Micromanagement basado en formaciones de grupo implementado con scripting dinámico

Daniel Trevisani[1] and Laura Cecchi[1]

[1] *Grupo de Investigación en Lenguajes e Inteligencia Artificial*
Departamento de Teoría de la Computación
Facultad de Informática
Universidad Nacional del Comahue
Buenos Aires 1400 - (8300)Neuquén - Argentina

{daniel.cipo,lcecchi}@gmail.com

Resumen El éxito de una partida en un juego de estrategia en tiempo real (RTS) depende de la capacidad de cooperación y reacción en forma adaptativa, ante los movimientos del oponente, por parte del jugador. En muchos videojuegos, esto se logra a través del uso de las llamadas formaciones de "unidades". El propósito de este trabajo es presentar el diseño de un micromanagement de las formaciones de grupo de agentes que se desenvuelven en un ambiente dinámico y que permite adaptar en forma "online" su comportamiento. La toma de decisión del agente propuesto que decide la táctica del juego, se realiza a partir de un árbol de decisión, codificado en XML, que es generado por la herramienta Weka, lo que permite automatizar el proceso de generación del árbol de decisión a partir de una base de datos. Asimismo, se presenta el diseño e implementación de un traductor que a partir del árbol de decisión en formato XML genera reglas simples IF-THEN. Los cambios "online" que se producen en el comportamiento de las formaciones, se reflejan en el juego utilizando la técnica scripting. Dicho micromanagement fue implementado, a partir de una arquitectura propuesta de videojuegos RTS, sobre la plataforma de StarCraft: Brood War.

1. Introducción

En la actualidad, los videojuegos son de gran importancia dentro de la industria del entretenimiento. Sin embargo, su aplicación ha ido mas allá del divertimento, encontrando utilidad en otros campos como el entrenamiento militar y/o deportivo, educación y en general ludificación (gamification) [3,2,7,16]. Existen características comunes en los juegos, como las severas restricciones de tiempo y la fuerte demanda de Inteligencia Artificial (IA) en tiempo real, la cual debe ser capaz de resolver tareas de decisión de manera rápida y satisfactoria.

En un videojuego los jugadores interactúan con las diferentes entidades presentes en el mundo virtual proporcionadas por el entorno. La calidad de esta

interacción está asociada, en buena parte, a la inteligencia mostrada por los personajes controlados por la computadora (de ahora en más NPC, por sus siglas en inglés, Non-Player Character) que habitan el mundo virtual.

En el género de los juegos RTS (Real Time Strategy), la inteligencia de los NPCs debe ser lo suficientemente abarcativa de manera que exhiba coordinación en las acciones de las unidades, toma de decisiones estratégicas y gestión efectiva de los recursos y las unidades. Sin embargo, la mayoría de los videojuegos RTS exhiben un motor de Inteligencia Artificial (o Inteligencia Artificial del Juego, IAJ) de baja calidad, presentando acciones y estrategias por defecto, las cuales son percibidas rápidamente por el jugador humano.

A su vez, el éxito de una partida en un juego depende de la capacidad de cooperación y reacción, ante los movimientos del oponente, por parte del jugador. En muchos videojuegos, esto se logra a través del uso de las llamadas formaciones de "unidades" [15]. La gestión de unidades de combate, comúnmente denominada *micromanagement*, tiene el propósito de maximizar el daño causado al enemigo y minimizar el propio. Las formaciones de grupo pueden proveer estas capacidades a través de movimientos cohesivos de grupo [15].

En un ambiente dinámico, como lo es el de los juegos RTS, es fundamental que el comportamiento de las formaciones propias responda a cómo se presenta el oponente durante el combate y se adapte en tiempo real al comportamiento mostrado por el oponente.

En este trabajo se presenta el diseño de un micromanagement de las formaciones de grupo de agentes que se desenvuelven en un ambiente dinámico y que permite adaptar su comportamiento. La toma de decisión del agente propuesto que decide la táctica del juego, se realiza a partir de un árbol de decisión, codificado en XML. El formato del árbol de decisión se corresponde con el generado por la herramienta WekatextToXML [12]. Esta herramienta convierte un árbol de decisión producido por la herramienta Weka [6] en un archivo XML. Esto permite automatizar el proceso de generación del árbol de decisión a partir de una base de datos, utilizando un reconocido sistema. Asimismo, se presenta el diseño e implementación de un traductor que a partir del árbol de decisión en formato XML genera reglas simples IF-THEN. Dicho micromanagement fue implementado, a partir de la arquitectura propuesta de videojuegos RTS, sobre la plataforma de StarCraft: Brood War [1]. Este juego fue seleccionado por su éxito comercial y por ser reconocido en el ámbito académico.

Se utilizó la técnica de scripting para reflejar los cambios "online" que se producen en el comportamiento de las formaciones. Esto es, los cambios que se produzcan al editar el archivo XML se reflejan en la actualización de las reglas que conforman el scripting dinámico.

En [13], se propone la utilización de scripting dinámico para reflejar cambios sobre el comportamiento individual de los NPCs, a diferencia de nuestro trabajo, donde los cambios se producen sobre la formación. Por otra parte, aquel trabajo implementa las variaciones en el comportamiento con modificando las prioridades(peso) de las reglas que contiene a priori. En nuestro trabajo, los cambios pueden introducir reglas nuevas modificando el árbol de decisión.

El resto de este trabajo está estructurado como sigue. En la sección 2, se introduce la arquitectura propuesta para juegos RTS, describiendo cada uno de los módulos. La sección 3, presenta el diseño del agente Manager Táctico que es el encargado de las decisiones. En la sección 4, se explican los detalles de la implementación del comportamiento adaptativo de los grupos utilizando scripting. Finalmente, en la última sección se presentan las conclusiones y los trabajos futuros.

2. Arquitectura Propuesta

Basados en el modelo arquitectónico genérico propuesto por Millington y Funge en [10], se diseñó una arquitectura para juegos RTS, que se puede apreciar en la Fig.1. A diferencia de la presentada en [10], se distinguen la capa estratégica y táctica del juego, diferenciada de la de personaje y se incluyen los módulos *modelo del oponente* y *manager de recursos*.

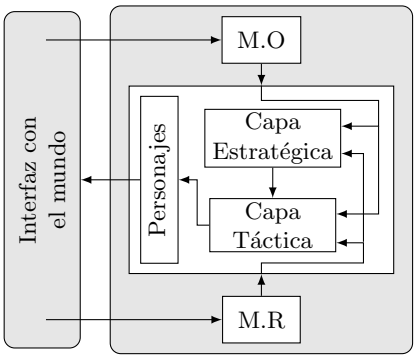


Figura 1. Arquitectura propuesta del juego

A continuación, describimos cada uno de los módulos que componen la arquitectura.

Capa Estratégica

Determina la estrategia a seguir de una formación, para esto se contemplan dos decisiones estratégicas diferentes que una formación puede encarar: ataque y defensa. Para cada decisión estratégica se corresponderá una o más decisiones tácticas. Algunas estrategias propuestas en [8] incluyen, "unirse y defender la base más cercana", "ataque en masa"y "ataque desplegado", entre otras.

Capa Táctica

El propósito de esta capa es elegir, en función de lo decidido por la Capa Estratégica, la formación más adecuada para los grupos de combate en cada

situación de juego. Para lograr su propósito, la Capa Táctica utiliza dos entradas de información: el modelo del oponente generado por el módulo M.O y los datos actuales sobre los recursos provistos por el módulo M.R.

Es esperable que los personajes en un videojuego posean capacidad de reacción y actúen de manera cooperativa en pos de exhibir un comportamiento que desafíe al jugador humano. Por esta razón, consideramos apropiado emplear una metodología basada en agentes para abordar nuestros desarrollos.

En este trabajo se eligió para el análisis y diseño de los agentes a la metodología Gaia [17], dado que aspectos fundamentales, como autonomía, reactividad y proactividad, pueden ser expresados de manera clara y precisa. Además, el concepto clave de esta metodología, el rol, es adecuado si consideramos que nuestro objetivo es modelar los distintos roles que desempeñan los personajes presentes en un juego RTS.

Los roles surgidos de la etapa de análisis son: *"líder de formación"*, *"observador"*, *"scout"*, *"soldado"* y *"manager táctico"*.

Estos roles fueron modelizados teniendo en cuenta que nuestro objetivo final es el comportamiento coordinado de los agentes. En este sentido, la toma de decisión sobre qué formación adoptar de acuerdo a lo observado en el ambiente, la realiza el "manager táctico", a partir de un árbol de decisión.

Personajes

Este módulo se encarga de ejecutar las acciones necesarias a nivel de personaje, para establecer en el juego la formación elegida por la Capa Táctica. Cada rol requerido en la Capa Táctica es diseñado e implementado en este módulo, a través de los servicios que ofrece.

Modelo del Oponente (M.O)

En juegos con información imperfecta, como los juegos RTS, un jugador no puede realizar suposiciones estándares sobre los oponentes.

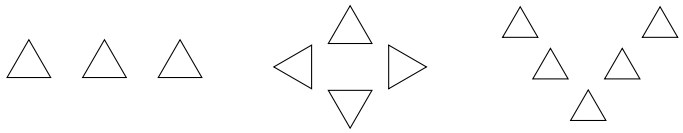
Un factor importante que influencia la elección de una estrategia en particular, es la estrategia del oponente. Por ejemplo, si el jugador conoce qué tipo de unidades posee el oponente, entonces es esperable pensar que el jugador elija construir unidades que sean más fuertes que las del oponente. Para realizar predicciones acerca de la estrategia del oponente, debe ser posible establecer un modelo del oponente. Sin embargo, realizar esta tarea en un juego RTS representa un desafío particular ya que se carece de información perfecta acerca del entorno de juego. Ledezma et al. [9] presentan un enfoque para generar modelos del oponente de manera automática en entornos con información imperfecta.

Manager de Recursos (M.R)

Este módulo provee información sobre los recursos económicos disponibles (procesados y no procesados o potenciales), las unidades, de combate y de trabajo y las construcciones realizadas.

3. Agente Manager Táctico

El éxito de una partida de juego depende de la capacidad de cooperación y reacción, ante los movimientos del oponente por parte del jugador. En muchos juegos esto es implementado a través de las *formaciones*. Una formación puede definirse como una disposición de unidades. Entre las formaciones más comúnmente empleadas podemos citar a la formación en "V", la formación en *cuña* o "V" invertida, la formación "fila", y la formación "circular", entre otras[4,15]. Algunas de ellas se muestran en la Figura 2.



(a) Filas (b) Circular (c) Vee

Figura 2. Algunas de las formaciones más comunes

Las unidades dentro de la formación del grupo deberán seleccionar de manera inteligente la unidad oponente a atacar. Las reglas que establecen esta selección se describen en un *esquema de ataque*. Existen diversos esquemas propuestos en [15] y en [14], que incluyen la selección del más cercano, la selección del líder, la selección relativa y la selección jerárquica, entre otras.

Dentro de la capa Táctica se desarrolló el agente *Manager Táctico* encargado de manterner coordinado al grupo de unidades y de decidir la formación y el esquema de ataque. En este sentido, se caracterizó a las formaciones como grupo de agentes y se definieron posibles operaciones.

Definition 1. [Grupo de Combate]
Definimos a un grupo de combate $\mathcal{G}_\mathcal{C}$ mediante la estructura $\langle S_L, L, S_R, F, E\rangle$ donde S_L y S_R son secuencias finitas, posiblemente vacías, de agentes a izquierda y derecha, respectivamente, del líder L, con $L \notin S_L$ y $L \notin S_R$. F y E indican, el tipo de formación utilizada y el esquema de ataque a emplear por el grupo, respectivamente.

A continuación mostramos algunos aspectos de la notación empleada:

- $[\,]$ denota la secuencia vacía.
- ST denota la concatenación de las secuencias de agentes S y T.
- $S[Ag]$ denota la secuencia obtenida agregando el agente Ag a la secuencia de agentes S en la última posición. Consecuentemente, $[Ag]S$ denota la secuencia obtenida agregando el agente Ag a la secuencia de agentes S en la primer posición. Si $S = [\,]$ entonces $S[Ag] = [Ag]S = [Ag]$
- $|S|$ denota la longitud de una secuencia finita S, y S_i denota al i-ésimo elemento de S, para $1 \leq i \leq |S|$. En particular, $|[\,]| = 0$.

Sobre esta estructura necesitaremos definir operaciones que contemplen las situaciones de juego que serán habituales para una formación de grupo. Esto es, esperamos que durante el desarrollo del juego, el líder de una formación deba ser reemplazado, o un nuevo soldado deba incorporarse o incluso que una formación deba cambiar su esquema de ataque y su disposición de las unidades sobre el terreno.

El reemplazo del líder requerirá que un agente de las secuencias a izquierda o derecha ocupe su lugar. A fin de no desbalancear las secuencias de agentes, el nuevo líder será elegido de aquella secuencia que posea mayoría de agentes con respecto a la otra, o en caso de igualdad, será elegido de la secuencia a izquierda.

Definition 2. [Reemplazo del líder]
Sean $\mathsf{G_C}$ el conjunto de grupos de combate y $\mathsf{G'_C} \subseteq \mathsf{G_C}$ el conjunto de grupos de combate que verifican $|S_L|+|S_R| \geq 1$, entonces definimos la función reemp_líder: $\mathsf{G'_C} \rightarrow \mathsf{G}$ como:

$$\textit{reemp\_líder}(\langle [Ag_{\ell m} Ag_{\ell m-1} \cdots Ag_{\ell 1}], L, [\,], F, E\rangle) = \langle [Ag_{\ell m} Ag_{\ell m-1} \cdots Ag_{\ell 2}], Ag_{\ell 1}, [\,], F, E\rangle$$

$$\textit{reemp\_líder}(\langle [\,], L, [Ag_{r1} \cdots Ag_{rn-1} Ag_{rn}], F, E\rangle) = \langle [\,], Ag_{r1}, [Ag_{r2} \cdots Ag_{rn-1} Ag_{rn}], F, E\rangle$$

$$\begin{array}{l}\textit{reemp\_líder}(\langle [Ag_{\ell m} Ag_{\ell m-1} \cdots Ag_{\ell 1}], L, [Ag_{r1} \cdots Ag_{rn-1} Ag_{rn}], F, E\rangle) = \\ \quad \langle [Ag_{\ell m} Ag_{\ell m-1} \cdots Ag_{\ell 2}], Ag_{\ell 1}, [Ag_{r1} \cdots Ag_{rn-1} Ag_{rn}], F, E\rangle \ \textit{si}\ m \geq n \\ \quad \langle [Ag_{\ell m} Ag_{\ell m-1} \cdots Ag_{\ell 1}], Ag_{r1}, [Ag_{r2} \cdots Ag_{rn-1} Ag_{rn}], F, E\rangle \ \textit{si}\ m < n\end{array}$$

En caso de bajas en el combate, el manager táctico puede decidir incorporar soldados de otro grupo a la formación.

Definition 3. [Incorporar soldado]
Sean $\mathsf{G_C}$ el conjunto de grupos de combate y AG el conjunto de agentes, entonces definimos la función incorp_soldado: $\mathsf{G_C} \times \mathsf{AG} \rightarrow \mathsf{G_C}$, como:

incorp_soldado$(\langle S_L, L, S_R, F, E\rangle, A) = \langle [A]S_L, L, S_R, F, E\rangle$, con $A \in \mathsf{AG}$ y $|S_R| \geq |S_L|$

incorp_soldado$(\langle S_L, L, S_R, F, E\rangle, A) = \langle S_L, L, [A]S_R, F, E\rangle$, con $A \in \mathsf{AG}$ y $|S_R| < |S_L|$

Nótese que al reemplazar un líder, el líder desaparece de la formación. Esto tiene su razón en que el reemplazo siempre se debe a la baja del líder en combate. Sin embargo, el diseño permite que un líder pueda tomar nuevamente rol de soldado dentro de la formación, después de ser reemplazado. Esto se realiza primero reemplazándolo y luego incorporándolo como soldado.

El cambio de formación requerirá la reagrupación de las secuencias de agentes a izquierda y derecha del líder, de manera que éstas se encuentren balanceadas luego de realizar el cambio. A continuación mostramos la definición de la operación 'Cambiar formación' detallando sólo los casos donde la secuencia de agentes S_R supera en cantidad a la secuencia S_L. Los restantes casos pueden encontrarse en [14].

Definition 4. [Cambiar formación]
Sean $\mathsf{G_C}$ el conjunto de grupos de combate, F el conjunto de formaciones y F y F' miembros de F. Definimos la función cambiar_formac.: $\mathsf{G_C} \times \mathsf{F} \rightarrow \mathsf{G_C}$ como:

$$\text{cambiar\_formac.}(\langle S_L, L, S_R, F, E\rangle, F') \;=$$

$$\langle S_1 S_L, L, S'_R, F', E\rangle \text{ si } (|S_R| > |S_L| + 1) \wedge (|S_R| + |S_L| = 2 \times k + 1), \text{ para algún } k \in \mathbb{Z}$$
$$\text{donde } S_R = [Ag_{r1}, Ag_{r2} \cdots Ag_{rn}], S_1 = [Ag_{r1}, Ag_{r2} \cdots Ag_{rj}],$$
$$S'_R = [Ag_{rj+1} \cdots Ag_{rn}]$$
$$\text{con } j = \tfrac{|S_R|-|S_L|-1}{2}$$

$$\langle S_1 S_L, L, S'_R, F', E\rangle \text{ si } (|S_R| > |S_L| + 1) \wedge (|S_R| + |S_L| = 2 \times k), \text{ para algún } k \in \mathbb{Z}$$
$$\text{donde } S_R = [Ag_{r1}, Ag_{r2} \cdots Ag_{rn}], S_1 = [Ag_{r1}, Ag_{r2} \cdots Ag_{rj}],$$
$$S'_R = [Ag_{rj+1} \cdots Ag_{rn}]$$
$$\text{con } j = \tfrac{|S_R|-|S_L|}{2}$$

Finalmente, caracterizamos el cambio en el esquema de ataque, lo que permitirá a cada unidad distinguir al oponente a atacar.

Definition 5. [Cambiar esquema de ataque]
Sean $\mathsf{G_C}$ el conjunto de grupos de combate y E el conjunto de esquemas de ataque, entonces definimos la función cambiar_esquema_ataque: $\mathsf{G_C} \times \mathsf{E} \rightarrow \mathsf{G_C}$, como:

$$\textit{cambiar\_esquema\_ataque}(\langle S_L, L, S_R, F, E\rangle, E') = \langle S_L, L, S_R, F, E'\rangle$$

con E *y* $E' \in \mathsf{E}$.

El agente manager táctico diseñado e implementado, realiza el proceso de toma de decisiones basándose en un árbol de decisión. En la Fig.3 puede verse, a modo de ejemplo ilustrativo, un árbol de decisión en el cual la elección de la formación a emplear depende del cumplimiento de ciertas condiciones de juego. En este árbol, los nodos enmarcados representan los tests a realizar sobre las condiciones de juego, y los nodos hoja o terminales representan las decisiones, es decir, la formación a emplear.

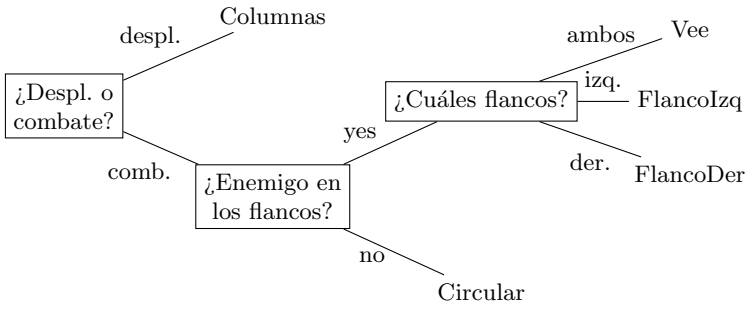


Figura 3. Árbol de decisión para la elección de la formación

4. Implementación del comportamiento adaptativo de grupo utilizando scripting

En este trabajo se utilizaron árboles de decisión como mecanismo de toma de decisiones de nivel táctico. El árbol se compone de los atributos y las decisiones para cada situación de juego descripta. El árbol de decisión está codificado en un archivo XML cuyo aspecto general puede verse en la Fig.4.

```
<DecisionTree type= "tactica-ataque" >
   ...
   <Test attribute="despl-comb" operator="=" value="comb">
     <Test attribute="EnemigoEnFlancos" operator="=" value="yes">
        <Test attribute="CualesFlancos" operator="=" value="ambos">
           <Output decision="Vee"/></Test>
      ...
   </Test>
```

Figura 4. Formato XML del árbol de decisión

El formato del árbol de decisión se corresponde con el generado por la herramienta desarrollada por Luc Sorel, WekatextToXML [12]. Esta herramienta convierte un árbol de decisión producido por la herramienta Weka [6], cuyo formato es texto con sintaxis Weka, en un archivo XML. Consideramos útil la utilización del formato XML producido por la herramienta WekatextToXML, dado que la integración del sistema Weka, un reconocido framework para el análisis de datos y el modelamiento predictivo, nos permitirá automatizar el proceso de generación del árbol de decisión a partir de una base de datos, tal como se muestra en la Fig.5.

Figura 5. Proceso de Traducción

Actualmente, el archivo XML ha sido construido de manera *ad-hoc*, sin embargo, el framework desarrollado permitirá utilizar técnicas de aprendizaje automático para generar el árbol de decisión a partir del modelo del oponente.

Un problema a enfrentar al adaptar comportamiento, es la magnitud y la complejidad del espacio de búsqueda, el que debería ser explorado de manera rápida y eficiente.

El enfoque basado en scripting permite dar solución a parte del problema, ya que, por un lado, los cambios que se realicen no requieren de tiempo de rebuild y por el otro, en muchos casos limitan el tamaño del espacio estados-acciones.

Debido al dinamismo del juego, es posible editar el archivo XML de manera "online" y ver como las modificaciones se reflejan en el juego, luego de que el

archivo es guardado en disco. Cada modificación realizada en el árbol provoca una actualización en las reglas que conforman el script. Esta característica es importante ya que permite cambiar la táctica empleada por los grupos de combate en tiempo de ejecución, evitando así la creación de una nueva partida de juego para tal fin.

El traductor desarrollado [14] permite transformar el árbol de decisión en reglas simples de la forma IF-THEN. Seguidamente, dichas reglas son incorporadas al código del agente a través de scripting.

Para tal fin, en este trabajo se eligió el lenguaje y la interfaz de programación GameMonkey [11]. De esta manera, y de acuerdo a la dinámica del ambiente, es posible extender o modificar el comportamiento del agente sin la necesidad de reconstruir su código. En la Fig.6 se muestra la traducción a reglas IF-THEN del árbol de decisión mostrado en la Fig.3.

```
 if (despl-comb == "comb"){
      if(EnemigoEnFlancos){
        if(CualesFlancos=="ambos"){decision= "Vee";}
        if(CualesFlancos=="izquierdo"){decision= "FIzq";}
        if(CualesFlancos=="derecho"){decision= "FDer";}
      }
      if(!EnemigoEnFlancos){decision= "Circular";}
}
```

Figura 6. Traducción a reglas IF-THEN

Los atributos de test presentes en el archivo XML fueron implementados a través de un arreglo booleano de *flags* asociado a cada grupo de combate. Al iniciarse el juego, el valor de cada flag es *falso*, luego, cuando el grupo de combate entra en acción, cada flag tomará el valor *true* en el momento que se satisfagan sus condiciones asociadas. Es esperable que las características dinámicas del entorno provoquen cambios en los flags con mucha frecuencia.

5. Conclusión y Trabajos Futuros

En este trabajo se presentó el diseño de un micromanagement de las formaciones de grupo de agentes que se desenvuelven en un ambiente dinámico y que permite adaptar en forma “online” su comportamiento.

En este sentido, se diseñó y desarrolló el agente Manager Táctico, cuya toma de decisiones está implementada a través de un árbol de decisión codificado en XML y que puede ser editado y adaptado “online”, mostrando cambios en el comportamiento de las formaciones. Estos cambios se reflejan en cambios en las reglas que genera el traductor desarrollado: transforma código XML a reglas IF-THEN. La técnica de scripting fue elegida para reflejar los cambios “online” que se producen en el comportamiento de las formaciones, dado que ésta permite mostrar rápidamente la adaptación.

El micromanagement fue implementado, a partir de la arquitectura propuesta de videojuegos RTS, sobre la plataforma de StarCraft: Brood War[1].

En esta primer etapa, el árbol de decisión fue generado en forma ad-hoc. Entre nuestros trabajos futuros se encuentra la aplicación de aprendizaje automático a fin de poder generar árboles de decisión para seleccionar la formación adecuada, a partir del modelo del oponente.

Asimismo, está entre nuestro trabajo futuro, utilizar al sistema DeLP [5] como mecanismo de toma de decisión en la capa estratégica.

Referencias

1. Blizzard Entertainment. *StarCraft: Brood War*, 1998.
2. Bohemia Interactive. *Virtual Battlespace 2*, 2007.
3. Cork Institute of Technology. *The Serious Sports Project.* `http://serious-sports.org/`.
4. Chad Dawson. Formations. In Steve Rabin, editor, *AI Game Programming Wisdom*, pages 273–281. Charles River Media, 2002.
5. A. García and G. Simari. Defeasible Logic Programming: An Argumentative Approach. *Theory and Practice of Logic Programming*, 4(1):95–138, 2004.
6. Mark Hall, Eibe Frank, Geoffrey Holmes, Bernhard Pfahringer, Peter Reutemann, and Ian H. Witten. The WEKA Data Mining Software: An Update. *SIGKDD Explorations*, 11(1):10–18, 2009.
7. Karl M. Kapp. *The Gamification of Learning and Instruction: Game -based Methods and Strategies for Training and Education.* Pfeiffer, 2012.
8. M. Lanctot and M. Buro. Simulation-Based Planning in RTS Games. In Steve Rabin, editor, *AI Game Programming Wisdom 4*, pages 405–418. Charles River Media, 2008.
9. Agapito Ledezma, Ricardo Aler, Araceli Sanchis, and Daniel Borrajo. Ombo: An opponent modeling approach. *AI Communications*, 22(1):21–35, 2009.
10. I. Millington and J. Funge. *Artificial Intelligence for Games.* Morgan Kaufmann Publishers, second edition, 2009.
11. Mathew Riek and Greg Douglas. *GameMonkey Script Reference.* `http://www.gmscript.com/`.
12. Luc Sorel. *WekatextToXML: Convert Weka decision trees into interoperable XML files!* Disponible en `http://www.lucsorel.com/media/downloads/WekatextToXml.jar`.
13. Pieter Spronck. Dynamic Scripting. In Steve Rabin, editor, *AI Game Programming Wisdom 3*, pages 661–675. Charles River Media, 2006.
14. Daniel Trevisani. *Estrategias de formación de grupo en videojuegos RTS con ambientes dinámicos. Tesis de Licenciatura en Ciencias de la Computación.* Universidad Nacional del Comahue, 2013.
15. Marcel van der Heijden, S. Bakkes, and P. Spronck. Dynamic Formations in Real-Time Strategy Games. In *CIG'08 2008 IEEE Symposium on Computational Intelligence in Games*, pages 47–54, 2008.
16. Kevin Werbach and Dan Hunter. *For the Win: How Game Thinking can Revolutionize your Business.* Wharton Digital Press, 2012.
17. Michael Wooldridge, Nicholas Jennings, and David Kinny. The Gaia Methodology for Agent-Oriented Analysis and Design. *Autonomous Agents and Multi-Agent Systems*, 3(3):285–312, 2000.