Simulation of Predator/Prey Roaming Grid-World (sprog)

---

outline

Sprog is an ongoing project which aims to develop an evolving artificial life system running in a 2D graphics world. Sprog may contain many interacting alife-forms (individuals) from multiple alife species.

the level-0 prototype (sprog-0)

The level-0 prototype provides a fragment of skeletal structure for the system. It defines primitive behaviour for twp species: foxes and rabbits. It uses NetLogo for the graphics environment and a mixture of NetLogo script and Java to specify behaviors. Fox behaviour is specified in NetLogo script and Rabbit behaviour in Java.

system requirements

- NetLogo – version 4.1 or later
- Java

NB: for later sprogs you may require additional software support.

set-up

- create a new folder called "sprog" in the Netlogo/extensions folder;
- create a new folder called "src" in the Netlogo/extensions/sprog folder;
- unpack sprog0.zip into Netlogo/extensions/sprog/src;
- copy sprog.jar to Netlogo/extensions/sprog;
- copy sprog-0(1a).nlogo to Netlogo/extensions/sprog;
- copy sprog-0(clean).nlogo to Netlogo/extensions/sprog;

entry point

The entry point for sprog-0 (ie: how you start it up) is NetLogo. Startup NetLogo and load **sprog-0(1a).nlogo** which you should have copied into the the NetLogo sprog extension folder. Click the "setup" button, then click "go".

modifying the behaviours

The behaviours of foxes & Rabbits can be modified by changing the NetLogo side and/or the Java side.

NetLogo-side modification

The NetLogo side can be modified by editing the code in the procedures tab of the NetLogo environment.

rabbit behaviour

Currently all rabbit behaviour is delegated to the Java side by specifying "sprog:move" which calls ***Java>SprogManager.move*** (NB: we would like to extend development of this behaviour on the Java side).

```
to move-rabbits
  ask rabbits
  [ run sprog:move          ;; delegate behaviour to the Java side
  ]
end
```

fox behaviour

All fox behaviour is currently specified on the NetLogo side – we are happy for more of this to be delegated to the Java side.

```
to move-foxes
  ask foxes
  [ let r min-one-of rabbits [distance myself]     ;; find nearest rabbit
    if r != nobody
    [ face r                          ;; face nearest rabbit
      right 45 - random 90            ;; randomly turn a bit
      forward 1.5                     ;; move forward a step & a half
      ask rabbits-here [die]          ;; any rabbits here die
    ]
  ]
end
```

Java-side modification

All behaviour is currently routed through to the move method of the SprogManager class. This occurs as a result of the call to "sprog:move" in the NetLogo code. Which is currently specified as follows...

```
public String move( String breed,
                    org.nlogo.agent.Turtle turtle,
                    Context context )
throws AgentException
{     if( breed.equals( "RABBITS" ))
      {     turtle.turnRight( 45 - random(90) );
            turtle.jump( 1 );
            return "";
      }
      else
            return "";
}
```

the next steps for T-PIGers

1 enhance rabbit behaviour

Code up some more "intelligent" behaviour for Rabbits (trying to get away from the nearest Fox would be an improvement).

2 code fox behaviour on the Java side

Recode the Fox behaviour on the Java side. Do this by calling the move method in SprogManager from the NetLogo move-foxes procedure.

3 restructure the SprogManager class

If you add new species behaviour to the Java>SprogManager carelessly you could end up with an ugly clump of "`if( breed.equals( ??? ))`" statements which either contain behaviour or redirect it. There are better ways to do this.

4 slave the NetLogo environment

The entry point is currently via NetLogo. We would like to investigate calling NetLogo (as a slave subsystem) from a Java application. The NetLogo documentation gives an example of this.

other notes

1. NetLogo caches extensions so once you have loaded a model which uses an extension (the sprog extension in this case) editing & recompiling the Java code that defines the extension will not (necessarily) cause the model behaviour to change. To force the extension to be reloaded you should load some other model then reload your sprog???.nlogo model. You should have copied **sprog-0(clean).nlogo** into the the NetLogo sprog extension folder. This does nothing but will allow you to switch models without having to navigate folders using some file-chooser.

2. When you are compiling on the Java-side make sure Netlogo.jar is in the Java classpath.

3. any NetLogo extension must be in a folder with the same name as the extension and the extension should be defined by a .jar file (the name of the .jar is defined in the jar manifest; the manifest we have made available uses the name **sprog.jar** - you do not need to change this).

4. When you modify any Java files you will need to rebuild the jar. Assuming you are in the **src** folder (NetLogo/extensions/sprog/src) and you have not modified the manifest, you can rebuild the jar as follows...

```
jar  cvfm  ../sprog.jar  manifest.txt  .
```