados.

Finalmente, el hecho de que Tileworld haya aparecido en libros de formación académica (tales como [72], [65] o [25]) reforzó la idea de utilizarlo como el campo de pruebas elegido. Ya que el objetivo planteado de tener una plataforma que soportase los distintos tipos de entornos de trabajo también sería *clave* desde el punto de vista de formación académica: las dimensiones bajo las cuales los entornos de trabajo pueden categorizarse determinan, en gran medida, la aplicabilidad de cada una de las *principales familias de técnicas que se utilizan* para la implementación de agentes. A su vez, estas técnicas son introducidas en el dictado de materias relacionadas a la Inteligencia Artificial en todo el mundo. De esta forma, la plataforma sería una herramienta didáctica, lo suficientemente flexible como para permitirle, a los docentes y alumnos, experimentar con cada una de estas técnicas ligadas a los distintos tipos de entornos de trabajo.

□

[3]Al menos para el autor de este trabajo.

CAPÍTULO IV

UNA PLATAFORMA PARA ESTUDIAR AGENTES INTELIGENTES

Este capítulo está enteramente dedicado a la plataforma construida. Se comienza por la sección 4.1, en donde se describe el campo de pruebas, resaltando algunas de sus características más importantes. Luego en la sección 4.2 se estudia de manera general la implementación y las tecnologías involucradas en el desarrollo del sistema que implementa el campo de pruebas y demás funcionalidades de la plataforma. Posteriormente, en la sección 4.3 se hace una breve descripción de los principales componentes de T-World, desde un enfoque de usuario. Finalmente, este capítulo concluye con el desarrollo de un experimento sencillo, en donde utilizando la plataforma se estudian tres tipos diferentes de agentes y se analizan los resultados obtenidos.

§4.1 El Campo de Pruebas

El campo de pruebas de T-World es esencialmente una expansión del Tileworld descrito en el capítulo anterior. Esta expansión se realizó debido a la necesidad de tratar algunas limitaciones presentes en Tileworld y mejorar ciertos aspectos del mismo con el fin de satisfacer los objetivos planteados en este trabajo. Así, de acuerdo a ca-

da uno de los requerimientos expuestos en la sección 3.2 se realizaron las siguientes modificaciones:

- **Medida de éxito:** Se amplía el criterio para medir el grado de éxito del agente por medio de la recolección de estadísticas en base a las siguientes métricas:
  - *Puntuación final:* Puntaje obtenido al terminar la simulación. Notar que esta es la *única* medida disponible en la versión original de Tileworld.
  - *Puntuación acumulada:* Ciertas configuraciones de T-World permiten que el agente pierda puntos, por ejemplo al restaurar su batería. En estos casos, la puntuación final puede ser menor que la puntuación acumulada durante toda la simulación.
  - *Número de movimientos realizados:* se consideran tanto los movimientos válidos como los inválidos. Un movimiento es inválido cuando no es posible realizarlo, y válido cuando sí lo es—un ejemplo de movimiento inválido es avanzar cuando se tiene un obstáculo enfrente.
  - *Número de celdas rellenadas.*
  - *Número de huecos rellenados.*
  - *Batería:* Una de las configuraciones de T-World permite que los agentes mantengan un nivel energético. En este caso, se añaden tres elementos más:
    - *Cantidad de energía consumida durante la simulación.*
    - *Número de recargas de batería realizadas.*
    - *Número de restauraciones de batería realizadas.*

  Luego está en el experimentador analizar e interpretar estos valores y, de acuerdo a la tarea creada, medir el grado de éxito o fracaso del agente.

- **Criterio de parada:** Además, se amplía la manera en la que las simulaciones pueden finalizar[1], ya que pueden seleccionarse condiciones especificas para terminarlas, según si se ha:

  - Alcanzado cierto tiempo de simulación (condición clásica del Tileworld);
  - Llenado una cierta cantidad de celdas o huecos;
  - Alcanzado una cierta puntuación;
  - Realizado una cierta cantidad de movimientos (válidos o inválidos);
  - Consumido una cierta cantidad de energía;
  - Restaurado o recargado la energía una cierta cantidad de veces;
  - Los agentes se han ubicado en ciertas posiciones.

  Opcionalmente, para cada una de estas condiciones, también es posible especificar si la simulación se rotula con "éxito" o "fracaso" al terminar, almacenando este rótulo en las estadísticas.

- **Puntuación:** También se expande levemente la forma en la que el agente obtiene los puntos. Mientras que en Tileworld el agente los gana *únicamente* al rellenar *todas* las celdas de un hueco, en T-World se permite que el agente gane puntos por cada celda rellenada —independientemente de si llena completamente el hueco o no. Esta modificación se debe a que el sistema de obtención de puntos original a veces resulta ser demasiado binario, por ejemplo, un hueco puede desaparecer cuando el agente ya ha rellenado la mayoría de sus celdas, impidiendo que pueda obtener los puntos y haciendo que todo el trabajo realizado no se refleje en el puntaje. Así, bajo el sistema de puntuación original, tanto un agente que rellena la mayoría de las celdas de cada hueco como el que sólo rellena unas pocas (o ninguna) obtienen la misma cantidad de puntos

[1] Recordar que en el Tileworld original todas las simulaciones son de una duración fija.

(cero puntos) —impidiendo, entre otras cosas, el estudio de satisfacción parcial de objetivos. Por lo tanto, en T-World los puntos se manejan de la siguiente manera:

  - Al rellenar *todas* las celdas de un hueco se obtiene una puntuación igual al tamaño del hueco multiplicada por diez. Por ejemplo, si el hueco está formado por 4 celdas, al rellenarlo completamente el agente obtiene 40 puntos (10 x 4) —formato original del Tileworld.
  - Opcionalmente, si se desean las "recompensas parciales" por celdas, la puntuación obtenida por cada una es igual al número de celdas rellenadas hasta el momento multiplicada por dos. Por ejemplo, supóngase un hueco de 4 celdas, al tapar la primera celda se obtienen 2 puntos; la segunda, 4; la tercera, 6 puntos; y finalmente, al tapar la última celda se obtienen los correspondientes 40 puntos.

- **Complejidad del mundo y eventos exógenos:** se agregan más elementos (opcionales) para hacer que los eventos exógenos sean más interesantes y que los agentes tengan más sobre qué deliberar:

  - *Batería:* Los agentes pueden tener un nivel de energía que mantener, en cuyo caso: (a) se agrega una celda de recarga a la que el agente puede ir para recargar su energía, a cambio de 10 puntos. (b) si el agente se queda sin energía, no podrá moverse a menos que elija ceder la mitad de su puntaje para restaurarla.
  - *Multiplicador de puntos:* Al rellenar un hueco completamente, se activa un multiplicador con un valor inicial de 2 que permanece activo durante un tiempo configurable. Mientras esté activo todos los puntos obtenidos se multiplican por su valor. A su vez, si se vuelve a rellenar otro hueco antes que se desactive, aumenta su valor en uno y se reinicia su tiempo.

– *Distribuciones de probabilidad y patrones:* Tileworld tiene una estructura causal simple, ya que si bien se puede controlar el grado de dinamismo del ambiente, no soporta un grado variable del mismo. Como lo expone la autora en [54] (ρ. 8,‡7):

> "Una complicación interesante, que todavía no hemos realizado en el Tileworld, involucra la regularidad del grado de dinamismo. Aunque podemos modelar ambientes que, digamos, cambian muy despacio (o muy rápido), actualmente no podemos modelar ambientes en los que la frecuencia de cambio sea variable."

Tampoco se puede definir que los objetos aparezcan/desaparezcan siguiendo algún patrón cuasi-sistemático, lo que podría ser necesario para hacer interesante el proceso de predicción de los agentes. Por este motivo, haciendo uso del método de simulación Montecarlo[2] se expandió el sistema del siguiente modo:

* Los eventos no necesariamente ocurren con una distribución de probabilidad uniforme, el experimentador puede seleccionar la distribución que desea utilizar o incluso crearla manualmente. Permitiendo así, por ejemplo, que se puedan modelar ambientes en los que la frecuencia de cambio es variable (no uniforme).
* Por cada tipo de objeto dinámico (baldosas y huecos) se puede especificar la probabilidad con la que aparecen en cada celda de la cuadrícula. Esto permite que los objetos aparezcan en ciertas zonas con mayor/menor frecuencia que en otras.

- **Sensores y actuadores:** Se añade la posibilidad de tener sensores y actuadores defectuosos: introduciendo ruido en la percepción del agente y/o limitándola a

[2]Es una técnica que combina conceptos estadísticos (muestreo aleatorio) con la capacidad que tienen las computadoras para generar números pseudo-aleatorios y automatizar cálculos.

un rango especifico de celdas alrededor; además las acciones del agente no siempre producen el resultado deseado, el experimentador puede configurar cuáles son los posibles resultados de las mismas y con qué probabilidad ocurren.

- **Múltiples agentes:** El ambiente puede estar habitado por numerosos agentes conviviendo de forma concurrente en el mismo. Se dispone de tres modos diferentes para realizarlo:
  - *Entorno cooperativo:* Todos los agentes de la simulación comparten los niveles de energía y la puntuación; o
  - *Entorno competitivo:* Todos los agentes compiten entre sí, cada uno con su propio nivel energético y puntuación; o
  - *Entorno híbrido:* Los agentes pueden dividirse en equipos, cada equipo puede estar formado por un número variable de agentes. Los agentes de un mismo equipo comparten energía y puntuación. Por ejemplo, se podrían tener tres equipos, A, B y C; los dos primeros con tres agentes y el último con dos.

Además, con el fin de ampliar el rango y tipo de ambientes soportados por la plataforma, se agregan las siguientes extensiones:

- **Versión simplificada:** Se da la opción de elegir que el ambiente no tenga baldosas y las celdas se rellenen simplemente ubicando al agente encima de las mismas. Esto reduce la complejidad del razonamiento a nivel de **objeto** requerido por el agente; útil, por ejemplo, para investigaciones iniciales y/o para probar políticas, estrategias o fenónemos que no dependan directamente de la dificultad de la tarea en sí misma.
- **Ambiente estático:** Si se desea, se puede establecer que el ambiente en sí mismo no cambie. En cuyo caso el experimentador puede indicar la disposición

de los diferentes objetos en la cuadrícula —Deseable, por ejemplo, para diseñar estados iniciales para los algoritmos de resolución de problemas por medio de búsqueda. Adicionalmente, en este ambiente estático se puede hacer que el valor de los huecos vaya decayendo cada una cierta cantidad de tiempo —permitiendo simular entornos de trabajos semidinámicos.

Finalmente, en relación a los requerimientos de implementación, desde un punto de vista de ingeniería de software, se tiene:

- **Un modelo bien definido del tiempo:** Se mejora la manera en la que se maneja el paso del tiempo, cambiándola por una forma más limpia y natural[3]. El tiempo transcurre en la simulación de forma independiente a lo que el agente haga —el agente es capaz de percibir cuánto tiempo ha transcurrido pero no puede afectar el paso del tiempo en el mundo. Por defecto, la medida de tiempo del simulador se corresponde con la del mundo real, esto es, 1 t-segundo[4] es igual a 1 segundo. Esta correspondencia entre el tiempo de simulación y el real puede ser modificada para producir simulaciones más lentas o más rápidas según así lo requiera el experimentador. Por ejemplo, haciendo que 1 t-segundo sea igual a 10 segundos la simulación transcurre 10 veces más lenta, ya que cada 10 segundos reales transcurre uno en la simulación. Finalmente, se define el costo temporal de las acciones de la siguiente manera:

  - Al agente le toma 350 t-milisegundos moverse hasta una celda adyacente. Pero si al hacerlo rellena un hueco, debe esperar 250 t-milisegundos extras antes de poder moverse —este es el tiempo que le toma a la baldosa caer y rellenar la celda.

  - El agente pierde 500 t-milisegundos por cada movimiento inválido.

[3]Recordar que en el Tileworld original el agente controla el paso del tiempo.

[4]Se antepondrá la letra t antes de la unidad de tiempo para referir a *"unidad de tiempo de simulación"*.

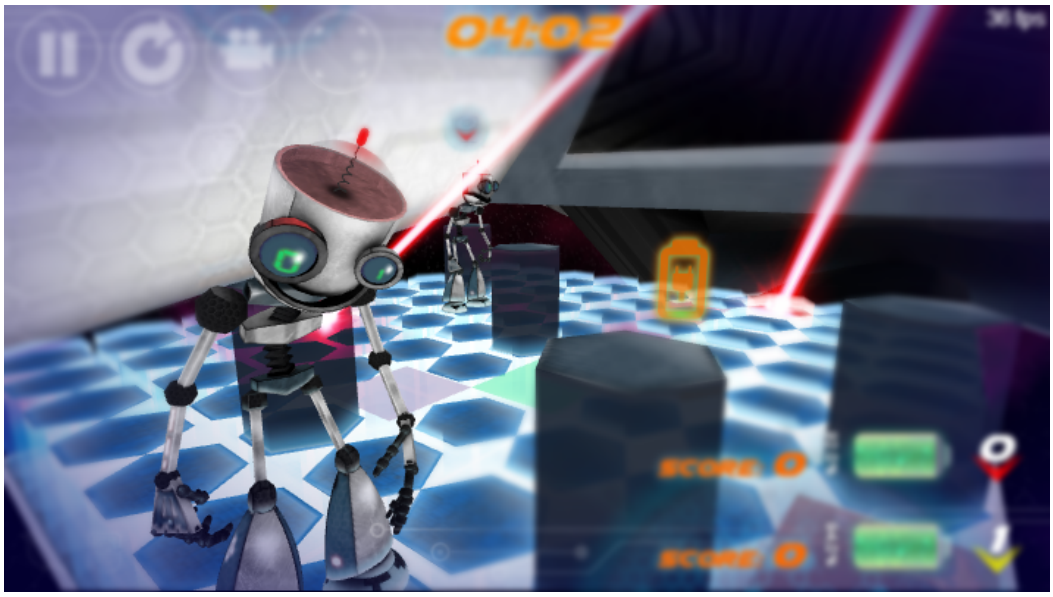


Figura 4.1: Instantánea de la simulación de un experimento en T-World.

- **Una interfaz limpia:** La interfaz define de qué manera el experimentador conecta sus programas agente a la simulación. Por este motivo, se mejora la interfaz entre el agente y el ambiente cambiándola por una más natural y limpia. Se brindan un conjunto de acciones y un formato de percepción bien definidos y documentados; únicamente a través de ellos el agente puedo realizar y percibir cambios en y desde el ambiente, respectivamente. Este punto se describe con más detalle en la sección 4.3.

- **Soporte para la experimentación:** Se expande la familia de "perillas" originales para brindar un soporte controlado y parametrizado de las nuevas modificaciones. Además, se mejora considerablemente la manera en la que se puede monitorear el comportamiento de los agentes: como se observa en la Figura 4.1, la simulación transcurre en un ambiente tridimensional (similar al contexto de un videojuego); el movimiento de los agentes y demás objetos se realiza de forma continua (no discreta), aproximándose a un experimento real. Por ejemplo, se puede observar el efecto visual del robot empujando las baldosas, la reacción del robot al chocar con un obstáculo, las baldosas al caer y rellenar los huecos,

etc.

Por último, se enriquece la información recolectada en las estadísticas, incluyendo no sólo las métricas dadas al inicio (ver ítem *"Medida de éxito"* en página 46) sino también elementos externos al agente tales como, entre otros, la cantidad total de huecos que fueron creados, la velocidad y duración de la simulación (en tiempo real y de simulación). Adicionalmente, estas estadísticas se recolectan tanto individualmente (por agente) como grupalmente (por equipo) ligándolas a una cierta ejecución de simulación (*trial*). Luego, por medio de un historial de ejecuciones se puede tener acceso a cada una de ellas.

§4.2 El Sistema

En la sección anterior se describió el campo de pruebas de T-World. Para ponerlo en operación y poder ser utilizado por el experimentador se construyó un sistema de software. Este sistema no soló permite configurar y ejecutar el campo de pruebas, sino que le brinda al usuario una interfaz gráfica y una rica variedad de funcionalidades, tales como: alta, baja y modificaciones de sus *entornos de trabajo* y *programas agente*; administración de estadísticas por medio de historial de ejecuciones; sistema de usuarios para permitir el trabajo cooperativo y almacenamiento de datos a través de Internet; junto a todo lo necesario para hacer del mismo una plataforma para el estudio de agentes inteligentes y el aprendizaje de Inteligencia Artificial.

En esta sección se describe de manera general la implementación y las tecnologías involucradas en el desarrollo de dicho sistema. Antes de comenzar es importante mencionar que, en base a los criterios de *accesibilidad* y *existencia propia* dados en §3.2.2 (véase página 33), se optó por utilizar el navegador web como plataforma de

implementación. Notar que no se habla de "la web", sino del *programa* "navegador web" ejecutando en la computadora del usuario. Esto aumenta la portabilidad del sistema debido a que todos los dispositivos actuales tales como smartphones, tablets y computadoras personales disponen de un navegador web. Además permite que el sistema pueda ser accedido vía Internet (o localmente) y ejecutado directamente en el dispositivo del usuario sin necesidad de instalar o configurar software adicional.

§4.2.1 Codificando la Lógica

Al elegir el navegador web como plataforma de implementación, toda la codificación del sistema tiene que ser hecha en algún lenguaje de programación capaz de ejecutar nativamente dentro de él. En este punto no hubo demasiadas opciones a considerar ya que el único lenguaje soportado por los navegadores es JavaScript —de hecho, sus creadores y muchos otros lo consideran como el "lenguaje ensamblador de la Web"[40]. Antes de comenzar a describir este lenguaje, se cree conveniente señalar que para implementar la plataforma T-World se necesitaron escribir alrededor de 20.000 lineas de código, de las cuales aproximadamente 13.000 están en JavaScript —esto equivale a más de 500 hojas A4 solamente de código JavaScript. Además, el código fue liberado por el autor bajo la licencia de código abierto *Affero General Public License (AGPL)* de *GNU*[29], para ello previamente se necesitó registrarlo en la *U.S Copyright Office*, de forma tal que cualquier interesado pueda redistribuirlo y/o modificarlo libremente[5], así como también contribuir al desarrollo y crecimiento de este proyecto. El código puede ser descargado desde el sitio de la plataforma, `http://tworld-ai.com`.

[5]Siempre respetando las condiciones de la licencia.

4.2.1.1 El Lenguaje de Programación

JavaScript (JS) es un lenguaje de programación dinámico[26], comúnmente usado como parte de los navegadores web en donde se lo utiliza para interactuar con el usuario, controlar el navegador, comunicarse asíncronamente con el servidor, y alterar el contenido del documento mostrado[26].

A JavaScript se lo clasifica como un lenguaje de *scripting basado en prototipos*, con tipos dinámicos y *first-class functions*. Donde scripting refiere a programas escritos para un ambiente en tiempo de ejecución especial. Un lenguaje de scripting puede verse como un lenguaje de dominio especifico para un ambiente particular. También son a veces referidos como lenguajes de programación de "muy alto nivel", ya que operan a un nivel de abstracción muy alto. Que sea basado en prototipos significa que tiene un estilo de programación orientada a objeto en la que la herencia se realiza clonando objetos existentes que sirven como prototipos. Finalmente que tenga *first-class functions* significa que las funciones son tratadas como objetos de datos, e.d. pueden ser pasadas como argumentos, retornadas como valores, y asignadas a variables o almacenadas en estructuras de datos. Toda esta mezcla de características hacen que JavaScript sea un lenguaje multi-paradigma, soportando el estilo orientado a objeto, imperativo y funcional.

Más allá del nombre, JavaScript y Java no están relacionados y tienen semánticas muy diferentes. La sintaxis de JavaScript de hecho deriva de C, mientras que la semántica y diseño están influenciados por los lenguajes de programación *Self* y *Scheme*[17].

Los navegadores internamente tienen lo que se conoce como un *JavaScript engine*, encargado de interpretar y ejecutar el código fuente. JavaScript ha sido tradicionalmente implementado como un lenguaje interpretado, pero los navegadores más recientes realizan compilación *Just-In-Time (JIT)*, una técnica de compilación que realiza la traducción durante la ejecución en lugar de previo a ella, lo que mejora los

tiempos de ejecución considerablemente.

4.2.1.2 El Framework para la Arquitectura

Como ya fue explicado más arriba, se necesita implementar el sistema haciendo uso del navegador como plataforma para su ejecución. Siendo más precisos, se necesita desarrollar lo que se conoce como una *apliación web de una única página*: un "programa de computadora" ejecutando en una única página HTML directamente desde adentro del navegador. Para ayudar y acelerar el desarrollo de este tipo de aplicaciones se han creado diversos frameworks tales como *AngularJS*[44], *Backbone.js*[6] y *Ember.js*[74]. Se decidió utilizar *AngularJS* porque fue el que pareció mostrar una *API* más clara y simple de comprender, que junto a todo el material de apoyo y didáctico disponible desde su sitio web lograron que fuese relativamente rápido aprender a utilizarlo.

AngularJS es un framework libre para desarrollar aplicaciones web mantenido por Google Inc. Fue construido para abordar muchos de los desafíos enfrentados en la construcción de *aplicaciones web de una única página*. Su objetivo es el de simplificar tanto el desarrollo como la prueba de las mismas brindado una arquitectura *Modelo-Vista-Controlador (MVC)* para codificarlas. Dicha arquitectura sigue el patrón de diseño estructural mostrado en la Figura 4.2, en donde de forma general:

- *El Modelo:* Es la representación de la información con la cual el sistema opera, por lo tanto gestiona todos los accesos a dicha información, tanto consultas como actualizaciones. Envía a la Vista aquella parte de la información que se solicita mostrar en cada momento.

- *El Controlador:* Responde a eventos (usualmente acciones del usuario) e invoca peticiones al Modelo cuando se hace alguna solicitud sobre la información. También puede enviar comandos a la Vista si se solicita un cambio en la forma

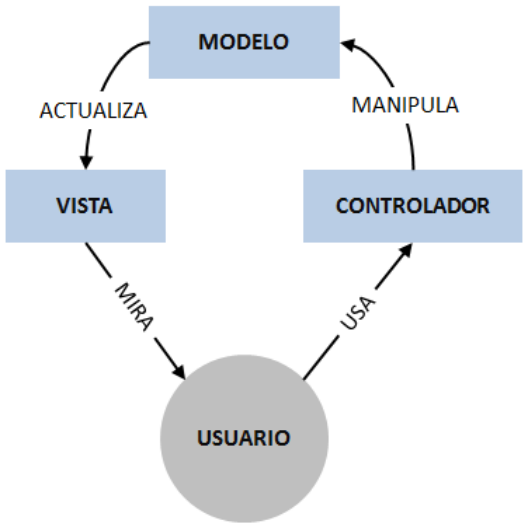


Figura 4.2: Típica colaboración entre los componentes de un MVC.

en que se visualiza el Modelo, por lo tanto se podría decir que el Controlador actúa como intermediario entre la Vista y el Modelo.

- *La Vista:* Presenta el Modelo en un formato adecuado para interactuar, usualmente es la interfaz gráfica de usuario.

El funcionamiento básico de *AngularJS* es el siguiente: primero se lee el documento HTML, el cual tiene embebido elementos y atributos HTML adicionales, propios de *AngularJS*. Estos elementos luego son interpretados como directivas diciéndole a *AngularJS* que enlace partes de la página a un modelo que está representado por variables JavaScript estándares. Los valores de estas variables pueden ser establecidos manualmente desde el código o asignados dinámicamente.

§4.2.2 Creando una Interfaz Gráfica de Usuario

Se ha hablado sobre el lenguaje de programación y el framework empleado para desarrollar la *aplicación web*. Ahora se prestará atención a la interfaz usuario/T-World y las tecnologías empleadas para crear los elementos visuales y multimedia utilizados

para construirla. Con el fin de lograr que el usuario interactuara con la plataforma de una manera visual e intuitiva, ofreciéndole mayor orientación y haciendo que sea más fácil comenzar a utilizarla, se decidió construir una *Interfaz Gráfica de Usuario* (comúnmente denominada *GUI*). Como se observa en la Figura 4.3, la interfaz cuenta con una gran colección de elementos visuales tales como menús, iconos, botones e imágenes, junto al ambiente en tres dimensiones para las simulaciones.

4.2.2.1 Aspecto Visual de la Aplicación

Para proporcionar el estilo visual deseado y construir los elementos visuales de la GUI se utilizó el lenguaje *Cascading Style Sheets (CSS)*. Este es el lenguaje que se usa para especificar el aspecto y el formato de un documento estructurado escrito en HTML. Junto con HTML y JavaScript, CSS es la tecnología fundamental usada por la mayoría de los sitios web para crear páginas visualmente atractivas, interfaces de usuario para aplicaciones web y aplicaciones móviles.

CSS está diseñado principalmente para permitir una separación entre el contenido neto de un documento y su presentación, incluyendo elementos tales como el diseño, los colores y las fuentes[69]. Esta separación puede, entre otras, mejorar la accesibilidad al contenido, brindar mayor flexibilidad y control en la especificación de las características de la presentación, permitir que múltiples páginas HTML compartan el formato escribiendo el CSS en un archivo separado, reduciendo así la complejidad y repetición en el HTML.

Por otro parte, con el fin de permitir que el estilo de las páginas web sea más compatible con los estándares y fácil de desarrollar, se han creado numerosas bibliotecas pre-preparadas denominadas *frameworks CSS*. Algunos de los frameworks CSS más populares son Bootstrap, Foundation, Blueprint, Cascade Framework y Materialize.

Bootstrap de *Twitter Inc.* es una colección de herramientas de código abierto para crear sitios y aplicaciones web. Contiene plantillas de diseño basadas en CSS y HTML

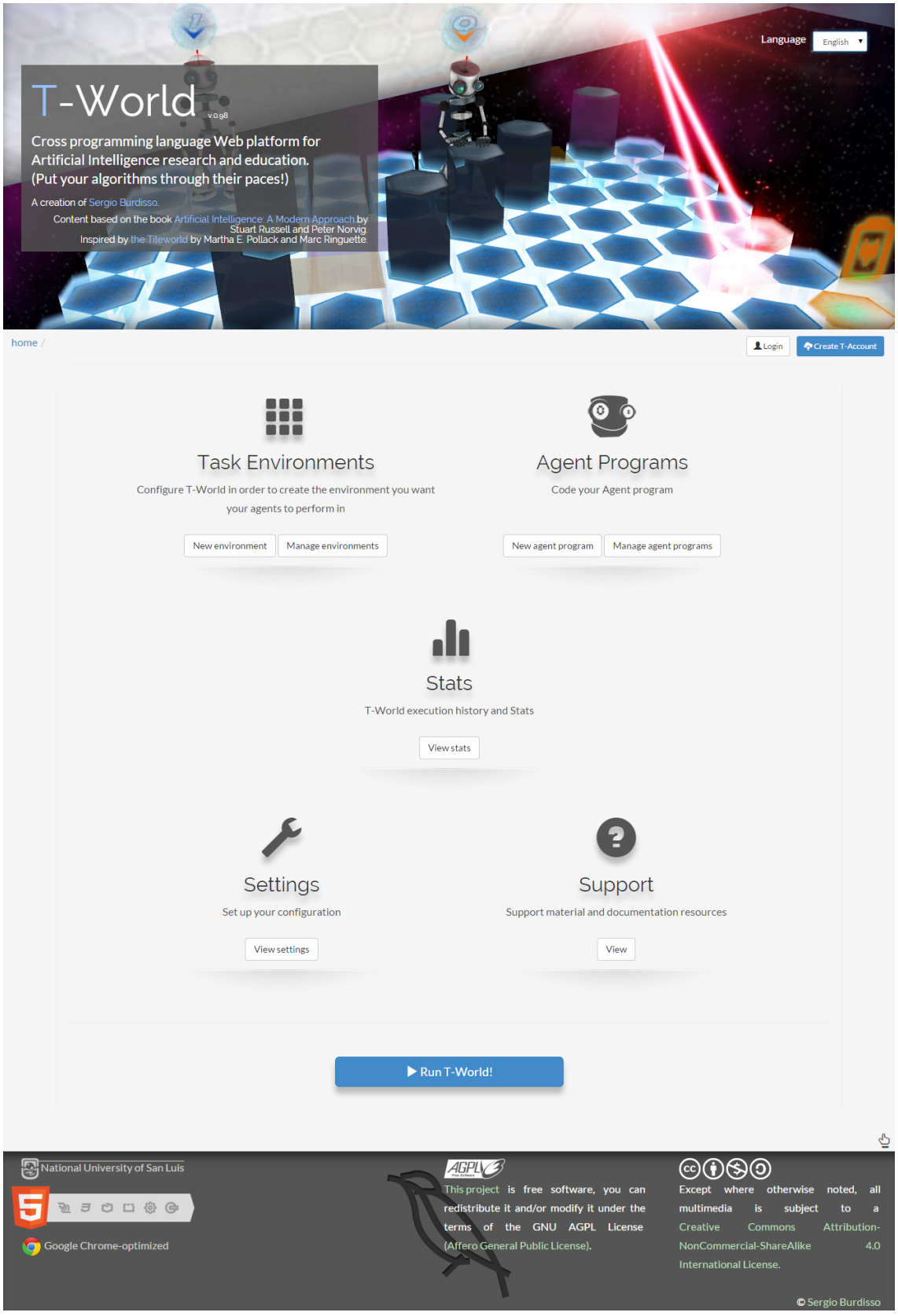


Figura 4.3: Instantánea del menú principal de T-World.

para tipografía, formularios, botones, y otros componentes de navegación e interfaz. De entre todos se decidió utilizarlo porque además de ser uno de los más populares[6] también posee un conjunto de características que fueron deseables:

- *Velocidad de desarrollo:* En lugar de codificar desde cero, permite utilizar bloques de código CSS ya preparados con soporte para múltiples navegadores, lo que ahorra muchas horas de codificación.

- *Responsividad:* El uso de dispositivos móviles continua creciendo año tras año. Con Boostrap no se necesita hacer ningún trabajo extra para hacer que el diseño de la página se adapte a los dispositivos móviles, ya que provee un conjunto de clases listas para realizar esta tarea automáticamente y de forma transparente.

- *Consistencia:* Los resultados de diseño son uniformes en todas las plataformas de modo que la salida es la misma ya sea que se esté usando, por ejemplo, Firefox, Chrome o Internet Explorer.

- *Personalizable:* Puede ser modificado de acuerdo a las necesidades del proyecto. Los desarrolladores tienen la opción de seleccionar y elegir sólo las características que se necesitan y el resto puede ser descartado.

- *Soporte:* Existe una enorme comunidad detrás de Bootstrap, por lo que se puede obtener ayuda fácilmente en caso de problemas.

Como se puede observar, Bootstrap permite un desarrollo rápido, responsivo y consistente con el respaldo de una gran comunidad de desarrolladores y diseñadores. Lo que motivó su uso para desarrollar la Interfaz Gráfica de Usuario de la plataforma.

[6]Es el proyecto con más estrellas en *GitHub*, con más de 78.000 estrellas y más de 30000 bifurcaciones (forks)[28].

4.2.2.2 Simulación en 3-Dimensiones

Para visualizar el ambiente tridimensional de la simulación, primero se tuvieron que modelar individualmente en 3D todos los objetos que conforman el ambiente (el robot, la celda, el obstáculo, la baldosa, etc.), luego se crearon todas las texturas y animaciones necesarias. Una vez realizado este trabajo fueron exportados individualmente a archivos gráficos de DirectX (con extensión .x), para ser posteriormente incorporados al ambiente de la simulación dentro del navegador haciendo uso de la tecnología descrita a continuación.

Web Graphics Library (WebGL) es una API JavaScript para renderizar gráficos interactivos 3D y 2D dentro de un navegador Web sin el uso de plug-ins[34]. Está completamente integrado en todos los estándares de la web del navegador, permitiendo el uso de física y procesamiento de imágenes acelerados por la Unidad de Procesamiento Gráfico (GPU) como parte de un elemento *canvas* de HTML5 en una página web. De esta forma pueden ser mezclados con otros elementos HTML e integrados con otras partes de la página. Los programas WebGL consisten de código de control escrito en JavaScript y código gráfico (*shader*) que es ejecutado en la GPU.

Como está basado en OpenGL y está integrado en múltiples navegadores, WebGL ofrece un gran número de ventajas, entre ellas:

- Una API que está basado en un estándar de gráficos 3D familiar y ampliamente aceptado.
- Compatibilidad multi-navegador y multi-plataforma.
- Fuerte integración con el contenido HTML, incluyendo composición en capas, interacción con otros elementos HTML, y uso de los mecanismos de manejo de eventos HTML estándares.
- Gráficos 3D acelerados por hardware para el ambiente del navegador.

Sin embargo, la API WebGL puede ser muy tediosa de usar directamente sin alguna biblioteca de utilidad. Es por esto que se han creado bibliotecas para trabajar a un nivel de abstracción más alto y proporcionar funcionalidades adicionales. Una lista no exhaustiva de las bibliotecas más conocidas que proveen estas características de alto nivel incluye a CopperLicht, three.js, O3D, OSG.JS y GLGE. De entre estos, el único con soporte para importar los archivos de DirectX creados para los objetos del ambiente fue CopperLicht.

CopperLicht es un motor 3D (en Inglés, *3D engine*) JavaScript y una biblioteca WebGL de código libre utilizada para crear juegos y aplicaciones 3D interactivas en el navegador Web, desarrollada por *Ambiera*[5]. Esta biblioteca es una de las más conocidas y utilizadas en todo el mundo[21, 67, 33]; el objetivo de la biblioteca es el de proporcionar una API para hacer que el desarrollo de contenido 3D en la web sea más simple abstrayendo el API WebGL original.

Se cree adecuado en este punto mencionar que durante el desarrollo de T-World el autor de este trabajo tuvo que realizar numerosos cambios en el código fuente de CopperLitch para mejorar aspectos relacionados mayormente con su rendimiento. Estas mejoras fueron bastante significativas y, una vez notificadas a los autores de CopperLitch, decidieron incorporarlas a la biblioteca oficial. También, como muestra de agradecimiento, agregaron a T-World en la página oficial de la biblioteca (`http://www.ambiera.com/copperlicht/`).

§4.2.3 Dando un Formato Estándar a las Percepciones

Si bien hasta ahora se ha hablado sobre aspectos generales de implementación, como ser el lenguaje utilizado y la Interfaz Gráfica de Usuario, en adelante se tratarán aspectos de implementación relacionados al campo de prueba del sistema. Así, para evitar que el experimentador tenga que aprender y adaptarse a nuevos lenguajes de

programación, uno de los objetivos a alcanzar fue que los programas agente pudieran ser codificados en cualquier lenguaje. Para lograrlo, lo primero sobre lo que se necesitó trabajar fue el formato con el que se transmitiría la percepción de manera que, al ser recibida por un programa agente, pudiera ser interpretada sin importar el lenguaje de este programa agente. Para lograrlo se estableció que los datos de la percepción enviados durante la simulación deberían tener una estructura predefinida e independiente del lenguaje. De esta forma, se optó por utilizar los formatos *JSON* o *XML*[7], dependiendo de lo que el usuario elija. A continuación se describe el formato JSON —se decide omitir a XML en este informe por simplicidad.

JavaScript Object Notation (JSON) es un *formato estándar abierto que utiliza texto legible por humanos para transmitir objetos de datos* formado por pares atributo/valor. Es principalmente usado para transmitir datos entre un servidor y una aplicación web, como una alternativa a *Extensible Markup Language (XML)*. Leerlo y escribirlo es simple para los seres humanos, a la vez que para las máquinas es simple interpretarlo y generarlo. Utiliza convenciones que son ampliamente conocidas por los programadores de la familia de lenguajes C —incluyendo C, C++, C#, Java, JavaScript, Perl, Python, y muchos otros. Estas propiedades hacen que JSON sea un lenguaje ideal para el intercambio de datos.

Aunque originalmente se derivó de JavaScript, JSON es un formato de datos completamente independiente del lenguaje. Las rutinas para leer y generar datos en JSON se encuentran disponibles para todos los lenguajes de programación populares[18] —*ActionScript, C, C++, C#, ColdFusion, Common Lisp, Delphi, E, Eiffel, Java, JavaScript, ML, Objective-C, Objective CAML, Perl, PHP, Python, Rebol, Ruby, Lua, Visual FoxPro*, etc.

Los tipos básicos de JSON son:

- *Número* — un número es similar a un número en C o Java, excepto que no

[7]Opcionalmente, para brindar soporte a Prolog, también pueden ser enviados en forma de un hecho Prolog.

se usan los formatos octales ni hexadecimales. JSON no permite valores no numéricos tales como *NaN*, tampoco hace distinción alguna entre enteros y puntos flotantes.

- *String* — una secuencia de cero o más caracteres delimitada por comillas dobles (`""`).
- *Boolean* — cualquiera de los valores `true` o `false`.
- *Array* — una lista ordenada de cero o más valores, cada uno de los cuales puede ser de cualquier tipo. Los Arrays usan notación con corchetes (`[]`), con los elementos separado por una coma (`,`).
- *Object* — una colección no ordenada de pares nombre/valor donde los nombres (también llamado claves, o *keys* en Inglés) son Strings. Los objetos se delimitan con llaves ({ }) y usan comas (`,`) para separar cada par, mientras que dentro de cada par el caracter dos puntos (`:`) separa la clave de su valor.
- *Nulo* — un valor vacío, se utiliza la palabra `null`

En el siguiente ejemplo se muestra un representación posible para describir a un objeto de dato *persona* en JSON:

```
{
    "nombre": "René Géronimo",
    "apellido": "Favaloro",
    "soltero": false,
    "edad": 77,
    "alturaCm": 182.6,
    "direccion": {
       "direccionCalle": "Calle 1 1791",
       "ciudad": "La Plata",
       "provincia": "Buenos Aires",
       "codigoPostal": 1900
    },
    "numerosDeTelefono": [
       {
          "tipo": "hogar",
          "numero": "0221 421-1190"
```

```
        },
        {
            "tipo": "oficina",
            "numero": "011 4378-1200"
        }
    ],
    "hijos": [],
    "alergias": null
}
```

Para aquellos interesados en saber cómo realmente luce una percepción en JSON dentro de T-World, en el Apéndice A se muestra una de ellas, capturada desde una simulación realizada.

§4.2.4 Comunicando los Agentes con el Ambiente

Una vez decidido el formato de los datos, se necesita analizar cómo efectivamente se va a crear la comunicación para transferirlos. Desde el comienzo del diseño del T-World quedó claro que la comunicación con los agentes tendría que ser bidireccional, ya que tanto el ambiente como los agentes pueden recibir respectivamente acciones y percepciones, sin seguir ningún patrón preestablecido, e.d. ambos pueden hacerlo en cualquier momento y en cualquier orden, incluso en simultáneo. Sin embargo, este requerimiento se contrapone a la idea original con la cual fue concebida la Web, donde el usuario especifica una URL para recuperar un recurso y como respuesta el servidor entrega una instancia del mismo —por ejemplo, un documento HTML, un archivo de imagen o de cualquir otro tipo. Lamentablemente debido a las características del protocolo utilizado en la web (*Hypertext Transfer Protocol (HTTP)*) los clientes se comunican con el servidor usando una comunicación *half-duplex* con patrón cliente/servidor, esto significa que solamente el cliente puede enviar mensajes e iniciar la comunicación, la otra parte se limita a responder. Normalmente, en un esfuerzo de

simular una comunicación full-duplex sobre HTTP half-duplex, los desarrolladores han inventado trucos y técnicas usando dos conexiones: una para recibir y otra para enviar. Varias de estas técnicas usan *polling* o *long-polling* para simular envíos iniciados desde el servidor. El mantenimiento y coordinación de dos conexiones genera importantes demoras por la utilización de recursos y agrega bastante complejidad. Además estas técnicas no proveen una verdadera comunicación full-duplex en la cual los datos entre el cliente y el servidor puedan ser transmitidos al mismo tiempo. En resumen, HTTP no fue diseñado para soportar una comunicación bidireccional en tiempo real.

Afortunadamente, con la reciente aparición de WebSocket de HTML5, ahora sí es posible tener un modelo de comunicación full-duplex para el navegador web:

El protocolo *WebSocket* proporciona canales de comunicación *full-duplex* sobre una única conexión TCP. El protocolo fue estandarizado por la Internet Engineering Task Force (IETF) con el RFC 6455 [42] en el 2011 y está diseñado para ser implementado en los navegadores y servidores web, sin embargo puede ser usado por cualquier aplicación. Esto se debe a que es un protocolo independiente basado en TCP, su única relación con el protocolo HTTP es que su *handshake*[8] es interpretado por los servidores como un mensaje HTTP. El protocolo WebSocket aumenta la interacción entre el navegador y el sitio web, por ejemplo facilitando el contenido en directo y la creación de juegos en tiempo real. Esto es posible al proveer una manera estandarizada para que el servidor le envíe contenido al navegador sin tener que ser explícitamente solicitado por el cliente, y permitiendo que los mensajes fluyan en ambas direcciones mientras se mantiene la conexión abierta. Además, es importante considerar que como el tráfico WebSocket puede fluir sobre los puertos de HTTP estándar (80 y 443), no hay necesidad de habilitar puertos adicionales para conseguir

[8] *Handshaking* es un proceso de negociación que establece los parámetros de un canal de comunicaciones establecido entre dos entidades antes de que comience la comunicación normal.

comunicación bidireccional.

Con este protocolo se abrió la posibilidad de que T-World pudiese conectar los agentes con el ambiente sin las dificultades ni demoras de las aplicaciones anteriores, como se detalla en la siguiente subsección.

§4.2.5 Uniendo las Partes

Juntando los elementos vistos hasta ahora, el esquema de conexión sería el siguiente: (a) enlazar cada programa agente de usuario con su respectivo robot (agente) en el navegador web por medio del protocolo WebSocket —permitiendo una comunicación bidireccional entre ambos; (b) las percepciones se enviarán en un formato estándar (JSON o XML) para poder ser interpretadas por el programa agente receptor sin importar su lenguaje.

Sin embargo, este esquema presenta una falla. Si bien se utiliza un formato de datos estándar para representar los datos, la comunicación se realiza sobre un protocolo específico (WebSocket). Esto implica que el usuario/experimentador tenga que aprender a implementar dicho protocolo solamente para poder comunicar su programa agente con el ambiente. Esto no es deseable, ya que el usuario simplemente debería ser capaz de conectar su programa agente, recibir las percepciones, y enviar las acciones sin tener que preocuparse en absoluto por los detalles de la comunicación subyacente. Este problema se solucionó con la creación de un *proxy*.

Un *servidor proxy* (o simplemente *proxy*) es un programa o sistema informático que rompe la conexión directa entre un emisor y un receptor, funcionando como un intermediario entre ellos. Ambos, el emisor y el receptor piensan que se están comunicando entre sí, pero en realidad, lo hacen sólo con el proxy. De allí su nombre, ya que la palabra proxy en inglés significa “actuar en nombre de otro”. Este concepto

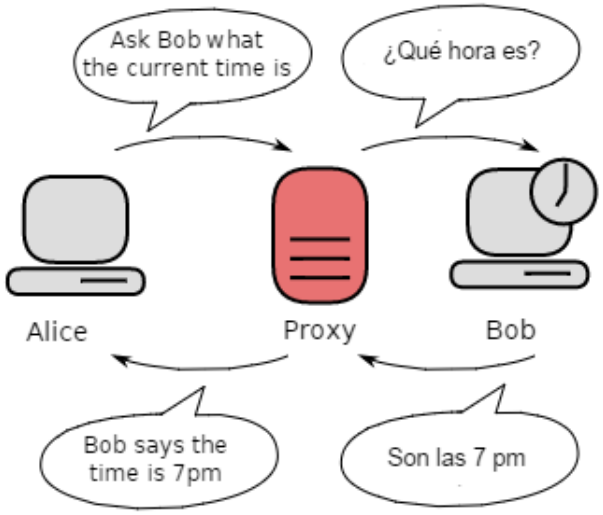


Figura 4.4: Concepto ilustrado de un Proxy.

se ilustra en la Figura 4.4, en donde se ve la comunicación entre dos computadoras (en gris) conectadas a través de una tercera computadora (en rojo) que actúa como un Proxy. Notar que Bob desconoce que Alice hable Inglés y Alice que Bob Español, pero aún así pueden comunicarse normalmente ya que el Proxy se encarga de hacer que ésto sea completamente transparente para ellos.

A continuación se describe detalladamente la implementación y funcionamiento del proxy de T-World, el cual básicamente sigue el mismo esquema planteado en el ejemplo de la Figura 4.4, donde Bob se correspondería con el programa agente del usuario, Alice con T-World, el Inglés con el protocolo WebSocket y el Español con las percepciones y las acciones.

4.2.5.1 El Servidor Proxy de T-World

El proxy se encarga de enlazar cada agente (robot) de la simulación 3D con su respectivo programa agente. Dado que la simulación transcurre dentro del navegador web, T-World asigna un WebSocket a cada robot y conecta cada WebSocket a su respectivo proxy. Es por esto que el diálogo entre T-World y el proxy se hace por medio del protocolo WebSocket. Por otra parte, el programa del usuario sólo debe conectarse

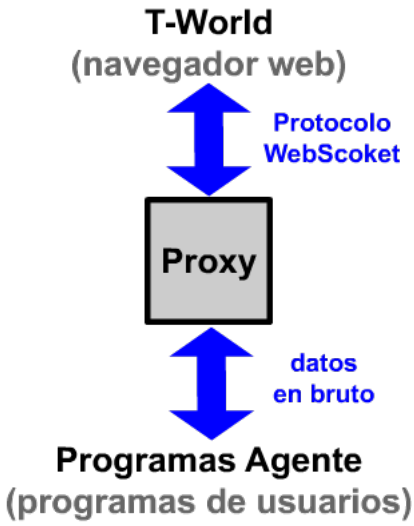


Figura 4.5: Flujo de comunicación entre T-World, Proxy y Programas Agente.

al proxy, recibir las percepciones JSON desde T-World y enviar las correspondientes acciones. Este proceso se ilustra en la figura 4.5. El servidor proxy ha sido codificado en lenguaje C, haciendo uso sólo de sockets estándares (POSIX). Esto asegura la portabilidad a todas las plataformas y sistemas operativos.

Para poder realizar su labor, el proxy dispone de un socket principal cuya tarea es la de escuchar nuevos pedidos de conexión y, cada vez que uno de éstos arriba al servidor, se crea una nueva conexión. Esto último se lleva a cabo creando un nuevo socket por cada nueva conexión. Cuando uno de estos sockets recibe el mensaje correspondiente al *handshake* del protocolo WebSocket (ver sección 1.3 Opening Handshake de [42]), el proxy lo identifica como un WebSocket proveniente de T-World. Luego ambas partes, tanto los programa de usuario como los WebSockets envían un mensaje especial, llamado *CONNECT*, para que el proxy pueda identificarlos y aparearlos de forma apropiada. El mensaje CONNECT se forma por la cadena "CONNECT:" concatenada con la *Magic String* que el usuario le asignó al programa agente durante la creación del mismo en T-World. La Magic String es una cadena dada por el usuario que sirve como el identificador del programa agente, de forma tal que éste se enlace sólo con el robot correcto –aquel que tenga asignado la misma Magic String

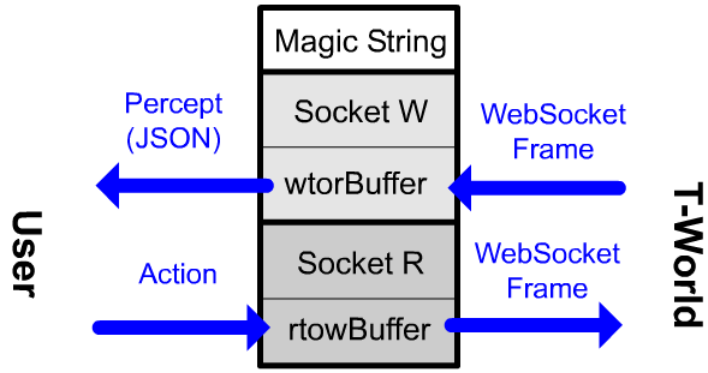


Figura 4.6: Estructura dedicada para cada par de conexión.

que su programa agente. Este Magic String se utiliza para armar pares de conexiones WebSocket/Socket bajo un mismo valor de Magic String. Estos pares se arman de la siguiente manera: cuando un mensaje CONNECT arriba desde un socket, a este socket se le asigna una etiqueta con el valor de la Magic String recibida. Posteriormente cuando un WebSocket envía el mensaje CONNECT se busca un socket libre[9] con la misma Magic String, de encontrarse uno se arma el par WebSocket/Socket con ellos. Este proceso también puede ocurrir a la inversa, que el WebScoket sea el primero en enviar el mensaje CONNECT y que posteriormente un socket lo haga. En este caso se busca un socket libre para este WebSocket.

Para poder llevar a cabo el proceso antes descrito, el proxy genera una *estructura registro* por cada par de conexión. Esta estructura se ilustra en la Figura 4.6. Como se observa en la figura, la estructura tiene un campo para almacenar la Magic String bajo la cual están apareados los sockets. También hay dos campos para guardar el socket de usuario (Socket R) y el WebSocket (Socket W) del par[10]. Los campos wtorBuffer y rtowBuffer son buffers de 16KB utilizados para el reenvío de mensajes desde el WebSocket hacia el usuario y viceversa, proceso que se describe en las siguientes secciones.

[9]Por libre se refiere a que no pertenezca ya a un par WebSocket/Socket.
[10]Socket R y Socket W guardan el descriptor de archivo de cada socket.

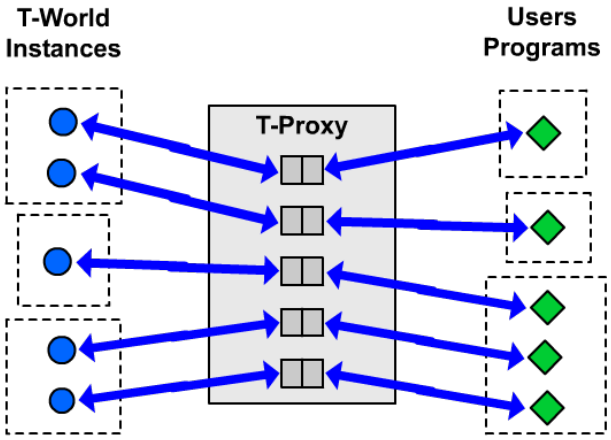


Figura 4.7: Múltiples conexiones simultaneas a través del proxy de T-World.

Cabe destacar que esta estructura le brinda al proxy la capacidad de soportar múltiples pares de conexiones, todos ellos conectados de manera concurrente al proxy. Como se observa en la Figura 4.7, dentro del proxy cada estructura mantiene un par de conexiónes. Los círculos representan WebSockets asociados a los robots 3D, y el cuadrado donde éstos se encuentran instancias de T-World. Los rombos representan sockets de usuarios asociados a los programas agente, cada cuadrado contenedor representa una instancia de programa de usuario.

4.2.5.2 Interacción Proxy/T-World

Como se mencionó anteriormente la comunicación entre el proxy y T-World se hace por medio del protocolo WebSocket. Esto implicó implementar manualmente dicho protocolo durante el desarrollo del proxy, siguiendo las especificaciones detalladas en su respectivo RFC [42]. Los mensajes que se envían utilizando el protocolo WebSocket tienen un formato específico el cual se denomina como "WebSocket Frame" o en este contexto simplemente Frame. Los datos que viajan dentro de Frames que se envían desde T-World hacia el proxy lo hacen enmascarados —esto es uno de los requerimientos del protocolo. Por esta razón cuando se recibe un Frame desde T-World el proxy debe desenmascarar el dato enviado (la percepción) y almacenarlo en el buf-

fer wtorBuffer para ser posteriormente reenviado hacia su correspondiente programa agente (ver figura 4.6). Posteriormente cuando el programa de usuario responde con la acción a realizar por el robot, ésta se guarda en el buffer rtowBuffer. El proxy obtiene dicha acción desde este buffer, la encapsula dentro de un WebSocket Frame y finalmente envía este Frame al WebSocket correspondiente dentro de T-World.

4.2.5.3 Interacción Programa Agente/Proxy

La interacción en este caso es mucho más simple, ya que no se realiza utilizando ningún protocolo en particular. La percepción almacenada en el buffer wtorBuffer se envía hacia el programa agente sin modificarla. Esto es, la secuencia de caracteres que se encuentran en el buffer se envían hacia el programa agente. Esta secuencia de caracteres tiene un formato especial (JSON o XML) por lo que el programa agente receptor primero la interpreta, luego típicamente realizará algún cómputo en base a la misma y posteriormente devolverá una acción —las acciones también son secuencias de caracteres. Una vez recibida la acción, el proxy la encapsula dentro de un WebSocket Frame y escribe los bytes del Frame en el buffer rtowBuffer. Acto seguido, este Frame se envía hacia el correspondiente WebSocket —lo que provoca que el robot correspondiente actúe dentro de la simulación 3D.

§4.3 Componentes de T-World

En esta sección se hace una breve descripción de los principales componentes de la plataforma, no desde una perspectiva de implementación como en la sección anterior, sino más bien desde una perspectiva de usuario.

§4.3.1 La Percepción

La percepción que es enviada hacia el programa agente está compuesta por dos partes: el *encabezado* (header) y los *datos* propiamente dichos (data). El usuario utiliza el valor contenido en el *encabezado* para saber cómo interpretar los valores de los *datos*, dependiendo de si se trata de información sobre el estado del ambiente o si se trata de un evento especial como un mensaje o la finalización de la simulación. La parte de datos se distribuye en tres compartimentos:

- *Información externa al agente* tales como la disposición de elementos físicos y de otros agentes en el mundo, tiempo transcurrido, etc.
- *Información relativa al agente receptor de la percepción*, como ser su nivel de energía, puntuación, posición actual, etc.
- *Conocimiento incorporado*, son los valores constantes establecidos por el usuario para ese ambiente en particular. Por ejemplo, las dimensiones del escenario, los valores y las reglas que definen cómo evoluciona el ambiente, etc.

En el Apéndice A se muestra un ejemplo concreto de una percepción, en donde se puede observar detalladamente cómo lucen y se estructuran cada uno de estos componentes.

§4.3.2 Las Acciones

Las acciones en T-World son secuencias de caracteres específicas[11]. El usuario debe enviar estas cadenas desde el programa agente para lograr que su robot se desempeñe durante la simulación. Las más importantes se describen a continuación:

[11] Siendo más precisos, cada acción tiene asociado una *expresión regular* que abarca una secuencia de caracteres, un número entero o formas híbridas más complejas.

- Desplazamiento en la grilla:
  - *"north"* —el agente se mueve una celda hacia arriba;
  - *"south"* —el agente se mueve una celda hacia abajo;
  - *"west"* —el agente se mueve una celda hacia la izquierda;
  - *"east"* —el agente se mueve una celda hacia la derecha.
- Restauración de la batería:
  - *"restore"* — restaura el nivel energético del agente, sólo puede ser aplicada cuando ya no se tiene energía. Antes de realizar esta acción el agente debe considerar su costo, ya que a cambio de la energía deberá ceder la mitad de su puntaje actual.
- Envío de mensajes a otros agentes: el contenido de un mensaje es una secuencia de caracteres que idealmente deberá estar convenientemente estructurada, por ejemplo, siguiendo los estándares de comunicación entre agentes como la establecida por FIPA [7]. Es importante destacar que mediante este servicio de mensajes se permite que haya comunicación dentro de sistemas multi-agente compuestos con agentes que han sido programados en diferentes lenguajes o creados bajo distintas plataformas de agentes.
  - *"message:<id>:<contenido>"* —envía un mensaje a un agente específico. <*id*> debe ser reemplazado por el identificador númerico del agente receptor y <*contenido*> por el contenido del mensaje.
  - *"team_message:<contenido>"* —difunde un mensaje a todos los miembros del equipo, <*contenido*> debe ser reemplazado por el contenido del mensaje.
- Existe una acción especial que es hacer *nada*, empleada por ejemplo para indicar que el agente necesita más tiempo para decidir la próxima acción, cuando no es

posible realizar ninguna otra acción ó cuando simplemente se necesita percibir sin actuar.

- *"none"* —acción nula.

§4.3.3 Programas Agente

En T-World, el usuario puede crear un programa agente realizando los siguientes pasos:

1. Asignar un nombre al programa agente, esto es necesario entre otras cosas, para que el usuario identifique a su programa agente dentro del sistema.

2. Determinar cómo se controlará el robot, las posibilidades son que sea el usuario humano mediante el teclado o un dispositivo móvil; ó algún algoritmo escrito en un lenguaje de programación.

3. Ingresar una descripción del programa agente, útil para ayudar a reconocer las características del programa agente como el algoritmo escogido para razonar.

Dependiendo del lenguaje de programación elegido en el paso (2) para codificar el programa agente, es la manera en la que se procede y las facilidades que se brindan en la plataforma.

4.3.3.1 Programa Agente en JavaScript

En este caso, T-World además de incorporar un editor de código para programar a los robots desde la misma plataforma, brinda una API con facilidades para desarrollarlos. Así, al finalizar la creación del programa agente se abre automáticamente un editor de código con un esqueleto básico para comenzar a programar. Dentro del editor se presentan cuatro secciones de código: la primera, "Agent Program", es para

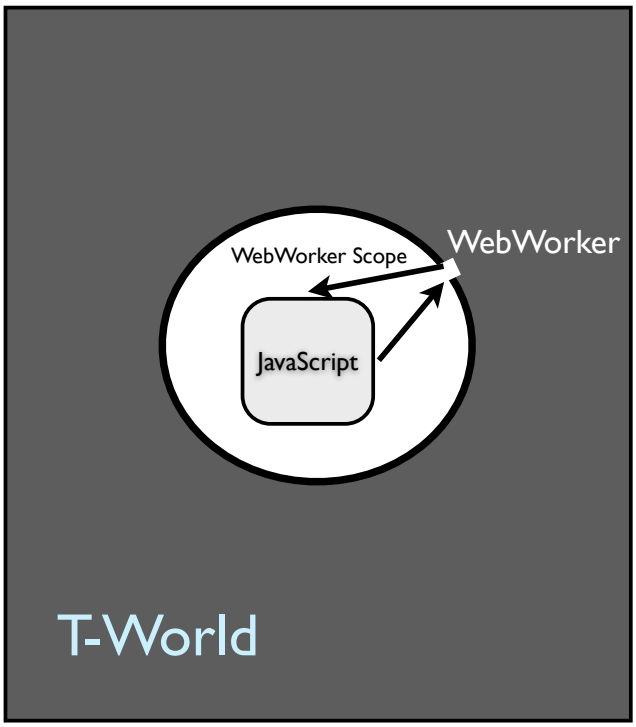


Figura 4.8: Programa Agente JavaScript en T-World

el algoritmo principal del programa agente. La segunda, "Global Scope", se usa para definir funciones y variables de alcance global. La tercera sección, "Start Event", es para manejar el evento de "inicio de simulación" que se dispara al comenzar la simulación. Esto es útil para inicializar variables o estructuras de datos que dependan de la primer percepción y que luego usará el agente. Generalmente se usa para obtener la configuración inicial del ambiente: condiciones de finalización, tamaño de la grilla, etc. La última sección, "Message Received Event", se debe implementar solamente cuando el agente requiere recibir mensajes, o sea, cuando integra un sistema multi-agente.

Una vez que el programa agente ha sido codificado, desde el mismo editor el usuario puede seleccionar el entorno de tarea en el que desea probarlo y comenzar la simulación del mismo.

Cuando el programa agente está codificado en JavaScript, internamente ejecuta como un *WebWorker* (véase Figura 4.8), es decir como un *thread* en segundo plano, independiente de la ejecución del código principal que atiende la simulación.

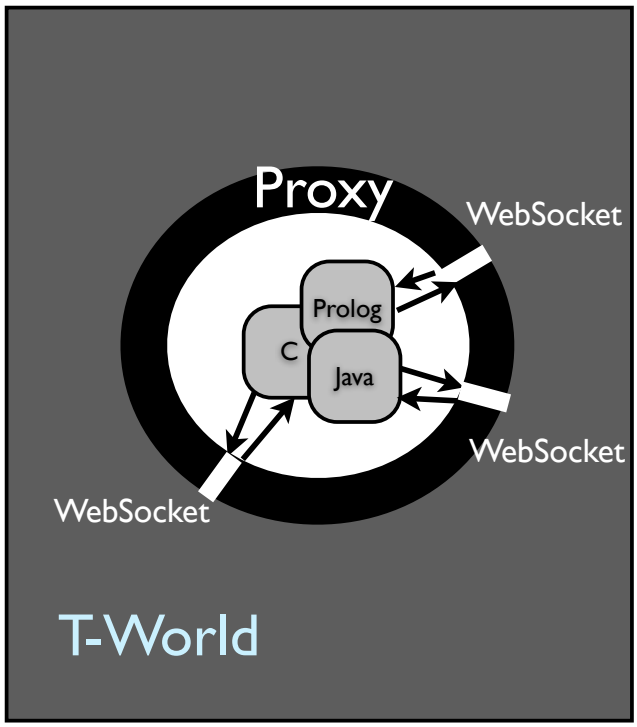


Figura 4.9: Distintos Programas Agente en T-World

4.3.3.2 Programa Agente en otro Lenguaje

La diferencia está en el segundo paso de la creación del programa, ya que en este caso no se edita el programa dentro de T-World como en el caso de JavaScript. Debido a que el usuario utilizará un lenguaje de programación externo al navegador web, se le debe indicar a T-World cómo enlazar el programa del usuario con su correspondiente programa agente. Esto último se lleva a cabo ingresando la dirección IP y el puerto donde ejecutará el Proxy, y finalmente, el *Magic String* que identifique al programa agente [12]. Notar que los programas agente escritos en lenguajes externos al navegador, tales como C o Java, presentan una diferencia significativa con los desarrollados en JavaScript en la manera en que se comunican con T-World, ya que éstos requieren necesariamente del uso del *proxy* para comunicarse con él (véase Figura 4.9).

Por esta razón, finalmente el usuario deberá conectarse al Proxy desde su programa agente, mediante un socket para recibir la percepción y devolver la respectiva acción

[12]Por defecto se le asigna el valor del nombre del programa agente.

a realizar por el agente. De este modo, el cuerpo del programa agente típicamente va a corresponderse con el siguiente algoritmo:

0. Crear socket.
1. Conectar socket a la dirección IP y puerto del proxy.
2. Enviar mensaje especial ''*CONNECT:<Valor de Magic String>*''.
3. Recibir percepción.
4. Según los datos percibidos y demás elementos, determinar la próxima acción.
5. Enviar dicha acción.
6. Recibir nueva percepción.
7. Si el encabezado de esta percepción es distinto a ''fin de simulación'' saltar a 4.

§4.4 Prueba de Concepto

Para cerrar este capítulo, en esta sección se desarrollará un experimento simple, a modo de prueba de concepto, haciendo uso de la plataforma construida. El experimento consiste en lo siguiente: se comienza creando un programa agente muy sencillo del tipo reflejo-simple y se evalúa su desempeño. Luego, a este mismo agente se le agrega la capacidad de mantener un estado interno, transformándolo en un agente basado en modelo y analizando el cambio que ésto produce en su desempeño. Finalmente, se realiza una última modificación agregándole la capacidad de planificar acciones futuras en base a objetivos, midiendo el impacto producido en su rendimiento.

1		25		#	26	53			56		58		#		74		76
	#	#	22	#		#	#	#	#	#	#		#	72	#	#	77
	#	20		32		2		65		63		61			81	#	
4	#		#	#		#	#	#	#	#	#	67	#	#		#	
	17		33		30	#	#	#	#	#	#			88		84	
	#		#	#		#	#	#	#	#	#	69	#	#	85	#	96
7	#	15			36		40		A	47			89			#	
	#	#	13	#		#	#	#	#	#	#	50	#	90	#	#	97
	10			#		42			45		52		#	91		93	

A = *agente*, # = *obstáculo*, < dígito > = *hueco*

Figura 4.10: Estado del ambiente construido para el experimento.

Figura 4.11: Visualización del ambiente creado.

§4.4.1 Experimento

El primer paso a realizar es la creación del entorno de trabajo en el que se van a estudiar los agentes. Por simplicidad se crea un entorno estático, determinístico, con un único agente, en donde no se utiliza la batería y los huecos se rellenan simplemente ubicando el robot arriba de la celda. Además, haciendo uso de la herramienta, al ambiente se lo diseña gráficamente ubicando 44 huecos, el agente y los obstáculos como se muestra en la Figura 4.10 —durante la simulación esto se visualiza como en la Figura 4.11. Finalmente, se configuran las condiciones de fin de simulación de manera que el agente tenga éxito si rellena la totalidad de los 44 huecos en menos de 2 minutos, o fracaso en caso contrario.

Una vez creado el entorno de trabajo se procede a codificar los programas agente en JavaScript utilizando el editor de código que brinda la plataforma. En los Apéndices B, C y D se muestran los fuentes de cada uno de éstos.

4.4.1.1 Agente Reflejo Simple

Su funcionamiento es muy sencillo, percibe las 4 celdas adyacentes, es decir, la que se encuentra al frente, detrás, a la izquierda y a la derecha. Si una de estas celdas es un hueco, elige la acción que lo desplace hasta ahí, sino simplemente elige una acción válida aleatoria. El algoritmo sería el siguiente:

```
Si la celda de adelante es un hueco, retornar acción adelante
Si la celda de atrás es un hueco, retornar acción atrás
Si la celda a la izquierda es un hueco, retornar acción izquierda
Si la celda a la derecha es un hueco, retornar acción derecha
Si ninguno de estos casos se cumple, retornar una acción válida
al azar
```

Se lanza este programa agente en un lote de 50 simulaciones y en todas se obtiene exactamente el mismo resultado: el tiempo se acababa sin que el agente pueda terminar de tapar los huecos. En promedio el agente alcanza a rellenar unos 22 huecos por simulación. Además, al monitorear el agente durante las simulaciones se observa que el comportamiento es mayormente aleatorio, muchas veces caminando una y otra vez sobre el mismo grupo de celdas en forma de bucle.

4.4.1.2 Agente Reflejo Basado en Modelo

Al programa agente anterior se le agrega un estado interno en el que se almacenan todas las celdas que se van percibiendo junto a un contador para cada celda vacía indicando la cantidad de veces que se ha pasado por cada una de ellas. El algoritmo se modifica para que no retorne una acción aleatoria, sino la acción que desplace el agente hacia la celda con el menor contador —esto hace que se le dé prioridad a las celdas que han sido visitadas con menor frecuencia. Además, para actualizar esta información el agente hace uso del siguiente "modelo del mundo":

- El ambiente no cambia por sí solo (es estático)
- La acción "derecho" me desplaza una celda hacia a la derecha
- La acción "izquierda" me desplaza una celda hacia a la izquierda
- La acción "adelante" me desplaza una celda hacia adelante
- La acción "atrás" me desplaza una celda hacia atrás

En base a esta información él determina qué celdas modificar en su estado interno al recibir una percepción. Por ejemplo, supóngase que el agente sabe que la última acción lo desplazó una celda hacia la derecha, si él percibe que la celda a su izquierda está vacía, entonces deduce que tiene que incrementar el contador de esta celda en su estado interno —para reflejar el hecho de que acaba pasar por esa celda.

Se lanza este programa agente en un lote de 50 simulaciones y en cada una de ellas el agente logra rellenar todos los huecos antes de los 2 minutos. En promedio le toma 90 segundos realizarlo. De la visualización del comportamiento del agente se observa una mejora significativa, ya que sus desplazamientos dejan de ser aleatorios y el robot tiende a explorar nuevos sitios tratando de no pasar por celdas ya visitadas.

4.4.1.3 Agente Basado en Objetivo

Por último, al programa agente se le añade la capacidad de elaborar planes en base a cumplir el objetivo "rellenar el hueco más cercano". Así, el agente primero arma un plan que consiste de la secuencia de acciones que lo lleven hasta el hueco más cercano, luego ejecuta cada una de estas acciones en orden hasta llegar al hueco en cuestión, y una vez allí vuelve a elaborar otro plan con un nuevo hueco. Este proceso se repite sucesivamente hasta que ya no queden huecos en la cuadrícula.

Se lanza este programa agente en el mismo lote de simulaciones que los anteriores y el agente logra tener éxito rellenando todos los huecos antes de los 2 minutos, tomándole en promedio 50 segundos hacerlo. Al monitorear el comportamiento del mismo se observa que a diferencia del agente anterior, no tiende a explorar la cuadrícula, directamente se dirige a cada uno de los huecos, rellenándolos uno a uno sin desperdiciar movimientos extras —similar al comportamiento que se esperaría observar cuando el agente es controlado por un ser humano.

§4.4.2 Generalización de los resultados

Analizando los resultados obtenidos en las ejecuciones, resumidos en la Tabla 4.1, se puede observar que el primer programa agente no logró tener éxito en la tarea dada, en los 2 minutos (120 segundos) sólo alcanzó a rellenar en promedio un 50 % de los huecos totales. En contraste los dos últimos agentes sí lograron completar con éxito

Tabla 4.1: Resumen de resultados.

Programa agente	Huecos	Tiempo(s)	Éxito
Reflejo simple	22	120	No
Basado en modelo	44	90	Sí
Basado en objetivo	44	50	Sí

la tarea, siendo el agente basado en objetivo el más eficiente ya que la realizó un 45 % más rápido que el basado en modelo.

En base a estas observaciones uno podría inferir que en un entorno de tarea con propiedades similares al empleado en este experimento podría no ser adecuada la utilización de agentes del tipo reflejo-simple ya que al parecer la tarea es lo suficientemente difícil como para poder resolverse solamente en base a la percepción actual. El agregado de una memoria para almacenar la información ya percibida y ciertos aspectos importantes para la toma de decisiones, como la cantidad de veces que se visitó una celda, puede mejorar considerablemente el comportamiento del agente y hasta incluso lograr que éste pueda completar la tarea. Finalmente, si se dispone de la suficiente información como para permitirle al agente elaborar y ejecutar planes, la tarea podría realizarse más eficientemente.

□

CAPÍTULO V

CONCLUSIONES Y TRABAJO FUTURO

Actualmente, los desarrolladores de sistemas informáticos producen aplicaciónes que deben hacer frente a requerimientos de creciente complejidad. Entre ellos, podemos mencionar la conexión directa de los sistemas con el ambiente en el que deben desempeñarse y la necesidad de comportamientos flexibles y adaptativos que permitan enfrentar aspectos tales como los recursos computacionales limitados, distribuídos, y heterogéneos, condiciones altamente cambiantes, necesidad de razonar y reaccionar de manera adecuada, etc.

En este contexto, los enfoques basados en *agentes inteligentes*, representan una alternativa interesante para abordar esta clase de problemas, como lo demuestra su creciente popularidad tanto en ámbitos académicos como industriales. Obviamente, cada tipo de problema podrá requerir de las técnicas y arquitecturas que mejor se adaptan a cada dominio particular. Se torna así fundamental, disponer de herramientas adecuadas que permitan diseñar, implementar, experimentar y analizar los resultados obtenidos con agentes inteligentes bajo distintas características de los ambientes y con distintas arquitecturas de los agentes. Esto es así tanto en el ámbito educativo como en el de investigación. En el primer caso, estas herramientas complementan y facilitan la comprensión de los conceptos adquiridos en la teoría. En el segundo, facilitan el análisis, verificación y/o reformulación de las hipótesis de investigación planteadas, proveyendo un soporte que brinda una mejor visualización del

comportamiento de los agentes, una mejor evaluación de los resultados obtenidos y una mayor flexibilidad para variar las condiciones experimentales de las simulaciones.

Los campos de prueba proveen una alternativa interesante a los aspectos arriba mencionados. Lamentablemente, para el caso particular de los agentes inteligentes, diversos campos de prueba como Tileworld y otras plataformas de agentes conocidas, suelen presentar algunos inconvenientes que dificultan su utilización tanto en investigación como en docencia. Entre estos inconvenientes están: limitaciones inherentes al entorno de trabajo del campo de pruebas, dificultades para portarlos a diferentes plataformas, restricciones sobre el lenguaje de programación de los agentes y una interfaz poco atractiva. Considerando este tipo de inconvenientes, este trabajo final propuso como objetivo general, y alcanzó con éxito, el desarrollo de un nuevo entorno de trabajo gráfico, flexible y portable, denominado T-World, que logró cumplir diversos objetivos particulares planteados en el capítulo de introducción de este informe, algunos de esos logros se detallan a continuación.

En primer lugar, se proveyó una interfaz gráfica que responde a la mayoría de los requerimientos de realismo y entretenimiento que es posible esperar hoy en día en un juego de computadoras. Este no es un aspecto secundario, si consideramos que T-World puede cumplir un rol importante en la enseñanza de diversos aspectos de la Inteligencia Artificial en general y de las arquitecturas de agente en particular. En este caso, consideramos que disponer de una herramienta que se asemeje al tipo de aplicaciones con los cuales los alumnos suelen estar muy familiarizados, favorecería o al menos motivaría en mayor grado a su uso y aprendizaje.

La *portabilidad* y *flexibilidad* en T-World son otros dos aspectos que merecen destacarse. Como se pudo observar en el Capítulo 4, en T-World es muy sencillo codificar el motor de razonamiento de los agentes en una variedad de lenguajes de programación. Además, la posibilidad de acceso a través de Internet lo convierten en una herramienta sencilla y fácilmente accesible para cualquier usuario utilizando un

navegador estándard desde sus dispositivos móviles o computadoras personales.

También se mostró que en T-World es muy sencillo diseñar distintas clases de entornos para evaluar las habilidades de los agentes, crear los motores de razonamiento para distintos tipos de agentes y evaluar el desempeño de diferentes clases de agentes en distintos tipos de ambientes. Estas facilidades fueron ejemplificadas con la implementación de un agente reflejo simple, un agente reflejo simple basado en modelo y un agente basado en objetivos, cuyos detalles de implementación están disponibles en los apéndices de este informe. Para el investigador avanzado, por otra parte, se dan diversas facilidades para generar las estadísticas del desempeño de los agentes y para lograr una evaluación de "gránulo más fino" de dicho desempeño.

Por otra parte, a los objetivos planteados originalmente podemos agregar el cumplimento de otros objetivos que se vinculan a los requerimientos generales definidos para los campos de prueba (ver Capítulo 3). En este sentido, es importante observar que T-World satisface distintos criterios de investigación deseables para un campo de prueba, tales como la ocurrencia de *eventos exógenos*, el realismo en el *paso del tiempo*, la incoproración de *características complejas* del mundo, la definición de *sensores* y *actuadores* "realistas", la flexibilidad en la especificación de la *medida de éxito* y el soporte para problemas *multi-agente*. Respecto a los requerimientos de ingeniería, se puede decir que T-World provee una interfaz "limpia", con un modelo bien definido del tiempo y buen soporte para la experimentación. Otras propiedades deseables que cumple T-World tienen que ver con ser una pieza de software independiente, de libre acceso, autocontenida y ejecutable. También se puede mencionar que el hecho de adoptarse para este informe y para el sistema implementado, una terminología universalmente aceptada en la bibliografía del área [59](AIMA), facilita su comprensión y discusión por parte de investigadores, docentes y alumnos.

Hasta el momento se han detallado diversos objetivos vinculados a las propiedades del producto obtenido (T-World) y de su uso potencial por parte de investigadores,

docentes y estudiantes en el área de los agentes inteligentes. Nos centraremos ahora en el cumplimiento de los objetivos alcanzados como Trabajo Final de la Licenciatura en Ciencias de la Computación.

A nuestro entender, dos objetivos principales a lograrse en un trabajo final de Licenciatura son la aplicación e integración de conceptos adquiridos a lo largo de la carrera, y demostrar la capacidad para investigar, comparar e incorporar nuevos conceptos que no forman parte del currículo de la carrera pero que deben ser adquiridos debido a la naturaleza del problema específico abordado. Este último aspecto es crucial si consideramos lo dinámico y cambiante que es nuestra disciplina que demanda un continuo aprendizaje y actualización de nuevas tecnologías, paradigmas, etc.

Respecto al primer punto, este trabajo hizo un uso intenso de conceptos adquiridos en las asignaturas de "Inteligencia Artificial" y la optativa de "Agentes y Sistemas Multi-agente". También fueron fundamentales los conceptos de lenguajes de programación adquiridos en "Análisis Comparativo de Lenguajes", de estructuras de datos y algoritmos adquiridos en "Estructuras de Datos y Algoritmos" y "Organización de Archivos y Bases de Datos I", y los conceptos de sistemas distribuidos y redes adquiridos en "Sistemas Operativos y Redes" y "Sistemas Distribuidos y Paralelismo".

Este trabajo involucró además el aprendizaje y utilización de nuevos conceptos, técnicas, lenguajes y herramientas que no habían sido abordados durante el cursado normal de las asignaturas del plan. Entre otras podemos mencionar, los lenguajes multi-paradigma como JavaScript, el framework *AngularJS*, el lenguaje *Cascading Style Sheets*, el framework *Bootstrap*, la modelización individual 3D de los objetos, sus texturas y animaciones, la API *Web Graphics Library*, el motor 3D *CopperLicht*, el formato *JSON* y el protocolo *WebSocket* de HTML5.

§5.1 Contribuciones

Como resultado de este trabajo, y más allá de los objetivos cumplidos que se detallaron en la sección previa, algunos de los resultados parciales fueron publicados en:

1) "*T-World: un entorno gráfico, flexible y portable para la enseñanza e investigación de agentes inteligentes*". Sergio Burdisso, Guillermo Aguirre y Marcelo Luis Errecalde. Anales del XX Congreso Argentino de Ciencias de la Computación (CACIC 2014), Universidad Nacional de la Matanza, San Justo, Buenos Aires, Argentina, 20 al 24 de Octubre de 2014. Pgs. 83 - 92. ISBN 978-987-3806-05-6.

En este trabajo se describen las componentes principales de T-World y su presentación fue premiada como uno de los mejores trabajos presentados en el área.

2) "*La plataforma web T-World*". Sergio Burdisso, Guillermo Aguirre y Marcelo Luis Errecalde. Trabajo presentado en la categoría de Trabajos Estudiantiles del 2do. Congreso Nacional de Ingeniería Informática/Sistemas de Información (CoNaIISI 2014). San Luis. UNSL.

En este caso se describe con más detalle el funcionamiento del proxy implementado y el artículo correspondiente será enviado próximamente a un Congreso relevante en el área.

3) Mejoras realizadas en el rendimiento de *CopperLitch*, que fueron incorporadas posteriormente en la biblioteca oficial, incorporando además a T-World en su página oficial (`http://www.ambiera.com/copperlicht/`).

§5.2 Trabajo Futuro

En este trabajo se han realizado varias contribuciones originales que deben ser aún publicadas en Congresos Nacionales e Internacionales para lograr una mayor difusión y uso de la plataforma T-World.

En particular, los detalles de implementación y funcionamiento del proxy que ya fueran descriptos en la publicación 2 de la sección anterior, serán ampliados y enviados a una conferencia Nacional próximamente.

T-World será utilizado además, como herramienta de soporte en distintas unidades de la asignatura Inteligencia Artificial de la Licenciatura en Ciencias de la Computación. Como resultado de estas experiencias, se planea publicar un trabajo donde se reporten las ventajas y desventajas que muestra T-World como herramienta de soporte pedagógico en los diversos temas de la asignatura.

Actualmente, una becaria de postgrado realiza en el LIDIC su tesis doctoral sobre arquitecturas BDI, procesos Markov y tecnologías de acuerdo, y utiliza como tarea de referencia al Tileworld. Integraremos próximamente sus módulos de decisión al T-World a los fines de experimentar con distintas configuraciones de los ambientes y observar el comportamiento de las distintas arquitecturas.

Una vez realizada las dos etapas previas, se planea la publicación de un artículo internacional donde se detallen todos los aspectos principales de T-World y los resultados obtenidos, con el fin de que sirvan de referencia para aquellos docentes e investigadores interesados en trabajar con esta herramienta.

□

"We can only see a short distance ahead,

but we can see plenty there that needs to be done."

—*A. M. Turing*

(Computing Machinery and Intelligence, 1950)

APÉNDICES

APÉNDICE A

Ejemplo de una Percepción en JSON

En este apéndice se muestra un ejemplo de una percepción en JSON capturada desde una simulación. En la Figura A.1 se puede observar una captura de pantalla de dicha simulación, en ella se muestra la disposición de los diferentes elementos y cuatro agentes con identificadores numéricos 0,1,2 y 3. El entorno de trabajo está compuesto por dos equipos, el equipo "verde" y el "naranja", cada uno de ellos está conformado por dos robots.

En §4.3.1 se describe de manera general la estructura de la percepción. En contraste, aquí se muestra una percepción en concreto, donde con más detalle la estructura es la siguiente:

- *header*: es el encabezado;
- *data*: son los datos propiamente dichos.
  - *environment*: contiene la información externa al agente;
  - *agent*: contiene la información relativa al agente receptor de la percepción;
  - *builtin_knowledge*: contiene el "conocimiento incorporado", e.d. los valores constantes de configuración del ambiente fijados por el experimentador.

Figura A.1: Instantánea de simulación utilizada para capturar la percepción.

A continuación se muestra dicha percepción, la cual fue enviada hacia el agente con identificador 0 y capturada a los 52 segundos de simulación.

```
{
  "header":"ready_for_next_action",
  "data":{
    "environment":{
      "grid":[
        [" ","T","#","A","A"],
        ["T"," "," "," "," "],
        [" "," ","A"," "," "],
        ["T", 3 ," "," ","C"],
        ["C", 3 ,"A"," ","#"]
      ],
      "time":52,
      "battery_chargers":[
        {"row":4,"column":0},
        {"row":3,"column":4}
```

```
    ],
    "agents":[
      {
          "id":0,
          "team_id":0,
          "location":{"row":0,"column":3},
          "score":0,
          "battery":1000
      },
      {
          "id":1,
          "team_id":0,
          "location":{"row":0,"column":4},
          "score":0,
          "battery":1000
      },
      {
          "id":2,
          "team_id":1,
          "location":{"row":2,"column":1},
          "score":0,
          "battery":1000
      },
      {
          "id":3,
          "team_id":1,
          "location":{"row":4,"column":1},
          "score":0,
          "battery":1000
      }
    ],
    "holes":[
      {"id":3,"cells":[{"row":4,"column":1},{"row":3,"column":1}],"
```

```
          size":3,"value":30,"time_elapsed":8,"lifetime_left":6}
      ],
      "tiles":[
        {"row":0,"column":1},
        {"row":1,"column":0},
        {"row":3,"column":0}
      ],
      "obstacles":[
        {"row":0,"column":2,"time_elapsed":1,"lifetime_left":12},
        {"row":4,"column":4,"time_elapsed":1,"lifetime_left":7}
      ]
    },
    "agent":{
      "id":0,
      "team_id":0,
      "location":{"row":0,"column":3},
      "score":0,
      "battery":1000,
      "stats":{
        "good_moves":0,
        "bad_moves":0,
        "filled_cells":0,
        "filled_holes":0,
        "battery_used":0,
        "battery_recharge":0,
        "battery_restore":0,
        "total_score":0
      }
    },
    "builtin_knowledge":{
      "grid_total_rows":5,
      "grid_total_columns":5,
      "teams":[
```

```
        {"id":0,"leader":0,"members":[0,1]},
        {"id":1,"leader":2,"members":[2,3]}
      ],
      "end":{
        "neutral":{
          "time":300,
          "good_moves":4
        },
        "success":{
          "agents_location":[{"row":0,"column":0}],
          "filled_holes":1,
          "filled_cells":6,
          "score":100
        },
        "failure":{
          "bad_moves":6,
          "battery_used":8,
          "battery_recharge":6,
          "battery_restore":6
        }
      },
      "costs":{
        "good_move":350,
        "bad_move":1000,
        "filled_hole":1000,
        "battery":{
          "bad_move":5,
          "good_move":20,
          "slide_tile":10
        }
      },
      "probability":{
        "holes_size":{
```

```
            "range":[1,3],
            "prob":["0.334","0.333","0.333"]
        },
        "num_holes":{
            "range":[2,3],
            "prob":["0.500","0.500"]
        },
        "num_obstacles":{
            "range":[1,2],
            "prob":["0.500","0.500"]
        },
        "difficulty":{
                "range":[0,0],
                "prob":[]
        },
        "dynamism":{
            "range":[6,13],
            "prob":["0.125","0.125","0.125","0.125","0.125","0.125",
               "0.125","0.125"]
        },
        "hostility":{
            "range":[1,13],
            "prob":["0.077","0.077","0.077","0.077","0.077","0.077",
               "0.077","0.077","0.077","0.077","0.077","0.077","0.076"]
        },
        "model_of_action":{
          "intended":0.7,
          "left":0.1,
          "right":0.1,
          "backward":0.1,
          "refuses":0
        }
      }
```

```
145         }
146     }
147 }
```

APÉNDICE B

Programa Agente Reflejo Simple en JavaScript

A continuación se da el código fuente JavaScript de un programa agente de tipo reflejo-simple codificado utilizando el editor de código provisto por la plataforma.

Sección "Agent Program":

```
/*
  SIMPLE REFLEX AGENT
*/
function AGENT_PROGRAM(percept){
  // generate an abstracted description of the current state of the
     world
  state = INTERPRET_INPUT(percept);
  // see what rule matches the given state description
  rule = RULE_MATCH(state, rules);
  // get the action of the matched rule
  action = rule.ACTION;
  // and return it
  $return(action);
}
```

Sección "Global Scope":

```
1
2  var state; // the agent's current conception of the world state
3  var rules = new Array(4); // a set of condition-action rules
4
5  /*
6    INTERPRET_INPUT function generates an abstracted of the current state
7    from the percept
8  */
9  function INTERPRET_INPUT(percept){
10   var state = new Array(4);
11   var loc = percept.agent.location;
12
13   state[_NORTH] = $isValidMove(percept, _NORTH);
14   state[_SOUTH] = $isValidMove(percept, _SOUTH);
15   state[_WEST ] = $isValidMove(percept, _WEST );
16   state[_EAST ] = $isValidMove(percept, _EAST );
17
18   if (state[_NORTH] && $isTile(percept, loc.row-1, loc.column))
19     state[_NORTH] = _GRID_CELL.TILE;
20
21   if (state[_SOUTH] && $isTile(percept, loc.row+1, loc.column))
22     state[_SOUTH] = _GRID_CELL.TILE;
23
24   if (state[_WEST] && $isTile(percept, loc.row, loc.column-1))
25     state[_WEST] = _GRID_CELL.TILE;
26
27   if (state[_EAST] && $isTile(percept, loc.row, loc.column+1))
28     state[_EAST] = _GRID_CELL.TILE;
29
30   return state;
31 }
32
33 /*
```

```
34   RULE_MATCH function returns the first rule in the set of rules that
         matches
35   the given state description, or a random rule in case of no matches
36 */
37 function RULE_MATCH(state, rules){
38   var r_rules = [];
39
40   for(var i=rules.length;i--;)
41     if (rules[i].match(state))
42       return rules[i];
43     else
44     if (state[i])
45       r_rules.push(rules[i]);
46
47   return r_rules[random(r_rules.length)];
48 }
49
50 /*
51   rules: a set of condition-action rules
52 */
53 rules[_NORTH] = {
54     ACTION: _NORTH,
55     match: function(state){return state[_NORTH] == _GRID_CELL.TILE}
56   };
57
58 rules[_SOUTH] = {
59     ACTION: _SOUTH,
60     match: function(state){return state[_SOUTH] == _GRID_CELL.TILE}
61   };
62
63 rules[_EAST] = {
64     ACTION: _EAST,
65     match: function(state){return state[_EAST] == _GRID_CELL.TILE}
```

```
66    };
67
68  rules[_WEST] = {
69      ACTION: _WEST,
70      match: function(state){return state[_WEST] == _GRID_CELL.TILE}
71    };
```

APÉNDICE C

Programa Agente Basado en Modelo en JavaScript

A continuación se da el código fuente JavaScript de un programa agente de tipo "reflejo basado en modelo", codificado utilizando el editor provisto por la plataforma.

Sección "Agent Program":

```
/*
  MODEL-BASED REFLEX AGENT
*/
function AGENT_PROGRAM(percept){var rule;
  // create a new, updated, internal state
  state = UPDATE_STATE(state, action, percept, model);
  // see what rule matches the current internal state
  rule = RULE_MATCH(state, rules);
  // get the action of the matched rule
  action = rule.ACTION;
  // and return it
  $return(action);
}
```

Sección "Global Scope":

```

```

```
2  var model = {}; // a description of how the next state depends on the
       current state and action
3  var rules = new Array(8); // a set of condition-action rules
4  var action = _NONE; // the most recent action, initially none
5  var state = { // the agent's current conception of the world state
6                grid:[
7                  ["","",""],
8                  ["","",""],
9                  ["","",""]
10               ],
11               loc:{row:0, column:0} //percept matrix origin
12             };
13 var _STATE_HOLE = 'H';
14
15 /*
16   UPDATE_STATE function is responsable for creating a new internal state
17 */
18 function UPDATE_STATE(state, action, percept, model){
19   var grid, loc = state.loc;
20
21   // 1) updating the internal state structure according to what my
         action was
22   // and the new percept object and making room for the new data
         provided by the percept
23   model.apply_action(action, state);
24
25   // 2) updating the cells value of our internal state grid according to
         our current perception
26   for (var r=0, percept_cell, state_cell; r < 3; ++r)
27     for (var c=0; c < 3; ++c){
28       percept_cell = percept.environment.grid[r][c];
29       state_cell = state.grid[loc.row+r][loc.column+c];
30
```

```
31        if (state.grid[loc.row+r].length-1 < loc.column+c)
32          state.grid[loc.row+r].length = loc.column+c+1;
33
34        if (percept_cell != _GRID_CELL.AGENT){
35          if (!state_cell && state_cell!==0)
36            state.grid[loc.row+r][loc.column+c] = isHoleCell(percept_cell)
                ?
37                                                  _STATE_HOLE :
38                                                  (percept_cell ==
                                                      _GRID_CELL.EMPTY?
                                                      0 : percept_cell);
39        }else{
40          if (!state_cell || state_cell == _STATE_HOLE)
41            state.grid[loc.row+r][loc.column+c] = 0;
42
43          state.grid[loc.row+r][loc.column+c]++;
44        }
45      }
46
47    // 3) printing the new internal state
48    grid = copy(state.grid);
49    grid[loc.row+1][loc.column+1] = _GRID_CELL.AGENT;
50    $printMatrix(grid);
51
52    return state;
53  }
54
55  /*
56    RULE_MATCH function returns the first rule in the set of rules that
57    matches the given state description.
58  */
59  function RULE_MATCH(state, rules){
60    for (var i=0; i < rules.length; ++i)
```

```
61      if (rules[i].match(state))
62        return rules[i];
63  }
64
65  /*
66    model: a description of how the next state depends on the current
          state and action
67  */
68  model.apply_action = function(action, state){ var r, loc = state.loc;
69    // update the internal state structure according to what my action was
70    // and the new percept object;
71    // if needed, make room in the grid for the new data provided by the
          percept
72    switch(action){
73      case _NORTH:
74        state.loc.row--;
75        if (state.loc.row < 0){
76          state.loc.row=0;
77          state.grid.unshift(["","",""]); // adding new row on demand at
                the top
78        }
79        break;
80      case _SOUTH:
81        state.loc.row++;
82        if (state.grid.length < state.loc.row+3)
83          state.grid.push(["","",""]); // adding new row on demand at the
                bottom
84        break;
85      case _WEST:
86        state.loc.column--;
87        if (state.loc.column < 0){
88          state.loc.column=0;
89          for (r=0; r < state.grid.length; ++r)
```

```
90            state.grid[r].unshift(""); // adding new column on demand on
                 the left side
91          }
92          break;
93        case _EAST:
94          state.loc.column++;
95          for (r=loc.row; r < loc.row+3; ++r)
96            if (state.grid[r].length < loc.row.column+3)
97              state.grid[r].push(""); // adding new column on demand on the
                 right side
98          break;
99      }
100 };
101
102 /*
103   rules: a set of condition-action rules
104 */
105 rules[0] = {
106   ACTION: _NORTH,
107   match : function(state){
108     return  state.grid[(state.loc.row+1)-1][state.loc.column+1] ==
          _STATE_HOLE;
109   }
110 };
111 rules[1] = {
112   ACTION: _SOUTH,
113   match : function(state){
114     return  state.grid[(state.loc.row+1)+1][state.loc.column+1] ==
          _STATE_HOLE;
115   }
116 };
117 rules[2] = {
118   ACTION: _WEST,
```

```
119   match : function(state){
120     return  state.grid[state.loc.row+1][(state.loc.column+1)-1] ==
          _STATE_HOLE;
121   }
122 };
123 rules[3] = {
124   ACTION: _EAST,
125   match : function(state){
126     return  state.grid[state.loc.row+1][(state.loc.column+1)+1] ==
          _STATE_HOLE;
127   }
128 };
129 rules[4] = {
130   ACTION: _NORTH,
131   match : function(state){ return  findCellWithTheLeastNumber(state) ==
        _NORTH; }
132 };
133 rules[5] = {
134   ACTION: _SOUTH,
135   match : function(state){ return  findCellWithTheLeastNumber(state) ==
        _SOUTH; }
136 };
137 rules[6] = {
138   ACTION: _WEST,
139   match : function(state){ return  findCellWithTheLeastNumber(state) ==
        _WEST; }
140 };
141 rules[7] = {
142   ACTION: _EAST,
143   match : function(state){ return  findCellWithTheLeastNumber(state) ==
        _EAST; }
144 };
145
```

```
146
147
148 /*
149   AUXILIARY STUFF ZONE
150 */
151
152 // returns the action that takes the agent to the cell with the least
      number
153 // (note: this number represent the number of times the agent has been
      located at a certain cell)
154 function findCellWithTheLeastNumber(state){
155   var a_r = state.loc.row+1;
156   var a_c = state.loc.column+1;
157   var cells = [];
158
159   if (isNumber(state.grid[a_r-1][a_c]))
160     cells.push({ACTION: _NORTH, val: state.grid[a_r-1][a_c]});
161   if (isNumber(state.grid[a_r+1][a_c]))
162     cells.push({ACTION: _SOUTH, val: state.grid[a_r+1][a_c]});
163   if (isNumber(state.grid[a_r][a_c-1]))
164     cells.push({ACTION: _WEST, val: state.grid[a_r][a_c-1]});
165   if (isNumber(state.grid[a_r][a_c+1]))
166     cells.push({ACTION: _EAST, val: state.grid[a_r][a_c+1]});
167
168   //sorting the cells (ascending)
169   cells.sort(function(a,b){return a.val - b.val});
170
171   // returing action that takes the agent to the cell with the least
        number
172   return cells[0].ACTION;
173 }
174
175 function isHoleCell(cell){return isNumber(cell);}
```

APÉNDICE D

Programa Agente Basado en Objetivo en JavaScript

A continuación se da el código fuente JavaScript de un programa agente de tipo 'basado en objetivo", codificado utilizando el editor de código brindado por la plataforma construida.

Sección "Agent Program":

```
/*
  SIMPLE PROBLEM-SOLVING AGENT
*/
function AGENT_PROGRAM(percept){var action;
  state = UPDATE_STATE(state, percept);

  // if you don't have a solution
  // (i.e. if the action sequence that solve the problem is empty)
  if (seq.empty()){
    // Formulate a new goal and
    goal = FORMULATE_GOAL(state);
    // problem to solve, then
```

```
13      problem = FORMULATE_PROBLEM(state, goal);
14      // search for a solution (an action sequence)
15      seq = SEARCH(problem, _SEARCH_ALGORITHM.A_STAR);
16      if (seq.empty()) $return(_ACTION.NONE);
17    }
18
19    // use the solution to guide your actions
20    // doing whatever the solution recommends as the next thing to do
21    action = FIRST(seq);
22    //and then remove that step from the sequence
23    seq = REST(seq);
24
25    $return(action);
26  }
```

Sección “Global Scope”:

```
1
2   var seq = []; // an action sequence, initially empty
3   var state; // some description of the current world state
4   var goal = null; // a goal, initially null
5   var problem; // a problem formulation
6
7   function UPDATE_STATE(state, percept){return percept;}
8   function FIRST(seq){if (!seq.empty()) return seq[0]; else return _ACTION
        .NONE;}
9   function REST(seq){seq.shift(); return seq;}
10  function FORMULATE_PROBLEM(state, goal){ return {state: state, goal:goal
        } }
11
12  function FORMULATE_GOAL(state){
13    if (goal === null)
14      goal = {
15        agent:{ stats:{ filled_holes:0 } }
```

```
16      };
17
18    goal.agent.stats.filled_holes++;
19
20    return goal;
21  }
22
23  function SEARCH(problem, strategy){ var illustrated = true;//true;
24    switch(strategy){
25      case _SEARCH_ALGORITHM.BFS:    return $breadthFirstSearch(problem.
          state, problem.goal, illustrated);
26      case _SEARCH_ALGORITHM.DFS:    return $depthFirstSearch(problem.
          state, problem.goal, illustrated);
27      case _SEARCH_ALGORITHM.IDFS:   return $iterativeDepthFirstSearch(
          problem.state, problem.goal, illustrated);
28      case _SEARCH_ALGORITHM.A_STAR: return $aStarBestFirstSearch(problem.
          state, problem.goal, illustrated);
29      case _SEARCH_ALGORITHM.GREEDY: return $greedyBestFirstSearch(problem
          .state, problem.goal, illustrated);
30    }
31  }
```

BIBLIOGRAFÍA

BIBLIOGRAFÍA

[1] Dan Chen A, Georgios K. Theodoropoulos A, Stephen J. Turner B, Wentong Cai B, Robert Minson A, and Yi Zhang A. Large scale agent-based simulation on the grid. *Future Generation Computer Systems 24*, pages 658–671, 2008.

[2] P. Agre and D. Chapman. An implementation of a theory of activity. In *Proceedings of the Sixth National Conference on Artificial Intelligence*, pages 268–272, Menlo Park, Calif: American Association for Artificial Intelligence, 1987.

[3] B. Al-Badr and S. Hanks. Critiquing the tileworld: Agent architectures, planning benchmarks, and experimental methodology. Technical report, TR 91-11-01, 1991.

[4] Natasha Alechina and Brian Logan. Verifying bounds on deliberation time in multi-agent systems. In *In EUMAS*, pages 25–34, 2005.

[5] Ambiera. Copperlicht - free and fast webgl javascript 3d engine with world editor. `http://www.ambiera.com/copperlicht/index.html`.

[6] Jeremy Ashkenas. Backbone.js. `http://backbonejs.org`.

[7] Fabio Luigi Bellifemine, Giovanni Caire, and Dominic Greenwood. *Developing Multi-Agent Systems with JADE*. John Wiley & Sons, NJ, April 2007.

[8] Hamid R. Berenji and David Vengerov. Cooperation and coordination between fuzzy reinforcement learning agents in continuous state partially observable markov decision processes. In *in Proceedings of the 8th IEEE International Conference onFuzzy Systems (FUZZ-IEEE99)*, pages 621–627, 1999.

[9] Hamid R. Berenji and David Vengerov. Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In *In Proceedings of 9th IEEE International Conference on Fuzzy Systems*, 2000.

[10] Mark Boddy and Thomas Dean. Solving time-dependent planning problems. *Proceedings IJCAI. AAAI*, August, 1989.

[11] J.; Bratman, M.; Israel, M. Martha E. Pollack Pollack, and Paul R. Cohen. Plans and resource-bounded practical reasoning. *Computational Intelligence*, (4):349–355, 1988.

[12] Mao Chen, Klaus Dorer, Ehsan Foroughi, Fredrick Heintz, ZhanXiang Huang, Spiros Kapetanakis, Kostas Kostiadis, Johan Kummeneje, Jan Murray, Itsuki Noda, Oliver Obst, Pat Riley, Timo Steffens, Yi Wang, and Xiang Yin. *Users Manual: RoboCup Soccer Server — for Soccer Server Version 7.07 and Later*. The RoboCup Federation, February 2003.

[13] K.W. Choy, Adrian Hopgood, L. Nolle, and B.C. O'Neill. Implementation of a tileworld testbed on a distributed blackboard system. In G. Horton, editor, *18th European Simulation Multiconference (ESM2004)*, pages 129–135, Magdeburg, Germany, June 2004.

[14] L. Chrisman and R. Simmons. Senseful planning: Focusing perceptual attention. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 756–761, Menlo Park, Calif: American Association for Artificial Intelligence, 1991.

[15] T. Dean and M. Boddy. An analysis of time-dependent planning. In *Proceedings of the Seventh National Conference on Artificial Intelligence*, pages 49–52, Menlo Park, Calif: American Association for Artificial Intelligence, 1988.

[16] T. Dean and M. Boddy. Practical temporal projection. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 158–163, Menlo Park, Calif: American Association for Artificial Intelligence, 1990.

[17] ECMA. Proposed ECMAScript 4th Edition Language Overview. `http://www.ecma-international.org/activities/Languages/Language%20overview.pdf`, Consultado 4 Marzo 2015.

[18] ECMA-404. Introducing json. `http://json.org/`.

[19] Martha E. Pollack Eithan Ephrati and Sigalit Urt. Deriving multi-agent coordination through filtering strategies. *Department of Computer Science and Intelligent Systems Programt*, 1994.

[20] Sean P. Engelson and Niklas Bertani. Ars magna: The abstract robot simulator manual. Technical report, Department of Computer Science, Yale University, October 1992.

[21] WEBGL GAME ENGINES. Copperlicht. `http://www.webgl-game-engines.com/copperlicht.html`.

[22] Marcelo Luis Errecalde. *Aprendizaje Basado en Múltiples Fuentes de Experiencia*. PhD thesis, Departamento de Ciencias e Ingeniería de la Computación, Universidad Nacional del Sur, 2004.

[23] R. J. Firby. *Adaptative Execution in Complex Dynamic Worlds*. PhD thesis, Dept. of Computer Science, Yale Univ., 1989.

[24] R. J. Firby and S Hanks. A simulator for mobile robot planning. In *Proceedings of the DARPA Knowledge-Based Planning Workshop, 23123-7*, In . Washington, D.C.: Defense Advanced Research Projects Agency. 1987.

[25] Paul A. Fishwick. *Handbook of Dynamic System Modeling*. Chapman and Hall/CRC; 1 edition, June 1, 2007.

[26] Paula Flanagan, David; Ferguson. *JavaScript: The Definitive Guide*. O'Reilly & Associates. ISBN 0-596-10199-6, 5th edition, 2006.

[27] Simon Parsons Gerardo Simari. On approximating the best decision for an autonomous agent. In *Proceedings of the Sixth Workshop on Game Theoretic and Decision Theoretic Agents*, pages 91–100, 2004.

[28] GitHub. "search - stars:>1". `https://github.com/search?l=&o=desc&q=stars%3A%3E1&ref=advsearch&s=stars&type=Repositories`. Consultado 5 Marzo 2015.

[29] GNU. Affero general public license 3.0. `http://www.gnu.org/licenses/agpl-3.0.html`.

[30] Abdelkader Gouach and Fabien Michel. Towards a unified view of the environment(s) within multi-agent systems. In *In this issue*, pages 423–432, 2005.

[31] M. Greenberg and L. Westbrook. The phoenix test bed. Technical report, COINS TR 9019, Dept. of Computer and Information Science, Univ. of Massachusetts, 1990.

[32] Kieran Greer. A metric for modelling and measuring complex behavioural systems, 2011.

[33] Khronos Group. Webgl - user contributions. `https://www.khronos.org/webgl/wiki/User_Contributions#CopperLicht`.

[34] Khronos Group. Webgl specifications. `https://www.khronos.org/registry/webgl/specs/latest/`.

[35] P. Haddawy and S. Hanks. Utility models for goal-directed decision-theoretic planners. Technical report, Dept. of Computer Science and Engineering, Univ. of Washington, 1993.

[36] P. Haddawy and S. Hanks. Issues in decision-theoretic planning: Symbolic goals and numeric utilities. *In workshop on Innovative Approaches to Planning, Scheduling, and Control*, November 1990.

[37] S. Hanks. *Projecting plans for uncertain worlds*. PhD thesis, Dept. of Computer Science, Yale Univ., 1990.

[38] S. Hanks and D. McDermott. Modeling a dynamic and uncertain world i: Symbolic and probabilistic reasoning about change. *Artificial Intelligence*, 65(2), 1994.

[39] Steve Hanks, Martha E. Pollack, and Paul R. Cohen. Benchmarks, test beds, controlled experimentation, and the design of agent architectures. *AI Magazine*, 14(4):17–42, 1993.

[40] Scott Hanselman. JavaScript is Assembly Language for the Web: Part 2 - Madness or just Insanity? `http://www.hanselman.com/blog/JavaScriptIsAssemblyLanguageForTheWebPart2MadnessOrJustInsanity.aspx`, July 19, 2011.

[41] D. Hart and P. Cohen. Phoenix: A test bed for shared planning research. In *Proceedings of the NASA Ames Workshop on Benchmarks and Metrics*, In , Moffett Field, California, June 1990.

[42] I. Fette (Google, Inc.) and A. Melnikov (Isode Ltd.). The WebSocket Protocol. RFC 6455, RFC Editor, December 2011.

[43] H. Iba. Emergent cooperation for multiple agents using genetic programming. In *Parallel Problem Solving from Nature PPSN IV. H.-M. Voight, W. Ebeling, I. Rechenberg, and H.-P. Schwefel*, pages 32–41, Berlin, September 1996.

[44] Google Inc. Angularjs. `http://angularjs.org`.

[45] T. Ishida. Real-time search for autonomous agents and multiagent systems. *Autonomous Agents and Multi-Agent Systems*, 1:1–139, 1998.

[46] Hong Jiang. From rational to emotional agents, 2007.

[47] D. N. Kinny and M. P. Georgeff. Commitment and effectiveness of situated agents. In *Proceedings of the Twelfth International Joint Conference on Artificial Intelligence (IJCAI-91)*, pages 82–88, Sydney, Australia, 1991.

[48] Lyndon C. Lee. Designing progressive multiagent negotiations, 1999.

[49] Michael Lees. A history of the tileworld agent testbed. Technical report, School of Computer Science and Information Technology, University of Nottingham. UK, 2002.

[50] Michael Lees and Brian Logan. Distributed simulation of agent-based systems with hla. *ACM Transactions on Modeling and Computer Simulation*, page 11, 2007.

[51] Shingo Mabu, Kotaro Hirasawa, and Jinglu Hu. Genetic network programming with reinforcement learning and its performance evaluation, 2004.

[52] S.; Nguyen, D.; Hanks and C. Thomas. The truckworld manual. Technical report, Dept. of Computer Science and Engineering, Univ. of Washington, 1993.

[53] David; Nunes Arthur; Ur Sigalit Pollack, Martha E.; Joslin. Experimental investigation of an agent design strategy. Technical report, Univ. of Pittsburgh Dept. of Computer Science, Pittsburgh, PA, 1993.

[54] David; Nunes Arthur; Ur Sigalit; Pollack, Martha E.; Joslin and Eithan Ephrati. Experimental investigation of an agent-commitment strategy. Technical report, Univ. of Pittsburgh Dept. of Computer Science, Pittsburgh, PA, 1994.

[55] Martha E. Pollack and John F. Horty. There's more to life than making plans: Plan management in dynamic, multi-agent environments. *AI Magazine*, 20:4–20, 1999.

[56] Martha E. Pollack and Marc Ringuette. Introducing the tileworld: Experimentally evaluating agent architectures. In *Proceedings of the Eighth National Conference on Artificial Intelligence*, pages 183–189, 1990.

[57] Juan A. Rodríguez-Aguilar, Francisco J. Martín, Pablo Noriega, Pere Garcia, and Carles Sierra. Competitive scenarios for heterogeneous trading agents. In *K. Sycara and M. Wooldridge (Eds.), Proceedings of the Second International Conference on Autonomous Agents*, pages 293–300, 1998.

[58] S. Russell and E. Wefald. Do the right thing: Studies in limited rationality. *Cambridge, Mass.: The MIT Press*, 1991.

[59] Stuart J. Russell and Peter Norvig. *Artificial Intelligence - A Modern Approach (3. internat. ed.)*. Pearson Education, 2010.

[60] Stuart J. Russell and Eric H. Wefald. Principles of metareasoning. *Proceedings, KR89. Morgan-Kaufmann*, 1989.

[61] Matthias J. Scheutz. The cost of communication: Efficient coordination in multi-agent territory exploration tasks, 2006.

[62] Martijn Schut and Michael Wooldridge. Intention reconsideration in complex environments. In *Proceedings of the Fourth International Conference on Autonomous Agents*, 2000.

[63] Martijn Schut and Michael Wooldridge. The control of reasoning in resource-bounded agents. *The Knowledge Engineering Review*, 16:1–200, 2001.

[64] Martijn Schut and Michael Wooldridge. Principles of intention reconsideration. In *In AGENTS01*, pages 340–347. ACM Press, 2001.

[65] Gerardo I. Simari and Simon D. Parsons. *Markov Decision Processes and the Belief-Desire-Intention Model: Bridging the Gap for Autonomous Agents*. Springer; 2011 edition, September 17, 2011.

[66] Gerald Jay Sussman. *A Computer Model of Skill Acquisition*. PhD thesis, Laboratorio de Inteligencia Artificial, MIT, 1973.

[67] TechSlides. Html5 game engines and frameworks. http://techslides.com/html5-game-engines-and-frameworks.

[68] A. M. Uhrmacher and B. Schattenberg. Agents in discrete event simulation. In *Proc. of the ESS'98*, pages 129–136, Ghent, SCS Publications, October 26, 1998, Nottingham.

[69] World Wide Web Consortium (W3C). What is css? http://www.w3.org/standards/webdesign/htmlcss#whatcss.

[70] D. Weld and J. deKleer. Readings in qualitative reasoning about physical systems. *Los Altos, Calif.: Morgan Kaufmann Publushers*, 1989.

[71] M. Wellman and J. Doyle. Preferential semantics for goals. In *Proceedings of the Ninth National Conference on Artificial Intelligence*, pages 698–703, Menlo Park, Calif: American Association for Artificial Intelligence, 1991.

[72] Michael Wooldridge. *An Introduction to MultiAgent Systems*. Wiley; 2nd edition, June 9, 2009.

[73] Chen Yan and Chen Yan. Study on stock trading and portfolio optimization using genetic network programming, 2010.

[74] Tom Dale Yehuda Katz and Ember.js contributors. Ember.js. http://emberjs.com.