FAIA: Framework para la enseñanza de agentes en IA

Jorge Roa, Milagros Gutiérrez y Georgina Stegmayer

CIDISI-CONICET, Universidad Tecnológica Nacional-Fac. Reg. Sta. Fe
Lavaise 610, 3000 Santa Fe
Argentina
mmgutier@frsf.utn.edu.ar, gstegmayer@santafe-conicet.gov.ar

Resumen: Este trabajo presenta un framework que permite resolver problemas académicos de la asignatura Inteligencia Artificial (IA) aplicando agentes basados en objetivos que usan búsqueda para la toma de decisión. Con el uso del framework, denominado *FAIA: Framework para un Agente que resuelve problemas de IA*, el alumno puede centrar su atención en la definición de las propiedades del agente y su estrategia para decidir qué acción emprender en cada interacción con el ambiente, sin tener que preocuparse por el simulador del ambiente en el cual se desenvuelve el agente. Para su desarrollo se usaron técnicas de diseño orientado a objeto y patrones de diseño. El objetivo de este trabajo es proveer una herramienta flexible con la cual el alumno pueda aprender a construir agentes inteligentes, comprendiendo la relación de éste con su ambiente. De esta forma pueden proponerse problemas a resolver más complejos, requiriendo menos tiempo en la implementación de una solución.

Palabras clave: enseñanza, Inteligencia Artificial, framework, agente inteligente.

Abstract: This paper presents a framework that allows solving academic problems posed in the Artificial Intelligence (AI) subject using objective based agents that use searching for decision making. Using the framework, called *FAIA: Framework for an Agent that solves AI problems*, the student can focus on defining the properties of the agent and its strategy to decide what action to take in each interaction with the environment, without worrying about the environment simulator. For its development object-oriented design techniques and design patterns were used. The purpose of this work is to provide a flexible tool with which students can learn to build intelligent agents, comprehending its relationship with the environment. This way, more complex problems can be proposed to be solved, requiring less time to implement a solution.

Keywords: teaching, Artificial Intelligence, framework, intelligent agent.

1. Introducción

En la cátedra Inteligencia Artificial (IA) dictada en la carrera Ingeniería en Sistemas de información de la Universidad Tecnológica Nacional-Facultad Regional Santa Fe (UTN-FRSF, Argentina) uno de los objetivos que se persigue es que el alumno sea capaz de aplicar técnicas de IA en la resolución de problemas. Así, desde la asignatura se trata de proveer oportunidades (en forma de situaciones problemáticas a resolver) para que el alumno vea cuáles son los problemas típicos de la IA y cómo éstos pueden ser resueltos desde un punto de vista ingenieril. Una de las actividades que se propone es el desarrollo de un agente de software que debe desenvolverse (interactuar) en un cierto ambiente (recibiendo percepciones y ejecutando acciones sobre el mismo) para tratar de lograr su objetivo de diseño.

El problema que surge aquí es que, para problemas complejos, los alumnos no logran su resolución en el tiempo que se les asigna, dado que no sólo tienen que tomar decisiones acerca del diseño del agente (tema desconocido para ellos hasta entonces) sino que

 © ADIE, Asociación para el Desarrollo de la Informática Educativa

también se requiere la simulación del ambiente donde el agente va a ser probado. Ante estos inconvenientes, se deben plantear problemas más simples que puedan ser tratados en tiempo y en forma.

Distintos autores proponen variadas definiciones de qué es un agente [Maes 95][Wooldridge 02]. Según la definición de agente propuesta en [Russell et al. 02] adoptada en la cátedra, un agente es todo aquello que puede considerarse que percibe su ambiente mediante sensores y que responde o actúa en tal ambiente por medio de efectores o actuadores. Un agente percibe su entorno a través de sensores que le permiten conocer el universo o ambiente en donde actúa y decidir más acertadamente qué acción emprender sobre el mismo. Los sensores son implementados en hardware cuando el agente trabaja en un entorno físico (p.e. cámara de video, sensores de temperatura) o se simulan por software. La salida de los sensores es una percepción (p.e. imagen, temperatura), es decir, datos que le interesan al agente para llevar a cabo su misión. El diseñador del agente tiene que definir cuáles son las acciones que el agente es capaz de realizar. Ésta será ejecutada por los actuadores, modificando el ambiente. El ambiente en el cual se desenvuelve el agente es un elemento importante en el diseño del mismo, dado que representa la situación problemática que el agente debe resolver.

El objetivo de la IA es diseñar el programa del agente. Éste es una función que implementa la relación que establece el agente entre percepciones y acciones. Se asumirá que este programa será ejecutado sobre algún tipo de dispositivo computacional. Las arquitecturas posibles para el programa del agente son realmente arquitecturas de software para sistemas de toma de decisión que están embebidas en un entorno. No todos los agentes son iguales y el diseñador debe seleccionar la arquitectura más conveniente de acuerdo a sus objetivos de diseño. Las distintas arquitecturas vistas en la cátedra son: (i) agente reactivo simple, (ii) agente reactivo basado en modelos, (iii) agente basado en objetivos (iv) agente basado en utilidad y por último (v) agente que aprende [Russell et al. 02].

Una de las actividades prácticas que se le solicita a los alumnos es construir un agente basado en objetivos para resolver problemas, para comprender como éste se relaciona con el mundo en donde se desenvuelve y para comprender cómo, utilizando las técnicas para solución de problemas presentadas en la materia, va tomando decisiones sobre qué acciones ejecutar para lograr su objetivo de diseño: resolver un problema. Comparado con otros tipos de agentes, se puede decir que la implementación de este tipo de agente es de complejidad media. En este trabajo se propone un framework para la solución de problemas académicos, orientado a que el alumno se familiarice con la temática de agentes y vea también plasmada en forma práctica y didáctica la interacción entre arquitecturas de software y sistemas de agente. Si bien existe un proyecto que propone estrategias y estructuras de datos codificadas en Java [4], éstas no están orientadas a ser una guía que ayude al alumno en el diseño del agente sino a facilitar la implementación, ofreciendo código reutilizable. Así las clases que se encuentran en este sitio, no están integradas como en FAIA.

Por otro lado, analizando los frameworks disponibles, de uso libre, se detectó que los mismos no están orientados a la enseñanza de arquitecturas de agentes, sino que fueron específicamente desarrollados para modelar las relaciones entre sistemas multi-agente como JADE [Bellifemine et al. 06] o entre agente móviles, como Aglets [Aridor 98], lo cual agrega complejidad al momento de querer definir simplemente un solo agente. Otro ejemplo es Zeus [Collis et al. 00], una biblioteca de componentes de software para desarrollo de agentes reactivos, que usan reglas y envío de mensajes. Sin embargo, ninguno de estos trabajos está orientado a la enseñanza académica, no poseen la flexibilidad suficiente que permita modelar agentes que se mueven en distintos tipos de ambientes, ni incorporan explícitamente los conceptos de IA que son necesarios para el diseño de un agente basado en objetivos que resuelve problemas con búsqueda. Es por eso que se propone el framework FAIA explicado en detalle más adelante.

A continuación el trabajo se organiza de la siguiente manera: se hace una introducción sobre agentes basados en objetivos que resuelven problemas, luego se explica la arquitectura del framework, sus clases y los patrones que fueron utilizados mediante una instanciación del framework con un ejemplo. Luego se muestra el plug-in

desarrollado para FAIA y una encuesta hecha a los alumnos de la cátedra de IA. Finalmente se presentan las conclusiones del trabajo.

2. Agente basado en Objetivos

Un agente basado en objetivos necesita información sobre el objetivo que persigue, la cual describe las situaciones deseables. Este tipo de agente, para decidir qué hacer, tiene en cuenta el estado del ambiente así como también la descripción de su objetivo. De esta manera, para seleccionar la acción necesaria en cada momento, combina la información sobre el objetivo y el conocimiento del resultado de las posibles acciones. En la toma de decisión, tiene en cuenta dos aspectos fundamentales: i) ¿Qué ocurre si se ejecuta tal o cual acción? (resultado de las acciones); ii) ¿Qué acción lo complace más? (utilidad).

En algunas situaciones, el resultado de una acción alcanza el objetivo deseado en forma directa. Pero en otros casos, el agente tiene que realizar algún proceso más complejo de búsqueda para determinar la secuencia de acciones que lo lleven a alcanzar su objetivo. Búsqueda y planificación son técnicas de IA encargadas de encontrar esta secuencia de acciones que le permitan al agente alcanzar sus objetivos. Como muestra la figura 1, un agente basado en objetivos tiene una memoria o estado interno que representa el conocimiento que tiene del entorno y de su experiencia en él (secuencia de percepciones). Cuando percibe, actualiza esta memoria con nuevo conocimiento, y elige la mejor acción basándose en la misma. Luego, vuelve a actualizar su estado interno de acuerdo al conocimiento que posee de cómo la acción modificará al ambiente. En este tipo de agentes existe una función siguiente que establece el siguiente estado interno a partir del estado interno actual y la percepción actual. Finalmente retorna la acción elegida para que los actuadores hagan su trabajo.

Desde la cátedra de IA, en forma grupal, se solicita que los alumnos definan un agente basado en objetivos para resolver un problema, es decir, un agente que no tiene conocimiento del universo o ambiente en el cual interactúa pero que sí mantiene en su estado interno la secuencia de percepciones que recibe de éste para poder aprender de ellas y tomar mejores decisiones. En particular se les solicita que esta toma de decisiones se realice con búsqueda [Russell et al. 02]. Este trabajo práctico se divide en etapas, siendo la primera la definición de la arquitectura del agente de software, que será implementado en JavaTM, lo cual permite aprovechar las ventajas de portabilidad, software libre, escalabilidad y reuso que brinda este lenguaje.

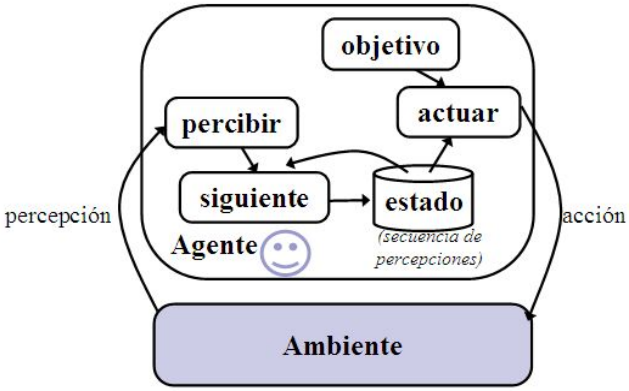


Figura 1. Esquema de un Agente basado en objetivo.

Desde el punto de vista de la Ingeniería de software, el objetivo de este trabajo es ofrecer un diseño uniforme de la solución antes de pasar a su implementación. Desde el punto de vista de la IA, el objetivo de este trabajo es disminuir la brecha (que a menudo a los alumnos les es difícil superar) entre resolver un problema de búsqueda en una hoja y codificar esa solución en un lenguaje de programación de tal manera que sea algo casi transparente.

3. Framework FAIA

Un Framework orientado a objeto es una arquitectura de software reusable, que provee tanto código como diseño. Éste representa una aplicación parcial de software, tanto en diseño como en código, para solucionar problemas de un dominio dado. Una de las definiciones de framework más utilizadas es la dada por Johnson y Foote: "A framework is a set of classes that embodies an abstract design for solutions to a family of related problems" [Johnson et al. 88].
Por la definición dada, un framework se compone de un conjunto de clases y de flujos de control y de esta

manera, provee una estructura precisa para definir nuevas aplicaciones que resuelven problemas dentro de un dominio dado. Existen dos tipos de framework: de caja negra (o black-box) y de caja blanca (o white-box) [Fayad 99]. En los primeros, los mecanismos de control están ocultos al usuario y la forma de utilizarlos es a través de la composición de objetos. Los segundos, son usados extendiendo las clases que aparecen en el framework. Estas clases son conocidas como los puntos de flexibilización o hot-spots.

A diferencia del uso tradicional de las bibliotecas de clases, un framework tiene implementados los hilos de control y las llamadas a código de las aplicaciones las cuales son realizadas cuando se necesitan. Esto es, el framework provee un contexto para los componentes de una biblioteca de clases que será reusada. El desarrollo de un framework no es una tarea trivial, si se tiene en cuenta que el objetivo del mismo es proponer una solución parcial a un conjunto de problemas de un dominio dado. Esto quiere decir que la solución que el framework proporciona debe tener componentes de software que puedan ser adaptados y reusados en la implementación total de una solución para un problema específico. Las técnicas de desarrollo de software orientado a objeto han demostrado ser una herramienta poderosa en la obtención de componentes de software reusables, tales como los patrones de diseño [Gamma et al. 95] los cuales son un excelente ejemplo de la aplicación del diseño orientado a objetos para obtener componentes de software flexibles y reusable. Estos fueron plasmados por los expertos, representando las mejores prácticas de diseño en la obtención de componentes reusables. Estos patrones deben ser tenidos en cuenta a la hora de diseñar e implementar un framework.

El uso de un framework en el desarrollo de aplicaciones de software no solo reduce el tiempo necesario para obtener resultados sino también facilita el mantenimiento de la aplicación. Al reusar los componentes, se reusa también el diseño, logrando obtener aplicaciones de alta calidad como un experto, aún siendo principiante [Pree 96].
De esta manera, se realiza un esfuerzo grande una sola vez al definir el framework, el cual puede ser reutilizado muchas veces, en forma simple y manteniendo la calidad de los resultados.

El framework propuesto (FAIA) plantea una solución al problema de implementar agentes inteligentes basados en objetivos que resuelven problemas. Esto es, dado un problema específico, es posible crear un agente instanciando clases y completando métodos de las mismas de acuerdo a características específicas del problema en cuestión, mientras que existen elementos comunes al dominio de agentes que resuelven problemas que ya están implementados en el framework y el usuario sólo necesita reusarlos, como pueden ser por ejemplo, las estrategias de búsquedas que el agente utilice, las estructuras de datos para representar su estado, la construcción del árbol de búsqueda, el programa del agente (percibir, decidir y actuar) entre otras.

De esta manera, FAIA combina tanto conceptos de ingeniería de software – como son los patrones de diseño – como también conceptos de IA – como son las distintas estrategias de búsqueda. Con este trabajo, se pretende proponer a los alumnos un ejemplo concreto de aplicación de la ingeniería de software, definida según la IEEE como “la aplicación de un enfoque sistemático, disciplinado y cuantificable al desarrollo, operación y mantenimiento de software”[1]. Esta aplicación en concreto se ve realizada en el ámbito de la IA, para el desarrollo de un agente basado en objetivos que resuelve problemas.

Las clases dentro de FAIA están agrupadas en paquetes: *Agent*, *Enviroment*, *Simulator*, *Solver* y *State*. En esta sección se describe la utilización del framework FAIA a través de un caso de estudio propuesto por la cátedra y disponible en [2]. El mismo consiste en diseñar dentro de un video juego, un agente-Pac-Man que tiene como objetivo la supervivencia dentro de su mundo. Para poder sobrevivir, el agente necesita alimentarse y evitar ser eliminado por sus enemigos. Este objetivo debe ser explicitado de modo que en cada momento en que el agente debe tomar una decisión, pueda evaluar la información que tiene respecto de su estado y sepa si su objetivo ya ha sido cumplido o debe seguir actuando hasta alcanzarlo. Si el objetivo se plantea como "comer todo el alimento" y "matar a todos los enemigos" del ambiente, éstas premisas no podrían ser evaluadas debido a que el agente no sabe a-priori la ubicación ni la cantidad de comida y enemigos. Por lo tanto, una forma concreta y evaluable de

plantear el objetivo "sobrevivir" es hacer que el agente haya sensado todo su mundo y en ese mundo conocido no existan alimentos sin comer (ya que cada vez que detecte un alimento, irá a comerlo). Esto hace que el agente no deba necesariamente visitar todas las celdas ni matar a todos sus enemigos.

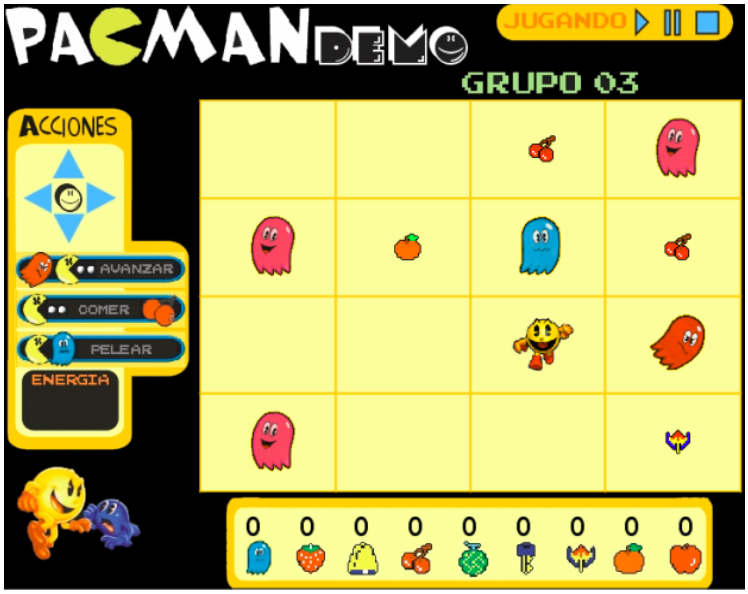


Figura 2. El agente-Pac-Man y su mundo.

Inicialmente tiene una cierta cantidad de energía que le permite moverse por su mundo, compuesto por celdas o habitaciones, en busca de alimento (figura 2). El tamaño del mundo es de 4x4. Es un mundo estático, los enemigos y el alimento están fijos, el que se mueve por el mundo es el agente. Para poder aumentar sus energías, Pac-Man necesita alimentarse, lo que implica llegar hasta las celdas donde se encuentra la comida. Si llega a una celda donde existe un enemigo, puede entrar en combate y el enemigo siempre morirá y Pac-man siempre perderá algo de energía. Si su energía queda en cero significará que ha muerto.

Pac-Man no tiene conocimiento de todo su entorno, sino que lleva consigo un equipo con sensores que le provee la siguiente información: un contador de energía que le indica su energía actual y sensores de celdas vecinas (arriba, abajo, izquierda y derecha) que emiten un 2 si hay alimento, un 1 si hay enemigo y 0 si la celda esta vacía. Para diseñar el agente es necesario definir los siguientes elementos:

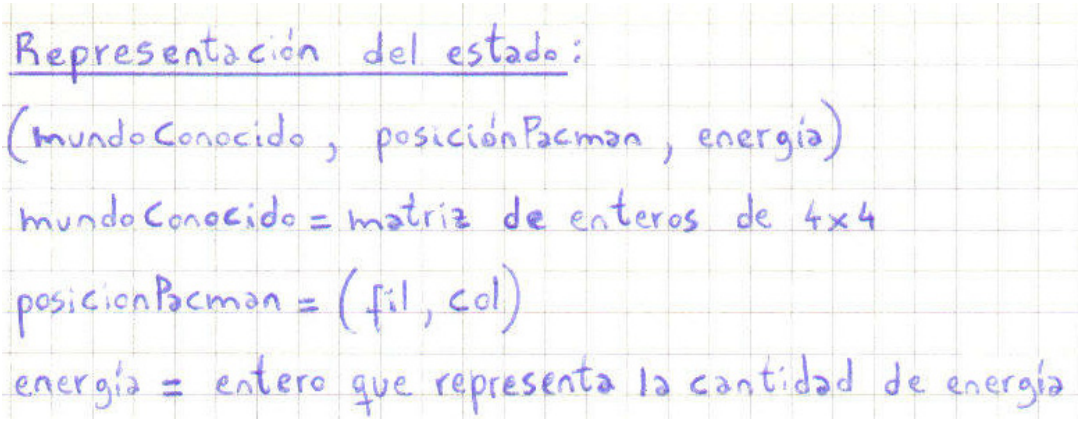
- representación interna (estado) del agente,
- percepciones que recibe del ambiente,
- acciones que podrá realizar en el ambiente,
- el problema a resolver en cada iteración (estado + acciones),
- prueba de meta (cómo evaluar cada estado para saber si ya llegó a la solución),
- estrategia de búsqueda para llegar a la solución.

Nótese que para el agente que se está definiendo, se presupone el uso de búsqueda como método para decidir la acción a emprender, es por eso que los tres últimos ítem están relacionados a la definición del proceso de búsqueda. Por otro lado se puede ver que existen en la definición del problema datos que no están claramente especificados, como por ejemplo el objetivo del agente. El mismo aparece con un lenguaje ambiguo "sobrevivir". Definir correctamente este objetivo es tarea del alumno, el cual deberá proponer una condición lógica que evalúe el estado en que el agente se encuentra y pueda determinar si el mismo corresponde o no a un estado objetivo. Del mismo modo existen varias posibilidades para el modelado del problema en general, por ejemplo la estrategia de búsqueda seleccionada, las estructuras de datos usadas para la representación del estado del agente, etc. Se mostrará una de las posibles interpretaciones de este objetivo en el punto 3.4 cuando se defina el problema.

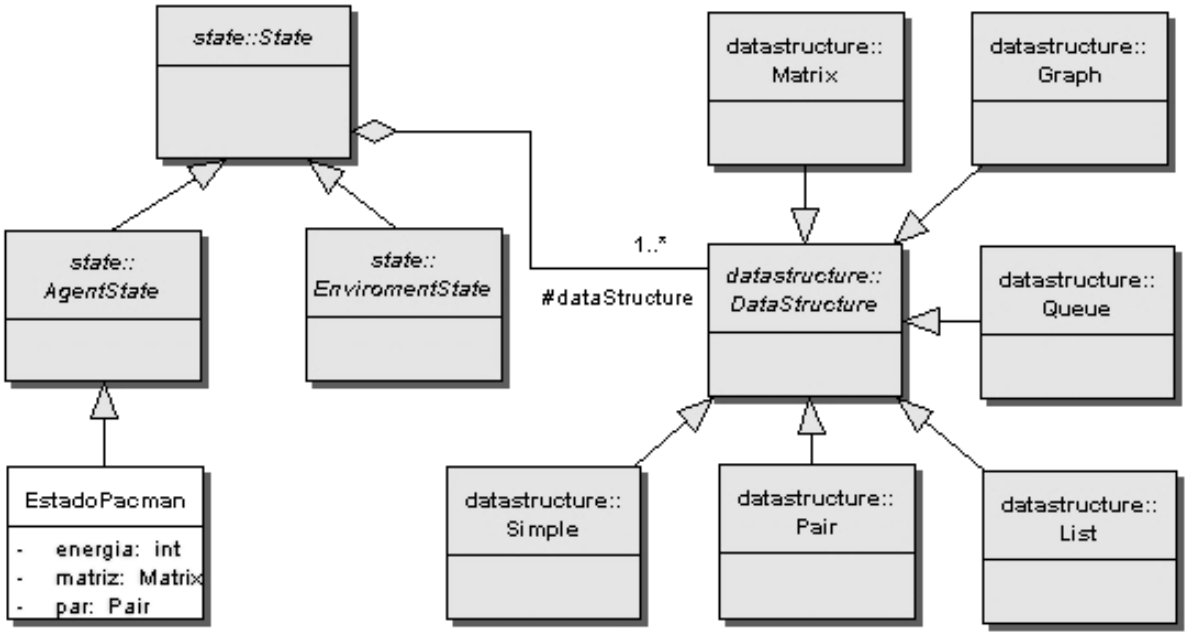
La metodología de enseñanza en la cátedra es presentar al alumno el problema para resolverlo en papel, esto permite entender el problema, identificar los datos relevantes (descartando aquellos que no lo son) e identificar los componentes del agente y de su mecanismo de decisión. Luego realizar el diseño lógico y detallado y finalmente implementarlo y correrlo utilizando diferentes situaciones problemáticas.

A continuación se describe paso a paso la resolución de este problema mostrando el bosquejo en papel que se realiza como primera instancia y cómo este se ve plasmado en la instanciación del framework. FAIA es utilizado por los alumnos como un plug-in en Eclipse [4] y presenta como puntos de

flexibilización las clases que representan los componentes a definir, tales como *AgentState*, *Perception*, *Action,* entre otras.



a)



b)

Figura 3. Representación del estado del agente, a) en papel; b) utilizando FAIA.

3.1. Definiendo la representación del estado

En la figura 3 a) se muestra la definición "en papel" que un alumno realiza para representar el estado de un agente, y en 3 b) el diagrama de clases con las clases provistas por el framework (en gris) que pueden ser usadas para representar el estado del agente.

De esta manera se facilita el diseño del mismo, ya que no es necesario preocuparse por el desarrollo de estructuras de datos complejas que no son inherentes al problema del agente desde el punto de vista de la IA, brindando una mayor simpleza al diseñador.

El alumno puede definir la clase *EstadoPacman* como subclase de *AgentState*, para representar el estado del agente Pac-Man. Los atributos que se definen en esta nueva clase son: `mundoConocido` es

una matriz que representa el tablero que le agente va conociendo. Los valores de las celdas se irán completando a medida que el agente conoce el mundo. Estos valores indicarán si la misma está vacía, si tiene un enemigo o si tiene comida; `posicionPacMan` representa la posición del agente dentro del tablero, con coordenadas (x,y); y `energía` que es un entero que guarda la energía que posee el agente. Esta estructura le permite al agente actualizar su estado interno de acuerdo a las percepciones que recibe del ambiente.

3.2. Definiendo las percepciones

Las percepciones le permiten al agente conocer el mundo donde se mueve cuando el mismo no es totalmente accesible [Russell et al. 02]. Al recibirlas del ambiente, actualiza su estado interno y, de esta manera, debe decidir cuál es la mejor acción a ejecutar que le permita alcanzar el estado objetivo. La figura 4 muestra la clase abstracta *Perception* definida en FAIA y la clase *PercepcionPacman* definida por el usuario. Dicha clase representa las percepciones para el caso particular del agente pacman.

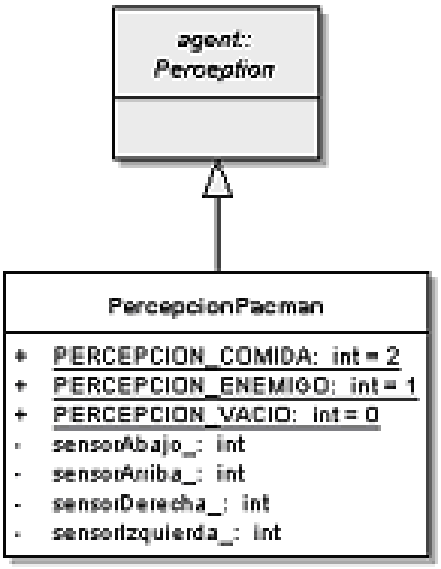


Figura 4. Percepciones del agente Pac-Man

La percepción para el Pacman está definida por 4 sensores: arriba, abajo, izquierda y derecha que permite observar el contenido de las celdas contiguas a la celda donde se encuentra el agente. Estos sensores pueden tener valores en el rango [0-2] donde: 0 representa que la celda está vacía, 1 representa que hay un enemigo y 2 representa la presencia de comida.

3.3. Definiendo los operadores

Siguiendo la descripción del problema del pacman, se puede afirmar que las acciones posibles del agente sobre el mundo son *irArriba, irIzquierda, irDerecha, irAbajo, comer* y *pelear*.

En la parte a) de la figura 5, vemos la definición en papel de alguna de las acciones, y en b), cómo definirlas extendiendo la clase abstracta *Action* del framework FAIA.

Cada una de estas clases representa una acción que puede realizar el agente-Pac-Man sobre el tablero. Estas clases deben implementar el método *execute(State)* de tal manera que se codifique el cambio de estado que produce la ejecución de las acciones.

3.4. Definiendo el problema

Un agente basado en objetivos necesita información de la meta que persigue. El programa del agente puede combinar información sobre la meta y el resultado de cada una de las acciones posibles, para decidir cómo ir seleccionando acciones que lo lleven a alcanzar su objetivo.

En particular, un objetivo y un conjunto de medios para alcanzarlo se denomina problema. Para generar una instancia de *Problem* es necesario crear clases concretas de las clases abstractas: *GoalTest, AgentState* y *Action* (figura 6).

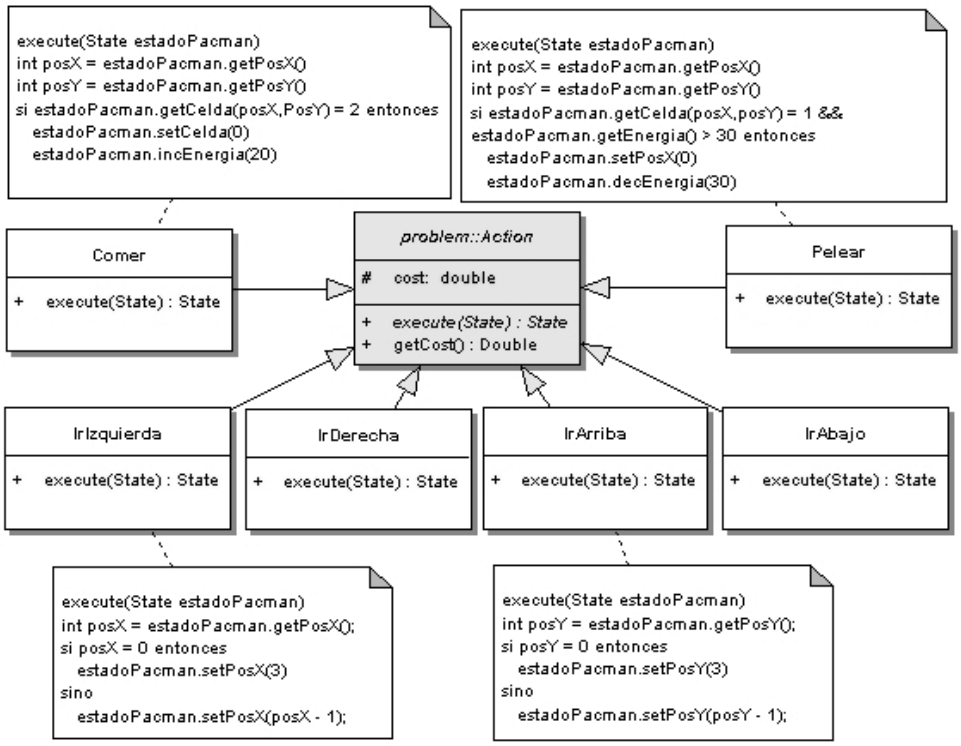
La prueba de meta es una función utilizada por el agente para determinar si un estado alcanzado es un estado objetivo. Esta función recibe un estado como argumento y retorna un valor booleano (true o false). Para definir la prueba de meta del pacman, se crea la clase *MetaPacman* (figura 7) que extiende de la clase *GoalTest*.

En esta implementación se debe definir el método *isGoalState()* el cual, basándose en un estado y una meta, debe determinar si el estado corresponde a un estado objetivo.

```
- irAbajo
  Si (fil==3)
      fil = ϕ
  Sino
      fil = fil + 1
  Si (mundoConocido [fil, col] == ϕ)
      mundoConocido [fil, col] = 3
      posicionPacman = [fil, col]

- Comer
  Si (mundoConocido [fil, col] == 2)
      mundoConocido [fil, col] = ϕ
      energía = energía + 2ϕ
- pelear
  Si (mundoConocido [fil, col] == 1 y energía > 3ϕ)
      mundoConocido [fil, col] = ϕ
      energía = energía - 3ϕ
```

a)



b)

Figura 5. Operadores del agente Pac-Man, a) en papel; b) utilizando FAIA

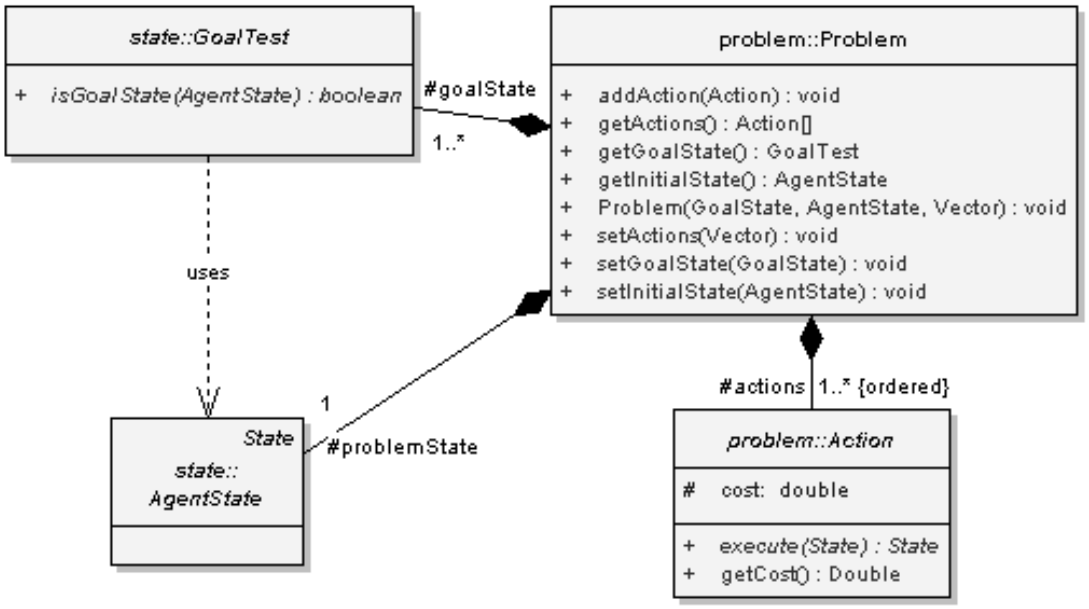
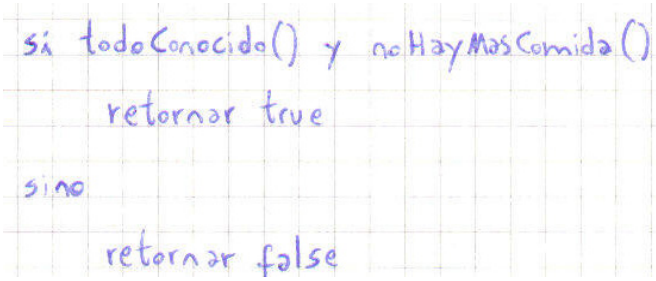
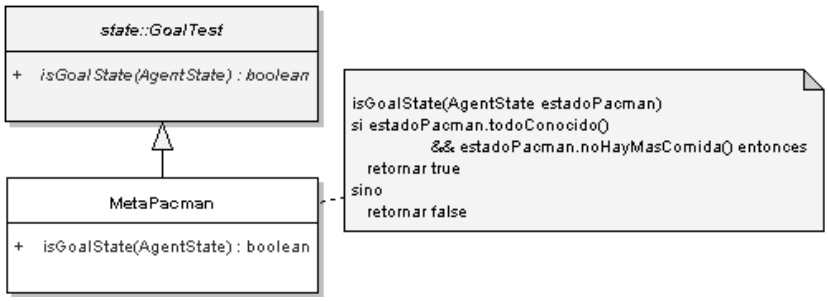


Figura 6. Definición del problema con FAIA.

Para el ejemplo propuesto, como se explicó anteriormente, la meta del agente Pac-Man es conocer todo el ambiente y comer todo el alimento, esto implica que el mismo debe sensar todas las celdas para determinar si el objetivo fue o no alcanzado.



a)



b)

Figura 7. Definiendo el objetivo del agente, a) en papel; b) utilizando FAIA.

3.5. Definiendo la estrategia

En el proceso de decidir qué acción emprender, el agente utiliza alguna estrategia de búsqueda. FAIA provee las implementaciones correspondientes a los algoritmos de las estrategias vistas en la cátedra: informadas (A* y avara) y no informadas (amplitud, profundidad, y costo uniforme) como se muestra en la figura 8.

En la representación se utiliza el patrón de diseño *Strategy* [Gamma et al. 95] que permite cambiar la estrategia en forma dinámica. Así, es posible comparar los resultados obtenidos al aplicar distintas estrategias en la búsqueda de la solución.

Para utilizar las estrategias provistas por el framework se debe generar una instancia de la clase *Search* indicando la estrategia que va a utilizar el agente. Si la misma es amplitud o profundidad se puede utilizar directamente. Pero si ésta es de tipo informada, se debe definir una clase que implemente la interfaz *IheuristicFunction* y para el caso de A* y costo uniforme es necesario definir también una clase que implemente la interfaz *IStepCostFunction*.

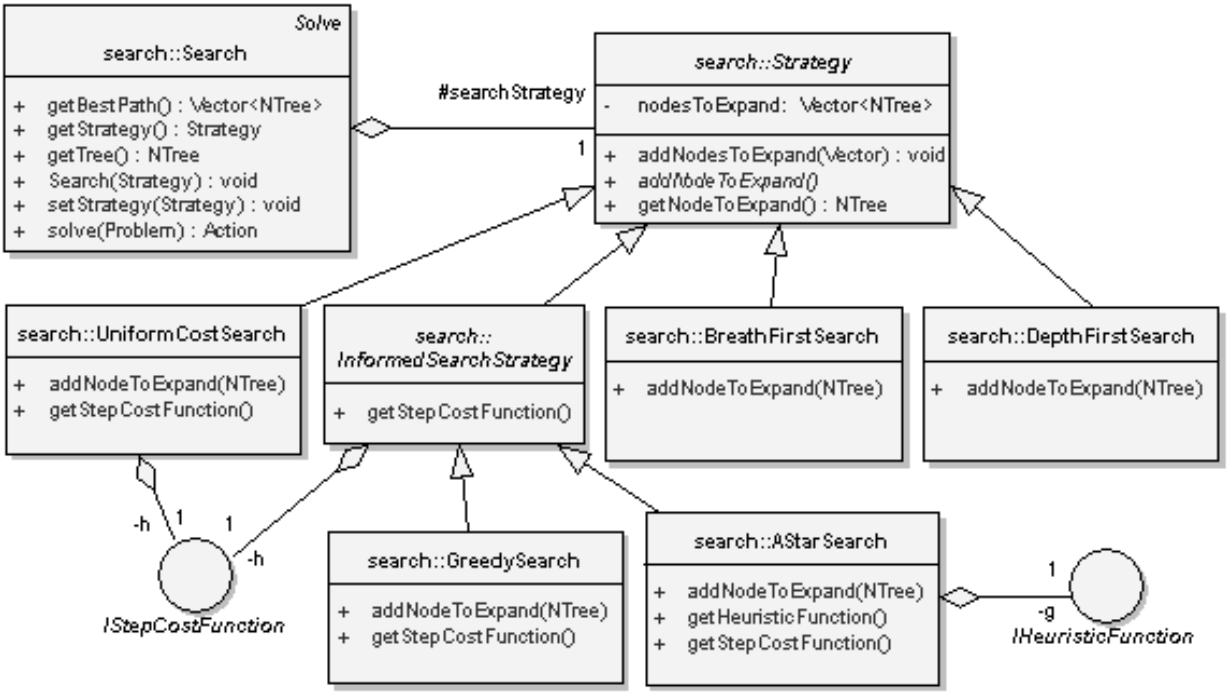


Figura 8: Búsqueda con el patrón Strategy.

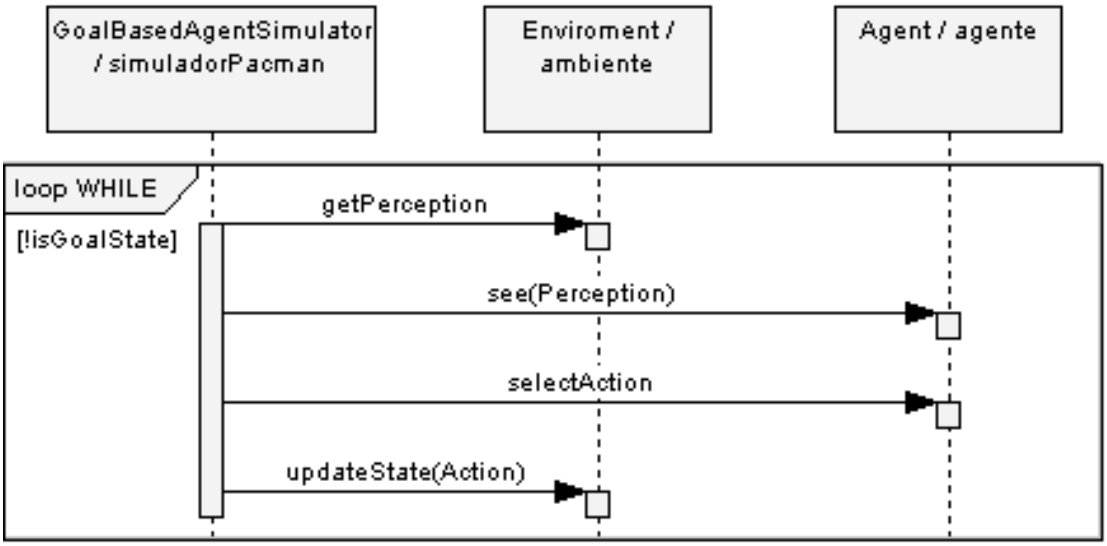


Figura 9. El proceso de simulación.

Mediante el uso de estas interfaces se obtienen la función heurística y de costo utilizadas en las estrategias de búsqueda informada. La estrategia seleccionada para el agente Pac-Man es la de costo uniforme donde *g=cantidad de celdas visitadas.*

3.6. Construyendo el agente

Una vez que se han definido todos los elementos que un agente necesita, finalmente se debe extender la clase *GoalBasedAgent* propuesta en el framework que define el comportamiento mínimo que debe tener un agente basado en objetivos.

3.7. Realizando la simulación

La simulación del agente basado en objetivos se realiza generando una instancia de la clase *GoalBasedAgentSimulator*. El comportamiento básico de una instancia de esta clase se ve reflejado en la figura 9. Así, el simulador obtiene del entorno las percepciones que le dará al agente a través del método *see()*. Luego, el simulador le pide al agente que seleccione una acción a realizar. El método *selectAction()* es el encargado de retornar la acción que el agente obtiene utilizando alguna estrategia de búsqueda, la cual es usada por el simulador para actualizar el entorno real donde el agente se desenvuelve. Con este diseño se logra que el simulador sea completamente independiente de la definición particular del agente.

3.8. Generando el árbol de búsqueda

FAIA permite generar los árboles de búsqueda que el agente basado en objetivos utiliza en el proceso de decisión de la acción a emprender. La clase *NTree* tiene definido un método que permite almacenar el árbol de búsqueda en un archivo XML [3]. De esta manera, se utiliza este estándar para generar la vista del árbol de búsqueda. El árbol en formato XML facilita la depuración de los alumnos en sus trabajos prácticos y la corrección de los mismos por parte de los profesores. La figura 10 muestra una sección del árbol de búsqueda en formato XML para la estrategia de costo uniforme generado para el ejemplo del agente-Pac-Man.

En el árbol se muestra el nodo raíz. El atributo `acción` determina la acción que dio origen al nodo dentro del árbol. En este caso es "null" debido a que se trata del nodo raíz. La `posicion` establece el lugar inicial del Pacman en la matriz en la celda de coordenadas (3,3) y `celdas_visitadas` (que representa el valor de la función de costo en ese nodo) es igual a 1 (la inicial). Los puntos suspensivos indican más nodos dentro del árbol que no son mostrados en la figura.

```
<NTree>
<Nodo
id="0"
orden_de_ejecucion="1"
accion="null"
posicion="(3,3)"
celdas_visitadas="1">
Nro: 1 pos: 1,1 acc: null celdas_visitadas: 1
<Nodo>
...
    <Nodo>
    ...
    </Nodo>
</Nodo>
</Nodo>
</NTree>
```

Figura 10. Árbol de búsqueda en formato XML.

4. Plug-in FAIA

Se ha desarrollado el framework como plug-in de la herramienta de desarrollo Eclipse [4] de IBM, generando un ambiente totalmente integrado que permite obtener el diseño y la implementación del agente que se desea desarrollar. Una captura de pantalla del plug-in se muestra en la figura 11. El objetivo principal del mismo es dar soporte al diseño conceptual de agentes basados en objetivos mediante una interfaz gráfica amigable. De esta manera el alumno se puede centrar específicamente en el diseño conceptual del agente, dejando inicialmente de lado su implementación.

Para esto el plug-in provee una barra de herramientas que brinda acceso a cada uno de los conceptos principales en el diseño de un agente basado en objetivos: el agente, el estado, el ambiente, los operadores, las percepciones y la estrategia, correspondientes a los puntos de flexibilización del framework. En la parte inferior de la figura 11 se muestran las propiedades de la estrategia del agente. En particular se muestra cómo mediante una lista desplegable el alumno puede definir la estrategia que va a utilizar el agente. Una vez definidos los conceptos elementales y seleccionada una estrategia, se genera automáticamente el código Java correspondiente que permite la simulación del ambiente, dejándole al alumno la tarea de codificar los operadores y el estado del agente.

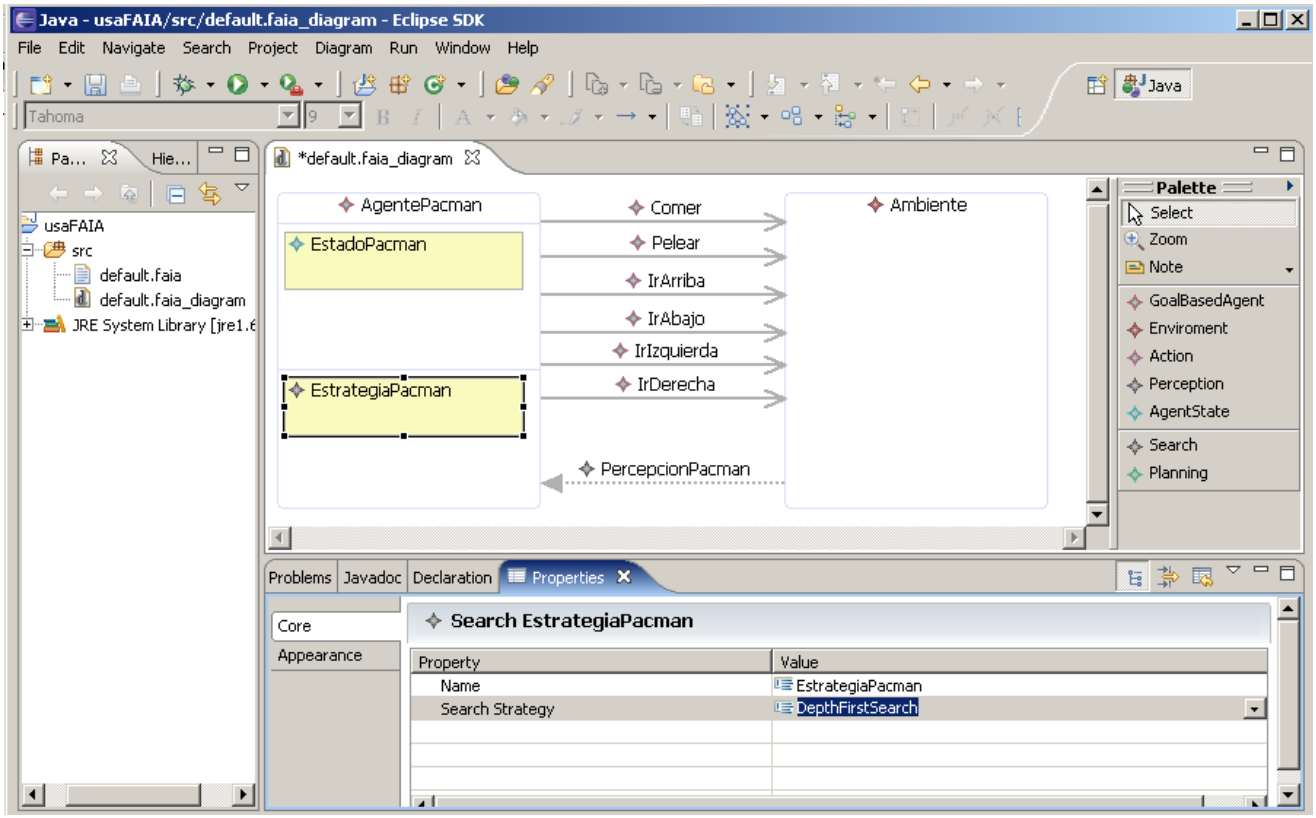


Figura 11. Captura de pantalla del Plug-in FAIA en Eclipse.

5. Encuesta sobre FAIA

Durante el ciclo lectivo 2007 FAIA fue desarrollado en el centro de investigación CIDISI por los profesores de la cátedra Inteligencia Artificial, sin embargo, los alumnos no tuvieron la posibilidad de usarlos dado que la misma se dicta en el primer cuatrimestre del año, cuando todavía se tenía muy poco desarrollado. Los alumnos resolvieron el trabajo práctico sin contar con ningún diseño previo. Antes de finalizar el ciclo lectivo 2007 se presentó el framework como una herramienta que da soporte a la construcción de agentes como los desarrollados en el práctico. Se explicó el modo de utilización del mismo y se les solicitó que respondieran un cuestionario como el que se muestra a continuación:

1) El desarrollo del TP, fue de utilidad para aprender los conceptos de búsqueda y para entender cómo un agente de software interactúa con un ambiente y usa búsqueda para resolver problemas?
2) Cómo resultó el desarrollo del trabajo?
3) Llevó más tiempo de lo esperado?
4) Cambió mucho el diseño original de la solución del problema planteado al finalmente implementado?
5) Si hubiera tenido la opción de usar FAIA para hacer el TP, lo hubiera usado?
6) Si respondió SI en 1, cree que FAIA le hubiera ayudado a hacer el TP más rápidamente?
7) Si respondió SI en 1, cree que FAIA le hubiera ayudado a hacer el TP más fácilmente?
8) Cree que FAIA le hubiera ayudado a entender mejor las técnicas de búsqueda?
9) El desarrollo del TP, fue de utilidad para aprender los conceptos de búsqueda y para entender cómo un agente de software interactúa con un ambiente y usa búsqueda para resolver problemas?
10) Cómo resultó el desarrollo del trabajo?
11) Llevó más tiempo de lo esperado?

12) Cambió mucho el diseño original de la solución del problema planteado al finalmente implementado?
13) Si hubiera tenido la opción de usar FAIA para hacer el TP, lo hubiera usado?
14) Si respondió SI en 1, cree que FAIA le hubiera ayudado a hacer el TP más rápidamente?
15) Si respondió SI en 1, cree que FAIA le hubiera ayudado a hacer el TP más fácilmente?
16) Cree que FAIA le hubiera ayudado a entender mejor las técnicas de búsqueda?

La encuesta fue respondida voluntariamente por veinte alumnos de IA, los cuales, en su gran mayoría (80% de los casos) coincidieron en que el TP les fue de utilidad para entender cómo funciona un agente y cómo se usa la búsqueda para la resolución de problemas. La mayoría consideró también al trabajo pedido como moderado y concuerdan en que el diseño del agente fue cambiando a medida que iban implementando. En cuanto a si hubieran usado FAIA de haber contado con la herramienta, más del 90% respondió afirmativamente.

6. Conclusiones

Se presentó un framework orientado a resolver problemas académicos en la enseñanza de Inteligencia Artificial en una carrera de Ingeniería. Se desarrolló un ejemplo mostrando cómo el framework fue utilizado para resolver un problema particular.

Las principales ventajas en el uso de esta propuesta son:

1. es una guía teórico-práctica que orienta al alumno a crear sin error los elementos que conforman un agente;
2. al definir el mecanismo de decisión del agente para elegir una acción, se reduce a definir cuestiones relacionadas con la resolución de un problema de búsqueda (estado, operadores, etc.) ya que la estructura básica del agente ya está diseñada en FAIA lo cual facilita la implementación;
3. el uso del patrón strategy permite cambiar la estrategia en forma dinámica;
4. es posible visualizar los distintos árboles de búsqueda en formato XML que son generados durante el proceso de alcanzar el objetivo del agente, lo cual permite al alumno entender dicho proceso.
5. finalmente, como asistencia al trabajo docente, en clase, es posible armar los árboles de búsqueda dinámicamente y mostrar cómo se van generando los mismos según las diferentes estrategias.

7. Trabajo futuro

En un trabajo futuro se pretende incorporar otras estrategias que le permitan al agente resolver problemas tales como planificación o cálculo situacional. Utilizar los archivos XML que almacenan los árboles de búsqueda para generar la vista de los mismos mediante una estructura de árbol jerárquica. Asociar las estructuras de datos con vistas predeterminadas, de manera tal que se facilite la implementación de la interfaces gráficas de las mismas al utilizarlas en la resolución de los problemas propuestos.

Referencias

[Aridor 98] Y. Aridor, D. Lange, “Agent Design Patterns: Elements of Agent Applications Design”, Proc. Second Int. Conf. Autonomous Agents, pp. 108-115 (1998)

[Bellifemine et al. 06] F. Bellifemine, G. Caire, D. Greenwood, “Developing Multi-Agent Systems with JADE”, Ed. Wiley & sons, New York (2006)

[Collis et al. 00] J. Collis, D. Ndumu, C. van Buskirk, “The Zeus. Agent Building Toolkit - ZEUS Methodology Documentation” (2000) http://labs.bt.com/projects/agents/zeus

[Maes 95] P. Maes, “Intelligent Software”, Scientific American, vol. 273, no. 3, pp. 84-86 (1995)

[Fayad 99] M.E. Fayad, R.E. Johnson, “Domain-Specific application Frameworks. Frameworks experience by Industry”, John Wiley and Sons, Inc. New York (1999)

[Gamma et al. 95] E .Gamma, R. Helm, R. Johnson, J. Vlissides, “Design Patterns: Elements of Reusable

Object-Oriented Software", Ed. Addison-Wesley Professional Computing Series (1995)

[Johnson et al. 88] R. Johnson, B. Foote, « Designing Reusable Classes". Journal of Object-Oriented Programming, vol. 1, no. 2, pp 22-35 (1988)

[Pree 96] W. Pree, "Framework Patterns". Sigs books and multimedia, New York (1996)

[Russell et al. 02] S.Russell, P.Norvig, "Artificial Intelligence: A Modern Approach!. Ed. Prentice-Hall (2002) http://aima.cs.berkeley.edu/

[Wooldridge 02] M. Wooldridge, "An Introduction to Multiagent Systems", John Wiley & Sons, New York (2002)

[1] http://standards.ieee.org/software/

[2]http://www.frsf.utn.edu.ar/matero/visitante/index.php?id_catedra=142

[3] http://www.w3.org/XML/

[4] http://eclipse.org