Evaluación de estrategias de razonamiento para sistemas basados en reglas

Marco Antonio Gómez Martín, Belén Díaz Agudo
Facultad de Informática
Universidad Complutense de Madrid
C/ Prof. José García Santesmases, s/n
28040 Madrid (España)
marcoa@fdi.ucm.es, belend@sip.ucm.es

Resumen

La enseñanza de sistemas basados en el conocimiento es aconsejable acompañarla de prácticas en la que los estudiantes implementen algún sistema basado en reglas (*rule based systems, RBS*). En este artículo proponemos el uso de juegos de tablero como un escenario de aplicación pequeño pero realista y motivante para el alumno. En particular, describimos los ejercicios que hemos pedido a nuestros estudiantes y el software que hemos desarrollado para facilitar su desarrollo.

1. Introducción

Pedagógicamente hablando, relacionar las prácticas y trabajos de los alumnos con juegos de ordenador es favorecedor, ya que en general son muy bien aceptados por parte de los alumnos [1]. Así lo refleja un número creciente de publicaciones relacionadas con experiencias educativas basadas en juegos [2] [3] [4].

Desde los orígenes de la Inteligencia Artificial (IA) los juegos de tablero, tipo damas, ajedrez o tres en raya, fueron elegidos para aplicar técnicas de resolución de problemas principalmente basadas en búsqueda heurística. Dejando a un lado el tema de la motivación, los juegos reúnen una serie de características que los hacen idóneos para aplicar otras técnicas de IA. Por ejemplo, un tamaño del problema razonable, reglas claramente establecidas que no requieren conocimiento de sentido común y métodos de evaluación perfectamente definidos.

La aproximación descrita en este artículo se encuadra dentro de la asignatura *Ingeniería de Sistemas Basados en el Conocimiento* (ISBC), impartida en la Ingeniería en Informática de la Facultad de Informática de la Universidad Complutense de Madrid. Describiremos una de las prácticas que los autores del artículo, como profesores de la asignatura, crearon para el curso 2005/2006. En esta práctica se propone la programación de un sistema basado en reglas para la resolución del buscaminas, un solitario famoso universalmente por formar parte de la instalación estándar de Windows.

En el siguiente apartado presentamos el contexto académico de la asignatura. El apartado 3 describe los objetivos de la práctica. En los apartados 4 y 5 describimos las reglas del juego y el diseño del esqueleto de aplicación sobre los que los alumnos desarrollaron su solución. El apartado 6 evalúa la experiencia. Antes de las conclusiones el apartado 7 relaciona este trabajo con aproximaciones similares.

2. Marco académico

La asignatura optativa de 9 créditos Ingeniería de Sistemas Basados en el Conocimiento[1] se lleva impartiendo desde hace cuatro años en la Facultad de Informática de la UCM. Su principal objetivo es servir como una especie de laboratorio de la asignatura troncal *Inteligencia Artificial e Ingeniería del Conocimiento* (IAIC). En las clases teóricas se repasan algunas de las técnicas vistas en IAIC (se espera que los alumnos ya la hayan cursado), y durante las sesiones de laboratorio se realizan prácticas que se orientan hacia la resolución de problemas reales. El número medio de alumnos durante estos años está alrededor de 90, agrupados de tres en tres para realizar los trabajos prácticos.

La asignatura introduce los sistemas basados en conocimiento y distintos paradigmas de razonamiento, principalmente los sistemas basados en reglas y los sistemas basados en casos. Se estudia

[1] http://www.fdi.ucm.es/profesor/belend/ISBC/isbc.html

también el uso de ontologías, técnicas avanzadas de procesamiento de lenguaje natural y las arquitecturas de agentes. En este artículo nos centramos en la práctica que propusimos para la parte de razonamiento basado en reglas durante el curso 2005/06. La práctica tiene tres partes pero en este artículo nos centraremos principalmente en la propuesta para resolver tableros del buscaminas.

En cursos anteriores, para los sistemas de reglas se proponía una primera parte en la que se proporcionaba el conjunto de reglas completo para un sistema experto muy sencillo para el diagnóstico en un dentista. El cometido era introducirles en la sintaxis y funcionamiento de CLIPS (y JESS), y una sesión de laboratorio era suficiente para terminarla. En la segunda parte se les pedía la programación de un sistema experto en un dominio de su elección. Los alumnos debían elegir el área que más les interesara para realizar la extracción del conocimiento y la programación de las reglas necesarias.

En el curso académico 2005/2006 hemos mantenido la primera parte como método para introducirlos en la sintaxis y funcionamiento del lenguaje de reglas y hemos sustituido la segunda por dos prácticas de enunciado fijo. Las razones se detallan a continuación.

3. Objetivos

Permitir a los alumnos la creación de un sistema experto de tema libre tiene dos problemas. El primero de ellos es que se dificulta enormemente su corrección, ya que cada práctica es completamente diferente de la anterior, y el profesor no puede aprovechar la homogeneidad para compararlas y evaluarlas. Además es muy difícil detectar copias entre prácticas de años distintos. El segundo problema es que en algunos casos los alumnos se sienten incapaces de elegir un tema adecuado sobre el que hacer el sistema experto. Sin embargo destacamos como ventaja el hecho de que, una vez elegido el tema, el resultado suele ser una buena práctica, debido al alto grado de motivación por haber escogido ellos mismos la temática.

Para eliminar los problemas anteriores y mantener una alta motivación en los alumnos, hemos propuesto prácticas de enunciado fijo en el dominio de los juegos que son motivantes por sí mismos.

El primero de los juegos consistió en la programación mediante reglas del control del jugador (agente) en el mundo artificial del *Wumpus* descrito en el capítulo 7 del libro "*Artificial Intelligence: A Modern Approach*" [5].

El mundo del Wumpus es una cuadrícula en la que cada casilla puede contener una bolsa de oro, un agujero o el Wumpus (el enemigo al que hay que evitar). El agente, basándose en su percepción del entorno y en ciertas reglas de razonamiento, debe encontrar (deducir) un camino seguro hacia el oro, evitando los agujeros y el Wumpus. Una vez recogido, debe volver a la salida. Las reglas del juego pueden consultarse en [5].

Para realizar la práctica, permitíamos a los alumnos utilizar cualquiera de los dos motores de inferencia más utilizados: CLIPS [6] y Jess [7]. Tras la realización de esta práctica los alumnos asimilaban la sintaxis del sistema de reglas, así como el encaminamiento hacia delante y el manejo de tableros. Además, la práctica del mundo de Wumpus pone de manifiesto la necesidad de adquisición del conocimiento y de evaluación de la capacidad de resolución de problemas de un sistema basado en reglas.

Para probar la práctica, utilizaban una configuración de tablero (tamaño, posición del Wumpus, oro y agujeros) definida a partir de hechos asertados en la base de reglas. Les proporcionamos 300 tableros para probar sus estrategias e incluir el resultado en la memoria final de la práctica.

En el enunciado de la práctica[2] les explicábamos que la estrategia debería tener una fase de percepción que depende de la posición que ocupa el agente en un cierto momento. Sin embargo, nada les impedía hacer trampas y utilizar todos los hechos asertados para "ver" el contenido de las casillas más allá del alcance del agente. Este problema es uno de los que tratamos de resolver en la siguiente parte de la práctica que consistía en la implementación con Jess de una estrategia inteligente para resolver tableros del buscaminas de diferentes niveles de dificultad

Los objetivos principales que guiaron el diseño de la práctica fueron los siguientes:

[2] Disponible a través de http://www.fdi.ucm.es/profesor/belend/ISBC/isbc.html

- Mantener un alto nivel de motivación: para eso, seguimos confiando en el entorno de los juegos, por las razones antes referidas.
- Facilidad de corrección y evaluación: igual que en la mayoría de los juegos, es muy fácil saber si una estrategia está funcionando o no: si resuelve el tablero antes de un cierto tiempo, gana; si no, pierde. Además, las herramientas que desarrollamos y que describiremos más adelante nos facilitaron en gran medida la evaluación.
- Imposibilidad de hacer trampas: para implementar las estrategias tanto en el mundo de Wumpus como en el buscaminas, el razonador debe tener acceso al estado del mundo (tablero). En el caso del Wumpus, el mundo completo estaba al alcance de cualquier regla, por lo que una estrategia podía hacer trampas. La única forma de asegurarse de que la práctica no infringía las leyes de la percepción definidas era revisar con sumo cuidado y una a una todas las reglas del sistema, algo tedioso que quisimos evitar en el buscaminas. Posteriormente veremos qué acciones tomamos para conseguirlo.
- Evitar trabajo lateral: en la práctica del Wumpus, muchos de los grupos terminaron programando rutinas para el dibujado más o menos sofisticado del tablero, para facilitar su labor en las tareas de depuración. Para evitar este trabajo adicional, y permitirles centrarse por completo en la extracción del conocimiento y el razonamiento, definimos la práctica de tal forma que pudimos proporcionarles código Java que se encargaba de dibujar el tablero mientras se resolvía.

La siguiente sección describe brevemente las reglas del buscaminas y la Sección 5 detalla el conjunto de programas Java que implementan el esqueleto de la práctica.

4. ¿Cómo se juega al buscaminas?

En el solitario del buscaminas el jugador comienza con un tablero en forma de cuadrícula de casillas, todas ellas cubiertas. Algunas ocultan debajo una mina, y el objetivo es levantar todas las que no lo hagan. Para cada casilla puede elegir si quiere levantarla para ver lo que esconde o marcarla como peligrosa porque piensa que bajo esa casilla hay una mina. El jugador puede añadir el icono  a una casilla para indicar que cree que en esa casilla hay una mina.

Para la toma de decisiones, cuando se levanta una casilla, aparece el número de minas que hay en las casillas circundantes. Así si una casilla tiene el número 3 significa que de las ocho casillas que hay alrededor (si no está en una esquina o borde) hay 3 con minas y 5 sin minas. Si se descubre una casilla que no tiene minas alrededor, el interfaz suele mostrarla sombreada sin número y descubre automáticamente las adyacentes. Si el jugador levanta una casilla con una mina () se pierde la partida.

5. Diseño e implementación

Para facilitar el trabajo de los alumnos en la resolución de la práctica y de los profesores para su corrección, hemos desarrollado un conjunto de clases de Java que hemos proporcionado a los alumnos para que las usen a través de dos aplicaciones que detallaremos en los apartados siguientes.

Para conseguir los objetivos descritos en la Sección 3, las aplicaciones no permiten a los alumnos hacer trampas, ocultando el estado completo del tablero al sistema de reglas, y se encargan de presentarlo por pantalla para facilitar la depuración de la práctica. La aplicación Java se encarga de crear y almacenar toda la información del tablero y de hacer públicas una serie de funciones al sistema de reglas. Esas funciones son las únicas disponibles desde Jess para acceder al tablero, y coinciden con las que puede realizar un jugador humano sobre un interfaz de usuario típico:

- Descubrir (levantar) una casilla: si contiene una mina, el jugador (que en nuestro caso es el razonador Jess) habrá perdido, y el programador debe asegurarse de que no se dispare ninguna regla más. La función devuelve el número de minas adyacentes a la casilla o un valor especial indicando que se ha levantado una mina (y por tanto ha perdido).
- Marcar/desmarcar una casilla: cuando el jugador deduce que una casilla tiene una mina escondida, puede marcarla para que aparezca con un dibujo identificativo (). Esta función sería útil para un razonador humano pero es

poco probable que el razonador JESS marque visualmente las casillas que ha identificado como peligrosas; aún así, añadimos la posibilidad de hacerlo.

- Examinar el contenido de una casilla: mientras se juega, el usuario tiene accesible el contenido (número de minas adyacentes) de todas aquellas casillas que han sido levantadas. La función recibe unas coordenadas, y devuelve el número de minas adyacentes. Obviamente, la implementación tiene en cuenta qué casillas se han levantado, y devuelve un código de error en caso de ser llamada con una casilla que no descubierta. Normalmente el sistema de reglas va construyendo una representación interna del tablero que conoce, por lo que no es probable que utilice esta función. No obstante, igual que con las anteriores, la hemos hecho accesible para proporcionar el conjunto completo de funciones.
- Número total de minas: devuelve el número de minas escondidas en el tablero inicial. Dependiendo del nivel de dificultad del tablero, el número de minas cambiará.

Existe una última función que también se hace pública desde Java: un generador de números aleatorios. Resulta evidente que el sistema de reglas necesita números aleatorios al menos para poder empezar la partida, ya que en ese momento, con el tablero completamente vacío, es imposible hacer ninguna deducción sobre la colocación de las minas. En realidad Jess tiene una función nativa para generar números aleatorios que los estudiantes podrían utilizar directamente. Sin embargo, les animamos a utilizar la nuestra, por las razones que veremos en la Sección 5.4.

El diseño general del interfaz Java-Jess está hecho con dos objetivos en mente. El primero, como ya se ha dicho, es evitar las trampas, evitando que el razonador tenga acceso a partes a las que las reglas del juego se lo impiden. El segundo es proporcionar un método de acceso al tablero que sea completamente transparente y no fuerce ni dé pistas sobre el tipo de representación interna que deben utilizar los razonadores. Cada estrategia particular puede elegir el método de representación que más facilite la programación de sus reglas; el interfaz con el tablero ni estorba ni da pistas para encontrar la mejor forma para hacerlo.

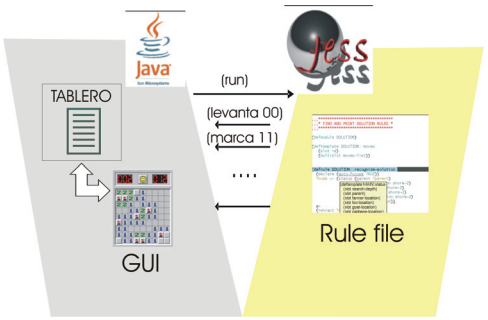

Figura 1. Arquitectura

Una vez establecido el interfaz de acceso implementamos tres aplicaciones que se describen en los subapartados siguientes. La primera para generar tableros de ejemplo y las otras para probar las estrategias de resolución de uno o varios tableros, respectivamente.

Las dos últimas aplicaciones, por tanto, deben cargar las reglas del razonador. Para eso, crean un contexto de ejecución de Jess, en el que se cargan las reglas del sistema experto. Acto seguido, publicitan las funciones que dan acceso al tablero creado en Java, así como el hecho que indica su tamaño. Cada vez que el tablero es modificado, el interfaz de usuario que presenta el tablero en la pantalla es actualizado, gracias al patrón de diseño Observer [8]. Cuando la aplicación Java está preparada para la resolución del tablero, ejecuta la orden `(run)` de Jess para empezar la ejecución de las reglas (ver Figura 1).

5.1. Generador de tableros

Los tableros del buscaminas se pueden generar de forma aleatoria. Sin embargo, mientras se depura el razonador, es mucho mejor poder utilizar la misma configuración del tablero una y otra vez, para poder comprobar si los fallos supuestamente corregidos han sido efectivamente arreglados.

Debido a esto, creamos un sencillo formato de fichero de texto para almacenar una configuración de un tablero. Como veremos, la aplicación de prueba de estrategias admitía como parámetro un nombre de archivo para cargarlo.

Por otro lado, cuando una estrategia es mejorada añadiendo un nuevo conjunto de reglas, es bueno probarla con un gran número de tableros para ver si resuelve más de lo que lo hacía una versión anterior del razonador.

Nivel	Tamaño	Número de minas
Fácil	9x9	10
Medio	16x16	40
Difícil	30x16	99

Tabla 1. Niveles de dificultad

La aplicación de generación de tableros empaqueta en dos ficheros ZIP 300 y 3000 tableros generados aleatoriamente. La aplicación crea tableros de los tres niveles de dificultad cuyas características aparecen en la Tabla 1. Estos tableros se entregaron a los alumnos para que los resolvieran usando sus estrategias con la aplicación JessStrateyTester detallada en la Sección 5.3.

5.2. MinesweeperJess

Esta aplicación tenía como objetivo su uso durante el desarrollo y depuración de una estrategia. La aplicación se ejecuta a través de la línea de comandos y recibe, al menos, el nombre del fichero que contiene el conjunto de reglas Jess capaces de resolver el tablero.

Si no se indica lo contrario, prueba la estrategia con un tablero generado aleatoriamente; el nivel de dificultad del tablero puede cambiarse mediante parámetros.

Sin embargo, la aplicación también es capaz de utilizar un tablero fijo almacenado en un fichero. Utilizando los parámetros se puede indicar el nombre el fichero, que puede incluso formar parte de un ZIP comprimido (en ese caso, se utiliza el nombre del fichero ZIP, seguido del nombre del fichero dentro él, separándolos por el símbolo #). Gracias a esto, los alumnos podían probar el razonador con el mismo tablero, para resolver errores.

La aplicación crea el contexto de ejecución y lanza el conjunto de reglas. Para poder ver el proceso de resolución del juego, el programa escribe en la consola el estado del tablero cada vez que éste varía (cuando las reglas llaman a las funciones de acceso al tablero). Sin embargo, esto no es suficiente si se quiere que la aplicación ayude en el proceso de depuración. El desarrollo y depuración de SBR requiere capacidades para la ejecución de las reglas una a una, ver qué hechos hay asertados, qué reglas hay en la agenda planificadas, etc. Para permitirlo, añadimos un parámetro adicional, `--debug`, que cuando se activa en vez de poner a ejecutar el sistema de reglas (ejecutar la orden `(run)` de Jess), una vez creado el contexto abre una consola de Jess, y se queda esperando la intervención del usuario. Mediante la consola, los estudiantes podían ir ejecutando las reglas en bloques de N reglas (con `(run n)`), consultar qué reglas estaban activas (con `(agenda)`), listar los hechos (`(facts)`), o llamar directamente a las funciones de acceso al tablero.

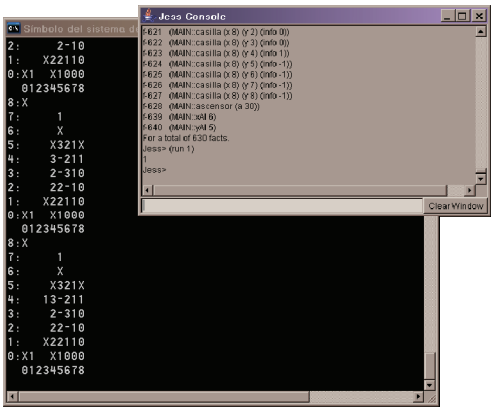

Figura 2. MinesweeperJess en ejecución

La Figura 2 muestra una captura de la aplicación funcionando. En ella aparece en segundo plano la ventana con la aplicación de consola en donde se escribe el estado actual del tablero, y una ventana en primer plano con la consola de Jess.

5.3. JessStrategyTester

La aplicación MinesweeperJess es útil para comprobar cómo funciona una estrategia determinada sobre un único tablero y está diseñada para ser utilizada durante la fase de construcción del razonador.

Una vez completada, o al menos una vez implementada una parte de la estrategia, es útil ver cómo ha mejorado el razonador con los últimos cambios. Para eso lo mejor es poner a prueba al sistema de reglas con un conjunto grande de tableros.

La aplicación JessStrategyTester hace precisamente eso. Recibe como parámetro un fichero ZIP con tableros comprimidos, y el fichero que contiene las reglas. Crea el contexto de ejecución del razonador, y lo prueba con todos los tableros contenidos en el ZIP. Una vez terminados todos,

genera un informe indicando el número de tableros resueltos y no resueltos de cada nivel.

La aplicación la podían utilizar para comparar las estadísticas conseguidas por distintas versiones de su razonador utilizando los dos conjuntos de tableros que les dimos. Además, les pedimos que las estadísticas de la última versión las incluyeran en la memoria final de la práctica.

La aplicación tiene un parámetro extra para indicar el tiempo máximo que se le permite a un razonador intentar resolver el tablero. Es una forma de protegerse contra estrategias defectuosas que entran en bucles infinitos. Por defecto, ese tiempo estaba fijado en 500 milisegundos, pasados los cuales, el programa terminaba con la resolución de ese tablero y pasaba al siguiente.

El informe final generado también lleva la cuenta del número de tableros sin terminar por falta de tiempo. En la Tabla 2 se ve la información generada en la ejecución de una de las prácticas sobre el conjunto de 3000 tableros.

	Fácil	Medio	Difícil	Total
Nº de tableros	1000	1000	1000	3000
Resueltos	562	310	13	890
No resueltos	437	683	963	2078
Fuera de tiempo	1	7	24	32

Tabla 2. Estadísticas de una de las prácticas

5.4. Aleatoriedad del buscaminas

Como queda dicho, el principal objetivo de la aplicación JessStrategyTester es permitir a los alumnos la comparación de distintas versiones de su razonador. Sin embargo, también es útil para que los profesores puedan comparar la efectividad y eficiencia de las estrategias de los distintos grupos.

El uso de un conjunto prefijado de tableros hace más justa esa comparación al eliminar ese factor de aleatoriedad. Sin embargo, las estrategias necesitan en algún momento depender de la suerte. Al principio de la ejecución, o en determinados momentos de la resolución, el tablero está en un estado que hace imposible cualquier deducción segura sobre la colocación de minas. En esos momentos, la estrategia utilizará números aleatorios para elegir la casilla a levantar.

La situación es especialmente importante en el principio del juego, ya que nuestro tablero no garantiza (como suelen hacer los GUIs del buscaminas) que la primera casilla que se levante no contenga una bomba. Este componente aleatorio adicional añade ruido en las comparaciones de dos estrategias, por lo que ideamos un método para evitarlo.

La solución adoptada fue proporcionar un generador de números aleatorios propio, controlado desde la parte de Java. Instamos a los alumnos a utilizar nuestra función en vez de la nativa de Jess. La aplicación MineSweeperJess de la Sección 5.2 utiliza el generador de números tradicional, mientras que JessStrategyTester (descrita en la Sección 5.3) genera siempre la misma secuencia de números. De esta forma, se garantiza que todo razonador siempre empezará descubriendo la misma casilla, eliminando ese factor de ruido en la comparación de distintas estrategias.

6. Evaluación

Hemos observado que la motivación con las prácticas de este año ha sido muy alta. Las razones han sido principalmente el uso de la temática de juegos y el hecho de tener medidas fiables de la eficacia de sus razonadores, lo que incluye una componente de competitividad entre los grupos. Las memorias de final de práctica que tuvieron que entregar nos han permitido extraer ciertas conclusiones, que pasamos a detallar.

La mayoría de los grupos (unos 24) dejaron claro que pensaban que la aproximación basada en reglas es buena para resolver el problema, ya que, alegan, el experto humano utiliza este tipo de razonamiento al jugar. Algunos no obstante, expresaron que otro tipo de aproximación también es válida, como el razonamiento basado en casos o razonamiento basado en el uso de probabilidades.

Comparando las prácticas del mundo del Wumpus y la del buscaminas, hemos apreciado que el número y complejidad de las reglas en el primero ha sido inferior que en el del segundo. No creemos que sea debido a que el buscaminas es un problema más complejo por lo que podría deberse a que la práctica del buscaminas ha sido más motivante, por ser más conocida, o por el hecho de enfrentarse a un tablero oculto.

La mayoría de los grupos también se han quejado del alto componente aleatorio del buscaminas. En muchas ocasiones el tablero tiene una configuración en la que no se puede inferir una decisión completamente segura.

También nos han hecho notar que el tiempo máximo permitido para resolver cada tablero en la aplicación JessStrategyTester era demasiado bajo (500 milisegundos). A la hora de definirlo, no teníamos ninguna referencia del tiempo que tardarían los razonadores implementados en resolver un tablero, por lo que tuvimos que hacerlo un poco a ciegas. Durante las prácticas, algunos de los grupos mostraron su preocupación por tener razonadores "lentos" que no podían resolver los tableros de niveles difíciles en ese tiempo.

Por último, algunos grupos hubieran querido tener accesible el código de las aplicaciones Java. De haber sido así, hubieran cambiado, decían, el interfaz del tablero para hacerlo más atractivo al usuario. Uno de nuestros objetivos al separar claramente el acceso al tablero de su almacenamiento fue precisamente que los alumnos pudieran centrarse por completo en la extracción del conocimiento y su implementación. Sin embargo, a posteriori, y teniendo en cuenta sus comentarios, creemos que puede ser interesante hacer el código accesible, por los siguientes motivos:

- Tienen acceso al código fuente de un programa Java que utiliza Jess.
- En caso de dudas sobre el interfaz con el tablero, pueden utilizar el código fuente como documentación.
- Pueden cambiar o ampliar los programas, especialmente los de pruebas automáticas para sus propios propósitos. Por ejemplo, algunos de los grupos tenían estrategias "parametrizadas" por niveles umbral de probabilidad y cosas semejantes. Otros en la memoria ponían los resultados para distintos tiempos máximos de resolución. Si hubieran tenido accesible el código fuente, la generación de estadísticas para distintos valores de estos parámetros les hubiera resultado mucho más sencilla.

7. Trabajo relacionado

El uso de juegos para prácticas en Informática se ha utilizado tradicionalmente como medio de motivación. Una revisión interesante del uso que se puede hacer a algunos juegos típicos puede consultarse en [1].

Sin embargo, lo más habitual es utilizarlos en los primeros años de la carrera para enseñar programación, en cursos equivalentes a CS1 y a CS2 del currículum de ACM. El juego del buscaminas no se ha escapado a estos usos, ya que resulta útil para distintos temas de ambas asignaturas, como programación procedimental, operaciones sobre vectores bidimensionales, conceptos de orientación a objetos y gestión de eventos en GUIs [9].

En cuanto a métodos de Inteligencia Artificial para resolver el buscaminas, existen algunas publicaciones sobre distintas estrategias [10] o incluso aproximaciones que consideran qué paradigma de programación es más adecuado [11]. Sin embargo, no se ha utilizado demasiado como práctica de Inteligencia Artificial. Obviamente, hay algunas excepciones. John D. Ramsdell tiene un trabajo interesante llamado PGMS (*Programmer's Minesweeper*[3]). Proporciona un conjunto de clases para jugar al buscaminas. Cualquier programador puede en Java implementar un interfaz que resuelva el tablero de un buscaminas. Utilizando un método muy similar al nuestro, proporciona acceso al tablero que se encarga de dibujarse en pantalla después de cada operación. En la página Web hay disponible un applet en el que se pueden ver en ejecución distintas estrategias. Sin embargo, la idea es implementar los razonadores utilizando Java.

Algo más parecido a nuestra práctica es la propuesta en [12], donde crearon un lenguaje que llamaron S4 (de *SmartSweeper Specification Syntax*), con el que los alumnos pueden programar sistemas basados en reglas utilizando un lenguaje específicamente diseñado para hacer reglas para el buscaminas.

Por último, [13] es la más parecida a nuestro enunciado. Utiliza el buscaminas y Jess como práctica para la asignatura de sistemas basados en el conocimiento. No obstante, no parece que tengan ninguna herramienta para probar automáticamente la estrategia programada con varios tableros.

8. Conclusiones

En este artículo hemos descrito las prácticas propuestas para la asignatura de Ingeniería de Sistemas Basados en Conocimiento en la parte correspondiente a razonamiento con sistemas basados en reglas.

[3] http://www.ccs.neu.edu/home/ramsdell/pgms/

Hemos comprobado un alto nivel de motivación conseguido con prácticas de juegos y con una componente de competitividad entre los grupos. Hemos hecho especial hincapié en la última de las tres prácticas, un razonador para resolver tableros del buscaminas, describiendo a fondo el enunciado de la práctica así como las aplicaciones que les proporcionamos para su desarrollo.

Como conclusión, podemos decir que la implementación tanto del Wumpus como del buscaminas ha tenido una acogida favorable. No obstante, parece que repartir el código Java de las aplicaciones que ayudaron en la implementación de esta última puede favorecer a algunos de los grupos, a pesar de que rompe con nuestro objetivo principal, que era que los alumnos pudieran centrarse en la implementación del conocimiento y dejar de lado los temas del interfaz.

Agradecimientos

Financiado por el Ministerio de Educación y Ciencia (TIN2005-09382-C02-01).

Referencias

[1] K. Becker and J. R. Parker, "All I Ever Needed to Know About Programming, I Learned From Re-writing Classic Arcade Games," presented at International Academic Conference on the Future of Game Design and Technology, Michigan, USA, 2005.

[2] K. Claypool and M. Claypool, "Teaching Software Engineering Through Game Design," presented at 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal, 2005.

[3] M. Kölling and P. Henriksen, "Game Programming in Introductory Courses with Direct State Manipulation," presented at 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal, 2005.

[4] E. Sweedyk and R. M. Keller, "Fun and Games: A New Software Engineering Course," presented at 10th Annual SIGCSE Conference on Innovation and Technology in Computer Science Education, Monte de Caparica, Portugal, 2005.

[5] S. J. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach*, 2nd ed: Prentice Hall, 2002.

[6] G. D. Riley, "CLIPS, a tool for building expert systems."

[7] E. J. Friedman-Hill, "Jess, the Expert System Shell for the Java Platform," 2003.

[8] E. Gamma, R. Helm, R. E. Johnson, and J. Vlissides, *Design Patterns: Elements of Reusable Object-Oriented Software*: Addison Wesley Professional, 1995.

[9] N. Parlante, J. K. Estell, D. Reed, D. Levine, D. Garcia, and J. Zelenski, "Nifty assignments," presented at SIGCSE '02: Proceedings of the 33rd SIGCSE technical symposium on Computer science education, Cincinnati, Kentucky, USA, 2002.

[10] L. P. Castillo and S. Wrobel, "Learning Minesweeper with Multirelational Learning," presented at IJCAI-03, Proceedings of the Eighteenth International Joint Conference on Artificial Intelligence, Acapulco, Mexico, 2003.

[11] R. Collet, "Playing the minesweeper with constraints," presented at Multiparadigm Programming in Mozart/Oz, Second International Conference, MOZ 2004. Revised Selected and Invited Papers, Charleroi, Belgium., 2005.

[12] R. Davis, N. Manokharan, and J. Eisenstein, "Knowledge-based application systems. Assignment 1: Minesweeper," 2005.

[13] F. Kurfess, "Knowledge-Based Systems Assignments: Mine Sweeper in Jess," 2005.