Boris

NLoops-Lite – light-weight Objects for NetLogo

NLoops-lite ref:0.4a

brief

Our original version of NLoops provides a variety of object features including inheritance, the ability to specify class-level variables, etc. The result is a subsystem that offers some useful tools but requires some learning time by programmers.

After some feedback & consultation we have reworked NLoops to build NLoops-Lite, massively simplifying its implementation (NLoops-Lite is now no more than a handful of NetLogo procedures). While the range of features it offers is been reduced NLoops-Lite is easier to use than the full version.

The main differences with NLoops-Lite is that much of its capability relies on NetLogo programmers sticking to some simple naming conventions for their procedures (see below).

naming conventions

procedure names

Procedure names should be prefixed with the breed name that will call them, eg: if you have a breed named "ants" with a procedure "walking" the procedure should be called "ants.walking"...

```
to ants.walking
  ;; ant walking code
end
```

note...

- if you do not have breeds for your turtles use the prefix "turtles" (use the plural form for this ie: "turtles" not "turtle" & "ants" not "ant", etc);
- prefix link procedures with "links" and patch procedures with "patches";

breed variables

In our more recent examples, we put a "$" character at the start of our breed variables. This naming convention does not change the function NLoops but we think it helps readability.

calling procedures / breed methods

explicitly with prefix

procedures / breed methods can be called explicitly by name as is usually done with NetLogo so the "ants.walking" procedure can be called as normal...

```
ask ants
[ ants.walking
]
```

by name without prefix

procedures / breed methods can be called by name without the breed prefix. This must be done in the context of the correct breed (ie: within an appropriate "ask" block"). NLoops uses #run to do this...

```
ask ants
[ #run "move" []   ;; this will call ants.move
]
ask bees
[ #run "move" []   ;; this will call bees.move
]
```

notes...

- #run selects the correct method for the breed (simply by prepending the breed name on the method name);

with variable referencing

The method name can be supplied in a variable – it does not have to be a string literal, eg:

```
ants-own [ $state ]
...

create-ants 5
[ set $state (one-of ["moving" "sleeping"])
]

...

ask ants
[ #run $state []   ;; call "ants.moving" or "ants.sleeping"
]                  ;; depending on the value of $state
```

reporters/results

#run is used to call procedures that do not report values. #rep is used to call reporter procedures, use #rep like this...

```
ask ants
[ set x #rep "reporter" []   ;; this will call ants.reporter
]                            ;; & store result in x
```

arguments / parameters

Procedures may be defined with or without arguments/parameters but #run and #rep require a list of arguments to be supplied (note the empty list used in the examples above). The rules for supplying arguments to procedures are as follows...

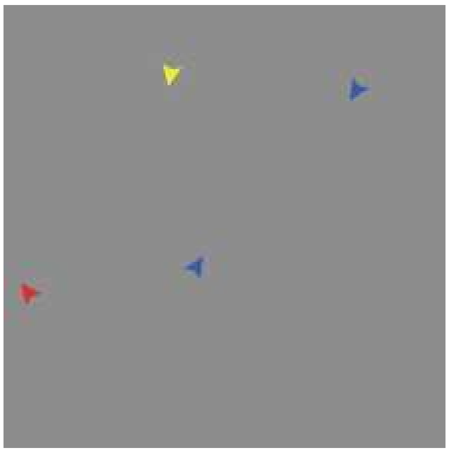
1. if the procedure is defined with no arguments an empty list of arguments should be supplied (see examples above);
2. if the procedure is defined with arguments, these should be supplied in a list, eg...

```
to ants.do-stuff [x y]
  ;; do some stuff with x & y
end
...
ask ants
[  #run “do-stuff” (list value-1 value-2)
]
```

example-1   (NLoops-Lite-model1 (1a).nlogo)

This example uses 1 breed of turtle: “ants”. Ants cycle round three different states: "feeding" "playing" and "sleeping" and are shown blue, red or yellow depending which of these states they are in. Sleeping ants are stationary, feeding ants spin and playing ants run around. Ants move from one state to another after a specified number of ticks and when an ant changes state it prints a message to the command centre window.

The model creates 4 ants named "alf" "ralf" "sue" and "nancy". When it is running, the model may look similar to the figure below...



snapshot of running model

```
alf starts playing
ralf starts playing
sue starts feeding
nancy starts feeding
alf starts sleeping
sue starts playing
```

a fragment of printed output

code details

Each ant has 3 breed variables: $state, $name and $clock. $name holds its name (used for printing output), $clock counts the ticks remaining before the next state change and $state holds the current state of the ant as a string.

There are also a number of global variables initialised as shown below...

```
  set ants.names    ["alf" "ralf" "sue" "nancy"]
  set ants.states   ["feeding" "playing" "sleeping"]

  set ants.feeding-color blue
  set ants.playing-color red
  set ants.sleeping-color yellow

  set ants.feeding-time    10
  set ants.playing-time    15
  set ants.sleeping-time   20
```

The model runs from its "go" procedure which uses #run to call the "move" method for ants. The "ants.move" method uses #run to call "ants.feeding", "ants.playing" or "ants.sleeping" depending on the value of their $state variable. See below...

```
to go
  ask ants
  [ #run "move"  [] ]
  tick
end

to ants.move
   ;; ask ants to run the procedure with the name of their $state
    ;; ie: if $state = feeding then run ant.feeding
    #run $state []
end


to ants.feeding
  ;; quietly munch for another clock tick (feeding ants spin)
  ...
end

to ants.playing
  ;; run around
  ...
end

to ants.sleeping
  ;; sleeping ants do nothing!
  ;; except check to see if it's time to wake up & eat
  ...
end
```

Look at the model code for more details.

examples 2 & 3

These examples ("NLoops-Lite-eg1 (1a).nlogo" and "NLoops-Lite-eg2 (1a).nlogo") use 3 classes/breeds of turtle which perform 3 different types of movement. Green turtles head in straight lines, read turtles circle right, blue turtles circle left (see output below).

You may reasonably decide that you can write this example without objects & methods and achieve the same results with less code & less fuss – this is true. Here we use this model because (we hope) it is fairly easy to understand and because we can use it to illustrate NLoops.

We build our objects-based model on 3 different classes/breeds...

red-ant	– a sub-class of ant that tends to move right
blue-ant	– a sub-class of ant that tends to move left
green-ant	– a sub-class of ant that tends to move forwards

The different classes/breeds use different *methods* for moving, check the code tabs in the models.