

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

Simulación Computacional

Simulación - Definición

- Una **simulación** es un programa de computadora que imita o duplica un sistema real.
- Los resultados de una simulación representan una “historia artificial” del sistema real.
 - Observamos estos datos para comprender las características operativas del sistema real.
- El modelado de simulación puede considerarse una herramienta
 - Análisis
 - Diseño

Simulación - Definición

- Uno de los fundamentos de cualquier simulación es la generación de una secuencia de **números pseudoaleatorios**.
 - Secuencia determinista de números cuyas propiedades se aproximan lo más posible a una secuencia de **números puramente aleatorios**.
 - Usamos la aleatoriedad para aproximarnos a los conceptos del mundo real en nuestros modelos de simulación.
 - Existe un cierto grado de interdependencia estadística entre los números aleatorios generados por computadora, y esto puede crear artefactos.
 - La calidad de los generadores de números aleatorios puede ser muy diferente.

Modelos de Simulación

- Un **modelo de simulación** permite estudiar el **comportamiento de un sistema** a lo largo del tiempo.
 - Construimos el modelo en base a **suposiciones** o **hechos** asociados al sistema.
 - Se trata de relaciones matemáticas, lógicas y simbólicas entre los componentes o **entidades** que forman un sistema.
 - Un modelo especifica los **mecanismos esenciales** del fenómeno.
 - Tenemos que validar un modelo antes de producir resultados valiosos.
 - Un modelo permite...
 - Explicar
 - Experimentar
 - Hacer analogías
 - Predecir

Modelos de Simulación

- Las variables cuyos valores son determinados por el modelador se denominan “**parámetros libres**”.
 - Estos valores suelen ajustarse a medida que maduramos el modelo o comprendemos mejor el sistema.
 - Estos parámetros controlan los supuestos del modelo.

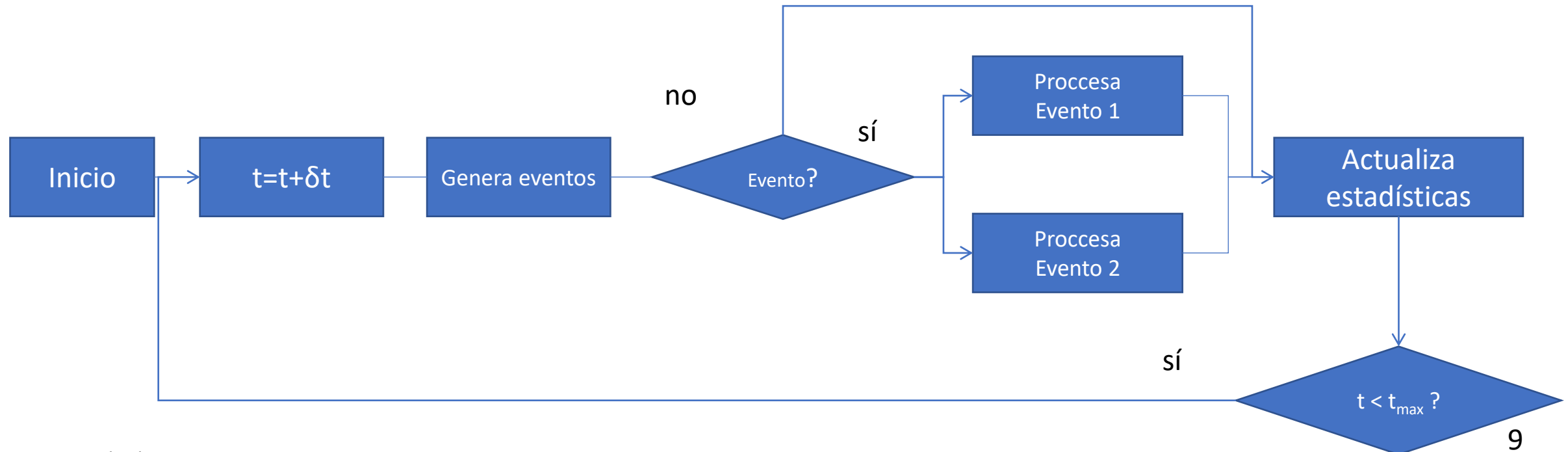
¡Los modelos son simplificaciones de la realidad!

Simulación Computacional

- **Dinámica** de sistemas (**análisis temporal**)
 - Simulación de eventos discretos (DES)
 - Simulación basada en el tiempo (TBS)
- Enfoque de modelado
 - **De arriba hacia abajo**
 - Orientado a procesos / comportamiento macro / flujo de entidades a través del sistema
 - Modelamos el sistema, no las entidades
 - **De abajo hacia arriba (Modelado basado en agentes (ABS))**
 - Basado en individuos / comportamiento micro / emergente
 - Enfoque en modelar las entidades y las interacciones entre ellas
 - Los **agentes** representan individuos
 - ✓ **La confianza en el modelo** se logra a través de pruebas y el efecto de caja blanca de poder ver una relación uno a uno entre la estructura del modelo y la estructura del sistema que se está modelando
 - ❖ “Nuestro modelo puede estar más cerca de la realidad”
 - Usamos este enfoque para **simular sistemas urbanos** de tamaño razonable.
 - Los “**agentes**” también pueden constituir colectivamente una técnica computacional (por ejemplo, la inteligencia de enjambre de Langton)
 - Por ejemplo, las estructuras de autoorganización surgen de abajo hacia arriba derivadas de las interacciones entre los agentes individuales

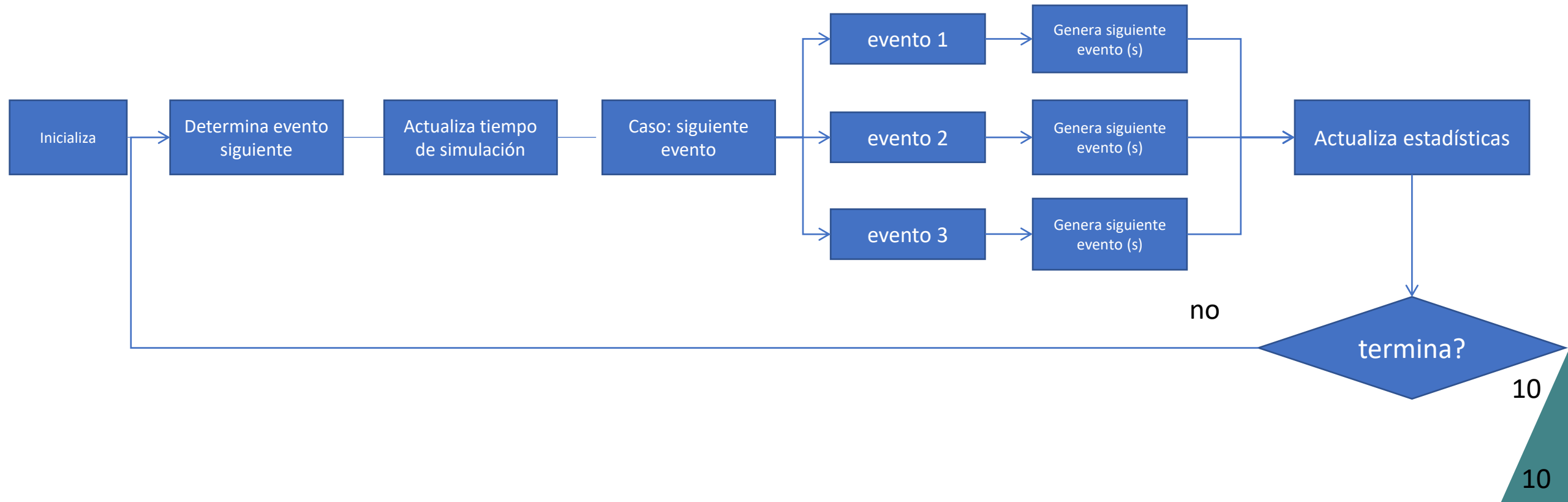
Simulación Basada en Tiempo

- Realizamos un **seguimiento continuo de la dinámica del sistema** a lo largo del tiempo.
 - Esto se hace mediante **incrementos de tiempo δt**
 - La ejecución del bucle de control principal avanza utilizando estos incrementos de reloj.
 - Normalmente se genera un número aleatorio para determinar la ocurrencia del evento.
 - Suponemos que el orden de los eventos dentro de un intervalo de tiempo no es importante.



Simulación Basada en Eventos

- El sistema se modela como una **secuencia discreta de eventos en el tiempo**.
 - Un evento cambia el estado del sistema.
 - Cada evento ocurre en un instante particular.
 - Los eventos se programan para su ejecución en un orden temporal.



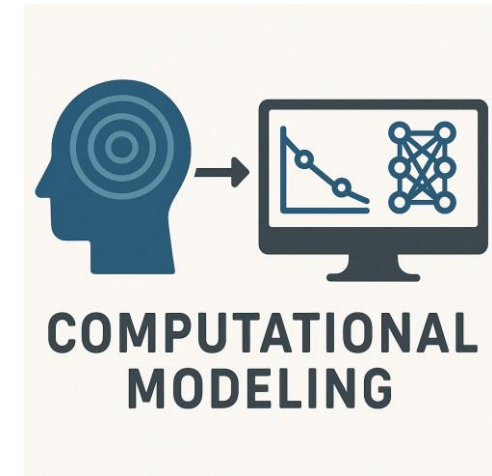
Modelado Basado en Agentes

Descarga del software

<https://github.com/gama-platform/gama/releases>

Modelado Basado en Agentes (MBA)

- Un modelo es una representación abstracta de fenómenos, procesos o *sistemas* del mundo real, en donde se *seleccionan* *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.



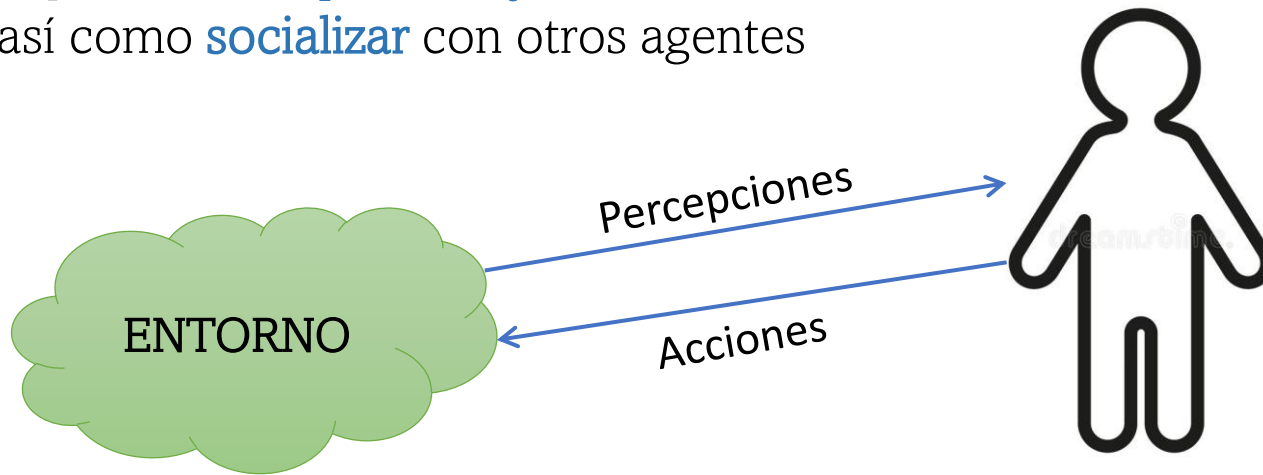
Modelado Basado en Agentes (MBA)

- 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

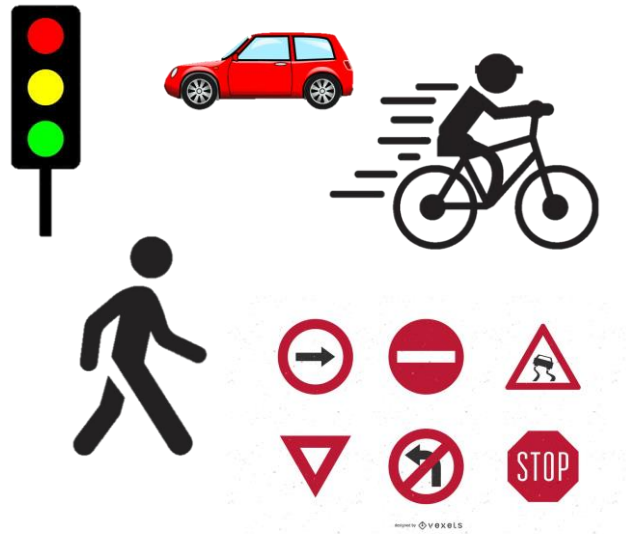


Características:

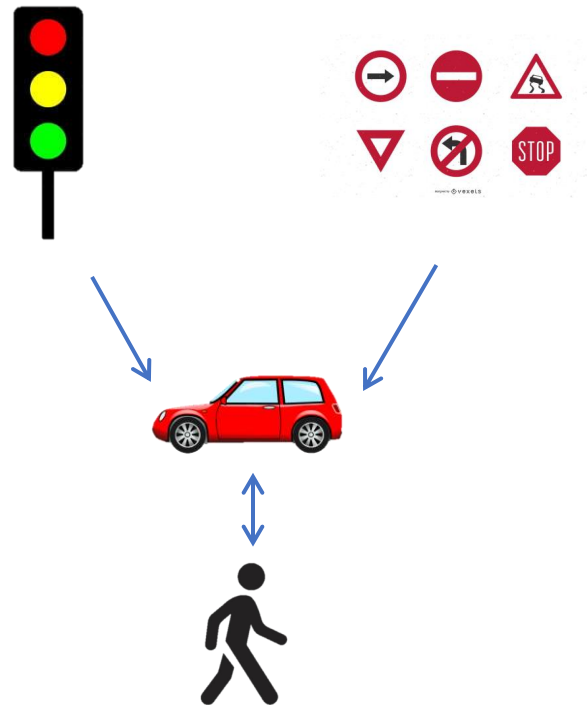
- Autonomía
- Proactividad
- Habilidad Social
- Reactividad
- Persistencia
- Razonamiento
- Productividad
- Movilidad
- Personalidad

Estructura

Conjunto de agentes



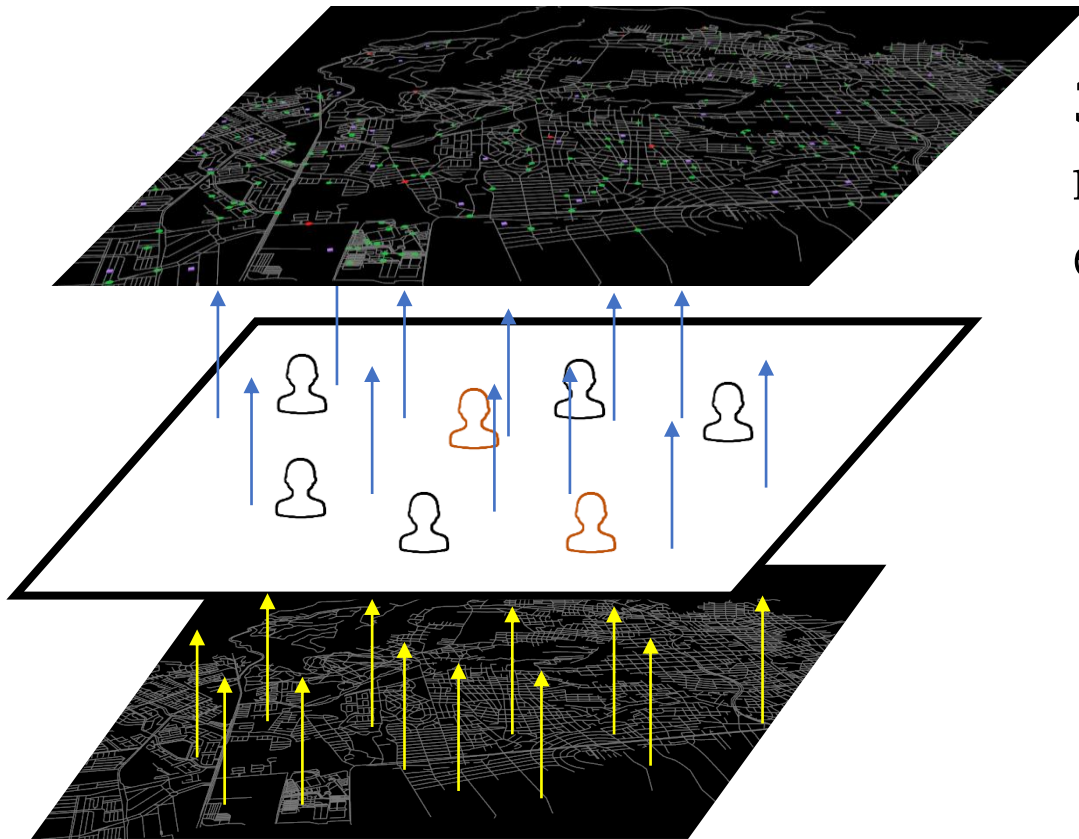
Relaciones e interacciones



Ambiente



Modelado y Simulación Computacional

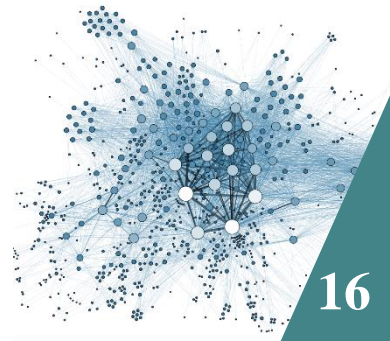


3) Ejecución de la simulación usando las reglas de comportamiento para los agentes en el escenario determinado.

2) Los agentes se ubican e inicializan

1) Creación de un escenario y carga de datos

- Se especifica el entorno



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)

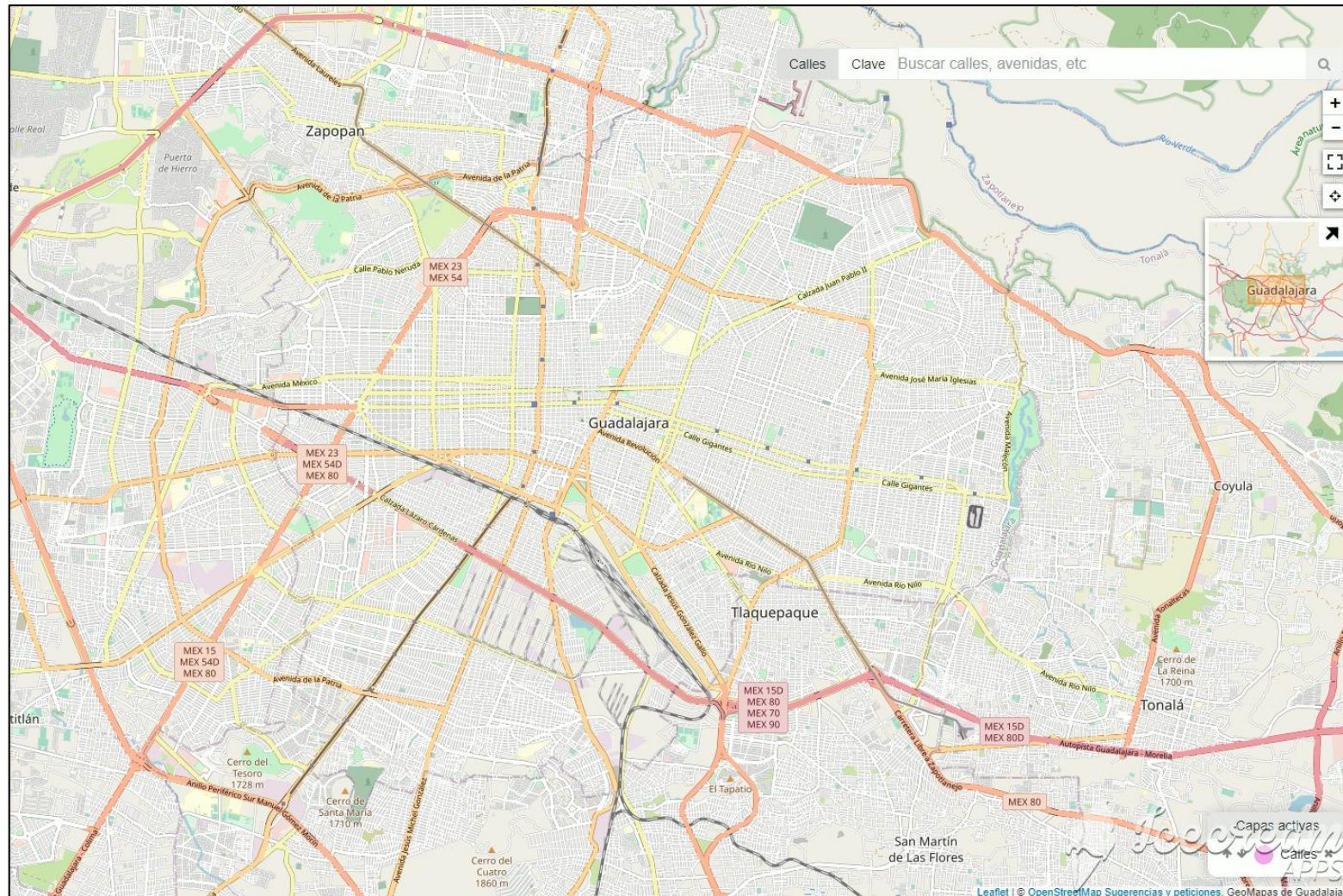
Predecir la
propagación de
epidemias
(Bagni et al, 2002)

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 a su entorno.



GAMA Platform

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

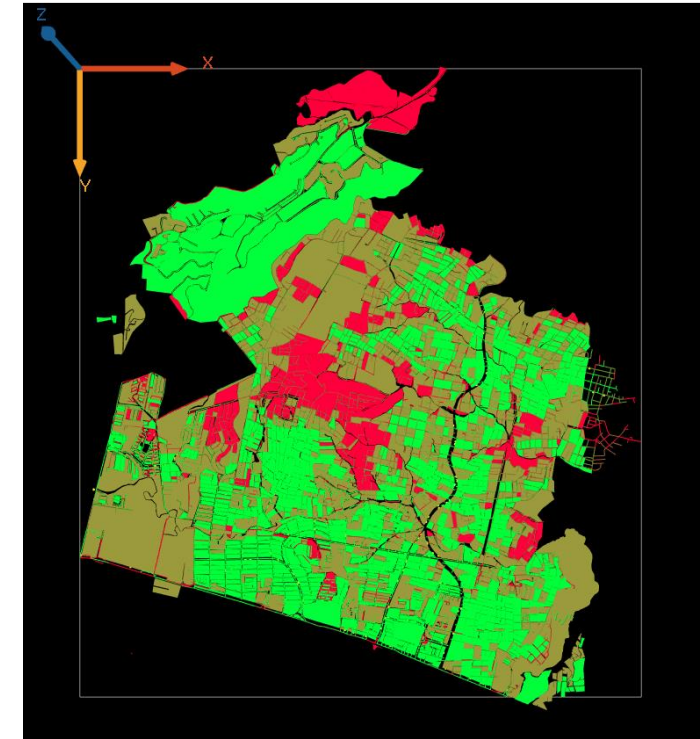
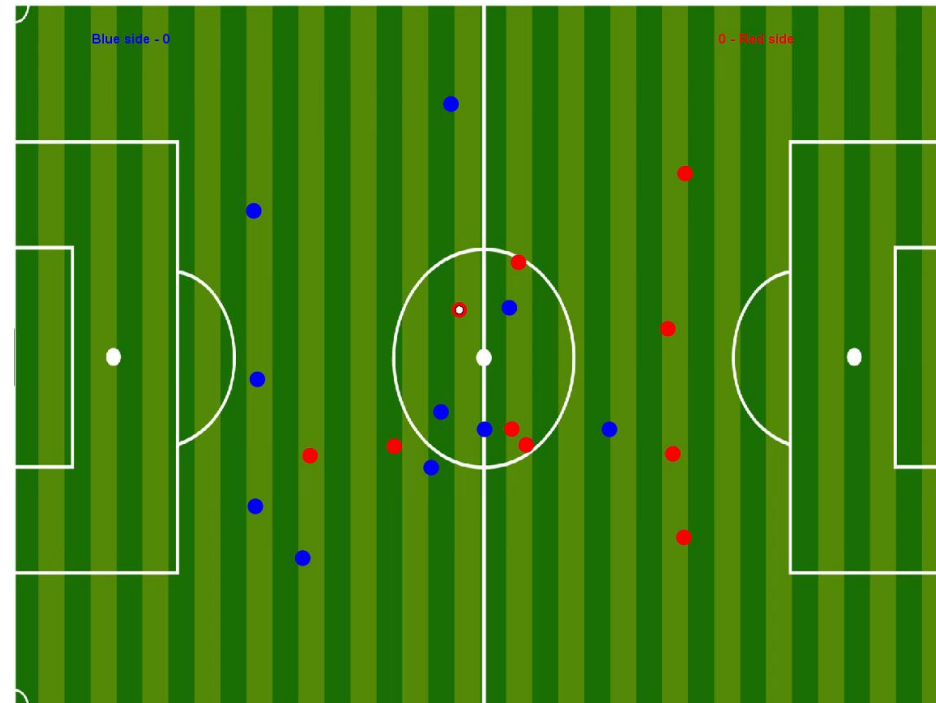
Entre sus características podemos resaltar las siguientes:

- 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.



<https://github.com/gama-platform/gama/releases>

Simulación en GAMA PLATFORM



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

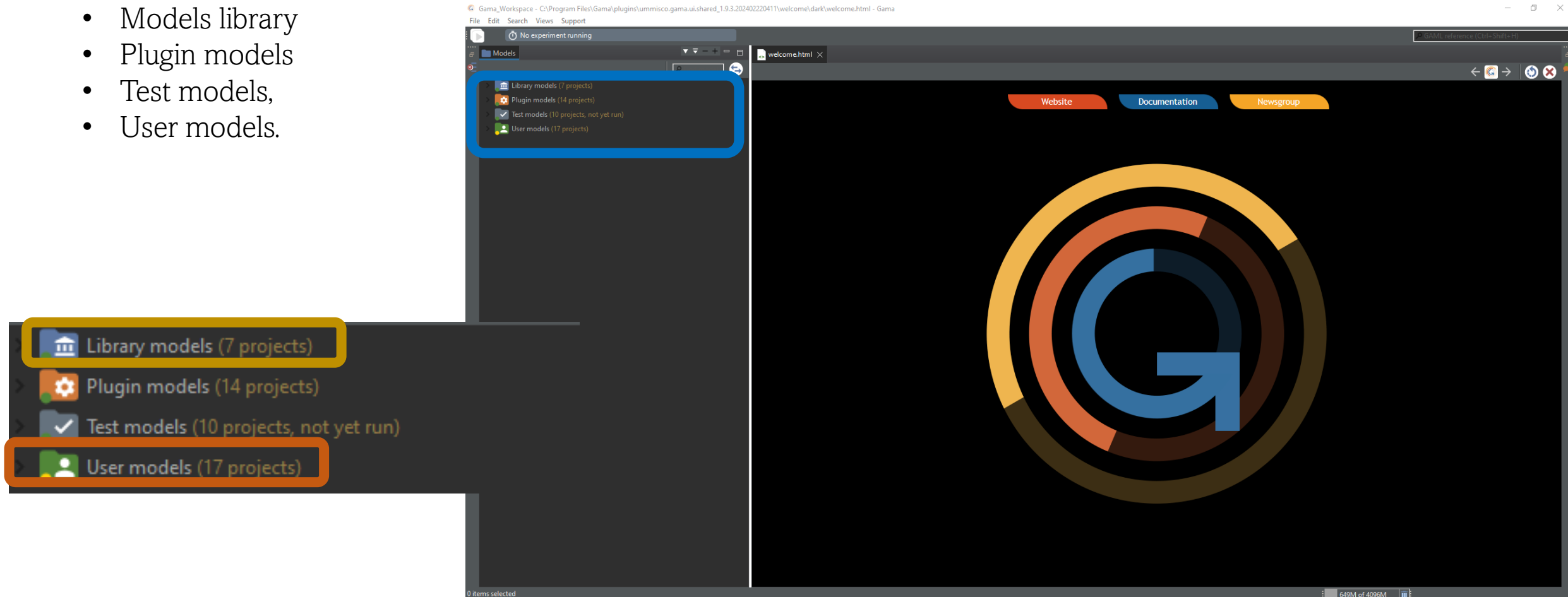
Posibilidad de realizar intervenciones en la simulación y analizar el resultado de las mismas

Página de bienvenida



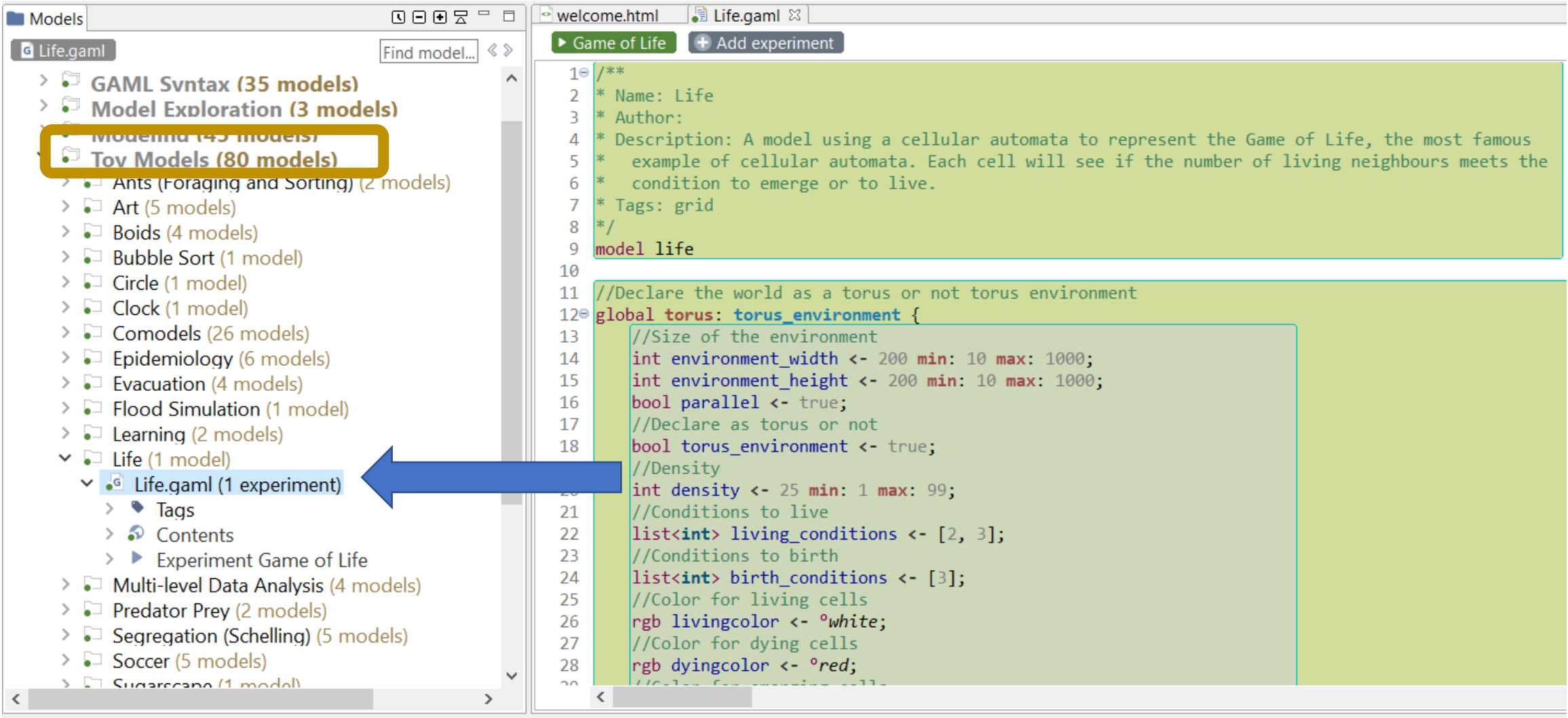
Navegar a través del espacio de trabajo

- 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,
 - User models.



Inspeccionar modelos

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



Models

Life.gaml

Find model...

- GAML Syntax (35 models)
- Model Exploration (3 models)
- Modeling (45 models)
- Toy Models (80 models)**
- Ants (Foraging and Sorting) (2 models)
- Art (5 models)
- Boids (4 models)
- Bubble Sort (1 model)
- Circle (1 model)
- Clock (1 model)
- Comodels (26 models)
- Epidemiology (6 models)
- Evacuation (4 models)
- Flood Simulation (1 model)
- Learning (2 models)
- Life (1 model)
 - Life.gaml (1 experiment) ←
 - Tags
 - Contents
 - Experiment Game of Life
- Multi-level Data Analysis (4 models)
- Predator Prey (2 models)
- Segregation (Schelling) (5 models)
- Soccer (5 models)
- Sugarscape (1 model)

welcome.html Life.gaml

Game of Life Add experiment

```

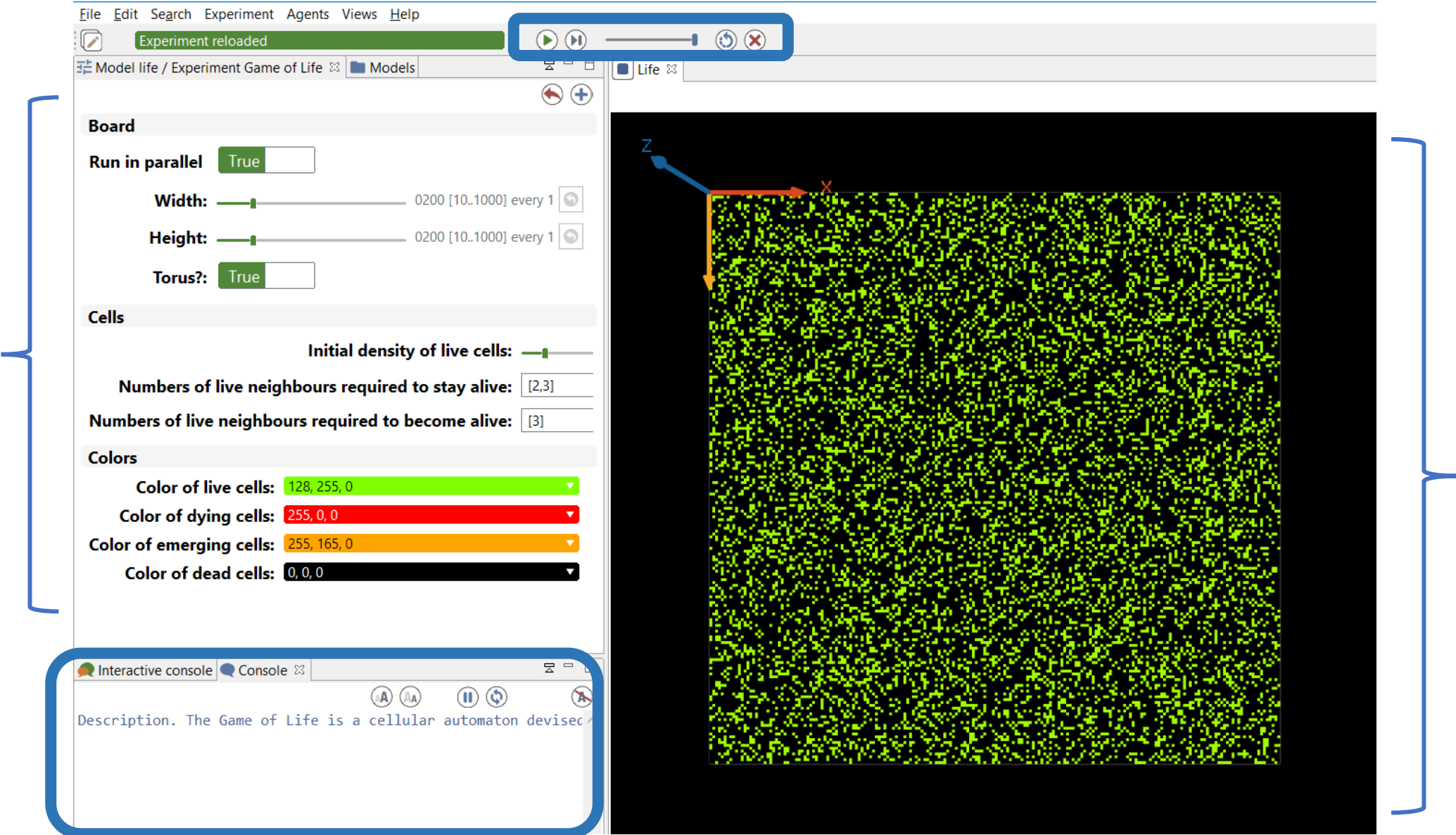
1 /**
2  * Name: Life
3  * Author:
4  * Description: A model using a cellular automata to represent the Game of Life, the most famous
5  *   example of cellular automata. Each cell will see if the number of living neighbours meets the
6  *   condition to emerge or to live.
7  * Tags: grid
8  */
9 model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

```

Ejecutar un experimento

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



File Edit Search Experiment Agents Views Help

Experiment reloaded

Model life / Experiment Game of Life Models Life

Board

Run in parallel ☒

Width: 0200 [10..1000] every 1

Height: 0200 [10..1000] every 1

Torus?: ☒

Cells

Initial density of live cells: 0.5

Numbers of live neighbours required to stay alive: [2,3]

Numbers of live neighbours required to become alive: [3]

Colors

Color of live cells: 128, 255, 0

Color of dying cells: 255, 0, 0

Color of emerging cells: 255, 165, 0

Color of dead cells: 0, 0, 0

Interactive console Console

Description. The Game of Life is a cellular automaton devised

Estructura de un modelo en GAMA

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

Agentes



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

name. identificar al agente de manera única.

location. controlar la posición del agente.

shape. describe la forma geométrica del agente.

Comportamiento del agente

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

Habilidades

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



1. Variables

Las variables 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).

```
float evaporation_per_cycle <- 5.0 min: 0.0 max: 240.0 parameter: 'Evaporation of the signal (unit/cycle):' category: 'Signals';
```

```
int numero <- 5;
float altura <- 60.4;
string nombre <- "Agente";
bool bandera <- true;
```

```
//Conditions to live
list<int> living_conditions <- [2, 3];
//Conditions to birth
list<int> birth_conditions <- [3];
```

```
rgb emergingcolor <- #orange;
```

1. Variables

- Para monitorear el comportamiento de una variable, podemos escribir su valor en la consola.
- 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* 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
```


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...  
}
```

```
reflex update_status {  
  distance_to_closest_enemy <- 100.0;  
  loop pl over:player where (each.team != team) {  
    float distance_to_enemy <- self distance_to pl;  
    if (distance_to_enemy < distance_to_closest_enemy) {  
      distance_to_closest_enemy <- distance_to_enemy;  
    }  
  }  
}
```

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;  
  }  
}
```

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;
```

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*.



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.