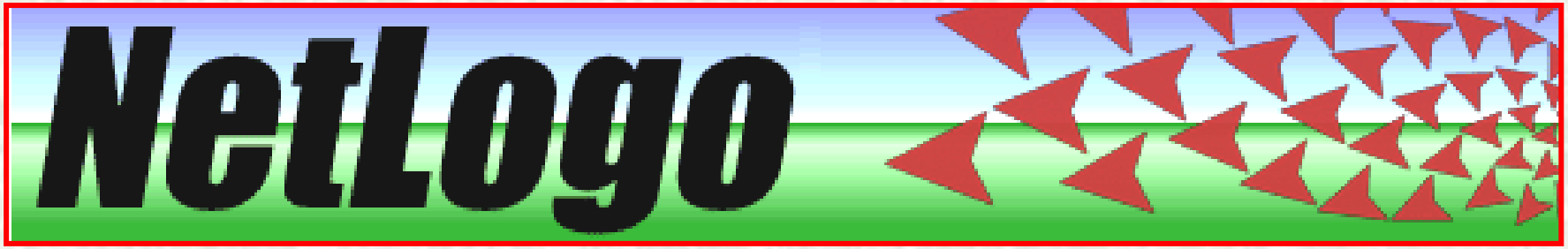
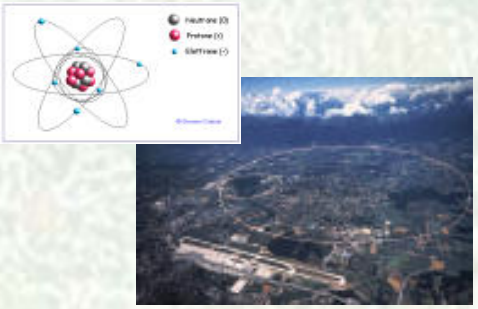
Alessandro Pluchino



Introduzione alla programmazione di simulazioni ad agenti

Dipartimento di Fisica e Astronomia and INFN sezione di Catania
Università di Catania - Italy
Group web page: www.ct.infn.it/~cactus



Infinitamente Complesso

Cosmologia, Astronomia, Astrofisica, Teoria della Relativitá Generale...

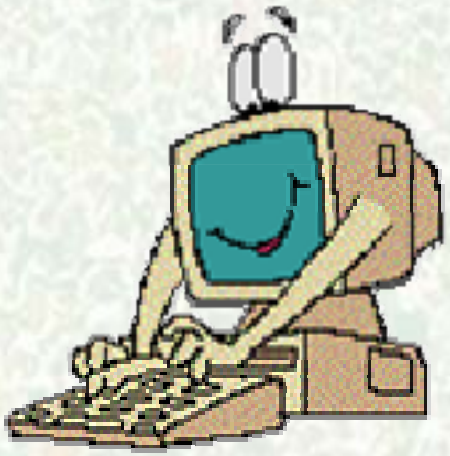
Fisica Nucleare e particellare, Meccanica Quantistica, GUT, Teoria delle Stringhe, ...

Infinitamente Grande

Metà del XX secolo

Infinitamente Piccolo

Nascita delle Scienze Fisiche XVI secolo



il Computer!

...e la possibilità di simulare i sistemi complessi!

Lo studio dei sistemi complessi e le teorie delle tre "C"

anni Settanta

Teoria delle Catastrofi

Studia i “punti critici” che modificano bruscamente il comportamento di un sistema

anni Ottanta

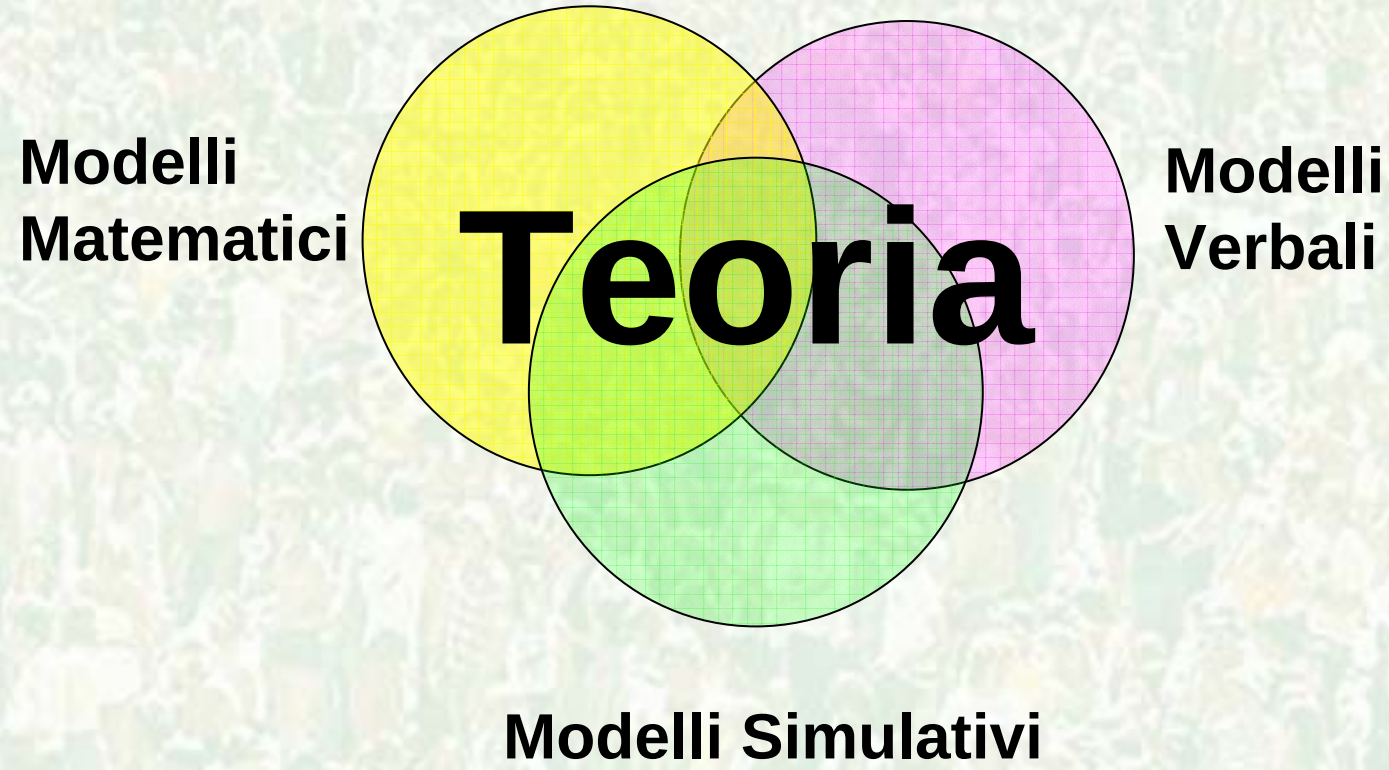
Teoria del Caos

Sistemi a pochi gradi di libertá, non linearitá, caos deterministico, sensibilità alle condizioni iniziali ed “effetto farfalla”, autosimilarità, geometria frattale

anni Novanta

Teoria della Complessità

Sistemi a molti gradi di libertá, criticità auto-organizzata, sistemi lontani dall’equilibrio termodinamico con interazioni a lungo range, sistemi “at the edge of chaos”, networks e fenomeni emergenti


Modelli
Matematici
Teoria
Modelli
Verbali
Modelli Simulativi

NetLogo

Il software NETLOGO, sviluppato originariamente da Uri Wilensky nel 1999 presso il Center for Connected Learning and Computer-Based Modeling della Northwestern University (http://ccl.northwestern.edu/netlogo/), offre un ambiente di sviluppo ideale per la realizzazione di modelli di simulazioni ad agenti, di networks e di sistemi dinamici.

NetLogo

All’interno di NETLOGO é possibile riprodurre molte delle caratteristiche di un sistema complesso, studiare la sua evoluzione nel tempo e visualizzarla in tempo reale all’ interno di un laboratorio virtuale 2D o 3D.

NETLOGO é stato scritto in JAVA ma é completamente programmabile per mezzo di un meta-linguaggio ad oggetti semplificato basato sulla sintassi del Logo (S.Papert, anni ‘60) ed introdotto originariamente nell’ambiente Starlogo, il progenitore di Netlogo creato all’MIT nel 1990 (M.Resnick).

NetLogo

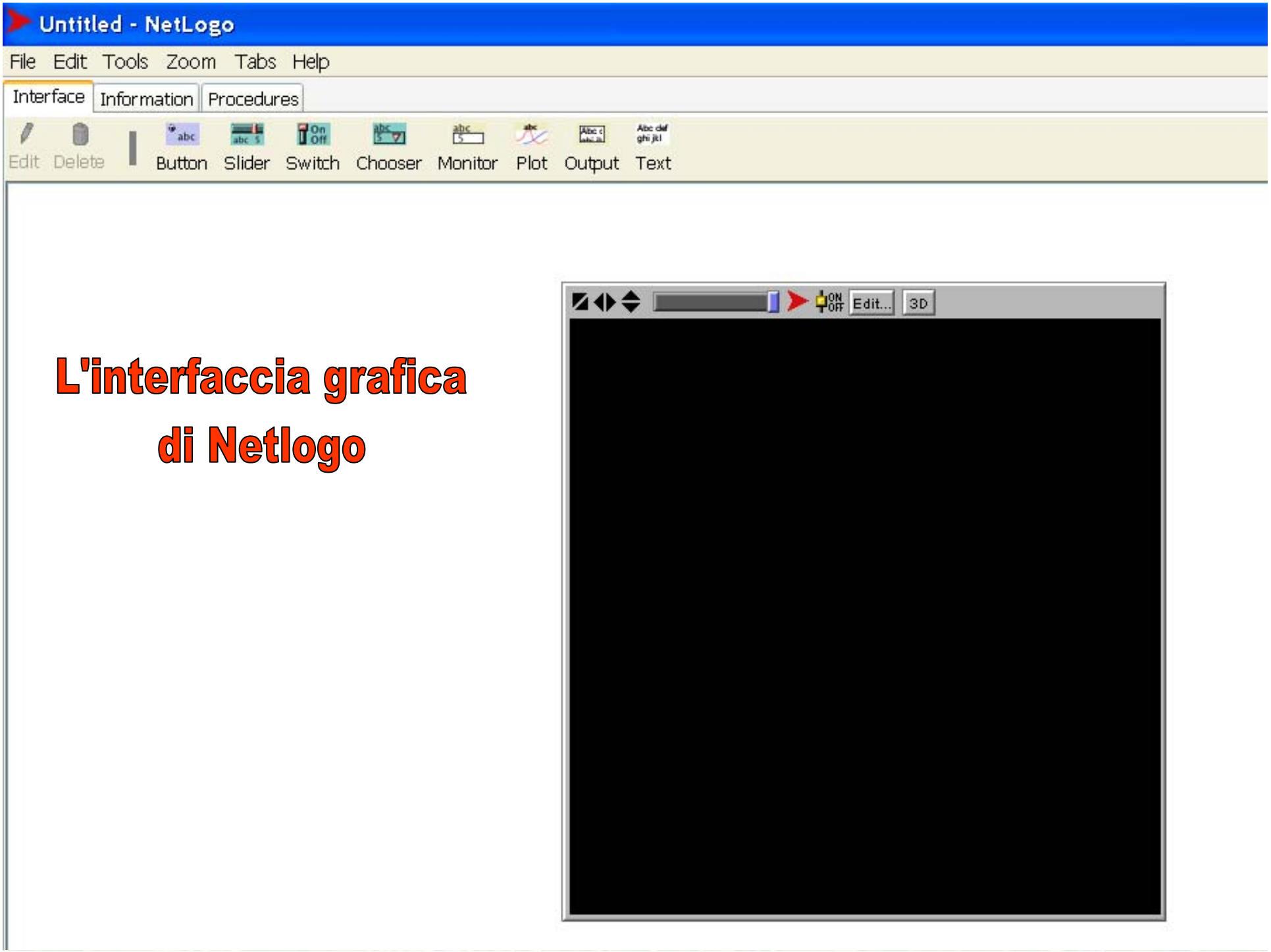
L’ambiente di sviluppo di NETLOGO interpreta direttamente il codice senza bisogno di compilarlo, quindi al suo interno é possibile, in tempo reale:

- interagire con il sistema attraverso pulsanti e sliders per modificare i parametri di controllo;
- visualizzare variabili, grafici e istogrammi relativi alla simulazione;
- importare/salvare immagini e dati da/su files esterni;

NetLogo

E’possibile inoltre:

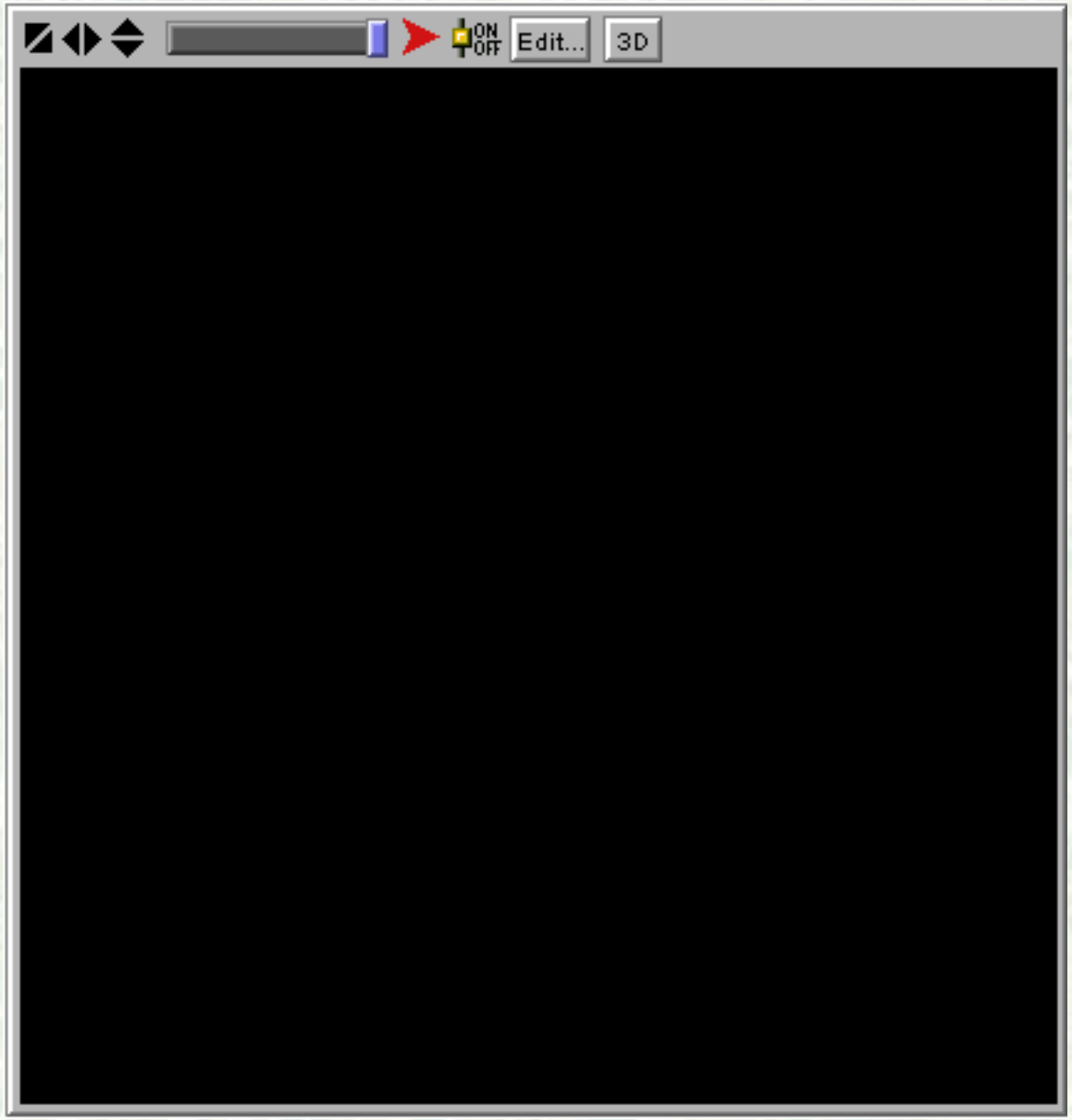
- effettuare set di esperimenti al variare delle condizioni iniziali o dei parametri di controllo;
- lanciare le simulazioni piú pesanti in modalitá off-line su computer remoti;
- usufruire di una ricca libreria di modelli di esempio da cui prendere spunto e di un help pratico ed ipertestuale;
- salvare le proprie simulazioni come applet JAVA e visualizzarle immediatamente dentro un web browser.



L'interfaccia grafica di Netlogo

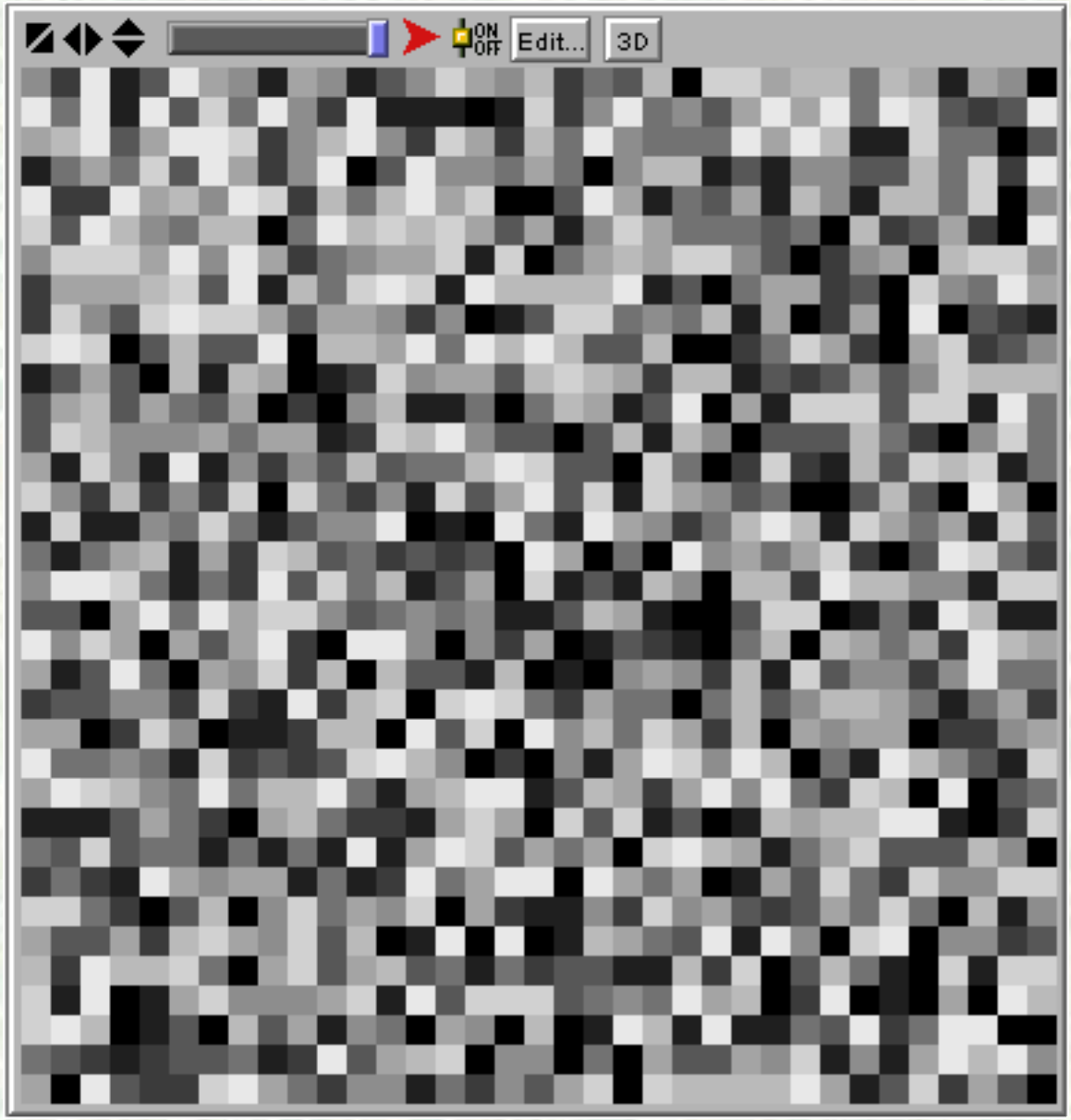
NetLogo

L'ambiente di sviluppo: WORLD, il mondo virtuale



NetLogo

L'ambiente di sviluppo: WORLD, il mondo virtuale

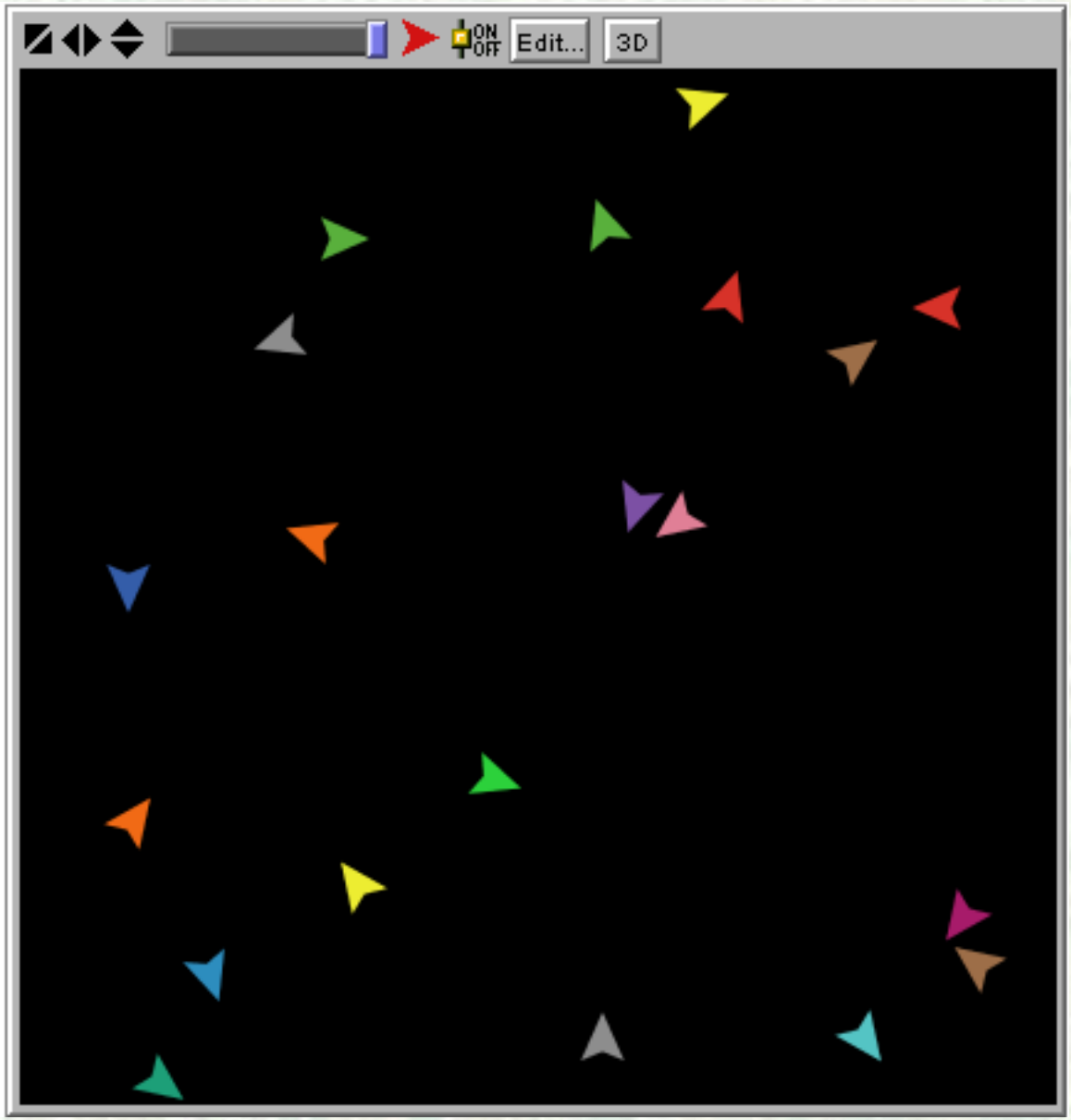


Patches

Corrispondo ai pixel del mondo virtuale e sono quindi immobili. Possono cambiare colore o contenere informazioni sotto forma di variabili proprietarie

NetLogo

L'ambiente di sviluppo: WORLD, il mondo virtuale

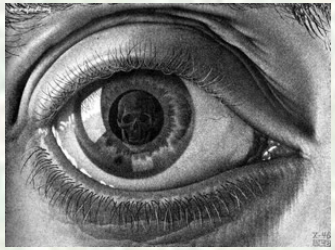


Turtles

Sono oggetti in grado di muoversi, cambiare colore, forma e proprietá, anche stavolta sotto forma di variabili proprietarie.

NetLogo

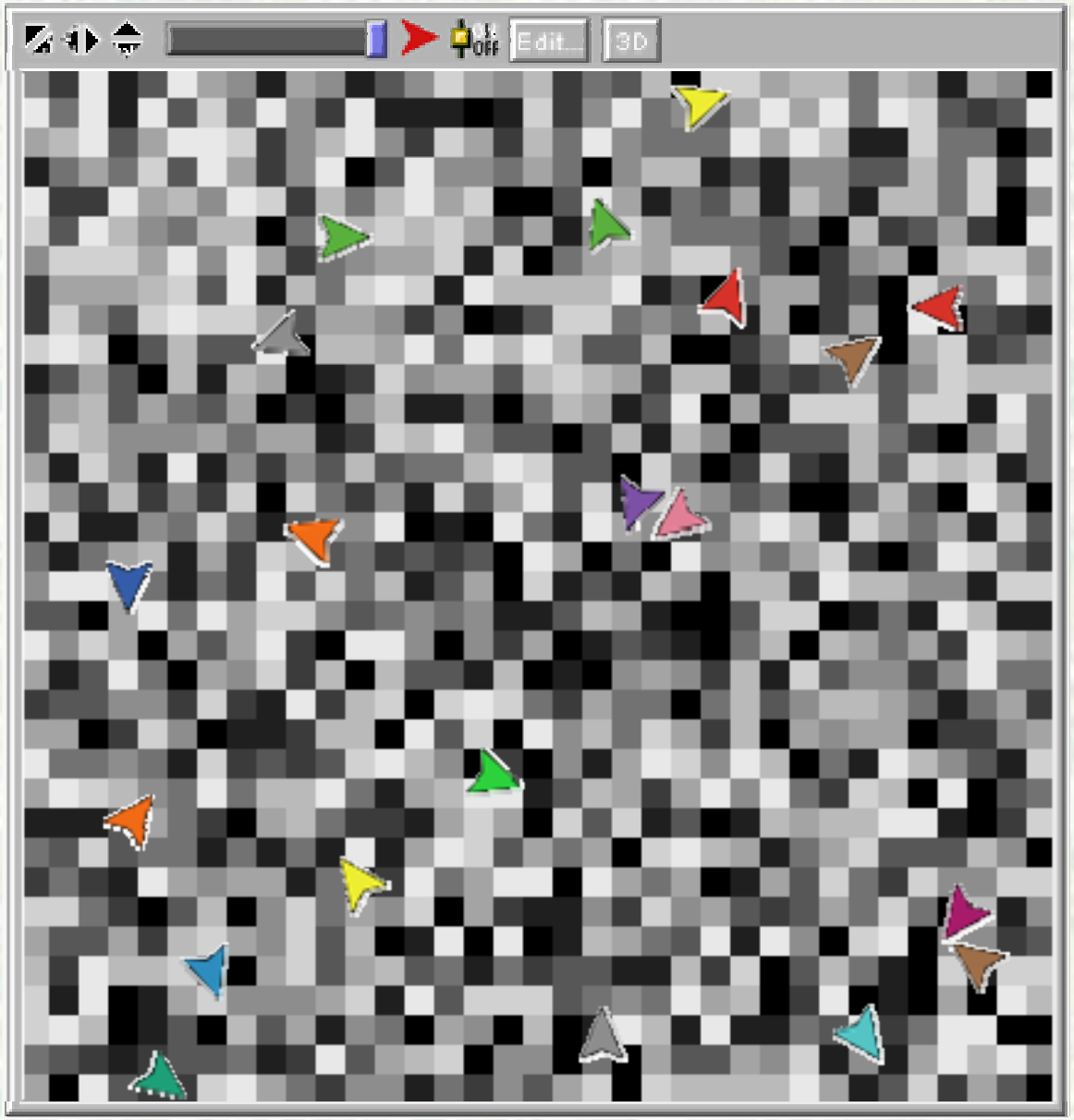
L'ambiente di sviluppo: WORLD, il mondo virtuale



Observer

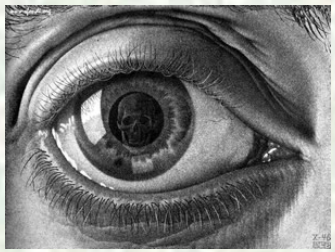
Patches

Turtles



NetLogo

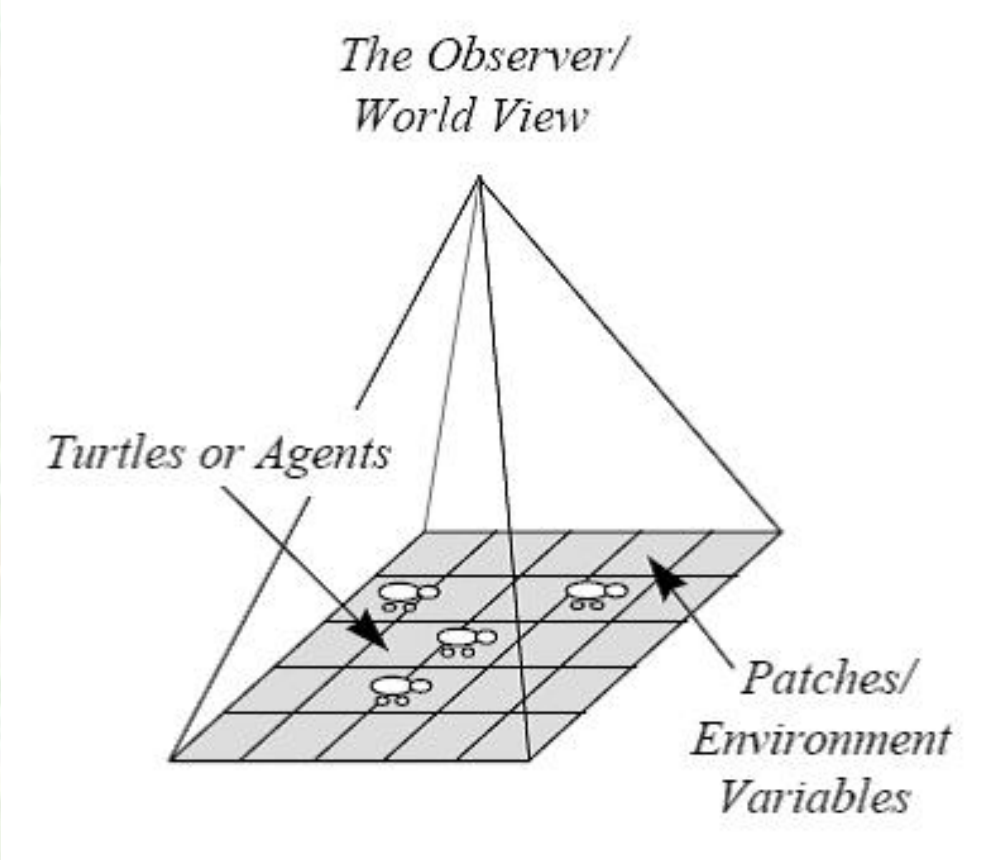
L'ambiente di sviluppo: WORLD, il mondo virtuale



Observer

Patches

Turtles



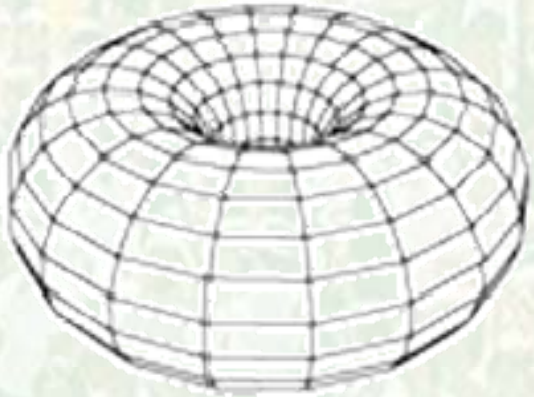
NetLogo

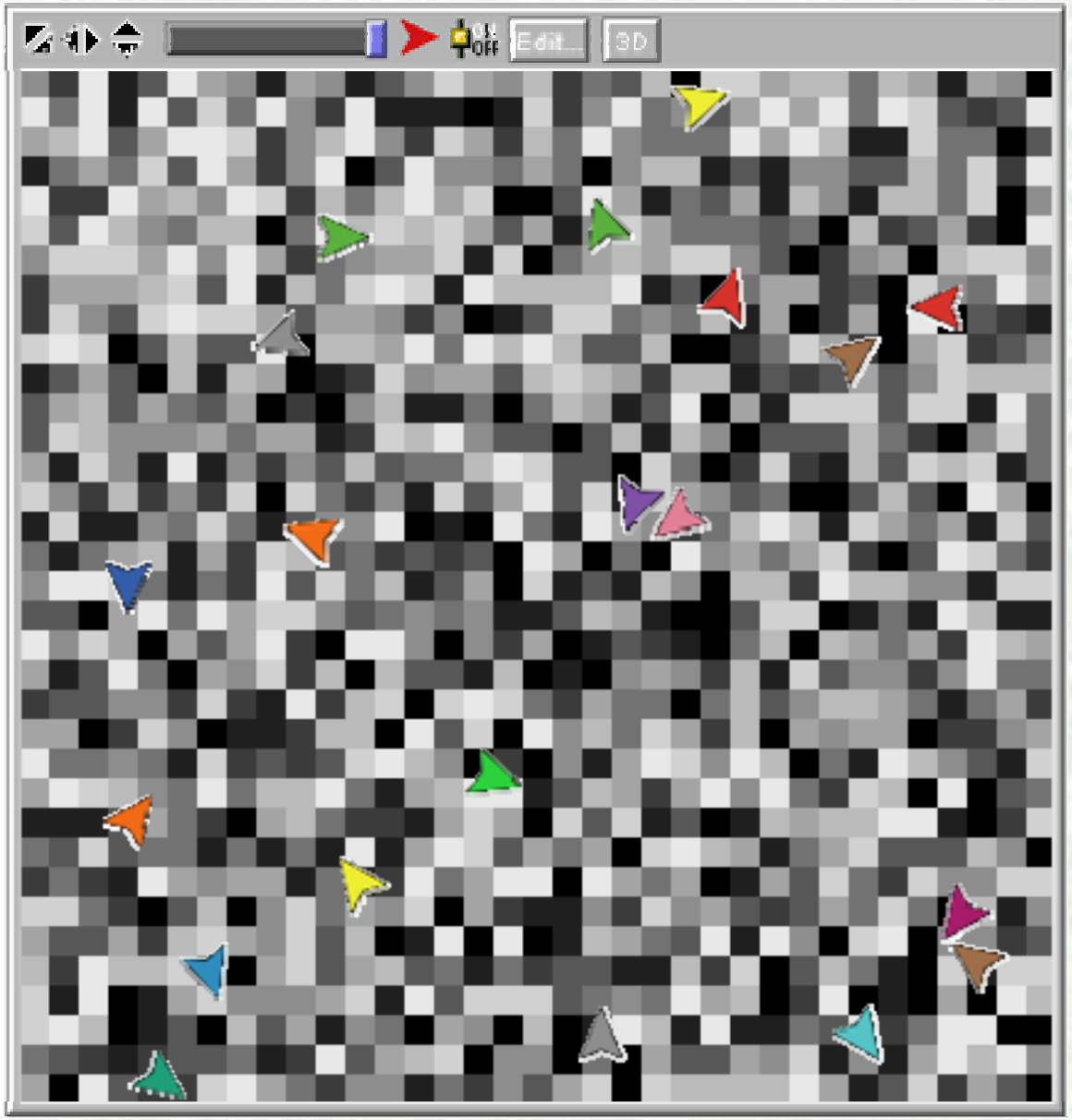
L'ambiente di sviluppo: WORLD, il mondo virtuale

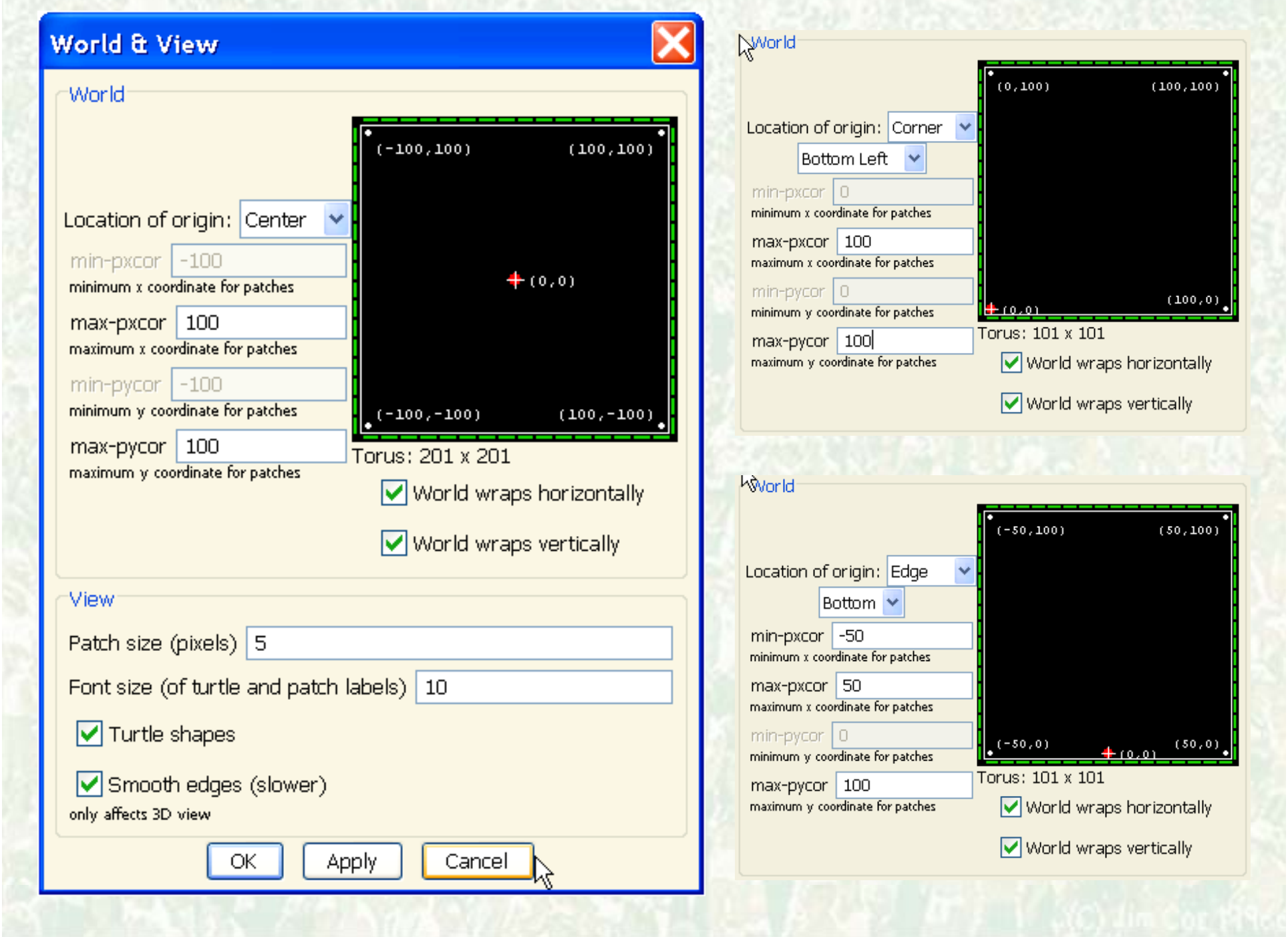
topologie:

rettangolare, limitata

toroidale, illimitata

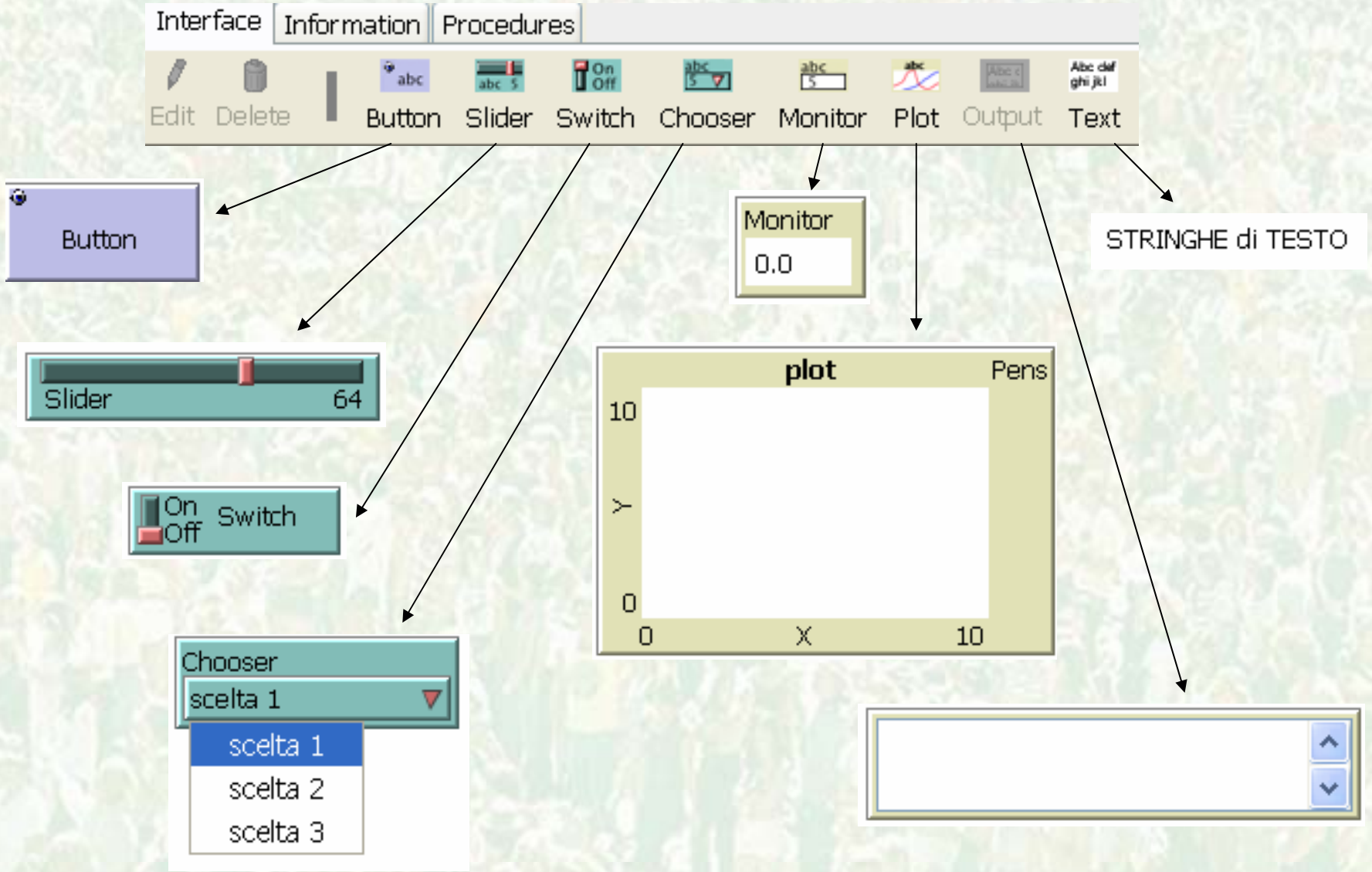





World & View
World
Location of origin:
Center
min-pxcor
-100
minimum x coordinate for patches
max-pxcor
100
maximum x coordinate for patches
min-pycor
-100
minimum y coordinate for patches
max-pycor
100
maximum y coordinate for patches
(-100,100)
(100,100)
(0,0)
(-100,-100)
(100,-100)
Torus: 201 x 201
World wraps horizontally
World wraps vertically
View
Patch size (pixels)
5
Font size (of turtle and patch labels)
10
Turtle shapes
Smooth edges (slower)
only affects 3D view
OK
Apply
Cancel
World
Location of origin:
Corner
Bottom Left
min-pxcor
0
minimum x coordinate for patches
max-pxcor
100
maximum x coordinate for patches
min-pycor
0
minimum y coordinate for patches
max-pycor
100
maximum y coordinate for patches
(0,100)
(100,100)
(0,0)
(100,0)
Torus: 101 x 101
World wraps horizontally
World wraps vertically
World
Location of origin:
Edge
Bottom
min-pxcor
-50
minimum x coordinate for patches
max-pxcor
50
maximum x coordinate for patches
min-pycor
0
minimum y coordinate for patches
max-pycor
100
maximum y coordinate for patches
(-50,100)
(50,100)
(-50,0)
(0,0)
(50,0)
Torus: 101 x 101
World wraps horizontally
World wraps vertically

NetLogo

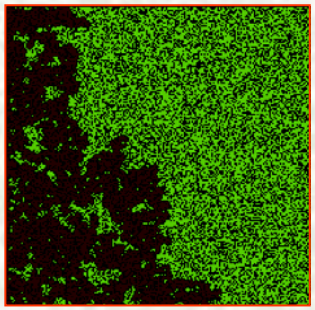
Altri elementi dell'interfaccia grafica



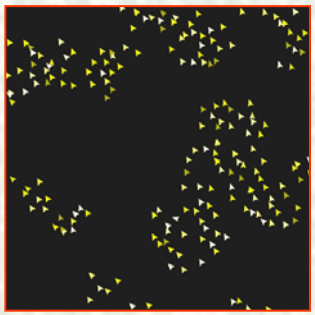
NetLogo

Vediamo qualche esempio per familiarizzare con l'interfaccia grafica:

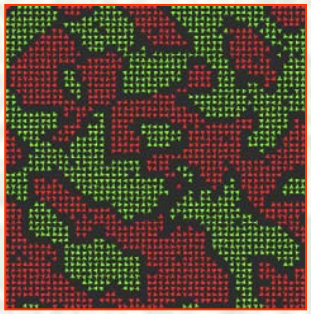
incendi



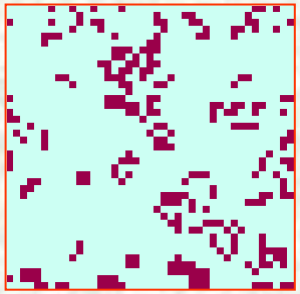
stormi



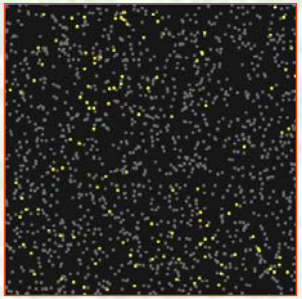
segregazione



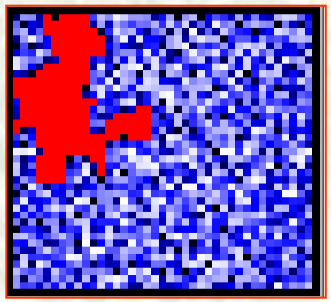
il gioco LIFE



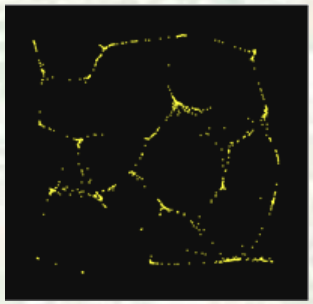
sincronia



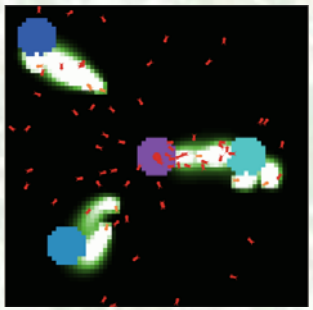
terremoti



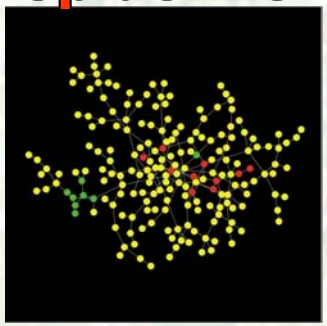
opinioni



formiche

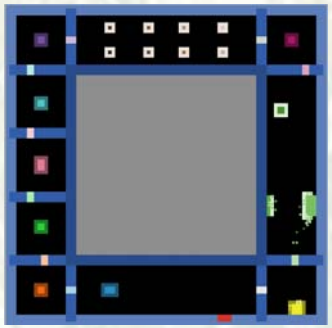


epidemie

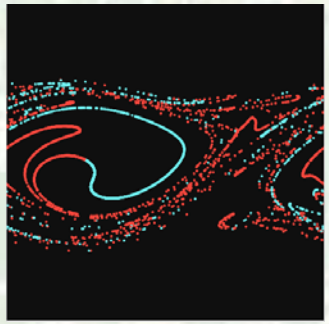


rivelatori

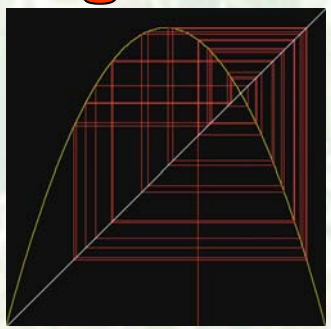
c.Ursino



HMF

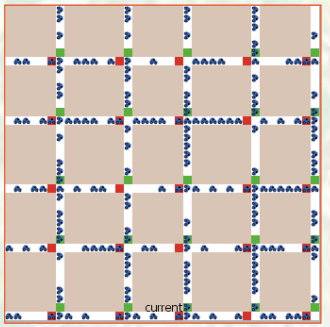


mappa logistica



sociodinamica

traffico

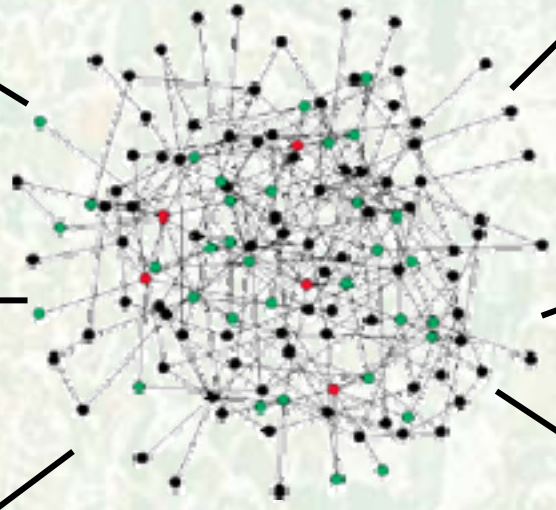


NetLogo

Caratteristiche tipiche di un sistema complesso riproducibili con NETLOGO:

non-linearitá e correlazioni a lungo range

il tutto é maggiore della somma delle parti



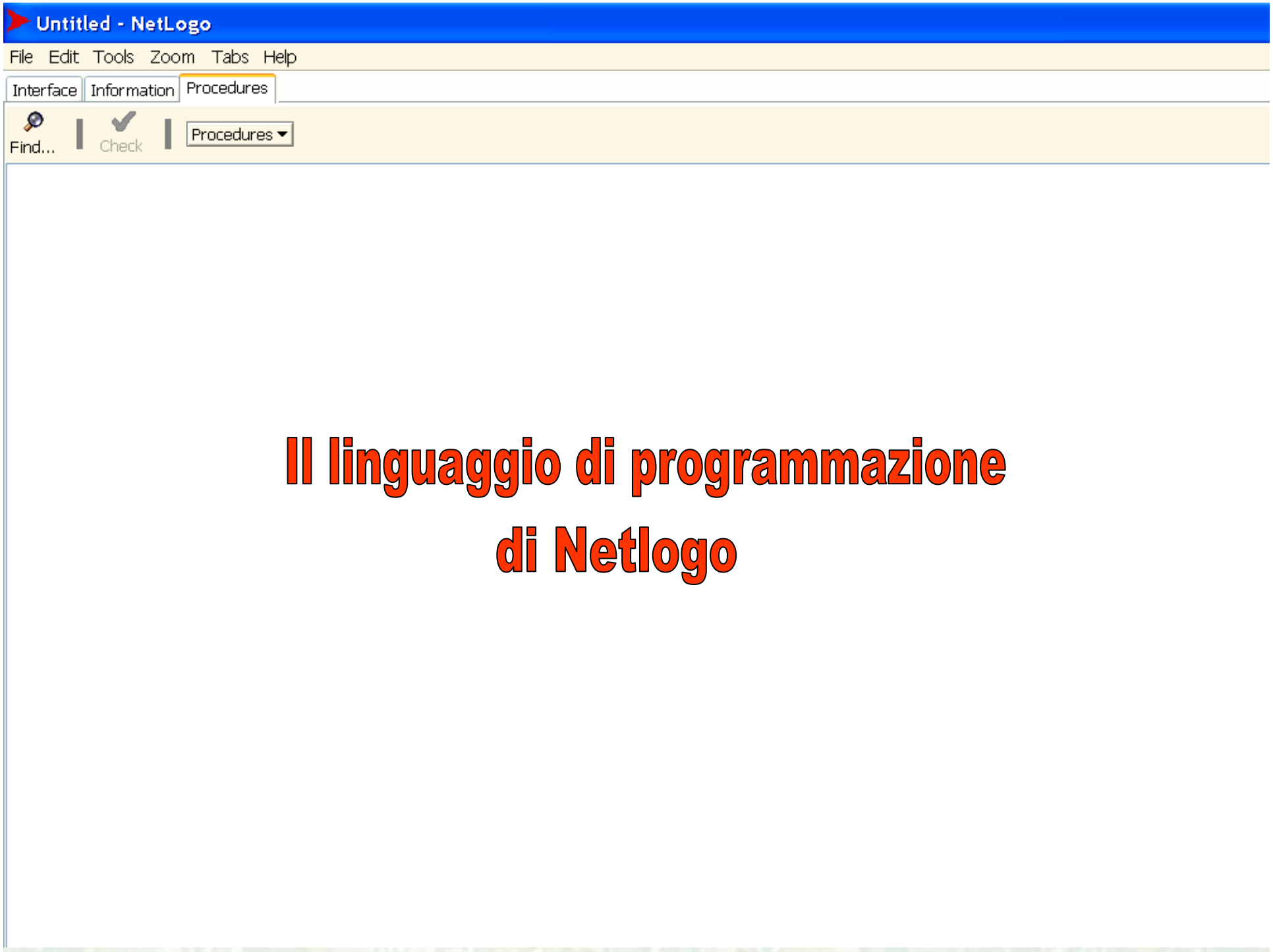
flessibilitá e adattabilitá

dinamica "at the edge of chaos"

auto-organizzazione, assenza di leader e controllo distribuito

comportamenti globali imprevedibili emergenti da regole locali deterministiche

presenza di soglie critiche



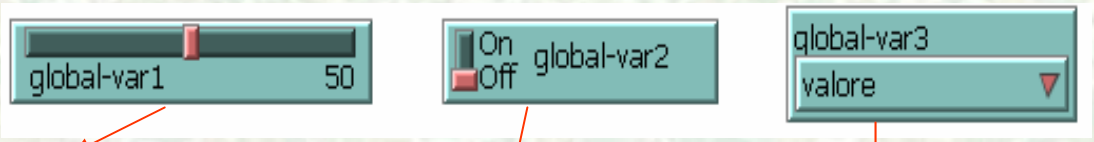
Il linguaggio di programmazione di Netlogo

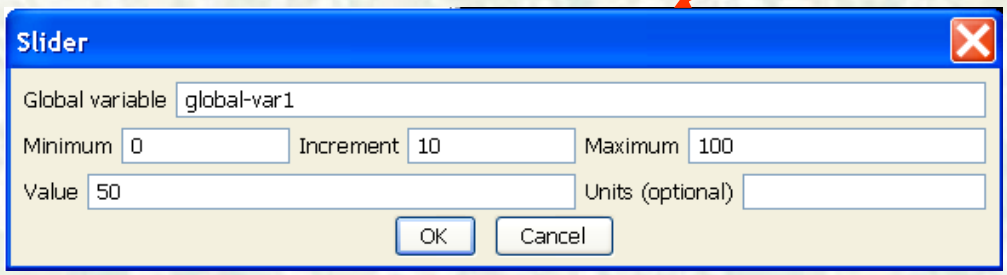
Il linguaggio di programmazione: le variabili

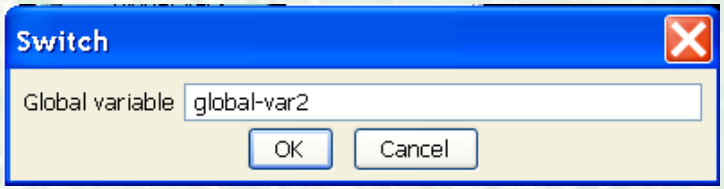
In NETLOGO si possono definire 3 tipi di variabili:

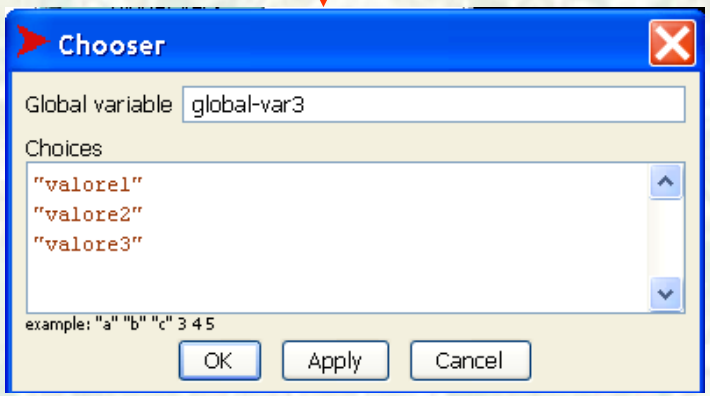
Variabili Globali

Esistono in esemplari unici e sono visibili da qualunque punto del codice. Vengono dichiarate fuori da ogni altra procedura e aggiornate col comando “set nome_var valore” oppure direttamente dall’interfaccia di sviluppo attraverso gli Sliders, gli Switch o i Chooser









NetLogo

Il linguaggio di programmazione: le variabili

Variabili Locali

Esistono anch'esse in esemplari unici e sono visibili solo all'interno della routine dove sono state dichiarate con "let nome_var valore"

NetLogo

Il linguaggio di programmazione: le variabili

Variabili Proprietarie

Appartengono alle turtles (turtles-own) o alle patches (patches-own) e sono replicate per ciascuna di esse (in JAVA sarebbero attributi degli oggetti appartenenti alle classi ‘turtles’ e ‘patches’).

Variabili Proprietarie: esempi

turtles-own

who
xcor, ycor
heading
color
shape
size
label
turtles-own [var1 var2 ...]

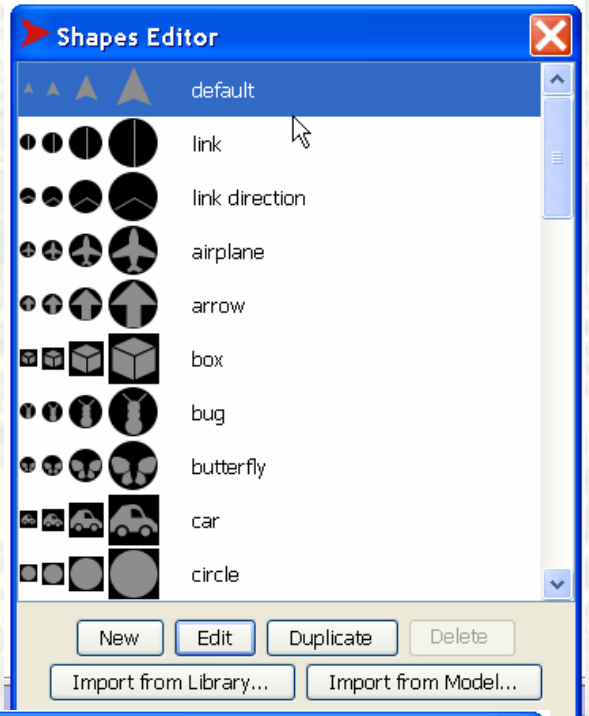
patches-own

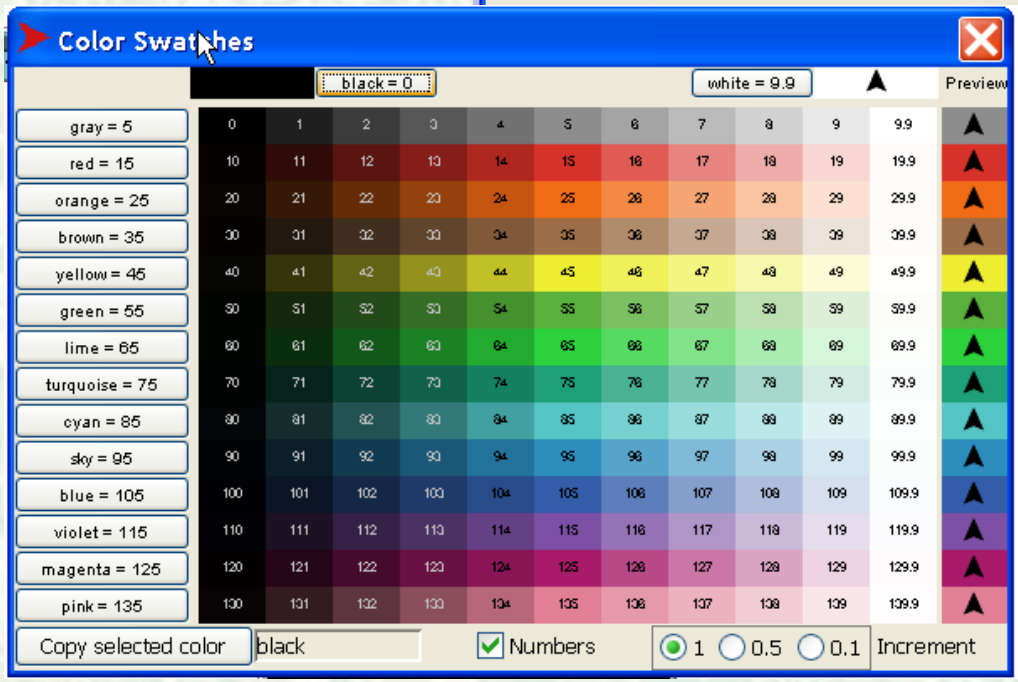
pxcor, pycor
pcolor
plabel
plabel-color
patches-own [pvar1 pvar2 ...]

breeds-own

breed [boxes box]
breed [cars car]

agentset



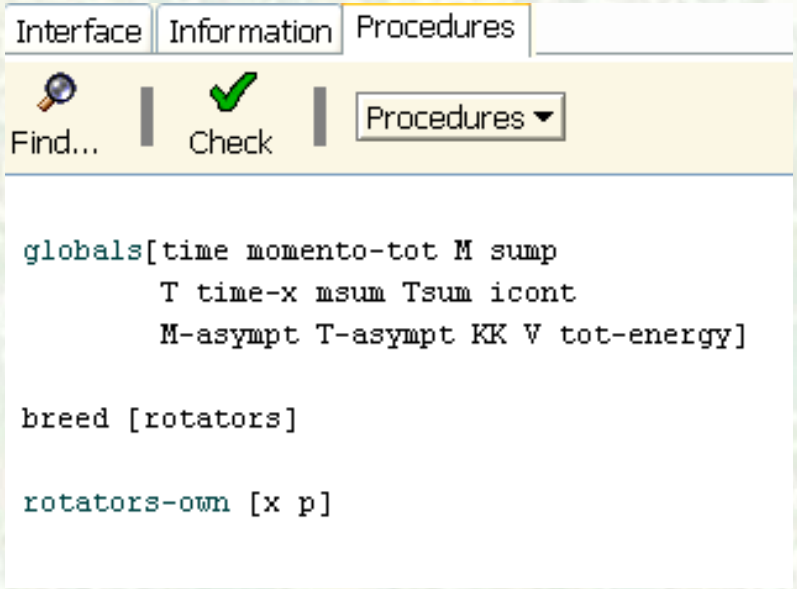


NetLogo

Il linguaggio di programmazione: le dichiarazioni

Dichiarazioni di variabili

Precedono le procedure vere e proprie e servono a dichiarare le variabili globali, le 'breeds' e le variabili proprietarie. Il loro corpo é delimitato da parentesi quadre.



NetLogo

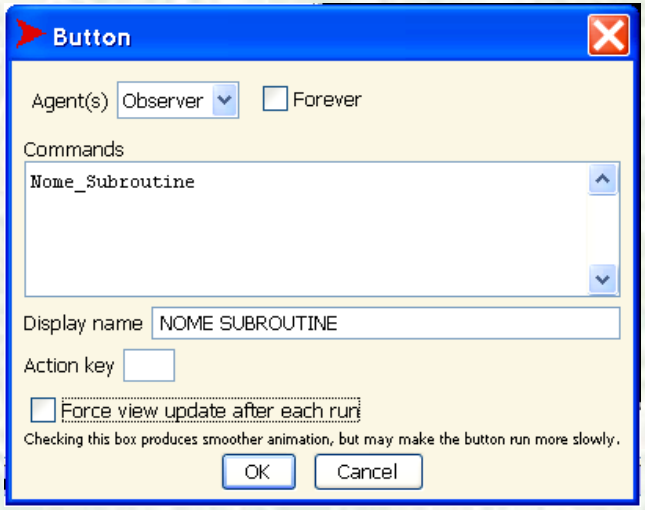
Il linguaggio di programmazione: le procedure

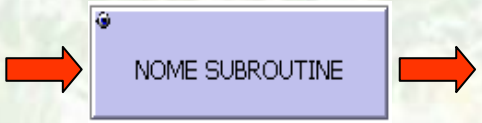
In NETLOGO tutto il codice del modello risiede nello stesso listato, suddiviso in procedure che possono essere destinate all'esecuzione da parte del modello, delle patches o delle turtles.

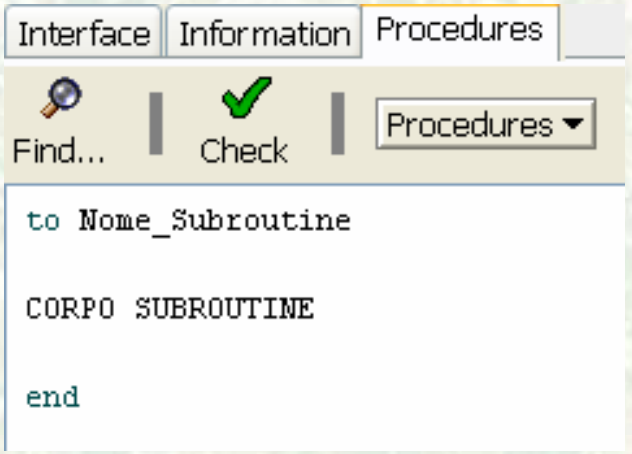
Le procedure possono essere di due tipi diversi:

Subroutines

Sono richiamate da altre subroutines oppure direttamente dall'interfaccia grafica per mezzo della pressione di un 'button'. Possono prendere valori in ingresso ma non restituiscono alcun valore in uscita. Iniziano sempre con la parola chiave "to" e finiscono con "end".







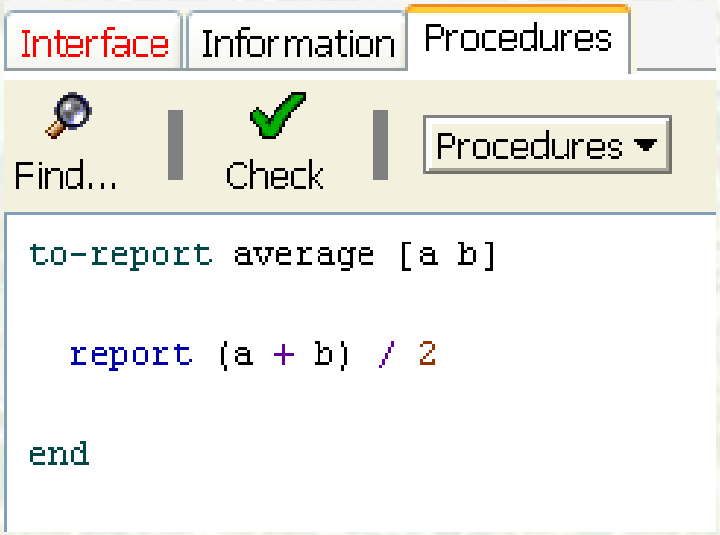
Il linguaggio di programmazione: le procedure

In NETLOGO tutto il codice del modello risiede nello stesso listato, suddiviso in procedure che possono essere destinate all'esecuzione da parte del modello, delle patches o delle turtles.

Le procedure possono essere di due tipi diversi:

Reporters

Sono chiamati da altri reporters o subroutines. Possono prendere valori in ingresso e restituiscono sempre un valore in uscita, attraverso il comando 'report'. Iniziano sempre con la parola chiave "to-report" e finiscono con "end".



NetLogo

Il linguaggio di programmazione: le procedure

Le due principali subroutines, presenti in quasi tutte le simulazioni create con Netlogo sono:

Setup

Chiamata dal relativo pulsante sull'interfaccia grafica: serve ad inizializzare le variabili, lo spazio virtuale e gli eventuali grafici. Serve anche:
- a creare le turtles che verranno usate nella simulazione e a fissarne le condizioni iniziali;
- ad inizializzare le patches e gli altri elementi d'ambiente;

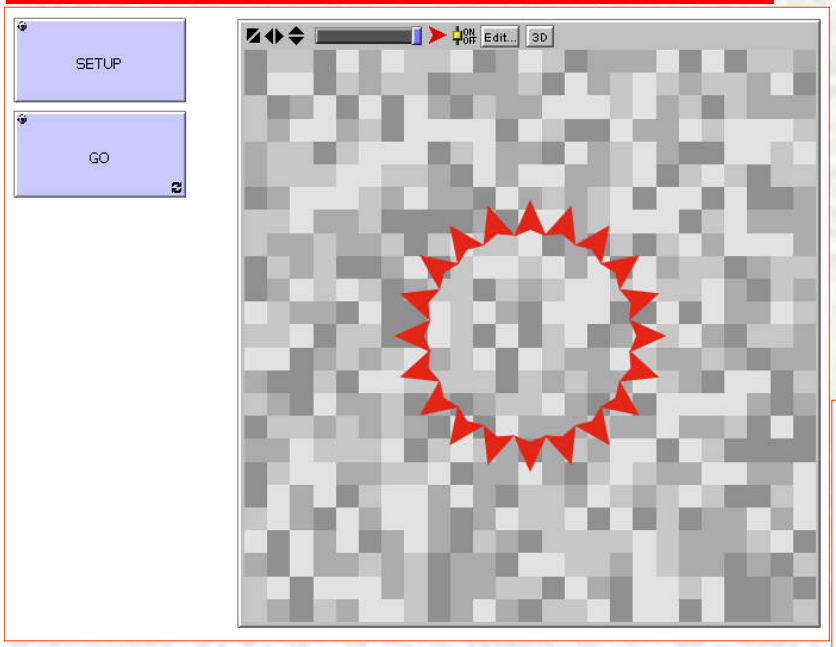
Go

Chiamata dal relativo pulsante sull'interfaccia grafica: serve a fare partire la simulazione, a far muovere le turtles o a cambiare stato alle patches.

In realtá di solito contiene le istruzioni per eseguire un singolo run della simulazione. Attivando peró la casella 'Forever' si impone alla routine di ripetersi fino a quando non venga nuovamente premuto il pulsante GO.

NetLogo

Il linguaggio di programmazione: le procedure



Il comando ‘crt *n*’ crea *n* turtles nell’origine. In alternativa ‘sprout *n* [commands]’ crea *n* turtles sulla patch corrente ed esegue i comandi contenuti nel blocco.

```
;; viene eseguito una volta quando si preme il tasto SETUP
to setup
  ;; cancella tutte le turtle e resetta le patches
  ca
  ;; crea 20 turtles
  crt 20
  ask turtles
    ;; chiede ad ogni turtle di assumere colore rosso e variare dimensione
    [
      set color red
      set size 2
      fd 5
    ]
  ask patches
    ;; chiede alle patches di assumere un colore random in scala di grigio
    [ set pcolor (5 + random 4) ]
end

;; forever button; viene ripetuto finché non si ripreme il tasto GO
to go
  ask turtles
  [
  ;; chiede ad ogni turtle di avanzare di un passo ed aspettare
    fd 1 wait 0.25
  ]
end
```

Uno dei comandi principali di Netlogo é la parola chiave ‘ask’, applicabile sia alle turtles che alle patches.

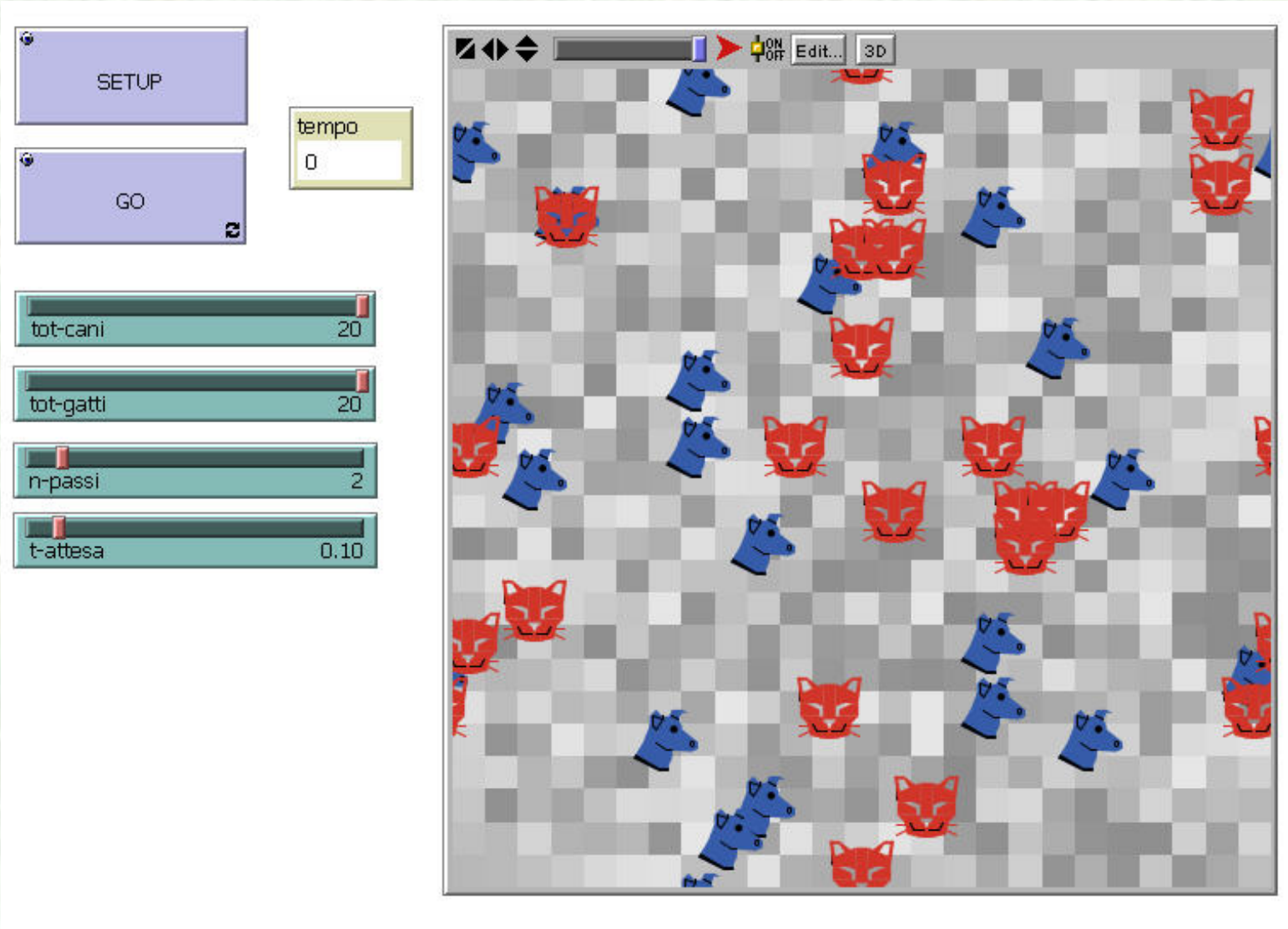
Il comando ‘fd’ é l’abbreviazione di ‘forward’ e fa avanzare le turtles

Vediamo ora il listato di un nuovo modello di esempio che arricchisce il precedente introducendo due sottospecie (breeds) di turtles, i cani e i gatti, che avranno un comportamento diverso:

i gatti si muoveranno a caso;

i cani invece si muoveranno cercando di inseguire uno dei gatti ad essi piú vicini.

```
globals[ time ] ;; dichiarazione delle variabili globali

breed [cani]  ;; definisce la specie 'cani'
breed [gatti] ;; definisce la specie 'gatti'

;; viene eseguito una volta quando si preme il tasto SETUP
to setup
  ca   ;; cancella tutte le turtles e resetta le patches
  set time 0
  set-default-shape cani "dog"  ;; assegna alla specie cani la shape 'dog'
  set-default-shape gatti "cat"  ;; assegna alla specie gatti la shape 'cat'

  ask n-of tot-cani patches  ;; sceglie un numero 'tot-cani' di patches a caso sulla griglia
    [
      sprout-cani 1 [ set color blue  set size 2 ] ;; crea un cane blu di size 2 su ciascuna patch
    ]
  ask n-of tot-gatti patches  ;; sceglie un numero 'tot-gatti' di patches a caso sulla griglia
    [
      sprout-gatti 1 [ set color red  set size 2 ] ;; crea un gatto rosso di size 2 su ciascuna patch
    ]
  ask patches  ;; chiede alle patches di assumere un colore random in scala di grigio
    [ set pcolor (5 + random-float 4) ]
end

to go ;; forever button; viene ripetuto finché non si ripreme il tasto GO
  ask cani
  [ ;; chiede ad ogni cane di puntare un gatto a caso in un raggio di 10 patches, di avanzare
    ;; di n-passi verso di lui e di aspettare una frazione di secondo pari a t-attesa
    set heading towards target-cani 10 fd n-passi wait t-attesa
  ]
  ask gatti
  [ ;; chiede ad ogni gatto di ruotare a destra di una quantitá casuale di gradi, di avanzare
    ;; di n-passi e di aspettare una frazione di secondo pari a t-attesa  [
    rt random 360 fd n-passi wait t-attesa
  ]
  set time (time + 1)
end

to-report target-cani [raggio-intorno]          ;; restituisce uno dei gatti presenti in un raggio
 report one-of gatti in-radius raggio-intorno  ;; di 10 patches attorno al cane
end
```


SETUP
GO
tempo
0
tot-cani
20
tot-gatti
20
n-passi
2
t-attesa
0.10
Edit...
3D

Arrivati a questo punto, potremmo sentire l'esigenza di complicare ulteriormente il comportamento dei cani e dei gatti, per esempio facendo sí che i cani 'mangino' gli eventuali gatti che finiscono sulla loro stessa patch e che i gatti cerchino di sfuggire agli attacchi dei cani.

Basta poco peró per rendersi conto che ci mancano ancora degli importanti elementi di programmazione indispensabili per consentire agli agenti di compiere simili scelte: passiamo dunque ad introdurre le cosiddette 'strutture per il controllo di flusso'.

NetLogo

Il linguaggio di programmazione: strutture di controllo di flusso

if *condition* [*commands*]

...dove *condition* é un'espressione boolena

ifelse *condition* [*commands1*] [*commands2*]

ifelse-value *condition* [*value1*] [*value2*]

loop [*commands*]

...per uscire dal loop si usa il comando stop

repeat *number* [*commands*]

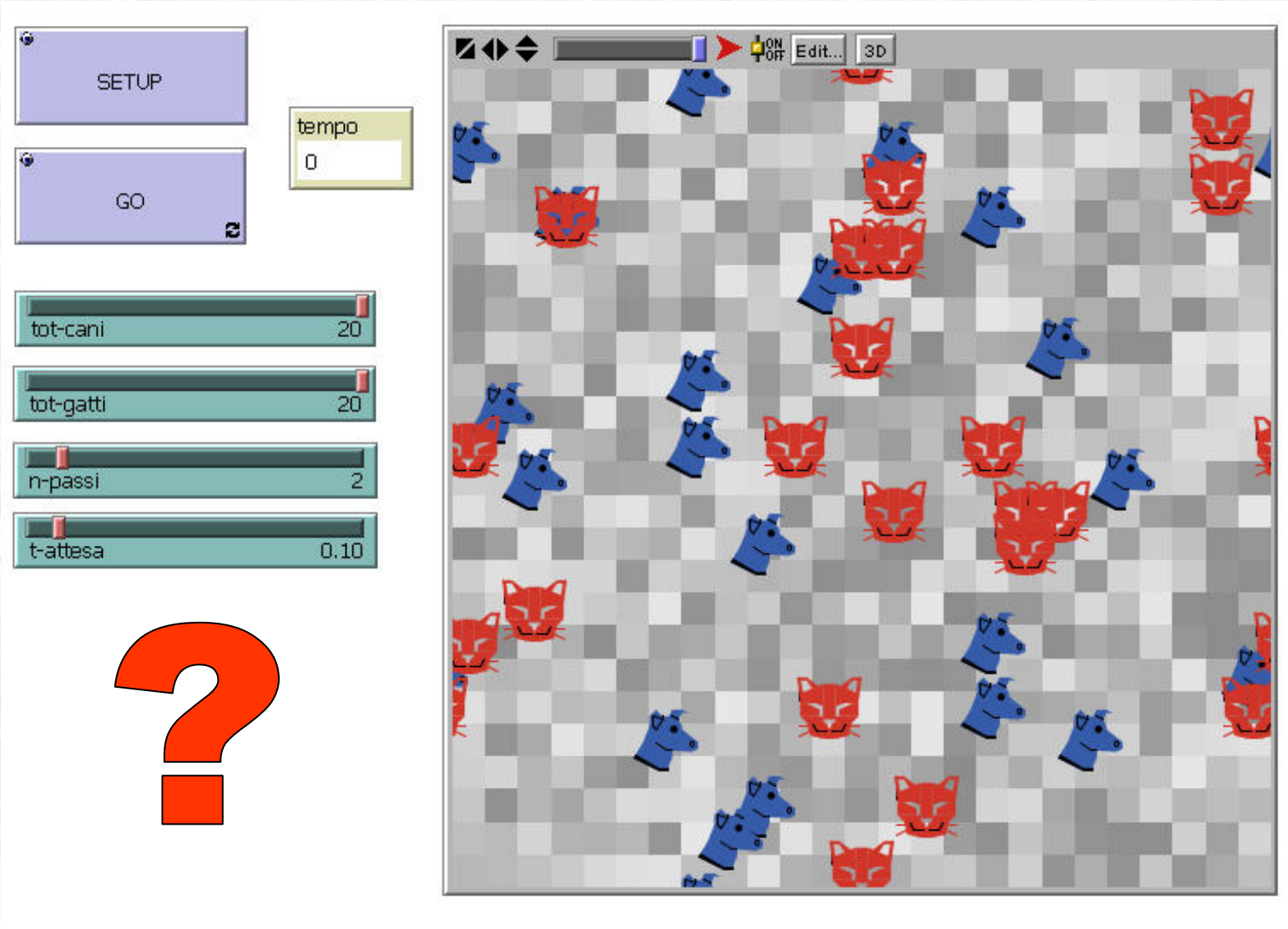
while [*conditions*] [*commands*]

foreach *list* [*commands*]

sono in pratica degli array

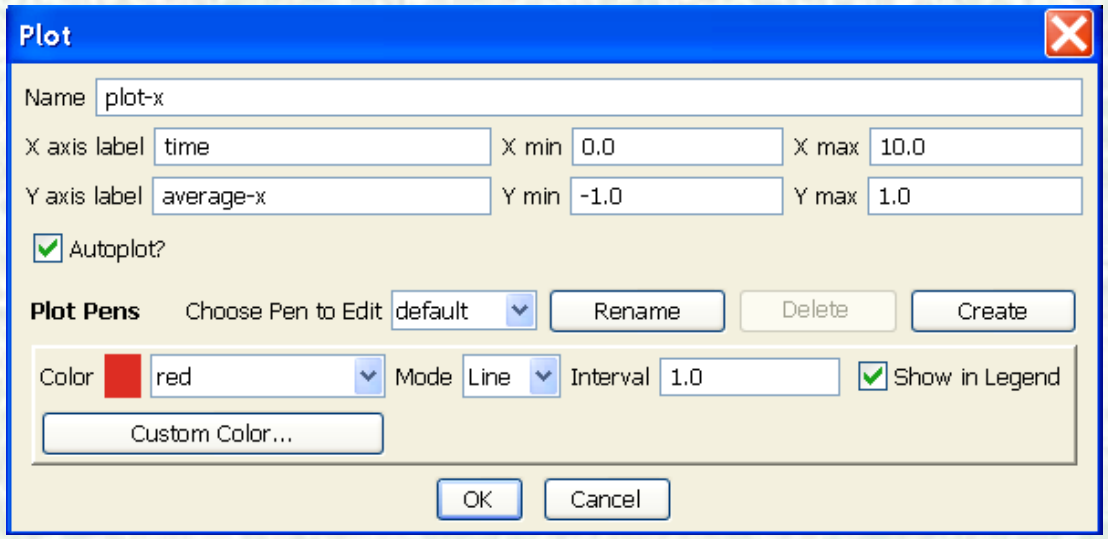
```
;; si applica agli elementi di 'liste' tipo
;; mylist = [2 7 10 "Bob"] . Con:
set mylist lput 42 mylist
;; mylist diventa [2 7 10 "Bob" 42]
```

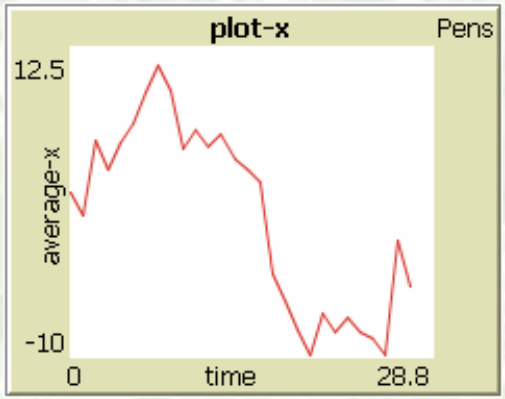
```
to go ;; forever button; viene ripetuto finché non si ripreme il tasto GO
  ask cani
  [ ;; chiede ad ogni cane di puntare un gatto a caso in un raggio di 10 patches, di avanzare
    ;; di n-passi verso di lui e di aspettare una frazione di secondo pari a t-attesa
    set heading towards target-cani 10 fd n-passi wait t-attesa
    if (any? gatti-here)                ;; se ci sono gatti sulla patch su cui si trova il cane
        [ ask one-of gatti-here [die] ]  ;; uno a caso di essi viene eliminato
  ]
  ask gatti
  [ ;; chiede ad ogni gatto di individuare un cane a caso in un raggio di 3 patches da lui, di
    ;; puntare nella direzione opposta a quella del cane, di avanzare di n-passi e di aspettare
    ;; una frazione di secondo pari a t-attesa
    set heading towards target-gatti 3 rt 180 fd n-passi wait t-attesa
  ]
  set time (time + 1)
end

to-report target-cani [raggio-intorno]
  ifelse (any? gatti in-radius raggio-intorno)  ;; controlla se ci sono gatti nell'intorno considerato
     [ report one-of gatti in-radius raggio-intorno] ;; se ce ne sono ne restituisce uno come target
     [ let target 0                                  ;; se non ci sono gatti restituisce come target
       ask patch-here [set target one-of neighbors]  ;; una delle 8 patches attorno alla patch su cui
       report target]                                ;; si trova il cane che ha chiamato il reporter
end

to-report target-gatti [raggio-intorno]
  ifelse (any? cani in-radius raggio-intorno)  ;; controlla se ci sono cani nell'intorno considerato
     [ report one-of cani in-radius raggio-intorno] ;; se ce ne sono ne restituisce uno come target
     [ let target 0                                 ;; se non ci sono cani restituisce come target
       ask patch-here [set target one-of neighbors]  ;; una delle 8 patches attorno alla patch su cui
       report target]                                ;; si trova il gatto che ha chiamato il reporter
end
```


SETUP
GO
tempo
0
tot-cani
20
tot-gatti
20
n-passi
2
t-attesa
0.10
?
Edit...
3D

NetLogo

Il linguaggio di programmazione: i grafici





Inizializzazione del grafico

```
set-current-plot "plot-x"
set-plot-x-range 0 10
set-plot-y-range -1.0 1.0
```

plot *number*

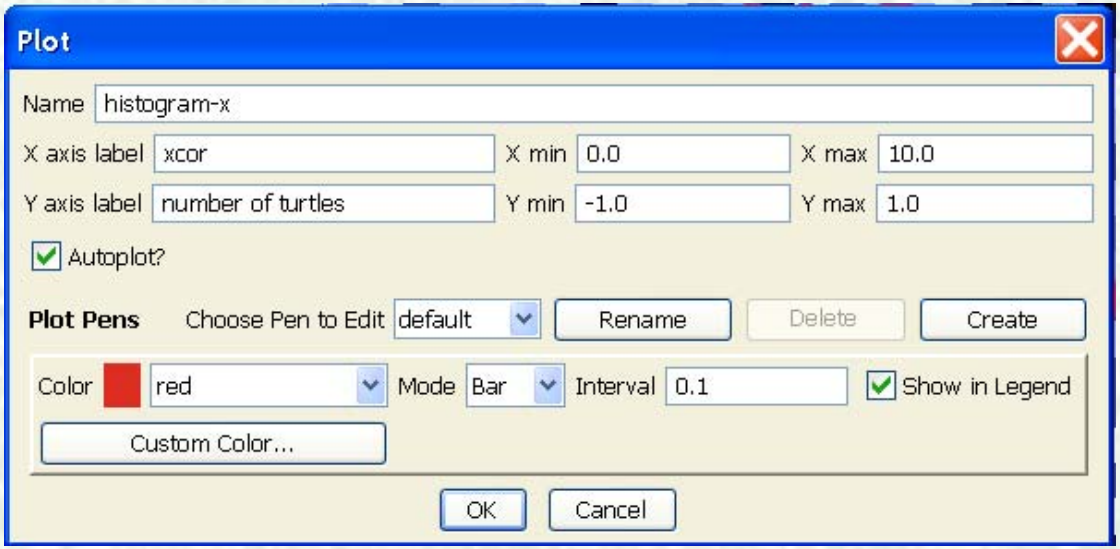
Incrementa il valore x della *plot-pen* corrente di un intervallo *plot-pen-interval*, e poi disegna un punto (o tratto di linea, o barra) al nuovo valore di x e al valore di y pari al valore della variabile *number*.

plotxy *number1 number2*

Muove la plot-pen corrente al punto di coordinate (*number1*, *number2*). Se la plot-pen é down, verrá disegnata una linea, una barra, o un punto (a seconda del *modo* della penna).

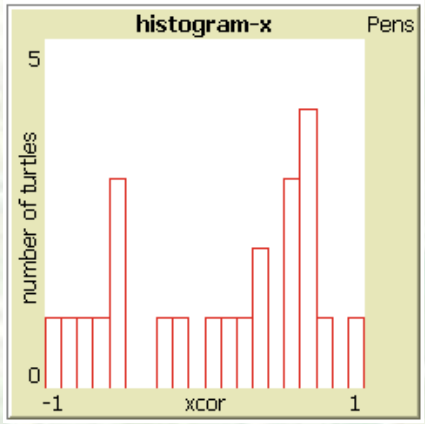
NetLogo

Il linguaggio di programmazione: gli istogrammi



histogram-from *agentset* [*variable*]

Disegna un istogramma mostrando la distribuzione di frequenza (pdf) dei valori della variabile proprietaria *variable* di un certo *agentset* (ad esempio *turtles*). L'altezza delle barre rappresenta il numero di agenti con il valore della variabile compreso in un dato range.



Inizializzazione dell'istogramma

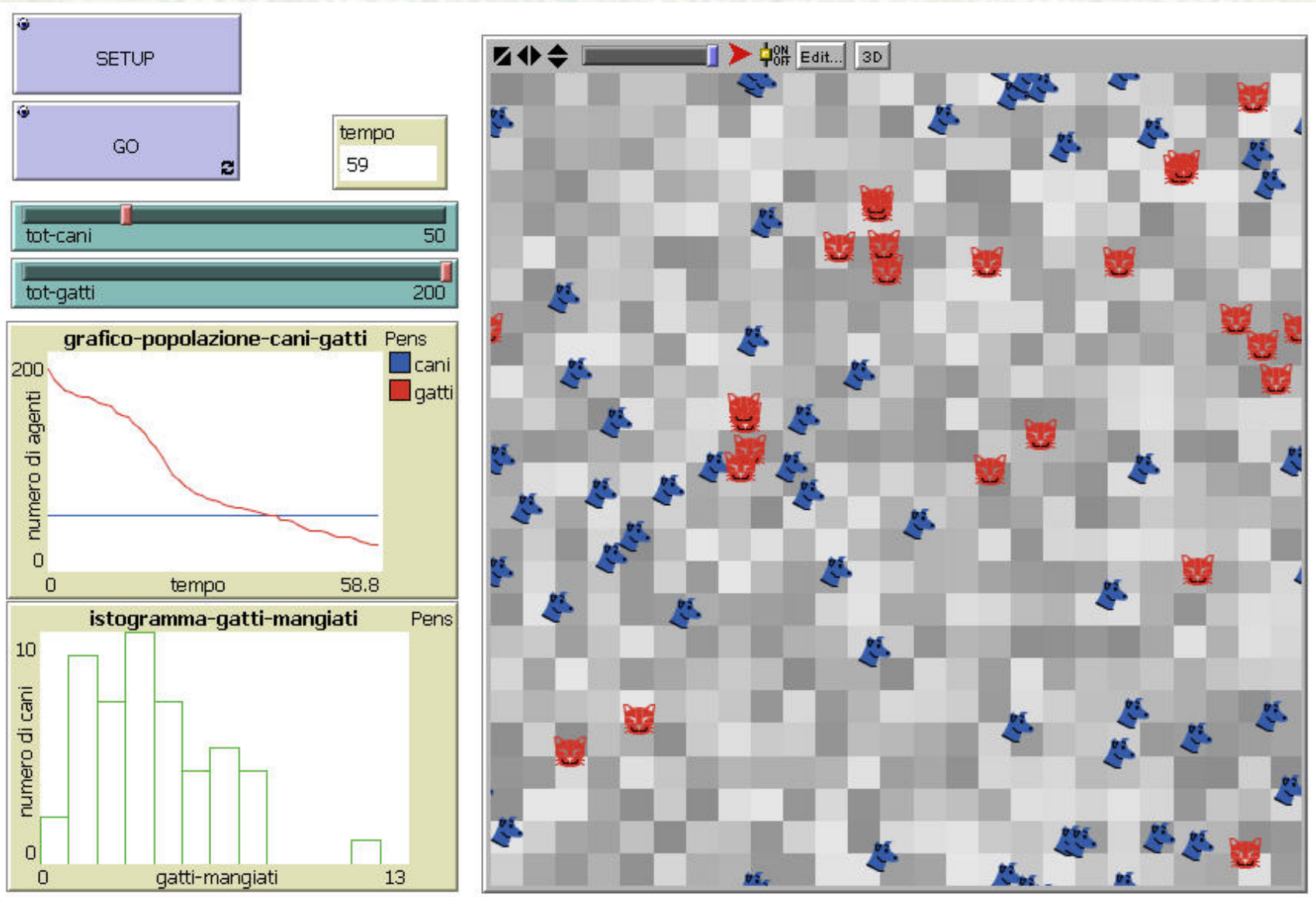
```
set-current-plot "histogram-x"
set-plot-x-range -1.0 1.0
set-plot-y-range 0 1
```

histogram-list *list*

Come sopra, solo che i valori per creare l'istogramma sono presi da una lista (array).

```
globals[ time ] ;; dichiarazione delle variabili globali

breed [cani]  ;; definisce la specie 'cani'
breed [gatti] ;; definisce la specie 'gatti'
cani-own [gatti-mangiati]

to setup
  ca   ;; cancella tutte le turtles e resetta le patches
  set time 0
  set-default-shape cani "dog"  ;; assegna alla specie cani la shape 'dog'
  set-default-shape gatti "cat"  ;; assegna alla specie gatti la shape 'cat'

  set-current-plot "grafico-popolazione-cani-gatti"
  set-plot-x-range 0 10
  set-plot-y-range 0 ifelse-value (tot-cani > tot-gatti) [tot-cani] [tot-gatti]

  set-current-plot "istogramma-gatti-mangiati"
  set-plot-x-range 0 10
  set-plot-y-range 0 1

  ask n-of tot-cani patches  ;; sceglie un numero 'tot-cani' di patches a caso sulla griglia
    [
      sprout-cani 1 [ set color blue  set size 1 set gatti-mangiati 0] ;; crea un cane blu di size 2 su ciascuna patch
    ]
  ask n-of tot-gatti patches  ;; sceglie un numero 'tot-gatti' di patches a caso sulla griglia
    [
      sprout-gatti 1 [ set color red  set size 1 ] ;; crea un gatto rosso di size 2 su ciascuna patch
    ]
  ask patches  ;; chiede alle patches di assumere un colore random in scala di grigio
    [ set pcolor (5 + random-float 4) ]
end
```

```
to go
  ask cani
  [ ;; chiede ad ogni cane di puntare un gatto a caso in un raggio di 10 patches, di avanzare
    ;; di 1 passo verso di lui
    set heading towards target-cani 10 fd 1
    if (any? gatti-here)
    [ ask one-of gatti-here [die]              ;; se ci sono gatti sulla patch su cui si trova il cane
      set gatti-mangiati (gatti-mangiati + 1)] ;; uno a caso di essi viene eliminato
  ]
  ask gatti
  [ ;; chiede ad ogni gatto di individuare un cane a caso in un raggio di 3 patches da lui, di
    ;; puntare nella direzione opposta a quella del cane e di avanzare di 1
    set heading towards target-gatti 3 rt 180 fd 1
  ]

  set-current-plot "grafico-popolazione-cani-gatti"
  set-current-plot-pen "cani"
  plot (count cani)
  set-current-plot-pen "gatti"
  plot (count gatti)

  set-current-plot "istogramma-gatti-mangiati"
  set-plot-x-range 0 value-from (max-one-of cani [gatti-mangiati]) [gatti-mangiati]
  set-plot-y-range 0 1
  histogram-from cani [gatti-mangiati]

  set time (time + 1)
end
```


SETUP
GO
tempo
59
tot-cani
50
tot-gatti
200
grafico-popolazione-cani-gatti
Pens
cani
gatti
200
numero di agenti
0
0
tempo
58.8
istogramma-gatti-mangiati
Pens
10
numero di cani
0
0
gatti-mangiati
13
Edit...
3D

Primitives Dictionary

Alphabetical: A B C D E F G H I J L M N O P R S T U V W X Y ?

Categories: Turtle - Patch - Agentset - Color - Control/Logic - World - Perspective
Input/Output - Files - List - String - Math - Plotting - Links - Movie - System - HubNet

Special: Variables - Keywords - Constants

Categories of Primitives

This is an approximate grouping. Remember that a turtle-related primitive might still be called by patches or observers, and vice versa. To see which agent (turtles, patches, observer) can actually run each command, consult each individual entry in the dictionary.

Turtle-related

back (bk) <*breeds*>-at <*breeds*>-here <*breeds*>-on can-move? clear-turtles (ct) create-<*breeds*> create-custom-<*breeds*> create-custom-turtles (cct) create-turtles (crt) die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch hatch-<*breeds*> hideturtle (ht) home inspect is-<*breed*>? is-turtle? jump left (lt) myself nobody -of other-turtles-here other-<*breeds*>-here patch-ahead patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead pen-down (pd) pen-erase (pe) pen-up (pu) random-xcor random-ycor right (rt) self set-default-shape __set-line-thickness setxy shapes showturtle (st) sprout sprout-<*breeds*> stamp stamp-erase subject subtract-headings __tie towards towardsxy turtle turtles turtles-at turtles-from turtles-here turtles-on turtles-own __untie uphill value-from

Patch-related primitives

clear-patches (cp) diffuse diffuse4 distance distancexy import-pcolors inspect is-patch? myself neighbors neighbors4 nobody nsum nsum4 -of patch patch-at patch-ahead patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead patches patches-from patches-own random-pxcor random-pycor self sprout sprout-<*breeds*> subject value-from

Agentset primitives

any? ask at-points <*breeds*>-at <*breeds*>-here <*breeds*>-on count histogram-from in-cone in-radius is-agent? is-agentset? is-patch-agentset? is-turtle-agentset? max-one-of min-one-of n-of neighbors neighbors4 one-of other-turtles-here other-<*breeds*>-here patches patches-from sort sort-by turtles with with-max with-min turtles-at turtles-from turtles-here turtles-on values-from

Color primitives

color extract-hsb extract-rgb hsb import-pcolors pcolor rgb scale-color shade-of? wrap-color

Control flow and logic primitives

and carefully end error-message foreach if ifelse ifelse-value let loop map not or repeat report run runresult ; (semicolon) set stop startup to to-report wait while without-interruption xor

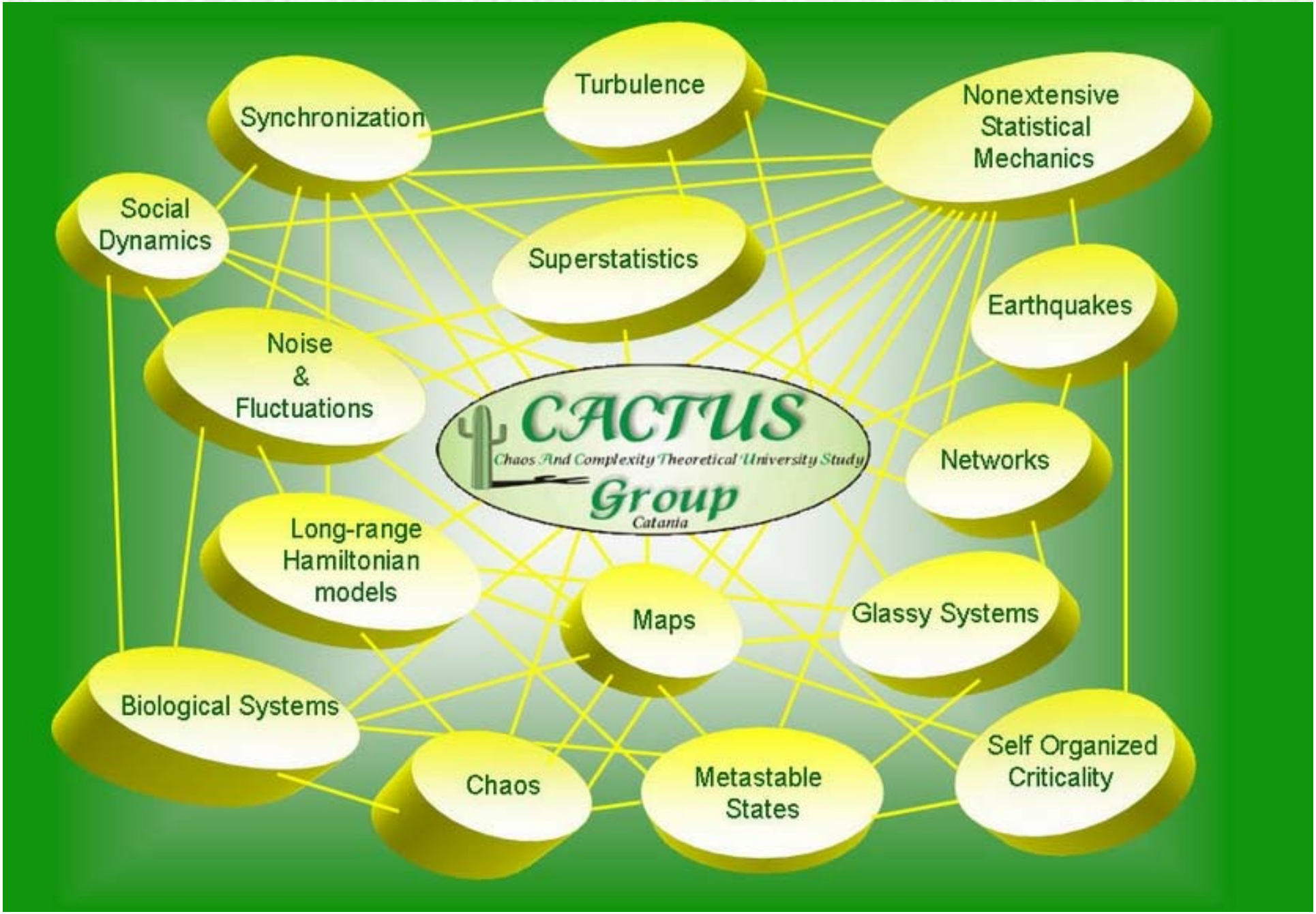
World primitives

clear-all (ca) clear-drawing (cd) clear-patches (cp) clear-turtles (ct) display import-drawing import-pcolors no-display max-pxcor max-pycor min-pxcor min-pycor world-width world-height

Perspective primitives

follow follow-me reset-perspective (rp) ride ride-me subject watch watch-me

HubNet primitives

hubnet-broadcast hubnet-broadcast-view hubnet-enter-message? hubnet-exit-message? hubnet-fetch-message hubnet-message hubnet-

http://www.ct.infn.it/cactus/