Implementación de un simulador para la comparación de estrategias de resolución de problemas de un agente computacional basado en objetivos

María de las Nieves Cornier, Valentino Mantovani, Manuel Schnidrig

Universidad Tecnológica Nacional, Facultad Regional Santa Fe

Resumen

El objetivo de este trabajo es construir un agente inteligente basado en objetivos, para comprender como se relaciona con el mundo en el cual se desenvuelve (el ambiente) y como utiliza las diferentes técnicas existentes (búsqueda informada/no informada, cálculo situacional, plantificación, lógica, etc.) para tomar las decisiones pertinentes sobre las acciones que el mismo puede emprender. Se implementó un entorno de simulación, en donde se puede observar como un divertido agente computacional, llamado Ronly, utiliza diferentes estrategias de búsqueda y calculo situacional con algunas variantes para sortear un conjunto de laberintos predefinidos. La simulación permite comparar estas técnicas y sacar conclusiones sobre qué tipo de agente es mejor utilizar para resolver un problema determinado.

Palabras Clave

Inteligencia Artificial, agente, ambiente, búsqueda informada, búsqueda no informada, cálculo situacional, lógica, planificación, agente basado en objetivos, simulación.

Introducción

Este trabajo surge en el marco de la Cátedra Inteligencia Artificial de la UTN Facultad Regional Santa Fe con el fin de aplicar los conocimientos adquiridos en un caso práctico.

El problema que se planteó consiste en desarrollar un agente inteligente capaz de desenvolverse en el escenario planteado, el cual consiste en 3 laberintos interconectados entre sí de tal modo que las salidas de un laberinto se corresponden con las entradas del siguiente, tal como se muestra en la figura 1.

Cada laberinto contiene caminos de los cuales al menos uno de ellos permite ir desde la entrada a la salida.

Las salidas del laberinto pueden tener puertas, las cuales pueden estar cerradas con candado. Para abrir una puerta el agente debe estar ubicado en la misma casilla en donde se encuentra, y si la misma está

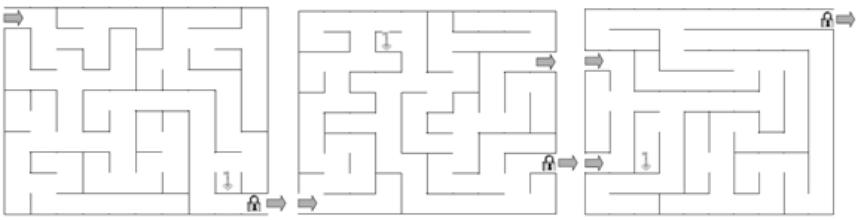

Figura 1 El escenario de laberintos planteado

cerrada debe poseer la llave correspondiente para abrirla.

Para añadirle una cuota de diversión a este trabajo, y para que nos acompañe a lo largo de la resolución del mismo, le hemos dado vida a nuestro agente resuelve-laberintos el cual se muestra en la figura 2.

Figura 2 Ronly, el agente resuelve-laberintos

Nuestro personaje se llama Ronly, tiene 22 años, vive en la capital de la provincia de Santa Fe (Argentina) y tiene una amplia experiencia en la resolución de puzles lógicos y laberintos.

Ronly puede desplazarse en las direcciones Norte, Sur, Este, Oeste. Para esto dispone de dos motores. Uno le permite girar sobre su eje 90° hacia la izquierda o hacia la derecha, mientras el otro le permite desplazarse hacia adelante en la dirección en la que se encuentra.

Para la resolución de los laberintos, Ronly utiliza dos estrategias de resolución de problemas: Búsqueda y Cálculo Situacional a fin de poder luego comparar los resultados obtenidos y seleccionar la mejor.

Cuando utiliza búsqueda el agente percibe el laberinto que tiene que resolver en cada nivel (mundo totalmente observable), conociendo la posición exacta de las entradas, salidas, de cada pared, de las llaves y si las puertas están abiertas o cerradas.

Cuando utiliza cálculo situacional, el agente recibe una percepción correspondiente a los elementos ubicados en la celda en la que está ubicado. Esto es, percibe las paredes a su alrededor y los objetos (si los hay) que se encuentran en dicha celda (mundo parcialmente observable).

Elementos del Trabajo y metodología

A fin de resolver las consignas planteadas, se utilizó el lenguaje JAVA dentro del entorno de desarrollo Eclipse. La cátedra de Inteligencia Artificial de nuestra facultad provee un framework para simular agentes inteligentes denominado FAIA [1], y una herramienta denominada IDEM-IA [2], la cual permite modelar con un lenguaje gráfico el problema a resolver, y generar el código automáticamente utilizando para tal fin las clases de FAIA. También se ha utilizado SWI-Prolog, a fin de modelar la base de conocimiento del agente al utilizar la estrategia de cálculo situacional.

El uso de estas herramientas FAIA e IDEM-IA hicieron posible el desarrollo del agente en tiempo y forma, agilizando principalmente el cambio de las estrategias usadas por el agente. Estas herramientas proponen un método de desarrollo de agentes llamado MDT-IA (Model Driven Teaching – Artificial Inteligence) el cual soporta la definición de modelos conceptuales a partir del cual se obtiene el código ejecutable.

Se utilizó una metodología ágil para el desarrollo del agente, utilizando un sistema de versionado (GIT) para darle un mejor seguimiento a los cambios y mejoras que se fueron añadiendo.

El libro que seguimos en cuanto a la teoría de agentes computacionales fue el de Russell, S. & Norvig, P. [5].

Implementación del Simulador

El simulador se genera instanciando alguna de las subclases *Simulator* de FAIA, dependiendo el tipo de simulación que se quiera realizar. Si bien el Framework nos fue de gran utilidad para no tener que escribir un simulador completo desde cero, se fueron presentando ciertas cuestiones por las que

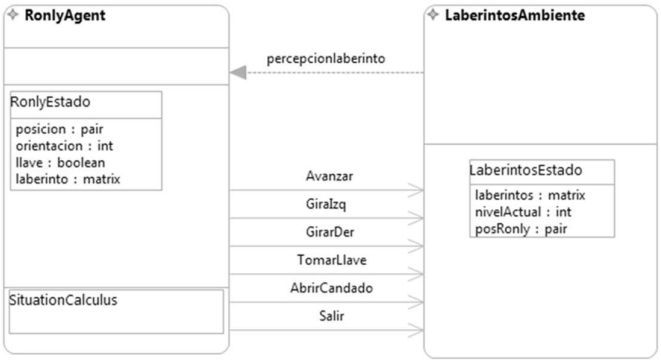

Figura 3 Diagrama IDEM-IA que resuelve el problema

tuvimos que ir adaptándolo. A continuación describimos los procesos que fuimos llevando a cabo para lograr llegar a nuestra solución a partir de la base de FAIA.

Definición del problema en IDEM-IA

A través de la herramienta IDEM-IA se construyó el diagrama que vemos en la figura 3, el cual define la arquitectura del simulador.

En el mismo se pueden observar las interacciones entre el Agente *RonlyAgent* y el ambiente *LaberintosAmbiente*. Por empezar, el Agente recibe una percepción de su entorno llamada *perepcionLaberinto*. Dependiendo de la estrategia utilizada, la misma adopta diferentes formas.

En el caso de búsqueda informada, la percepción que recibe es un laberinto completo, con la información de paredes, entradas, salidas y llaves.

En el caso de cálculo situacional, el agente percibe la información respecto de las celdas vecinas a la celda en la que se encuentra actualmente. En este caso, cuando se genera el código FAIA a partir de esta definición conceptual, se genera la clase *percepcionLaberinto* como subclase de *Perception* y es tarea del programador realizar las definiciones necesarias en dicha clase. También se definieron las acciones posibles para el agente: *Avanzar, GirarIzq, GirarDer, TomarLlave, AbrirCandado* y *Salir.*

Una vez que se recibió la percepción, Ronly resuelve internamente el problema con la información obtenida a través de la estrategia que se haya decidido utilizar, y así, concluye en realizar una o varias acciones que lo posicionarán en un estado diferente.

Finalmente el Agente ejecuta el conjunto de acciones que planificó y el laberinto se actualiza con esta información, para comenzar un nuevo ciclo de percepción-acción. La figura 4 nos muestra esta secuencia de ejecución del simulador.

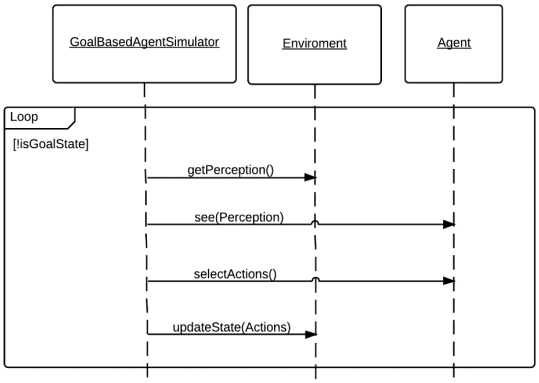

Figura 4. Secuencia de ejecución del simulador

Modelo de diseño del agente

Antes de mostrar la implementación de las estructuras de datos que permiten dar solución a nuestro problema, realizamos un modelo de diseño con un nivel de detalle mayor a la estructura del agente, identificando su estado, posibles acciones, y la prueba de meta que le permitirá saber si ha alcanzado su objetivo. El diagrama de la figura 5 muestra con más

detalle la definición conceptual que se hizo del agente.
Dicho modelo es común a ambas estrategias, con la diferencia de que cuando se utiliza la estrategia de búsqueda se lleva el registro completo del nivel actual (el "mapa" del laberinto), mientras que en cálculo situacional el agente va descubriendo el laberinto a medida que lo recorre.

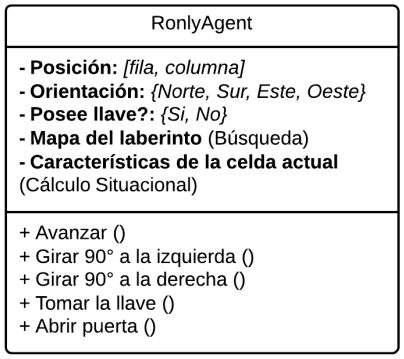

Figura 5. Modelado conceptual del agente

También definimos las precondiciones y poscondiciones propias de cada una de las acciones posibles:

Avanzar
- Precondiciones: No debe haber una pared ni una puerta cerrada en frente de Ronly que le impida el paso.
- Poscondiciones: Ronly avanza una casilla en la dirección de su orientación actual, y actualiza su nueva posición en su estado.

Girar 90° a la izquierda
- Precondiciones: Ninguna.
- Poscondiciones: Ronly gira 90° a la izquierda y actualiza su orientación.

Girar 90° a la derecha
- Precondiciones: Ninguna.
- Poscondiciones: Ronly gira 90° a la derecha y actualiza su orientación.

Tomar la llave
- Precondiciones: Debe haber una llave en la casilla actual
- Poscondiciones: Ronly toma la llave y lo registra en su variable *posee llave?*

Abrir una puerta
- Precondiciones: Debe haber una puerta cerrada en frente de Ronly (en la casilla contigua en dirección de su orientación), y éste debe poseer la llave.
- Poscondiciones: Ronly abre la puerta.

Implementación del Estado del Agente

En el caso de Búsqueda, el estado está definido por una clase en el modelo de objetos la cual no difiere más que en cuestiones de implementación de la representación conceptual.

En el caso de Cálculo Situacional, el estado está definido por un conjunto de predicados Prolog, los cuales se mantienen en una Base de Conocimiento (BC) dentro del objeto agente. Los predicados que definen el estado son:

situacionActual(S): hace referencia a la situación actual de cada uno de los valores que toma el estado del agente.

nivelActual(N): es el nivel en el que el agente se encuentra actualmente.

executedAction(A,S): *Indica que se ejecutó la acción A en la situación S.*

en(X,Y,S): *indica que Ronly se encuentra en la posición X,Y en la situación S.*

orientación(O,S): *Indica que Ronly se encuentra mirando hacia la dirección O en el momento S.*

sostiene(N): *Utilizada para modelar que el agente posee la llave en el nivel N.*

candadoAbierto(N): *Indica que el candado ya ha sido abierto en el nivel N.*

Toma de decisión para Búsqueda

Ronly utiliza dos estrategias de búsqueda distintas: *A** y *Costo Uniforme*.

Para estas estrategias de define la siguiente función de costo:

g(x) = cantidad de giros x 5

La heurística que diseñamos para resolver el problema de que las salidas pueden ser más de una y algunas pueden requerir encontrar una llave de antemano, es elegir la menor de las distancias entre el agente y cada una de las salidas. Si la salida está cerrada con candado y Ronly no posee la llave para abrirla, entonces, en ese caso, la distancia que se calcula es la de Ronly hasta la llave más la distancia de la llave hasta la salida con candado como muestra la expresión (1).

Toma de decisión para Cálculo Situacional

Para el caso de Calculo Situacional, utilizamos distintas estrategias de valoración de acciones para decidir cuál será la próxima acción a tomar.

La primera estrategia se denomina *"Celda menos visitada"*, la cual consiste en elegir siempre por sobre las demás aquellas acciones que nos conduzcan a alguna celda que haya sido visitada la menor cantidad de veces de las que es posible alcanzar desde la posición actual.

La otra estrategia se denomina *"Regla de la Mano Derecha"*. La estrategia consiste en pegar una mano contra una de las paredes del laberinto (la derecha en este caso) e ir avanzando sin despegar la mano de dicha pared.

Como una mejora a ambas estrategias, ideamos un algoritmo que le permite a Ronly volver rápidamente al candado cuando lo descubre antes de obtener la llave. Esta mejora (que puede activarse o no en la GUI) la denominamos *"Algoritmo de Hansel-Gretel"* y consiste en que Ronly deje una miga de pan por cada celda visitada luego de ver un candado (y no poseer la llave para abrirlo). Al momento de encontrar dicha llave, se añade la regla de que siempre que en alguna celda contigua haya depositada una miga, el agente deberá ir en esa dirección (caso contrario aplican las reglas normales de la estrategia actual), lo cual llevará a Ronly rápidamente a la puerta con candado.

Representación del Estado del Ambiente

El ambiente almacena tanto el conjunto de laberintos contiguos (el mundo) como la ubicación de Ronly en el instante actual. En función de simplificar el desarrollo, la posición de Ronly se almacena de manera relativa al nivel en el que se encuentra, por lo que podemos interpretar que la posición global de Ronly viene dada por el par *(nivelActual, posRonly)*.

Cada vez que el agente decide ejecutar una acción o secuencia de acciones, una lista de acciones es pasada al ambiente por el simulador, y éste entonces actualiza su estado.

Percepciones

En cada ciclo el simulador solicita una percepción al ambiente. Lo que se obtiene es la información que puede recabar Ronly del lugar donde se encuentra a través de sus sensores.

Cuando utilizamos búsqueda, lo que se obtiene es un mapa completo del laberinto actual. Cuando utilizamos cálculo situacional, lo que se obtiene es un predicado ***percepción***.

Dicho predicado percepción contiene el conjunto de parámetros que recibe el agente del ambiente. Está conformado por los siguientes parámetros que dan información sobre la celda actual:

- *[arriba, abajo, izquierda, derecha]*
- *[llave, candado, salida]*

```
s tal que s Min({r / r = distancia(ronly,salida) si (!ronly.tieneLlave
Y salida.tieneCandado) O r = dist(ronly, llave) + distancia(llave,sa-
lida) si (ronly.tieneLlave O !salida.tieneCandado) }
```

Exp. (1)

- *Nivel*

Cuestiones y Desafíos que se fueron planteando

Luego del desarrollo inicial del diagrama en IDEM-IA, la generación de las clases extendidas de FAIA y la construcción de ciertas estructuras auxiliares, que nos fueron útiles para representar cuestiones que no nos proveen los tipos predefinidos de java, se nos presentaron ciertas cuestiones y desafíos que tuvimos que ir resolviendo para poder ajustar el simulador y de esta manera resuelva de manera correcta y elegante nuestro problema. A continuación, detallamos dichas cuestiones.

Modificación del solver de FAIA

El primer desafío con el que nos encontramos al comenzar la implementación de nuestra solución fue el hecho de que el framework FAIA está más bien diseñado para un entorno cambiante, por lo que toma como verdad que al resolver un árbol de búsqueda, solo la primera acción del camino encontrado será realmente llevada a cabo en el ambiente, y es por ello que el método que resuelve el árbol de búsqueda nos devuelve solo esta primera acción. Por ello, tuvimos que modificar dicho método para que en el momento de tomar la siguiente decisión, nos devuelva el camino completo de acciones, dado que nuestro entorno no cambiará y por ende este camino será válido para resolver el laberinto en el que nos encontramos.

Representación de los laberintos

Una vez realizados estos cambios, tuvimos que definir una adecuada estructura para representar los laberintos. Nuestra elección fue hacerlo a través de una matriz cuadrada de enteros (de tamaño de byte). Cada entero en dicha matriz representa una de las celdas en las que puede estar parado Ronly, y posee información acerca de las paredes existentes en cada una de las direcciones, además de información acerca de los objetos que contiene dicha celda (llave, candado) y si es un tipo especial de celda (entrada, salida).

Para guardar toda esta información en un simple entero, utilizamos un sistema de máscaras, donde cada uno de los primeros 8 bits del entero se corresponde a un elemento diferente de la celda, tal como se muestra en la siguiente figura.

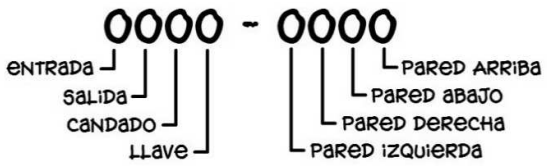

Figura 6 Interpretación de los bits de cada entero de la matriz del laberinto

Utilizando operaciones de bits podemos obtener separadamente la información requerida de una celda. Utilizamos la notación hexadecimal siempre que nos referimos a estos números dado que de esta manera obtenemos un número de dos dígitos, el primer digito representa las paredes, mientras que el segundo representa el resto de objetos.

La matriz que representa el laberinto es encapsulada en una clase Laberinto, la cual nos provee métodos que permiten manipularla de una manera más sencilla.

El agente en búsqueda

Para representar el agente, si bien utilizamos los mecanismos que nos provee *FAIA*, hicimos una ligera modificación: Dado que nuestro modelo plantea que en cada ciclo de percepción-acción nuestro agente resuelve un laberinto entero, pero no todos los laberintos de una sola vez, tuvimos que definir dos objetivos, los cuales denominamos *objetivo parcial* y *objetivo final* del agente.

Objetivo General o Final del agente: consiste en superar los tres niveles planteados. Este es el objetivo que le pasamos al simulador dado que nos define el corte de la simulación.

Objetivo Parcial del agente: consiste en llegar a alguna de las celdas de salida para el nivel actual. No consideramos el hecho de que Ronly atraviese la casilla de salida (se "salga" del laberinto) por motivos de simplificación, dado que si el agente llego a la casilla de salida es evidente que ya resolvió el problema, el "salto" se da por el cambio de nivel. Este objetivo parcial es el que le pasamos al ***solver*** para que nos encuentre un camino a alguna de las salidas.

El agente en cálculo situacional

Para el caso del cálculo situacional, la lógica está completamente embebida en la Base de Conocimientos, la cual fue implementada en SWI-Prolog gracias a una librería que nos permite utilizar dicho motor lógico dentro de Java [3]. La forma en que implementamos el cálculo situacional es la propuesta en [4].

Diseño de una Interfaz Gráfica

Con el objetivo de visualizar de una manera más intuitiva (y divertida) la resolución de los laberintos paso a paso, diseñamos una interfaz gráfica que nos permite ver a nuestro agente desplazándose sobre los laberintos a medida que los resuelve (figura 7).

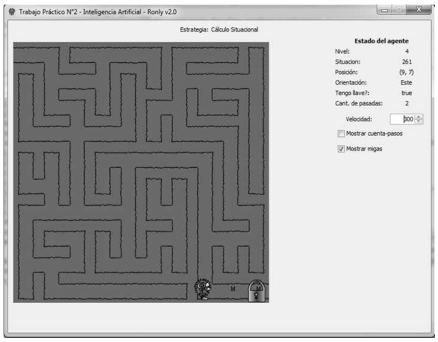

Figura 7: Interfaz gráfica

Esta interfaz además nos permite seleccionar la estrategia que queremos utilizar, y sus respectivas variantes

Resultados

Finalmente, podemos hacer una breve comparación de los resultados que obtuvimos aplicando las diferentes estrategias. Para ello hicimos diversas corridas sobre un mismo laberinto, mostrado en la figura 8.

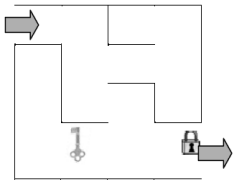

Figura 8: Laberinto de pruebas

Por empezar, debemos notar que a diferencia de todas las demás estrategias, A* es una estrategia informada, y por tanto es intuitivamente la que resultará en el camino más corto frente a las demás. De hecho, siempre que la heurística elegida haya sido correcta, el camino que nos arroje dicho algoritmo de decisión será óptimo en distancia.

Se hizo primero una comparación de las estrategias de búsqueda A* (usando la heurística planteada más arriba en este documento), y dos versiones de una estrategia de Costo Uniforme: una en donde decidir girar sobre la misma posición tiene mayor costo que avanzar a una posición adyacente, y otra a la inversa. Los resultados obtenidos pueden compararse gráficamente en la figura 9. Dichos árboles muestran los nodos que han sido expandidos en la búsqueda del camino a la salida.

En cuanto al Cálculo de Situaciones, la solución no es la óptima hacia la salida. Esto se debe a que a diferencia de una estrategia informada como A*, Ronly ya no percibe el laberinto completo para poder decidir qué acciones conviene tomar para resolverlo. Otra diferencia es que en el caso de cálculo situacional, Ronly no se detiene al principio del laberinto a buscar una solución completa

Figura 9: Interfaz gráfica del laberinto y el agente desplazándose sobre él.

a su problema si no que va resolviéndolo a medida que avanza.

Discusión

El uso de las herramientas FAIA e IDEM-IA facilitó la implementación de la solución propuesta y nos permitió modificar las estrategias usadas por el agente para resolver el problema gracias al diseño modular que brindan. De esta forma fue posible comparar las mismas agregando un componente de análisis necesario para comprender y experimentar en forma práctica las teorías aprendida durante el cursado.

Conclusión

Este trabajo nos ha permitido experimentar las etapas básicas por las que debe pasar un equipo que se involucre en la construcción de un agente inteligente, desde el modelado conceptual hasta la implementación del mismo. Hemos comparado distintas técnicas de resolución de problemas basados en objetivos, conociendo cómo funciona cada una de éstas y pudiendo así sacar conclusiones tanto de performance como de conveniencia.

Para finalizar podemos decir que este trabajo fue un fuerte motivador y propulsor del aprendizaje de la inteligencia artificial, una herramienta didáctica a través de la cual es posible llevar a la práctica los contenidos teóricos de la cátedra.

Agradecimientos

Agradecemos a los docentes de la cátedra Inteligencia artificial por la colaboración en la redacción de este informe y al proyecto PID 25/O 128 de UTN – FRSF por financiar este proyecto.

Referencias

[1] Jorge Roa, Milagros Gutiérrez y Georgina Stegmayer. FAIA: Framework para la enseñanza de agentes en IA. IE Comunicaciones: Revista Iberoamericana de informática educativa. 2008.

[2] Walter Santana, Jorge Roa, Milagros Gutiérrez, Georgina Stegmayer. IDEM-IA: Un entorno de desarrollo integrado para el modelado de agentes inteligentes. IE Comunicaciones: Revista Iberoamericana de informática educativa. 2012.

[3] Jorge Roa, Milton Pividori, Milagros Gutiérrez, Georgina Stegmayer, Jorge Vega,: La tecnología educativa al servicio de la educación tecnológica. FAIA: Un Framework para el desarrollo de agentes inteligentes. Ed. EduTecNe. Pp. 459-485, 2010

[4] Inteligencia Artificial: evitar pensar dos veces. http://nacho.larrateguy.com.ar/2007/06/19/inteligencia-artificial-evitar-pensar-dos-veces/

[5] Russell, S. & Norvig, P. Artificial Intelligence: A Modern Approach. Prentice-Hall, 2nd edition. 2003

Datos de Contacto:

Manuel Schnidrig. UTN FRSF. Colastine Norte – Santa Fe. el.manu@gmail.com

Valentino Mantovani. UTN FRSF. San Jeronimo 3232. Santa Fe. vale_manto@gmail.com

Maria de las Nieves Cornier. UTN FRSF. 25 de Mayo 3089. Santa Fe. pelum@gmail.com