Agente inteligente autónomo que mediante aprendizaje automatizado es capaz de jugar al Sokoban

Luis Arancibia[1], Francisco Pirra[2]

[1]Facultad de Ingeniería, UBA
aran.com.ar@gmail.com

[2]Facultad de Ingeniería, UBA
fpirra@lsia.fi.uba.ar

Resumen. Como trabajo final de la materia: "Sistemas de Soporte para Celdas de Producción Flexible", se propuso crear un Agente (Sistema inteligente capaz de jugar a un videojuego, en este caso el Sokoban). Para ello, se detalla en este informe el proceso de investigación realizado, y los métodos utilizados. Se utilizó como referencia la competencia "The General Video Game AI Competition".

Keywords: agente, inteligencia artificial, fiuba, pirra, arancibia, gvgai

1 Introducción

"Sin afán de sorprenderlos y dejarlos atónitos, debo informarles lisa y llanamente que actualmente en el mundo existen máquinas capaces de pensar, aprender y crear. Además, su capacidad para hacer lo anterior aumentará rápidamente hasta que –en un futuro previsible- la magnitud de problemas que tendrán capacidad de manejar irá a la par con la capacidad de la mente humana para hacer lo mismo. ***(Herbert Simon, 1957, Premio Nobel de economía,1978, por sus trabajos en la Teoría de las decisiones).***

Muchas de las actividades mentales humanas, tales como escribir programas de cómputo, demostrar teoremas, razonar con sentido común y aún conducir un vehículo, normalmente se dice que requieren "inteligencia". Durante las décadas pasadas se ha logrado construir programas que pueden realizar tareas como esas. Existen programas que pueden diagnosticar enfermedades, resolver ecuaciones diferenciales en forma simbólica, analizar circuitos electrónicos, entender en forma limitada frases habladas y escritas o escribir programas de cómputo cumpliendo especificaciones. Se puede decir que tales sistemas poseen cierto grado de Inteligencia Artificial (IA).

Para el desarrollo del presente trabajo, se utilizó como lenguaje de programación JAVA, ya que el framework proveído por la competencia GVG-AI[1], la cual utilizamos como referencia para el armado de este trabajo, está programado en JAVA.

[1] El concurso GVG-AI explora el problema de la creación de controladores para videojuegos en general.

2 Estado del arte

Es sabido que los primeros estudios con repercusión en el mundo de la ciencia acerca de la inteligencia artificial, fueron publicados por Alan Turing. Sin embargo existen estudios acerca de esto desde la década de los 40. Los cuales no causaron el interés que si logro Alan Turing en 1950 con su artículo "Computing Machinery and Intelligence" donde propone que la máquina puede imitar el comportamiento de un ser humano, en este mismo artículo propone el conocido método del "Test de Turing" que sirve para saber si una determinada máquina es inteligente o no. Y no sería hasta 1956 que se establecería la Inteligencia Artificial como una nueva ciencia en la conferencia de "Dartmouth Summer Research Conference on Artificial Intelligence". A partir de esta conferencia, se intentaron llevar a cabo diversas aplicaciones como por ejemplo: traducción automática, juegos, reconocimiento de formas, integración simbólica, demostración de teoremas, resolución de problemas generales, etc. Posteriormente entre los 50's y 60's se escribirían los primeros programas de IA, incluyendo juegos de damas. Según Aiolli y Enrico, el primer videojuego creado fue un simple juego de tenis para dos desarrollado por Higginbotham en 1958. Sin embargo el primer juego que aprendía por sí solo fue el de ajedrez, teniendo ya cierta base de lo que es la IA hoy en día.

A lo largo de la historia de la IA diferentes disciplinas, como la filosofía, las matemáticas, neurociencias, la psicología, etc.; han contribuido tanto para el desarrollo de nuevas técnicas como para sentar las bases de lo que es la IA hoy en día y todo lo que este término abarca. Además han surgido y seguramente seguirán surgiendo nuevos paradigmas sobre la IA, entre los cuales debemos resaltar los siguientes:

- **Procesamiento Simbólico** Paradigma presente durante los años 60 y 80 que cree que la mente humana es una máquina de procesamiento de información a partir de la simbología presente en su entorno. LISP fue el primer lenguaje para procesamiento simbólico.
- **Búsqueda Heurística** Es la utilización de la programación heurística para buscar y determinar caminos óptimos hacia el logro de una meta u objetivo de un problema no algorítmico o con algoritmos que generan explosión combinatoria como los juegos de damas o ajedrez.
- **Sistemas Expertos** Los sistemas expertos tienen una inclinación hacia el lado funcional ya que en lugar de dar solución a problemas generales, estos analizan la actividad de una persona experta en una actividad y tratan de emular su comportamiento y ayudarle en el desarrollo de dicha actividad, además estos sistemas deben ser capaz de, que al igual que el experto humano, adquirir experiencia.
- **Sistemas Basados en Conocimientos** Estos sistemas abarcan el concepto de sistemas expertos pero de un modo más amplio. Los sistemas basados en conocimientos, adquieren el conocimiento de un agente externo (humano), este lo almacena y debe ser capaz de aplicarlo según sea necesario.
- **Agentes Inteligentes** Un agente inteligente es una entidad que es capaz de percibir e interactuar con su entorno y se caracteriza por ser reactivo, capaz de responder a los cambios de su entorno; proactivo, capaz de intentar cumplir sus propias metas; y social, capaz de comunicarse con otros agentes mediante algún tipo de lenguaje de comunicación entre agentes.

Por lo tanto, como mencionamos anteriormente, en este trabajo nos enfocamos en la construcción de un agente inteligente.

3 Descripción del problema

3.1 Sokoban

Sokoban es un clásico rompecabezas inventado en Japón, normalmente implementado como videojuego. El juego original fue creado por Hiroyuki Imabayashi, que en 1980 ganó con su juego una competición contra un ordenador. Hiroyuki Imabayashi es presidente de la empresa Thinking Rabbit Inc. en Japón. Con los años han aparecido muchas versiones del juego para todas las plataformas, y continuamente se crean nuevas colecciones de niveles.

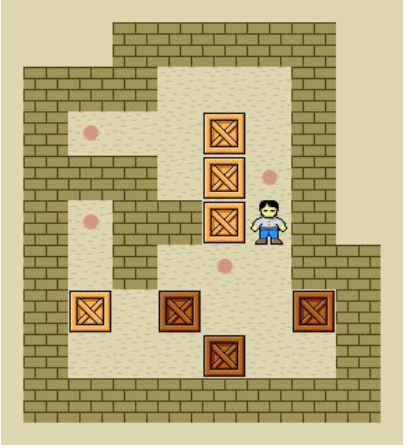

Fig. 1. Imagen ilustrativa del juego Sokoban

Sokoban significa "encargado de almacén" en japonés. El objetivo del juego es empujar las cajas (o las bolas) hasta su lugar correcto dentro de un reducido almacén, con el número mínimo de empujes y de pasos. Las cajas se pueden empujar solamente, y no tirar de ellas, y sólo se puede empujar una caja a la vez. Parece fácil, pero los niveles van desde muy fáciles a extremadamente difíciles, y algunos lleva horas e incluso días resolverlos. La simplicidad y la elegancia de las reglas han hecho de Sokoban uno de los juegos de ingenio más populares.

3.2 Agente inteligente

En base a las “simples” reglas del juego Sokoban, se pretende armar un Agente inteligente que sea capaz de jugarlo y pasar los niveles por sí mismo, sin la ayuda de un ser humano.
Es por esto, que vamos a utilizar de referencia la competencia de GVG-AI, de modo que el bot generado por nosotros, pueda ejecutarse en el entorno proveído por el concurso online.

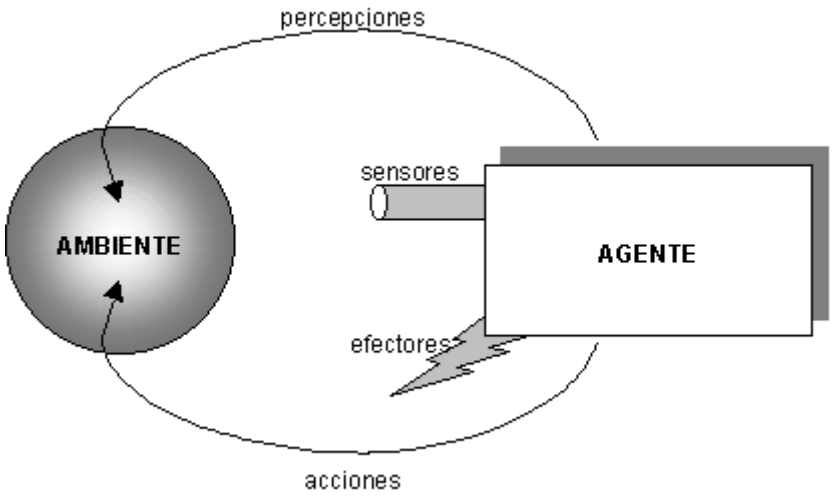

Fig. 2. Esquema del funcionamiento del sistema, al utilizar un agente

Es vital, para la culminación de este trabajo, que el agente sea capaz de pasar, al menos, el primer nivel propuesto, de forma independiente.

4 Desarrollo de la solución

4.1 Toma de decisiones

Para el desarrollo del Agente, lo primero que se pensó es: Cómo lograr que pueda tomar decisiones, en base al estado en que se encuentra?

De modo que decidimos generar una base de conocimiento con "teorías" precargadas, de forma que el sistema pueda evaluar, en base a la situación actual, que acción es preferible tomar.

Debido a que es imposible predecir todas las situaciones posibles, y en cada caso cuál es la mejor opción a tomar, se planteó un set de teorías básico, para que le sirva de inicio al sistema.

Para la decisión de que acción tomar, se planteó un método que evalúe, según el estado actual y las posibles alternativas a realizar, cuál es el mejor panorama.
Para lograr esto, se utilizó un árbol de decisión, el cual nos permite brindarle al Agente una forma de evaluar que tan útil es cada posible acción a tomar.

Fig. 3.1 Arbol de decision

Si al buscar en su base de conocimiento, no encuentra ninguna teoría con la que pueda contrastar el medio sensado en su situación actual, se va a realizar un movimiento aleatorio y almacena el movimiento para futuros aprendizajes.

Por otro lado, si encuentra en su base de conocimiento la misma situación en la que se encuentra, elije la acción a tomar, en base a una ponderación del valor de utilidad (**u**), según la función incorporada, y teniendo en cuenta la cantidad de veces que se utilizó (**k**), contrastado con la cantidad de veces que se utilizó con éxito (**p**).

4.2 El proceso de aprendizaje

El proceso de "aprendizaje" basado en teorías, va a seguir el siguiente flujo:

- Cuando el Agente está en una "situación" nueva, se va a crear una teoría temporal, la cual estará formada por la situación anterior, los efectos y la situación final
- Luego se busca contra la base de teorías, para encontrar una que refleje el mismo escenario. En este punto se pueden presentar dos situaciones:
    - Que la teoría no se encuentre en la base: Entonces se agrega directamente al archivo.
    - Que la teoría si se encuentra en la base: En este caso, también tenemos dos alternativas:
        - Que los efectos coincidan: Se aumenta el valor de **p** y **k** a la teoría de la base. Osea los valores de cuántas veces se utilizó y cuántas con éxito.
        - Que los efectos no coincidan: Se aumenta solo el valor de **k**

La idea de esto, es hacer que el Agente pueda aprender, y así ir mejorando en cada partida su funcionamiento.

Para ayudar en este aprendizaje, se utilizó la base precargada de teorías, de modo que se pudieran evitar la mayor cantidad de casos, en los que de antemano sabemos que va a fallar. Para lograr esto, simplemente se coloca en la teoría ambas situaciones, los efectos y un valor de utilidad igual a cero.

4.3 Mejoras al aprendizaje

Para mejorar y agilizar el aprendizaje, como antes mencionamos, se crearon lo que llamamos teorías precargadas. De esta forma, podemos evitar casos simples de atascamiento, y hacer que el Agente llegue lo más lejos posible en su aprendizaje.

A continuación se muestran algunos casos, en los que se pueden elaborar teorías "precargadas"

En la figura 3.1, podemos ver que estamos en el inicio del nivel, donde prácticamente lo único que podemos hacer es mover la caja hacia la izquierda o movernos por los casilleros vacíos, si somos el caballero, y si somos el mago, también podemos movernos por los casilleros vacíos, o bien llegar hasta la caja que esta alado del caballero y empujarla, dejándola inutilizable.

De modo que, claramente cualquier movimiento de parte del mago seria una perdida de procesamiento, y el único movimiento posible para el caballero que puede generar un avance al juego es mover la caja hacia la izquierda. Por lo tanto elaboramos una teoría que tenga un valor cercano a 1 en utilidad, alentando a empujar esa caja.

Fig. 4.1 Comienzo del nivel 2

Por otro lado, puede ocurrir que no se tenga en cuenta que la caja que está “debajo” no se debe empujar hacia abajo, ya que quedaría inutilizable. De modo que habría que generar también una teoría que contemple estos casos, sino vamos a terminar teniendo una situación como la que se muestra en la figura 3.2.

Fig. 4.2 Situación en la que se coloco correctamente una caja, pero la otra será imposible de quitar

Por otro lado, cambiamos de nivel y podemos observar algunos casos en donde será crítico tener establecidas teorías que contemplan el hecho de que no se empuje la caja para ciertos lugares, ya que la dejarian inutilizable. Le dejamos al lector, la difícil tarea de encontrar cuales son estos movimientos prohibidos.

Fig. 5.1 Ilustración de un caso, en el que no se debe mover a derecha el Agente (caballero)

Fig. 5.2 Ilustración de un caso, en el que no se debe mover a derecha el Agente (caballero)

4.4 Detalles de implementación

En cuanto a la visión del bot, se la limitó a 3 casilleros de "radio" del agente, lo que le permite tomar los objetos a su alrededor y poder determinar en que situación se encuentra. Dado este rango de visión, y con las distintas posibilidades que podemos encontrar en el mapa, se presenta la siguiente tabla de definiciones:

Simbolo	Descripción
A	Agente A
B	Agente B
.	Casillero vacío
W	Pared (inmóvil)
0	Destino
1	Caja (móvil)
?	Objeto desconocido

Tabla 1.1 Se muestran los distintos símbolos posibles y su significado

Como detalle importante a resaltar, se utilizó la clase provista por Juan Manuel Rodriguez, "Perception.java", para facilitar el procesamiento del estado del escenario.

También se modificó el tiempo de desclasificación del Agente, de 50 milisegundos a 500 milisegundos, debido a que a medida que el sistema inteligente posee más teorías, el tiempo de procesamiento aumenta. Si bien no es infinito esto, puede crecer bastante, y el programa termina eliminando al Agente.

Por último, para el uso de la base de conocimiento (o teorías), se utilizó el algoritmo de Dijkstra, siendo el peso de las aristas de las distintas situaciones entre teorías, es la utilidad de las mismas, osea el valor **u**.

4.5 Tecnologías utilizadas

En cuanto a la tecnología, se utilizó git como sistema de versionado de código.

Se utilizó el framework proveído por los creadores de la competencia GVG-AI, y en base al mismo se desarrolló el Agente, utilizando el lenguaje JAVA.

Por ultimo, se utilizó el IDE Eclipse, para el desarrollo del Agente.

5 Conclusion

Se logró construir un sistema inteligente, que sea capaz de jugar y aprender a resolver el Sokoban.
Dada la dificultad de contemplar la extensa cantidad de situaciones posibles, y crear teorías, solamente podemos afirmar que el Agente es capaz de resolver niveles sencillos, donde no se requiere una excesiva lógica.

Fue muy interesante ver, que en el caso de hacer competir dos Agentes que tuvieran el mismo código, pudieron aprender a "interactuar", si es que podemos darnos el lujo de llamarlo así, y que entre ambos puedan resolver el nivel. El aprendizaje llevó más tiempo, y no podemos afirmar que en todos los casos que jueguen, resuelvan correctamente el puzzle, ya que al "variabilidad" en ambos agentes, las opciones se multiplican, y cada uno toma decisiones independientes (Lo que nos parece que lleva a concluir lo interesante e infinito en posibilidades que es la construcción de sistemas inteligentes).

Por otro lado, cuando se jugó con un Agente "inteligente" y otro que simplemente se quedaba quieto, luego de un poco de entrenamiento, la capacidad de resolución del nivel para el bot es instantánea. En este caso, sí podemos afirmar que tenemos un Agente que sabe resolver correctamente y de forma veloz, el puzzle de Sokoban.

6 Futuras líneas de estudio

Para futuros trabajos en el ámbito de la inteligencia artificial aplicada a resolución de videojuegos, proponemos los siguientes ítems:

- Generalización de comportamiento, para cualquier juego
- Creación de teorías específicas para cada juego, y en base a esto, busqueda de interconexión en esas teorias de modo que se puedan encontrar patrones que permitan la creación de Agentes lo más independientes posibles, incluso del tipo de juego.
- Uso de algoritmos genéticos para el desarrollo de niveles, cada vez más extensos y complejos
- Mejora del proceso de aprendizaje, de forma que sea automatico y pueda decidir en que momento dejar de aprender (en base a los conceptos de overfitting y underfitting)

7 Referencias

[1] Fritz, W., García Martínez, R., Rama, A., Blanqué, J., Adobatti, R. y Sarno, M.. (1989). *The Autonomous Intelligent System*.

[2] J. Togelius, S. M. Lucas, and R. De Nardi. “Computational intelligence in racing games”. In: Advanced Intelligent Paradigms in Computer Games. Springer, 2007, pp. 39–69.

[3] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius. “Matching Games and Algorithms for General Video Game Playing”. In: AIIIDE. 2016.

[4] I. Bravi, A. Khalifa, C. Holmg˚ard, and J. Togelius. “Evolving UCT Alternatives for General Video Game Playing”. In: The IJCAI-16 Workshop on General Game Playing, p. 63.

[5] T. Cazenave. “Evolving Monte Carlo tree search algorithms”. In: Dept. Inf., Univ. Paris 8 (2007).

[6] G. Chaslot. “Monte-carlo tree search”. In: Universiteit Maastricht (2010).
[7] M. Ebner, J. Levine, S. M. Lucas, T. Schaul, T. Thompson, and J. Togelius. “Towards a video game description language”. In: (2013).

[8] F. Frydenberg, K. R. Andersen, S. Risi, and J. Togelius. “Investigating MCTS modifications in general video game playing”. In: Computational Intelligence and Games. IEEE. 2015, pp. 107–113.

[9] R. D. Gaina, D. P´erez-Li´ebana, and S. M. Lucas. “General Video Game for 2 Players: Framework and Competition”. In: ().

[10] P. Hingston. Believable Bots. Springer, 2012.

[11] E. J. Jacobsen, R. Greve, and J. Togelius. “Monte mario: platforming with MCTS”. In: Proceedings of the 2014 Annual Conference on Genetic and Evolutionary Computation. ACM. 2014, pp. 293–300.

[12] A. Khalifa, D. Perez-Liebana, S. M. Lucas, and J. Togelius. “General Video Game Level Generation”. In: ().

[13] A. Mendes, A Nealen, and J Togelius. “Hyperheuristic general video game playing”. In: IEEE Computational Intelligence and Games (2016).

[14] S. Ontan´on, G. Synnaeve, A. Uriarte, F. Richoux, D. Churchill, and M. Preuss. “A survey of real-time strategy game ai research and competition in starcraft”. In: IEEE Transactions on Computational Intelligence and AI in games 5.4 (2013), pp. 293–311.

[15] D. Perez, S. Samothrakis, J. Togelius, T. Schaul, S. Lucas, A. Cou¨etoux, J. Lee, C.-U. Lim, and T. Thompson. “The 2014 general video game playing competition”. In: (2015).

[16] D. Perez-Liebana, S. Samothrakis, J. Togelius, S. M. Lucas, and T. Schaul. “General Video Game AI: Competition, Challenges and Opportunities”. In: Thirtieth AAAI Conference on Artificial Intelligence. 2016.

[17] J. Pettit and D. Helmbold. “Evolutionary learning of policies for MCTS simulations”. In: Proceedings of the International Conference on the Foundations of Digital Games. ACM. 2012, pp. 212–219.

[18] M. Preuss. “Adaptability of Algorithms for Real-Valued Optimization”. In: Applications of Evolutionary Computing. Springer, 2009, pp. 665–674. [19] J. Renz. “AIBIRDS: The Angry Birds Artificial Intelligence Competition.” In: AAAI. 2015, pp. 4326–4327.

[20] D. J.N. J. Soemers, C. F. Sironi, T. Schuster, and M. H. M. Winands. “Enhancements for Real-Time Monte-Carlo Tree Search in General Video Game Playing”. In: Proceedings of the IEEE Conference on Computational Intelligence and Games. 2016.