Modelo formal para la fusión de Sistemas Multiagentes y Sistemas Gráficos

Resumen Se presenta un modelo formal basado en gramáticas que unifica lo mejor de los Sistemas Multiagentes y Sistemas Gráficos. Se define los diferentes elementos necesarios para implementar características esenciales de los Sistemas Multiagentes usando para su visualización las herramientas aportadas por los Sistemas Gráficos. Dicho modelo formal nos aporta varias ventajas como la separación entre la implementación de la actividad del sistema de la de los dispositivos hardware o la fácil reusabilidad de los componentes generados en otros sistemas. Para clarificar el modelo se muestra un caso práctico que implementa la quema forestal de árboles por caída fortuita de rayos.

1. Introducción

La creciente influencia de los Sistemas Multi-Agentes (SMA) en diversos campos de investigación originando una importante evolución en su desarrollo. Por otro lado, el espectacular avance de los Sistemas Gráficos (SG) ha contribuido a que la información se presente de forma más amigable, dando a conocer nuevas formas de análisis.

La definición de un agente suele consistir en un conjunto de características genéricas [4] que hacen difícil su modelado, originando algunos problemas [1],[4]. La representación visual de los SMA suele tener un carácter secundario, aunque existen excepciones [8,10,16].

En los SMA tiene una vital importancia la relación de los agentes con su entorno condicionando su comportamiento. Sería interesante usar el Motor Físico (MF) de los SG para esta relación, sobre todo si está implementado en hardware dedicado. Además, los SG tratan de modelar la realidad incorporando otros aspectos de comportamiento, en los que un SMA podría aportar una conducta inteligente.

Hoy en día existen muchos entornos de trabajo para desarrollar SMA. Se distinguen dos tipos: entornos específicos que utilizan soluciones basadas en SMA y entornos de trabajo para una arquitectura genérica.

Una de las áreas donde se está incorporando con mucho éxito los SMA es en sociología ([1,2,4,14]). Se utilizan soluciones específicas muy orientadas al estudio sociológico. Como ejemplos, se puede hacer referencia a entornos que simulan el movimiento de multitudes [11,12,15] o el movimiento de estampidas para grupos de individuos [12]. En algunos casos utilizan sistemas gráficos como el descrito en [12] o en [16] con NetLogo.

En el ámbito de los videojuegos se encuentran usos de SMA aplicados a los juegos[7]. Son programados como personajes del juego, que tiene objetivos, estrategias y realizan acciones.

Actualmente está surgiendo una gran variedad de entornos de desarrollo genérico que tienen más o menos la misma filosofía: implementación de las características más importantes de los agentes. Se van a destacar dos MASON [8] y Repast [10], por ser dos entorno que usan SG.

MASON, diseñado para un amplio tipo de aplicaciones, define un entorno de simulación basado en eventos discretos. Los agentes se dividen en dos capas: la capa del modelo que implementa el comportamiento y la capa de visualización si se desea su visualización.

Repast es un entorno para estudiar simulaciones de grupos. El sistema se basa en modeladores que definen el comportamiento de los agentes. Estos se sitúan en el espacio para luego dibujarse en 2D o 3D.

La definición de un modelo que unifique lo mejor de los SMA y los SG supondría un paso importante dentro de la evolución de estos tipos de sistemas. Este modelo especificaría todas las características necesarias para definir un agente, facilitando su visualización e interacción.

El presente artículo describe una propuesta que incorpora características de los SMA y los SG mediante un modelo basado en gramáticas.

2. Objetivos

Los objetivos pretenden conseguir un modelo que integre los SMA y SG con un lenguaje descriptivo y un sistema de eventos discretos. Para lograr esto se propone:

1. Dotar al motor gráfico de flexibilidad para cambiar la representación de los elementos sin necesidad de modificar la descripción de la escena. Debe implementar la extracción y uso de la geometría de los elementos de la escena, necesaria para que los agentes sepan cómo es el entorno que les rodea.
2. Definir un MF que se adapte a los diferentes componentes hardware si es que los hay. Podrá transmitir información del entorno a los agentes, y poder tomar decisiones considerando las limitaciones del entorno.
3. Definir un sistema de interacción para poder manipular los agentes de forma interactiva.
4. Reutilizar los diferentes agentes de forma casi inmediata en cualquier sistema definido.

3. Diseño del sistema

Un escenario se caracteriza por elementos dinámicos o agentes que realizan una actividad y elementos estáticos. Se visualizan con una secuencia de primitivas y transformaciones definidas en un sistema geométrico representado por un conjunto G.

El concepto de primitiva se debe considerar, no sólo como una primitiva de dibujo usual (dibuja una esfera), sino como una acción que se debe ejecutar en un sistema geométrico visual o no.

Una transformación es una modificación del comportamiento de las primitivas. Tienen un ámbito de aplicación y sólo se aplicará al conjunto de primitivas que se encuentren dentro de ese ámbito.

El agente es el elemento dinámico y está compuesto por actividades y un conjunto opcional de estados internos. Toda actividad consiste en un proceso que se ejecuta como reacción a un determinado evento. Puede tener una representación geométrica que depende de su estado interno definida con transformaciones y primitivas.

Un evento ejecuta una actividad y su generación es independiente del dispositivo que lo genera. Los eventos define las diferentes formas de comunicación que puede existir en un SMA (agente-agente y agente-entorno).

Estos son los principales elementos que componen el sistema. A continuación, se va a realizar su formalización en un modelo matemático que defina de forma precisa sus características.

Formalización del sistema A cada elemento que compone una escena se le puede asignar un símbolo. Estos contruyen diferentes cadenas que describen una escena mediante una sintaxis definiendo un lenguaje. Una sintaxis se presenta como una gramática [18] definida por $M = < \Sigma,\ N,\ P,\ S >$ y determina el lenguaje $L(M)$. Su definición se expresa como:

1. Sea $\Sigma = P \cup T \cup O \cup A_s^D$ el conjunto de símbolos terminales donde, P definen las primitivas, T las tranformaciones, O = { · ( ) } el conjunto de separadores y operaciones y A_S^D es el conjunto que representa agentes que ejecutan actividades cuando reciban los eventos definidos en D, donde D son todos los tipos de eventos generados por el sistema y S el conjunto de los estados internos. Así, el agente a_s^d, que está en el estado s, ejecutará la actividad d cuando se produzca un evento e^d.
2. Sea N = {MUNDO, VARIOSOBJETOS, OBJETO, ACTOR, TRANSFORMACIÓN, FIGURA} el conjunto de símbolos no terminales.
3. Sea S = {MUNDO} el símbolo inicial de la gramática.
4. Se definen las reglas gramaticales R como:

Regla 1.- **MUNDO** :- VARIOSOBJETOS
Regla 2.- **VARIOSOBJETOS** :- OBJETO | OBJETO · VARIOSOBJETOS
Regla 3.- **OBJETO** :- FIGURA | TRANSFORMACIÓN | ACTOR
Regla 4.- **ACTOR** :- a_s^d(VARIOSOBJETOS), $a_s^d \in A_S^D, d \in D, s \in S$
Regla 5.- **TRANSFORMACION** :- t(VARIOSOBJETOS), $t \in T$
Regla 6.- **FIGURA** :- p, $p \in P$

A continuación se va a asignar a cada regla una función matemática según el método denotacional para describir semánticas.

Función semántica para FIGURA y TRANSFORMACIÓN (Regla 5 y 6) La regla 6 define la sintaxis de una figura como una secuencia de primitivas.

Cada primitiva es definida mediante la función α definida como $\alpha : P \rightarrow G$. Es decir, un símbolo de P ejecutará una primitiva sobre el sistema geométrico G. Esto significa que dependiendo de la función α y del sistema geométrico G el resultado puede variar. Por ejemplo, G ejecuta primitivas de motor gráfico (OpenGL o Direct3D). Otros ejemplos son: cálculo de colisiones mediante el cálculo de limites, la ejecución de un sonido, el movimiento de un robot, la transmisión de la escena por una red, etc.

Los ejemplos anteriores ilustran cómo la función α aporta la abstracción necesaria para homogeneizar las diferentes implementaciones de un SG o proceso geométrico. Con una cadena descriptiva se tienen diferentes formas de extraer información que representa cada uno de los elementos que hay en una escena. Esto es importante para relacionar los agentes con el MF. Los procesos de extracción de información del entorno se ejecutan en módulos geométricos que componen el MF. La información procesada se envía a los agentes para realizar su actividad.

La regla 5 define la sintaxis de una transformación. Esta transformación tiene un ámbito de aplicación acotado por los símbolos '()'. Se definen dos funciones para describir la semántica de una transformación, estas son: $\beta : T \rightarrow G$. que se ejecuta al procesar el símbolo '(' y $\delta : T \rightarrow G$ que se ejecuta con el símbolo ')'. Estas dos funciones tienen las mismas características que la función α y se definen en el mismo sistema geométrico G.

Con P, T y las funciones α, β, δ, se define una función que dada una cadena w con símbolos de T y P, ejecute la secuencia de primitivas y transformaciones en el sistema geométrico G. Esta función es φ denominada función de visualización de la escena en un conjunto G y se define como:

$$\varphi(w) = \left\{ \begin{array}{lll} \alpha(w) & Si & w \in P \\ \beta(t); \varphi(v); \delta(t) & Si & w = t(v) \wedge v \in L(M) \\ \varphi(s); \varphi(t) & Si & w = s \cdot t \wedge s, t \in L(M) \end{array} \right\} \quad (1)$$

La función no depende de G, sólo depende de α, β, δ. Los detalles de implementación que hay entre un sistema de visualización, de cálculo geométrico o uno que transporte escenas por una red están encapsuladas en α, β, δ.

Función semántica para AGENTE (Regla 4) La regla 4 referencia a los agentes, la parte dinámica del sistema. La semántica del agente se describe como una función denomina función evolutiva λ y se define como: $\lambda : L(M) \times E^D \rightarrow L(M)$ donde E^D es el conjunto de eventos para los dispositivos D (considerando estos dispositivos como cualquier proceso que emite eventos). Al aplicar la función $\lambda(w, e^f)$, w se transforma en otra cadena u. La función λ tendrá diferente expresión dependiendo de su evolución. Sin embargo, se puede definir la expresión general como:

$$\lambda(a^d(v), e^f) = \left\{ \begin{array}{lll} u \in L(M) & Si & f = d \\ a^d_s(v) & Si & f \neq d \end{array} \right\} \quad (2)$$

La cadena resultante puede contener al propio agente, puede generar otro evento para la siguiente etapa o cambiar el estado 's' del agente. Los nuevos eventos generados por el agente se acumulan para la siguiente etapa. Así, se modela la comunicación entre agentes tan importante en SMA. En el caso de que el evento no coincida con el evento activador entonces la función devuelve el mismo agente.

Con la función λ se puede definir un algoritmo recursivo que dado un conjunto de eventos y una cadena, describa la evolución del sistema. A esta función se la denomina función de evolución del sistema η. Para su definición se necesita un conjunto de eventos $e^i, e^j, e^k \cdots$ que abreviado se denota como e^v, siendo $v \in D^+$. Esta función se define como:

$$\eta(w, e^v) = \left\{ \begin{array}{ll} w & Si \quad w \in P \\ t(\eta(v, e^v)) & Si \quad w = t(v) \\ \prod\limits_{\forall f \in v} (\lambda(a_s^d(\eta(y, e^v)), e^f)) & Si \quad w = a_s^d(y) \\ \eta(s, e^v) \cdot \eta(t, e^v) & Si \quad w = s \cdot t \end{array} \right\} \tag{3}$$

El operador $\prod\limits_{\forall f \in v} (\lambda(a_s^d(\eta(y, e^v)), e^f))$ realiza la concatenación de las cadenas generadas por λ. Para cadenas que sólo son transformaciones y primitivas, el sistema se queda como está, formando sólo parte del entorno.

Para la visualización de un agente se necesita primero transformar los agentes a cadenas de primitivas y transformaciones. Este proceso se realiza con un tipo especial de funciones λ. A estas funciones se las denominaran funciones de visualización y su definición es: $\theta : L(M) \times E^V \rightarrow L(E)$ donde E^V son los eventos que se generan en la visualización, y L(E) es el lenguaje L(M) sin agentes. La expresión de la función de visualización se define como:

$$\theta(a^d(v), e^f) = \left\{ \begin{array}{ll} u \in L(E) & Si \quad f = d \wedge d \in V \\ \epsilon & Si \quad f \neq d \end{array} \right\} \tag{4}$$

Se puede observar que existen pequeñas diferencias entre λ y θ. La primera es que θ devuelve cadenas que pertenecen a L(E). La segunda es que si no coincide con el evento se devuelve una cadena vacía no teniendo representación para eso evento.

El tipo de evento V es importante y no se corresponde con ningún dispositivo de entrada en concreto, si no que es un evento que se encarga de establecer los diferentes tipos de vista que tiene el sistema o procesos geométricos.

Al igual que con λ, se define un algoritmo para θ que devuelve una cadena $z \in L(E)$, dada una cadena $w \in L(M)$ y un conjunto de eventos e^v con $v \in V^+$, y $V \subseteq D$. A esta función se le denomina función de visualización π ($\pi : L(M) \times E^V \rightarrow L(E)$) y se define como:

$$\pi(w, e^v) = \left\{ \begin{array}{ll} w & Si \quad w \in P^+ \\ t(\pi(y, e^v)) & if \quad w = t(y) \\ \prod\limits_{\forall f \in v} (\theta(a^v(\pi(y, e^v)), e^f)) & Si \quad w = a^v(y) \\ \pi(s, e^v) \cdot \pi(t, e^v) & Si \quad w = s \cdot t \end{array} \right\} \tag{5}$$

Función semántica para OBJETO, VARIOSOBJETOS y MUNDO (Regla 1,2,3) Las funciones semánticas de estas reglas descomponen las cadenas en subcadenas y dependiendo de si es un agente, una transformación o una primitiva ejecutan las funciones antes especificadas. Se debe observar que las primitivas que no son generadas por los agentes con la función de visualización representan la parte estática del sistema, o lo que es lo mismo, el entorno donde se desarrolla la actividad del sistema.

3.1. Actividad y eventos

En los SMA se deben establecer mecanismos para que la actividad dentro del sistema pueda ser modelada. Esta actividad se realiza en los agentes y se activan con eventos.

Se define e_c^d como el evento de tipo $d \in D$ que se va a ejecutar bajo la condición c y tiene como datos e.

Se define e^d como el evento de tipo $d \in D$ y tiene como datos e.

Se puede observar que el origen de la creación de los eventos no es importante y sí el tipo de evento y sus datos. Esto no quiere decir que en el evento no se pueda incluir datos que identifiquen quién envía el mensaje. Por ejemplo, en la comunicación entre dos agentes los eventos pueden contener información sobre quién envió el mensaje. De esta manera se establece un sistema genérico de comunicación que puede implementar KMQL, FIPA [3] o cualquier otro.

Dispositivos de entrada y generadores de eventos Se desea establecer una independencia entre el sistema y los dispositivos que generan eventos (hardware o software). Para ello, se especifica una función que genera los eventos que se necesitan para hacer reaccionar al sistema. Esta función define un mecanismo para dar información del entorno a los agentes. Este elemento es el generador de eventos y su definición es:

Se define como $C^d(t)$ al generador que crea los eventos del tipo d en el instante t, siendo $d \in D$ donde D es el conjunto de todos tipos de eventos que pueden ser generados por el sistema.

Se debe destacar que los eventos se generan en un instante t por motivos de sincronización. El generador es la parte dependiente del dispositivo de entrada encapsulada en una única función, asilando esta dependencia del sistema.

Para que no se produzcan ambigüedades se va a establecer un orden de prioridad entre los generadores, de tal manera que, dados dos generadores C_i y C_j con $i < j$, si crean dos eventos del mismo tipo entonces prevalecerá el evento creado por C_i sobre el C_j.

El proceso para la obtención de los eventos producidos por todos los dispositivos de entrada y sus generadores asociados se define como:

Si C^* es el conjunto de todos los generadores de eventos del sistema y la función $E(C^*, t)$ recolecta todos los eventos de todos los generadores entonces:

$$E(C^*,t) = \left\{ \begin{array}{ll} e(z, C_i(t)) & Si\ z = E(C^* - C_i, t) \\ \epsilon & Si\ C^* = \emptyset \end{array} \right\} \ con\ e(z, e^i) = \left\{ \begin{array}{ll} z \cdot e^i & Si\ e^i \notin z \\ z & Si\ e^i \in z \end{array} \right\}$$

Tanto con los generadores de eventos como con los eventos que genera los agentes, se puede modelar todo lo relacionado con los procesos de comunicación que existen en los SMA (tanto entre dos agentes como entre agente y entorno).

3.2. Algoritmo del sistema

Una vez definidos todos los elementos implicados en el modelo que gestionará un SMA, se puede establecer el algoritmo que ejecuta todo el sistema:

1. $w = w_o$; $t = 0$
2. $e^* = E(G^*, t)$
3. $e^v =$ eventos de e^* donde $v \in V^+$
4. $e^u = e^* - e^v$
5. $w_{next} = \eta(w, e^u)$
6. $v = \pi(w, e^v)$
7. $g = \varphi(v)$
8. $w = w_{next}$; $t = t + 1$
9. Si $w = \epsilon$ entonces Ir a 12
10. Ir a 3
11. Fin

w_o Cadena inicial del sistema.
e^* Eventos generados en el frame t.
$C^* = \{$Todos los generadores de eventos$\}$
$D = \{$Todos los eventos del sistema$\}$
$V = \{$Todos los eventos visuales$\}$ $V \subseteq D$
e^v Eventos visuales generados.
e^u Eventos no visuales generados.
g dispositivo de salida gráfica.

En primer lugar se inicializa el sistema (w_o y $t = 0$). Los pasos 2, 3 y 4 gestionan los eventos del sistema. En el paso 5 se llama a la función de evolución para calcular el siguiente frame. En los pasos 6 y 7 se visualiza el frame. En 8 se prepara para la siguiente iteración y el paso 9 se mira si se ha llegado al final (cadena vacía, resultado de un evento especial) si no es así se pasa a 3.

Se puede destacar que al no compartir datos los pasos 6 y 7-8 pueden ser facilmente paralelizable optimizando el algoritmo de forma significativa.

4. Caso Práctico

Este ejemplo es una aplicación que simula incendios forestales causados por la caída aleatoria de rayos. Se busca la mejor distribución de árboles para reducir el número de árboles quemados [6]. El sistema basicamente consiste en un agente que define el bosque. Este agente crea otros agentes que definen árboles y relampagos. Dada una posición (i, j) de un tablero 2D, representado por el bosque, se puede situar un árbol con una probabilidad g o un rayo con una probabilidad f. En el caso de que caiga un rayo sobre un árbol este arderá y todos los que esten a su alrededor provocando una reacción en cadena. El ejemplo tiene que definir 4 elementos principales: eventos, generadores de eventos, agentes y primitivas gráficas.

Los eventos son los causantes de la actividad del sistema. Los evento definidos para este ejemplo son:

- t: Se genera cada cierto tiempo t.
- c: Crea un árbol en la posición (i, j).
- f: Crea un rayo en la posición (i, j).
- e: Elimina el árbol en la posición dada (i, j).
- b: Quema un árbol en la posición (i, j).
- v: Dibuja la escena usando una librería gráfica (en este caso OpenGL).

El siguiente paso se debe definir los generadores de eventos.

- *Generador de eventos de tiempo* (C_{time}): Es responsable de las animaciones del sistema. Cada cierto tiempo t, se genera un evento e^t.
- *Generador de elementos del bosque* (C_{forest}): Genera eventos para crear árboles (e^c) con provabilidad g y relámpagos (e^f) con probabilidad f.
- *Generador de visualización* ($C_{visualization}$): Captura las órdenes de dibujado y produce un evento para dibujar la escena en el sistema gráfico.

El cuadro 1 muestra las primitivas y las transformaciones para dibujar la escena. Las funciones α, β y δ definen el comportamiento de ambas. En este caso implementan el dibujado en la librería gráfica OpenGL.

El cuadro 2 define los agentes mediante su función de evolución λ y su representación gráfica por su función π.

El agente que define un árbol (TR) tiene tres estados internos $st = \{s1, s2, s3\}$ donde $s1$ corresponde al estado de creciendo, $s2$ a crecido y $s3$ a en llamas. Esto es un ejemplo de como un agente puede terner diferente representación según su estado interno. La llamada de la función Nbo envía eventos de quemado e^b a todos sus vecinos provocando la reacción en cadena (comunicación agente-agente). Un ejemplo de comunicación entorno-agente es entre el bosque y el generador C_{forest} que cambia el entorno.

S representa el factor de escala en el crecimiento de un árbol y depende del estado actual del agente. El valor es definido por la expresión $s = t/N$, donde N es el total del número de frames. En el caso de TR_b el factor de escala es decrementado s^{-1}. En general, todas las animaciones depende de t que modifica el estado actual del agente y su representación.

Por último, la cadena inicial se define como: $w_0 = B^{cfe}$.

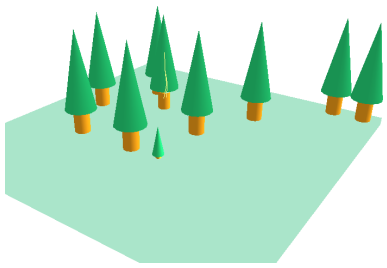

Figura1. Vista del bosque

Primitivas	*Descripción*
TR	Dibuja un árbol
TR_b	Dibuja un árbol quemándose
FA	Dibuja un rayo
BO	Dibuja un tablero de NxN

Transformaciones	*Descripción*
$D_{i,j}$	Desplazar (i,j)
S_s	Escalar (s)

Cuadro1. (Izquierda) Primitivas, (Derecha) Transformaciones

Agente	*Descripción*	*Función* λ *y* π
BO	Bosque	$\lambda(BO^{cfe}, e^i) = \begin{Bmatrix} TR_{s1}^{t=1} \cdot BO^{cfv} & i = c \\ FA^{t=1} \cdot BO^{cfv} & i = f \\ BO^{cfe} & i = e \\ BO^{cfe} & i \neq c, f, e \end{Bmatrix}$ $\pi(BO^v, e^i) = \begin{Bmatrix} BO & i = v \\ \epsilon & i \neq v \end{Bmatrix}$
TR	Árbol	$\lambda(TR_{st}^x, e^i) = \begin{Bmatrix} TR_{s1}^{t+1} & i = t \wedge t+1 \leqslant N \wedge s = s1 \\ TR_{s2}^{b} & i = t \wedge t+1 > N \wedge s = s1 \\ TR_{s3}^{t=1} \wedge \Delta Nbo^b & i = b > N \wedge s = s2 \\ TR_{s3}^{t+1} & i = t \wedge s = s3 \\ \Delta e^e & i = t \wedge t+1 > N \wedge s = s3 \\ TR_{st}^{t} & i \neq t \wedge i \neq b \end{Bmatrix}$ $\pi(TR_{st}^v, e^i) = \begin{Bmatrix} D_{(i,j)}(S_s(TR)) & i = v \wedge st = s1 \\ D_{(i,j)}(TR) & i = v \wedge st = s2 \\ D_{(i,j)}(S_{s^{-1}}(TR)) & i = v \wedge st = s3 \\ \epsilon & i \neq v \end{Bmatrix}$
FA	Rayo	$\lambda(FA^t, e^i) = \begin{Bmatrix} FA^{t+1} & i = t \wedge t+1 \leqslant N \\ \Delta e^b & i = t \wedge t+1 > N \\ FA^t & i \neq t \end{Bmatrix}$ $\pi(FA^v, e^i) = \begin{Bmatrix} D_{(i,j)}(FA) & i = v \\ \epsilon & i \neq v \end{Bmatrix}$

Cuadro2. Definición de agentes

5. Conclusiones y trabajos futuros

Se ha diseñado un sistema de comunicación entre los procesos geométricos que definen un MF y los agentes que componen una escena mediante el generador de eventos.

El uso de un lenguaje para la descripción de los elementos de la escena y la independencia del sistema gráfico hace que estas cadenas descriptivas puedan ser reutilizadas.

Este modelo abre diferentes expectativas que podrán ser exploradas en el futuro. Por ejemplo, se podrían estudiar el desarrollo de algoritmos de modificación genética. Dado un agente como r^{abc}, se podría definir una función de evolución que dado un evento pueda producir la reproducción de r pero con modificaciones en la cadena de eventos. Es decir, se podría reproducir como la cadena r^{ab} o incluso incluir otros eventos como r^{abd}.

Una posible área de aplicación del modelo podría ser en robótica. Un robot puede comportarse como un agente o un conjunto de ellos. Los diferentes sensores pueden definir generadores de eventos que tratarían la señal y pasar a los agentes la información procesada.

En definitiva, se ha pretendido dar una solución que integra lo mejor de los SMA y SG y cuya aplicación puede aplicarse en toda la gran variedad de problemas.

Referencias

1. Robert Axelrod Advancing the Art of Simulation in the Social Sciences *Simulating Social Phenomena*, Springer, 1997
2. Alexis Drogoul, Jacques Ferber, Christophe Cambier Multi-agent Simulation as a Tool for Analysing Emergent *Processes in Societies* 1992
3. Michael R. Genesereth, Steven P. Ketchpel Software Agents *Communications of the ACM* 1995
4. Nigel Gilbert Agent-Based Models *SAGE Publications* 2008
5. Jonathan Gratch, Jeff Rickel, Elisabeth Andr, Justine Cassell, Eric Petajan, Norman I. Badler Creating Interactive Virtual Humans:Some Assembly Required. *IEEE Intelligent Systems* July-August 2002
6. John H.Miller Complex Adaptative Systems *Princeton University Press* 2007
7. Aaron Khoo, Robert Zubek Applying Inexpensive AI Techniques to Computer Games *IEEE Intelligent Systems* July-August 2002
8. Sean Luke, Claudio Cioffi-Revilla, Liviu Panait, and Keith Sullivan MASON: A New Multi-Agent Simulation Toolkit *Simulation Vol. 81, SAGE Journals* 2005
9. Pattie Maes, Trevor Darrell, Bruce Blumberg, Alex Pentland The ALIVE System:Wireless, Full-body Interaction with Autonomous Agents, *ACM Multimedia Systems, Vol.5, No.2, pp.105-112* 1997
10. M.J. North, T.R. Howe, N.T. Collier, J.R. Vos The Repast Simphony Runtime System *Conference on Generative Social Processes, Models, and Mechanisms, Proceedings of the Agent* 2005
11. Reza Olfati-Saber Flocking for Multi-Agent Dynamic Systems:Algorithms and Theory *IEEE Transactions on Automatic Control* 2004
12. Craig Reynolds Interaction with Groups of Autonomous Characters *Game Developers Conference* 2000
13. Theresa-Marie Rhyne Computer GamesInfluence on Scientific and Information Visualization *Entertainment Computing* 2000
14. R. Keith Sawyer Social Emergence: Societies As Complex Systems *Cambridge University Press* 2005
15. Branislav Ulicny, Daniel Thalmann Crowd simulation for interactive virtual environments and VR training systems *Computer Animation and Simulation, Springer* 2001
16. Wilensky, U., NetLogo User Manual, 1999
17. Ipke Wachsmuth, Yong Cao Interactive Graphics Design with Situated Agents *Graphics and Robotics* 1995
18. Davis Martin D.,Sigal R.,Weyuker E. J.: *Computability, Complexity, and Languages, Fundamentals of Theoretical Computer Science*, 2nd ed. San Diego: Elsevier Science, 1994.