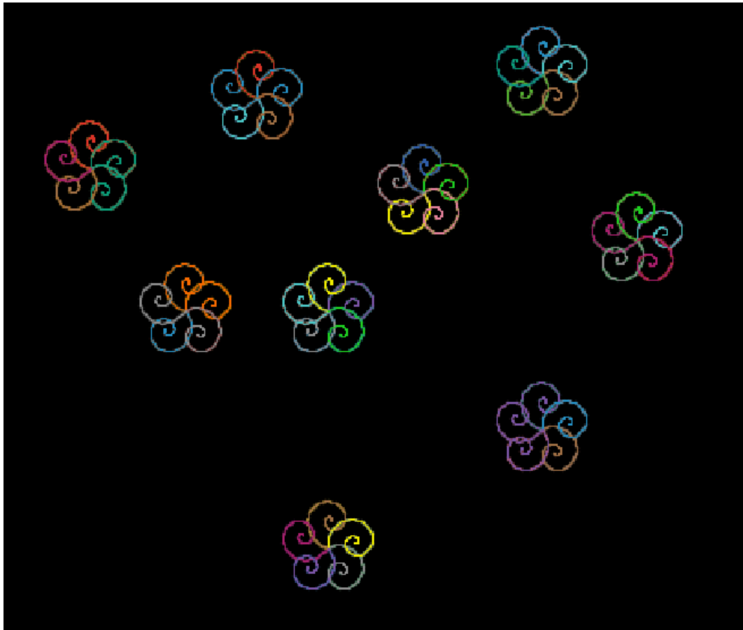
Introduzione alla modellizzazione ad agenti con NetLogo 3

- Mouse, plot, links,

Franco Bagnoli 2017

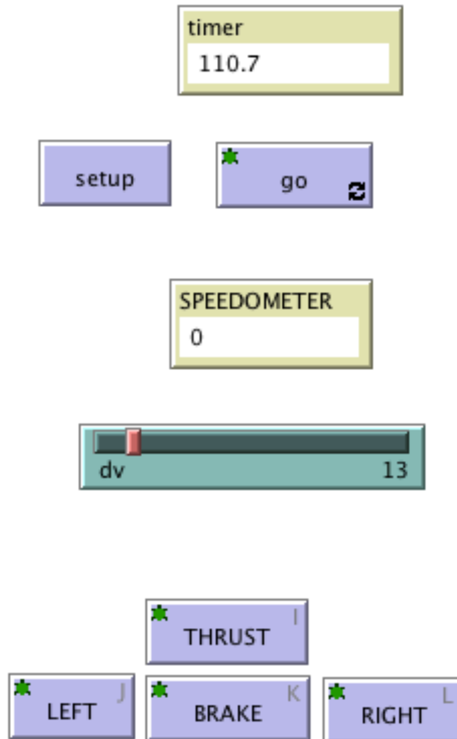
Esercizio con il mouse

Disegnare fiorellini.



```
to clear
  ca
end

to go
  if mouse-down? [
    ask patch mouse-xcor mouse-ycor [
      sprout 5 [
        set size 0
        set heading who * 360 / 5
        pd
        let i 0
        while [i < 80] [
          fd .2 * (1 - sin(i * 90 / 80))
          right 10
          set i i + 1
        ]
      ]
    ]
  ]
end
```

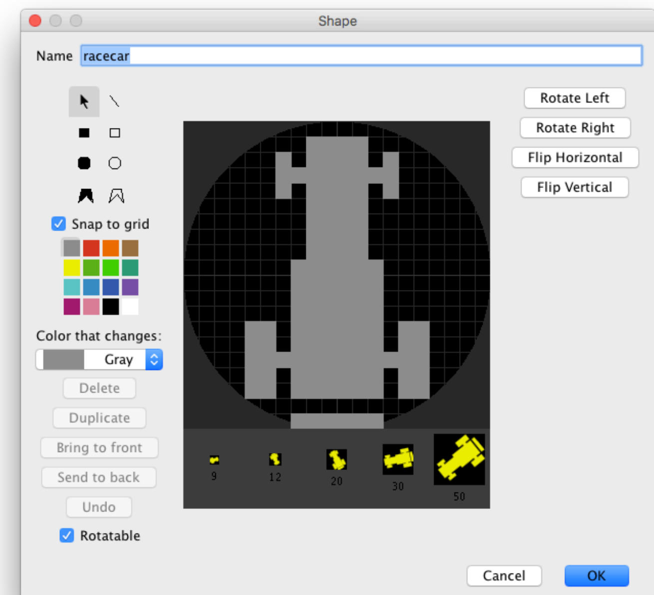
Racing



Racing

```
turtles-own [speed]
globals [started?]


to setup
  clear-all
  clear-turtles
  import-pcolors "racing.png"
  create-turtles 1 [
    set shape "racecar"
    set speed 0
    set size 10
    set heading 90
    set color grey
    setxy 8 -40
  ]
  set started? false
end


to go
  if not started? [
    set started? true
    reset-timer
    set speed dv / 1000000
  ]
  forward speed
  if pcolor = black [
    set color red
    user-message "you are dead"
    stop
  ]
  if shade-of? pcolor green [
    set color blue
    user-message (word "ce l'hai fatta in " timer "secondi!!")
    stop
  ]
end
```

```
to turn-left
  lt 5
  fd speed
end

to turn-right
  rt 5
  forward speed
end

to thrust
  set speed speed + dv / 1000000
end

to brake
  if speed > 0 [set speed speed - dv / 1000000]
end
```



Dynamical systems/Logistic map

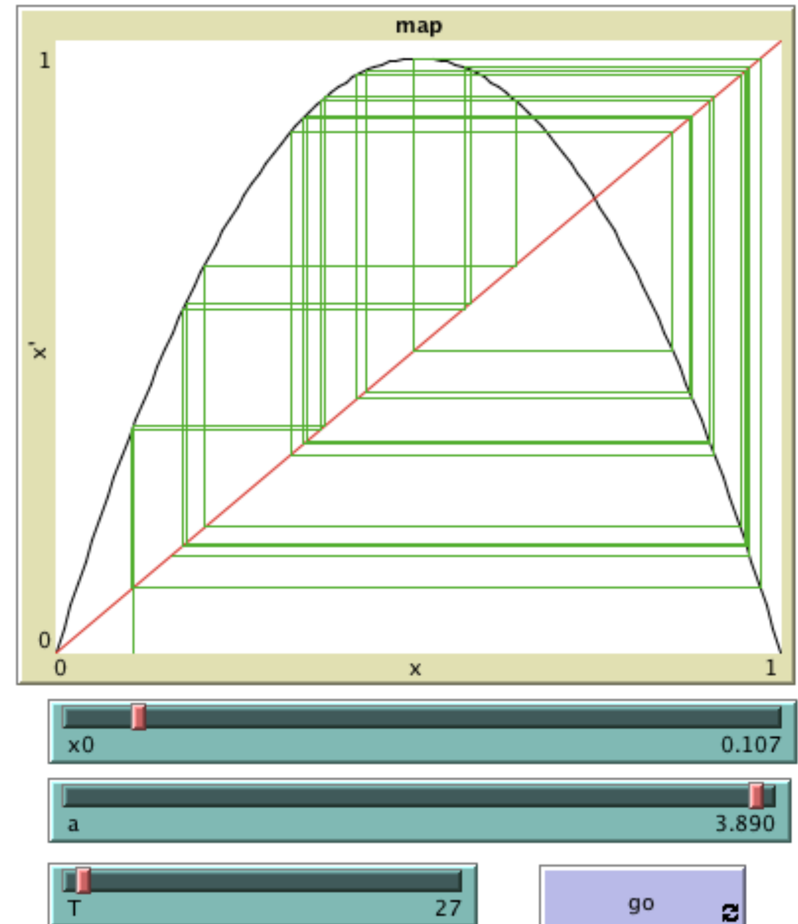
Prima di andare avanti, vediamo come si può usare NetLogo per visualizzare dei sistemi dinamici.

Proviamo a visualizzare un semplice sistema:

x(t+1) = f(x(t), a) ⬄ x' = f(x)

dove 0<x<1 e a è un parametro. Per esempio la mappa logistica f(x) = a*x*(1-x)

Vogliamo visualizzare la mappa nel grafico x-x'



Dynamical systems/Logistic map

Il codice è abbastanza semplice: si una un plot con varie penne per disegnare la mappa, la diagonale e il coweb partendo da x0.

Notare che non si usa il world (che è nascosto sotto il plot).

Notare anche che (purtroppo) non esiste l'istruzione for, bisogna utilizzare un while (e se uno si dimentica l'incremento della variabile non esce mai)

```
to go
  set-current-plot "map"
  clear-plot
  ; draw map
  let x 0
  while [x < 1.005] [
    plotxy x f(x)
    set x x + 0.01
  ]
  ; draw diagonal
  set-current-plot-pen "red"
  plotxy 0 0
  plotxy 1 1
  ; draw coweb
  set-current-plot-pen "green"
  set x x0
  let y 0
  plotxy x 0
  repeat T [
    set y f(x)
    plotxy x y
    plotxy y y
    set x y
  ]
end

to-report f [x]
  report a * x * (1 - x)
end
```

Diagramma di biforcazione della mappa logistica

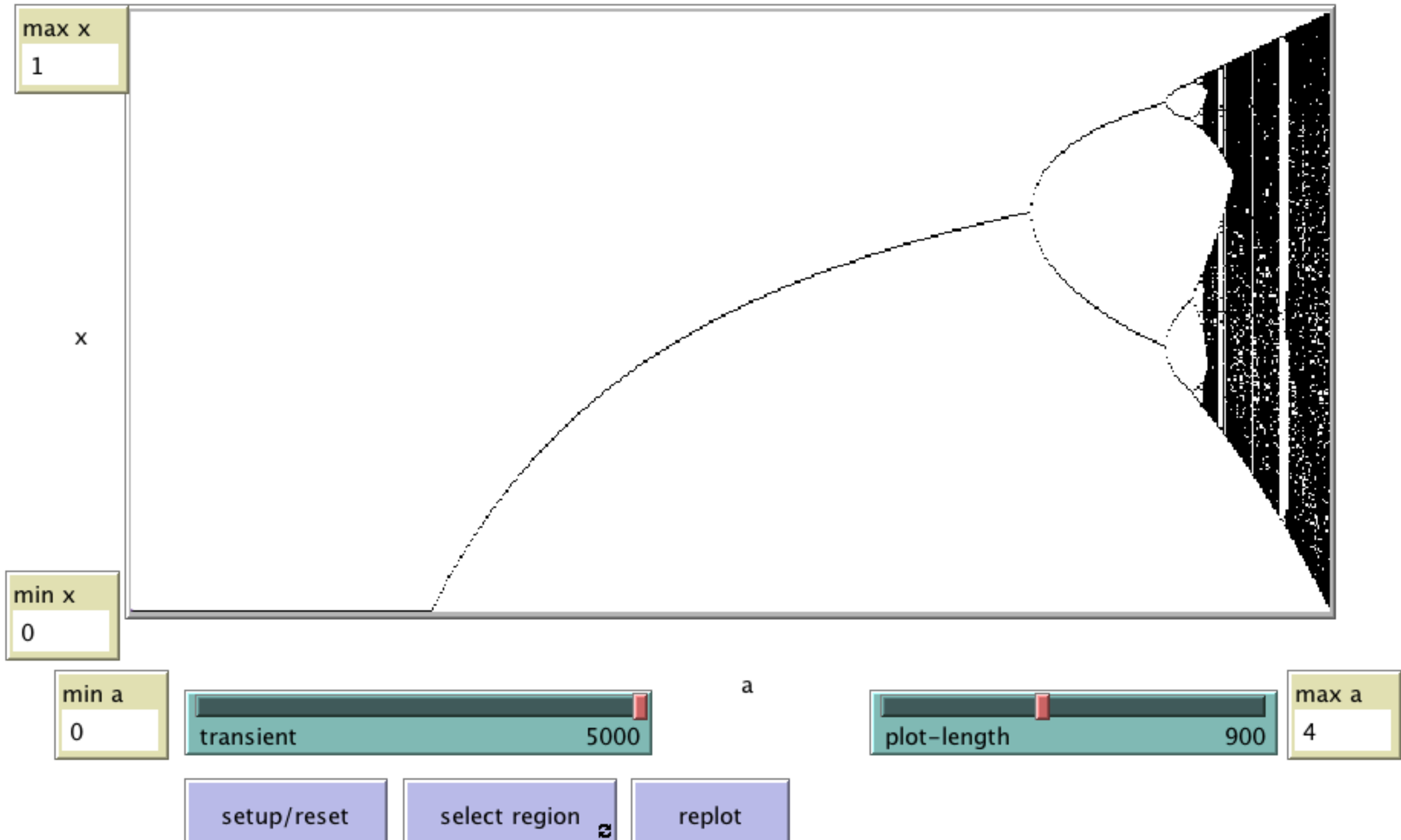


Diagramma di biforcazione della mappa logistica

Questa volta il disegno è fatto usando le patch nel world.

Il codice è alquanto complesso, copiato da un esempio nella model library

Giusto come esempio di cosa si può fare.

Mean-field vs real simulation

I sistemi dinamici (equazioni differenziali o mappe) rappresentano l'evoluzione MEDIA di un sistema, ovvero, con un linguaggio da fisici, l'approssimazione di CAMPO MEDIO o, con quello da chimici, un sistema ben "agitato" in cui gli individui saltano da una parte all'altra.

Ma anche se abbiamo un certo numero di collegamenti casuali "fissi" l'effetto small-world dà un comportamento quasi uguale al campo medio.

Mean-field vs real simulation

Prendiamo un automa cellulare probabilistico con R vicini. Detta x la probabilità di avere un “1” nel reticolo, l’equazione di campo medio è

$$x' = \sum_{k=0}^{\mathrm{R}} \binom{R}{k} x^k (1-x)^{R-k}\, \tau(k)$$

dove $\tau(k)$ è la probabilità di avere 1 se ho k vicini.

Adesso supponiamo di volere una equazione di campo medio uguale a quella della mappa logistica

$$x' = ax(1-x)$$

Come devo scegliere R minimo e $\tau(k)$?

Mean-field vs real simulation

Per R piccolo (per esempio 3) si ha

$$\tau_0 + (3\tau_1 - 3\tau_0)x + (3\tau_2 - 6\tau_1 + 3\tau_0)x^2$$
$$+ (\tau_3 - 3\tau_2 + 3\tau_1 - \tau_0)x^3 = ax - ax^2$$

da cui

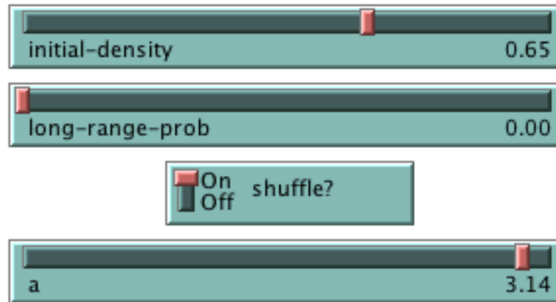
$$\tau_0 = 0$$
$$\tau_1 = a/3$$
$$\tau_2 = a/3$$
$$\tau_3 = 0$$

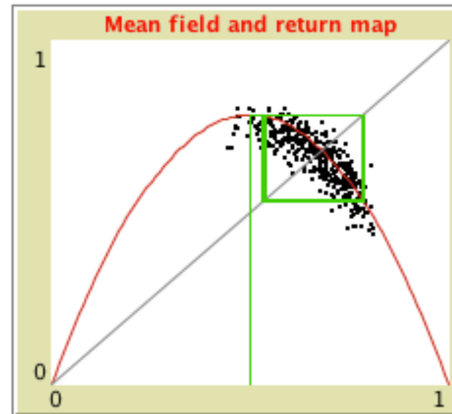
e quindi, dato che le τ_i devono essere minori di 1, a al massimo può essere uguale a 3, e non si vede nessuna oscillazione.

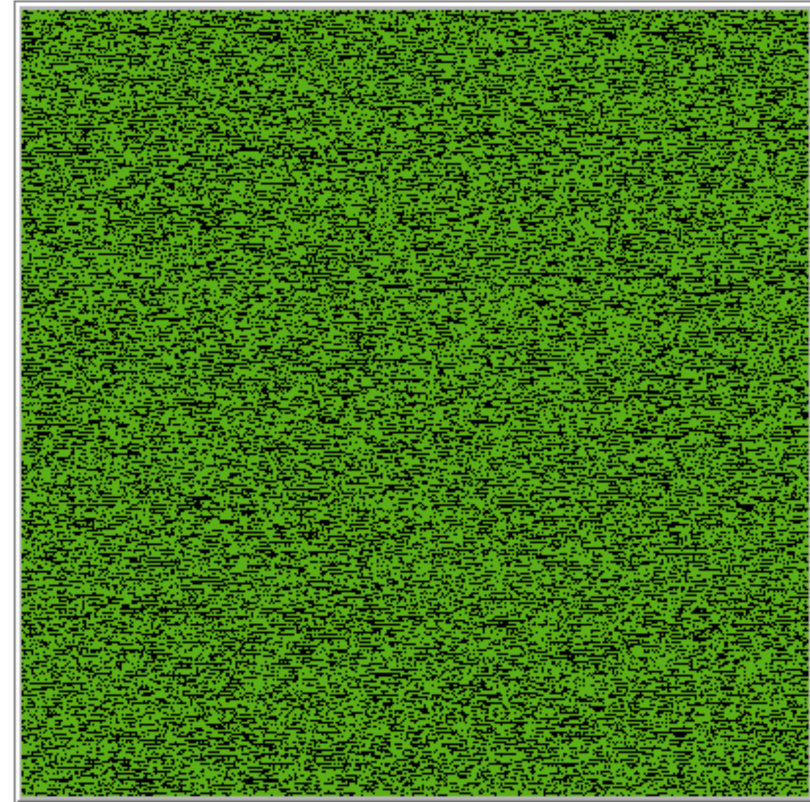
Per vedere qualcosa di interessante si deve andare almeno a $R = 5$, così che a può andare fino a 10/3 = 3.33

Mean-field vs real simulation

La cosa interessante è: cosa succede se prendiamo un reticolo regolare con 5 vicini? E se rimescoliamo gli spin a ogni passo?







E se invece non rimescoliamo nulla ma mettiamo una frazione di link a caso? (logisticCA.nlogo)

Mean-field vs real simulation

il codice è un po' lungo..

prima parte

```
globals [
  R          ;; range
  tau        ;; list of transition probability
  old-dens
]

patches-own [state neighs]

to setup
  clear-all
  clear-all-plots
  reset-ticks
  ; set R and tau
  set R 5
  set tau (list 0 (a / 5) (3 * a / 10) (3 * a / 10) (a / 5) 0)
  ;; setting up neighbourhood
  ask patches with [pycor = 0] [
    ;; implement different neighborhood
    set neighs n-values R [ ?1 ->  ifelse-value (random-float 1 < long-range-prob)
      [random world-width][int (R / 2) - ?1] ]
  ]
  ask patches with [pycor != 0] [
    set neighs [neighs] of patch pxcor 0
  ]
  ;; place cells across the top of the world
  ask patches with [pycor = (- (ticks mod world-height))] [
    set state ifelse-value ((random-float 1) < initial-density) [1][0]
    color-patch
  ]
  set old-dens (sum [state] of patches with
    [pycor = (- (ticks mod world-height))]) / world-width
  mean-field-plot
end
```

Mean-field vs real simulation

seconda parte

```
to mean-field-plot
  ; mean field plot
  let x 0
  set-current-plot-pen "mf"
  while [x <= 1.001] [
    plotxy x mf(x)
    set x x + 0.01
  ]
  set-current-plot-pen "diag"
  plotxy 0 0
  plotxy 1 1
  reset-ticks
  set-current-plot-pen "green"
  let xx 0.5
  let yy 0
  plotxy xx 0
  repeat 200 [
    set yy f(xx)
    plotxy xx yy
    plotxy yy yy
    set xx yy
  ]
end

to-report mf [x] ; mean-field map
  let y (item 0 tau) * (1 - x) ^ R ; first element, also get rids of 0 ^ 0 = 1
  let i 1
  while [i <= R] [
    set y y +  (choose R i) * (item i tau) * (x ^ i) * ((1 - x) ^ (R - i))
    set i i + 1
  ]
  report y
end

to-report f [x]
  report a * x * (1 - x)
end
```

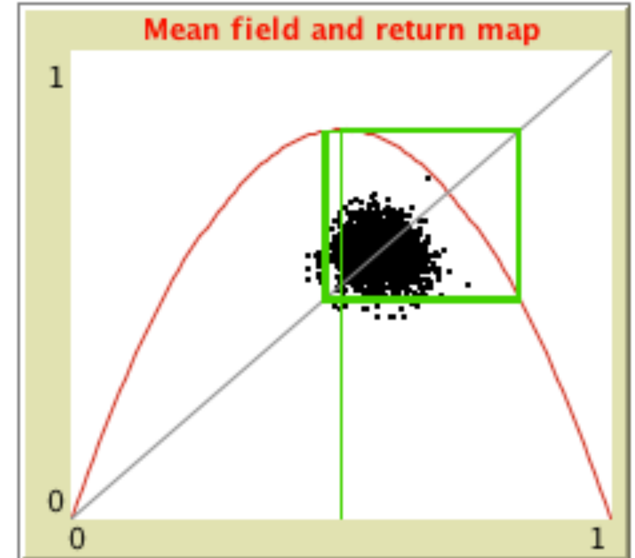
Mean-field vs real simulation

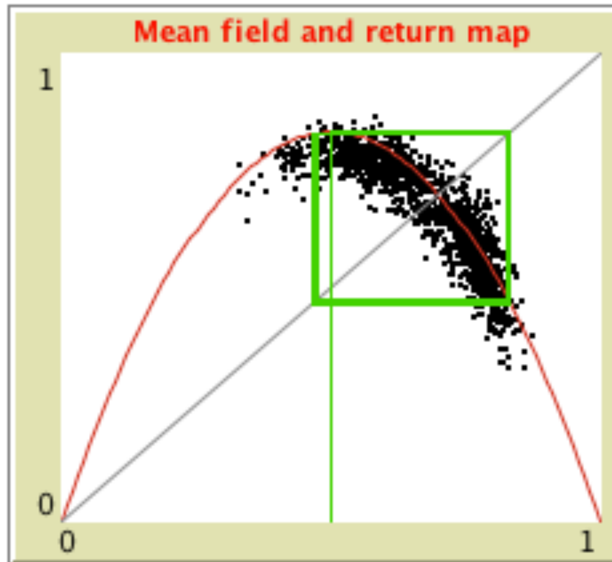
terza parte

```
to-report choose [n k] ; binomial coefficients
  ifelse k = 0 [
    report 1
  ][
    report choose (n - 1) (k - 1) * n / k
  ]
end

to go
  tick
  if shuffle? [
    repeat world-width [
      let ii random world-width
      let jj random world-width
      let tmp [state] of patch ii (- (ticks mod world-height) + 1)
      ask patch ii (- (ticks mod world-height)) [
        set state [state] of patch jj (- (ticks mod world-height) + 1)
      ]
      ask patch jj (- (ticks mod world-height) + 1) [
        set state tmp
      ]
    ]
  ]
  ask patches with [ pycor = (- (ticks mod world-height))][
    let T reduce + map [i -> [state] of patch-at i 1] neighs
    let prob item T tau
    set state ifelse-value (random-float 1 < prob) [1] [0]
    color-patch
  ]
  let dens (sum [state] of patches with [pycor = (- (ticks mod world-height))]) / world-width
  set-current-plot-pen "dens"
  plotxy old-dens dens
  set old-dens dens
end

;; color the patch based on whether on? is true or false
to color-patch  ;; patch procedure
  set pcolor ifelse-value (state = 1) [green][black]
end
```

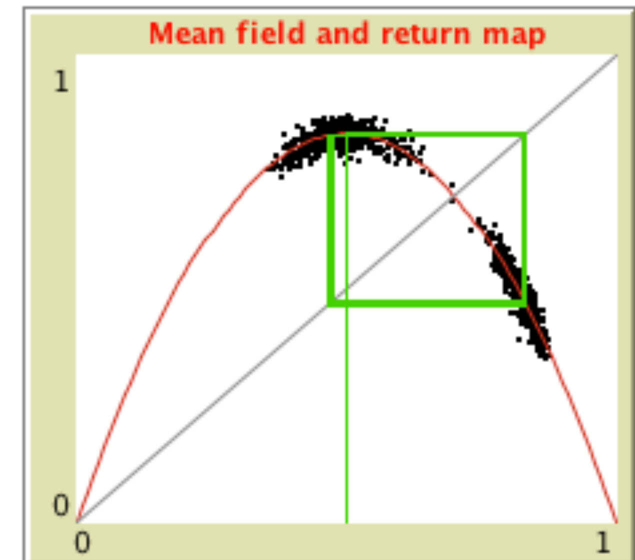
Mean-field vs real simulation

Quello che si nota è che senza nulla il campo medio è molto diverso dalla simulazione





Con il rimescolamento si comincia ad avere qualcosa di simile al campo medio

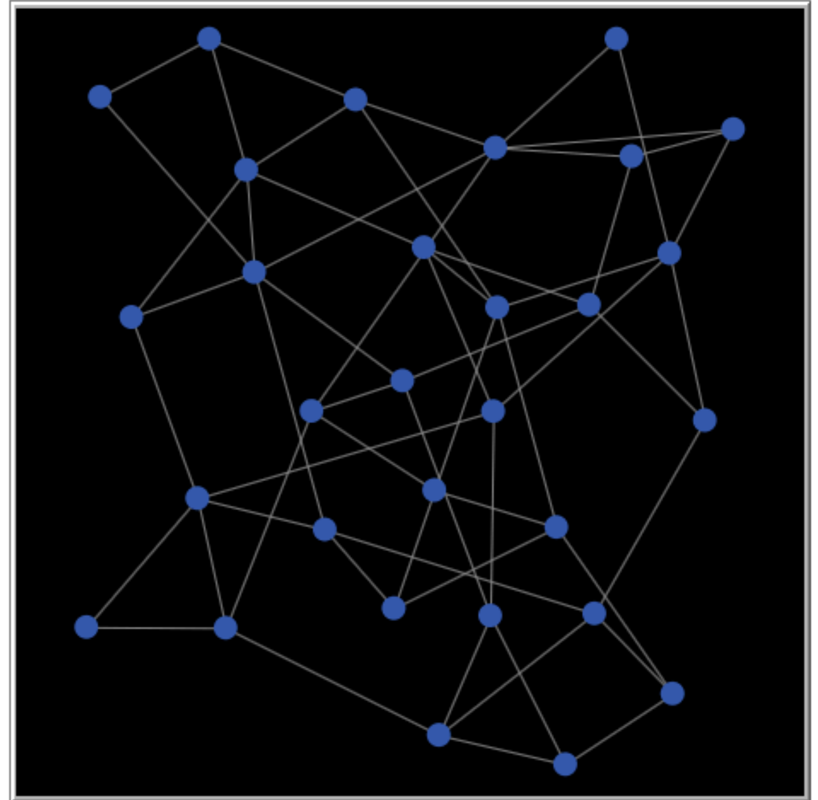


se tutti i link sono a lungo raggio si ha un buon accordo

Links

max-num-links 1

setup



I link sono utili quando vogliamo disegnare o lavorare con dei grafi. Per prima cosa quindi generiamo delle turtle in maniera casuale e chiediamo loro di stabilire dei link con le altre,

e poi usiamo la caratteristica che i link possono comportarsi come delle molle per avere un layout piacevole (graph.nlogo).

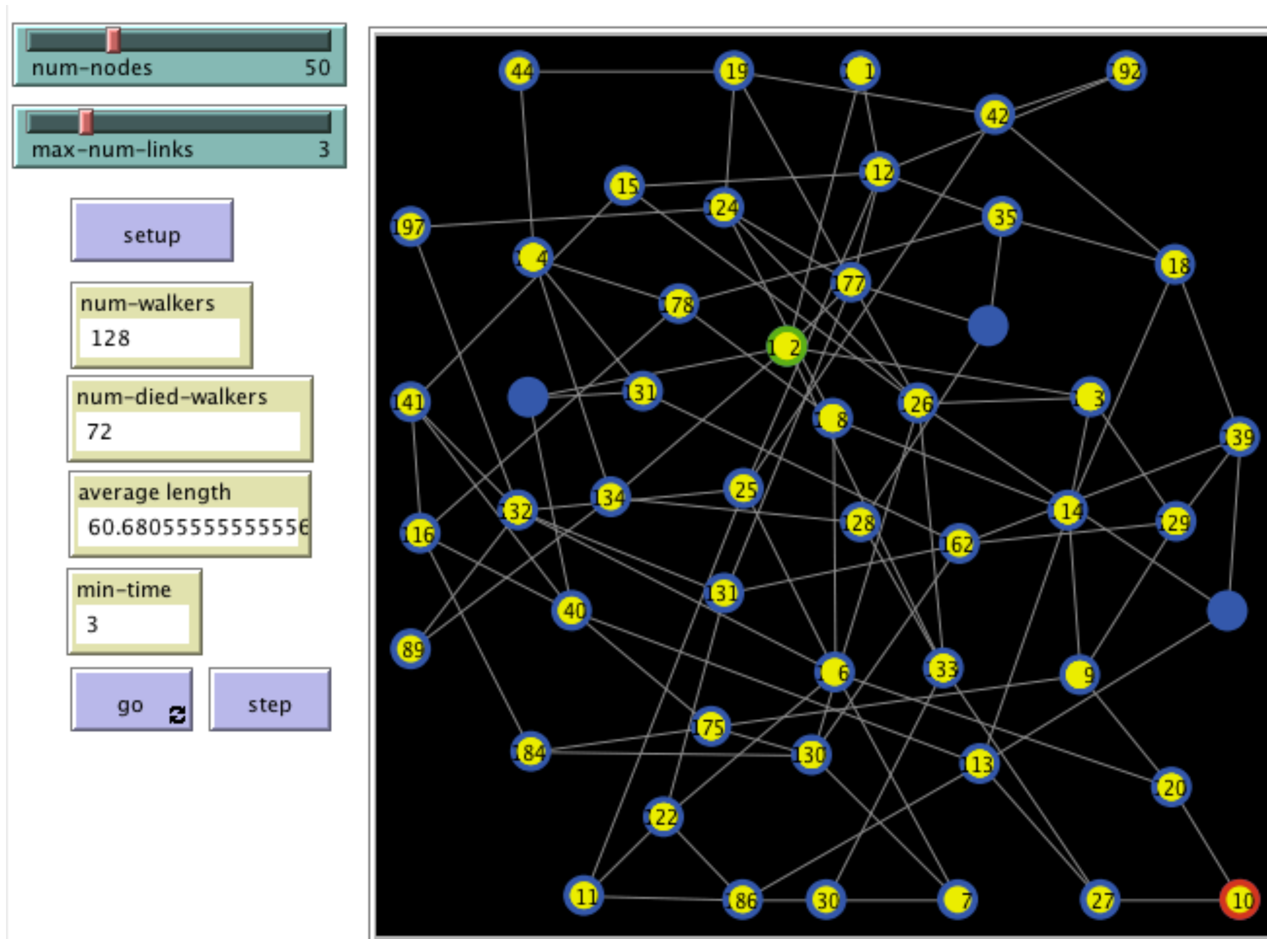
Links

```
breed [nodes node]

to setup
  clear-all
  set-default-shape nodes "circle"
  ;; create some random network
  create-nodes 30 [ set color blue ]
  ask nodes [
    create-links-with n-of (random max-num-links + 2) other nodes
  ]
  ;; lay it out so links are not overlapping
  repeat 500 [ layout ]
  ;; leave space around the edges
  ask nodes [ setxy 0.95 * xcor 0.95 * ycor ]
  ;; put some "walker" turtles on the network
  reset-ticks
end

to layout
  layout-spring nodes links 0.5 2 40
end
```

Notare che layout è chiamato un certo numero di volte (500) per essere sicuro che il grafo abbia trovato una configurazione stabile.

Links

Adesso inseriamo una sorgente, un pozzo e dei walker che camminano sul grafo (graph-walkers.nlogo)



Links Il codice è un po’ più lungo

```
breed [nodes node]
breed [sources source]
breed [wells well]
breed [walkers walker]
globals [num-walkers num-died-walkers sum-time-died-walkers min-time]

walkers-own [counter location] ; location is one of nodes

to setup
  clear-all
  set-default-shape nodes "circle"
  set-default-shape sources "circle"
  set-default-shape wells "circle"
  set-default-shape walkers "circle"
  ;; create some random network
  create-nodes num-nodes [
    set color blue
    set size 1.5
  ]
  ask nodes [
    create-links-with n-of (random (max-num-links - 2) + 2) other nodes
  ]
  ;; lay it out so links are not overlapping
  repeat 500 [ layout ]
  ;; leave space around the edges
  ask nodes [ setxy 0.95 * xcor 0.95 * ycor ]
  ask one-of nodes [
    hatch-sources 1 [
      set color green
      set size 1.5
    ]
  ]
  ask one-of nodes [
    hatch-wells 1 [
      set color red
      set size 1.5
    ]
  ]
  set min-time 100000
  reset-ticks
end
```

```
to go
 no-display
 ask sources [
   let new-location one-of nodes-here
   hatch-walkers 1 [
      set color yellow
      set size 1
      set location new-location
    ]
    set num-walkers num-walkers + 1
  ]
  ask wells [
    ask walkers-here [
      set num-died-walkers num-died-walkers + 1
      set sum-time-died-walkers
          sum-time-died-walkers + counter
      set num-walkers num-walkers - 1
      if counter < min-time [
        set min-time counter
      ]
      die
    ]
  ]
  ask walkers [
    let new-location one-of [link-neighbors]
       of location
    face new-location
    move-to new-location
    set location new-location
    set counter counter + 1
    set label counter
    set label-color black
  ]
  display
  tick
end
```