

Estudio de nuestras ciudades a partir del modelado y simulación computacional para el desarrollo de su capa tecnológica

Dr. Mario Siller

M.C. Diego Orozco

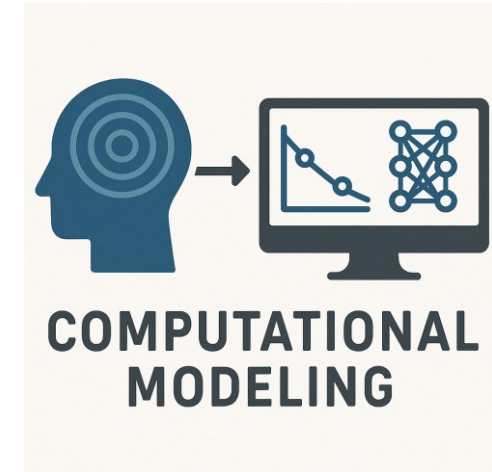
M.C. Giovana Pérez

Cinvestav Unidad Guadalajara

P2 – Introducción al Modelado Basado en Agentes

Modelado Basado en Agentes (MBA)

- Un modelo es una representación abstracta de fenómenos, procesos o *sistemas* del mundo real, en donde se *seleccionan los aspectos particulares* para *describir, explicar, analizar o predecir* su comportamiento.
- Los modelos basados en agentes (MBA) permiten *modelar la estructura de un sistema complejo* y simular su evolución dinámica a lo largo del tiempo.



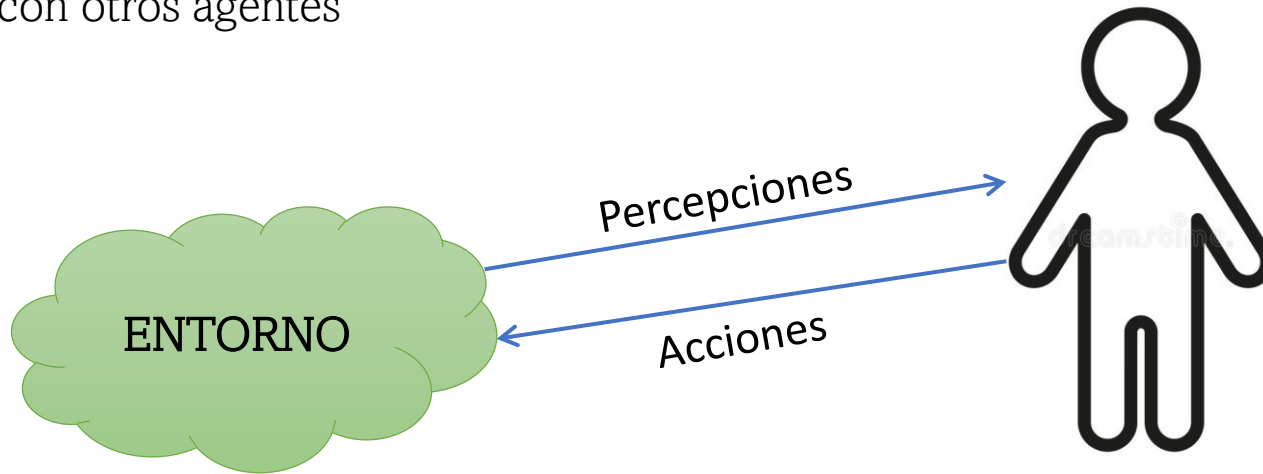
Modelado Basado en Agentes (MBA)

- Los agentes tienen **comportamientos**, a menudo **descritos por reglas simples**, e interacciones con otros agentes, que a su vez influyen en sus comportamientos.
- Cada agente es una **entidad autónoma** con distintas **metas y acciones** dentro de un contexto particular
- Al **modelar a los agentes individualmente**, se pueden observar todos los efectos de la diversidad que existe entre los agentes en sus **atributos y comportamientos**, ya que **da lugar al comportamiento del sistema en su conjunto**.



Modelado Basado en Agentes (MBA)

Un **agente** puede ser definido como un **sistema computacional** que es capaz de exhibir **comportamiento autónomo** con capacidad de **percibir y afectar el entorno**, así como **socializar** con otros agentes

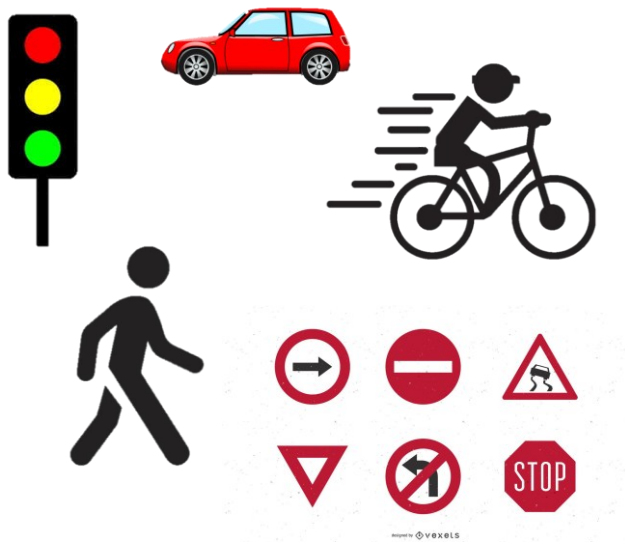


Características:

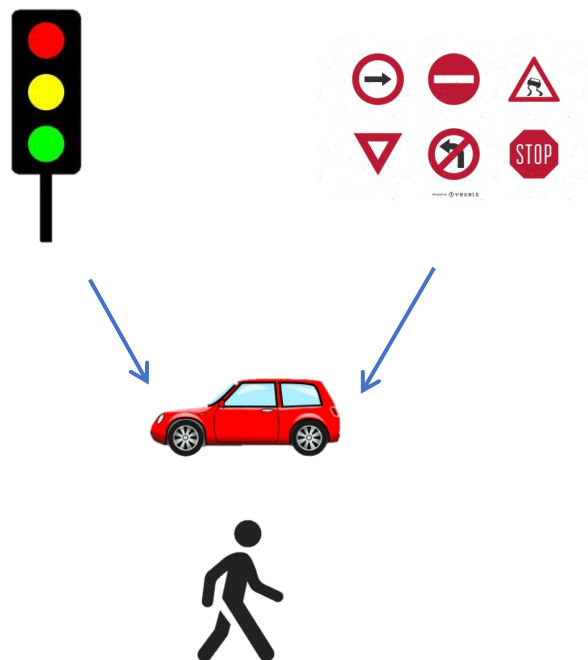
- Autonomía
- Proactividad
- Habilidad Social
- Reactividad
- Persistencia
- Razonamiento
- Productividad
- Movilidad
- Personalidad

Estructura

Conjunto de agentes



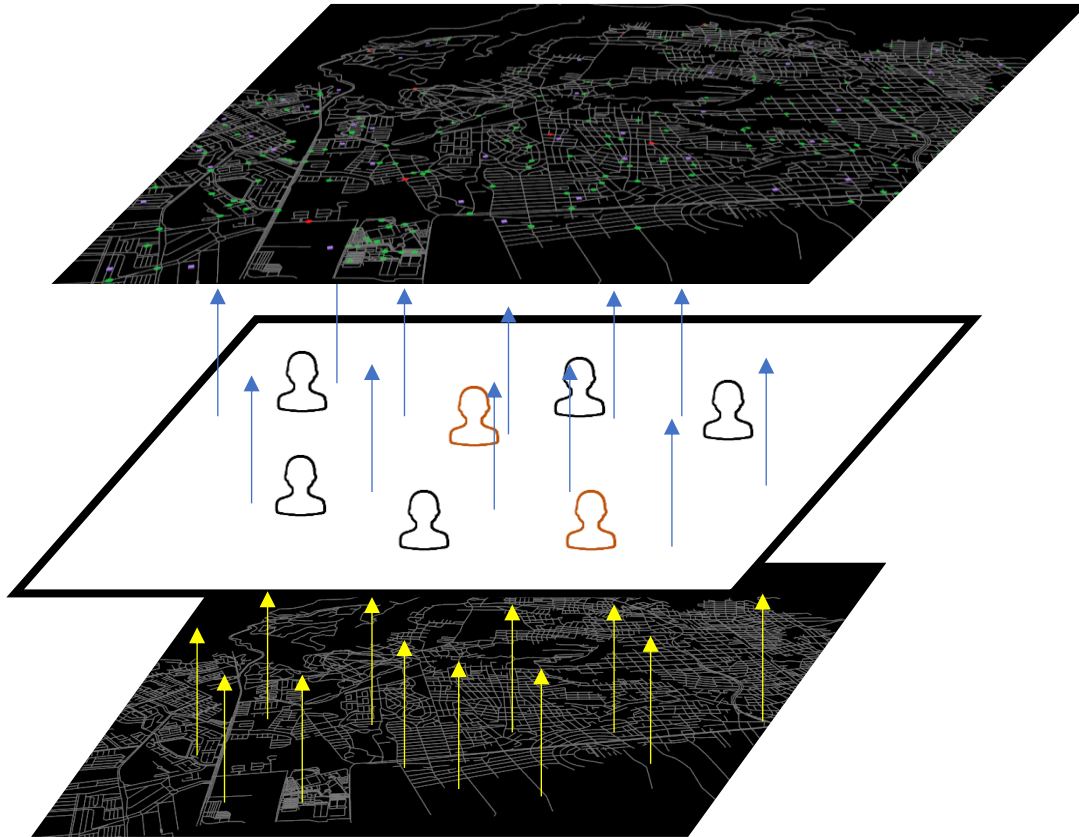
Relaciones e interacciones



Ambiente



Modelado y Simulación Computacional



3) Ejecución de la simulación usando las reglas de comportamiento para los agentes en el escenario determinado.

2) Los agentes se ubican e inician

1) Creación de un escenario y carga de datos

- Se especifica el entorno
- Uso de archivos SIG



Aplicaciones

Las aplicaciones del modelado basado en agentes abarcan una amplia gama de áreas y disciplinas.

Ejemplos:

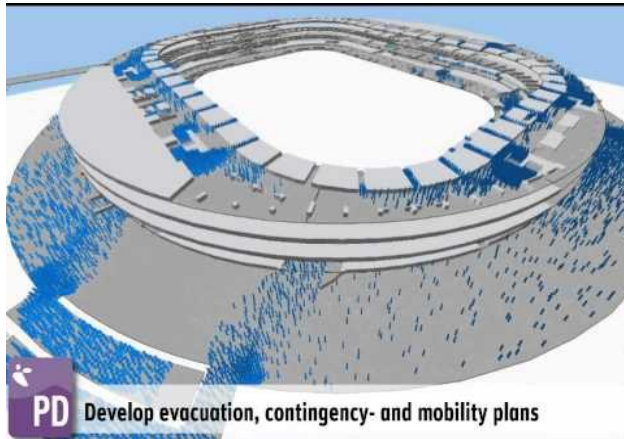
Modelo del
comportamiento de los
agentes en el mercado de
valores
(Arthur et al, 1997)

Cadenas de suministro
(Macal, 2004)

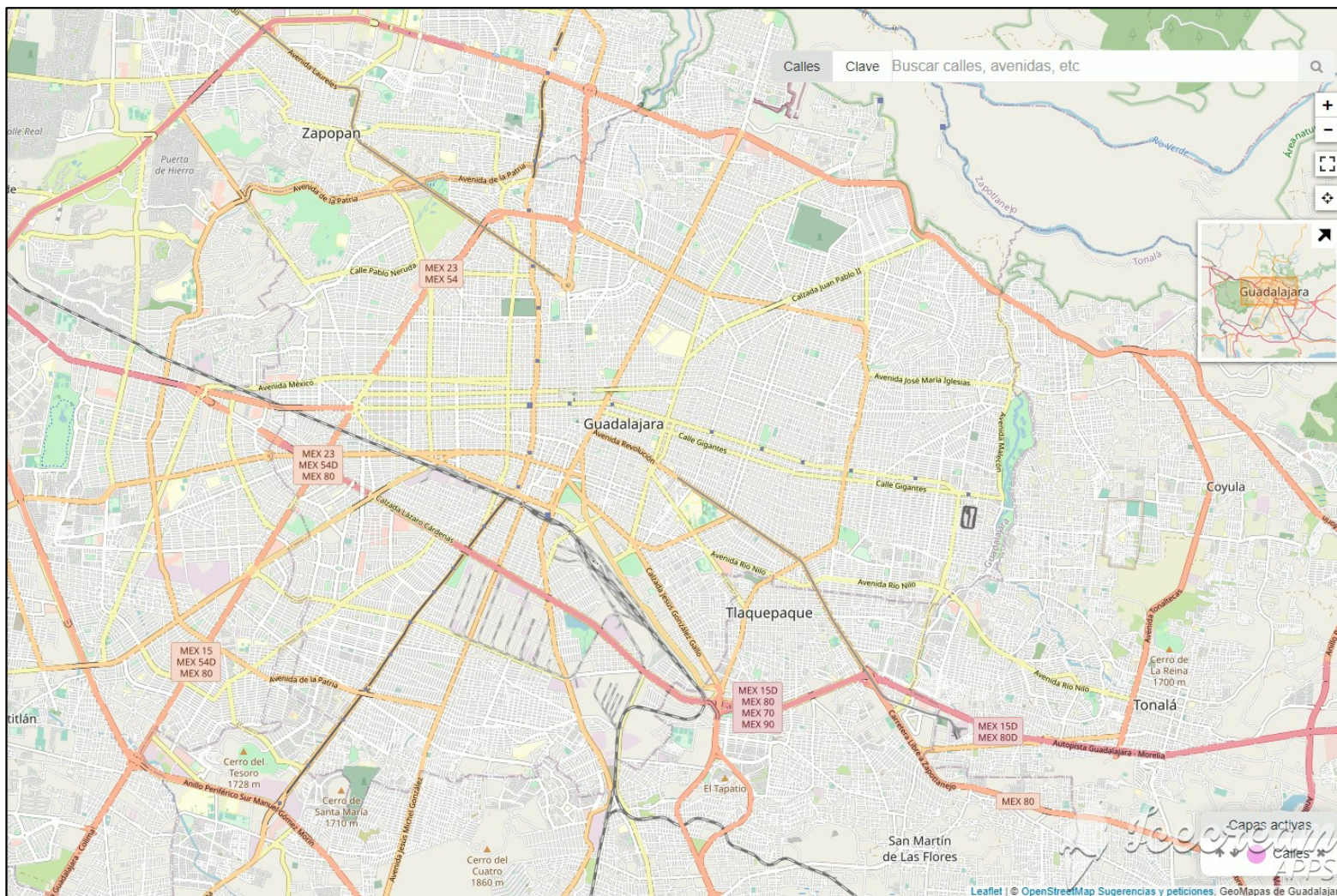
Predecir la propagación
de epidemias
(Bagni et al, 2002)

Comprender el
comportamiento de
compra del consumidor
(North et al, 2009)

Ejemplos de MBA



La ciudad como un **sistema complejo**

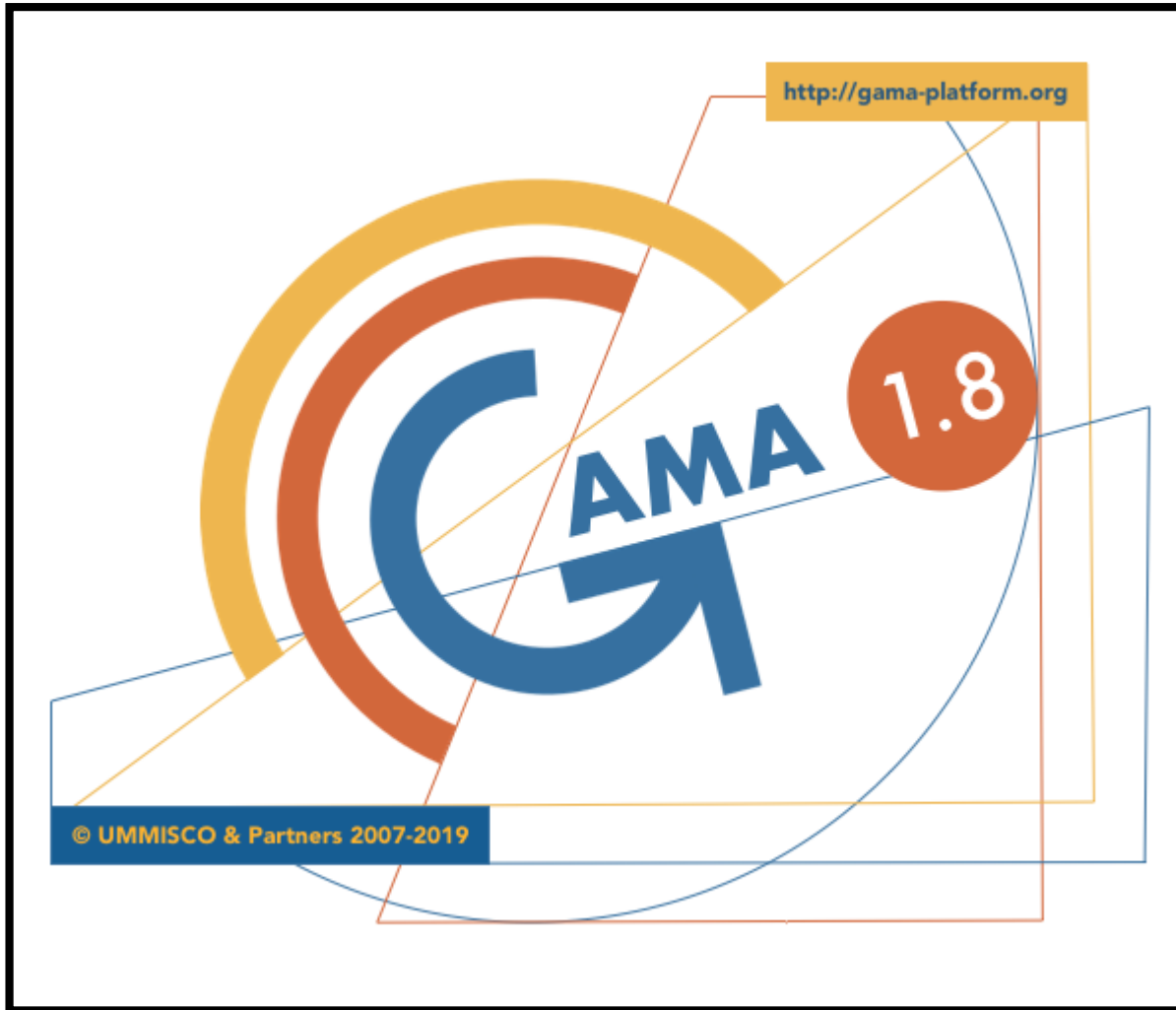


Modelado Basado en Agentes (MBA)

- El **modelado basado en agentes** ofrece **una forma de modelar sistemas sociales** que están compuestos por agentes que interactúan e influyen entre sí, aprenden de sus experiencias y adaptan sus comportamientos para que se adapten mejor a su entorno.



Capa Tecnológica de la ciudad



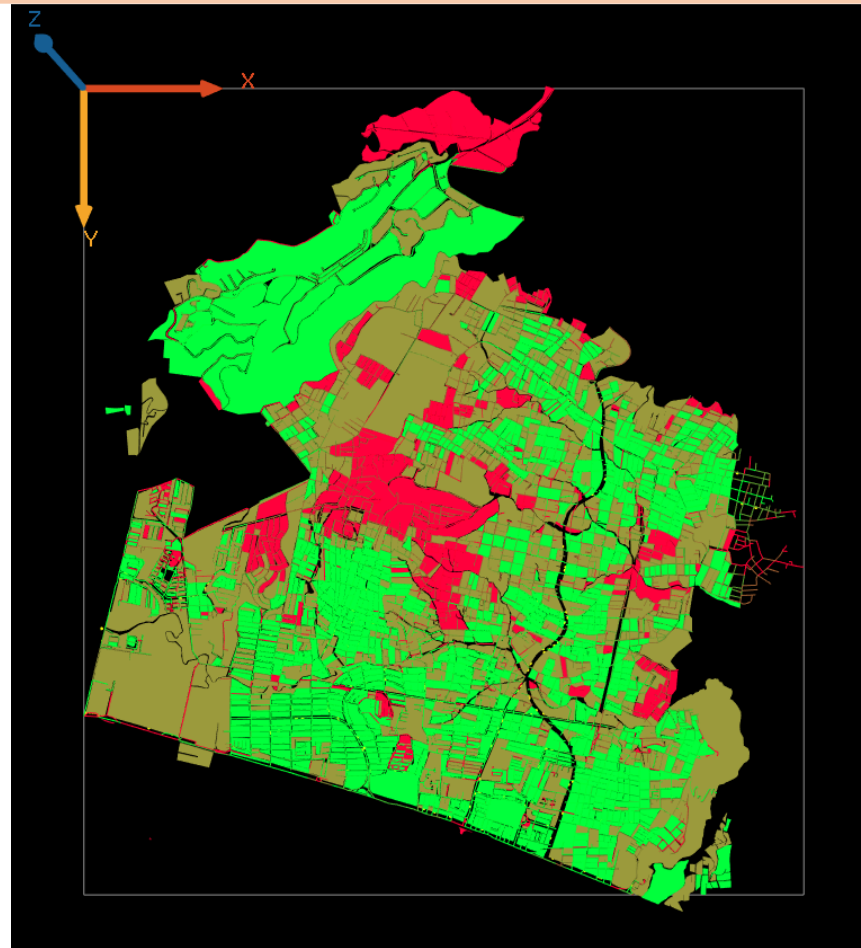
GAMA es un entorno de desarrollo de modelado y simulación para construir simulaciones basadas en agentes.

Características:

- Apta para múltiples dominios de aplicación
- Lenguaje basado en agentes
- Extensa biblioteca de funciones nativas (funciones matemáticas, graficas, métodos para agentes, etc.)
- Modelos guiados por datos y GIS
- Código abierto

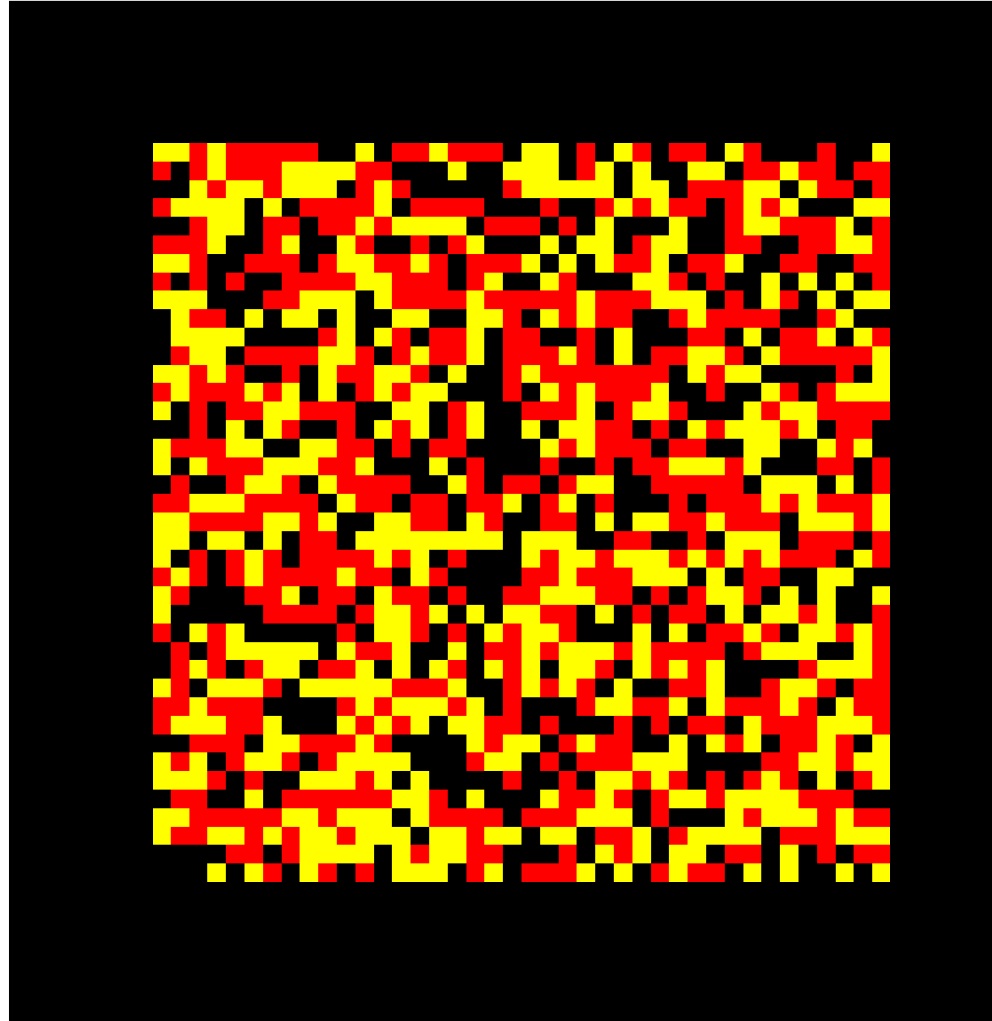
<https://gama-platform.github.io>

Posibilidad de realizar intervenciones en la simulación y analizar el resultado de las mismas



Percepción de Inseguridad en Lomas del Centinela
utilizando como valor preponderante la iluminación
artificial y datos de INEGI.

Simulación en GAMA PLATFORM



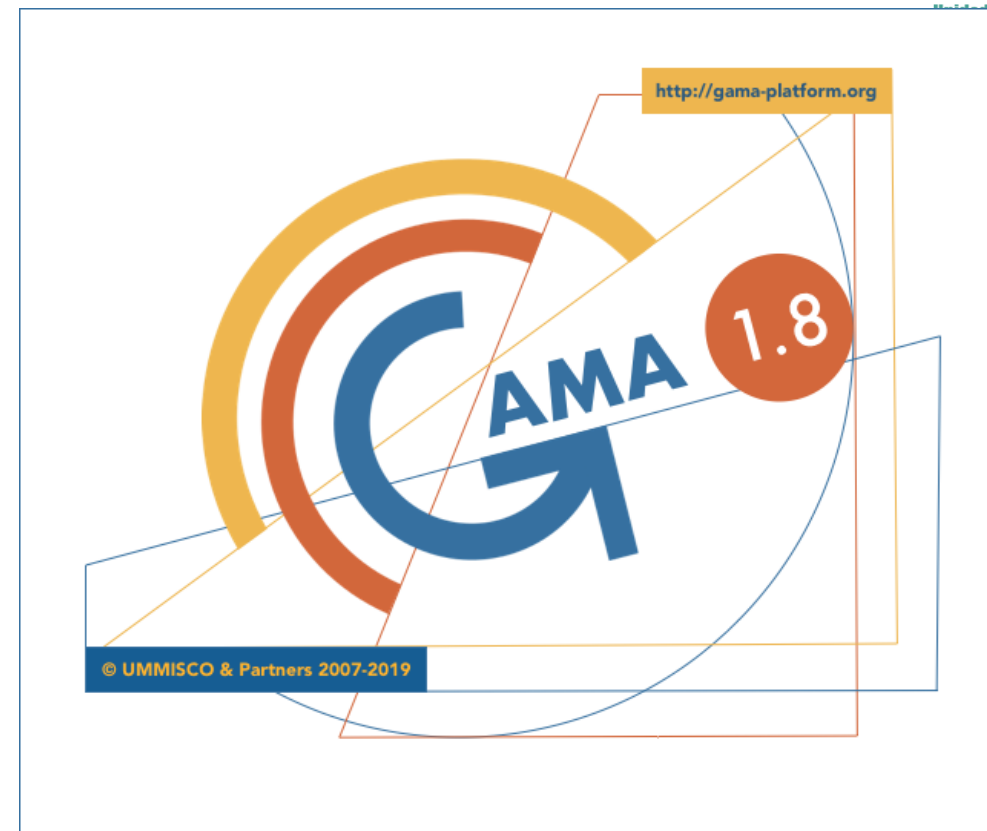
MBA de segregación basado en Schelling,
ejecución en GAMA ("*Segregation*").

GAMA Platform

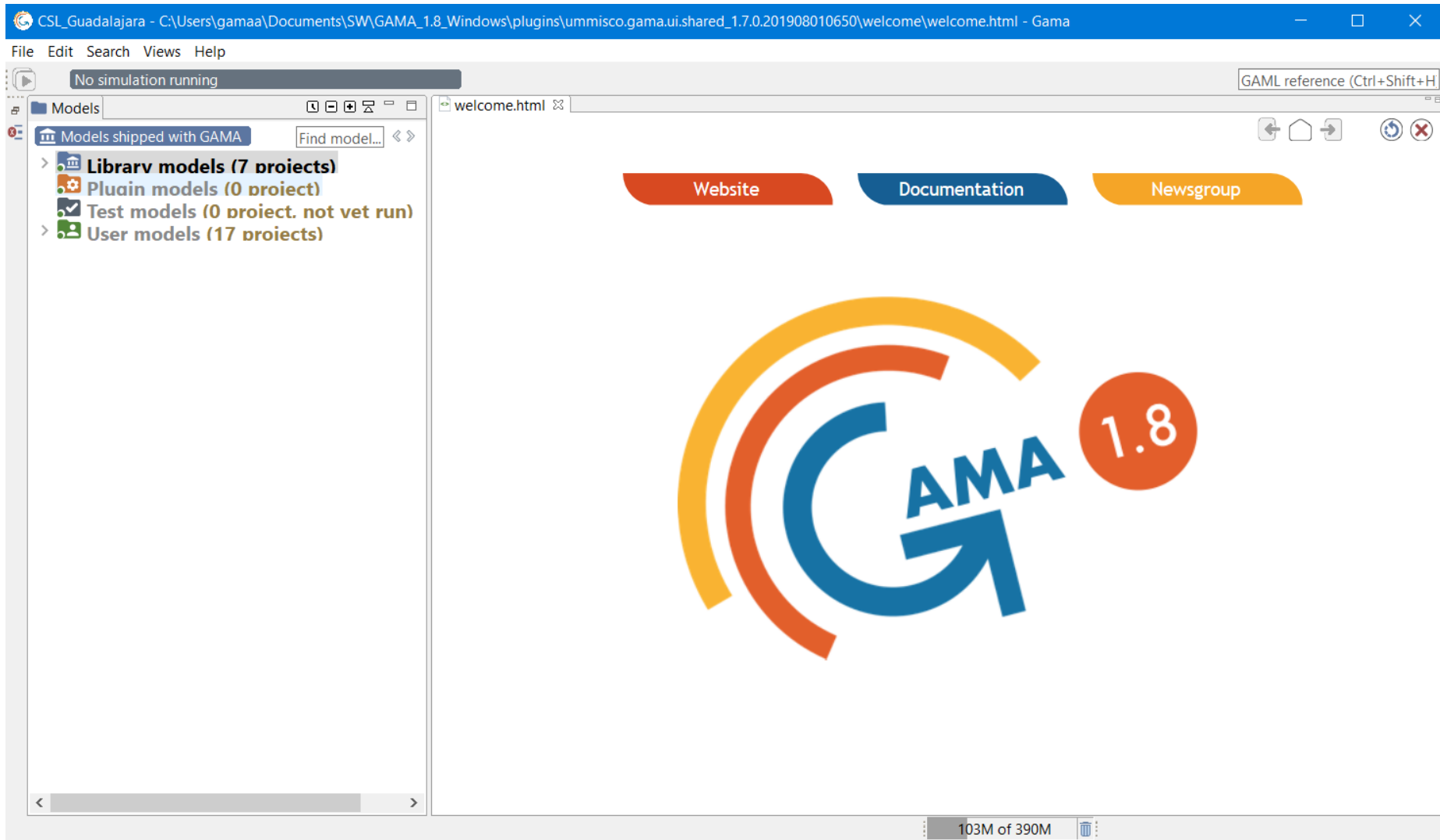
Es una plataforma que provee un entorno de desarrollo de modelos y simulaciones a expertos en diversas áreas, modeladores, e investigadores.

Entre sus características podemos resaltar las siguientes:

- Múltiples dominios de aplicación.
- Lenguaje de agentes, GAML, de alto nivel e intuitivo.
- Una biblioteca extensa de primitivas (funciones matemáticas, gráficas, movimiento de agentes, etc).
- Modelos guiados por datos y GIS.



Página de bienvenida



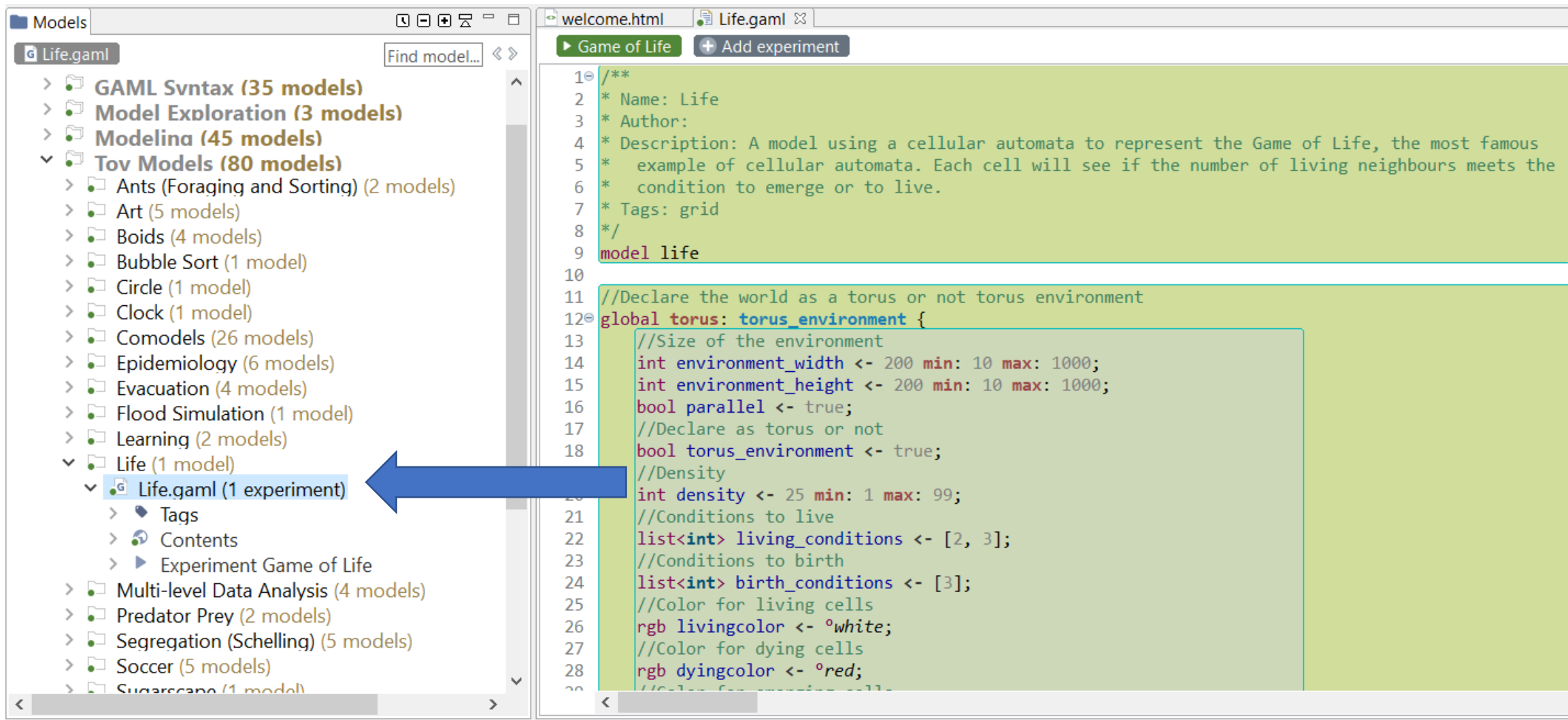
Navegar a través del espacio de trabajo

Todos los modelos que se editen o corran en GAMA están accesibles desde el **Navegador**, que siempre está a la izquierda de la pantalla principal. Tenemos cuatro categorías principales: *Models library*, *Plugin models*, *Test models*, y *User models*.



Inspeccionar modelos

Cada modelo es presentado como un nodo en el navegador del espacio de trabajo.



The screenshot displays the NetLogo workspace interface. On the left, the 'Models' sidebar shows a hierarchical tree of models. The 'Life.gaml' model is selected, and a blue arrow points to it. The main editor on the right shows the code for the 'Life.gaml' model, which is a Game of Life simulation. The code includes comments and declarations for the environment, density, and colors.

```

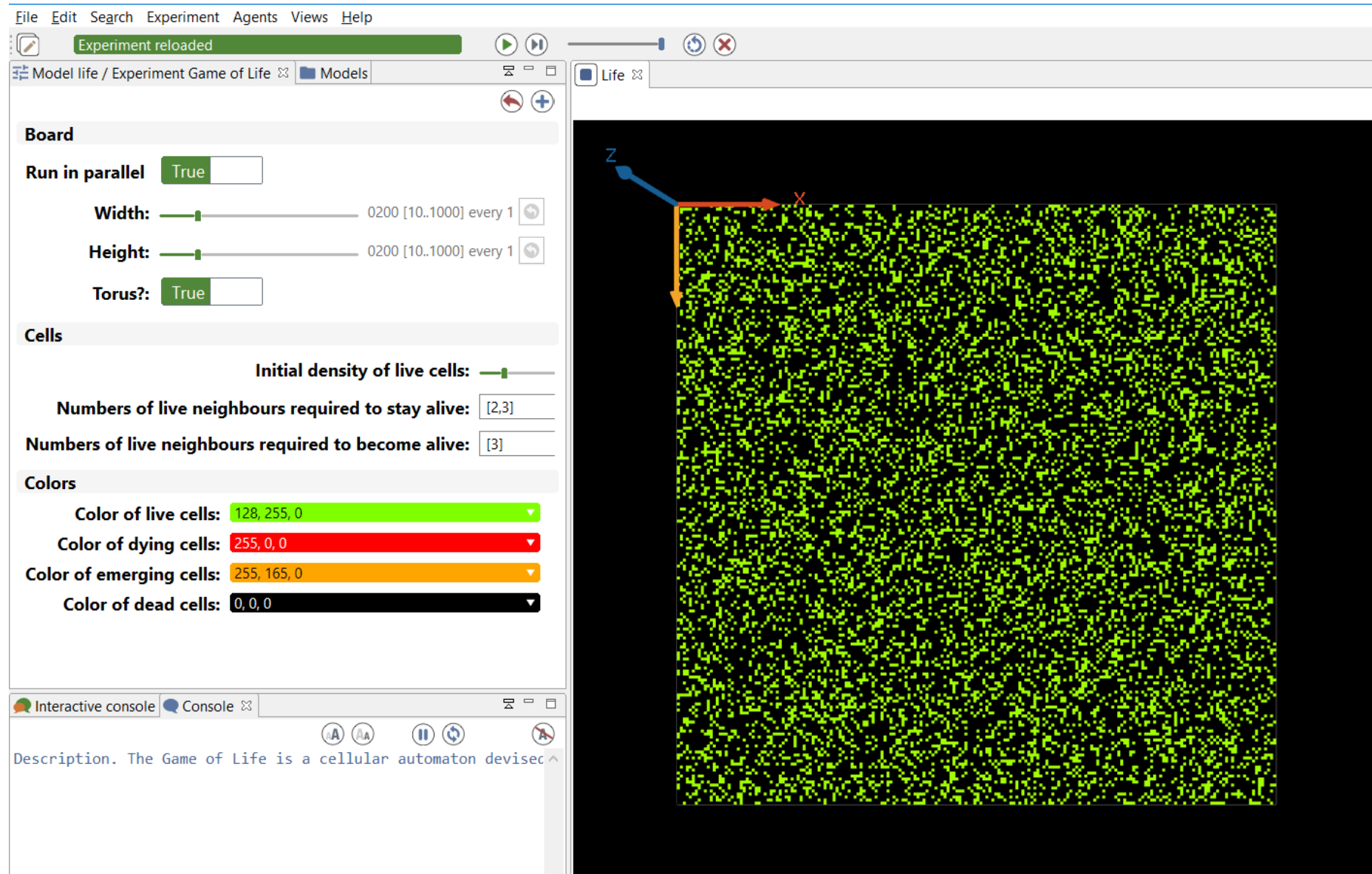
1  /**
2  * Name: Life
3  * Author:
4  * Description: A model using a cellular automata to represent the Game of Life, the most famous
5  * example of cellular automata. Each cell will see if the number of living neighbours meets the
6  * condition to emerge or to live.
7  * Tags: grid
8  */
9  model life

10
11 //Declare the world as a torus or not torus environment
12 global torus: torus_environment {
13   //Size of the environment
14   int environment_width <- 200 min: 10 max: 1000;
15   int environment_height <- 200 min: 10 max: 1000;
16   bool parallel <- true;
17   //Declare as torus or not
18   bool torus_environment <- true;
19   //Density
20   int density <- 25 min: 1 max: 99;
21   //Conditions to live
22   list<int> living_conditions <- [2, 3];
23   //Conditions to birth
24   list<int> birth_conditions <- [3];
25   //Color for living cells
26   rgb livingcolor <- ^white;
27   //Color for dying cells
28   rgb dyingcolor <- ^red;
29   //Color for empty cells

```

Ejecutar un experimento

Al ejecutar un experimento la interfaz cambia a la perspectiva de Simulación.



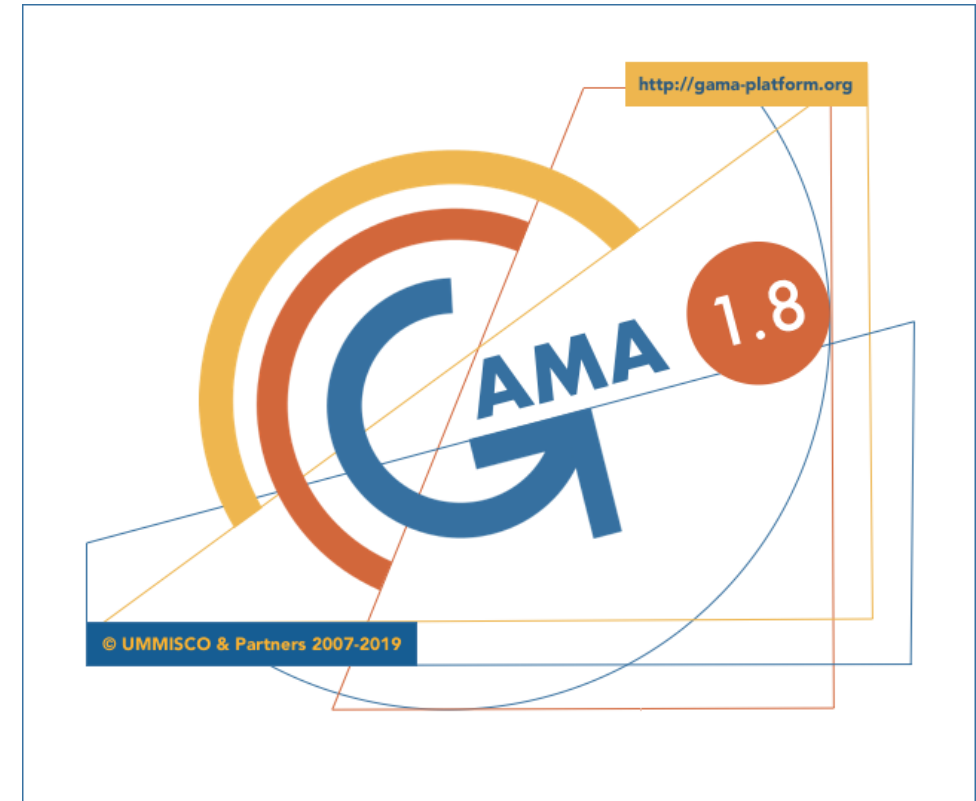
GAMA Modeling Language

GAML es un lenguaje orientado a agentes dedicado a la definición de simulaciones basadas en agentes.

Toma sus bases de lenguajes de programación orientados a objetos como Java.

Este lenguaje es parecido a otros lenguajes orientados a agentes, como NetLogo. La diferencia es que provee mayores herramientas para la creación de modelos como la herencia y agentes multinivel.

Este lenguaje es parecido a otros lenguajes orientados a agentes, como NetLogo. La diferencia es que provee mayores herramientas para la creación de modelos como la herencia y agentes multinivel.



Vocabulary correspondence with the object-oriented paradigm as in Java

GAML	Java
species	class
micro-species	nested class
parent species	superclass
child species	subclass
model	program
experiment	(main) class
agent	object
attribute	member
action	method
behavior	collection of methods
aspect	collection of methods, mixed with the behavior
skill	interface (on steroids)
statement	statement
type	type
parametric type	generics

1. Variables

Las variables son declaradas de manera sencilla en GAML. Se inician con la palabra clave del tipo de variable, seguido del nombre que se le quiere poner a la variable. La declaración de variables debe hacerse dentro de *global*, *experiment*, o *species*.

```
typeName myVariableName;
```

Todos los tipos “básicos” están presentes en GAML: *int*, *float*, *string*, *bool*.

El operador para la asignación en GAML es `<-` (ya que “=” es utilizado para comprobar igualdad entre variables).

```
int integerValue <- 3;  
float floatValue <- 2.5;  
string stringVariable <- "test"; // you can also write simple ' : <- 'test'  
bool booleanVariable <- true; // or false
```


1. Variables

Para monitorear el comportamiento de una variable, podemos escribir su valor en la consola. Regresemos al esqueleto básico de un modelo, y vamos a crear un comportamiento *reflex* en el alcance *global*. La sentencia *write* funciona de una manera muy sencilla, sólo necesitamos escribir *write* seguido de la variable que queremos mostrar.

```
model firstModel

global {
  int integerValue <- 3;
  float floatValue <- 2.5;
  string stringVariable <- "test"; // you can also write simple ' : <- 'test'
  bool booleanVariable <- true; // or false

  reflex writeDebug {
    write integerValue;
    write floatValue;
    write stringVariable;
    write booleanVariable;
  }
}

experiment myExperiment {}
```

2. Estructuras condicionales

Podemos escribir estructuras condicionales *if/else* en GAML de la siguiente manera:

```
if (integerVariable<0) {  
    write "my value is negative !! The exact value is " + integerVariable;  
}  
else if (integerVariable>0) {  
    write "my value is positive !! The exact value is " + integerVariable;  
}  
else if (integerVariable=0) {  
    write "my value is equal to 0 !!";  
}  
else {  
    write "hey... This is not possible, right ?";  
}
```

3. Valores aleatorios

Al momento de implementar un modelo necesitaremos manipular frecuentemente valores aleatorios. Para obtener un valor aleatorio en un cierto rango utilizaremos el operador *rnd*:

```
int var0 <- rnd (2); // var0 equals 0, 1 or 2
float var1 <- rnd (1000) / 1000; // var1 equals a float between 0 and 1 with a precision of 0.001
int var3 <- rnd (2, 4); // var3 equals 2, 3 or 4
float var4 <- rnd (2.0, 4.0); // var4 equals a float number between 2.0 and 4.0
float var5 <- rnd(3.4); // var5 equals a random float between 0.0 and 3.4
```

También utilizamos *flip* para tener un valor *boolean* con una probabilidad:

```
bool result <- flip(0.2); // result will have 20% of chance to be true
```

Atributos predeterminados

Algunos atributos ya existen en la especie global, como es el caso del atributo *shape*. Pero existen otros que son importantes y que pueden ser de mucha ayuda para quien construye el modelo:

cycle

```
reflex main{
  write cycle;
}
```

step

```
global {
  ...
  float step <- 10 #h;
  ...
}
```

time

```
global {
  ...
  int nb_minutes function: { int(time / 60)};
  ...
}
```

starting_date

```
init{
  starting_date <- date([2020,7,9,18,0,0]);
  create persona number:100;
}
```

current_date

```
reflex main{
  write current_date;
}
```

Species regulares

Ya hemos visto como declarar agentes regulares a nuestro gusto. Ahora veremos que estos agentes ya tienen algunos atributos predeterminados:

name. De tipo *string* es utilizado para identificar al agente de manera única.

location. De tipo *point* es utilizado para controlar la posición del agente.

shape. De tipo *geometry* describe la forma geométrica del agente.

```
species my_species {  
  init {  
    name <- "custom_name";  
    location <- {0,1};  
    shape <- rectangle(5,1);  
  }  
}
```


Comportamientos

Un comportamiento, o *reflex*, es un conjunto de sentencias que son invocadas cada paso de tiempo por un agente.

```
reflex my_reflex {  
    write ("Executing the unconditional reflex");  
    // statements...  
}
```

También podemos utilizar el parámetro *when*, para que el comportamiento se ejecute solamente cuando se cumpla la condición que se especifique.

```
reflex my_reflex when: flip(0.5) {  
    write ("Executing the conditional reflex");  
    // statements...  
}
```

Skills de los agentes

GAMA permite incrementar las capacidades de los agentes con *skills* a través del respectivo parámetro. Estos son módulos preprogramados que proveen un conjunto de atributos y acciones a los agentes.

Para declarar un skill se hace de la siguiente manera:

```
species my_species skills: [skill1,skill2] {  
}
```

Veamos el skill *moving*:

```
species my_species skills: [moving] {  
}
```

Una vez que el agente ya tiene el skill *moving* automáticamente obtiene los siguientes atributos: **speed**, **heading**, **destination**. Así como las siguientes acciones: **move**, **goto**, **follow**, **wander** y **wander_3D**.

Agentes desde archivos GIS

Podemos leer automáticamente archivos GIS que tengan la extensión **.shp* (shapefiles). Por lo tanto debemos definir variables globales de la siguiente manera:

```
global {  
  file shape_file_buildings <- file("../includes/building.shp");  
  file shape_file_roads <- file("../includes/road.shp");  
}
```

Lo siguiente declarar los agentes que necesitamos:

```
species road {  
  aspect default {  
    draw shape color: #black;  
  }  
}  
  
species building {  
  aspect default {  
    draw shape color: #gray border: #black;  
  }  
}
```



Agentes desde archivos GIS

Crear los agentes a partir de los archivos que acabamos de importar es muy sencillo. Para ello sólo tenemos que utilizar el comando **create**, con el parámetro **from**, de la siguiente manera:

```
create road from: roads_shapefile;  
create building from: buildings_shapefile;
```

Todo dentro del bloque *init* del agente *global*.



```

11 import "Traffic.gaml"
12
13 global {
14   float   traffic_light_interval parameter: 'Traffic light interval' init: 30#s;
15   float   seed                  <- 42.0;
16   float   step                   <- 0.5#s;
17   date    starting_date          <- date([2022,10,8,0,0,0]);
18   string  scenario               <- "experimento_1";
19   string  output_path            <- "../includes/output/";
20   bool    export                 <- false;
21   bool    activate_intervention  <- false;
22
23   string  map_name               <- "rouen";
24   file    shp_roads              <- file("../includes/" + map_name + "/roads.shp");
25   file    shp_nodes              <- file("../includes/" + map_name + "/nodes.shp");
26
27   geometry shape                 <- envelope(shp_roads) + 50;
28
29
30   graph road_network;
31   map edge_weights;
32   list<intersection_recolector> non_deadend_nodes;
33
34   // Variable para almacenar el no. de coches esperando para cruzar una intersección
35   map<string,int> congested_road <- ["Top1"::0,"Top2"::0,"Top3"::0,"Top4"::0,"Top5"::0];
36
37
38   + init {
39
40
41
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
71   // Reflejo que permite pausar la simulación
72   + reflex stop_simulation when: cycle = 600
73
74   }
75
76
77
78
79
80   + species vehicle_random parent: base_vehicle {
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96   + species intersection_recolector parent: intersection
97
98
99
100
101
102
103
104
105
106
107   + experiment city type: gui {

```

Constructor

Agente

Experimento

Referencias

- Ladyman, J., Lambert, J. H., & Wiesner, K. (2012). What is a complex system? *European Journal for Philosophy of Science*, 3(1), 33-67. <https://doi.org/10.1007/s13194-012-0056-8>
- Introduction to the Modeling and Analysis of Complex Systems. H. Sayama (Ed.). (2015, Open SUNY Textbooks). Free open access PDF, 498 pp. ISBN 978-1-942341-06-2 (deluxe color edition). ISBN 978-1-942341-08-6 (print edition). ISBN 978-1-942341-09-3 (ebook). (2016, 1 agosto). <https://ieeexplore.ieee.org/document/7547403>
- Maria, A. (1997, December). Introduction to modeling and simulation. In Proceedings of the 29th conference on Winter simulation (pp. 7-13).
- Duma, M. (2008). *Agents, agent architectures and multi-agent systems*. University of Johannesburg (South Africa). <https://www.proquest.com/docview/2562203586?pq-origsite=gscholar&fromopenview=true>
- Multiagent Systems, edited by Gerhard Weiss, MIT Press, 2013. ProQuest Ebook Central, <http://ebookcentral.proquest.com/lib/mit/detail.action?docID=3339590>
- Simmonds, J., Gómez, J. J. A., & Ledezma, A. (2019). The role of agent-based modeling and multi-agent systems in flood-based hydrological Problems: A Brief review. *Journal of Water and Climate Change*, 11(4), 1580-1602. <https://doi.org/10.2166/wcc.2019.108>
- Nwana, H. S. (1996). Software agents: An overview. *The knowledge engineering review*, 11(3), 205-244.