NetLogo

Andrea Sterbini – sterbini@di.uniroma1.it

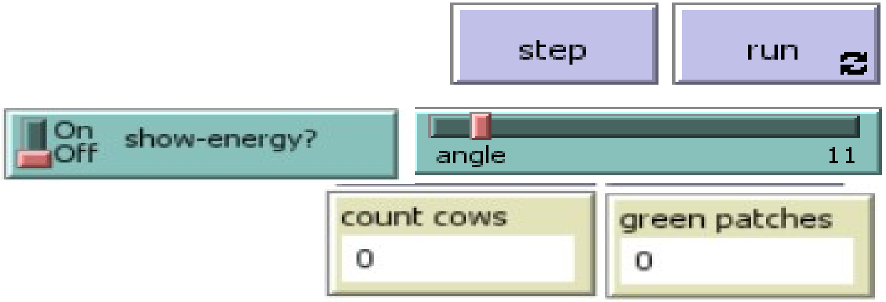
NetLogo and NetLogoWeb
turtles + patches = movable agent simulations

Full Logo:

- procedures + reporters (functions)
- lists and filters
- anonymous functions (parametric code blocks)
- new agent types with added properties (OOP without inheritance?)

<u>Easy GUI construction</u>:

- Buttons to call functions/procedures
- Sliders to change global variables
- Labelled boxes to show values
- Plot graphs of values during simulation
- 2 NetLogo versions: <u>2D</u> and <u>3D</u> canvas showing turtles, patches and edges



show-energy?

angle 11

count cows 0

green patches 0

3 type of Agents (+ custom agents)

Turtles: movable entities (with respect to the 2Dor 3D canvas)

Patches: the canvas is covered by a grid of unmovable squares (cubes)
- e.g. the grass of a field or a pixellated volume (2 or 3 dim. MATRIX concept)

Edges: links between two Turtles in 3D space

Other “custom animal groups” can be easily defined:
- breed [singular plural]

Separate breeds can have separate sets of properties:
- cows-own [energy]

The Turtles’ set contains all other breeds (like “object” in Java)

An agent can change its breed type! (set breed ‘breedname’)

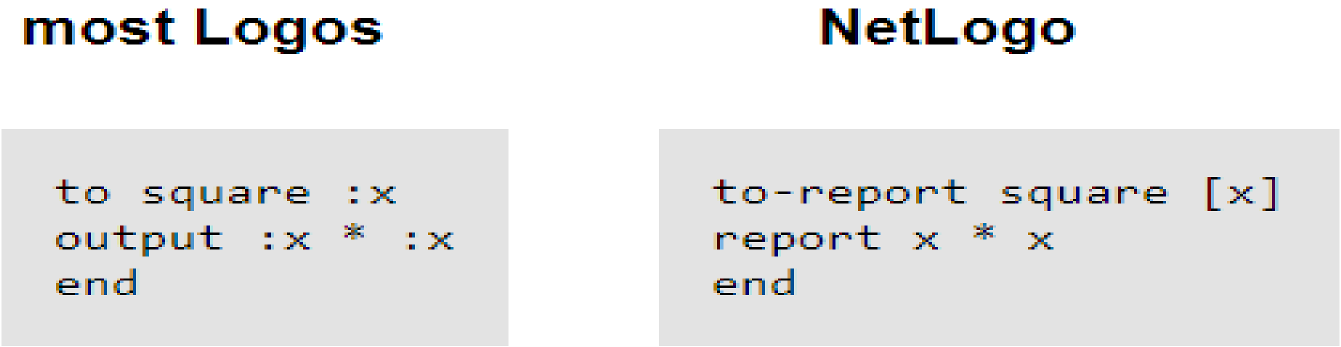
Programming style

Single-threaded	(BUT the order of set elements is random)
Procedural	(“to” procedures)
Functional	(“to-report” functions)
Data types:	
- lists	(immutable, untyped)
- arrays	(mutable, untyped)
- list-based operations	(map/filter/collect/ask/...)
- anonymous functions	(code blocks)

A LOT of built-in commands are functions/filters
THUS the language is very very readable

NetLogo and other Logos

Small syntactic differences



most Logos

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

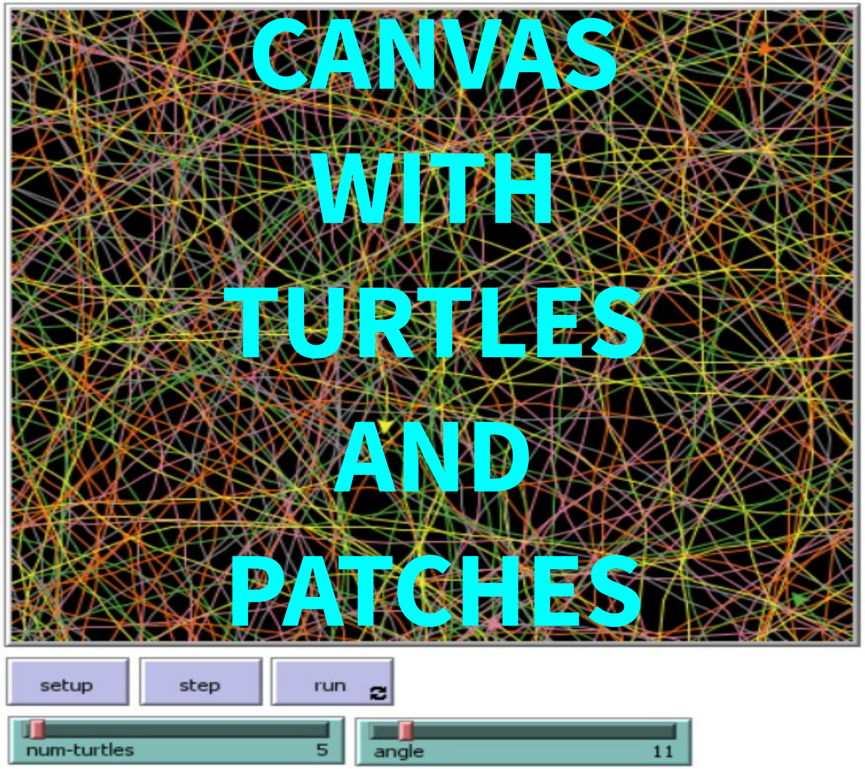
NetLogo

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

to-report	instead than	to
report	instead than	output
[args]	instead than	:arg

some precedence differences

Demo 1: Random walk



CANVAS WITH TURTLES AND PATCHES

- start with N randomly placed turtles
- move each turtle
    - by 1 step
    - by changing slightly its heading

NO main loop, just use a repeating button with a single simulation “step” procedure

Globals: (interactive slider) max turn angle, # of turtles

```
to step
  ask turtles [
    set heading (heading + (random (2 * angle)) - angle)
    forward 1
  ]
  tick          ;; update tick count
end
```

This is an anonymous block executed **IN THE TURTLE’s CONTEXT!**

Demo 2: a flock of birds

Here each turtle should:

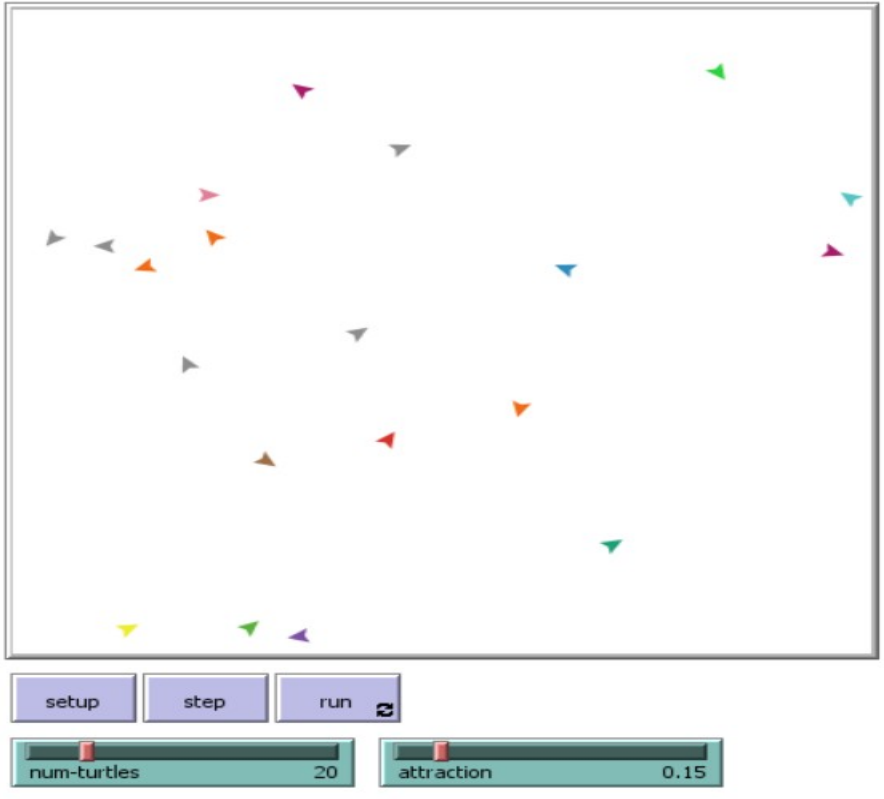
- turn towards her nearest neighbour
- and move

Globals:

- # of turtles, attraction towards nearest

```
to-report closest-turtle
  report min-one-of (other turtles) [
      distance myself ]
end
```

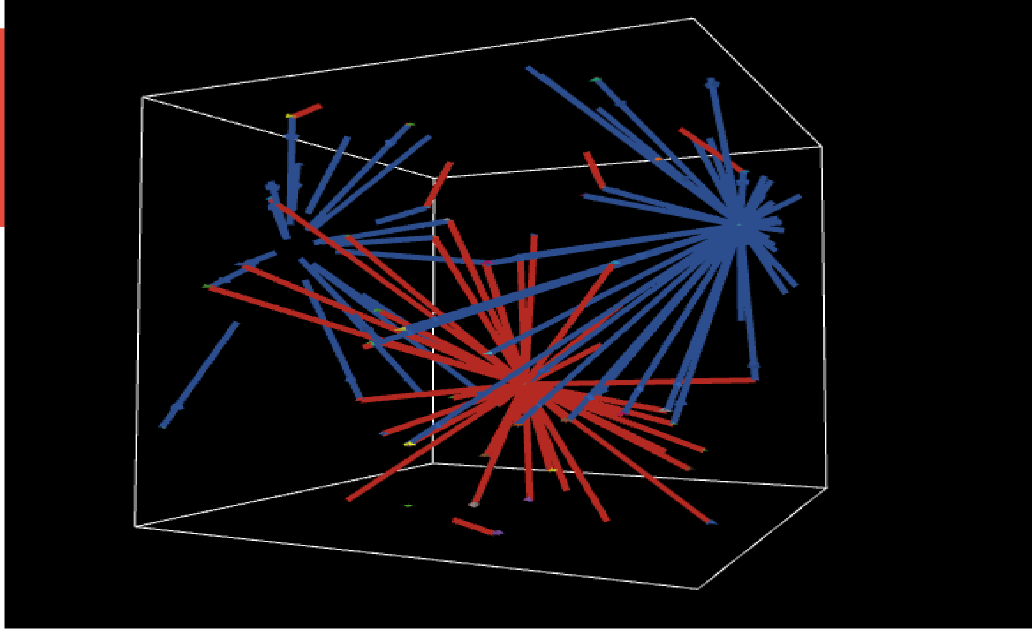
```
to turn-towards [somebody]
    let difference subtract-headings (towards somebody) heading
    set heading (heading + (attraction * difference)
end
```



num-turtles 20

attraction 0.15

Demo 3: 3D links



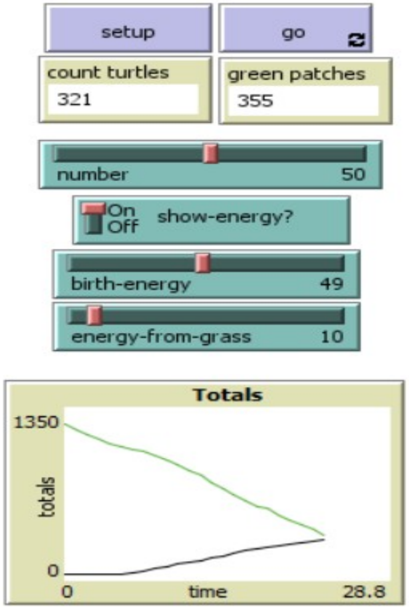
- N turtles in random 3D position

- 2 random turtles are connected to all other turtles with directed and undirected edges

- NOTICE: the world is a TORUS!

```
undirected-link-breed [ ulinks ulink ]
directed-link-breed   [ dlinks dlink ]
to setup
  clear-all
  create-turtles N [ setxyz random-xcor random-ycor random-zcor ]
  ask turtle random N
    [ create-ulinks-with other turtles [ set color red  ] ]
  ask turtle random N
    [ create-dlinks-to   other turtles [ set color blue ] ]
end
```

Demo 4:
cows on grass



count turtles 321

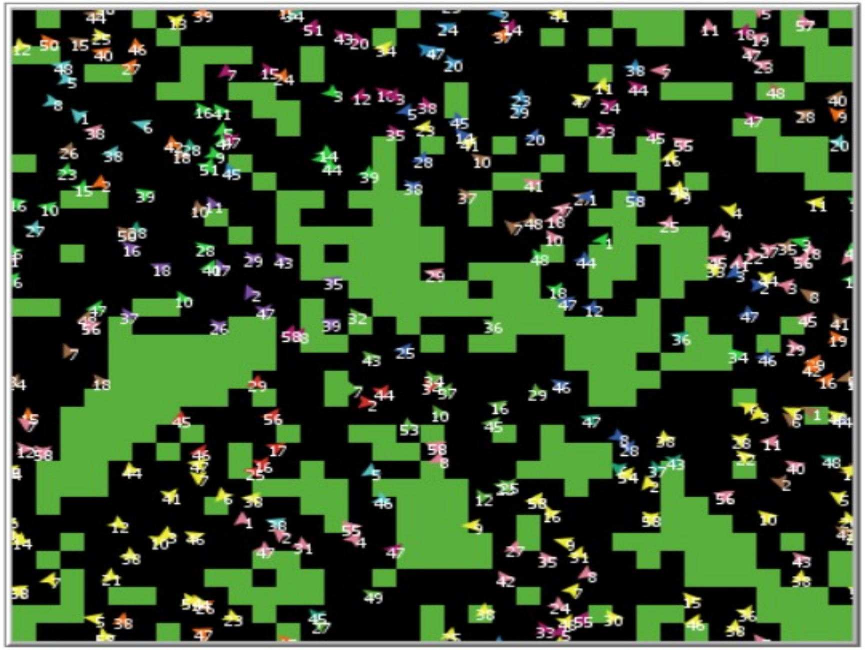
green patches 355

number 50

On Off show-energy?

birth-energy 49

energy-from-grass 10



Cows:

- loose 1 energy per tick
- move at random
- eat grass gaining 10 e.
- if energy>50 spawn

Grass:

- new grass grows with 3% probability

Globals:

- show cow energy?, energy to give birth, energy from grass

Display:

- # of cows, # of grass patches

Demo 4: implementation …

```
breed [ cows cow ]          ; define cows

cows-own [energy]           ; add attribute

… (setup removed)

to go                       ; single sim. step
  if ticks >= 500 [ stop ]
  move-cows
  eat-grass
  check-death
  reproduce
  regrow-grass
  tick
end
```

```
to eat-grass     ; eating increases energy
  ask cows [
    if pcolor = green [
      set pcolor black
      set energy (energy + energy-from-grass)
    ]
    ifelse show-energy?
      [ set label energy ]
      [ set label "" ]
  ]
end
```

… continue

```
to move-cows                    ; move all cows
  ask cows [
    right random 360            ; change direction
    forward 1                   ; move
    set energy energy - 1       ; lose energy
  ]
end

to reproduce                    ; healthy cows reproduce
  ask cows [
    if energy > birth-energy [
      set energy energy - birth-energy
      hatch 1 [ set energy birth-energy ]
    ]
]
end
```

```
to check-death          ; disallocate dead cows
  ask cows [
   if energy <= 0 [ die ]
  ]
end

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

Extensions!!!

Arduino

CSV

Continuous f. optimiz.

Modular models

Clustering

Cognitive Agents

GIS

Python

Webcam

GoGo boards

Database

Function roots

Linear programming

Freq. Distributions

Q-learning

Epidemiology

R

Isometric visualization Web

Profiler

Matrix math

Time series

Statistics

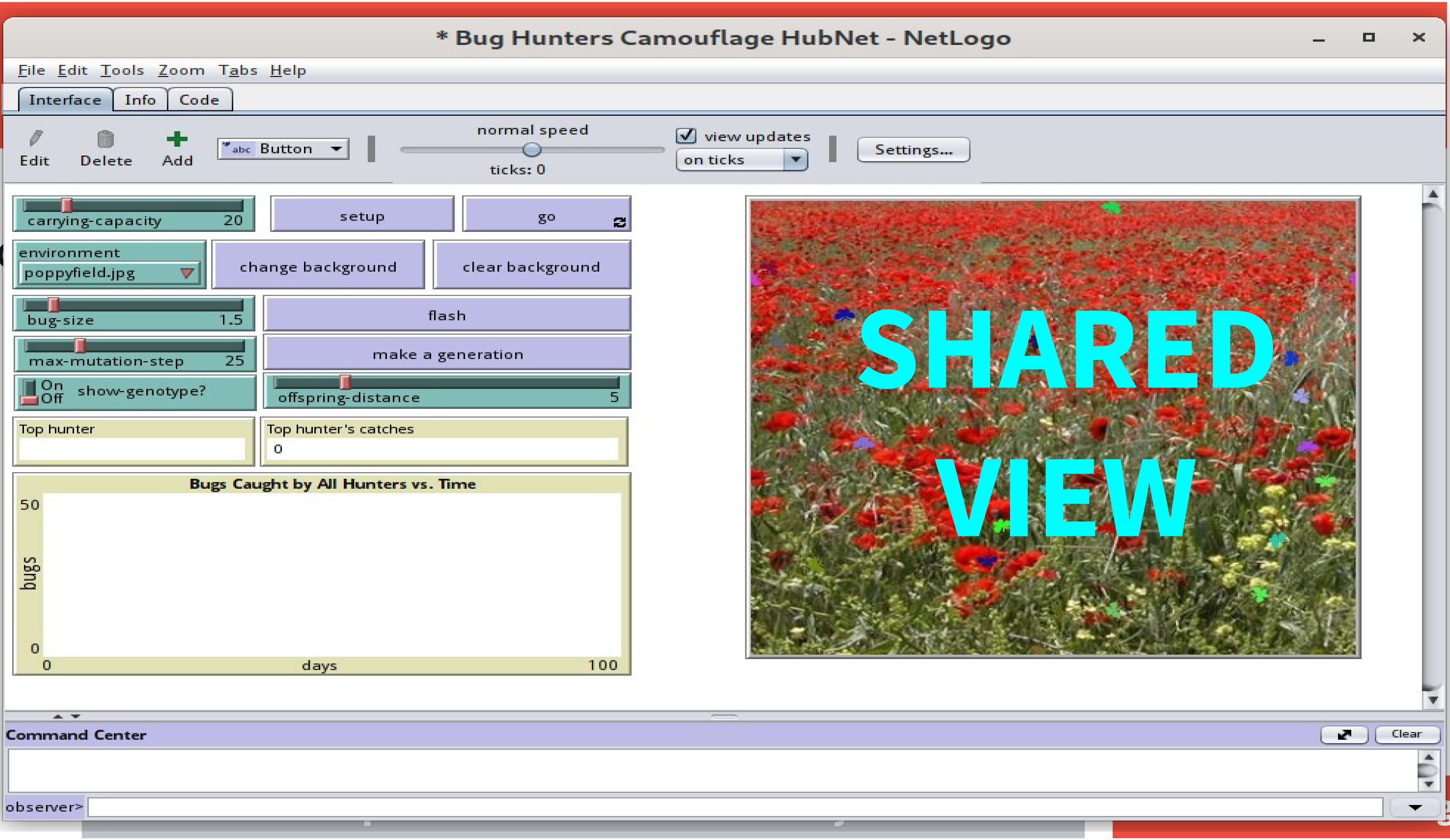
Fuzzy logic

Physics

Scala

A

- (



* Bug Hunters Camouflage HubNet - NetLogo

ticks: 0

on ticks

carrying-capacity 20

environment poppyfield.jpg

bug-size 1.5

max-mutation-step 25

On Off show-genotype?

offspring-distance 5

Top hunter

Top hunter's catches 0

go

Client GUI

Simplified GUI edited in the main app. The main code must handle the standard messages.

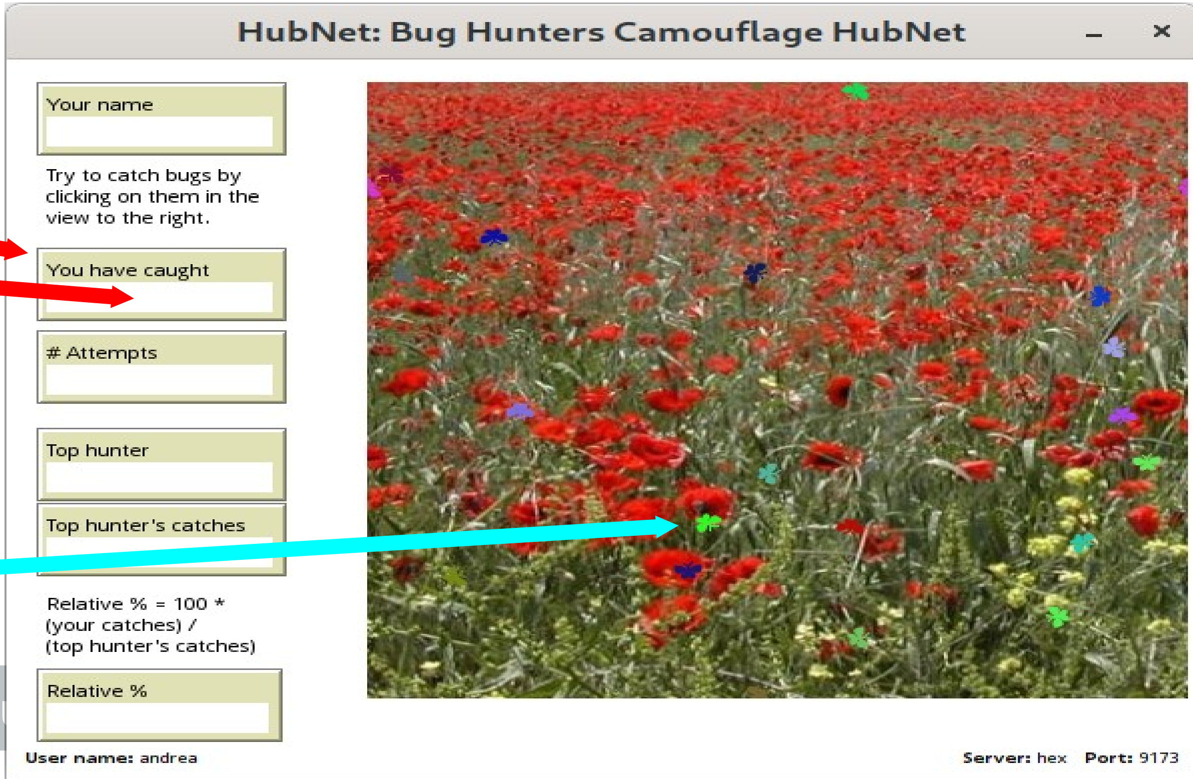
All widgets send/receive standardized messages

- client ID
- widget tag
- widget content

The View shows the canvas+turtles

Clicks of clients on views -> 2 messages

- mouse down / up
- mouse position



HubNet: Bug Hunters Camouflage HubNet

Your name

Try to catch bugs by clicking on them in the view to the right.

You have caught

Attempts

Top hunter

Top hunter's catches

Relative % = 100 * (your catches) / (top hunter's catches)

Relative %

User name: andrea

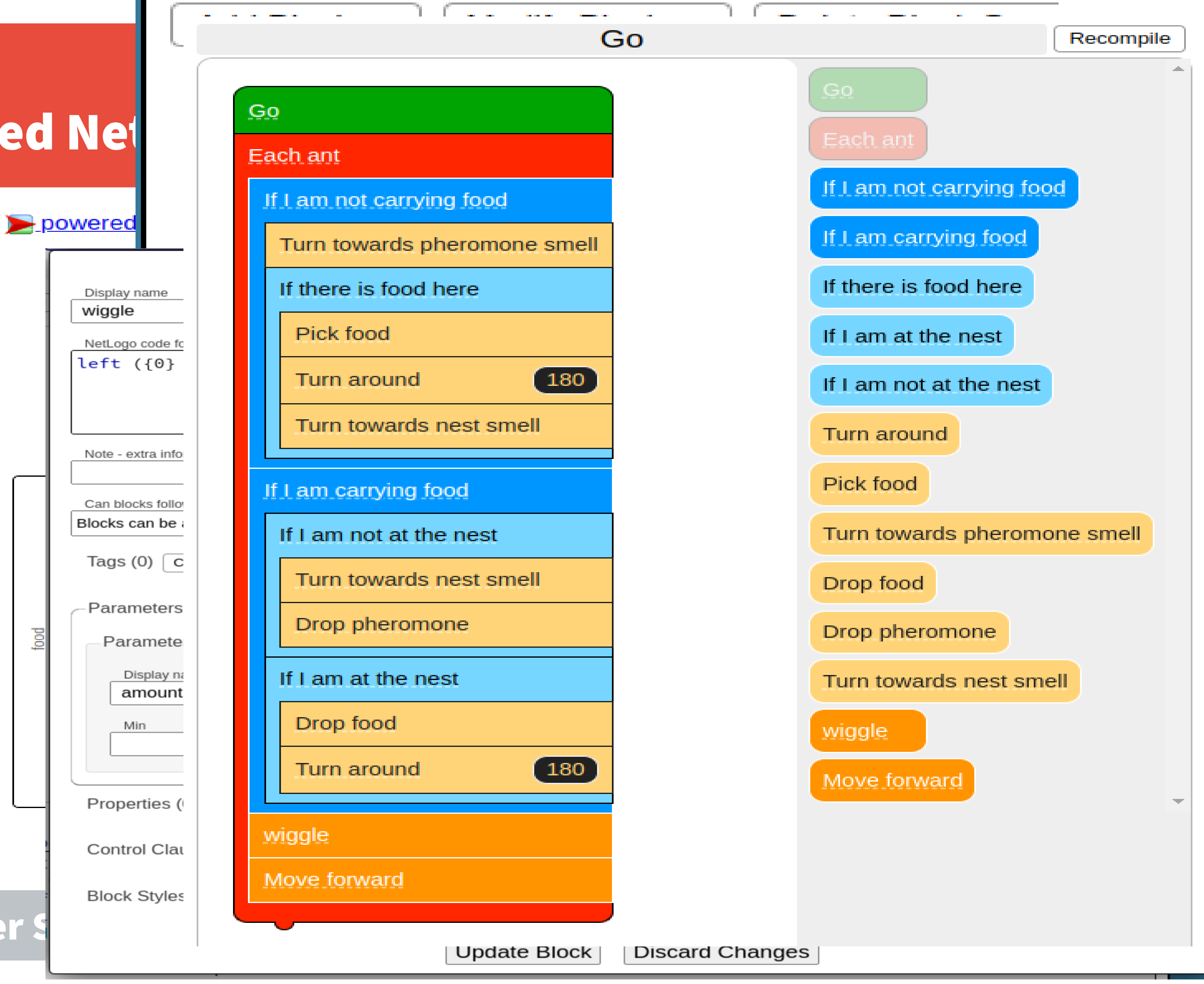
Server: hex Port: 9173

NetTango: block-based Net

Web-based NetLogo editor / simulator

Define blocks containing snippets of code (macro operations)

Kid program by using your defined blocks



Display name
wiggle

NetLogo code fo
```
left ({0}
```

Note - extra info

Can blocks follo
Blocks can be

Tags (0)

Parameters

Paramete

Display na
amount

Min

Properties (

Control Clau

Block Styles

Update Block | Discard Changes

```
Go
Each ant
    If I am not carrying food
        Turn towards pheromone smell
        If there is food here
            Pick food
            Turn around 180
            Turn towards nest smell
    If I am carrying food
        If I am not at the nest
            Turn towards nest smell
            Drop pheromone
        If I am at the nest
            Drop food
            Turn around 180
    wiggle
    Move forward
```

Other ideas

Modeling Commons:	cooperatively shared repository of models
Behavior Space:	hyper-parameters optimization over many model runs
System Dynamics:	high-level modeling (instead than agent-based) with evolution of global measures (e.g. #of sheeps vs # of wolves)
Mathematica Link:	call Mathematica from Netlogo

Demo

DEMO