Publicado en Septiembre de 2018. Este documento usa la versión 6.0 de NetLogo. Traducción por Haroldo Miranda.

GUIA DE PROGRAMACION DE NETLOGO

AGENTES

El mundo de NetLogo está compuesto de agentes. Los agentes son seres que pueden seguir instrucciones.

En NetLogo, hay cuatro tipos de agentes: tortugas, parcelas, enlaces y el observador.

Las tortugas son agentes que se mueven por el mundo. El mundo es bidimensional y se divide en una grilla de parcelas. Cada parcela es una pieza cuadrada de "tierra" sobre la cual las tortugas se pueden mover. Los enlaces son agentes que conectan dos tortugas. El observador no tiene una ubicación; se lo puede imaginar mirando el mundo de las tortugas y parcelas.

El observador no observa pasivamente: da instrucciones a los otros agentes.

Cuando se inicia NetLogo, no hay tortugas. El observador puede hacer nuevas tortugas. Las parcelas también pueden formar nuevas tortugas. (Las parcelas no se pueden mover, pero están tan "vivas" como las tortugas).

Las parcelas tienen coordenadas. La parcela en las coordenadas (0, 0) se llama origen y las coordenadas de las otras parcelas son las distancias horizontal y vertical desde ésta. Llamamos a las coordenadas de la parcela `pxcor` y `pycor`. Al igual que en el plano de coordenadas matemáticas estándar, `pxcor` aumenta a medida que se avanza hacia la derecha y la `pycor` aumenta a medida que se mueve hacia arriba.

El número total de parcelas está determinado por los ajustes de coordenadas `min-pxcor`, `max-pxcor`, `min-pycor` y `max-pycor`. Cuando se inicia NetLogo, `min-pxcor`, `max-pxcor`, `min-pycor` y `max-pycor` son -16, 16, -16 y 16, respectivamente. Esto significa que `pxcor` y `pycor` varían entre -16 y 16, por lo que hay 33 veces 33, o 1089 parcelas en total. (Se puede cambiar el número de parcelas con el botón de configuración o `setup`).

Las tortugas también tienen coordenadas: `xcor` e `ycor`. Las coordenadas de una parcela son siempre números enteros, pero las coordenadas de una tortuga pueden tener decimales. Esto significa que una tortuga puede colocarse en cualquier punto dentro de su parcela; no tiene que estar en el centro de la parcela.

Los enlaces no tienen coordenadas. Cada enlace tiene dos extremos, y cada extremo es una tortuga. Si cualquiera de las tortugas muere, el enlace también muere. Un enlace se representa visualmente como una línea que conecta las dos tortugas.

PROCEDIMIENTOS

En NetLogo, las instrucciones y reporteros les dicen a los agentes qué hacer. Una instrucción es una acción que un agente debe llevar a cabo, lo que resulta en algún efecto. Un reportero provee instrucciones para calcular un valor, que luego el agente "informa" a quien lo solicitó.

Normalmente, un nombre de instrucción comienza con un verbo, como "crear", "morir", "saltar", "inspeccionar" o "borrar". La mayoría de los nombres de los reporteros son sustantivos o frases nominales.

Las instrucciones y reporteros integrados en NetLogo se llaman primitivas. El diccionario de NetLogo tiene una lista completa de instrucciones y reporteros integrados.

Las instrucciones y reporteros que usted define se llaman procedimientos. Cada procedimiento tiene un nombre, precedido por la palabra clave `to` o `to-report`, dependiendo de si se trata de un procedimiento de instrucciones o un procedimiento de reportero. La palabra clave `end` es el final de las instrucciones en el procedimiento. Una vez que define un procedimiento, puede usarlo en cualquier otro lugar de su programa.

Muchas instrucciones y reporteros toman entradas, valores que la instrucción o reportero usa para llevar a cabo sus acciones o calcular su resultado.

Aquí hay dos procedimientos de instrucción:

```
to setup ;;para configurar
  clear-all ;;limpiar todo
  create-turtles 10 ;;crea 10 tortugas
  reset-ticks ;;redefinir pasos (de tiempo)
end ;; fin del procedimiento
```

```
to go ;; para ejecutar (correr el procedimiento)
  ask turtles [ ;; pedir a las tortugas
    fd 1        ;; avanzar 1 paso
    rt random 10 ;; girar a la derecha aleatoriamente entre 0 y 9
    lt random 10    ;; girar a la izquierda aleatoriamente entre 0 y 9
  ]
  tick
end
```

Tenga en cuenta el uso de punto y coma para agregar "comentarios" al programa. Los comentarios pueden hacer que su código sea más fácil de leer y comprender, pero no afectan su comportamiento.

En este programa, `setup` (configurar) and `go` (ir o ejecutar la aplicación) son instrucciones definidas por el usuario.

`clear-all` (limpiar todo), `create-turtles` (crear tortugas), `reset-ticks` (re-establecer pasos o tiempo), `ask` (pedir), `lt` ("giro a la izquierda" que en inglés es left turn), `rt` ("giro a la derecha", que en inglés es right turn) y `tick` (pasos en el tiempo), son todas instrucciones primitivas.

La expresión `random` (aleatorio) y `turtles` (tortugas) son reporteros primitivos. Al azar toma un solo número como entrada e informa un entero aleatorio que es menor que la entrada (en este caso, entre 0 y 9). `turtles` informa el conjunto de agentes que consiste en todas las tortugas. (Explicaremos sobre `agent sets` más tarde).

`setup` and `go` puede ser llamado por otros procedimientos, o por botones, o desde el Terminal de Instrucciones.

Muchos modelos de NetLogo tienen un botón que llama un procedimiento llamado configuración (`setup`) y un botón "continuamente" ("para siempre") que llama a un procedimiento llamado ir o ejecutar (`go`). Un botón "continuamente" significa que cuando se corre el modelo, la ejecución es continua, como si fuese de largo plazo y se verá el resultado después de muchos pasos. Como complemento, existe un botón `once` ("de una vez"), con el cual se puede ver qué va ocurriendo paso a paso.

En NetLogo, se puede especificar qué agentes (tortugas, parcelas o enlaces) deben ejecutar cada instrucción. Si no se especifica, el código es ejecutado por el observador. En el código anterior, el observador usa `ask` para hacer (en realidad, pedir, solicitar) que el conjunto de todas las tortugas ejecute las instrucciones entre los corchetes.

`clear-all` y `create-turtles` solo pueden ser ejecutadas por el observador. La instrucción `fd`, por otro lado, solo puede ser ejecutada por tortugas. Algunos otras instrucciones y reporteros, como `set` y `ticks`, pueden ser ejecutadas por diferentes tipos de agentes.

Aquí hay algunas características más avanzadas que puede aprovechar al definir sus propios procedimientos.

PROCEDIMIENTOS CON INPUTS

Los procedimientos pueden aceptar entradas, al igual que muchas primitivas. Para crear un procedimiento que acepte entradas, se debe colocar sus nombres entre corchetes después del nombre del procedimiento. Por ejemplo:

```
to draw-polygon [num-sides len]  ;; turtle procedure
  pen-down
  repeat num-sides [
    fd len
    rt 360 / num-sides
  ]
end
```

En otra parte del programa, puede usar el procedimiento pidiendo a las tortugas que dibujen un octágono con una longitud lateral igual a su número identificador `who`:

```
ask turtles [ draw-polygon 8 who ]
```

Procedimientos de Reportero

Al igual que se puede definir instrucciones propias, también se pueden definir informadores propios. Deben hacerse dos cosas especiales. Primero, usar `to-report` en lugar de `to,` para comenzar el procedimiento. Luego, en el cuerpo del procedimiento, use `report` para informar el valor que se desea reportar. Veamos un ejemplo:

```
to-report absolute-value [number]
  ifelse number >= 0
    [ report number ]
    [ report (- number) ]
end
```

En este caso, la condición (`ifelse`) instruye que, si el número es mayor o igual que cero, entonces reportar el número en valor absoluto, independiente de su signo.

Variables

Variables de Agentes

Las variables de agente son contenedores para almacenar valores (como números) en un agente. Una variable de agente puede ser una variable global, una variable de tortuga, una variable de parcela o una variable de enlace.

Si una variable es de carácter global, solo hay un valor para la variable y cada agente puede acceder a ella. Es decir, la variable global es accesible por cualquier agente. Puede pensar en variables globales como pertenecientes al observador.

Las variables de tortuga, parcela y enlace son diferentes. Cada tortuga tiene su propio valor para cada variable de tortuga. Lo mismo aplica para parcelas y enlaces. Por ejemplo, la variable ubicación de una tortuga específica es única en un momento determinado y su cambio de valor también.

Algunas variables ya están integradas en NetLogo. Por ejemplo, todas las tortugas y enlaces tienen una variable `color`, y todas las parcelas tienen una variable `pcolor`. (La variable de parcela comienza con "p" para que no se confunda con la variable de tortuga, ya que las tortugas tienen acceso directo a las variables de parcela). Si se define la variable, la tortuga o la parcela cambian de color. (Consulte la siguiente sección para más detalles).

Otras variables ya integradas en el lenguaje de programación son `xcor`, `ycor` y `heading`. Otras variables de parcela integradas incluyen `pxcor` y `pycor`. (Puede revisar una lista completa en el Diccionario, sección Variables Incorporadas).

También se pueden definir variables propias, como en otros lenguajes de programación. Se puede crear una variable global agregando un interruptor, una barra deslizadora, un selector o cuadro de entrada al modelo, o usando la palabra clave `global` al principio de su código, como en el siguiente ejemplo:

```
globals [score] ;; variable global denominada puntaje o score
```

También se pueden definir nuevas variables de tortuga, parcela y enlace utilizando las palabras clave `turtles-own`, `patches-own` y `links-own`, que corresponden a variables definidas por el constructor del modelo, como en el siguiente ejemplo:

```
turtles-own [energy speed] ;; variable energía y velocidad
patches-own [friction] ;; variable fricción
links-own [strength] ;; variable fuerza, resistencia, intensidad
```

Estas variables se pueden usar libremente en un modelo. Use la instrucción `set` para configurarlos. (Cualquier variable a la que no se le establezca un valor, tiene un valor inicial de cero).

Las variables globales pueden leerse y establecerse en cualquier momento por cualquier agente. Además, una tortuga puede leer y definir variables de parcela dentro de la parcela en la que se encuentra. Por ejemplo, el siguiente código hace que cada tortuga haga que la parcela se coloque en rojo. (***Debido a que las tortugas comparten las variables de parcela de esta manera, no se puede tener una variable de tortuga y una variable de parcela con el mismo nombre***).

```
ask turtles [ set pcolor red ] ;; pide a tortugas que la parcela sea roja
```

En otras situaciones en las que se desee que un agente lea la variable de un agente diferente, se puede usar la expresión `of`. Ejemplo:

```
show [color] of turtle 5 ;; mostrar el color de la tortuga número 5
;; imprime el color actual de la tortuga que tiene el número 5
```

También se puede usar la expresión `of` en una declaración más compleja como un solo nombre de variable, por ejemplo:

```
show [xcor + ycor] of turtle 5
;; imprime la suma de las coordenadas x e y
;; de la tortuga que tiene el número 5
```

Variables Locales

Una variable local se define y usa solo en el contexto de un procedimiento particular o parte de un procedimiento. Para crear una variable local, use la instrucción `let`. Si usa `let` en la parte superior de un procedimiento, la variable existirá durante todo el procedimiento. Si lo usa dentro de un conjunto de corchetes cuadrados, por ejemplo, dentro de un "`ask`", entonces existirá solo dentro de esos corchetes.

```
to swap-colors [turtle1 turtle2]
  let temp [color] of turtle1
  ask turtle1 [ set color [color] of turtle2 ]
  ask turtle2 [ set color temp ]
end
```

La declaración del procedimiento señala que para intercambiar colores entre la tortuga1 y la tortuga2, mantenga temporalmente el color de la tortuga1; luego pida a la tortuga1 que defina un color para la tortuga2 y, pídale a la tortuga2 que establezca ese color temporalmente.

Contador de Pasos o Momentos

En muchos modelos de NetLogo, el tiempo pasa en pasos discretos, llamados "ticks". NetLogo incluye un contador de ticks incorporado para que pueda realizar un seguimiento de la cantidad de ticks que han pasado.

El valor actual del contador de ticks se muestra arriba de la vista. (Puede usar el botón `Configuración` para ocultar el contador de ticks o cambiar la palabra "ticks" por otra cosa).

En el código, para recuperar el valor actual del contador de ticks, use el reportero o informador `ticks`. La instrucción `tick` avanza el contador de ticks en 1. La instrucción `clear-all` borra el contador de ticks junto con todo lo demás.

Cuando el contador de ticks está limpio o despejado, **no intente leerlo o modificarlo**. Use la instrucción `reset-ticks` cuando haya terminado de configurar su modelo, para iniciar el contador de pasos.

Si su modelo está configurado para usar actualizaciones basadas en ticks, entonces la instrucción `tick` generalmente también actualizará la vista. Consulte la sección posterior, Ver actualizaciones.

Cuando Marcar los Pasos

Use `reset-ticks` al final de su procedimiento de configuración.

Use `tick` al final de un procedimiento `go`.

```
to setup
  clear-all
  create-turtles 10
  reset-ticks ;; aquí se resetea ticks
end

to go
  ask turtles [ fd 1 ]
  tick ;; aquí se inician los pasos
end
```

Pasos Fraccionados

En la mayoría de los modelos, el contador de ticks comienza en 0 y se incrementa de 1 en 1, de entero a entero. Pero también es posible que el contador de ticks tome valores de punto flotante entre cada paso, vale decir, con decimales.

Para avanzar el contador de ticks en una cantidad fraccionaria, utilice la instrucción `tick-advance`. Esta instrucción toma una entrada numérica que especifica cuánto avanzará el contador de ticks.

Un uso típico de los pasos fraccionados es aproximar el movimiento continuo o curvo. Ver, por ejemplo, los modelos de GasLab en la Biblioteca de Modelos (en Chemistry & Physics). Estos modelos calculan la hora exacta en que se producirá un evento futuro, luego avanzan el contador de ticks exactamente en ese momento.

La importancia de distinguir los ticks enteros de los fraccionados se encuentra en los tipos de distribuciones estadísticas. Algunos fenómenos se comportan de modo discreto, para lo cual las distribuciones pertinentes son de tipo discreto; mientras que para otros fenómenos cuyas distribuciones son continuas, entonces lo pertinente es utilizar ticks fraccionados que recojan ese comportamiento continuo.

Colores

NetLogo representa los colores de diferentes maneras. Un color puede ser un número en el rango de 0 a 140, con la excepción del propio 140. A continuación, se muestra un cuadro que muestra el rango de dichos colores que puede usar en NetLogo.

	black = 0										white = 9.9
gray = 5	0	1	2	3	4	5	6	7	8	9	9.9
red = 15	10	11	12	13	14	15	16	17	18	19	19.9
orange = 25	20	21	22	23	24	25	26	27	28	29	29.9
brown = 35	30	31	32	33	34	35	36	37	38	39	39.9
yellow = 45	40	41	42	43	44	45	46	47	48	49	49.9
green = 55	50	51	52	53	54	55	56	57	58	59	59.9
lime = 65	60	61	62	63	64	65	66	67	68	69	69.9
turquoise = 75	70	71	72	73	74	75	76	77	78	79	79.9
cyan = 85	80	81	82	83	84	85	86	87	88	89	89.9
sky = 95	90	91	92	93	94	95	96	97	98	99	99.9
blue = 105	100	101	102	103	104	105	106	107	108	109	109.9
violet = 115	110	111	112	113	114	115	116	117	118	119	119.9
magenta = 125	120	121	122	123	124	125	126	127	128	129	129.9
pink = 135	130	131	132	133	134	135	136	137	138	139	139.9

El cuadro muestra que:

- Algunos de los colores tienen nombres. (Se pueden usar estos nombres en el código).
- Cada color con nombre, excepto blanco y negro, tiene un número que termina en 5.
- A cada lado de cada color nombrado, hay tonos más oscuros y más claros del color.
- 0 es negro puro 9.9 es blanco puro
- 10, 20, y así sucesivamente son todos tan oscuros que son casi negros.
- 19.9, 29.9 y así sucesivamente son todos tan ligeros que son casi blancos.

Código de Ejemplo: la tabla de colores se creó en NetLogo con la Carta de Colores del Modelo Color Chart Example.

Si se utiliza un número fuera del rango de 0 a 140, NetLogo repetidamente agregará o restará 140 del número hasta que esté en el rango de 0 a 140. Por ejemplo, 25 es naranja, por lo que 165, 305, 445, etc. también son de color naranja, igual que -115, -255, -395, etc. Este cálculo se realiza automáticamente siempre que establezca la variable `color` de tortuga o la variable `pcolor` de una parcela. Si necesita realizar este cálculo en algún otro contexto, use la primitiva de ajuste del color `wrap-color`.

Si se desea un color que no esté en el gráfico, existen más entre los enteros. Por ejemplo, 26.5 es un tono naranja a medio camino entre 26 y 27. Esto no significa que pueda hacer cualquier color en NetLogo; el espacio de color NetLogo es solo un subconjunto de todos los colores posibles. Contiene solo un conjunto fijo de matices discretos (un matiz por fila de la tabla o carta de colores). A partir de uno de esos tonos, puede disminuir su brillo (oscurecerlo) o disminuir su saturación (aclararlo), pero no puede disminuir tanto el brillo como la saturación. Además, solo el primer dígito después del punto decimal es significativo. Por lo tanto, los valores de color se redondean hacia abajo al siguiente 0.1, por lo que, por ejemplo, no hay diferencia visible entre 26.5 y 26.52 o 26.58.

Funciones Primitivas de Color

Hay unas pocas primitivas que son útiles para trabajar con colores.

Ya hemos mencionado la primitiva de envoltura de color `wrap-color`.

La primitiva `scale-color` es útil para convertir datos numéricos en colores.

`shade-of?` indicará si dos colores son ambos "tonos" del mismo tono básico. Por ejemplo, `shade-of? orange 27` es cierto (`true`), porque 27 es un tono más claro de naranja.

Ejemplo de código: El Ejemplo Scale-color Example demuestra el informador de color de escala.

Colores RGB y RGBA

NetLogo también representa colores como listas RGB (red / green / blue) y listas RGBA (red / green / blue / alfa). Al usar colores RGB, la gama completa de colores está disponible. Los colores RGBA permiten todos los colores que RGB permite y también puede variar la transparencia u opacidad de un color. Las listas RGB y RGBA se componen de tres o cuatro enteros, respectivamente, entre 0 y 255. Si un número está fuera de ese rango, 255 se resta repetidamente hasta que se encuentre en el rango. Puede establecer cualquier variable de color en NetLogo (`color` para tortugas y enlaces y `pcolor` para parcelas) en una lista RGB y ese agente se renderizará y representará adecuadamente. Entonces puede establecer el color de la parcela 0 0 en rojo puro usando el siguiente código:

```
set pcolor [255 0 0] ;; define colores en formato RGB
```

Las tortugas, los enlaces y las etiquetas pueden contener listas RGBA como variables de color, sin embargo, las parcelas no pueden tener colores RGBA, es decir no pueden tener opacidad o transparencia. Puede configurar el color de una tortuga para que sea de aproximadamente medio rojo puro transparente con el siguiente código:

```
set color [255 0 0 125]
```

Se puede convertir de un color NetLogo a RGB o HSB (hue / saturation / bright) utilizando `extract-hsb` y `extract-rgb`. Puede usar `rgb` para generar listas rgb y `hsb` para convertir de un color HSB a RGB.

Debido a que faltan muchos colores en el espacio de color de NetLogo, `approximate-hsb` y `approximate-rgb` no podrán darle el color exacto que usted solicite, pero intentan acercarse lo más posible.

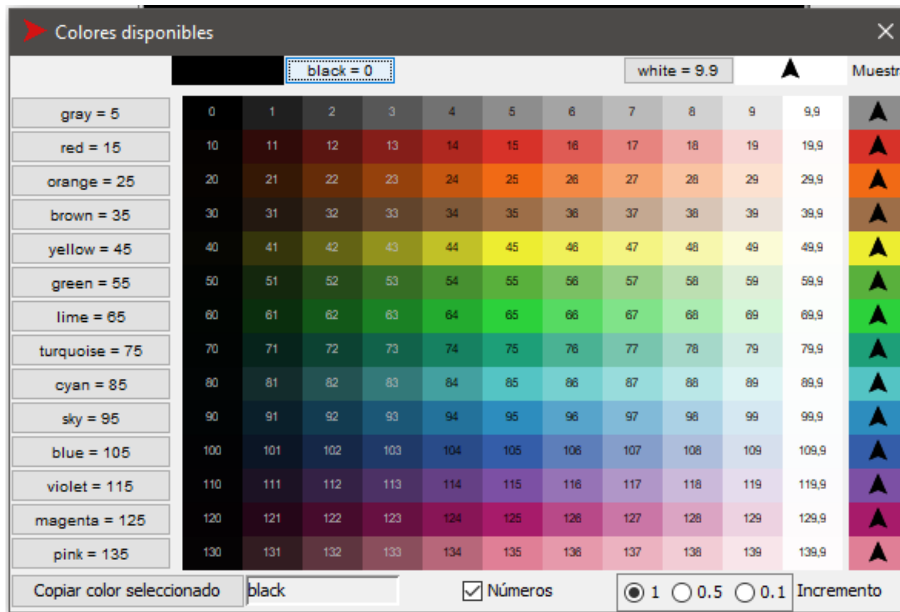
Ejemplo: puede cambiar cualquier color existente de NetLogo, en una tortuga, a una versión medio transparente de ese color usando:

```
set color lput 125 extract-rgb color
```

Ejemplos de código: HSB and RGB Example (le permite experimentar con los sistemas de color HSB y RGB), y también el modelo Transparency Example.

Cuadro de Diálogo de Muestras de Color

El cuadro de diálogo Muestras de color lo ayuda a experimentar y elegir colores. Ábralo eligiendo Muestras de Colores Disponibles en el menú Herramientas.



Al hacer clic en una muestra de color (o en un botón de color), ese color se mostrará con otros colores. En la parte inferior izquierda, se muestra el código del color seleccionado actualmente (por ejemplo, rojo + 2) para que pueda copiarlo y pegarlo en su código. En la parte inferior derecha hay tres opciones de incremento, 1, 0.5 y 0.1. Estos números indican la diferencia entre dos muestras adyacentes. Cuando el incremento es 1, hay 10 tonos diferentes en cada fila; cuando el incremento es 0.1, hay 100 matices diferentes en cada fila. 0.5 es una configuración intermedia.

Instrucción Ask

NetLogo usa la instrucción `ask` para aplicarlo a tortugas, parcelas y enlaces. Todo el código que deben ejecutar las tortugas debe ubicarse en un "contexto" de tortuga. Puede establecer un contexto de tortuga de tres maneras:

- En un botón, seleccionando "Tortugas" en el menú emergente. Cualquier código que ponga en el botón será ejecutado por todas las tortugas.
- En el Terminal de Instrucciones, seleccionando "Tortugas" en el menú emergente. Cualquier instrucción que ingrese será ejecutada por todas las tortugas.
- Al usar `ask turtles`, `hatch` u otras instrucciones que establecen un contexto de tortuga.

Lo mismo ocurre con las parcelas, enlaces y el observador, excepto que no se puede preguntar al observador. Cualquier código que no esté dentro de algún `ask`, por defecto es código de observador.

Aquí hay un ejemplo del uso de `ask` en un procedimiento de NetLogo:

```
to setup
  clear-all
  create-turtles 100   ;; crea 100 turtles con orientación aleatoria
  ask turtles
    [ set color red    ;; las cambia a color rojo
      fd 50 ]          ;; las extiende alrededor
  ask patches
    [ if pxcor > 0          ;; parcelas en el lado derecho
        [ set pcolor green ] ]  ;; define de color verde a las parcelas
  reset-ticks ;; resetea ticks
end
```

La Biblioteca de Modelos está llena de otros ejemplos. Un buen lugar para comenzar a buscar es la sección de Code Examples.

Por lo general, el observador usa `ask` para pedirle a todas las tortugas, a todas las parcelas o a todos los enlaces que ejecuten instrucciones. También puede usar `ask` para correr las instrucciones sobre una tortuga individual, parcela o enlace. Los reporteros `turtle`, `patch`, `link` y `patch-at` son útiles para esta técnica. Por ejemplo:

```
to setup
  clear-all
  crt 3                      ;; haga 3 turtles
  ask turtle 0               ;; dígale a la primera...
    [ fd 1 ]                 ;; ...que vaya hacia adelante
  ask turtle 1               ;; dígale a la segunda...
    [ set color green ]      ;; ...que se convierta al color verde
  ask turtle 2               ;; dígale a la tercera...
    [ rt 90 ]                ;; ...que gire a la derecha
  ask patch 2 -2             ;; pídale a la parcela en (2,-2)
    [ set pcolor blue ]      ;; ...que se convierta al color azul
  ask turtle 0               ;; pídale a la primera turtle
    [ ask patch-at 1 0 ;; ...que le pida a la parcela que está en el este
      [ set pcolor red ] ]         ;; ...que se convierta al color rojo
  ask turtle 0                     ;; dígale a la primera turtle...
    [ create-link-with turtle 1 ] ;; ...que se conecte con la segunda
  ask link 0 1                    ;; pídale al enlace entre la turtle 0 y 1
    [ set color blue ]             ;; ...que se convierta al color azul
  reset-ticks
end
```

Cada tortuga creada tiene un número que la identifica (`who`). La primera tortuga creada es el número 0, la segunda tortuga número 1, y así sucesivamente. Nótese que se trata de una notación matricial. En las matrices o arrays de programación, el primer elemento de la matriz o array siempre es 0.

La primitiva reportera de `turtle` toma el número como una entrada o input e informa a la tortuga con ese número para que sepa quién es (`who`). El reportero primitivo `patch` toma los valores de las coordenadas x,y (`pxcor` y `pycor`) e informa a la parcela de esas coordenadas. La primitiva `link` toma las dos entradas, es decir, los números correspondientes de las dos tortugas y las conecta. Y el reportero primitivo `patch-at` toma los desplazamientos: distancias, en las direcciones x e y, del primer agente. En el ejemplo anterior, se le pide a la tortuga con identidad (`who`) número 0 que obtenga la parcela al este (y no parcelas al norte) de sí misma.

También se puede seleccionar un subconjunto de tortugas, un subconjunto de parcelas o un subconjunto de enlaces y pedirles que hagan algo. Esto implica usar **`agentsets`**. La siguiente sección los explica en detalle.

Cuando se le pide a un conjunto de agentes que ejecute más de una instrucción, cada agente debe terminar antes de que comience el próximo agente. Un agente ejecuta todas las instrucciones, luego el siguiente agente ejecuta todas ellas, y así sucesivamente. Por ejemplo, si escribe:

```
ask turtles
  [ fd 1
    set color red ]
```

Entonces, primero una tortuga se mueve y se vuelve roja, luego otra tortuga se mueve y se pone roja, y así sucesivamente.

Pero si usted escribiera el código de otra forma, como, por ejemplo:

```
ask turtles [ fd 1 ]
ask turtles [ set color red ]
```

La acción sería que, primero todas las tortugas se mueven, luego todas se ponen rojas.

Conjuntos de Agentes (Agentsets)

Un agentsets es exactamente lo que su nombre indica, un conjunto de agentes. Un conjunto de agentes puede contener tortugas, parcelas o enlaces, pero no más de un tipo a la vez.

Un conjunto de agentes no está en un orden particular. De hecho, siempre está en un orden aleatorio. Y cada vez que lo utiliza, el conjunto de agentes está en un orden aleatorio diferente. Esto ayuda a evitar que su modelo trate a las tortugas, parcelas o enlaces en particular de forma diferente a los demás (a menos que usted lo desee). Dado que el orden es aleatorio todo el tiempo, ningún agente tiene que ir primero siempre.

Ya hemos visto que la primitiva `turtles` informa al conjunto de agentes de todas las tortugas, la primitiva `patches` informa al conjunto de agentes de todas las parcelas y la primitiva `links` informa al conjunto de agentes de todos los enlaces.

Pero lo que es poderoso acerca del concepto de conjunto de agentes es que se pueden construir grupos de agentes que contienen solo *algunas* tortugas, *algunas* parcelas o *algunos* enlaces. Por ejemplo, todas las tortugas rojas, o las parcelas con pxcor que pueden dividirse por cinco, o las tortugas en el primer cuadrante que están en una parcela verde o los enlaces conectados a la tortuga 0. Estos conjuntos de agentes pueden ser utilizados por `ask` o por varios reporteros que toman a los agentes como entradas o inputs.

Una forma es usar `turtles-here` o `turtles-at`, para hacer un conjunto de agentes que contenga solo las tortugas en mi parcela, o solo las tortugas dentro de otra parcela y sólo en algunos desplazamientos x e y. También está `turtles-on` para que pueda mantener un conjunto de tortugas dentro de o sobre una parcela determinada o un conjunto de parcelas, o el conjunto de tortugas manteniéndose en la misma parcela como una entidad determinada o un conjunto determinado.

Aquí hay algunos ejemplos más de cómo crear agentsets:

```
;; all other turtles:
other turtles
;; all other turtles on this patch:
other turtles-here
;; all red turtles:
turtles with [color = red]
;; all red turtles on my patch
turtles-here with [color = red]
;; patches on right side of view
patches with [pxcor > 0]
;; all turtles less than 3 patches away
turtles in-radius 3
;; the four patches to the east, north, west, and south
patches at-points [[1 0] [0 1] [-1 0] [0 -1]]
;; shorthand for those four patches
neighbors4
;; turtles in the first quadrant that are on a green patch
turtles with [(xcor > 0) and (ycor > 0)
              and (pcolor = green)]
;; turtles standing on my neighboring four patches
turtles-on neighbors4
;; all the links connected to turtle 0
[my-links] of turtle 0
```

Nótese el uso de `other` para excluir a este agente. Esto es común.

Una vez que haya creado un conjunto de agentes, aquí hay algunas cosas simples que puede hacer:

- Use `ask` para hacer que los agentes dentro del conjunto de agentes hagan algo
- Use `any?` para ver si el agente está vacío
- Use `all?` para ver si cada agente en un conjunto de agentes cumple una condición.
- Use `count` para saber exactamente cuántos agentes hay en el conjunto

Y aquí hay algunas cosas más complejas que puede hacer:

- Elija un agente aleatorio del conjunto usando `one-of`. Por ejemplo, podemos hacer que una tortuga elegida al azar se vuelva verde:

```
ask one-of turtles [ set color green ]
```

- dígale a una parcela elegida al azar que germine una nueva tortuga:

```
ask one-of patches [ sprout 1 ]
```

- Utilice los reporteros o informadores `max-one-of` o `min-one-of` para averiguar qué agente es el más (de algo) o el menos (de algo) a lo largo de una escala. Por ejemplo, para eliminar la tortuga más rica, se podría decir:

```
ask max-one-of turtles [sum assets] [ die ]
```

- Haga un histograma del conjunto de agentes usando la instrucción `histogram` (en combinación con `of`).
- Utilice `of` para hacer una lista de valores, uno por cada agente dentro del conjunto de agentes. Luego use una de las primitivas de lista de NetLogo para hacer algo con la lista. (Consulte la sección "Listas" a continuación.) Por ejemplo, para averiguar qué tan rico es el promedio de tortugas, podría decir:

```
show mean [sum assets] of turtles
```

- Use `turtle-set`, `patch-set` y `link-set` para crear nuevos grupos de agentes reuniendo agentes de una variedad de fuentes posibles.
- Use `no-turtles`, `no-patches` y `no-links` para crear grupos de agentes vacíos.
- Compruebe si dos grupos de agentsets son iguales usando `=`, o bien, `!=`.
- Use `member?` para ver si un agente en particular es miembro de un conjunto de agentes.

Esto solo araña la superficie. Consulte la Biblioteca de Modelos para obtener más ejemplos y consulte el Diccionario de NetLogo para obtener más información sobre todas las primitivas del conjunto de agentes.

Se proporcionan más ejemplos del uso de conjuntos de funciones en las entradas individuales para estas primitivas en el diccionario NetLogo.

Conjunto de Agentes Especiales

Los conjuntos de agentes `turtles` y `links` tienen un comportamiento especial porque siempre contienen los conjuntos de *todas* las tortugas y *todos* los enlaces. Por lo tanto, estos conjuntos pueden crecer.

La siguiente interacción muestra el comportamiento especial. Suponga que la pestaña Código tiene valor `globals [g]`. Entonces:

```
observer> clear-all
```

```
observer> create-turtles 5
observer> set g turtles
observer> print count g
5
observer> create-turtles 5
observer> print count g
10
observer> set g turtle-set turtles
observer> print count g
10
observer> create-turtles 5
observer> print count g
10
observer> print count turtles
15
```

El conjunto de agente `turtles` crece cuando nacen nuevas tortugas, pero otros grupos de agentes no crecen. Si escribimos `turtles-set turtles`, obtenemos un nuevo conjunto de agentes normal que contiene solo las tortugas que existen *actualmente*. Las nuevas tortugas no se unen cuando nacen.

Los conjuntos de agentes breed (razas, familias) son especiales de la misma manera como lo son las tortugas y los enlaces. Las familias se introducen y explican a continuación.

Conjuntos de Agentes y Listas

Anteriormente, dijimos que los conjuntos de agentes siempre están en orden aleatorio, un orden aleatorio diferente cada vez. Si necesita que sus agentes hagan algo en un orden fijo, debe hacer una lista de los agentes en su lugar. Ver la sección de Listas a continuación.

Ejemplo de código: Ask Ordering Example

Razas (Breeds)

Los modelos requieren de ciertas caracterizaciones de los actores que están contenidos en él. Por ejemplo, el estudio de las células puede hacerse a través de un modelo de redes celulares. En un modelo así, las células son un componente relevante, que podemos denominar clase, familia celular, o tratar a algunos conjuntos de células como tipo, grupo, raza, especie, casta, etc. NetLogo tipifica esto a través del tratamiento de las tortugas como un ente, una *clase* o *especie* representativa de *familias, o razas*. Otra tipificación son los enlaces, los que también pueden agruparse o formularse como clases, familias o “razas” de acuerdo con ciertas características. En la práctica, siguiendo las taxonomías de programación orientada a objetos, las *razas*, *familias, especies, castas,* se corresponden con las **clases** u **objetos y/o los tipos de variables**.

NetLogo permite definir diferentes "razas" (clases, objetos, especies, razas, castas, grupos) de tortugas y de enlaces. Una vez que haya definido las razas, puede continuar y hacer que las diferentes razas se comporten de manera diferente. Por ejemplo, se podría tener razas llamadas `sheep` y `wolves`, y hacer que los lobos intenten comerse las ovejas; o se podría tener familias enlazadas a través de enlaces denominados `streets` (calles) y `sidewalks` (aceras) donde el tráfico de peatones se enruta en las aceras y el tráfico de automóviles se enruta en las calles.

Se puede definir una raza de tortugas usando la palabra clave `breed`, en la parte superior de la pestaña Código, antes de realizar cualquier procedimiento:

```
breed [wolves wolf]
breed [sheep a-sheep]
```

Puede referirse a un miembro de la raza usando la forma singular, al igual que el reportero `turtle`. Cuando se impriman, los miembros de la raza serán etiquetados con el nombre singular.

Algunas instrucciones e informadores tienen el nombre plural de la raza en ellos, como `create- <breeds>`. Otros tienen el nombre singular de la raza, como `<breed>`.

El orden en el que se declaran las razas también es el orden en que se colocan en capas en la vista. Por lo tanto, las razas definidas más adelante aparecerán en la parte superior de las razas definidas anteriormente; en este ejemplo, las ovejas serán ubicadas sobre los lobos.

Cuando se define una breed (raza), tal como `sheep` (ovejas), se crea automáticamente un conjunto de agentes para esa raza, de modo que todas las capacidades del conjunto de agentes descritas anteriormente están disponibles de inmediato con y para el conjunto de agentes `sheep`.

Las siguientes nuevas primitivas también están disponibles automáticamente una vez que defina una raza: `create-sheep`, `hatch-sheep`, `sprout-sheep`, `sheep-here`, `sheep-at`, `sheep-on`, y `is-a-sheep?`

Además, se puede usar `sheep-own` (oveja-posee…*alguna propiedad/atributo*) para definir nuevas variables (propiedades o atributos) de tortuga que solo tienen las tortugas de la raza determinada. (Se permite que más de una raza posea la misma variable).

Un conjunto de agentes de la raza `turtle` se almacena en la variable `breed` de la raza `turtle`. Entonces se puede probar la raza de una tortuga, así:

```
if breed = wolves [ ... ] ;; si la raza es igual a lobos [instrucción]
```

Debe considerarse que las tortugas pueden cambiar de raza o familia o especie. Un lobo no tiene que seguir siendo un lobo toda su vida. Cambiemos un lobo aleatoriamente en una oveja:

```
ask one-of wolves [ set breed sheep ] ;; pídale a un lobo que se
;; convierta en oveja
```

La primitiva `set-default-shape` es útil para asociar ciertas formas de tortuga con ciertas razas, clases o especies. Vea la sección relativa a formas más abajo.

Los números de identidad (who number) son asignados independientemente de las razas. Si ya tiene un `frog 0` (una rana 0), entonces el primer ratón será `mouse 1`, no `mouse 0`, ya que el número 0 ya se tomó y asignó.

Aquí hay un ejemplo rápido del uso de razas:

```
breed [mice mouse] ;; familia o clase [ratones ratón]
breed [frogs frog] ;; familia o clase [ranas rana]
mice-own [cheese] ;; ratones poseen queso
to setup
  clear-all
  create-mice 50 ;; crea 50 ratones
    [ set color white  ;;de color blanco
      set cheese random 10 ] ;;con queso entre 0 y 9
  create-frogs 50  ;;crea 50 ranas
    [ set color green ]  ;;de color verde
```

```
  reset-ticks ;; resetea ticks
end
```

Ejemplo de Código: Breeds and Shapes Example

Razas o Clases de Enlaces (Links Breeds)

Las razas, clases, familias o especies de enlaces son muy similares a las razas de tortugas, sin embargo, existen algunas diferencias.

Cuando se declara una clase de enlace, se debe declarar si se trata de una variedad de enlaces dirigidos o no dirigidos mediante el uso de las palabras clave `directed-link-breed` y `undirected-link-breed`.

```
directed-link-breed [streets street] ;;calles calle
undirected-link-breed [friendships friendship] ;;amistades amistad
```

Una vez que se haya creado una clase de enlace, no se podrá crear enlaces sin clase y viceversa. (Sin embargo, puede tener enlaces dirigidos y no dirigidos en el mismo mundo, pero no en la misma clase)

A diferencia de la clase tortugas, se requiere un nombre de tipo singular para las clases de enlaces, ya que muchas de las instrucciones y reporteros de enlaces usan el nombre singular, como <link-breed>-neighbor?

Las siguientes primitivas también están disponibles automáticamente una vez que se establece una clase de enlace dirigida: `create-street-from`, `create-streets-from`, `create-street-to`, `in-street-neighbor?` `in-street-neighbors`, `in-street-from-my-in-streets`, `my-out-streets`, `out-street-neighbor?` `out-street-neighbors?` `out-street-to`.

Y las siguientes están disponibles automáticamente cuando se define una clase de enlace no dirigida: `create-friendship-with` `create-friendships-with` `friendship-neighbor?` `friendship-neighbors` `friendship-with` `my-friendships`.

Múltiples clases de enlaces pueden declarar la misma variable `-own`, pero una variable no se puede compartir entre una clase tortuga y una clase de enlace.

Al igual que con las clases de tortugas, el orden en que se declaran las clases de enlace define el orden en que se dibujan los enlaces, por lo que las amistades siempre estarán sobre o por arriba de las calles (si por alguna razón estas clases están en el mismo modelo). En otras palabras, se establece una jerarquía u orden de ejecución.

También se puede utilizar `<link-breeds>-own` para declarar variables de cada clase de enlaces separadamente.

Usted puede modificar la clase de un enlace con `set breed`. (Sin embargo, no se puede cambiar un enlace familiarizado a uno no familiarizado, con el objetivo de prevenir la existencia de enlaces familiarizados y no familiarizados en el mismo mundo). El concepto de "familiarizado" se refiere a la relación filial o de parentesco de un enlace con su clase (membrecía). Así los enlaces siempre poseen una relación de pertenencia con su clase y no pueden existir enlaces sin una clase a la cual pertenezcan.

```
ask one-of friendships [ set breed streets ]
ask one-of friendships [ set breed links ] ;; produce error en tiempo de ;; ejecución
```

`set-default-shape` puede también ser utilizada con razas, clases o familias de enlaces para asociarlas con una forma de enlace particular.

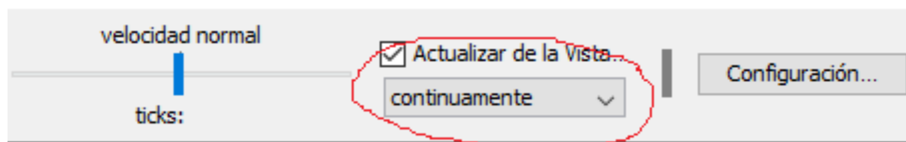
Código de Ejemplo: Link Breeds Example

Botones

Los botones en la pestaña Ejecutar proporcionan una manera fácil de controlar el modelo. Normalmente, un modelo tendrá al menos un botón setup para configurar el estado inicial del mundo y un botón go para que el modelo se ejecute continuamente. Algunos modelos tendrán botones adicionales que realizan otras acciones.

Un botón contiene algún código de NetLogo. Ese código se ejecuta cuando se presiona el botón.

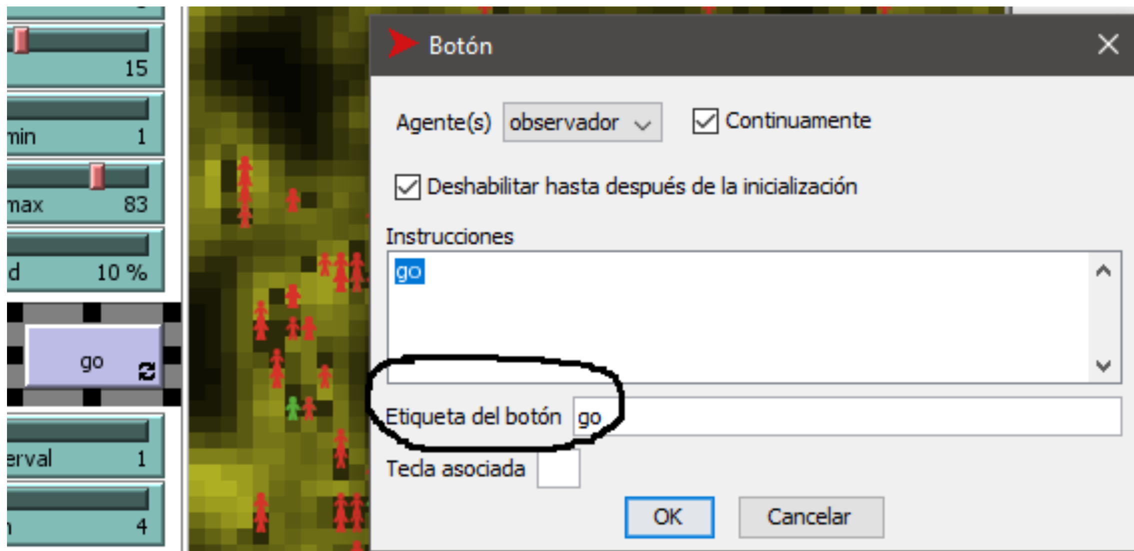
Un botón puede ser de tipo "una vez" (botón de una vez o de actividad discreta) o botón de ejecución continua. Puede controlar esto editando el botón y marcando o desmarcando la casilla "continuamente" (ver imagen siguiente). Una vez que los botones ejecuten el código una vez, es importante detenerse y volver a hacer una copia de seguridad. Los botones Continuamente siguen ejecutando su código una y otra vez.



Un botón caracterizado por "Continuamente" se detiene si el usuario presiona el botón nuevamente para detenerlo. El botón espera hasta que la iteración actual haya terminado, luego aparece.

Un botón "Continuamente" también se puede detener desde el código. Si el botón "Continuamente" llama directamente a un procedimiento, cuando el procedimiento se detiene, el botón se detiene. (En un botón Continuamente de tortuga o parcela, el botón no se detendrá hasta que cada tortuga o parcela se detenga; una sola tortuga o parcela no tiene la capacidad de detener el botón completamente).

Normalmente, un botón está etiquetado con el código que ejecuta. Por ejemplo, un botón que dice "go" generalmente contiene el código "go", que significa "ejecutar el procedimiento". (Los procedimientos se definen en la pestaña Código, ver más abajo). Pero también puede editar un botón e ingresar una "Etiqueta del botón" para el botón, que es un texto que aparece en el botón en lugar del código. Puede utilizar esta función si cree que el código real sería confuso para sus usuarios.



Cuando coloca el código en un botón, también debe especificar qué agentes desea ejecutar ese código. Puede elegir que el observador ejecute el código, o todas las tortugas, o todas las parcelas, o todos los enlaces. (Si desea que el código lo ejecuten solo algunas tortugas o algunas parcelas, puede hacer un botón de observador, y luego hacer que el observador use la instrucción `ask` para pedir solo a algunas de las tortugas o parcelas que hagan algo).

Cuando edita un botón, tiene la opción de asignar una "Tecla asociada", que se puede ver debajo de la "Etiqueta del botón" en la imagen anterior. Esto hace que esa tecla en el teclado se comporte como si se presionara un botón. Si el botón es "para siempre", permanecerá pulsado hasta que se vuelva a presionar la tecla (o se haga clic en el botón). Las teclas asociadas son particularmente útiles para juegos o cualquier modelo donde se requiera un disparo rápido de botones.

Botones que se Turnan

Se puede presionar más de un botón a la vez. Si esto sucede, los botones "se turnan", lo que significa que solo se ejecuta un botón a la vez. Cada botón ejecuta su código completamente una vez mientras los otros botones esperan, luego el siguiente botón obtiene su turno.

En los siguientes ejemplos, "setup" es un botón de una vez y "go" es un botón para siempre.

Ejemplo # 1: El usuario presiona "setup", luego presiona "go" inmediatamente, antes de que "setup" haya aparecido nuevamente. Resultado: "setup" termina antes de que comience "go".

Ejemplo # 2: Mientras el botón "go" está presionado, el usuario presiona "setup". Resultado: el botón "go" finaliza su iteración actual. Luego se ejecuta el botón "setup". Entonces "go" comienza a correr nuevamente.

Ejemplo n. ° 3: El usuario tiene dos botones siempre presionados al mismo tiempo. Resultado: el primer botón ejecuta su código completamente, luego el otro ejecuta su código hasta el final, y así sucesivamente, alternando.

Tenga en cuenta que, si un botón se atasca en un bucle infinito, no se ejecutarán otros botones.

Botones "para siempre" o de actividad continua de tortuga, parcela y enlace

Hay una diferencia sutil entre poner instrucciones en un botón de tortuga, parcela o enlace de tipo "continuamente", y poner las mismas instrucciones en un botón de observador que realiza `ask turtles`, `ask patches` o `ask links`. Una instrucción "`ask`" no se completa hasta que todos los agentes hayan terminado de ejecutar todas las instrucciones "`ask`". Entonces, los agentes, como todos ejecutan las instrucciones al mismo tiempo, pueden estar desincronizados entre ellos, pero todos se sincronizan nuevamente al final de la solicitud `ask`. Lo mismo no es cierto con los botones de actividad continua de tortuga, parcela y enlace. Como no se utilizó `ask`, cada tortuga o parcela ejecuta el código dado una y otra vez, por lo que pueden volverse (y permanecer) desincronizados entre sí.

En la actualidad, esta capacidad se utiliza muy poco en los modelos de nuestra Biblioteca de modelos. Un modelo que sí utiliza la capacidad es el modelo de termitas, en la sección Biology de la sección Sample Models. El botón "go" es un botón de actividad continua para tortugas, por lo que cada termita procede independientemente de cada otra termita, y el observador no está involucrado en absoluto. Esto significa que, si, por ejemplo, se desea agregar ticks y / o un gráfico al modelo, deberá agregar un segundo botón de actividad continua (un botón "para siempre" del observador) y ejecutar ambos botones "para siempre" al mismo tiempo. Tenga en cuenta también que un modelo como este no se puede usar con BehaviorSpace.

Ejemplo de código: State Machine Example muestra cómo las Termitas pueden ser recodificadas en una forma basada en pasos (ticks) sin utilizar un botón de tortuga de actividad continua.

Actualmente, NetLogo no tiene forma de que un botón de actividad continua (forever o "para siempre") inicie o active a otro. Los botones solo se inician cuando se los presiona.

Listas

En los modelos más simples, cada variable contiene solo una pieza de información, generalmente un número o una cadena. Las listas permiten almacenar múltiples piezas de información en un solo valor al recopilar esa información en una lista. Cada valor en la lista puede ser cualquier tipo de valor: un número, una cadena, un agente o conjunto de agentes, o incluso otra lista.

Las listas permiten un muy conveniente empaquetamiento de la información en NetLogo. Si sus agentes realizan un cálculo repetitivo en múltiples variables, podría ser más fácil tener una variable de lista, en lugar de múltiples variables numéricas. Varias primitivas simplifican el proceso de realizar el mismo cálculo en cada valor en una lista. De esta manera, se opera con listas bajo un contexto matricial.

El diccionario de NetLogo contiene una sección que enumera todas las primitivas relacionadas con una lista.

Listas de Constantes

Se puede hacer una lista simplemente colocando los valores que se desea en la lista entre paréntesis, por ejemplo, de esta manera: `set mylist [2 4 6 8]`. Tenga en cuenta que los valores individuales están separados por espacios. Puede hacer listas que contengan números y cadenas de esta manera, así como listas dentro de listas, por ejemplo `[[2 4] [3 5]]`.

La lista vacía se escribe poniendo nada entre corchetes, como este: `[]`.

Construyendo Listas al Vuelo

Si desea hacer una lista en la que los valores son determinados por los informadores, en lugar de ser una serie de constantes, use el reportero `list`. El reportero `list` acepta a otros dos informadores, los ejecuta e informa los resultados como una lista.

Si quisiera que una lista contenga dos valores aleatorios, podría usar el siguiente código:

```
set random-list list (random 10) (random 20)
```

Esto definirá una `random-list` (lista aleatoria) que creará una nueva lista de dos enteros aleatorios cada vez que se ejecute.

Para hacer listas más largas o más cortas, puede usar el reportero `list` con menos o más de dos entradas, pero para hacerlo, debe incluir la llamada completa entre paréntesis, por ejemplo:

```
(list random 10)
(list random 10 random 20 random 30)
```

Para mayor información ver: Varying number of inputs.

Algunos tipos de listas se crean más fácilmente utilizando el reportero `n-values`, que le permite construir una lista de una longitud específica ejecutando repetidamente un reportero determinado. Puede hacer una lista del mismo valor repetido, o todos los números en un rango, o muchos números aleatorios, u otras muchas posibilidades. Consulte la entrada del diccionario para obtener detalles y ejemplos.

La primitiva `of` permite construir una lista desde un agente. Informa una lista que contiene el valor de cada agente para el reportero dado. (El reportero podría ser un nombre de variable simple, o una expresión más compleja, incluso una llamada a un procedimiento definido usando `to-report`). Un modismo común es:

```
max [...] of turtles
```

```
sum [...] of turtles
```

Y así sucesivamente.

Se pueden combinar dos o más listas utilizando el informador `sentence`, que concatena las listas combinando sus contenidos en una única lista más grande. Como `list`, `sentence`, que normalmente toman dos entradas, pero pueden aceptar cualquier cantidad de entradas si la llamada está rodeada por paréntesis.

Cambiando los elementos de la Lista

Técnicamente, las listas no se pueden modificar, pero puede construir nuevas listas basadas en listas anteriores. Si desea que la nueva lista reemplace la lista anterior, use `set`. Por ejemplo:

```
set mylist [2 7 5 Bob [3 0 -2]]
; mylist is now [2 7 5 Bob [3 0 -2]]
set mylist replace-item 2 mylist 10
; mylist is now [2 7 10 Bob [3 0 -2]]
```

La expresión `replace-item` toma tres entradas. La primera entrada especifica qué elemento de la lista se va a cambiar. Dado que se trata de una expresión matricial, 0 significa el primer elemento, 1 significa el segundo elemento, y así sucesivamente.

Para agregar un elemento, digamos 42, al final de una lista, use el reportero `lput`, que significa "ponga al último". (`fput`, abrevia *first* put que significa "ponga primero", agrega un elemento al comienzo de una lista).

```
set mylist lput 42 mylist
; mylist is now [2 7 10 Bob [3 0 -2] 42]
```

¿Pero y si cambiase de opinión? La expresión `but-last` (`bl` para abreviar) informa todos los elementos de la lista, pero el último.

```
set mylist but-last mylist
; mylist is now [2 7 10 Bob [3 0 -2]]
```

Supongamos que quiere deshacerse del ítem 0, el 2 al principio de la lista.

```
set mylist but-first mylist
; mylist is now [7 10 Bob [3 0 -2]]
```

Supongamos que quiere cambiar el tercer elemento que está anidado dentro del elemento 3 de -2 a 9. La clave es darse cuenta de que el nombre que se puede usar para llamar a la lista anidada [3 0 -2] es el elemento 3 `mylist`. Luego, el reportero `replace-item` puede anidarse para cambiar la lista dentro de una lista. Los paréntesis se agregan para mayor claridad.

```
set mylist (replace-item 3 mylist
                  (replace-item 2 (item 3 mylist) 9))
; mylist is now [7 10 Bob [3 0 9]]
```

Iteraciones sobre Listas

Si se desea realizar alguna operación en cada elemento de una lista, la instrucción `foreach` y el reportero `map` pueden ser útiles.

`foreach` se usa para ejecutar una instrucción en cada elemento de una lista. Toma una lista de entrada y un nombre de instrucción o bloque de instrucciones, como este:

```
foreach [1 2 3] show
=> 1
=> 2
=> 3
foreach [2 4 6]
  [ n -> crt n
    show (word "created " n " turtles") ]
=> created 2 turtles
=> created 4 turtles
=> created 6 turtles
```

En el bloque, la variable `n` contiene el valor actual de la lista de entrada.

Aquí hay algunos ejemplos más de `foreach`:

```
foreach [1 2 3] [ steps -> ask turtles [ fd steps ] ]
;; turtles move forward 6 patches
foreach [true false true true] [ should-move? -> ask turtles [ if should-move? [ fd 1 ] ]
]
;; turtles move forward 3 patches
```

`map` es similar a `foreach`, pero es un reportero. Toma una lista de entrada y un nombre de reportero o un bloque de reportero. Tenga en cuenta que a diferencia de `foreach`, el reportero es lo primero, así:

```
show map round [1.2 2.2 2.7]
;; prints [1 2 3]
```

`map` informa una lista que contiene los resultados de aplicar el reportero a cada elemento en la lista de entrada. De nuevo, use la variable nombrada en el procedimiento anónimo (`x` en los ejemplos a continuación) para referirse al elemento actual en la lista.

Aquí hay un par de ejemplos más de `map`:

```
show map [ x -> x < 0 ] [1 -1 3 4 -2 -10]
;; prints [false true false false true true]
show map [ x -> x * x ] [1 2 3]
;; prints [1 4 9]
```

Además de `map` and `foreach`, otras primitivas para procesar listas enteras de una manera configurable incluyen `filter`, `reduce` y `sort-by`.

Debe considerarse que estas primitivas no siempre son la solución para cada situación en la que desee operar en una lista completa. En algunas situaciones, es posible que se deba usar alguna otra técnica, como un ciclo, utilizando `repeat` o `while`, o un procedimiento recursivo.

Los bloques de código que estamos dando para `map` y `foreach`, en estos ejemplos, son en realidad **procedimientos anónimos**. Los procedimientos anónimos se explican con más detalle más adelante.

Variando el Número de Entradas

Algunas instrucciones y reporteros que involucran listas y cadenas pueden tomar una cantidad variable de entradas. En estos casos, para pasarles una cantidad de entradas distinta a la predeterminada, la primitiva y sus entradas deben estar rodeadas por paréntesis. Aquí hay unos ejemplos:

```
show list 1 2
=> [1 2]
show (list 1 2 3 4)
=> [1 2 3 4]
show (list)
=> []
```

Note que cada una de estas primitivas especiales tiene un número predeterminado de entradas para las cuales no se requieren paréntesis. Las primitivas que tienen esta capacidad son: `list`, `word`, `sentence`, `map`, `foreach`, `run` y `runresult`.

Listas de Agentes

Anteriormente, dijimos que los conjuntos de agentes (`agentsets`) siempre están en orden aleatorio, un orden aleatorio diferente cada vez. Si se necesita que los agentes hagan algo en un orden fijo, se debe hacer una lista de los agentes.

Hay dos primitivas que ayudan a hacer esto, `sort` y `sort-by`.

Tanto `sort` como `sort-by` pueden tomar un agente como entrada. El resultado es siempre una nueva lista, que contiene los mismos agentes del conjunto de agentes, pero en un orden particular.

Si se utiliza `sort` sobre un conjunto de agentes de tortugas, el resultado es una lista de tortugas ordenadas en orden ascendente por el número identificador `who`.

Si se utiliza `sort` sobre un conjunto de agentes de parcelas, el resultado es una lista de parcelas ordenadas de izquierda a derecha, de arriba hacia abajo.

Si se utiliza `sort` sobre un conjunto de agentes de enlaces, el resultado es una lista de enlaces, ordenados en orden ascendente primero por `end1` y luego por `end2`, y cualquier vínculo restante se resuelve por raza o clase en el orden en que se declaran en la pestaña Código.

Si se requiere un orden descendente, se puede combinar `reverse` con `sort`, por ejemplo, `reverse sort turtles`.

Si se desea que los agentes se ordenen por algún otro criterio que no sean los estándares, debe usar `sort-by`. Por ejemplo:

```
sort-by [ [a b] -> [size] of a < [size] of b ] turtles
```

Esto devuelve una lista de tortugas clasificadas en orden ascendente por su variable tamaño de tortuga `size`.

Hay un patrón común para obtener una lista de agentes en orden aleatorio, usando una combinación de `of` y `self`, en un caso excepcional que no pueda usar `ask`:

```
[self] of my-agentset
```

Solicitando Algo a una Lista de Agentes

Una vez que tenga una lista de agentes, es posible que desee pedirle a cada uno que haga algo. Para hacer esto, use las instrucciones `foreach` y `ask` en combinación, como en este ejemplo:

```
foreach sort turtles [ the-turtle ->
  ask the-turtle [
    ...
  ]
]
```

Esto le solicitará a cada tortuga su número identificador (`who` number) en orden ascendente. Sustituya "patches" por "turtles" para pedir un ordenamiento de parcelas de izquierda a derecha, en orden de arriba hacia abajo.

Note que no se puede usar `ask` directamente en una lista de tortugas. `ask` solo funciona con conjuntos de agentes (agentsets) y agentes individuales.

Rendimiento con Listas

La estructura de datos subyacente en las listas de NetLogo es una estructura de datos sofisticada, basada en árboles, en la que la mayoría de las operaciones se ejecutan en tiempo casi constante. Eso incluye `fput`, `lput`, `butfirst`, `butlast`, `length`, `item` y `replace-item`.

Una excepción a la regla de rápido rendimiento es que la concatenación de dos listas con `sentence` requiere atravesar y copiar toda la segunda lista. (Esto puede corregirse en una versión futura).

Técnicamente, "tiempo casi constante" es en realidad tiempo logarítmico, proporcional a la profundidad del árbol subyacente, pero estos árboles tienen nodos grandes y un alto factor de ramificación, por lo que nunca son más que unos pocos niveles de profundidad. Esto significa que los cambios se pueden hacer en unos pocos pasos. Los árboles son inmutables, pero comparten estructura entre sí, por lo que no es necesario copiar el árbol completo para crear una versión modificada.

La estructura de datos real utilizada es la clase immutable Vector de la biblioteca de colecciones Scala. Se trata del desarrollo de intentos de mapeado de una matriz de ancho 32, implementados por Tiark Rompf, basados en parte en el trabajo de Phil Bagwell y Rich Hickey. En términos simples, se trata de algoritmos que buscan ser eficientes en la penetración de las estructuras de datos de tal modo que se pierda el menor tiempo posible en la búsqueda de datos dentro de la estructura de dichos datos.

Matemática

Todos los números en NetLogo se almacenan internamente como números de coma flotante de precisión doble, como se define en el estándar IEEE 754. Son números de 64 bits que constan de un bit de signo, un exponente de 11 bits y una mantisa de 52 bits. Ver el estándar IEEE 754 para más detalles.

Un "entero" en NetLogo es simplemente un número que no tiene parte fraccionaria. No se hace distinción entre 3 y 3.0; ellos son el mismo número (Esto es igual a como la mayoría de las personas usan los números en contextos cotidianos, lo que es diferente en algunos lenguajes de programación. Algunos lenguajes tratan los números enteros y los números flotantes como tipos distintos).

Los enteros siempre son impresos por NetLogo sin el final ".0":

```
show 1.5 + 1.5
observer: 3
```

Si se suministra un número con una parte fraccionaria en un contexto donde se espera un número entero, la parte fraccional simplemente se descarta. Entonces, por ejemplo, `crt 3.5` crea tres tortugas; el 0.5 adicional se ignora, puesto que no es real tener “tres y media tortugas”.

El rango de enteros es +/- 9007199254740992 (2 ^ 53, alrededor de 9 cuatrillones). Los cálculos que exceden este rango no causarán errores de tiempo de ejecución, pero la precisión se perderá cuando los dígitos menos significativos (binarios) se redondeen para ajustar el número en 64 bits. Con números muy grandes, este redondeo puede dar lugar a respuestas imprecisas que pueden ser sorprendentes:

```
show 2 ^ 60 + 1 = 2 ^ 60
=> true
```

Los cálculos con números más pequeños también pueden producir resultados sorprendentes si implican cantidades fraccionarias, ya que no todas las fracciones pueden representarse con precisión y puede producirse un redondeo. Por ejemplo:

```
show 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6
=> 0.9999999999999999
show 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9
=> 1.0000000000000002
```

Es importante señalar que cualquier operación que produzca cantidades especiales, tal como "infinito" o "no un número" causará un error de tiempo de ejecución.

Notación Científica

NetLogo muestra números de coma flotante muy grandes o muy pequeños usando "notación científica". Ejemplos:

```
show 0.000000000001
=> 1.0E-12
show 50000000000000000000
=> 5.0E19
```

Los números en notación científica se distinguen por la presencia de la letra E (para "exponente"). Significa "multiplicado por diez a la potencia de", por lo que, por ejemplo, 1.0E-12 significa 1.0 veces 10 a la potencia -12:

```
show 1.0 * 10 ^ -12
=> 1.0E-12
```

También el programador puede utilizar la notación científica en el código de NetLogo:

```
show 3.0E6
=> 3000000
show 8.123456789E6
=> 8123456.789
show 8.123456789E7
=> 8.123456789E7
show 3.0E16
=> 3.0E16
show 8.0E-3
=> 0.0080
show 8.0E-4
=> 8.0E-4
```

Estos ejemplos muestran que los números con partes fraccionarias se muestran usando notación científica si el exponente es menor que -3 o mayor que 6. Los números fuera del rango entero de NetLogo de -9007199254740992 a 9007199254740992 (+/- 2 ^ 53) también se muestran siempre en notación científica:

```
show 2 ^ 60
=> 1.15292150460684698E18
```

Al ingresar un número, la letra E puede ser mayúscula o minúscula. Al imprimir un número, NetLogo siempre usa una E mayúscula:

```
show 4.5e20
=> 4.5E20
```

Precisión del Punto Flotante

Debido a que los números en NetLogo están sujetos a las limitaciones de cómo se representan los números de punto flotante en binario, se pueden obtener respuestas que son ligeramente inexactas. Por ejemplo:

```
show 0.1 + 0.1 + 0.1
=> 0.30000000000000004
show cos 90
=> 6.123233995736766E-17
```

Este es un problema inherente a la aritmética de coma flotante; ocurre en todos los lenguajes de programación que usan números de coma flotante.

Si está tratando con cantidades de precisión fijas, por ejemplo, dólares y centavos, una técnica común es usar solo enteros (centavos) internamente, luego divida entre 100 para obtener un resultado en dólares para visualizar.

Si se requiere utilizar números de coma flotante, en algunas situaciones se puede requerir el reemplazo de una prueba de igualdad directa, como `if x = 1 [...]` con una prueba que tolere una ligera imprecisión, por ejemplo, `if abs (x - 1) < 0.0001 [...]`.

Además, la primitiva `precision` es útil para redondear números con fines de visualización. Los monitores de NetLogo redondean los números que muestran, con un número configurable de lugares decimales, también.

Números Aleatorios

Los números aleatorios utilizados por NetLogo son lo que se denomina "pseudoaleatorio". (Esto es típico en la programación de computadoras). Eso significa que parecen aleatorios, pero de hecho se generan mediante un proceso determinista. "Determinista" significa que obtendrá los mismos resultados cada vez, si comienza con la misma "semilla" al azar. Explicaremos en un minuto qué queremos decir con "semilla".

En el contexto del modelado científico, los números pseudoaleatorios son realmente necesarios. Esto se debe a que es importante que un experimento científico sea reproducible, por lo que cualquiera puede probarlo por sí mismo y obtener el mismo resultado que obtuviste. Como NetLogo usa números pseudoaleatorios, los "experimentos" que se hacen con él pueden ser reproducidos por otros.

Así es como funciona. El generador de números aleatorios de NetLogo se puede iniciar con un cierto valor de inicialización, que debe ser un número entero en el rango -2147483648 a 2147483647. Una vez que el generador ha sido "sembrado" con la instrucción de semilla aleatoria, siempre genera la misma secuencia de números aleatorios a partir de entonces. Por ejemplo, si ejecuta estas instrucciones:

```
random-seed 137
show random 100
show random 100
show random 100
```

Siempre obtendrá los números 79, 89 y 61 en ese orden.

Es preciso notar que, solo está garantizado que obtendrá esos mismos números si está utilizando la misma versión de NetLogo. A veces, cuando hacemos una nueva versión de NetLogo, cambia el generador de números aleatorios. (Actualmente, usamos un generador conocido como Mersenne Twister).

Para crear un número adecuado para sembrar el generador de números aleatorios, use el reportero `new-seed`. Este crea una nueva semilla, distribuida uniformemente sobre el espacio de posibles semillas, en base a la fecha y hora actuales. Nunca informa la misma semilla dos veces seguidas.

Código de Ejemplo: Random Seed Example

Si no configura usted mismo la semilla aleatoria, NetLogo la establece en un valor basado en la fecha y hora actual. No hay forma de averiguar qué semilla aleatoria eligió, por lo que, si desea que su ejecución de modelo sea reproducible, debe configurar la semilla aleatoria usted mismo con antelación.

Las primitivas de NetLogo con el adjetivo "`random`" (aleatorio) en sus nombres (`random`, `random-float`, etc.) no son las únicas que usan números pseudoaleatorios. Muchas otras operaciones también toman decisiones al azar. Por ejemplo, los conjuntos de agentes están siempre en orden aleatorio, `one-of` y `n-of` eligen agentes aleatoriamente, la instrucción `sprout` (germinar, brotar) crea tortugas con colores y encabezados aleatorios, y el reportero `down-hill` (cuesta-abajo) elige una parcela aleatoria cuando hay un empate. Todas estas elecciones aleatorias también se rigen por la semilla aleatoria, por lo que las ejecuciones del modelo pueden ser reproducibles.

Además de los enteros aleatorios distribuidos uniformemente y los números de coma flotante generados por `random` y `random-float`, NetLogo también ofrece varias otras distribuciones aleatorias. Consulte las entradas del diccionario para `random-normal`, `random-poisson`, `random-exponential` y `random-gamma`, las cuales son distribuciones estadísticas específicas.

Generador Auxiliar

El código ejecutado por botones o desde el Terminal de instrucciones usa el generador de números aleatorios principal.

El código en los monitores utiliza un generador auxiliar, por lo que incluso si un monitor hace un cálculo que usa números aleatorios, el resultado del modelo no se verá afectado. Lo mismo ocurre con el código en los controles deslizantes.

Aleatoriedad Local

Es posible que desee especificar explícitamente que una sección de código no afecte el estado del generador aleatorio principal, por lo que el resultado del modelo no se verá afectado. La instrucción `with-local-randomness` es proporcionado para este propósito. Vea su entrada en el diccionario NetLogo para más información.

Formas de Tortuga

En NetLogo, las formas de tortuga son formas vectoriales. Se construyen a partir de formas geométricas básicas; cuadrados, círculos y líneas, en lugar de una cuadrícula de píxeles. Las formas vectoriales son totalmente escalables y giratorias. NetLogo almacena en caché las imágenes de mapa de bits de las formas vectoriales tamaño 1, 1.5 y 2 para acelerar la ejecución.

La forma de una tortuga se almacena en su variable `shape` y se puede definir utilizando la instrucción `set`.

Las nuevas tortugas tienen una forma por "`default`". La primitiva `set-default-shape` es útil para cambiar la forma de tortuga predeterminada a una forma diferente, o tener una forma de tortuga predeterminada diferente para cada clase de tortuga.

La primitiva `shapes` informa una lista de las formas de tortuga actualmente disponibles en el modelo. Esto es útil si, por ejemplo, desea asignar una forma aleatoria a una tortuga:

```
ask turtles [ set shape one-of shapes ]
```

Utilice el Editor de formas de tortuga para crear sus propias formas de tortuga, o para agregar formas a su modelo desde nuestra biblioteca de formas, o para transferir formas entre modelos. Para obtener más información, consulte la sección Editor de formas de este manual.

El grosor de las líneas utilizadas para dibujar las formas vectoriales se puede controlar mediante la primitiva `set-line-thickness`.

Códigos de Ejemplo: Breeds and Shapes Example, Shape Animation Example

Formas de Enlaces

Las formas de enlace son similares a las formas de tortuga, solo que se usa el Editor de forma de enlace para crearlas y editarlas. Las formas de enlace consisten en entre 0 y 3 líneas que pueden tener diferentes patrones y un indicador de dirección que se compone de los mismos elementos que las formas de tortuga. Los enlaces también tienen una variable `shape` que se puede establecer en cualquier forma de enlace que esté en el modelo. Por defecto, los enlaces tienen la forma "default", aunque se pueden modificar usando `set-default-shape`. El informador `link-shapes` reporta todas las formas de enlace incluidas en el modelo actual.

El grosor de las líneas en la forma del enlace está controlado por la variable de enlace `thickness`.

Actualización de las Vistas

La "vista" en NetLogo permite ver los agentes de su modelo en la pantalla de su computadora. A medida que sus agentes se mueven y cambian, los ve moverse y cambiando en la vista.

Por supuesto, no se puede ver a los agentes directamente. La vista es una imagen que NetLogo pinta, que le muestra cómo sus agentes miran un instante en particular. Una vez que ese instante pasa y sus agentes se mueven y cambian un poco más, esa imagen debe ser repintada para reflejar el nuevo estado del mundo. Volver a pintar la imagen se denomina "actualizar" la vista.

¿Cuándo se actualiza la vista? En esta sección se explica cómo NetLogo decide cuándo actualizar la vista y cómo puede influir sobre cuándo se actualiza.

NetLogo ofrece dos modos de actualización, actualizaciones "continuas" (`continuous`) y actualizaciones "basadas en ticks" (`tick-based`). Puede alternar entre los dos modos de actualización de vista de NetLogo usando un menú emergente en la parte superior de la pestaña Interfaz.

Las actualizaciones continuas son las predeterminadas cuando inicia NetLogo o inicia un nuevo modelo. Sin embargo, casi todos los modelos de nuestra Biblioteca de modelos usan actualizaciones basadas en ticks.

Las actualizaciones continuas son las más simples, pero las actualizaciones basadas en ticks proveen más control sobre cuándo y con qué frecuencia ocurren las actualizaciones.

Es importante exactamente cuándo ocurre una actualización, porque cuando suceden las actualizaciones determina lo que se ve en la pantalla. Si se produce una actualización en un momento inesperado, es posible que vea algo inesperado, tal vez algo confuso o engañoso.

También es importante con qué frecuencia ocurren las actualizaciones, ya que las actualizaciones toman tiempo. Cuanto más tiempo pase NetLogo actualizando la vista, más lento se ejecutará su modelo. Con menos actualizaciones, su modelo se ejecuta más rápido.

Actualizaciones Continuas

Las actualizaciones continuas son muy simples. Con las actualizaciones continuas, NetLogo actualiza la vista un cierto número de veces por segundo, de manera predeterminada, 30 veces por segundo cuando el control deslizante de velocidad está en la configuración intermedia predeterminada.

Si mueve el control deslizante de velocidad a una configuración más lenta, NetLogo se actualizará más de 30 veces por segundo, reduciendo la velocidad del modelo. En una configuración más rápida, NetLogo se actualizará menos de 30 veces por segundo. En la configuración más rápida, las actualizaciones estarán separadas por varios segundos.

En configuraciones extremadamente lentas, NetLogo se actualizará con tanta frecuencia que verá a sus agentes moverse (o cambiar el color, etc.) de a uno por vez.

Si necesita cerrar temporalmente las actualizaciones continuas, use la instrucción `no-display`. La instrucción `display` vuelve a activar las actualizaciones y también obliga a una actualización inmediata (a menos que el usuario esté reenviando el modelo con el control deslizante de velocidad).

Actualizaciones Basadas en Ticks

Como se discutió anteriormente en la sección Contador de ticks, en muchos modelos de NetLogo, el tiempo pasa en pasos discretos, llamados "ticks". Normalmente, se requiere que la vista se actualice una vez por tick, entre ticks. Ese es el comportamiento predeterminado con actualizaciones basadas en ticks.

Si se desea actualizaciones de vista adicionales, se puede forzar una actualización utilizando la instrucción `display`. (La actualización se puede omitir si el usuario está acelerando el modelo con el control deslizante de velocidad).

No tiene que usar el contador de ticks para usar actualizaciones basadas en ticks. Si el contador de tics nunca avanza, la vista se actualizará solo cuando use la instrucción `display`.

Si mueve el control deslizante de velocidad a una configuración lo suficientemente rápida, con el tiempo, NetLogo omitirá algunas de las actualizaciones que normalmente habrían sucedido. Mover el control deslizante de velocidad a una configuración más lenta no causa actualizaciones adicionales; más bien, hace que NetLogo se detenga después de cada actualización. Cuanto más lenta sea la configuración, más larga será la pausa.

Incluso en actualizaciones basadas en ticks, la vista también se actualiza cada vez que aparece un botón en la interfaz (botones `once` y `forever`) y cuando termina una instrucción en el Terminal de instrucciones. Por lo tanto, no es necesario agregar la instrucción `display` a los botones `once` que no avanzan el contador de ticks. Muchos botones forever que no avanzan en el contador de ticks necesitan usar la instrucción `display`. Un ejemplo en la Biblioteca de Modelos es el modelo Life (en Computer Science -> Cellular Automata). Los botones `forever` que permiten al usuario dibujar en la vista utilizan la instrucción `display` para que el usuario pueda ver lo que están dibujando, aunque el contador de ticks no avance.

Eligiendo un Modo

Las ventajas de las actualizaciones basadas en ticks sobre actualizaciones continuas incluyen:

- Comportamiento de actualización de vista consistente y predecible que no varía de una computadora a otra o de una ejecución a otra.
- Las actualizaciones continuas pueden confundir al usuario de su modelo al permitirle ver los estados del modelo que se supone que no debe ver, lo que puede ser engañoso.
- Dado que los botones `setup` de configuración no avanzan en el contador de ticks, no se ven afectados por el control deslizante de velocidad; este es normalmente el comportamiento deseado.

Casi todos los modelos de nuestra Biblioteca de modelos usan actualizaciones basadas en ticks.

Las actualizaciones continuas son ocasionalmente útiles para aquellos modelos raros en los que la ejecución no se divide en fases cortas y discretas. Un ejemplo en la Biblioteca de Modelos es Termites. (Véase también, sin embargo, el modelo State Machine Example, que muestra cómo volver a codificar las termitas utilizando marcas).

Incluso para los modelos que normalmente se configuran como actualizaciones basadas en ticks, puede ser útil cambiar a actualizaciones continuas temporalmente para fines de depuración. Ver lo que sucede dentro de un tick, en lugar de solo

ver el resultado final de un tick, podría ayudar a resolver problemas. Después de cambiar a actualizaciones continuas, es posible que desee usar el control deslizante de velocidad para ralentizar el modelo hasta que vea que sus agentes se mueven uno a la vez. No olvide volver a las actualizaciones basadas en ticks cuando haya terminado, ya que la opción de modo de actualización se guarda con el modelo.

Cambiar el modo de actualización también afecta la velocidad del modelo. La actualización de la vista lleva tiempo; a menudo aplicando una sola actualización por tick (mediante el uso de actualizaciones basadas en ticks) hará que su modelo sea más rápido. Por otro lado, las actualizaciones continuas serán más rápidas cuando ejecutar un solo tick es más rápido que dibujar un cuadro del modelo. La mayoría de los modelos se ejecutan más rápido en las actualizaciones basadas en ticks, pero para un ejemplo de un modelo que es más rápido con actualizaciones continuas, consulte el modelo de biblioteca "Heroes and Cowards".

Los Cuadros por Segundo (Frame Rate)

Una de las configuraciones del modelo en el cuadro de diálogo "Configuración ..." de NetLogo es "Frecuencia de cuadros", que por defecto es de 30 fotogramas por segundo.

La configuración de velocidad de fotogramas afecta tanto a las actualizaciones continuas como a las actualizaciones basadas en ticks.

Con actualizaciones continuas, la configuración determina directamente la frecuencia de las actualizaciones.

Con las actualizaciones basadas en ticks, la configuración limita la cantidad de actualizaciones por segundo que recibe. Si la velocidad de fotogramas es 30, NetLogo se asegurará de que el modelo nunca se ejecute más rápido que cuando el control deslizante de velocidad está en la posición predeterminada. Si cualquier cuadro tarda menos de 1/30 de segundo para computar y mostrar, NetLogo hará una pausa y esperará hasta que haya pasado 1/30 de segundo completo antes de continuar.

La configuración de velocidad de fotogramas le permite configurar lo que considera que es una velocidad normal para su modelo. Entonces usted o el usuario de su modelo pueden usar el deslizador de velocidad para obtener temporalmente una velocidad más rápida o más lenta.

Graficando

Las funciones para graficar de NetLogo le permiten crear gráficos para ayudarlo a comprender lo que está sucediendo en su modelo.

Antes de poder graficar, necesita crear uno o más gráficos en la pestaña Ejecutar. Para obtener más información sobre el uso y edición de gráficos en la pestaña Ejecutar, consulte la Interface Guide.

Graficando Puntos

Las dos instrucciones básicas para graficar cosas efectivamente son `plot` y `plotxy`.

Con `plot` solo se requiere especificar el valor de la coordenada `y` que se desea graficar. El valor de `x` será automáticamente 0 para el primer punto que grafique, 1 para el segundo, y así sucesivamente. (Esto es, si el "intervalo" del lápiz graficador tiene el valor por defecto de 1; puede cambiar el intervalo).

La instrucción `plot` es especialmente útil cuando quiere que su modelo grafique un nuevo punto en cada paso de tiempo. Ejemplo:

```
plot count turtles
```

Si necesita especificar los valores x e y del punto que desea graficar, entonces use `plotxy` en su lugar. Este ejemplo supone que existe una variable global llamada `time`:

```
plotxy time count-turtles
```

Instrucciones para Graficar

Cada parcela y sus plumas tienen campos de configuración y actualización de código que pueden contener instrucciones (que generalmente contienen `plot` o `plotxy`). Estas instrucciones se ejecutan automáticamente activadas por otras instrucciones en NetLogo.

Las instrucciones de configuración de gráfico y las instrucciones de configuración de lápiz se ejecutan cuando se ejecutan las instrucciones `reset-ticks` o `setup-plots`. Si la instrucción de detención `stop` se ejecuta en el cuerpo de las instrucciones de configuración gráfica, las instrucciones de configuración del lápiz no se ejecutarán.

Las instrucciones de actualización de gráfico y las instrucciones de actualización de lápiz se ejecutan cuando se ejecutan las instrucciones `reset-ticks`, `tick` o `update-plots`. Si la instrucción de detención `stop` se ejecuta en el cuerpo de las instrucciones de actualización de gráfico, las instrucciones de actualización del lápiz no se ejecutarán.

Aquí están las cuatro instrucciones que desencadenan el gráfico explicado con más detalle.

- `setup-plots` ejecuta instrucciones para un diagrama a la vez. Para cada gráfico, se ejecutan las instrucciones de configuración de gráficos o trazado. Si la instrucción de detención `stop` no se encuentra al ejecutar esas instrucciones, se ejecutará el código de configuración de cada uno de los lápices de la gráfica.
- `update-plots` es muy similar a `setup-plots`. Para cada gráfico, se ejecutan las instrucciones de actualización de gráfico. Si la instrucción `stop` no se encuentra al ejecutar esas instrucciones, se ejecutará el código de actualización de cada uno de los lápices de la gráfica.
- `tick` es exactamente lo mismo que `update-plots`, excepto que el contador de ticks se incrementa antes de que se ejecuten las instrucciones de gráfico.
- `reset-ticks` primero restablece el contador de ticks a 0, y luego hace el equivalente a `setup-plots` seguido de `update-plots`.

Un modelo típico usará `reset-ticks` y `tick` de este modo:

```
to setup
  clear-all
  ...
  reset-ticks
end

to go
  ...
  tick
end
```

Tenga en cuenta que en este ejemplo graficamos desde los procedimientos `setup` y `go` (porque `reset-ticks` ejecuta la configuración de gráfico y las instrucciones de actualización de gráfico). Hacemos esto porque queremos que nuestro diagrama incluya el estado inicial del sistema al final de `setup`. Graficamos al final del procedimiento `go`, y no al principio, porque queremos que el gráfico esté siempre actualizado después de que se detenga el botón `go`.

Los modelos que no usan ticks, pero aún quieren hacer el gráfico usarán `setup-plots` y `update-plots`. En el código anterior, reemplace `reset-ticks` con `setup-plots update-plots` y reemplace tick con `update-plots`.

Código Ejemplo: Plotting Example

Otros Tipos de Gráfico

De forma predeterminada, los lápices de gráfico de NetLogo grafican en modo de línea, de modo que los puntos que traza están conectados por una línea.

Si se desea mover el lápiz sin graficar, se puede usar la instrucción `plot-pen-up`, que es como dejar el lápiz en suspensión en el aire para que no pueda rayar la hoja. Después de emitir esta instrucción, las instrucciones `plot` y `plotxy` mueven el lápiz, pero no dibujan nada. Una vez que el bolígrafo esté donde lo desee, use `plot-pen-down` para volver a colocar el lápiz en posición de graficar efectivamente.

Si desea graficar puntos individuales en lugar de líneas, o si desea dibujar barras en lugar de líneas o puntos, debe cambiar el "modo" del ploteo de la gráfica. Hay tres modos disponibles: línea, barra y punto. Line (línea) es el modo predeterminado.

Normalmente, se cambia el modo de una pluma editando el gráfico. Esto cambia el modo predeterminado del lápiz. También es posible cambiar el modo de la pluma temporalmente usando la instrucción `set-plot-pen-mode`. Esa instrucción toma un número como entrada: 0 para la línea, 1 para la barra, 2 para el punto.

Histogramas

Un histograma es un tipo especial de diagrama que mide con qué frecuencia ciertos valores, o valores en ciertos rangos, ocurren en una colección de números que surgen en su modelo.

Por ejemplo, supongamos que las tortugas en su modelo tienen una variable edad. Se podría crear un histograma de la distribución de edades entre las tortugas con la instrucción de histograma, así:

```
histogram [age] of turtles
```

Los números que requieran un histograma no tienen que venir de un conjunto de agentes; podrían ser cualquier lista de números.

Tenga en cuenta que el uso de la instrucción de histograma no cambia automáticamente el lápiz graficador actual al modo de barra. Si se desea barras, debe configurarse el lápiz graficador en modo de barra. (Como dijimos antes, puede cambiar el modo predeterminado de un lápiz editando el gráfico en la pestaña Ejecutar).

Al igual que otros tipos de gráficos, los histogramas se pueden configurar en escala automática. Sin embargo, los histogramas a escala automática no se redimensionan automáticamente de forma horizontal como lo hacen otros tipos de gráficos. Para establecer el rango mediante programación, puede usar la primitiva `set-plot-x-range`.

El ancho de las barras en un histograma está controlado por el intervalo del lápiz graficador. Se puede establecer el intervalo predeterminado editando el gráfico en la pestaña Ejecutar. También se puede cambiar el intervalo temporalmente con la instrucción `set-plot-pen-interval` o `set-histogram-num-bars`. Si usa esta última instrucción, NetLogo configurará el intervalo de manera apropiada para ajustarse al número especificado de barras dentro del rango x actual de la gráfica.

Código Ejemplo: Histogram Example

Limpiando y Reseteando

Puede borrar el gráfico actual con la instrucción `clear-plot` o borrar todos los gráficos de su modelo con `clear-all-plots`. La instrucción `clear-all` también borra todos los gráficos, además de borrar todo lo demás en su modelo.

Si desea eliminar solo los puntos que un lápiz específico ha dibujado, use `plot-pen-reset`.

Cuando se borra todo un gráfico, o cuando se reinicia un lápiz, eso no solo elimina los datos que se han graficado. También restaura el gráfico o el lápiz a su configuración predeterminada, tal como se especificaron en la pestaña Ejecutar cuando se creó o se editó por última vez. Por lo tanto, los efectos de tales instrucciones como `set-plot-background-color`, `set-plot-x-range` y `set-plot-pen-color` son solo temporales.

Rangos y Escalado Automático (Auto Escala)

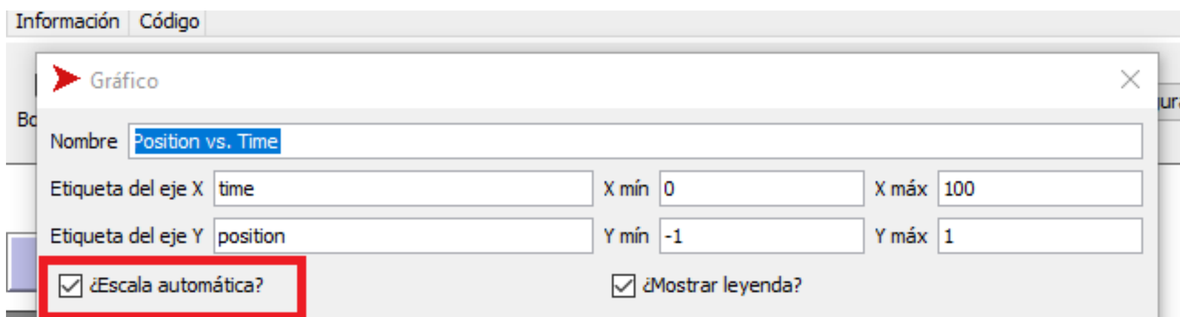
Los rangos predeterminados de x e y para un gráfico son números fijos, pero se pueden cambiar en el momento de la configuración o cuando se ejecuta el modelo.

Para cambiar los rangos en cualquier momento, use `set-plot-x-range` y `set-plot-y-range`. O bien, puede dejar que los rangos crezcan automáticamente. De cualquier manera, cuando se borre el gráfico, los rangos volverán a sus valores predeterminados.

De forma predeterminada, todos los diagramas de NetLogo tienen habilitada la función de escalado automático. Esto significa que si el modelo intenta graficar un punto que está fuera del rango visualizado actual, el rango de la gráfica crecerá a lo largo de uno o ambos ejes para que el nuevo punto sea visible. Las gráficas de histograma, sin embargo, no se escalan automáticamente de forma horizontal.

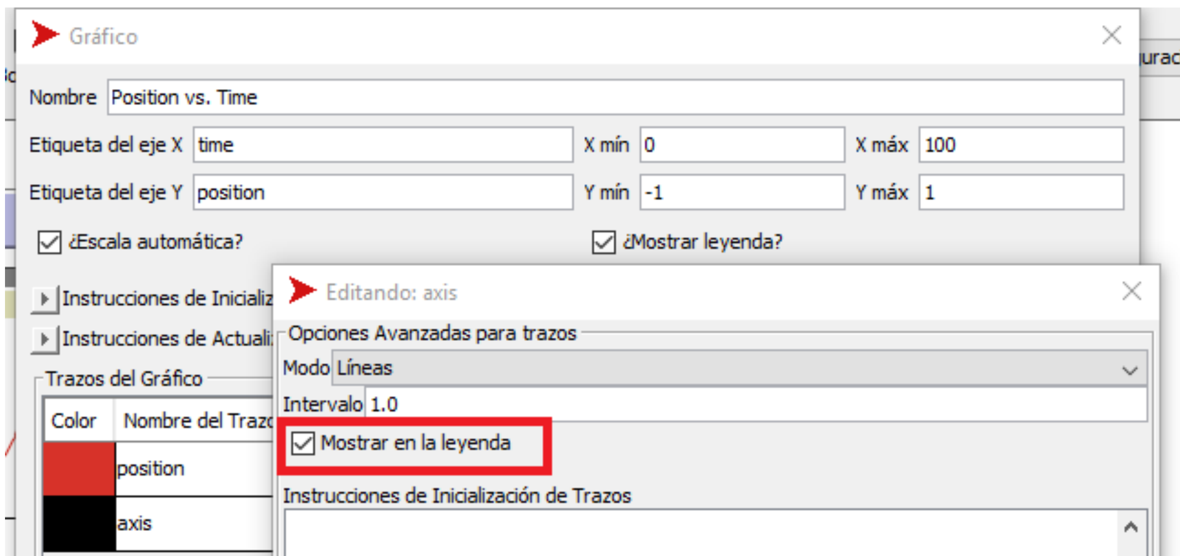
Con la esperanza de que los rangos no tengan que cambiar cada vez que se agrega un nuevo punto, cuando los rangos crecen, dejan algo de espacio extra: 25% si crece horizontalmente, 10% si crece verticalmente.

Si desea desactivar esta función, edite la gráfica y desmarque la casilla "¿Escala automática?" como aparece en la siguiente imagen abajo. Actualmente, no es posible activar o desactivar esta función solo en un eje; siempre se aplica a ambos ejes.



Utilizando una Leyenda

Se puede mostrar la leyenda de un gráfico marcando la casilla "¿Mostrar leyenda?", que se encuentra en el lado derecho inferior en la imagen anterior dentro del cuadro de diálogo de edición. Si no desea que aparezca un lápiz en particular en la leyenda, se puede desmarcar la casilla de verificación "Mostrar en la leyenda" para ese lápiz (ver imagen siguiente) en la configuración avanzada del lápiz de gráfico (la configuración avanzada del lápiz de gráfico se puede abrir haciendo clic en el botón del lápiz en la tabla de lápices del cuadro de edición del gráfico).



Lápices de Gráfico Temporal

La mayoría de los gráficos pueden obtenerse con un número fijo de lápices. Pero algunos gráficos tienen necesidades más complejas; pueden necesitar que el número de lápices varíe según las condiciones. En tales casos, se puede hacer lápices de gráfico "temporales" a partir del código y luego trazar con ellos. Estos lápices se llaman "temporales" porque se desvanecen cuando se borra el gráfico o trazado (por medio de las instrucciones `clear-plot`, `clear-all-plots`, o `clear-all`).

Para crear un lápiz de gráfico temporal, use la instrucción `create-temporary-plot-pen`. Normalmente, esto se haría en la pestaña Código, pero también es posible usar esta instrucción desde la configuración de gráfico o el código de actualización de gráfico (en el cuadro de diálogo de edición). De forma predeterminada, el nuevo bolígrafo está abajo (listo para dibujar), es de color negro, tiene un intervalo de 1 y traza en modo línea. Las instrucciones están disponibles para cambiar todas estas configuraciones; vea la sección Gráfico del Diccionario de NetLogo.

Antes de que pueda usar el lápiz, tendrá que usar las instrucciones `set-current-plot` y `set-current-plot-pen`. Estos se explican en la siguiente sección.

set-current-plot and set-current-plot-pen

Antes de NetLogo 5, no era posible poner instrucciones de trazado en el gráfico en sí mismo. Todo el código de la gráfica se escribía en la pestaña Código con el resto del código. Para compatibilidad con versiones anteriores, y para lapiceros temporales, esto todavía es compatible. Los modelos de versiones anteriores de NetLogo (y los que usan lápices de gráfico temporales) tienen que indicar explícitamente qué trazado es el gráfico actual con la instrucción `set-current-plot` y cuál es el lápiz actual con la instrucción `set-current-plot-pen`.

Para establecer la gráfica actual, use la instrucción `set-current-plot` con el nombre del gráfico entre comillas dobles, como en el siguiente ejemplo:

```
set-current-plot "Distance vs. Time"
```

El nombre del gráfico debe ser exactamente como fue escrito cuando se creó el gráfico. Tenga en cuenta que más adelante si cambia el nombre del gráfico, también deberá actualizar las llamadas `set-current-plot` en su modelo para usar el nuevo nombre. (Copiar y pegar puede ser útil aquí).

Para un diagrama con múltiples lápices, se puede especificar manualmente con qué bolígrafo se desea graficar. Si no especifica un bolígrafo, el gráfico se realizará con el primer bolígrafo en el diagrama. Para graficar con un bolígrafo diferente, se utiliza la instrucción `set-current-plot-pen` con el nombre del bolígrafo entre comillas dobles, como este:

```
set-current-plot-pen "distance"
```

Una vez que se establece el lápiz actual, se pueden ejecutar instrucciones `plot count turtles` para ese bolígrafo. Los modelos más antiguos con gráficos solían tener su propio procedimiento de trazado de planos (`do-plotting`) que se parecía a esto:

```
to do-plotting
  set-current-plot "populations"
  set-current-plot-pen "sheep"
  plot count sheep
  set-current-plot-pen "wolves"
  plot count wolves

  set-current-plot "next plot"
  ...
end
```

Una vez más, esto ya no es necesario en NetLogo 5, a menos que esté utilizando lápices de gráfico temporales.

Conclusion

No todos los aspectos del sistema de gráficos de NetLogo se ha explicado aquí. Consulte la sección Gráfico del Diccionario de NetLogo para obtener información sobre instrucciones y reporteros adicionales relacionados con graficar.

Muchos de los Modelos de Muestra en la Biblioteca de Modelos ilustran varias técnicas avanzadas de trazado. También puede dar un vistazo a los siguientes ejemplos de código:

Código Ejemplo: Plot Axis Example, Plot Smoothing Example, Rolling Plot Example

Cadenas (Strings)

Las cadenas pueden contener cualquier carácter Unicode.

Para ingresar una cadena constante en NetLogo, rodéelo con comillas dobles.

La cadena vacía se escribe poniendo nada entre comillas, como esta: "".

La mayoría de las listas primitivas también funcionan en cadenas:

```
but-first "string" => "tring"
but-last "string" => "strin"
empty? "" => true
empty? "string" => false
first "string" => "s"
item 2 "string" => "r"
last "string" => "g"
length "string" => 6
member? "s" "string" => true
member? "rin" "string" => true
member? "ron" "string" => false
position "s" "string" => 0
position "rin" "string" => 2
position "ron" "string" => false
remove "r" "string" => "sting"
```

```
remove "s" "strings" => "tring"
replace-item 3 "string" "o" => "strong"
reverse "string" => "gnirts"
```

Algunas primitivas son específicas de cadena, como `is-string?`, `subserie` y `word`:

```
is-string? "string" => true
is-string? 37 => false
substring "string" 2 5 => "rin"
word "tur" "tle" => "turtle"
```

Las cadenas se pueden comparar utilizando los operadores `=, !=, <,>, <= and > =.`

Si necesita incrustar un carácter especial en una cadena, use las siguientes secuencias de escape:

- `\n` = newline
- `\t` = tab
- `\"` = double quote
- `\\` = backslash

Salida

Esta sección trata acerca de la salida a la pantalla. La salida a la pantalla también se puede guardar más tarde en un archivo usando la instrucción `export-output`. Si necesita un método más flexible para escribir datos en archivos externos, consulte la siguiente sección, File I /O.

Las instrucciones básicas para generar salida a la pantalla en NetLogo son `print`, `show`, `type` y `write`. Estas instrucciones envían su salida al Terminal de Instrucciones.

Para obtener detalles completos sobre estas cuatro instrucciones, consulte sus entradas en el diccionario de NetLogo. Así es como se usan típicamente:

- `print` es útil en la mayoría de las situaciones.
- `show` le permite ver qué agente está imprimiendo qué.
- `type` le permite imprimir varias cosas en la misma línea.
- `write` le permite imprimir valores en un formato que puede leerse nuevamente usando la lectura de archivos.

Un modelo de NetLogo puede tener opcionalmente un "área de salida" en su pestaña Ejecutar, separada del Terminal de Instrucciones. Para enviar la salida allí en lugar del Terminal de Instrucciones, use las instrucciones `output-print`, `output-show`, `output-type` y `output-write`.

El área de salida se puede borrar con la instrucción `clear-output` y guardarse en un archivo con `export-output`. El contenido del área de salida se guardará con la instrucción `export-world`. La instrucción `import-world` borrará el área de salida y establecerá su contenido en el valor del archivo world importado. Se debe tener en cuenta que grandes cantidades de datos que se envían al área de salida pueden aumentar el tamaño de sus mundos exportados.

Si utiliza `output-print`, `output-show`, `output-type`, `output-write`, `clear-output` o `export-output` en un modelo que no tiene un área de salida separada, las instrucciones se aplican a la porción de salida del Centro de Comando.

Cómo difieren las Primitivas de Salida

Esta información es una referencia rápida para usuarios más avanzados.

Las primitivas `print`, `show`, `type` y `write` difieren en los siguientes aspectos:

- ¿Qué tipos de valores acepta la primitiva?
- ¿Hace la primitiva una nueva línea al final?
- ¿Las salidas de cadenas tienen comillas que las rodean?
- ¿La salida de la primitiva corresponde al agente que la imprimió?

La siguiente tabla resume el comportamiento de cada primitiva.

Primitiva	Valor Aceptables	Agrega Nuevas Líneas	Cadenas entre Comillas	Salidas (de sí misma)
`print`	cualquier valor NetLogo	Sí	No	No
`show`	cualquier valor NetLogo	Sí	Sí	Sí
`type`	cualquier valor NetLogo	No	No	No
`write`	booleano, número, cadena, listas conteniendo estos tipos de datos	No	Sí	No

Archivo I/O

En NetLogo, hay un conjunto de primitivas que le proveen la capacidad de interactuar con archivos externos. Todos comienzan con el archivo de prefijo **file-**.

Hay dos modos principales cuando se trata de archivos: leer y escribir. La diferencia es la dirección del flujo de datos. Cuando lee información de un archivo, los datos que se almacenan en el archivo fluyen a su modelo. Por otro lado, la escritura permite que los datos fluyan fuera de su modelo y hacia un archivo.

Cuando trabaje con archivos, siempre comience usando la primitiva `file-open`. Esto especifica con qué archivo interactuará. Ninguna de las otras primitivas funciona a menos que se abra un archivo primero.

La siguiente primitiva **file-** que use dicta en qué modo estará el archivo hasta que se cierre el archivo, leyendo o escribiendo. Para cambiar de modo, cierre y vuelva a abrir el archivo.

Las primitivas de lectura incluyen: `file-read`, `file-read-line`, `file-read-characters` y `file-at-end?` Tenga en cuenta que el archivo ya debe existir antes de poder abrirlo para leerlo.

Código Ejemplo: File Input Example

Las primitivas para escribir son similares a las primitivas que imprimen cosas en el Terminal de Instrucciones, excepto que la salida se guarda en un archivo. Incluyen `file-print`, `file-show`, `file-type` y `file-write`. Tenga en cuenta que nunca puede "sobrescribir" datos. En otras palabras, si intenta escribir en un archivo con datos existentes, todos los datos nuevos se agregarán al final del archivo. (Si desea sobrescribir un archivo, use el `file-delete` para eliminarlo, luego ábralo para escribir).

Código Ejemplo: File Output Example

Cuando haya terminado de usar un archivo, puede usar la instrucción `file-close` para finalizar su sesión con el archivo. Si luego desea eliminar el archivo, use la primitiva `file-delete` para eliminarlo. Para cerrar múltiples archivos abiertos, primero se necesita seleccionar el archivo usando `file-open` antes de cerrarlo.

```
;; Open 3 files
file-open "myfile1.txt"
file-open "myfile2.txt"
file-open "myfile3.txt"

;; Now close the 3 files
file-close
file-open "myfile2.txt"
file-close
file-open "myfile1.txt"
file-close
```

O bien, si desea cerrar todos y cada uno de los archivos, puede usar `file-close-all`.

Dos aspectos primordiales que vale la pena mencionar son `file-write` y `file-read`. Estas primitivas están diseñadas para guardar y recuperar fácilmente las constantes de NetLogo, como números, listas, booleanos y cadenas. `file-write` siempre generará la variable de forma que la lectura de archivos pueda interpretarla correctamente.

```
file-open "myfile.txt"  ;; Opening file for writing
ask turtles
  [ file-write xcor file-write ycor ]
file-close

file-open "myfile.txt"  ;; Opening file for reading
ask turtles
  [ setxy file-read file-read ]
file-close
```

Código Ejemplo: File Input Example and File Output Example

Permitiendo que el Usuario Elija

Las primitivas `user-directory`, `user-file` y `user-new-file` son útiles cuando se desea que el usuario elija un archivo o directorio para que su código funcione.

Películas - Videos

Esta sección describe cómo capturar una película ".mp4" de un modelo de NetLogo.

Primero, use la instrucción `vid: start-recorder` para iniciar el video.

Para agregar un marco a su película, use ya sea `vid: record-view` o `vid: record-interface`, dependiendo de si desea que la película muestre solo la vista actual, o toda la pestaña Ejecutar. En una sola película, la resolución será una de las siguientes:

- La resolución especificada en la llamada a `vid: start-recorder width height` si se especificó la resolución. Estos son parámetros opcionales.
- La resolución de la vista si no especificó una resolución en la llamada a `vid: start-recorder` y llamar a `vid: record-view` antes de llamar a `vid: record-interface`

- La resolución de la interfaz si no especificó una resolución en la llamada a `vid: start-recorder` y llamar a `vid: record-interface` antes de llamar a `vid: record-view`

Tenga en cuenta que, si la resolución de una imagen grabada no coincide con la resolución de la grabación, se ajustará a escala, lo que puede dar como resultado imágenes que se ven borrosas o desenfocadas.

Cuando haya terminado de agregar marcos, use `vid: save-recording`. El nombre de archivo que proporcione debe terminar con .mp4, la extensión para películas codificadas en MP4 (reproducible en QuickTime y otros programas).

```
;; export a 30 frame movie of the view
extensions [vid]

;...

setup
vid:start-recorder
vid:record-view ;; show the initial state
repeat 30
[ go
  vid:record-view ]
vid:save-recording "out.mp4"
```

Una película se reproducirá a 25 fotogramas por segundo. Para hacer que la reproducción de la película sea más rápida o más lenta, considere usar una herramienta de posprocesamiento de video.

Para verificar si está grabando, llame a `vid: recorder-status`, que informa una cadena que describe el estado de la grabadora actual.

Para descartar la película que se está grabando, llame a `vid: reset-recorder`.

Código Ejemplo: Movie Example

Las películas generadas cuando se ejecuta sin cabezal, o por ejecuciones en segundo plano en un experimento BehaviorSpace paralelo pueden usar solo `vid: record-view`. Las películas generadas en la GUI de NetLogo también pueden usar `vid: record-interface` y `vid: record-source`.

Las películas de NetLogo se exportan como archivos MP4 con codificación H.264. Para reproducir una película MP4, puede usar VLC Player, una descarga gratuita de la organización VideoLAN.

Las películas pueden ocupar mucho espacio en disco. Es probable que desee comprimir sus películas con software de terceros. El software puede ofrecerle diferentes tipos de compresión. Algunos tipos de compresión son sin pérdida, mientras que otros son con pérdida. "Lossy" significa que, para hacer los archivos más pequeños, se pierde parte del detalle de la película. Dependiendo de la naturaleza de su modelo, es posible que desee evitar el uso de compresión con pérdida, por ejemplo, si la vista contiene detalles finos a nivel de píxel.

Perspectiva

La vista en 2D y 3D muestra el mundo desde la perspectiva del observador. Por defecto, el observador está mirando hacia abajo en el mundo desde el eje z positivo en el origen. Puede cambiar la perspectiva del observador usando las instrucciones de tortuga `follow`, `ride` y `watch` y las instrucciones `follow-me`, `ride-me` y `watch-me`. Cuando se está en modo follow o ride, el observador se mueve con el agente alrededor del mundo. La diferencia entre follow y ride solo está visible en la vista 3D. En la vista 3D, el usuario puede cambiar la distancia detrás del agente usando el mouse. Cuando el observador está siguiendo a una distancia cero del agente, en realidad está montando (como si fuese un jinete) al agente. Cuando el observador está en modo vigilancia, rastrea los movimientos de una tortuga sin moverse. En ambas

vistas, verá un reflector en el sujeto y en la vista 3D el observador se volverá para mirar al sujeto. Para determinar qué agente es el foco, puede usar el reportero `subject`.

Código Ejemplo: Perspective Example

Dibujando

El dibujo es una capa donde las tortugas pueden hacer marcas visibles.

En la vista, el dibujo aparece en la parte superior de las parcelas, pero debajo de las tortugas. Inicialmente, el dibujo está vacío y transparente.

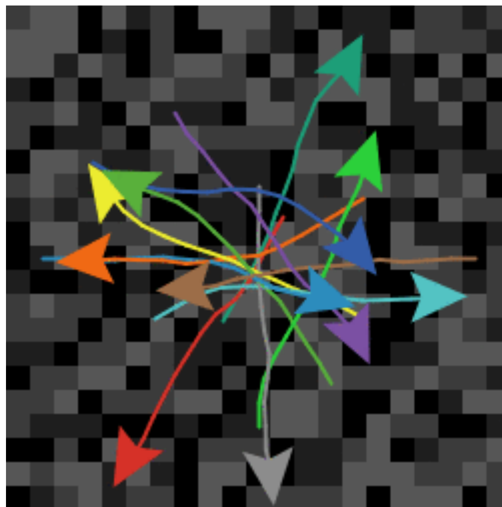
Puede ver el dibujo, pero las tortugas (y parcelas) no pueden. No pueden sentir el dibujo ni reaccionar ante él. El dibujo es solo para que la gente lo mire.

Las tortugas pueden dibujar y borrar líneas en el dibujo usando las instrucciones `pen-down` y `pen-erase`. Cuando el lápiz de una tortuga está abajo (o borrando), la tortuga dibuja (o borra) una línea detrás de ella cada vez que se mueve. Las líneas son del mismo color que la tortuga. Para dejar de dibujar (o borrar), use `pen-up`.

Las líneas dibujadas por las tortugas son normalmente de un píxel de grosor. Si desea un grosor diferente, establezca la variable de tortuga `pen-size` en un número diferente antes de dibujar (o borrar). En nuevas tortugas, la variable se establece en 1.

Las líneas hechas cuando una tortuga se mueve de una manera que no fija una dirección, como con `setxy` o `move-to`, se dibujará la línea de ruta más corta que obedece a la topología.

Aquí hay algunas tortugas que han hecho un dibujo sobre una grilla de parcelas sombreadas aleatoriamente. Observe cómo las tortugas cubren las líneas y las líneas cubren los colores del parche. El tamaño del bolígrafo utilizado aquí fue 2:



La instrucción `stamp` (sello) permite que una tortuga deje una imagen de sí misma en el dibujo y `stamp-erase` le permite eliminar los píxeles debajo de él en el dibujo.

Para borrar todo el dibujo, use la instrucción del observador `clear-drawing`. (También puede usar `clear-all`, que también borra todo lo demás).

Importando una Imagen

La instrucción de observador `import-drawing` le permite importar un archivo de imagen del disco al dibujo.

`import-drawing` es útil solo para proporcionar un telón de fondo para que las personas lo observen. Si desea que las tortugas y parcelas reaccionen a la imagen, debe usar `import-pcolors` o `import-pcolors-rgb` en su lugar.

Comparación con otros Logos

El dibujo funciona de forma algo diferente en NetLogo que algunos otros programas Logos.

Las diferencias notables incluyen:

- Los nuevos corrales de tortugas están arriba, no hacia abajo.
- En lugar de utilizar una instrucción de cercado `fence` para confinar a la tortuga dentro de los límites, en NetLogo edita el mundo y desactiva el envoltorio.
- No hay pantalla de color, `bgcolor` o `setbg`. Puede crear un fondo sólido al colorear las parcelas, por ejemplo, `ask patches [set pcolor blue]`.

Características de dibujo no compatibles con NetLogo:

- No hay instrucción `window`. Esto se usa en algunos otros Logos para dejar que la tortuga deambule en un plano infinito.
- No hay instrucción `flood` (inundación) o `fill` para llenar un área cerrada con color.

Topología

La forma en que el mundo de las parcelas está conectado puede cambiar. Por defecto, el mundo es un toro, lo que significa que no está delimitado, sino que se "envuelve", de modo que cuando una tortuga se mueve más allá del borde del mundo, desaparece y reaparece en el borde opuesto y cada parcela tiene el mismo número de parcelas “vecinas”. Si usted es una parcela al borde del mundo, algunos de sus "vecinos" están en el extremo opuesto.

Sin embargo, puede cambiar la configuración de ajuste con el botón Configuración. Si no se permite envolver en una dirección dada, entonces, en esa dirección (x o y) el mundo está limitado. Las parcelas a lo largo de ese límite tendrán menos de 8 vecinos y las tortugas no se moverán más allá del borde o límite del mundo.

La topología del mundo de NetLogo tiene cuatro valores potenciales: toro, caja, cilindro vertical o cilindro horizontal. La topología se controla habilitando o deshabilitando el ajuste en las direcciones x e y. El mundo predeterminado es un toro.

Un toro se enrolla en ambas direcciones, lo que significa que los bordes superior e inferior del mundo están conectados y los bordes izquierdo y derecho están conectados. Entonces, si una tortuga se mueve más allá del borde derecho del mundo, aparece de nuevo a la izquierda y lo mismo para la parte superior e inferior.

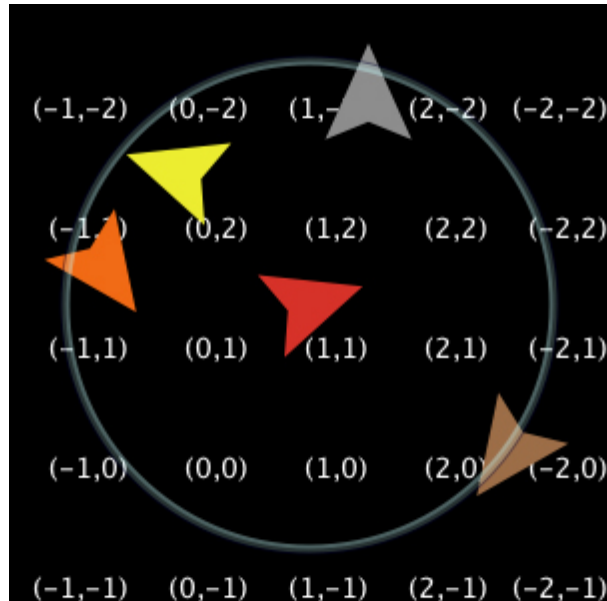
Una caja no se ajusta en ninguna dirección. El mundo está delimitado, por lo que las tortugas que intentan alejarse del mundo no pueden hacerlo. Tenga en cuenta que las parcelas alrededor del borde del mundo tienen menos de ocho vecinos; las esquinas tienen tres y el resto tienen cinco.

Los cilindros horizontales y verticales envuelven en una dirección, pero no en la otra. Un cilindro horizontal se enrolla verticalmente, por lo que la parte superior del mundo está conectada a la parte inferior. pero los bordes izquierdo y derecho están delimitados. Un cilindro vertical es lo opuesto; se envuelve horizontalmente para que los bordes izquierdo y derecho estén conectados, pero los bordes superior e inferior están delimitados.

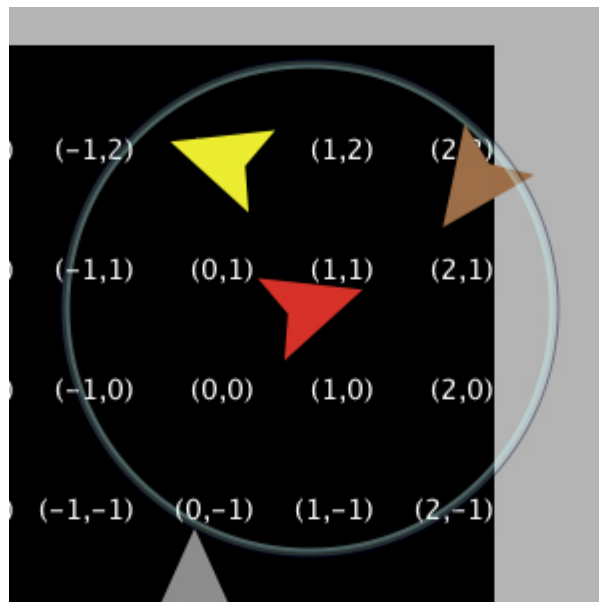
Código Ejemplo: Neighbors Example

Cuando las coordenadas se envuelven, las tortugas y los enlaces se envuelven visualmente en la vista también. Si una forma o enlace de tortuga se extiende más allá de un borde, parte de él aparecerá en el otro borde. (Las tortugas mismas son puntos que no ocupan espacio, por lo que no pueden estar en ambos lados del mundo a la vez, pero a la vista, parecen ocupar espacio porque tienen forma).

El envoltorio también afecta la apariencia de la vista cuando estás siguiendo a una tortuga. En un toro, donde quiera que vaya la tortuga, siempre verás todo el mundo a su alrededor:



Mientras que en una caja o cilindro el mundo tiene bordes, esto implica que las áreas más allá de esos bordes aparecen en la vista como grises:



Código Ejemplo: Termites Perspective Demo (torus), Ants Perspective Demo (box)

Los ajustes de topología también controlan el comportamiento de las primitivas de distancia (xy), in-radius, in-cone, face (xy) y towards (xy). La topología controla si las primitivas se envuelven o no. Siempre usan la ruta más corta permitida por la topología. Por ejemplo, la distancia desde el centro de las parcelas en la esquina inferior derecha (min-pxcor, min-pycor) y la esquina superior izquierda (max-pxcor, max-pycor) será la siguiente para cada topología dado que el mínimo y max pxcor y pycor son +/- 2:

- Torus - sqrt (2) ~ 1.414 (este será el mismo para todos los tamaños mundiales ya que las parcelas están directamente diagonales entre sí en un toro).
- Box - sqrt (ancho del mundo ^ 2 + altura del mundo ^ 2) ~ 7.07

- Vertical Cylinder - sqrt (altura del mundo ^ 2 + 1) ~ 5.099
- Horizontal Cylinder - sqrt (ancho del mundo ^ 2 + 1) ~ 5.099

Todas las demás primitivas actuarán de manera similar a la distancia. Si anteriormente usaba primitivas `no-wrap` en su modelo, le recomendamos que las elimine y cambie la topología del mundo.

Si su modelo tiene tortugas que se mueven, tendrá que pensar en lo que les sucederá cuando lleguen al borde del mundo, si la topología que está utilizando tiene algunos bordes que no se envuelven. Hay algunas opciones comunes: la tortuga se refleja de nuevo en el mundo (de forma sistemática o aleatoria), la tortuga sale del sistema (muere) o la tortuga está oculta. Ya no es necesario verificar los límites usando coordenadas de tortuga, en su lugar podemos preguntarle a NetLogo si una tortuga está en el borde del mundo. Hay un par de formas de hacerlo, la más simple es usar la primitiva `can-move?`

```
if not can-move? distance [ rt 180 ]
```

`can-move?` simplemente devuelve true (verdadero) si la distancia de posición en frente de la tortuga está dentro del mundo de NetLogo, de lo contrario es false (falso). En este caso, si la tortuga está en el borde del mundo, sencillamente vuelve al camino que vino. También puede usar `patch-ahead 1! = nobody` en lugar de `can-move?` Si necesita hacer algo más inteligente que simplemente darse la vuelta, puede ser útil usar `patch-at` con `dx` y `dy`.

```
if patch-at dx 0 = nobody [
  set heading (- heading)
]
if patch-at 0 dy = nobody [
  set heading (180 - heading)
]
```

Esto prueba si la tortuga está golpeando una pared horizontal o vertical y rebota en esa pared.

En algunos modelos, si una tortuga no puede avanzar, simplemente muere (sale del sistema, como en Conductor o Mousetraps).

```
if not can-move? distance [ die ]
```

Si está moviendo tortugas usando `setxy` en lugar de `forward`, debe probar para asegurarse de que la parcela a la que está a punto de moverse existe ya que `setxy` arroja un error de tiempo de ejecución si se le asignan coordenadas fuera del mundo. Esta es una situación común cuando el modelo está simulando un plano infinito y las tortugas fuera de la vista simplemente deben ocultarse.

```
let new-x new-value-of-xcor
let new-y new-value-of-ycor

ifelse patch-at (new-x - xcor) (new-y - ycor) = nobody
  [ hide-turtle ]
  [ setxy new-x new-y
    show-turtle ]
```

Varios modelos en la Biblioteca de Modelos usan esta técnica, Gravitation, N-Bodies y Electrostatics son buenos ejemplos.

Las instrucciones `diffuse` y `diffuse4` se comportan correctamente en todas las topologías. Cada parcela difunde e iguala la cantidad de la variable difusa a cada uno de sus vecinos, si tiene menos de 8 vecinos (o 4 si usa `diffuse4`), el resto permanece en la parcela de difusión. Esto significa que la suma global de la variable de parcela en todo el mundo permanece constante. Sin embargo, si desea que la materia difusa siga cayendo de los bordes del mundo como lo haría en un plano infinito, debe limpiar los bordes en cada paso como en el Ejemplo Diffuse Off Edges Example.

Enlaces

Al hablar de enlaces o conexiones es importante entender el lenguaje. Cuando se conectan dos o más objetos (personas, animales, computadores, células, etc.), cada uno de los objetos constituye un nodo, y es una de las partes de un sistema conectado. Así, los sistemas se forman de nodos y de conexiones entre dichos nodos. En NetLogo los nodos pueden ser las tortugas, e incluso un conjunto de tortugas, que es la clase o especie genérica que se utiliza dentro del software para representar un objeto de estudio y los enlaces constituyen la conexión entre ellos.

Un enlace es un agente que conecta dos tortugas. Estas tortugas a veces también se llaman nodos.

El enlace siempre se dibuja como una línea entre las dos tortugas. Los enlaces no tienen una ubicación como las tortugas, no se consideran en ninguna parcela y no se puede encontrar la distancia desde un enlace a otro punto.

Hay dos designaciones de enlace: no dirigido y dirigido. Un enlace dirigido está fuera de, o desde, un nodo y hacia, o para, otro nodo. La relación de un padre con un hijo se puede modelar como un enlace dirigido. Un enlace no dirigido aparece igual para ambos nodos, cada nodo tiene un enlace con otro nodo. La relación entre cónyuges o hermanos podría modelarse como un enlace no dirigido.

Hay un conjunto de agentes global de todos los enlaces, al igual que con las tortugas y parcelas. Se pueden crear enlaces no dirigidos utilizando las instrucciones `create-link-with` y `create-links-with`; y enlaces dirigidos utilizando las instrucciones `create-link-to`, `create-links-to`, `create-link-from` y `create-links-from`. Una vez que se ha creado el primer enlace dirigido o no dirigido, todos los enlaces no familiares deben coincidir (los enlaces también admiten familias, al igual que las tortugas, que se analizarán en breve); es imposible tener dos enlaces sin autorizar, donde uno es dirigido y el otro no está dirigido. Se produce un error de tiempo de ejecución si intenta hacerlo. (Si mueren todos los enlaces sin autorizar, puede crear enlaces de esas familias que sean diferentes en su designación de los enlaces anteriores).

En general, las primitivas de nombres de enlace indican qué tipo de enlaces son:

- Las primitivas que tienen "`out`" en su nombre utilizan enlaces salientes y no dirigidos. Puede pensar en estos como "los enlaces que puedo usar para llegar desde el nodo actual a otros nodos". En general, estos son probablemente las primitivas que desea usar.
- Las primitivas que tienen "`in`" en su nombre utilizan enlaces entrantes y no dirigidos. Puedes pensar en estos como "los enlaces que puedo usar para llegar al nodo actual desde otros nodos".
- Primitivas que no especifican "`in`" o "`out`", o que tienen "`with`" en su nombre utilizan todos los enlaces, tanto no dirigidos como dirigidos, entrantes y salientes.

Las variables `end1` y `end2` de un enlace contienen las dos tortugas conectadas por el enlace. Si el enlace está dirigido, va de `end1` a `end2`. Si el enlace no está dirigido, `end1` es siempre la más antigua de las dos tortugas, es decir, la tortuga con el número identificador (who) más pequeño.

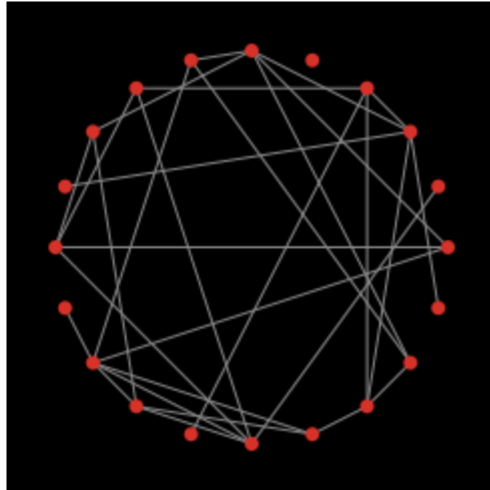
Las familias de enlaces, como las familias de tortugas, le permiten definir diferentes tipos de enlaces en su modelo. Las familias de enlaces deben ser dirigidas o no dirigidas, a diferencia de los enlaces no creados, esto se define en tiempo de compilación en lugar de tiempo de ejecución. Usted declara las familias de enlace usando las palabras clave `undirected-link-breed` y `directed-link-breed`. Los enlaces familiares se pueden crear usando las instrucciones `create-<breed>-with` y `create-<breeds>-with` para las familias no dirigidas y las instrucciones `create-<breed>-to`, `create-<breeds>-to`, `create-<breed>-from`, y `create-<breeds>-from` para enlaces dirigidos.

No puede haber más de un enlace no dirigido de la misma familia (o más de un enlace no familiar no dirigido) entre un par de agentes, ni más de un enlace dirigido de la misma familia en la misma dirección entre un par de agentes. Puede

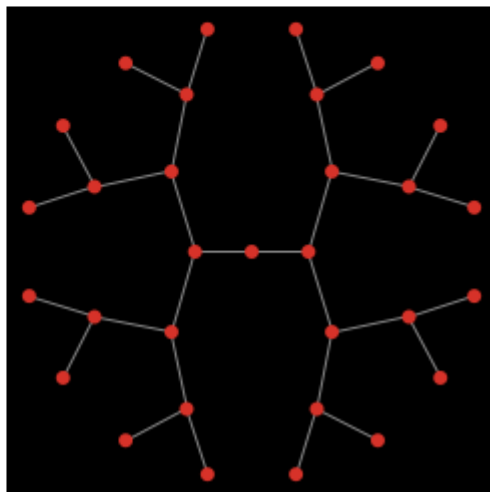
tener dos enlaces dirigidos de la misma familia (o dos enlaces dirigidos sin reproducir) entre un par si están en direcciones opuestas.

Diseño, Disposición (Layout)

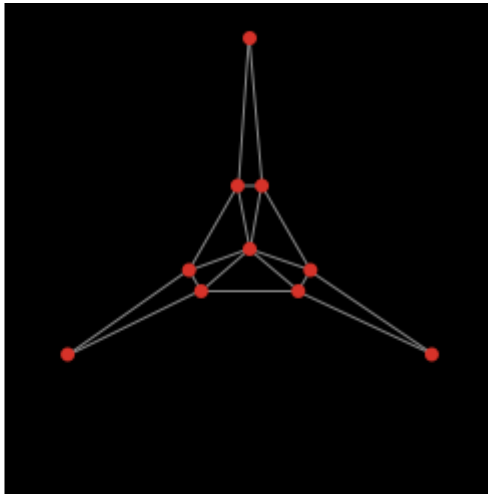
Como parte de nuestro soporte de red, también hemos agregado varias primitivas diferentes que le ayudarán a visualizar las redes. El más simple es el círculo de diseño que distribuye uniformemente a los agentes alrededor del centro del mundo con un radio dado.



`layout-radial` es un buen diseño si tiene algo así como una estructura de árbol, aunque incluso si hay algunos ciclos en el árbol, seguirá funcionando, aunque a medida que haya más y más ciclos, probablemente no se verá tan bien. `layout-radial` toma un agente raíz para que el nodo central lo coloque en el punto de coordenadas (0,0) y organiza los nodos conectados a él en un patrón concéntrico. Los nodos a un grado de la raíz se organizarán en un patrón circular alrededor del nodo central y el siguiente nivel alrededor de esos nodos, y así sucesivamente. `layout-radial` intentará dar cuenta de los gráficos asimétricos y dará más espacio a las ramas que son más anchas. `layout-radial` también toma una familia como entrada, por lo que utiliza una clase de enlaces para diseñar la red y no otra.



Dado un conjunto de nodos de anclaje, `layout-tutte` coloca a todos los otros nodos en el centro de masa de los nodos a los que está vinculado. El conjunto de anclaje se organiza automáticamente en una disposición circular con un radio definido por el usuario y los otros nodos convergerán en su lugar (esto, por supuesto, significa que deberá ejecutarlo varias veces antes de que el diseño sea estable).



`layout-spring` es útil para muchos tipos de redes. El inconveniente es que es relativamente lento ya que se necesitan muchas iteraciones para converger. En este diseño, los enlaces actúan como resortes que atraen los nodos que conectan entre sí y los nodos se repelen entre sí. La intensidad de las fuerzas se controla mediante entradas a las primitivas. Estas entradas siempre tendrán un valor entre 0 y 1; tenga en cuenta que los cambios muy pequeños aún pueden afectar la apariencia de la red. Los resortes también tienen una longitud (en unidades de parcela), sin embargo, debido a todas las fuerzas involucradas, los nodos no terminarán exactamente a esa distancia el uno del otro.

****Código Ejemplo: **Network Example, Network Import Example, Giant Component, Small Worlds, Preferential Attachment**

Procedimientos Anónimos

Los procedimientos anónimos permiten almacenar código para que se ejecute más tarde. Al igual que los procedimientos regulares de NetLogo, un procedimiento anónimo puede ser una instrucción (instrucción anónima) o un reportero (reportero anónimo).

Los procedimientos anónimos son valores, lo que significa que pueden pasarse como entrada, informarse como resultado o almacenarse en una variable.

Un procedimiento anónimo podría ejecutarse una vez, varias veces o no ejecutarlo.

En otros lenguajes de programación, los procedimientos anónimos se conocen como funciones de primera clase, cierres o lambda.

Primitivas de Procedimientos Anónimos

Las primitivas específicas de los procedimientos anónimos son `->`, `is-anonymous-command?`, y `is-anonymous-reporter?`

El símbolo `->` crea un procedimiento anónimo. El procedimiento anónimo que informa podría ser una instrucción o un informador, dependiendo del tipo de bloque sobre el cual sea pasado. Por ejemplo `[-> fd 1]` informa una instrucción anónima, porque `fd` es una instrucción, mientras que `[-> count turtles]` informa un reportero anónimo, porque el recuento es un reportero.

La instrucción de ejecución `run` acepta instrucciones anónimas, así como cadenas.
El reportero de ejecución `runresult` acepta reporteros anónimos y cadenas.
`run` y `runresult` permiten pasar entradas a un procedimiento anónimo. Al igual que con todas las primitivas que aceptan un número variable de entradas, la llamada completa debe estar rodeada de paréntesis, por ejemplo (`run my-`

`anonymous-command 5`) o (`runresult my-anonymous-reporter "foo" 2`). Cuando no se pasa la entrada, no se requieren paréntesis.

Entradas en Procedimientos Anónimos

Un procedimiento anónimo puede tomar cero o más entradas. Las entradas se referencian a las variables declaradas antes de la flecha. Por ejemplo, en el reportero anónimo `[[a b] -> a + b]`, a y b son entradas.

Procedimientos Anónimos y Cadenas

Crear y ejecutar procedimientos anónimos es rápido. Utilizar `run` o `runresult` en una nueva cadena por primera vez es aproximadamente 100 veces más lento que ejecutar un procedimiento anónimo. Los modeladores normalmente deben usar procedimientos anónimos en lugar de ejecutar cadenas, excepto cuando ejecutan cadenas ingresadas por el usuario.

Sintaxis Concisa

Los usos simples de `foreach`, `map`, `reduce`, `filter`, `n-values` y `sort-by` se pueden escribir con una sintaxis especialmente concisa. Puede escribir:

```
map abs [1 -2 3 -4]
;; => [1 2 3 4]
reduce + [1 2 3 4]
;; => 10
filter is-number? [1 "x" 3]
;; => [1 3]
foreach [1 2 3 4] print
;; prints 1 through 4
```

En las versiones anteriores de NetLogo (4 y anteriores), estas tenían que escribirse:

```
map [abs ?] [1 -2 3 -4]
;; => [1 2 3 4]
reduce [?1 + ?2] [1 2 3 4]
;; => 10
filter [is-number? ?] [1 "x" 3]
;; => [1 3]
foreach [1 2 3 4] [ print ? ]
;; prints 1 through 4
```

Procedimientos Anónimos como Cierres

Los procedimientos anónimos son "cierres"; eso significa que capturan o "cierran" las vinculaciones (no solo los valores actuales) de las variables locales y las entradas de procedimiento. No capturan las variables del agente y no capturan la identidad (o incluso el tipo de agente) del agente actual.

Salidas No Locales

Las instrucciones `stop` y `report` salen del procedimiento de inclusión dinámica, no del procedimiento anónimo adjunto. (Esto es compatible con versiones anteriores de NetLogo).

Procedimientos Anónimos y Extensiones

La Interfaz de Programación de Aplicaciones (API en inglés) de extensiones admite primitivas de escritura que aceptan procedimientos anónimos como entrada. Escríbanos para el código de muestra.

Limitaciones

Esperamos abordar al menos algunas de las siguientes limitaciones en futuras versiones de NetLogo:

- `import-world` no admite procedimientos anónimos.
- Los procedimientos anónimos no pueden ser variados (acepta un número variable de entradas).
- Los reporteros anónimos no pueden contener instrucciones, solo una única expresión del reportero. Entonces, por ejemplo, se debe usar `ifelse-value` y no `if`, y no usa `report` del todo. Si su código es demasiado complejo para ser escrito como un reportero, tendrá que mover el código a un procedimiento de reportero por separado, y luego llamar a ese procedimiento desde su reportero anónimo, pasándole las entradas necesarias.
- Los procedimientos anónimos no son intercambiables con los bloques de instrucciones y bloques informadores. Solo las primitivas mencionadas anteriormente aceptan procedimientos anónimos como entrada. Las primitivas de control como `ifelse` y `while` y las primitivas de agente como `of` y `with` no aceptan procedimientos anónimos. Así, por ejemplo, si tengo un reportero anónimo, `r [-> if random 2 == 0]` y dos instrucciones anónimas dejan que `c1 [-> tick]` y `let c2 [-> stop]`, no se puede escribir `ifelse r c1 c2`, sino escribir `ifelse runresult r [run c1] [run c2]`.
- La sintaxis concisa donde `->` puede omitirse solo está disponible para primitivas y primitivas de extensión, no procedimientos ordinarios. Entonces, por ejemplo, si tengo un procedimiento `p` que acepta un procedimiento anónimo como entrada, debe ser llamado, por ejemplo, `p [-> ...]` y no `p [...]`.

¿Qué es Opcional?

Hay varias formas diferentes de escribir procedimientos anónimos que permiten a los usuarios omitir una parte o la totalidad de la sintaxis del procedimiento anónimo. Estos se resumen en la tabla a continuación.

¿Cómo es el procedimiento anónimo?	**¿Qué se puede omitir?**	**Ejemplos**

El procedimiento anónimo es un solo una primitiva	• nombres de entrada
	• flecha
	• bloques de corchetes
		`foreach mylist stamp ; no inputs`
		
		`foreach mylist print ; single input`
		
		`(foreach xs ys setxy) ; multiple inputs`
		
		`map round [1.3 2.4 3.5] ; reporter, single input`
		
		`(map + [1 2 3] [4 5 6]) ; reporter, multiple inputs`

El procedimiento anónimo no toma entradas	• nombres de entrada
	• flecha
		`foreach mylist [ print "abc" ]`
		
		`map [ 4 ] mylist`

El procedimiento anónimo tiene cero o una entrada (s)	corchetes alrededor de los nombres de entrada	`foreach mylist [ -> stamp ] ; no inputs`
		
		`foreach mylist [ x -> print x ] ; single input`

		foreach mylist [x -> rt x fd x] ; multiple primitives, single input map [-> world-width] mylist ; reporter, no inputs map [x -> x ^ 2] mylist ; reporter, single input
El procedimiento anónimo requiere más de una entrada	nada	(foreach xs ys [[x y] -> setx x + y]) (map [[x y] -> x mod round y] xs ys)

Nota: siempre se requieren corchetes alrededor de los nombres de entrada en NetLogo 6.0.0. Si copia y pega código en NetLogo 6.0.0 utilizando procedimientos anónimos con nombres de entrada sin corchetes, el código no se compilará hasta que agregue los corchetes.

Código Ejemplo: State Machine Example

Solicitud Concurrente

NOTA: la siguiente información se incluye solo por compatibilidad con versiones anteriores. No recomendamos el uso de la primitiva `ask-concurrent` en absoluto en los nuevos modelos.

En versiones muy antiguas de NetLogo, `ask` simulaba un comportamiento concurrente por defecto. Desde NetLogo 4.0 (2007), `ask` es serial, es decir, los agentes ejecutan las instrucciones dentro del `ask` una a la vez.

La siguiente información describe el comportamiento de la instrucción `ask-concurrent`, que se comporta de la misma manera en que se comportó el `ask` anterior.

`ask-concurrent` produce concurrencia simulada a través de un mecanismo de turnos. El primer agente toma un turno, luego el segundo agente toma un turno, y así sucesivamente hasta que cada agente en el conjunto de agentes solicitado ha tenido un turno. Luego volvemos al primer agente. Esto continúa hasta que todos los agentes hayan terminado de ejecutar todas las instrucciones.

El "turno" de un agente finaliza cuando realiza una acción que afecta el estado del mundo, como mover o crear una tortuga o cambiar el valor de una variable global, de tortuga, parcela o de enlace. (Establecer una variable local no cuenta).

Las instrucciones `forward` (`fd`) y `back` (`bk`) se tratan especialmente. Cuando se utilizan dentro de `ask-concurrent`, estas instrucciones pueden tomar varios turnos para ejecutarse. Durante su turno, la tortuga solo puede moverse en un paso. Por lo tanto, por ejemplo, `fd 20` es equivalente a repetir `20 [fd 1]`, donde el turno de la tortuga termina después de cada ejecución de `fd`. Si la distancia especificada no es un número entero, la última fracción del paso tarda un turno completo. Entonces, por ejemplo, `fd 20.3` es equivalente a repetir `20 [fd 1] fd 0.3`.

La instrucción de salto `jump` siempre toma exactamente un turno, independientemente de la distancia.

Para entender la diferencia entre `ask` y `ask-concurrent`, considere las siguientes dos instrucciones:

```
ask turtles [ fd 5 ]
ask-concurrent turtles [ fd 5 ]
```

Con `ask`, la primera tortuga avanza cinco pasos, luego la segunda tortuga avanza cinco pasos, y así sucesivamente.

Con `ask-concurrent`, todas las tortugas dan un paso adelante. Entonces todos dan un segundo paso, y así sucesivamente. Por lo tanto, la última instrucción es equivalente a:

```
repeat 5 [ ask turtles [ fd 1 ] ]
```

Código Ejemplo: Ask-Concurrent Example shows the difference between `ask` and `ask-concurrent`.

El comportamiento de `ask-concurrent` no siempre puede reproducirse tan fácilmente usando `ask`, como en este ejemplo. Considere este comando:

```
ask-concurrent turtles [ fd random 10 ]
```

Para obtener el mismo comportamiento usando `ask`, tendríamos que escribir:

```
turtles-own [steps]
ask turtles [ set steps random 10 ]
while [any? turtles with [steps > 0]] [
  ask turtles with [steps > 0] [
    fd 1
    set steps steps - 1
  ]
]
```

Para prolongar el "turno" de un agente, use la instrucción `without-interruption`. (Los bloques de instrucciones dentro de algunas instrucciones, como `create-turtles` y `hatch`, tienen una implícita `without-interruption` a su alrededor).

Tenga en cuenta que el comportamiento de `ask-concurrent` es completamente determinista. Dado el mismo código y las mismas condiciones iniciales, siempre ocurrirá lo mismo (si está utilizando la misma versión de NetLogo y comienza su ejecución de modelo con la misma semilla aleatoria).

En general, le sugerimos que no use `ask-concurrent` en absoluto. Si lo hace, le sugerimos que escriba su modelo para que no dependa de los detalles exactos de cómo funciona `ask-concurrent`. No garantizamos que su semántica siga siendo la misma en las versiones futuras de NetLogo, o que seguirá siendo compatible.

Interacción del Usuario con las Primitivas

NetLogo presenta varias primitivas que permiten que un modelo interactúe con el usuario. Estas primitivas incluyen `user-directory`, `user-file`, `user-new-file`, `user-input`, `user-message`, `user-one-of`, y `user-yes-or-no?`

Estas primitivas difieren exactamente en la interacción que toman con el usuario. `user-directory`, `user-file`, y `user-new-file` son todos reporteros que solicitan al usuario seleccionar un elemento del sistema de archivos e informar la ruta del elemento seleccionado a NetLogo. `user-yes-or-no?`, `user-one-of`, y `user-input`, todos solicitan al usuario que proporcione información en forma de texto o una selección. `user-message` simplemente presenta un mensaje al usuario.

Tenga en cuenta que todos los botones activos forever se detendrán cuando se use una de estas primitivas y se reanudarán solo cuando el usuario complete la interacción con el botón.

¿Qué significa Halt? (Detener)

Las primitivas que solicitan la entrada del usuario, así como el mensaje de usuario, proporcionan un botón "Detener". El efecto de este botón es el mismo para todas estas primitivas: detiene el modelo. Cuando se detiene el modelo, se detiene todo el código en ejecución, incluidos los botones y el centro de comando. Como la detención detiene el código en el medio de lo que estaba sucediendo en el momento en que se detuvo, es posible que vea resultados extraños si continúa ejecutando el modelo después de un alto sin configurarlo nuevamente.

Lazo o Atadura (Tie)

El lazo conecta dos tortugas para que el movimiento de una tortuga afecte la ubicación y el rumbo de otra. El lazo es una propiedad de los enlaces, por lo que debe haber un vínculo entre dos tortugas para crear una relación de lazo.

Cuando un `tie-mode` se establece en "fixed" o "free", el extremo1 (`end1`) y el extremo2 (`end2`) se unen. Si el enlace está dirigido, `end1` es el "agente raíz" y `end2` es el "agente rama". Esto es, cuando se mueve `end1` (usando `fd`, `jump`, `setxy`, etc.) `end2` también mueve la misma distancia y dirección. Sin embargo, cuando `end2` se mueve, no afecta a `end1`.

Si el enlace no está dirigido, es una relación de vínculo recíproco, es decir, si cualquiera de las tortugas se mueve, la otra también se moverá. Entonces, dependiendo de qué tortuga se mueva, cualquier tortuga puede considerarse la raíz o la rama. La tortuga raíz es siempre la tortuga que inicia el movimiento.

Cuando la tortuga de raíz gira hacia la derecha o hacia la izquierda, la tortuga rama gira alrededor de la tortuga raíz la misma cantidad que si las tortugas estuviesen rígidas. Cuando `tie-mode` se establece en "fixed", la orientación de la tortuga rama cambia en la misma cantidad. Si el `tie-mode` se establece en "free", la orientación de la tortuga de hoja no cambia.

El `tie-mode` de un enlace se puede configurar como "fixed" usando la instrucción `tie` y se establece en "`none`" (lo que significa que las tortugas ya no están atadas) usando `untie.` Para establecer el modo en "free" se necesita `set-tie-mode` "free".

Código Ejemplo: Tie System Example

Múltiples Archivos Fuente

La palabra clave `__includes` le permite usar varios archivos de origen en un solo modelo de NetLogo.

La palabra clave comienza con dos guiones bajos para indicar que la característica es experimental y puede cambiar en futuras versiones de NetLogo.

Cuando se abre un modelo que usa la palabra clave __includes, o si lo agrega a la parte superior de un modelo y presiona el botón Verificar (Check), el menú de inclusión aparecerá en la barra de herramientas. Desde el menú de inclusión, puede seleccionar de los archivos incluidos en este modelo.

Cuando abre archivos incluidos, aparecen en pestañas adicionales. Consulte la Guía de interfaz para más detalles.

Puede tener cualquier cosa en los archivos fuente externos (.nls) que normalmente pondría en la pestaña Código: `globals`, `breed`, `turtles-own`, `patches-own`, `breeds-own`, definiciones de procedimientos, etc. Tenga en cuenta que todas estas declaraciones comparten el mismo espacio de nombres, es decir, si se declara `my-global` en la pestaña Código, no puede declarar una variable global (o cualquier otra cosa) con el nombre `my-global` en cualquier archivo que esté incluido en el modelo. `my-global` será accesible desde todos los archivos incluidos. Lo mismo sería cierto si `my-global` fuera declarado en uno de los archivos incluidos.

Sintaxis

Colores

En la pestaña Código y en cualquier otro lugar de la interfaz de usuario de NetLogo, el código del programa está codificado por colores mediante el siguiente esquema:

- Las palabras clave son verdes
- Las constantes son anaranjadas
- Los comentarios son grises
- Las instrucciones primitivas son azules
- Los reporteros primitivos son morados
- Todo lo demás es negro

Aviso

El resto de esta sección contiene terminología técnica que puede no ser familiar para algunos lectores.

Palabras Clave

Las únicas palabras clave en el idioma son `globals`, `breed`, `turtles-own`, `to`, `to-report` y `end`, más `extensions` y la palabra clave experimental `__includes`. (Los nombres de las primitivas incorporadas no se pueden sombrear o redefinir, por lo que también son un tipo de palabra clave).

Identificadores

Todas las primitivas, nombres de variables globales y de agentes, y nombres de procedimientos comparten un solo espacio de nombres global insensible a mayúsculas y minúsculas; los nombres locales (las variables `let` y los nombres de las entradas de procedimiento) no pueden sobreponerse a nombres globales o entre sí. Los identificadores pueden contener cualquier letra o dígito Unicode y los siguientes caracteres ASCII:

```
.?=*!<>:#+/%$_^'&-
```

Algunos nombres de primitivas comienzan con dos guiones bajos para indicar que son experimentales y es probable que cambien o se eliminen en futuras versiones de NetLogo.

Alcance

NetLogo tiene un alcance léxico. Las variables locales (incluidas las entradas a los procedimientos) son accesibles dentro del bloque de instrucciones en el que están declaradas, pero no accesibles por los procedimientos llamados por esas instrucciones.

Comentarios

El carácter de punto y coma introduce un comentario, que dura hasta el final de la línea. No hay sintaxis de comentario de varias líneas.

Estructura

Un programa consta de declaraciones opcionales (globals, breed, turtles-own, patches-own, <BREED>-own, extensions) en cualquier orden, seguidas de cero o más definiciones de procedimientos. Se pueden declarar múltiples familias con declaraciones de familias separadas; las otras declaraciones pueden aparecer una sola vez.

Cada definición de procedimiento comienza con to o to-report, el nombre del procedimiento y una lista opcional entre corchetes de los nombres de entrada. Cada definición de procedimiento finaliza con end. Entre medio hay cero o más instrucciones.

Instrucciones y Reporteros

Las instrucciones toman cero o más entradas; las entradas son reporteros, que también pueden tomar cero o más entradas. No hay puntuación que separe o finalice las instrucciones; ninguna puntuación separa las entradas. Los identificadores deben estar separados por espacios en blanco o entre paréntesis o corchetes. (Entonces, por ejemplo, a + b es un identificador único, pero a (b [c] d) e contiene cinco identificadores).

Todas las instrucciones son prefijos. Todos los reporteros definidos por el usuario son prefijos. La mayoría de los reporteros primitivos son prefijos, pero algunos (operadores aritméticos, operadores booleanos y algunos operadores de conjuntos de agentes como with e in-points) son infijos.

Todas las instrucciones y reporteros, tanto primitivas como definidas por el usuario, toman un número fijo de entradas por defecto. (Es por eso que el lenguaje puede analizarse, aunque no haya signos de puntuación para separar o terminar instrucciones y / o entradas). Algunas primitivas son variadas, es decir, pueden tomar opcionalmente un número diferente de entradas que el predeterminado; paréntesis se utilizan para indicar esto, por ejemplo, (list 1 2 3) (ya que la primitiva list solo toma dos entradas por defecto). Los paréntesis también se usan para anular la precedencia predeterminada del operador, por ejemplo, (1 + 2) * 3, como en otros lenguajes de programación.

A veces, una entrada a una primitiva es un bloque de instrucciones (cero o más instrucciones dentro de corchetes) o un bloque informador (una única expresión de indicador entre corchetes). Los procedimientos definidos por el usuario no pueden tomar una instrucción o un bloque informador como entrada.

Las precedencias de los operadores son las siguientes, de mayor a menor:

- with, at-points, in-radius, in-cone
- (todas las otras primitivas y procedimientos definidos por el usuario)
- ^
- *, /, mod
- +, -
- <, >, <=, >=
- =, !=
- and, or, xor

Comparación con Otros Logos

No hay una definición estándar acordada de Logo; es una familia de idiomas suelta. Creemos que NetLogo tiene suficiente en común con otros logotipos para ganarse el nombre de Logo. Aún así, NetLogo difiere en algunos aspectos de la mayoría de los otros Logos. Las diferencias más importantes son las siguientes.

Diferencias Superficiales

La precedencia de los operadores matemáticos es diferente. Los operadores matemáticos infijos (como +, *, etc.) tienen menor prioridad que los reporteros con nombres. Por ejemplo, en muchos Logos, si escribe sin x + 1, se interpretará

como `sin (x + 1)`. NetLogo, por otro lado, lo interpreta como lo harían la mayoría de los otros lenguajes de programación, y la forma en que la misma expresión se interpretaría en notación matemática estándar, a saber, como `(sin x) + 1`.

Los reporteros `y` u `o` son formas especiales, no funciones normales, y "cortocircuitan", es decir, solo evalúan su segunda entrada si es necesario.

Los procedimientos solo se pueden definir en la pestaña Código, no interactivamente en el Centro de comando.

Los procedimientos de reportero, es decir, los procedimientos que "informan" (devuelven) un valor, se deben definir con `to-report` en lugar de hacerlo con `to`. La instrucción para informar un valor de un procedimiento de reportero es `report`, no `output`.

Al definir un procedimiento, las entradas al procedimiento deben estar entre corchetes, por ejemplo, `to square [x]`.

Los nombres de variables siempre se usan sin ningún tipo de puntuación: siempre `foo`, nunca con dos puntos, como esto `: foo` o bien, `"foo`. (Para que esto funcione, en lugar de una instrucción `make` tomando un argumento entre comillas, suministramos una forma especial `set` que no evalúa la primera entrada). como resultado, los procedimientos y las variables ocupan un solo espacio de nombres compartido.

Las últimas tres diferencias se ilustran en las siguientes definiciones de procedimientos:

Mayoría de Logos

```
to square :x
output :x * :x
end
```

NetLogo

```
to-report square [x]
report x * x
end
```

Diferencias Más Profundas

- Las variables locales de NetLogo y las entradas a los procedimientos tienen un alcance léxico, no tienen un alcance dinámico.
- NetLogo no tiene ningún tipo de datos "palabra" (lo que Lisp llama "símbolos"). Eventualmente, podemos agregar uno, pero como rara vez se solicita, es posible que la necesidad no surja en el modelado basado en agentes. Nosotros tenemos cadenas. En la mayoría de las situaciones en las que un Logo tradicional usaría palabras, simplemente usamos cadenas en su lugar. Por ejemplo, en Logo, puede escribir `[see spot run]` (una lista de palabras), pero en NetLogo debe escribir `"see spot run"` (una cadena) o `["see" "spot" "run"]` (una lista de cadenas) en su lugar.
- La instrucción `run` de NetLogo funciona en procedimientos y cadenas anónimas, no en listas (ya que no tenemos ningún tipo de datos de "palabra"), y no permite la definición o redefinición de procedimientos.
- Las estructuras de control como `if` y `while` son formas especiales, no funciones ordinarias. No se puede definir formas especiales propias, por lo que no se puede definir estructuras propias de control. (Se puede hacer algo similar usando procedimientos anónimos, pero se deben usar las primitivas `->`, `run` y `runresult`, no se puede hacerlos implícitos).
- Los procedimientos anónimos (valores de función aka o lambda) son verdaderos cierres de alcance léxico. Esta característica está disponible en NetLogo y en Lisps modernos, pero no en Logo estándar.

Por supuesto, el lenguaje de NetLogo también contiene otras características que no se encuentran en la mayoría de los Logos, y lo más importante, agentes y conjuntos de agentes.