



UNIVERSIDAD
COMPLUTENSE
MADRID

Simulación basada en Agentes - Netlogo

Juan Pavón Mestras

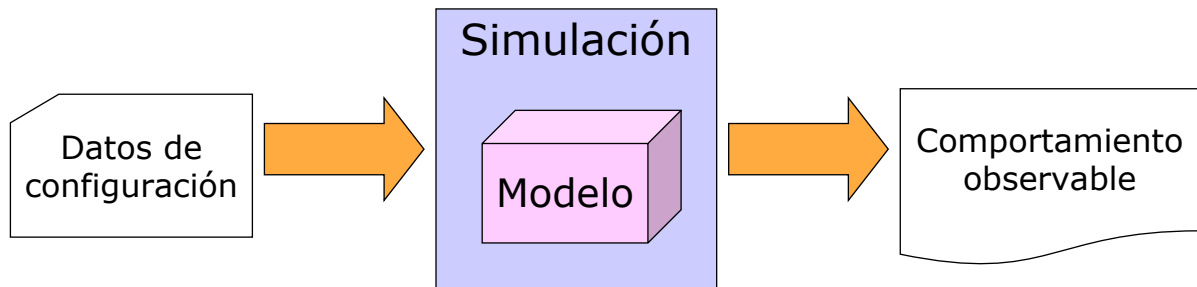
jpavon@fdi.ucm.es

Universidad Complutense Madrid

DASI

Simulación basada en agentes

Simulación

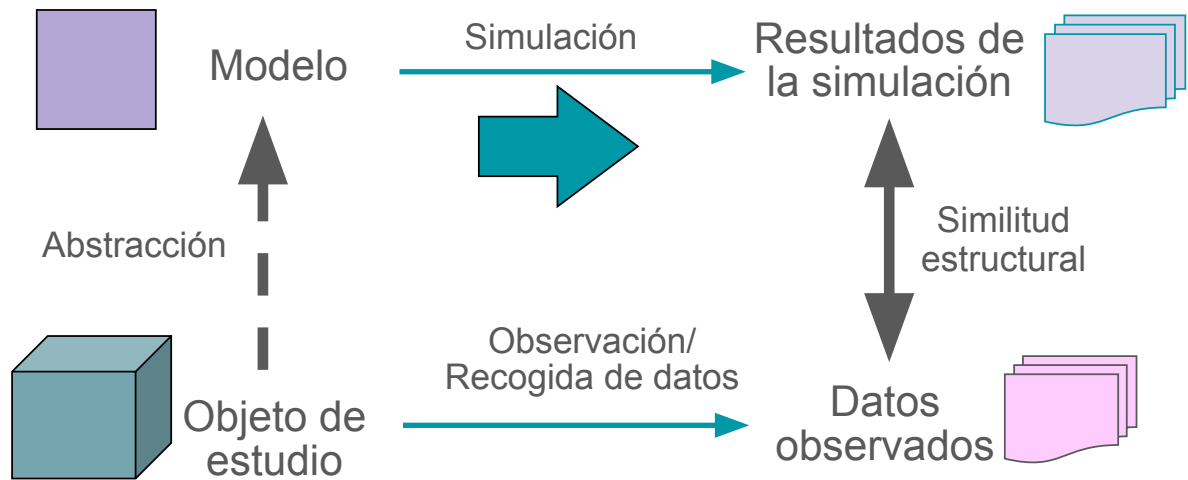


Simulación \Leftrightarrow Experimentación

- **Experimentación:** Se aplica un tratamiento a un grupo objetivo y se compara el efecto respecto a un grupo de control
 - Muchas veces esto no es posible
 - Demasiado caro
 - Demasiado complicado
 - Razones éticas

¿Se caerá un avión cuando se cambie su estructura?
¿Qué efecto tiene en la población limitar el número de hijos?
¿Cómo formar el mejor equipo para un proyecto concreto?
- **Simulación:** permite experimentar sobre un MODELO
 - Si el modelo es suficientemente bueno, reaccionará de forma similar al sistema objetivo
 - El experimento se puede repetir muchas veces, con diferentes configuraciones para analizar distintos escenarios, aleatoriedad, etc.

Modelado y simulación



Adaptado de: Nigel Gilbert and Klaus G. Troitzsch, Simulation for the Social Scientist, 2nd edition. Open University Press (2005)



J. Pavón (UCM)

DASI - Simulación basada en agentes

5

Simulación social

- Un **sistema social**
 - Una colección de individuos
 - Evolución autónoma
 - Motivados por sus propios deseos y objetivos personales
 - Y su percepción de su entorno
 - Todos estos factores evolucionan en el tiempo
 - Además: evolución demográfica
 - Interactúan y se comunican entre ellos
 - Directamente
 - A través del entorno



J. Pavón (UCM)

DASI - Simulación basada en agentes

6

... y sistemas multi-agentes

- Un paradigma de software
- Un **sistema multi-agentes** es
 - Una colección de individuos (agentes)
 - Evolución autónoma
 - Motivados por sus propios deseos y objetivos personales
 - Perciben su entorno
 - Interactúan entre ellos
 - Pueden formar organizaciones
 - Evolucionan en el tiempo

⇒ El paradigma de agentes puede ser una buena abstracción para modelar sistemas sociales

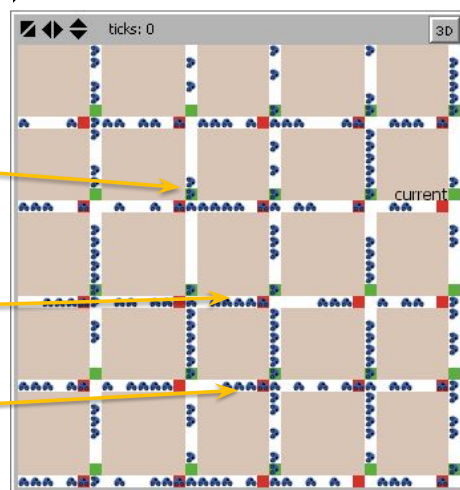


Modelado basado en agentes

Objeto de estudio

Abstracción

Modelo basado en agentes



Entidades ← ———— → Agentes

Interacciones entre entidades ← ———— → Interacciones entre agentes

Entorno ← ———— → Modelo del entorno



Modelado basado en agentes

- Los agentes definen su comportamiento individual
 - Pueden interactuar con otros agentes, perseguir objetivos, reaccionar y moverse en un entorno
 - Actúan en un entorno simulado

⇒ **Emergen propiedades** de mayor nivel (macro) a partir de las interacciones de los agentes

- Ejemplo:

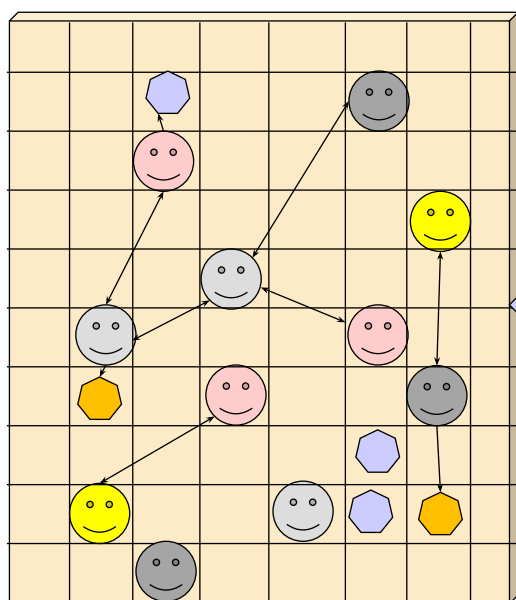
Gestión del tráfico

- Cada agente es un vehículo
- Los agentes reaccionan ante la presencia de otros vehículos
- Cada agente tiene un objetivo propio:
llegar a su destino

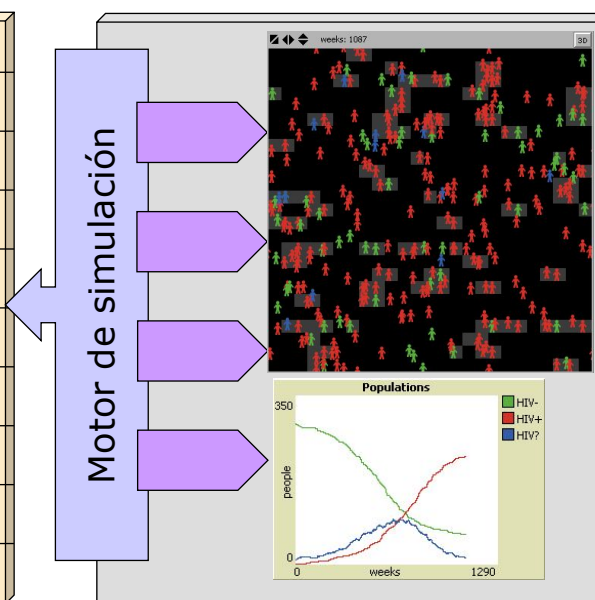
⇒ Los atascos de tráfico emergen de la interacción de los agentes



Nuestro laboratorio social



Modelo



Observador

Adaptado de: José M. Galán, *Simulación basada en agentes de juegos evolutivos en redes de normas*. Presentación UCM 2009



Simulación social basada en agentes

- **Ejecución de agentes en un entorno de simulación**
 - Los agentes modelan tipos de comportamiento específicos
 - Los agentes interactúan
 - Directamente (mensajes)
 - A través del entorno (espacio compartido, feromonas, etc.)
 - El resultado es un comportamiento emergente
 - Visualización de la simulación
 - Gráficos de resultados
 - Registros (logs) de la ejecución
 - **Los agentes**
 - Tienen una **percepción subjetiva**
 - No tienen un conocimiento global
- ⇒ Es más realista, flexible y sencillo si el agente solo puede ver su vecindad
- ... y una **racionalidad limitada**
 - Herbert Simon (1947-1957)

Movimiento e interacción con el entorno

- **Relevancia de las interacciones locales**
 - Las interacciones humanas ocurren en un lugar en el espacio
 - Las interacciones locales son más importantes que las distantes
- Los agentes están **localizados en el espacio**, con capacidad para moverse
 - Hay reglas que determinan su **movimiento**
- Los agentes pueden **reconocer a otros agentes**, y ver si son similares o no
 - Pueden exhibir comportamientos diferentes dependiendo del grado de similitud con otros agentes
 - Es posible establecer redes sociales, que determinan relaciones entre grupos de agentes

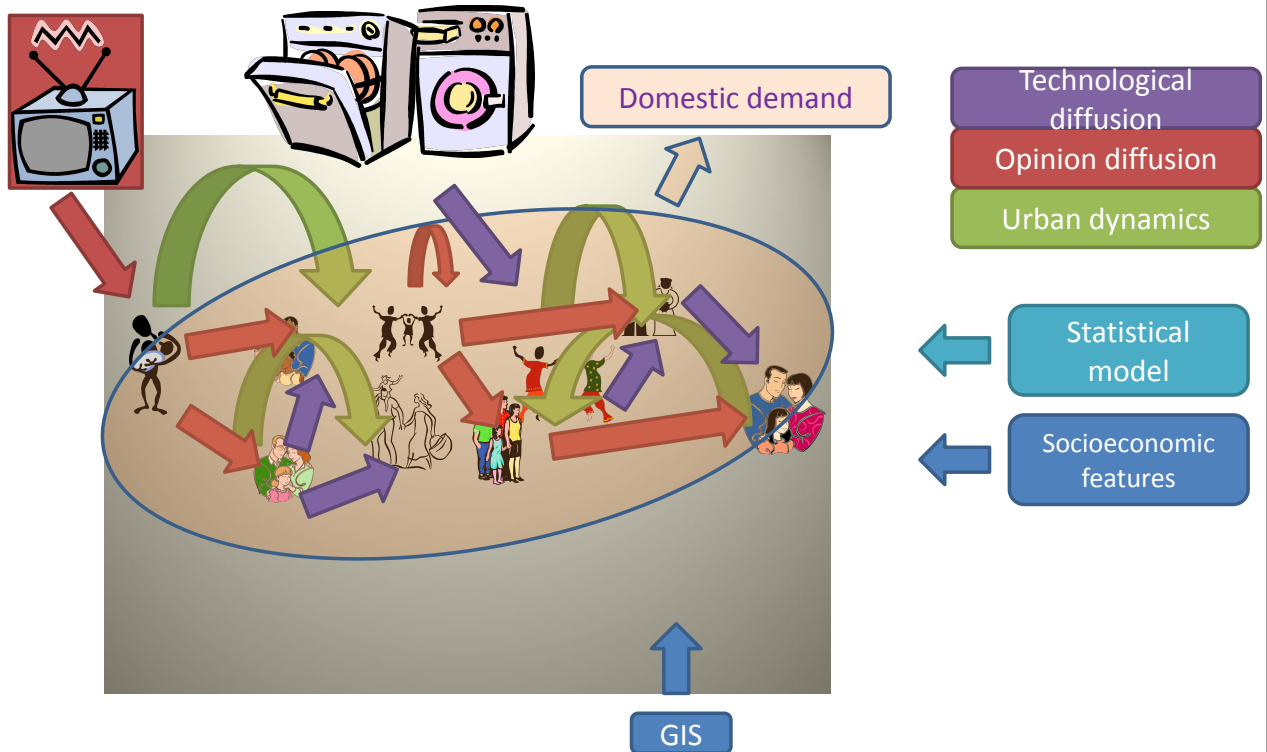
Herramientas para simulación con agentes

- Java
 - Swarm (www.swarm.org)
 - Modelo seguido por otras (Ascape, Mason, RePast)
 - RePast (repast.sourceforge.net)
 - **Mason** (cs.gmu.edu/~eclab/projects/mason/)
 - Anylogic (<http://www.xjtek.com/anylogic>)
 - SeSAm (www.simsesam.de)
- Otros lenguajes
 - **NetLogo** (ccl.northwestern.edu/netlogo/)
 - Evolución de StarLogo
 - Basado en el lenguaje Logo language, fácil de utilizar
 - Strictly Declarative Modeling Language, SDML (sdml.cfpm.org)
 - Multi-Agent Simulation Suite (mass.aitia.ai)
- Plataformas de agentes
 - JADE (<http://jade.tilab.com/>)

Aplicaciones de la simulación social con agentes

- Mejorar la comprensión de fenómenos sociales
 - Observando la evolución bajo ciertas premisas
 - Diagnóstico
- Descubrimiento de comportamientos emergentes
- Formalización y validación de teorías sociales
 - Del texto informal al modelo computacional
- Predicciones
 - Cómo evolucionará un sistema social bajo ciertas condiciones
- Formación
 - Modelos económicos
 - Mercados de bienes y servicios (electrico)

Ejemplo: Políticas de gestión del agua en una ciudad



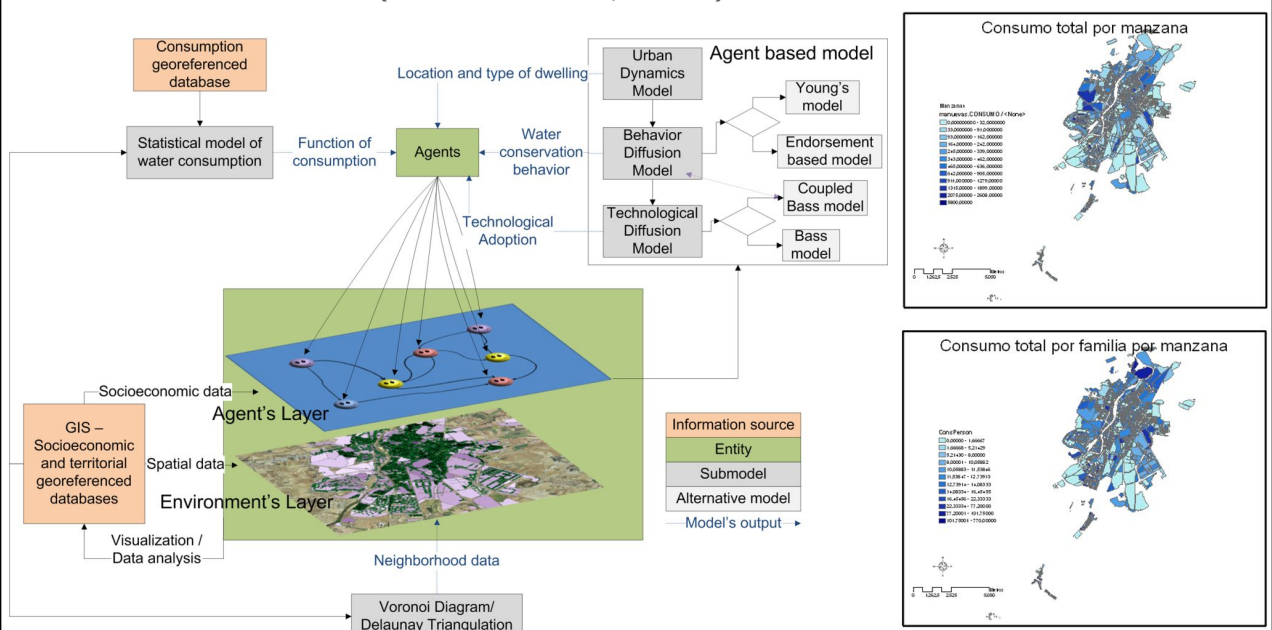
J. Pavón (UCM)

DASI - Simulación basada en agentes

15

Ejemplo: Políticas de gestión del agua en una ciudad

Domestic water management in Valladolid
José Manuel Galán (UBU INSISOC, 2007)



J. Pavón (UCM)

DASI - Simulación basada en agentes

16

Ejemplo: Agentfly [Michal Pěchouček, 2008]



J. Pavón (UCM)

DASI - Simulación basada en agentes

17

Ejemplo: Tactic Agentfly [Michal Pěchouček, 2008]



J. Pavón (UCM)

DASI - Simulación basada en agentes

18

Ejemplo: SociAAL



Menu Pause < 256.0 >

Date and Time
08:06:16 Mon, 1 Jan 2018

SimTalkToTV

This simulation models one of the patients symptoms which is that she used to talk to the people inside the TV. For this reason, when the patient is seated in front of the TV she has a 0.05 chance of saying Hello man inside the TV

Yellow: means Finished
Green: means started

FINISHED: BActivity2 Simtime: 07:31:22 Sleeping	DEFAULT: BActivity2 Simtime: 07:31:08 Sleeping
FINISHED: BActivity0 Simtime: 07:30:03	DEFAULT: BActivity0 Simtime: 07:30:03
STARTED: BActivity5 Simtime: 08:04:39 Making breakfast: - She made her breakfast... And she ate two eggs! She had american breakfast. (...) and she made our breakfast too (...) since I was little (...) -	FINISHED: BActivity4 Simtime: 08:04:14 Waking up: - She used to rise early, she always woke up at around 8 or 7.30, so she had little sleep. She would sleep for around 6 hours -

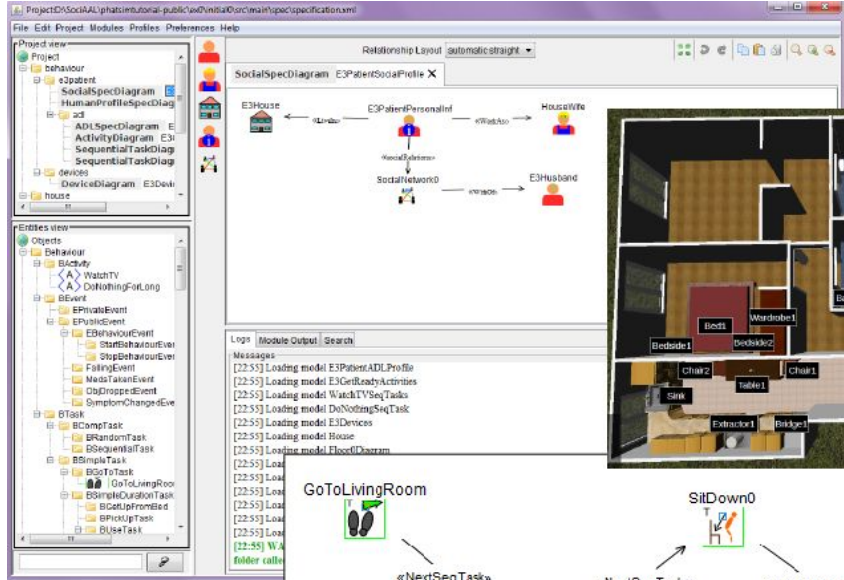
CC BY SA J. Pavón (UCM)


DASI - Simulación basada en agentes

19

Ejemplo: SociAAL

<http://grasia.fdi.ucm.es/hackwithpeople/>





GoToLivingRoom → «NextSeqTask» → SwitchOnTV → «NextSeqTask» → SitDown0 → «NextSeqTask» → Wait30secs → «NextSeqTask» → StandUp0

CC BY SA J. Pavón (UCM)

DASI - Simulación basada en agentes

20

- Juego de programación (Java) con el objetivo de codificar un carro de combate robótico para competir contra otros robots en un campo de batalla
- El jugador debe escribir la IA del robot diciéndole cómo comportarse y reaccionar ante los acontecimientos que se produzcan en la arena de batalla
- Las batallas se desarrollan en tiempo real y en pantalla



DASI

Netlogo

Netlogo

- Entorno de modelado para simular fenómenos sociales
 - Uri Wilensky et al. (1999)
Center for Connected Learning and Computer-Based Modeling,
Northwestern University (USA)
- Información y descargas:
<http://ccl.northwestern.edu/netlogo/>
- La instalación es fácil
 - Mac, Windows, Linux
 - Descargar fichero y usar el instalador
 - Lleva su propia máquina virtual de Java
 - Trae una **librería de modelos**
 - Biología, química, física, informática, juegos, matemáticas, redes, sociología, etc.

Historia

- LOGO (Papert & Minsky, 1967)
 - teoría educativa basada en el constructivismo de Piaget (creación sobre la marcha y prueba de conceptos)
 - Lenguaje sencillo derivado de LISP
 - Gráficos con una tortuga y exploración de *micromundos*
- StarLogo (Resnick, 1991), MacStarLogo, StarLogoT
 - Language de simulación basado en agentes
 - Explorar el comportamiento de sistemas descentralizados con la programación concurrente de 100 tortugas
- NetLogo (Wilensky, 1999)
 - Extensión de StarLogo (multi-plataforma, red, etc.)
 - El más popular (librería de modelos cooperativa)

El mundo Netlogo

- Netlogo es
 - un mundo 2D o 3D
 - con 4 tipos de agentes:
 - **turtles** – tortugas, se mueven por los patches (agentes móviles)
 - **patches** – parcelas del entorno (agentes estacionarios) por donde se mueven las tortugas
 - **links** – conexiones entre tortugas
 - **observer** – uno, quien observa y controla lo que hacen los agentes

El objeto mundo (world)

Configuración del mundo

World

Location of origin: Corner
Bottom Left

min-pxcor 0
minimum x coordinate for patches
max-pxcor 39
maximum x coordinate for patches
min-pycor 0
minimum y coordinate for patches
max-pycor 39
maximum y coordinate for patches
Torus: 40 x 40
☒ World wraps horizontally
☒ World wraps vertically

View

Patch size 10
measured in pixels
Font size 10
of labels on agents
Frame rate 30
Frames per second at normal speed

Tick counter

☒ Show tick counter
Tick counter label ticks

OK Apply Cancel

Este objeto aparece siempre en la ventana inicial de Netlogo. Permite visualizar el comportamiento de los agentes, y cambiar la visualización de 2D a 3D.

Netlogo – agentes

■ Elementos conceptuales

■ **Turtles:** son los agentes en el modelo de simulación

- Tienen un estado y coordenadas en el entorno
- Pueden moverse por el entorno



■ **Patches:** Son las subdivisiones del mundo, donde se ubican los agentes

- Se acceden mediante sus coordenadas.
- Pueden tener estado y evolucionar



■ **Links:** Permiten definir relaciones entre agentes

- No tienen coordenadas, solo dos extremos (tortugas que relacionan, si una muere, el link desaparece)
- Se representan con una línea entre los agentes

■ **Observer:** Es la persona que ve y realiza modificaciones al entorno de vida de los agentes



Modelos

■ Cada modelo tiene

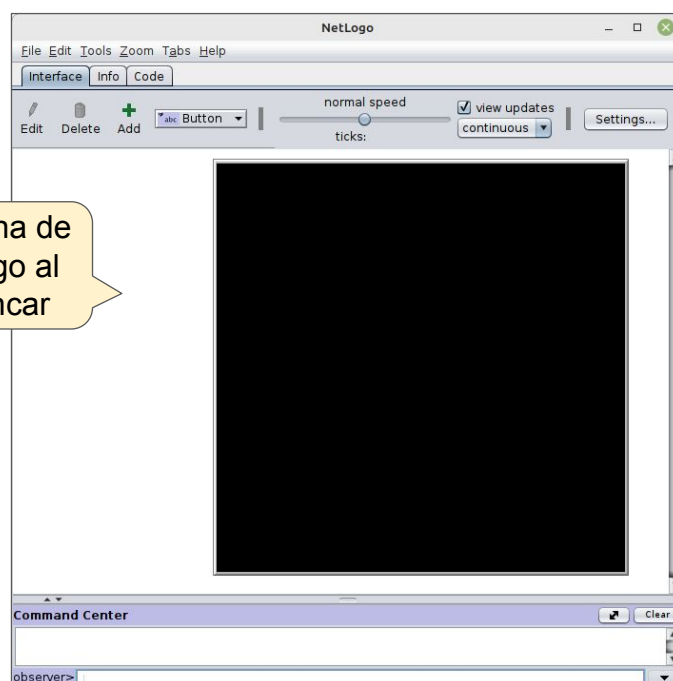
- **Interface**
(pestaña *Interface*)
- **Information**
(pestaña *Info*)
- **Procedures**
(pestaña *Code*)

- Al arrancar empieza con un modelo vacío

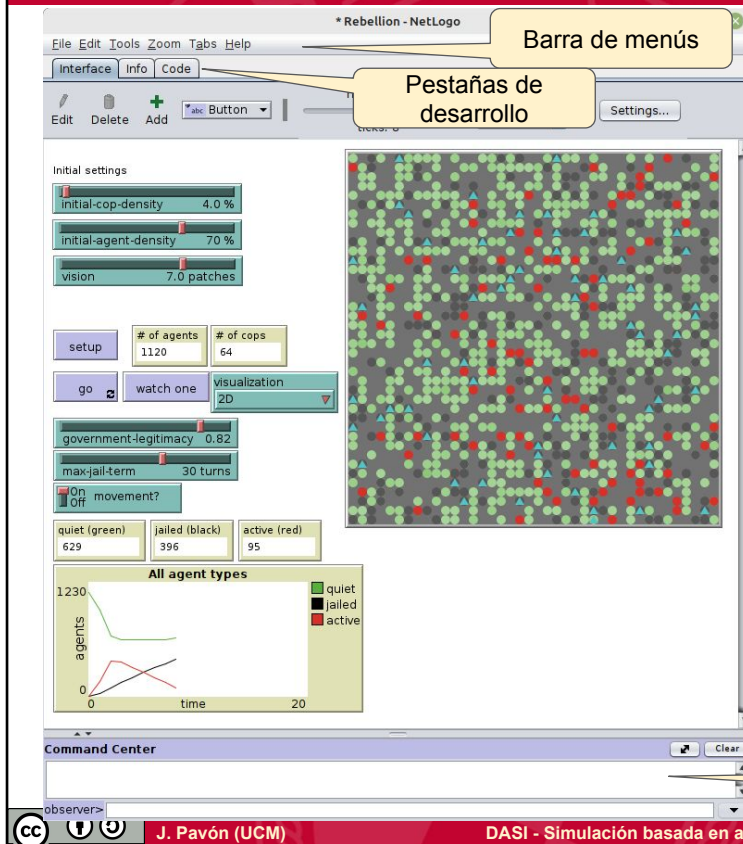
- Se puede crear uno nuevo con
File → **New**

- O utilizar uno de la librería
File → **Models library**

Ventana de netlogo al arrancar



Interfaz



Barra de menú:

Contiene los comandos clásicos de Windows

Pestañas de desarrollo:

Permiten elaborar por separado los componentes del modelo (interfaz, descripción y código fuente)

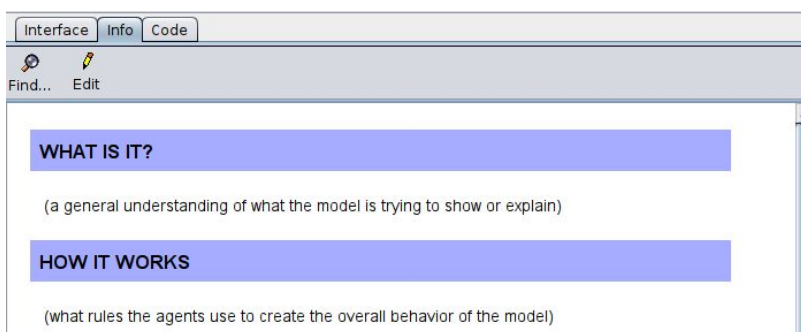
Línea de comandos:

Permite introducir instrucciones para interactuar con el modelo

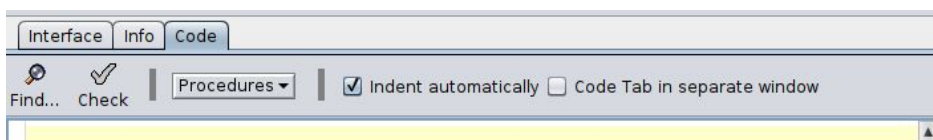
Línea de comandos

Pestañas de desarrollo

Info: Permite editar la descripción del modelo



Code: Permite tener acceso a la ventana de edición de código fuente, además de chequear la sintaxis



Pestañas de desarrollo

Interface Info Code

Edit Delete Add **Button** normal speed view updates continuous Settings...

ticks: 0

Controla la velocidad de la simulación

Button: crea botones de comando
Slider: crea un deslizador que permite cambiar el valor de las variables
Chooser: crea un objeto de selección de alternativas
Input: crea una caja de entrada de valores para una variable
Monitor: permite visualizar los valores de una variable
Plot: permite construir una gráfica de variables del modelo
Output: permite ver información por texto del modelo original
Note: permite colocar textos en el área de trabajo

Simulación: 1. Poner la configuración inicial

El Farol - NetLogo

File Edit Tools Zoom Tabs Help

Interface Info Code

Edit Delete Add **Button** normal speed view updates on ticks Settings...

ticks: 0

memory-size 5

number-strategies 10

overcrowding-threshold 60

setup go

Bar Attendance

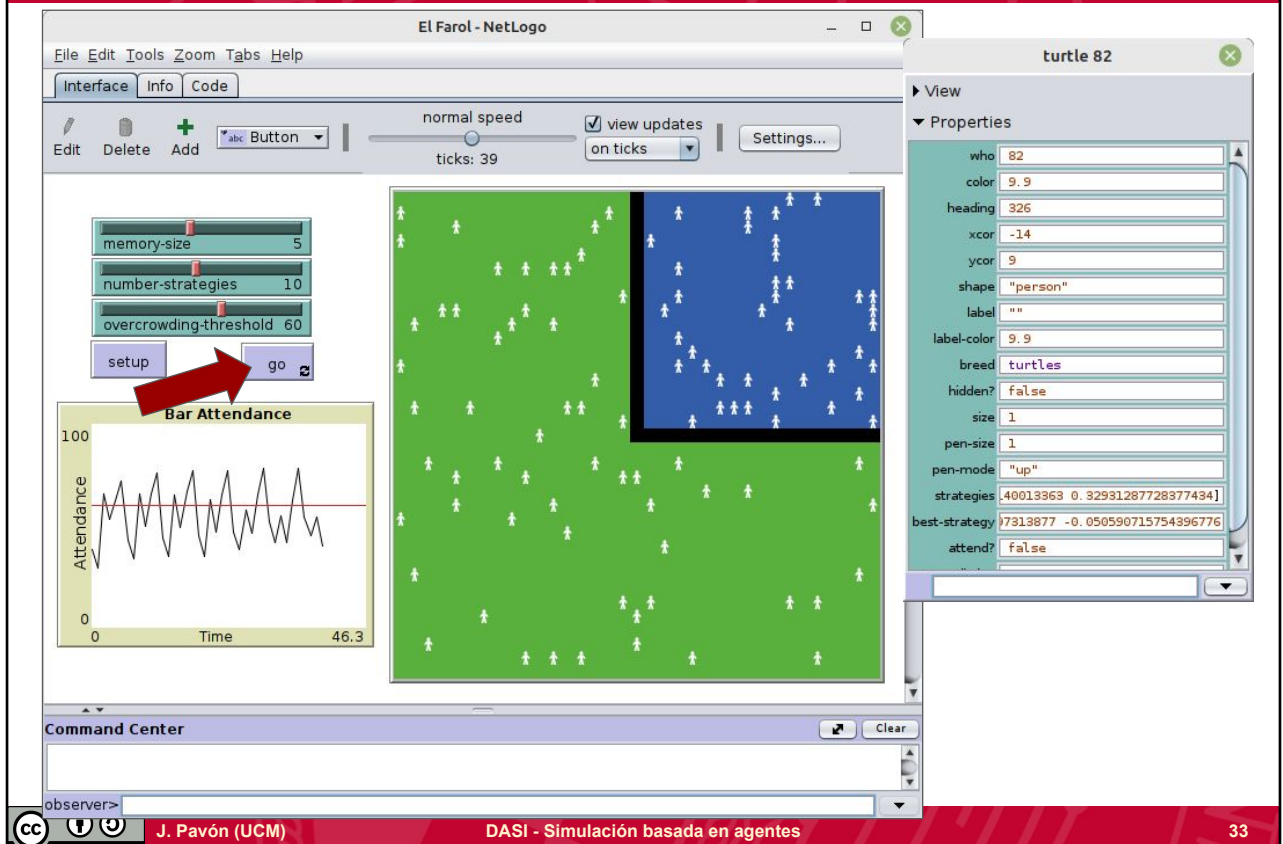
Attendance 66.3

0 1.36 Time 10

Command Center

observer>

Simulación: 2. Ejecutar el modelo



Programación de un modelo

- Variables globales
- Variables de los agentes
- Procedimientos
 - De la interfaz
 - Sub-procedimientos
 - Sin valores de retorno
 - Con valores de retorno

Variables globales

- No se especifica su tipo, pueden guardar cualquier cosa

```
globals [  
  attendance      ;; tiempo para ser atendidos en el bar  
  history         ;; lista de tiempos de espera pasados  
  home-patches    ;; agentset de patches verdes  
  bar-patches     ;; agentset patches azules (el bar)  
  crowded-patch   ;; patch con la etiqueta "CROWDED"  
]
```

Variables de los agentes

- Cada agente tiene su propia copia

```
turtles-own [  
  strategies      ;; lista de estrategias  
  best-strategy   ;; índice a la estrategia actual mejor  
  attend?        ;; true si el agente tiene previsto ir al bar  
  prediction      ;; predicción actual del tiempo de espera  
]
```


Procedimientos de la interfaz

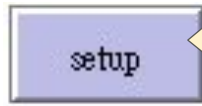
- Asociados a elementos de interfaz (p.ej. botones)

- Inicialización

`to setup`

`...`

`end`



- Activación de eventos

`to go`

`...`

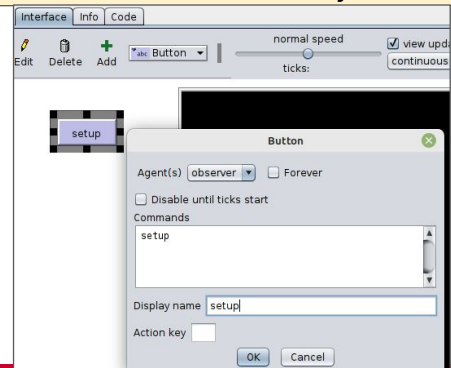
`end`



Para crear un botón “setup”:

- 1) Crear un Button
- 2) En instrucciones poner setup
- 3) En etiqueta poner setup
- 4) Escribir el procedimiento setup

- Si se pulsa un botón creado sin haber escrito el procedimiento asociado saldrá un mensaje de error



Sub-procedimientos sin valor de retorno

`to move-to-empty-one-of`

`...`

`end`

`to update-plots`

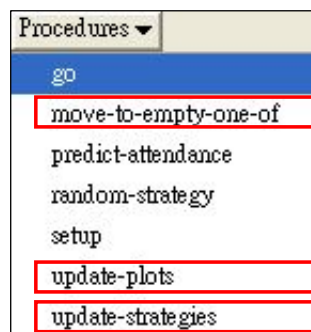
`...`

`end`

`to update-strategies`

`...`

`end`



Subprocedimientos con valor de retorno (reporters)

```
to-report predict-attendance
```

```
...
```

```
report ...
```

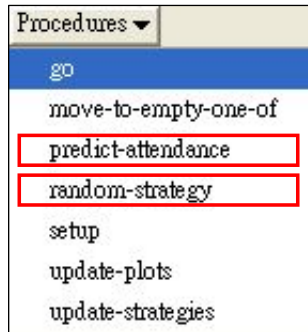
```
end
```

```
to-report random-strategy
```

```
...
```

```
report ...
```

```
end
```



Programación con Netlogo

- No distingue mayúsculas y minúsculas
- Asignar el valor de b a a: set a b
- Los operadores aritméticos deben tener un espacio a cada lado: set a 3 * b
- Se pueden usar paréntesis para controlar la precedencia
- Usar [] para definir bloques de instrucciones
- Los procedimientos empiezan con to, y acaban con end
- No hay tipos numéricos
- Las variables lógicas (true, false) acaban en ?
- ;; para indicar comentarios

Programación con Netlogo

- El comando **ask** permite pedir ejecutar un comando a un agente o varios:
 - **ask turtles [comandos] ;; al conjunto de tortugas**
 - Solo un agente puede consultar o modificar el valor de sus atributos
 - Un agente (p.ej. una tortuga) puede invocar a otros:
 - **other turtles** se refiere a las demás tortugas
 - **one-of other turtles** elige una aleatoriamente
- El comando **stop**
 - Para acabar un procedimiento o un ask que se está ejecutando

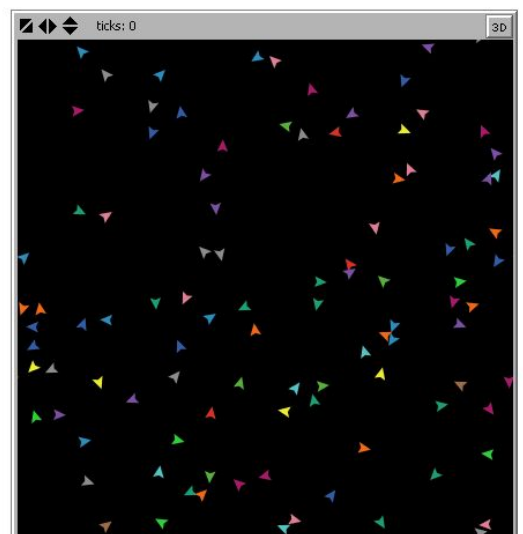
Programación de la interfaz

- Crear un botón setup y en Code poner el siguiente código

```
to setup
  clear-all
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
  reset-ticks
end
```



- Probar ahora setup
 - Y varias veces más...
- Puedes salvar el modelo
 - File → Save as



Programación de la interfaz

- Igualmente poner otro botón con instrucción *go* y con etiqueta **step** para ejecutar la simulación
 - Marcar la casilla *Disable until ticks start*
 - Se puede poner este código:

```
to go
  move-turtles
  tick
end

to move-turtles
  ask turtles [
    right random 360 ;; gira un nro aleatorio entre 0 y 359
    forward 1 ;; desplaza una posición hacia delante
  ]
end
```

primitiva de netlogo:
avanza el contador un tick

- Probar a ejecutar paso a paso
- Poner también otro botón igual pero con etiqueta **go** y marcando *Forever* (Continuamente) ⇒ Ejecutar

Procedimientos

- Es conveniente estructurar las acciones en procedimientos
 - Más modularidad y flexibilidad
 - Por ejemplo, para el método setup

```
to setup
  clear-all
  setup-patches
  setup-turtles
  reset-ticks
end
```

```
to setup-patches
  ask patches [ set pcolor green ]
end
```

```
to setup-turtles
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Variables

- Cada agente tiene unas variables por defecto y se le pueden añadir más
 - Variables de cada tortuga
 - breed (sub-tipo)
 - color (se asigna uno aleatorio al crearse)
 - heading (grados, 0 = arriba, 90 = derecha)
 - hidden?
 - label, label-color
 - pen-mode, pen-size
 - shape (forma)
 - size
 - who (un número)
 - xcor, ycor (por defecto: 0, 0)

Variables

- Se pueden añadir otras
- En el ejemplo, considerar una energía que la tortuga tiene que renovar (comiendo hierba) para no morir

```
turtles-own [energy]
```

debe ponerse antes de la declaración de los procedimientos

```
to go
```

```
  move-turtles ;; gastará energía
```

```
  eat-grass    ;; recuperará energía
```

```
  tick
```

```
end
```


Variables e instrucciones if y asignación

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor brown
      set energy (energy + 10)
    ]
  ]
end

to move-turtles
  ask turtles [
    right random 360
    forward 1
    set energy energy - 1
  ]
end
```

instrucción if



Monitorización de la simulación

- Para crear un **monitor**
 - Seleccionar monitor (opción más abajo de Button)
 - En la ventana de diálogo que aparece indicarle qué hacer:
 - count turtles**
 - **turtles** se refiere al conjunto de agentes (es un "agentset")
 - **count** dice cuántos agentes hay en el conjunto
 - También se puede pedir una condición, por ejemplo crea otro monitor para contar los patches verdes:
 - count patches with [pcolor = green]**

Monitorización de la simulación

- Para ver un atributo de las tortugas (la energía)
 - Se puede poner un switch asignándole una variable lógica (true/false)
 - Crear un elemento **switch** y darle el nombre de la variable (acabado en interrogación: "?"): **show-energy?**
 - En el código se puede acceder a ese valor y hacer que la variable label de las tortugas tenga un valor no nulo

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + 10)
    ]
    ifelse show-energy?
      [ set label energy ]
      [ set label "" ]
  ]
end
```

instrucción ifelse



Más procedimientos

- Para que las tortugas se muevan, coman, reproduzcan, puedan morir y pueda crecer la hierba

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick
end
```

```
to reproduce
  ask turtles [
    if energy > 50 [
      set energy energy - 50
      hatch 1 [ set energy 50 ]
    ]
  ]
end
```

Crea 1 tortuga con la energía indicada

```
to check-death
  ask turtles [
    if energy <= 0 [ die ]
  ]
end
```

elimina la tortuga

```
to regrow-grass
  ask patches [
    if random 100 < 3 [
      set pcolor green
    ]
  ]
end
```



Sacar grafos de estadísticas

- Hay que seleccionar en la interfaz para crear un “plot”
 - Indicar un nombre
 - Las variables que determinan los ejes x e y
 - Para cada línea de la gráfica hay que crear un Plot Pen con Create

Co...	Nombre del ...	Instrucciones de Actualización de Trazos		
	Tortugas	plot count turtles		
	Hierba	plot count patches with [pcolor = green]		

Sacar grafos de estadísticas

- Y hay que poner el código necesario
 - Para cada pen en su casilla
 - llamar a tick o reset-ticks en el programa

```
to setup
  clear-all
  setup-patches
  setup-turtles
  reset-ticks
end

to go
  move-turtles
  eat-grass
  check-death
  reproduce
  regrow-grass
  tick
end
```

antes en vez de tick había que implementar un método que hiciera todo esto pero ya no es necesario con la versión actual de netlogo porque lo hace automáticamente al llamar a tick para todos los plots

```
to do-plots
  set-current-plot "Totales"
  set-current-plot-pen "tortugas"
  plot count turtles
  set-current-plot-pen "hierba"
  plot count patches with [pcolor = green]
end
```

Número de pasos de la simulación

- La variable global `tick` indica el número de pasos de la simulación
 - El comando **`tick`** avanza el contador en 1
 - **`ticks`** es un reporter que indica el valor del contador de pasos
 - **`reset-ticks`** pone el contador de pasos a 0
- El siguiente código para la simulación a los 500 pasos:

```
to go
  if ticks >= 500 [ stop ]
  move-turtles
  eat-grass
  check-death
  reproduce
  regrow-grass
  tick
end
```

Configuración inicial del modelo

- Se pueden ajustar condiciones iniciales del modelo
- Por ejemplo, con un ***Slider***
 - Crear un slider con nombre *cuenta-inicial*
 - Se podrá usar en el código:

```
to setup-turtles
  create-turtles cuenta-inicial
  ask turtles [ setxy random-xcor random-ycor ]
end
```

- ***Ejercicio:***
 - Crear Sliders para configurar cuánta energía ganan las tortugas al comer o cuánta necesitan para reproducirse, y para la tasa de crecimiento de hierba

Ejercicio

- Modelo depredador-presa
 - 2 poblaciones (conejos y lobos) conviven en una región
 - Los lobos (depredadores) comen conejos (presas)
 - Los conejos viven del medio ambiente
 - El alimento para los conejos es el pasto, que crece a una tasa fija en el tiempo
- El balance entre las especies es delicado:
 - Muchas presas implica mucha comida
 - Mucha comida anima a aumentar el número de predadores
 - Más predadores necesitan más presas
 - Menos presas significan menos comida
 - Menos comida significa...

Modelo depredador-presa

- Habrá que definir dos tipos de tortugas
 - Con el comando **breed** (antes que cualquier procedimiento)
`breed [ovejas oveja] ;; el agentset y el miembro
breed [lobos lobo]`
 - Luego se pueden usar como el turtles agentset
`turtles-own [energy] ;; tanto lobos como ovejas tienen energía
lobos-own [hambre] ;; los lobos tienen hambre
set-default-shape lobos "lobo" ;; creada con el Shape Editor
create-lobos initial-number-lobos [
 set color black
 set size 1.5 ;; para verlos mejor
 set energy random (2 * ganacia-comida-lobo)
 setxy random-xcor random-ycor
]`

Más información

- En la documentación de netlogo, disponible en línea:
 - **Interface Guide** describe todos los elementos de la interfaz y su función
 - **Programming Guide** es el manual de referencia para escribir procedimientos
 - **NetLogo Dictionary** describe todas las primitivas del lenguaje

Bibliografía

- Básica:
 - N. Gilbert y K.G. Troikzsch (2005). *Simulation for the Social Scientist (2nd ed.)*. Open University Press.
- Referencias:
 - R. Axelrod (1997). *Advancing the art of simulation in the social sciences*. Complexity, 3(2):16-22.
 - B. Edmonds and S. Moss (2004). *From KISS to KIDS - An 'Anti-simplistic' Modelling Approach*. In P. Davidsson, B. Logan, and K. Takadama, editors, MABS, Lecture Notes in Computer Science 3415, Springer Verlag, 130-144.
 - A. Banos, C. Lang, N. Marilleau (2015, 2017). *Agent-based Spatial Simulation with NetLogo, Volume 1 & 2*. Elsevier.
 - Francisco J. Miguel Quesada. *Simulación Social: Una introducción*. <https://sct.uab.cat/llds/es/content/simulaci3n-social-una-introducci3n>