UNIVERSIDAD NACIONAL DE COLOMBIA

TALLER	SISTEMAS MULTIAGENTES Modelos

Profesor: Ingrid Durley Torres & Jaime Alberto Guzmán Luna

Contenido del taller:

1. Biblioteca de Modelos
2. Algoritmo de las hormigas
3. Ejemplo práctico Satisfacción de clientes
4. Revisión del ejemplo avanzado

NetLogo tiene una biblioteca, organizada con temas dónde ubica modelos de simulación, desarrollados con agentes. NetLogo incorpora una colección de modelos computacionales ya preparados para ser utilizados directamente por el usuario, que han sido verificados previamente por los gestores de la plataforma, y que pueden ser también modificados por un usuario avanzado ("Models Library").

El objetivo de la simulación con Netlogo, es economizar tiempo y recursos, permitiendo analizar muchas variables para posteriormente seleccionar las más significativas y proseguir así con la investigación. Luego el MbA, tiene grandes ventajas para el diseño de investigaciones sociales, biológicas y económicas, soportando la toma de decisiones en contextos profesionales.

INTRODUCCION

¿Cómo acceder a la biblioteca?

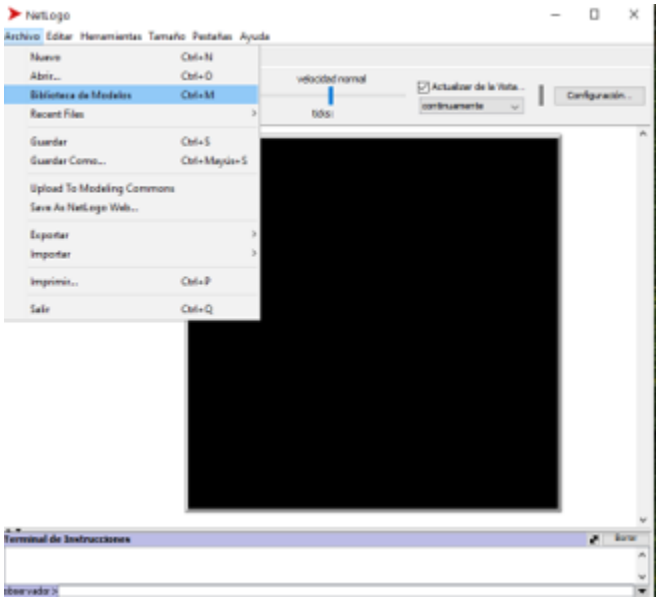


Figura 1. Interfaz biblioteca de modelos

La simulación basada en agentes es un nuevo método de investigación para las ciencias sociales y humanas, que permite explicar de manera sencilla fenómenos sociales, biológicos, políticos y económicos a través de mecanismos que hacen alusión a las acciones de los agentes y a la estructura de interacción entre ellos. Estos modelos abordan ´áreas de conocimiento muy diverso, como la biología, la medicina, la física, la química, las matemáticas (fractales, probabilidad, sistemas dinámicos),
la informática, la economía o la psicología social.

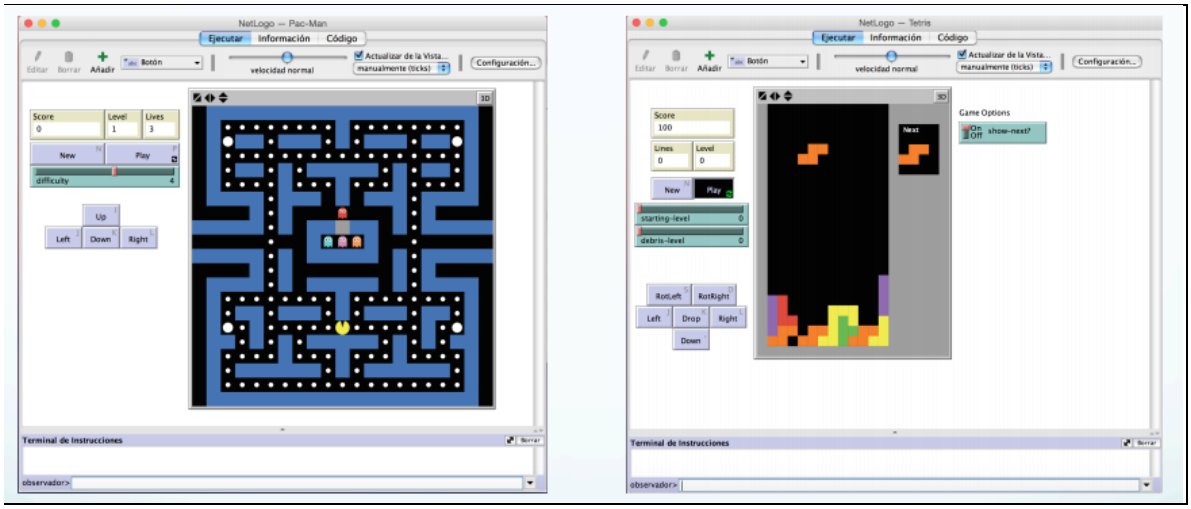


Figura 2. Pac-Man Tetris

Las instrucciones para llegar a esa ventana de entrada al modelo son: 1) abrir el programa NetLogo, 2) ir a "*File*" o archivo, 3) ir a "*Models Library*" o biblioteca de modelos, 4) escoger un área o ámbito desplegando las posibles opciones, y 5) hacer doble clic sobre el modelo escogido.

ACTIVIDAD 1. ALGORIMO COMPORTAMIENTO DE LAS HORMIGAS

las hormigas son prácticamente ciegas y, sin embargo, moviéndose prácticamente al azar, acaban encontrando el camino más corto desde su nido hasta la fuente de alimentos (y regresar). Es importante hacer algunas consideraciones:

- Por una parte, una sola hormiga no es capaz de realizar la labor anterior, sino que termina siendo un resultado del hormiguero completo, y
- lo hacen sin "instrumentos", ya que una hormiga, cuando se mueve, deja una señal química en el suelo, depositando una sustancia denominada feromona, para que las demás puedan seguirla.

Los siguientes pasos explican, de forma intuitiva, porqué la forma de proceder de las hormigas hace aparecer caminos de distancia mínima entre los nidos y las fuentes de comida:

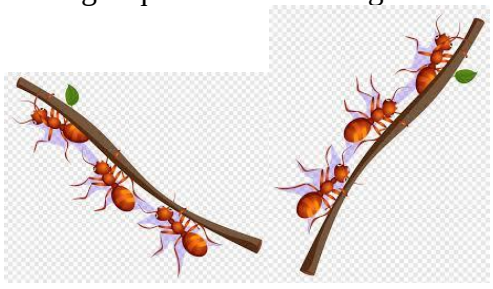
- Una hormiga (exploradora) se mueve de manera aleatoria alrededor de la colonia.

- Si esta encuentra una fuente de comida, retorna a la colonia de manera más o menos directa, dejando tras sí un rastro de feromonas.

- Estas feromonas son atractivas, las hormigas más cercanas se verán atraídas por ellas y seguirán su pista de manera más o menos directa (lo que quiere decir que a veces pueden dejar el rastro), que les lleva a la fuente de comida encontrada por la exploradora.

- Al regresar a la colonia con alimentos estas hormigas depositan más feromonas, por lo que fortalecen las rutas de conexión.

- Si existen dos rutas para llegar a la misma fuente de alimentos, en una misma cantidad de tiempo, la ruta más corta será recorrida por más hormigas que la ruta más larga.



- En consecuencia, la ruta más corta aumentará en mayor proporción la cantidad de feromonas depositadas y será más atractiva para las siguientes hormigas.

- La ruta más larga irá desapareciendo debido a que las feromonas son volátiles (evaporación).
- Finalmente, todas las hormigas habrán determinado y escogido el camino más corto.

Algoritmos de Optimización por Colonias de Hormigas (ACO)

Metodología inspirada en el comportamiento colectivo.

Se propone un camino con N lugares conectados entre sí, definidos como: $\{C_0, C_1, C_2, \ldots C_{N-1}\}$. Los N lugares forman un grafo. Y por distancia entre un nodo y otro se definirá la d_{ij} (el costo de la arista) entre los lugares C_i y C_j.

De forma explícita, los lugares pueden verse como puntos de un espacio de 2 dimensiones, y la distancia entre ellos se calcula por medio de la distancia euclídea habitual (entre los lugares-nodos): $C_i = (x_i, y_i)$ y $C_j = (x_j, y_j)$, luego el resultado será:

$$d_{i,j} = \sqrt{(x_i - x_j)^2 + (y_i - y_j)^2}$$

En esta solución las hormigas van construyendo soluciones al problema TSP (TSP: *Travelling Salesman Problem*) moviéndose por el grafo de un lugar a otro hasta que completan un ciclo. Durante cada iteración del algoritmo, cada hormiga construye su recorrido ejecutando una regla de transición probabilista que indica qué nodo debe añadir al ciclo que está construyendo. El número de iteraciones máximo que se deja correr al algoritmo depende de la decisión del usuario.

Para cada hormiga, la transición de un lugar *i* al lugar *j* en una iteración del algoritmo depende de:

1. **Si el lugar ha sido visitado o no en el ciclo que está construyendo**. Cada hormiga mantiene en memoria los lugares que ya ha visitado en el recorrido actual, y únicamente considera en cada paso los lugares que no ha visitado todavía, que denotaremos por j_i. De esta forma, aseguramos que al final la hormiga ha construido un recorrido válido.
2. **La inversa de la distancia a dicha ciudad, $v_{ij}=1/d_{ij}$, que es lo que se llama visibilidad**: Esta medida es una información local que mide, de alguna forma, la bondad de escoger c_j estando en la c_i, y puede ser usada por las hormigas para que la distancia entre ciudades consecutivas sea una característica que intervenga en la selección del recorrido que está construyendo.
3. **La cantidad de feromona que hay depositada en la arista que une ambos nodos**, que denotaremos por $T_{ij}(t)$ Esta cantidad se actualiza en cada paso, dependiendo de la cantidad de hormigas que han pasado por ella y de que el recorrido final de las hormigas que han usado esta conexión haya sido bueno (en relación con los demás caminos explorados). De alguna forma, mide la inteligencia colectiva del hormiguero, ya que es información que depende del conjunto de hormigas que están ejecutando el algoritmo.

Una vez consideradas las condiciones anteriores, la probabilidad de que la hormiga *k* vaya de c_i a c_j en la construcción del recorrido actual, viene dada por una expresión del tipo siguiente:

$$p_{ij}^{k}t = \frac{[T_{ij}t]^{\alpha}[v_{ij}]^{\beta}}{\sum l \in J_i^k\,[T_{il}t]^{\alpha}[v_{il}]^{\beta}}\,, si\ j \in J_i^k$$

$$p_{ij}^{k}t = 0, si\ j\ \notin J_i^k$$

Donde α y β son dos parámetros ajustables que controlan el peso relativo de cada una de las medidas en la heurística resultante.

- Si *α=0*, los lugares más cercanos en cada paso son las que tienen mayor probabilidad de ser seleccionadas, lo que se correspondería con el algoritmo voraz clásico en su versión estocástica (con múltiples puntos de inicio, ya que, como veremos, las hormigas se colocan inicialmente en un lugar al azar). En consecuencia, las hormigas no usan el conocimiento de la colonia para mejorar su comportamiento, que viene definido por la cantidad de feromona que hay en las aristas.
- Si *β=0* únicamente interviene la feromona, lo que experimentalmente se comprueba que puede llevar a recorridos no muy buenos y sin posibilidad de mejora.

tras completar un recorrido cada hormiga deposita una cantidad de feromona, $\Delta T_{ij}^{k}(t)$, en cada una de las aristas por las que ha pasado. Esta cantidad dependerá de lo bueno que ha sido ese recorrido en comparación con el del resto de las hormigas.
Por ejemplo, si la hormiga *k* ha realizado el recorrido $T^k(t)$, de longitud total $L^k(t)$, para cada par $(i, j) \in T^k(t)$, se puede hacer un depósito de feromona de $\Delta T_{ij}^{k}(t) = \frac{Q}{L^k\,(t)}$, donde Q es un parámetro del sistema (en la práctica, este parámetro se ajusta para que la influencia de ambas estrategias sea compensada). De esta forma, recorridos más largos tendrán menos ganancia de feromona en sus aristas, lo que disminuirá su probabilidad relativa de ser seleccionadas en etapas posteriores.

Para que este método funcione correctamente es necesario además dejar que la feromona no permanezca indefinidamente, sino que su influencia decaiga en el tiempo, de manera que aquellas aristas que no vuelvan a ser visitadas por las hormigas, y que por tanto no son reforzadas, tengan cada vez menos influencia en la heurística de decisión de cada paso. Esta disminución en el tiempo de la cantidad de feromona refleja un hecho que ocurre en la realidad, y es que la feromona usada en este tipo de procesos se evapora con una cierta tasa en cuanto es depositada, de forma que es útil únicamente si recibe un refuerzo constante en cada tramo.

Para conseguir este efecto, en los algoritmos de hormigas se introduce un nuevo parámetro, $0 \leq p\ \leq 1$, junto con una regla de actualización de feromona como sigue:

$$T_{ij}(t) \leftarrow\ (1-p)T_{ij}(t) + \sum_{k=1}^{m} \Delta T_{ij}^{k}\,(t)$$

y se supone que inicialmente en todas las aristas hay una cantidad pequeña de feromona, T_0.

El número total de hormigas que intervienen, que se denota por *m*, es otro parámetro importante a tener en cuenta:

- demasiadas hormigas tenderán rápidamente a reforzar recorridos que no son óptimos de manera que sea difícil salirse de ellos y diferenciar los buenos,
- mientras que muy pocas hormigas no provocarán el proceso de sinergia esperado debido a que no pueden contrarrestar el efecto de la evaporación de feromona, por lo que finalmente la solución que proporcionen sería equivalente al del algoritmo voraz estocástico.

Aunque no hay resultados teóricos que indiquen qué cantidad de hormigas es la óptima, una de las propuestas experimentales que más fuerza tiene es la de tomar tantas hormigas como elementos haya, es decir, $m=N$. Lo que supone un incremento lineal de los recursos.

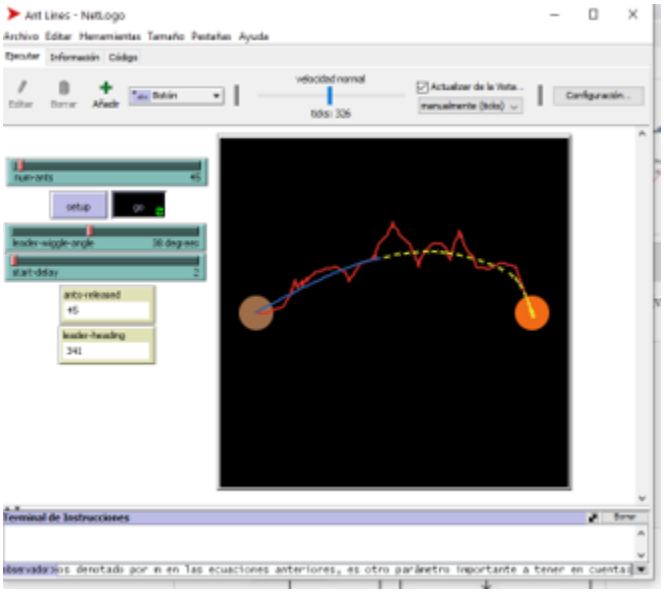


Figura 3. Ruta de ejemplo

A grandes rasgos, el algoritmo que hemos seguido y que puede servir de base para resolver otros problemas similares o hacer variantes es el siguiente: **Ants Simple**

Crear m hormigas
Repetir
  Mantener hormigas >= población
  Cada hormiga: Construir solución usando feromonas y costo de la arista
  Cada hormiga: Depositar feromonas en la arista de la solución
  Evaporar feromonas
Devolver la mejor solución encontrada

ACTIVIDAD 2. PROBLEMA IR DE COMPRAS

Crear un pequeño mundo social simulado. En esta "sociedad artificial" hay un cierto número de personas simuladas (**agentes-compradores**) y cada una de ellas tiene unos objetivos (o deseos) a cumplir en forma de su propia "lista de la compra". En el mundo hay también un cierto número de tiendas (**agentes-tiendas**), cada una de las cuales vende un producto que puede ser adquirido por los compradores. Un **agente-comprador** tiene capacidad de moverse por el "área comercial" o "mercado", pasar

por las tiendas que venden los productos y comprar lo que indica su lista de la compra. Cuando todos los agentes han conseguido su objetivo, esto es comprar todos los productos de sus listas, se consideran satisfechos y la simulación concluye.

En primer lugar, se construirá el entorno general, esto es, la configuración inicial del "mundo simulado" (un modelo muy simple de un centro comercial con diversas tiendas), y se incorporará a este mundo una primera versión, extremadamente simplificada, de los **agentes_compradores**. Tras comprobar que funcione esta simulación mínima, se irá reconstruyendo y ampliando recursivamente el modelo con agentes cada vez más complejos cognitivamente.

Para poder evaluar tales ampliaciones se establecerá un indicador de eficiencia agregada del sistema: el recuento de las unidades de tiempo que tardan todos los agentes en completar todas sus compras en un "mundo" de un tamaño determinado. Posteriormente se comparará la eficiencia de los diversos modelos, sucesivamente mejorados, observando el efecto de la incorporación de mecanismos de "inteligencia" adicional sobre la reducción del tiempo que tardan en completar todas sus compras la totalidad de los agentes.

> *Tras establecer un sistema inicial (**modelo mínimo, o modelo-0 teoría DBO-deseos, sus creencias y sus oportunidades**) se introducen sucesivas modificaciones en su diseño, derivadas de la implementación de las diversas **hipótesis que se deseen someter a contrastación**, para replicar las dinámicas sociales en esta "sociedad artificial" y así poder **comparar los resultados** con la versión de control anterior (o, **eventualmente, con resultados empíricos conocidos**).*

ACTIVIDAD 2.1 COMPRADORES TONTOS

Los agentes-compradores deambulan aleatoriamente, ciegos, sordos y mudos, encontrando tiendas y comprando en ellas hasta obtener todos los productos de su lista de la compra. No tienen capacidad de recordar a qué tienda han entrado anteriormente ni qué productos se venden en las tiendas visitadas. Tampoco son capaces de comunicarse con el resto de los compradores. Es decir, no pueden mapear el mundo, no se comunican. Sin memoria (DBO).

Única interacción: agente-comprador Vs. Agente-tienda

En este caso, el modelo supone dos tipos o clases de agentes: los compradores y las tiendas. (NetLogo permite especificar diferentes clases de objetos como "*razas/breeds*" diversas). La diferencia entre estos dos tipos de agentes está relacionada con cuestiones diversas:

- la posibilidad de *desplazarse o no* por el mundo,
- la existencia de una *memoria interna y la estructura de su contenido*, y
- en su "intención" de *conseguir* todos los productos de una lista de la compra o bien *vender* un producto concreto.

```
breed [compradores comprador ]
breed [tiendas tienda ]
```

Cada una de estas "razas" tiene sus propios atributos, esto es, espacios de memoria interna (registros, o variables) donde se almacenarán los datos que cada objeto-agente necesita guardar.

```
compradores-own [ lista-de-compra memoria]
```

Por su parte, cada tienda necesita saber cuál es el producto que vende.

```
tiendas-own [ producto ]
```

Además, es necesario definir atributos del entorno global del modelo, esto es características generales del "mundo" simulado. Mediante la declaración *globals* definiremos las variables a las que puede acceder cualquier objeto desde cualquier lugar de la simulación. Luego hasta ahora el código será:

CODIGO 1

```
breed [ compradores comprador ]
breed [ tiendas tienda ]

compradores-own [ lista-de-compra memoria ]
tiendas-own [ producto ]

globals [ productos ]
```

Ahora se debe crear el **botón Setup.**

Para inicializar nuestro modelo necesitamos llevar a cabo diversas operaciones:

- La inicialización comienza con la puesta a cero del marcador de ciclos temporales y con la eliminación de todos los objetos del mundo virtual, mediante la instrucción pertinente.

```
clear-all
reset-ticks
```

- se pedirá a todas las parcelas ponerse de color amarillo

```
ask patches [ set pcolor yellow ]
```

- A continuación, introduciremos un conjunto de datos en nuestra variable global "productos". Para introducir un conjunto de datos: set nombre-de-variable ["dato 1" "dato 2" …]

```
set productos ["aceite" "bebidas" "cerveza" "detergente"
               "especias" "flores" "gasoil" "huevos" "infusiones"
               "jamon" "kafe" "levadura"]
```

NetLogo utiliza algunas formas (shape) predefinidas para la representación gráfica de los agentes. A continuación, se asignará la forma "*person*" a todos los agentes-compradores,

- esto es, a todos los agentes de la raza compradores.

```
set-default-shape compradores "person"
create-compradores 10
```

Esta instrucción va seguida inmediatamente por una serie de instrucciones adicionales a la definición de tales agentes, entre corchetes []. Mediante diversas variantes de la instrucción set se asignarán valores (fijos o aleatorios) a los atributos propios de la raza compradores, por ejemplo:
Se establece un color (pink), una orientación geográfica (0 = "mirando hacia arriba") y un tamaño (3).

```
[
set color pink set heading 0 set size 2
```

Se coloca a cada uno de los compradores en una posición al azar "dentro" del mundo artificial; por lo que se deben restringir sus coordenadas X e Y "entre" los límites máximos de la pantalla o mundo, con la orden:

```
setxy (random world-width) + min-pxcor
      (random world-height) + min-pycor
```

- dar a cada agente valores para sus atributos específicos:
  - Para los compradores, una lista de la compra de productos a adquirir.

  A continuación, para cada agente, hay que "rellenar" su propia lista de la compra con 10 valores, extraídos de forma aleatoria de entre un conjunto formado por todos los productos disponibles. Cada lista de la compra se construye creando una lista personal de 10 productos (mediante la instrucción *n-values*), seleccionados aleatoriamente (*random*) de entre los que componen la "lista de todos los productos" almacenada en la variable global productos (algunos de ellos pueden estar duplicados en la lista de un comprador). Para conseguir esta asignación de valores concreta se utiliza la combinación de tres "funciones" predefinidas en el lenguaje NetLogo*: item* aplicado a una lista selecciona un elemento en función de su número de orden, *random* aplicado a un valor genera un número entero positivo (incluye el 0) menor que el valor de referencia, y *length* aplicado a una lista indica el número de elementos de tal lista.

```
set lista-de-compra n-values 10
[ item (random (length productos)) productos]
```

A continuación, se va a "rellenar" la memoria con el valor nulo, es decir, se vacía la memoria de cada agente:

```
set memoria [ ]
]
```

  - El conjunto de instrucciones para crear e inicializar las tiendas es:

```
set-default-shape tiendas "box"
 create-tiendas 12
 [
 set color black set heading 0 set size 2
 setxy (random world-width) + min-pxcor
 (random world-height) + min-pycor
 set producto item producto-numero productos
```

```
 set producto-numero producto-numero + 1
 set label producto set label-color red
 ]
```

Y ahora

```
end
```

Luego el código, hasta ahora el código del botón setup es:

CODIGO 2

```
to setup
  let producto-numero 0
  clear-all
  reset-ticks
  ask patches [ set pcolor yellow ]
  set productos ["aceite" "bebidas" "cerveza" "detergente"
                 "especias" "flores" "gasoil" "huevos" "infusiones"
                 "jamon" "kafe" "levadura"]
   set-default-shape compradores "person"
   create-compradores 10
   [
    set color pink set heading 0 set size 2
    setxy (random world-width) + min-pxcor
          (random world-height) + min-pycor
    set lista-de-compra n-values 10
         [ item (random (length productos)) productos ]
    set memoria [ ]
   ]


 set-default-shape tiendas "box"
 create-tiendas 12
  [
    set color black set heading 0 set size 2
    setxy (random world-width) + min-pxcor
    (random world-height) + min-pycor
    set producto item producto-numero productos
    set producto-numero producto-numero + 1
    set label producto set label-color red
  ]
End
```

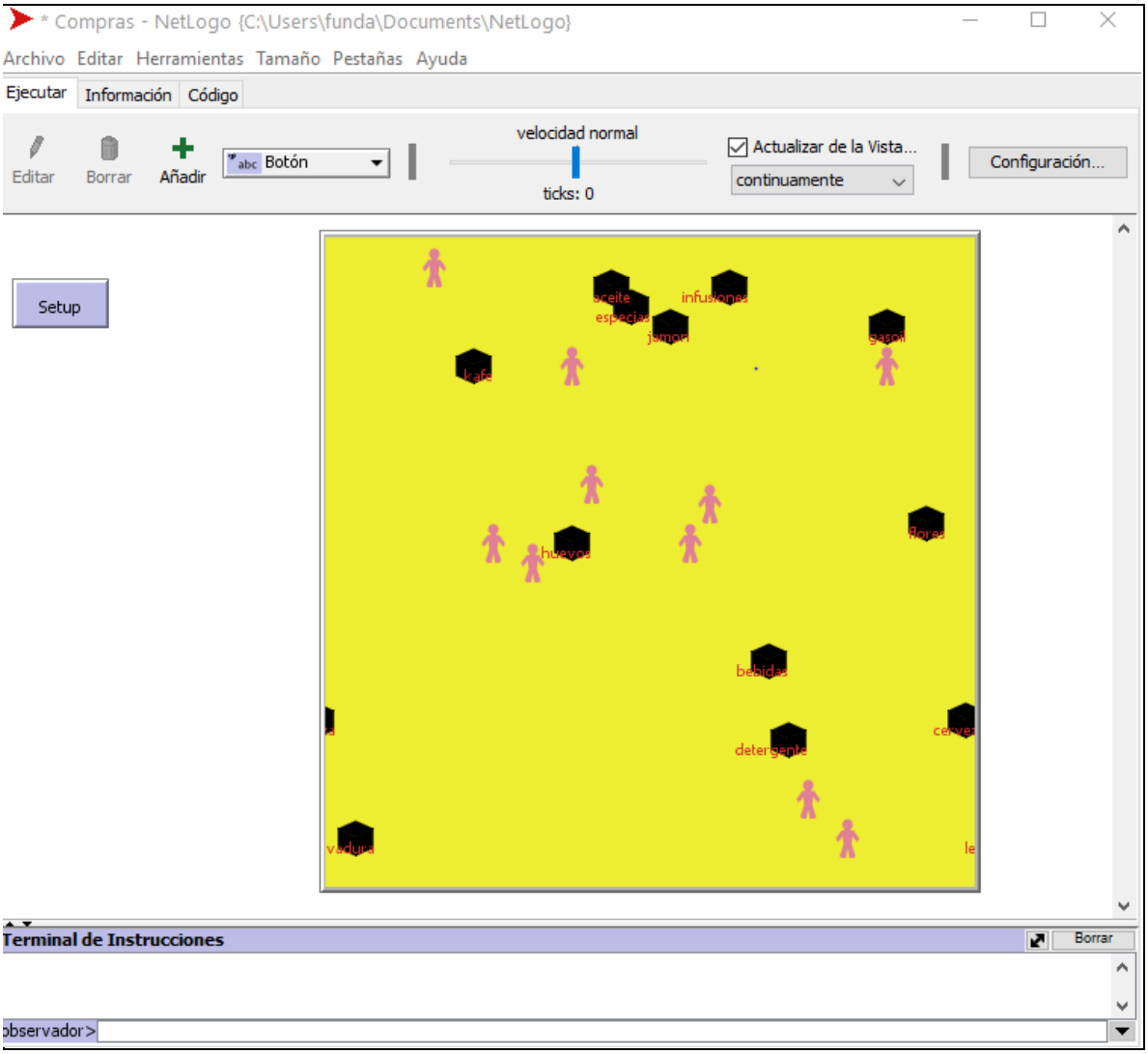


Figura 4. Pulsando el botón ejecutar

ACTIVIDAD 2.2 ACCEDIENDO A LA INFORMACIÓN DE CADA AGENTE

Una vez establecido el mundo virtual en su estado inicial y creados los agentes constituyentes de su población es posible acceder a la información "interna" de cada agente, y consultar el valor de sus atributos. Para comprobar la lista de la compra de un **agentecomprador**, en la interfaz gráfica [Ejecutar]:

- Sitúa el cursor (una flecha) sobre el comprador que se desees examinar,
- Haz clic-derecho, aparecerá un menú contextual,
- Selecciona el objeto a examinar (el cursor puede estar señalando una parcela y, eventualmente diversos agentes), por ejemplo, el "comprador 5" (los agentes aparecen en la parte inferior del menú contextual),
- Selecciona la opción "inspect" para mostrar la ventana de atributos de tal objeto.

  - Consulta el contenido de la variable lista-de-compra. Puedes hacer clic dentro de cualquier atributo y desplazar el cursor de texto.
- **Compare varios agentes**.

Modelos empíricamente calibrados (ECM): Se denominan modelos "calibrados" a aquellos constituidos por agentes cuyos atributos (en términos de distribución poblacional) modelan distribuciones de atributos observados en poblaciones reales. Por ejemplo, si se observa evidencia empírica de que la población de personas compradoras en Areas comerciales está compuesta por un 80% de género femenino y un 20% de género masculino, podríamos añadir el atributo "género" a los atributos de nuestros agentes-compradores, y "iniciar" nuestra población sintética calibrada con 8 agentes mujeres y 2 hombres.

ACTIVIDAD 2.3 CONFIRGURANDO EL GO

A continuación, crearás el procedimiento go que:

- le pedirá a cada agente que "actúe", esto es, que si aún no ha "tachado" todos los productos de su lista-de-compras busque una tienda para comprar los productos deseados, y
- a continuación, comprueba si la simulación debe finalizar. La condición de detención de la ejecución de este modelo es que todos los compradores hayan adquirido todo lo que indicaban sus respectivas listas.

En el primer caso el agente-comprador continuará ejecutando la simulación Go, usando el procedimiento comprar, definido en otro lugar del código; mientras que en el segundo caso el comprador simplemente permanecerá en su ubicación actual (marcado con un cambio al color azul) y la simulación continuará considerando el siguiente agente-comprador hasta revisarlos a todos.

Una vez descompuesto analíticamente el proceso de compra, para expresarlo como el resultado de una secuencia de procesos subyacentes, "comprar" consiste, desde el punto de vista de un agente-comprador, en:

- comprobar si hay algún agente-tienda en la misma parcela (patch) en la que se encuentra el agente-comprador actualmente, si es el caso, (por si hay más de una) elegir al azar una de tales tiendas, y "comprar" en ella, y
- después avanzar por el Centro Comercial "aleatoriamente".

```
to Go
 ask compradores [ actuar ]
   if count compradores with [ not empty? lista-de-compra ] = 0
     [ stop ]
   tick
end

to actuar
 ifelse not empty? lista-de-compra
    [ comprar ]
```

```
    [ set color sky ]
End

to comprar
 if any? tiendas-here [ compra-si-necesitas ]
 avanzar-al-azar
end

to compra-si-necesitas
 let tienda-visitada 0
 set tienda-visitada one-of tiendas-here
 if member? [producto] of tienda-visitada lista-de-compra
 [ set lista-de-compra remove [producto] of tienda-visitada lista-de-compra ]
end

to avanzar-al-azar
 set heading (random 360)
 avanzar
end
to avanzar
 forward 1
end
```

ACTIVIDAD 2.4 AÑADIR UN MONITOR DE TIEMPO

Dado que el indicador de eficiencia que se utilizará para evaluar los diferentes modelos será el "tiempo que tardan los compradores en realizar todas sus compras", es necesario añadir un control al interfaz gráfico para hacer un seguimiento de tal indicador y obtener su valor final cada vez que se ejecute una simulación del modelo. Se aprovecha la existencia de la variable predefinida global del sistema *ticks*, actualizada tras cada turno de ejecución mediante al orden *tick*, para visualizar el tiempo simulado transcurrido en un monitor.

- Añadir "monitor"
- En el apartado "Monitor" introduce el nombre de la variable a mostrar: ticks
- En el apartado "Etiqueta del Monitor" teclea, por ejemplo: Tiempo
- En el apartado "Núm. Decimales" indica: 0.

ACTIVIDAD 2.5 AÑADIR UN GRÁFICO: "EVOLUCIÓN DE COMPRADORES SATISFECHOS"

Para expresar los datos resultado de la simulación, en este caso la evolución en el tiempo del indicador seleccionado para evaluar la eficiencia del sistema (el número de compradores satisfechos), puede utilizarse un gráfico actualizado "en tiempo real" en la interfaz de usuario. Es preciso añadir este nuevo elemento, junto a los botones y el monitor ya existentes.

- Selecciona el control "Gráfico".

- En el apartado “Nombre” introduce el nombre del gráfico: Evolución de la satisfacción
- En el apartado “Etiqueta del eje X” introduce la etiqueta: Tiempo, y en “X máx.” 20000
- En el apartado “Etiqueta del eje Y” introduce la etiqueta: %, y en “Y máx.” 100
- En el apartado “Escala automática?” desactiva tal opción.

Para que se actualice el gráfico en cada momento de tiempo (simulado) hay que modificar un procedimiento que se ha escrito anteriormente: ejecutar. Se trata de introducir algunas líneas más de código.

- Desplázate a la pantalla [Código].
- Busca el procedimiento Go. Puedes desplazarte por el código con el cursor, o utilizar el menú desplegable buscador de procedimientos, situado en la barra de herramientas [Procedimientos V].
- Añade las líneas de código correspondientes, hasta que el procedimiento tenga el
- siguiente contenido:

```
to Go
 ask compradores [ actuar ]
 if Grafico? [
 plot count compradores with [ empty? lista-de-compra ] * 10
 ]
   if count compradores with [ not empty? lista-de-compra ] = 0
      [ stop ]
   tick
end
```

Finalmente, adicione el botón Interruptor y en variable global, denominar igual que como acaba de definir “Grafico?”

ACTIVIDAD 2.6 EVALUACIÓN DE LA EFICIENCIA DEL MODELO BÁSICO

El resultado agregado del conjunto de procedimientos es que los a*gentescompradore*s se dedican realizar un “paseo al azar” por toda la cuadrícula que modeliza el Centro Comercial (el “Paraiso de la Compra”). Dada una cantidad suficiente de ciclos de ejecución de la simulación, los compradores pueden llegar a pasar por todas las parcelas en que se ubican las tiendas, y desde luego transcurrirán muchos *ticks* hasta que todos ellos visiten todas las tiendas que necesitan para vaciar sus listas de la compra.

Ahora que el código esta completo, en sus aspectos fundamentales, ya puede inicializarse y ejecutarse tantas veces como se desee. Debido a la naturaleza fundamentalmente azarosa de los movimientos de los agentes, así como a las diferentes posiciones iniciales de compradores y tiendas, y la diversidad en las lista de la compra, el tiempo que tarda cada simulación en completarse variará con cada ejecución concreta.

Comprueba esto, haciendo clic en los botones correspondientes de la interfaz gráfica para inicializar el mundo y ejecutar varias simulaciones del modelo. Consulta, en el monitor, el tiempo final de cada una de ellas.

- Apunta el resultado obtenido ("Tiempo") para cada simulación realizada,
- Inicializar el modelo y replicar simulaciones un cierto número de veces (al menos 10),
- Calcula la media aritmética de Tiempo hasta que el sistema social simulado alcanza el punto de satisfacción.
- Has usado la metodología de Simulación Social como técnica de replicación de experimentos.

En una serie de 100 ejecuciones se obtuvo un valor tick final, cuando todos los agentes han comprado todo lo que necesitan, promedio de 14.310 (con desviación estándar de 4.150). Los agentes de esta primera versión del modelo son altamente ineficientes en cuanto a la localización de las tiendas que necesitan debido a su comportamiento sordo, ciego y mudo, basado en recorrer al azar el centro comercial. En definitiva, el modelo simula las actividades de un conjunto de agentes fundamentalmente "asociales".

Referencia

Quesada Miguel Francisco J. "Construyendo Simulaciones Sociales Multi-agente: Tutorial de auto-construcción, paso a paso, con NetLogo 6" LSDS-UAB - Dept. Sociologia UAB (v2.2, Septiembre 2017)