Concepte de Programare NetLogo

1. Agenţi şi Variabile

2. Interfaţă şi matematică

3. Desenare şi producţie

4. Sarcini şi Sintaxă

1. 1. Agenţi

Lumea NetLogo este compusă din agenţi. Agenţii sunt elemente care pot respecta instrucţiuni.

În NetLogo, există patru tipuri de agenţi: **turtles**, **patch-uri**, **link-uri** şi **observatorul**.

Turtles sunt agenţii care se pot deplasa în **lume**. Lumea este bidimensională şi este împărţită într-o reţea de patch-uri. Fiecare patch este o adevărată bucată de "teren" pe care turtles se pot deplasa. Link-urile sunt agenţii care leagă două turtles. Observatorul nu are o locaţie – ne putem imagina că acesta examinează lumea de turtles şi patch-uri.

Observatorul nu observă în mod pasiv – el dă instrucţiuni celorlalţi agenţi.

La iniţializarea NetLogo, nu există turtles. Observatorul poate crea turtles noi. De asemenea, şi patch-urile pot să creeze noi turtles. (Patch-urile nu se pot deplasa, dar în alte privinţe sunt la fel de "vii" ca şi turtles.)

Patch-urile au coordinate. Patch-ul de coordonate (0, 0) este denumit *origine* şi coordonatele celorlalte patch-uri sunt distanţele pe verticală şi orizontală faţă de acesta. Numim coordonatele unui patch `pxcor` şi `pycor`. La fel ca în planul standard de coordinate matematice, `pxcor` creşte pe măsură ce te deplasezi spre dreapta iar `pycor` creşte pe măsură ce te deplasezi în sus.

Numărul total de patch-uri este stabilit de setările `min-pxcor`, `max-pxcor`, `min-pycor`, şi `max-pycor`. La iniţializarea NetLogo, `min-pxcor`, `max-pxcor`, `min-pycor`, şi `max-pycor` sunt 16, 16, -16 şi, respectiv, 16. Aceasta înseamnă că `pxcor` şi `pycor` pot varia ambele între -16 şi 16, adică sunt de 33 ori 33, sau 1089 de patch-uri în total. (Numărul de patch-uri se poate modifica din Setări.)

Turtles au coordonatele `xcor` şi `ycor`. Coordonatele unui patch sunt întotdeauna numere întregi, dar coordonatele unui turtle pot avea zecimale. Acest lucru înseamnă că un turtle poate fi poziţionat în orice punct în interiorul patch-ului său; nu trebuie să fie în central patch-ului.

Link-urile nu au coordonate. Fiecare link are două capete, la fiecare capăt fiind câte un turtle. Dacă oricare turtle moare, moare şi link-ul. Un link este reprezentat vizual ca o linie ce leagă două turtles.

1.2. Proceduri

În NetLogo, **comenzile** şi **raportorii** indică agenţilor ce este de făcut. O comandă este o acţiune pe care un agent trebuie s-o ducă la îndeplinire, rezultând un oarecare efect. Un raportor reprezintă instrucţiuni pentru calcularea unei valori, pe care ulterior agentul o "raportează" oricui ar întreba.

De obicei, numele unei comenzi începe cu un verb, cum ar fi “create”, “die”, “inspect”, sau “clear”. Majoritatea numelor raportorilor sunt substantive sau locuţiuni substantivale.

Comenzile şi raportorii construiţi în NetLogo sunt numiţi **primitive**. Dicţionarul NetLogo are lista completă a comenzilor şi raportorilor încorporaţi.

Comenzile şi raportorii definiţi de utilizator sunt numiţi **proceduri**. Fiecare procedură are un nume, precedat de cuvântul-cheie `to` sau `to-report`, în funcţie de faptul că e o procedură de comandă sau una de raportor. Cuvântul-cheie `end` arată finalul comenzilor din procedură. Odată ce defineşti o procedură, poţi să o utilizezi şi în altă parte în program.

Multe comenzi şi raportori iau intrări -- valori pe care comanda sau raportorul le utilizează pentru îndeplinirea acţiunilor lor sau pentru calculul rezultatelor acestora.

Iată două exemple de proceduri de comandă:

```
to setup clear-all
       create-turtles 10
       reset-ticks
end
  to go ask turtles
[
              fd 1                ;; forward 1 step
              rt random 10  ;; turn right
              lt random 10  ;; turn left
       ] tick
end
```

Se remarcă utilizarea punct-virgulelor pentru adăugarea de “comentarii” programului. Comentariile pot face codul mai uşor de citit şi de înţeles, dar nu-i afectează comportamentul.

În acest program,

- `setup` şi `go` sunt comenzi definite de utilizator.
- `clear-all`, `create-turtles`, `reset-ticks`, `ask`, `lt` (“întoarcere la stânga”), `rt` (“întoarcere la dreapta”) şi `tick`, sunt toate comenzi primitive.
- `random` şi `turtles` sunt raportori primitive. `random` ia un număr unic ca pe o intrare şi raportează un întreg aleator mai mic decât intrarea (în acest caz, între 0 şi 9). `turtles` raportează

un set de agenţi constând din toţi turtles. (Setările agenţilor vor fi explicate ulterior.)

`setup` şi `go` pot fi denumite de alte proceduri, sau de butoane, sau prin Centrul de Comandă.

Multe modele NetLogo au un buton de o singură dată care apelează o procedură denumită `setup` şi un buton de tot timpul care apelează o procedură denumită `go`.

În Netlogo, se poate specifica care agenţi – turtles, patch-uri sau link-uri – trebuie să ruleze fiecare comandă. Dacă nu se specifică, atunci codul este condus de către observator. În codul de mai jos, observatorul utilizează `ask` pentru a face setul de toate turtles să ruleze comenzile dintre

parantezele pătrate. `clear-all` şi `create-turtles` pot fi rulate doar de către observator. Pe de altă parte, `fd` poate fi rulat doar de turtles. Alte comenzi şi raportori, cum ar fi `set` şi `ticks`, pot fi rulate de diferite tipuri
de agenţi.
Iată câteva caracteristici mai avansate, de care se poate profita pentru definirea propriilor proceduri.

Proceduri cu intrări

Procedurile pot prelua intrări, la fel ca și multe alte primitive. Pentru a crea o procedură care să accepte intrări, se pune numele acestora între paranteze pătrate după numele procedurii. De exemplu:

```
to draw-polygon [num-sides len] ;; turtle procedure
       pen-down repeat num-sides [ fd len
              rt 360 / num-sides
       ] end
```

În altă parte a programului, s-ar putea folosi procedura care cere fiecărui turtle să deseneze un octogon cu latura egală cu o lungime specificată:

```
ask turtles [ draw-polygon 8 who ]
```

Proceduri cu raportori

Aşa cum se pot defini propriile comenzi, se pot defini şi proprii raportori. Trebuie realizate două lucruri speciale. Mai întâi, se utilizează `to-report` în loc de `to` pentru a porni procedura. Apoi, în interiorul procedurii, se utilizează `report` pentru a raporta valoarea ce se doreşte a fi returnată:

```
to-report absolute-value [number] ifelse
       number >= 0
              [ report number ]
              [ report (- number) ]
end
```

1.3. Variabile

Variabilele agenţilor

Variabilele agenţilor sunt locurile în care se stochează valori (cum ar fi numerele) într-un agent. Variabila unui agent poate fi o variabilă globală, o variabilă a unui turtle, o variabilă a unui patch, sau o variabilă a unui link.

Dacă o variabilă este globală, există o singură valoare pentru ea, şi oricare agent o poate accesa. Se poate spune că variavilele globale aparţin observatorului.

Variabilele unor turtles, patch-uri sau linkuri sunt diferite. Fiecare turtle are ***propria*** sa valoare pentru fiecare variabilă. La fel este şi în cazul patch-urilor sau link-urilo.

Unele variabile sunt construite în NetLogo. De exemplu, fiecare turtle sau link are o variabilă `color`, şi toate patch-urile au o variabilă `pcolor`. (Variabila patch-ului începe cu "p" pentru a nu fi confundată cu variabila unui turtle, de vreme ce turtles au acces direct la variabilele patch-ului.) Dacă se stabileşte varibila, turtle sau patch-ul îşi vor schimba culoarea.

Alte variabile de turtle încorporate includ `xcor`, `ycor`,şi `heading`. Alte variabile de patch încorporate includ `pxcor` şi `pycor`. (Lista completă aici.)

Se pot defini, de asemenea, propriile variabile. Se poate construi o variabilă globală prin adăugarea unui comutator, cursor, selector, sau a unei casete de intrări la prpriul model, sau prin utilizarea cuvântului-cheie `globals` la începutul codului, ca de exemplu:

```
globals [score]
```

Se pot defini, de asemenea, noi variabile ale turtles, patch-urilor sau link-urilor folosind cuvintele cheie `turtles-own`, `patches-own` sau `links-own`, ca de exemplu:

```
turtles-own [energy speed] patches-own
[friction]
links-own [strength]
```

Aceste variabile pot fi folosite în mod liber în modelul propriu. Utilizați comanda set pentru a le configura. (Orice variabilă ce rămâne nesetată are valoarea de pornire zero.)

Variabilele globale pot fi citite și stabilite în orice moment de către oricare agent. De asemenea, un turtle poate citi și configura variabilele de patch ale patch-ului pe care se află. De exemplu, acest cod:

```
ask turtles [ set pcolor red ]
```

Determină fiecare turtle să facă patch-ul pe care stă roşu. (Datorită faptului că variabilele specifice patch-urilor sunt separate de cele specifice turtles în acest fel, nu se poate ca o variabilă turtle şi o variabilă de patch să aibă acelaşi nume.)

În alte situaţii în care se doreşte ca un agent să citească variabila altui agent, se poate utiliza `of`. Exemplu:

```
show [color] of turtle 5
;; prints current color of turtle with who number 5
```

De asemenea, se poate folosi `of` cu o expresie mult mai complicată decât doar un nume de variabilă, ca de exemplu:

```
show [xcor + ycor] of turtle 5
;; prints the sum of the x and y coordinates of
;; turtle with who number 5
```

Variabilele locale

O variabilă locală este definită şi utilizată doar în contextul unei proceduri speciale sau ca parte a unei proceduri. Pentru a crea o variabilă locală, se foloseşte comanda `let`. Dacă `let` este utilizată la începutul procedurii, această comandă va exista pe tot parcursul procedurii. Dacă va fi folosită în interiorul unor paranteze pătrate, de exemplu în interiorul unui “ask”, atunci ea va funcţiona doar între acele paranteze.

```
to swap-colors [turtle1 turtle2] let
       temp [color] of turtle1
       ask turtle1 [ set color [color] of turtle2 ] ask
       turtle2 [ set color temp ]
end
```

1.4. Contor de ticks

În multe modele NetLogo, timpul trece prin etape distincte, numite “ticks”. NetLogo include un numărător de ticks, pentru a putea urmări câte etape au trecut.

Valoarea curentă a contorului este arătată deasupra secţiunii view. (Se poate folosi butonul Setări pentru a ascunde contorul, sau pentru a schimba cuvântul ticks cu altul).

În cadrul codului, pentru a prelua valoarea curentă a contorului, se foloseşte raportorul `ticks`. Comanda `tick` măreşte contorul de ticks cu 1. Comanda `clear-all` resetează contorul împreună cu restul.

În cazul în care contorul este resetat, ar fi o eroare să se încerce citirea sau modificarea lui. Se foloseşte comanda `reset-ticks` atunci când modelul este definitivat, pentru a porni contorul.

Dacă modelul respectiv este setat să utilizeze actualizările pe baza de ticks, atunci, de obicei, comanda `tick` va actualiza de asemenea şi secţiunea view. A se vedea şi secţiunea ulterioară, Vizualizare Actualizări.

Când se foloseste ticks

Se foloseşte `reset-ticks` la sfârşitul procedurii de setare. Se foloseşte `tick` la finalul procedurii de iniţializare.

```
to setup clear-all
       create-turtles 10
       reset-ticks
end
 to go ask turtles [ fd 1
] tick
end
```

Ticks fracţionare

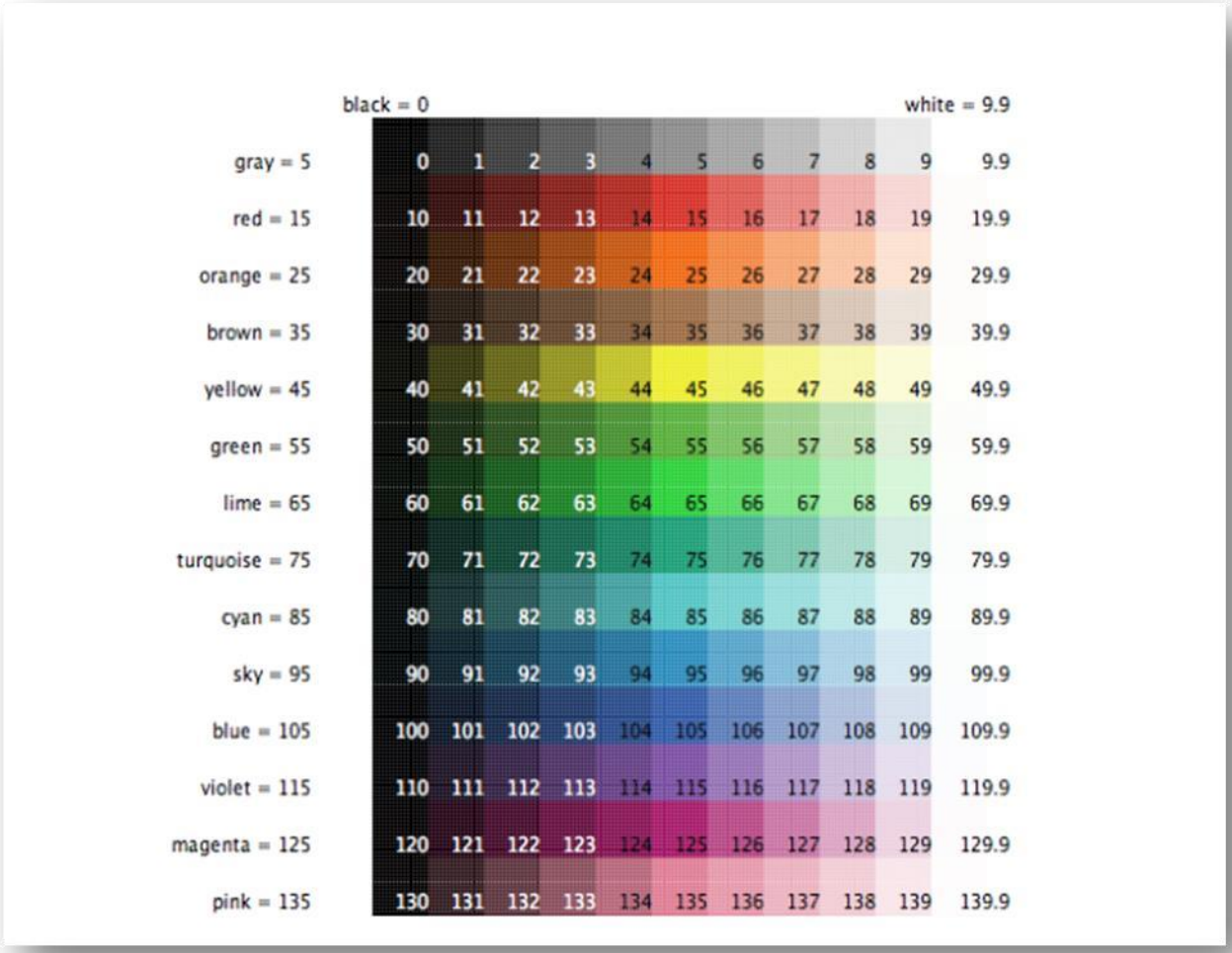
În majoritatea modelelor, contorul de ticks porneşte de la 0 şi creşte cu 1 la fiecare pas, din număr întreg în alt număr întreg. Dar, de asemenea, este posibil ca un contor de ticks să ia valori cu virgulă.

Pentru a creşte contorul cu o valoare fracţionară, se utilizează comanda tick-advance. Această comandă primeşte o intrare numerică ce specifică în ce măsură avansează contorul.

O utilizare tipică a bifărilor fracţionare este dată de aproximarea mişcărilor continue sau curbe. De exemplu, pot fi urmărite modelele GasLab din Models Library. Aceste modele calculează momentul exact la care un eveniment viitor se va petrece, şi avansează contorul în exact acel moment.

1.5. Culori

NetLogo reprezintă culorile în diferite moduri. O culoare poate fi un număr ce variază în intervalul 0140, cu excepţia lui 140. Mai jos este un grafic ce arată gama de culori ce poate fi utilizată în NetLogo.



Din grafic se obervă faptul că:

- Unele culori au nume. (Se pot folosi aceste nume în cadrul codului.)
- Fiecare culoare cu nume, exceptând negrul şi albul, are un număr care se termină cu 5.
- De fiecare parte a fiecărei culori cu nume sunt nuanţele mai închise şi mai deschise ale culorii respective.
- 0 reprezintă negrul pur. 9.9 este albul pur.
- 10, 20 şi aşa mai departe sunt valori atât de închise, încât sunt foarte aproape de negru.

- 19.9, 29.9 şi aşa mai departe sunt valori atât de deschise, încât sunt foarte aproape de alb.

Exemplu de cod: Graficul de culori a fost creat în NetLogo cu modelul Color Chart Example.

Dacă se foloseşte un număr în afara intervalului 0-140, atunci NetLogo adaugă sau scade în mod repetat 140 până când se ajunge în intervalul 0-140. De exemplu, 25 reprezintă portocaliul, deci şi 165, 305, 445 şi aşa mai departe sunt tot valori pentru portocaliu, la fel ca şi -115, -255, -395, etc. Acest calcul este făcut automat când se setează variabila turtle `color` sau variabila patch `pcolor`. Dacă acest calcul trebuie efectuat într-un alt context, se utilizează primitiva `wrap-color`.

Dacă se dorește o culoare ce nu există în grafic, se pot specifica valori intermediare. De exemplu, 26.5 este o nuanţă de portocaliu aflată la jumătatea spectrului dintre 26 şi 27.
Aceasta nu înseamnă că se poate crea orice culoare în NetLogo; spectrul de culori NetLogo este doar un set din toate culorile posibile. Acesta conţine doar un set fix de nuanţe distincte (câte o nunanţă pentru fiecare rând al graficului). Pornind de la una dintre aceste nuanţe, se poate fie scădea luminozitatea (nuanţă mai întunecată), fie reduce saturaţia (nuanţă mai luminoasă), dar nu se pot scădea albele simultan. De asemenea, doar prima cifră de după virgulă este semnificativă. Astfel, valorile culorii sunt rotunjite până la următorul 0.1. De exemplu, nu există o diferenţă vizibilă între 26.5, 26.52 și 26.58.

Primitive de culoare

Există câteva primitive utili pentru gestiunea culorilor.

Am menţionat deja primitiva `wrap-color`.

Primitiva `scale-color` este utilă pentru conversia datelor numerice în culori.

`shade-of?` indică dacă două culori sunt ambele nuanţe ale aceleiaşi culori de bază. De exemplu, `shade-of? orange 27` este valid, deoarece 27 este o nuanţă mai deschisă de portocaliu.

Exemplu de cod: Scale-color Example arată raportorul scării de culori.

Culorile RGB şi RGBA

NetLogo prezintă, de asemenea, şi culori în format RGB (roşu/verde/albastru) şi RGBA (roşu/verde/albastru/alpha). Atunci când se utilizează formatul RGB, întreaga gamă de culori este disponibilă pentru utilizator. Culorile RGBA permit toate culorile RGB, dar se poate specifica și transparenţa culorii. Listele RGB şi RGBA sunt formate din trei sau patru numere întregi între 0 şi 255; dacă un număr este în afara intervalului, se scade din el 255 în mod repetat până se ajunge în intervalul respectiv. Se pot seta orice variabilă de culoare în NetLogo (~~`color`~~ pentru turtles şi link-

uri şi pcolor pentru patch-uri) folosind o listă RGB. De exemplu, se poate seta culoarea patch-ului 0 0 la roşu pur folosind următorul cod:

```
set pcolor [255 0 0]
```

Toate turtles, link-urile şi etichetele pot conţine RGBA ca variabile de culoare; cu toate acestea, patch-urile nu pot avea pcolors RGBA. Se poate seta culoarea unui turtle la roșu pur semitransparent cu următorul cod:

```
set color [255 0 0 125]
```

Se poate converti de la o culoare NetLogo la o culoare în formatele RGB sau HSB (nuanţă / saturaţie / luminozitate), folosind extract-rgb sau extract-hsb. Se poate folosi rgb pentru a genera liste RGB sau hsb pentru a converti de la o culoare HSB la RGB.

Deoarece lipsesc multe culori din spectrul de culori NetLogo, approximate-hsb sau approximate-rgb nu pot da de multe ori culoarea cerută, dar vor încerca să ajungă cât mai aproape posibil.

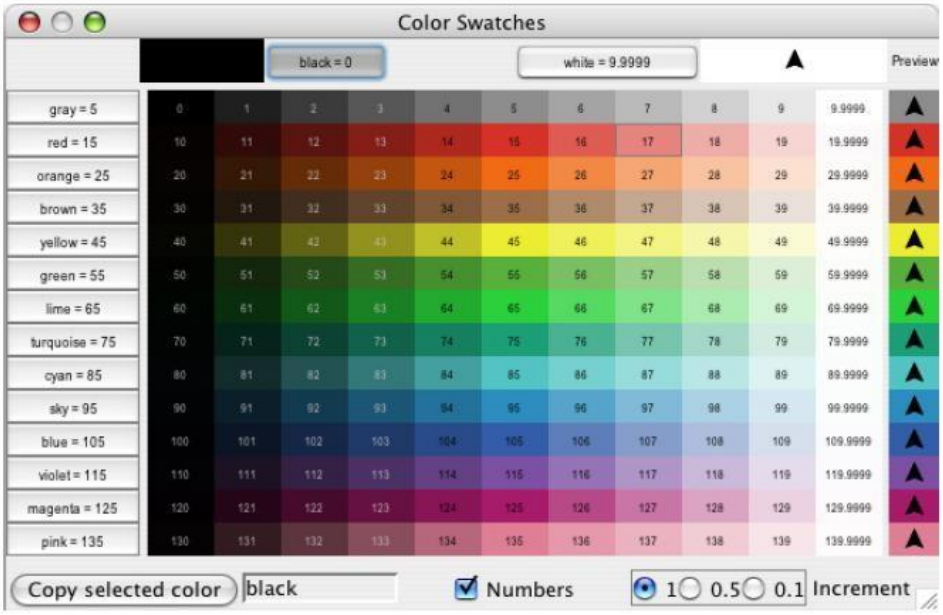
Exemplu: se poate schimba culoarea deja existentă în NetLogo a unui turtle la o variantă pe jumătate transparentă a acelei culori folosind:

```
set color lput 125 extract-rgb color
```

Exemple de cod: HSB şi RGB Example (permit încercarea sistemelor de culori HSB şi RGB), Transparency Example.

Fereastra de dialog a specimenelor de culoare

Aceasta permite utilizatorului să încerce şi să aleagă culori. Se deschide accesând Color Swatches din Tools Menu.



Când se face click pe un specimen de culoare (sau pe un buton de culoare), acea culoare va fi afișată alături de alte culori. În stânga jos, codul pentru culoarea selectată în prezent este afişat (de exemplu, `roşu + 2`), astfel că el poate fi copiat şi inserat în cod. În dreapta jos, sunt trei opţiuni de creştere, 1, 0.5 şi 0.1. Aceste numere indică diferenţa dintre două specimen vecine. Când creşterea este de 1, există 10 nuanţe diferite pe fiecare rând; când sporul este de 0.1, există 100 de nuanţe în fiecare rând. 0.5 este o setare intermediară.

1.6. Cereri

NetLogo utilizează comanda `ask` pentru a da comenzi pentru turtles, patch-uri şi link-uri. Toată secvenţa de cod ce va fi rulată de turtles ***trebuie*** să fie localizată într-un “context” turtle. Un context turtle poate fi stabilit în oricare din următoarele trei moduri:

- Într-un buton, prin alegerea “Turtles” din meniul pop-up. Orice cod s-a pus în buton va fi rulat de toţi turtles.
- În Centrul de comandă, prin alegerea “Turtles” din meniul pop-up. Orice comandă introdusă va fi executată de toţi turtles.
- Prin utilizarea `ask turtles`, `hatch`, sau alte comenzi ce stabilesc un context turtle.

Acelaşi lucru este valabil şi pentru patch-uri, link-uri sau pentru observator, cu excepţia faptului că nu se poate da comanda `ask` observatorului. Orice cod ce nu este inclus într-un `ask` este implicit un cod observator.

Iată un exemplu de utilizare `ask` într-o procedură NetLogo:

```
to setup clear-all
```

```
      create-turtles 100     ;; create 100 turtles with random headings ask
      turtles
            [ set color red ;; turn them red fd
            50 ]      ;; spread them around
      ask patches
            [ if pxcor > 0      ;; patches on the right side
                  [ set pcolor green ] ] ;; of the view turn green
      reset-ticks
end
```

Modelele din Models Library sunt pline de alte exemple. Un loc bun de pornire este în secţiunea Code Examples.

De obicei, observatorul foloseşte `ask` pentru a solicita execuția de comenzi de către toate turtles, patch-uri şi link-uri. De asemenea, se poate folosi `ask` şi pentru a cere unui singur turtle, patch sau link să execute o comandă. Raportorii `turtle`, `patch`, `link` şi `patch-at` sunt deosebit de utili în acest sens. De exemplu:

```
to setup clear-
      all
      crt 3                              ;; make 3 turtles
      ask turtle 0                         ;; tell the first
                                                  one...
            [ fd 1 ]                     ;; ...to go forward
      ask turtle 1                         ;; tell the second
                                                   one...
            [ set color green ]       ;; ...to become green
       ask turtle 2                      ;; tell the third one...
            [ rt 90 ]                    ;; ...to turn right
       ask patch 2 -2                    ;; ask the patch at (2,-2)
            [ set pcolor blue ]          ;; ...to become blue
       ask turtle 0                      ;; ask the first turtle
            [ ask patch-at 1 0           ;; ...to ask patch to the east
                  [ set pcolor red ] ]   ;; ...to become red
       ask turtle 0                      ;; tell the first turtle...
        [ create-link-with turtle 1 ]    ;; ...make a link with the
                                         second
       ask link 0 1                      ;; tell the link between turtle
                                         0 and 1
            [ set color blue ]           ;; ...to become blue
       reset-ticks
end
```

Oricare turtle creat are un număr. Primul turtle creat este numărul 0, următorul numărul 1, şi aşa mai departe.

Raportorul `turtle` ia un număr ca intrare, şi raportează turtle-ul cu acel număr.
Raportorul `patch` primeşte valori pentru pxcor şi pycor şi raportează patch-ul de la aceste coordonate. Primitiva `link` primeşte două intrări, numerele ale celor doi turtles pe care îi conectează. Raportorul `patch-at` primeşte ***plecări***: distanţe, în direcţiile x şi y, ***de la*** primul agent. În exemplul de mai sus, unui turtle i se cere să ia patch-ul est (şi nu patch-uri spre nord), de la sine.

De asemenea, se poate selecta un grup de turtles, de patch-uri sau de link-uri şi li se solicite să execute ceva. Acest lucru presupune folosirea **seturilor de agenţi**. Următoarea secţiune îi explică în detaliu.

Când i se cere unui set de agenţi să execute mai mult de o comandă, fiecare agent trebuie să termine înainte de a începe următorul. Un agent execută toate comenzile, apoi agentul următoar le rulează pe toate, şi aşa mai departe. De exemplu, dacă se scrie:

```
ask turtles [ fd 1 set color
       red ]
```

primul turtle se mişcă şi devine roşu, apoi alt turtle se mişcă şi devine roşu, şi aşa mai departe.

Dar dacă se scrie astfel:

```
ask turtles [ fd 1 ] ask
turtles [ set color red ]
```

atunci mai întâi toţi turtles se mişcă, apoi toţi iau culoarea roşu.

(O altă formă a comenzii `ask`, cu o altă regulă de ordonare, este de asemenea disponibilă. A se consulta secţiunea Ask-Concurrent de mai jos.)

1.7. Seturi de agenţi

Un set de agenţi poate conţine o submulțime de turtles, patch-uri, sau link-uri (doar agenți de un singur tip la un moment dat).

Un set de agenţi nu este sortat în vreo ordine specială. La fiecare utilizare, el va fi într-o ordine ***aleatoare***. Acest lucru ajută ca modelul să nu trateze în mod diferenţiat unii turtles, patch-uri şi linkuri (cu excepţia cazului în care se doreşte acest lucru). Deoarece ordinea este aleatoare de fiecare dată, nici un agent nu primeşte întotdeauna permisiunea de a porni primul.

Au fost prezentate primitivele turtles, care raportează setul de agenţi tuturor `turtles`, primitivele `patches`, ce raportează setul de agenţi tuturor patch-urilor, şi primitivele `links`, care raporteză setul de agenţi tuturor link-urilor.

Se pot construi seturi de agenţi care să cuprindă numai ***anumiţi*** turtles, ***anumite*** patch-uri sau ***anumite*** link-uri. De exemplu, toţi turtles roşii, sau toate patch-urile care au pxcor uniform divizibil

cu cinci, sau turtles din primul cadran ce sunt pe un patch verde, sau link-urile conectate la turtle 0. Aceste seturi de agenţi pot fi folosite apoi prin `ask` sau prin alţi diferiţi raportori ce le iau ca intrări.

O modalitate ar fi de a utiliza `turtles-here` sau `turtles-at`, pentru a crea un set de agenţi care să conţină doar turtles care să fie pe patch-ul specificat, sau doar turtles de pe alt patch la anumite poziții x şi y. Există de asemenea ~~`turtles-on`~~, astfel se poată obține setul de turtles de pe un anumit patch sau de pe un anumit set de patch-uri, sau setul de turtle ce se află pe acelaşi patch ca şi un anumit turtle sau un set de turtles.

Iată alte câteva exemple legate de modul în care se pot crea seturi de agenţi:

```
;; all other turtles:
other turtles
;; all other turtles on this patch:
other turtles-here ;;
all red turtles:
turtles with [color = red] ;;
all red turtles on my patch
turtles-here with [color = red]
;; patches on right side of view
patches with [pxcor > 0]
;; all turtles less than 3 patches away turtles
in-radius 3
;; the four patches to the east, north, west, and south patches
at-points [[1 0] [0 1] [-1 0] [0 -1]]
;; shorthand for those four patches neighbors4
;; turtles in the first quadrant that are on a green patch
turtles with [(xcor > 0) and (ycor > 0) and (pcolor =
green)]
;; turtles standing on my neighboring four patches turtles-on
neighbors4
;; all the links connected to turtle 0
[my-links] of turtle 0
```

De remarcat utilizarea `other` pentru excluderea unui agent.

O dată ce setul de agenţi a fost creat, se pot efectua unele operații simple:

- Utilizarea `ask` pentru a cere agenţilor din setul de agenţi să execute comenzi
- Utilizarea `any?` pentru a afla dacă setul de agenţi este gol
- Utilizarea ~~`all?`~~ pentru a afla dacă fiecare agent dintr-un set de agenţi satisface o condiţie
- Utilizarea `count` pentru a afla numărul de agenți din acel set.

Iată şi unele lucruri mai complexe ce se pot efectua:

- Alegerea la întâmplare a unui agent folosind `one-of`. De exemplu, un turtle ales aleator poate fi făcut verde:

```
ask one-of turtles [ set color green ]
```

Sau să se ceară unui patch ales aleator să genereze un nou turtle:

```
ask one-of patches [ sprout 1 ]
```

- Folosirea raportorilor `max-one-of` şi `min-one-of` pentru aflarea agentului care este maximul sau minimul de-a lungul unei anumite scale. De exemplu, pentru eliminarea celui mai mare turtle, se poate folosi:

```
ask max-one-of turtles [sum assets] [ die ]
```

- Realizarea unei histograme a setului de agenţi prin utilizarea comenzii `histogram` (în combinaţie cu `of`).
- Folosirea `of` pentru a realiza o listă de valori, câte una pentru fiecare agent din setul de agenţi. Apoi folosirea unui primitive din lista NetLogo pentru a schimba ceva în acea listă. (A se consulta secţiunea "Liste" de mai jos.) De exemplu, pentru a afla cât de mari sunt turtles în medie, se poate folosi:

```
show mean [sum assets] of turtles
```

- Folosirea raportorilor `turtle-set`, `patch-set` şi `link-set` pentru a crea noi seturi de agenţi reunind agenţii dintr-o varietate de surse posibile.
- Folosirea raportorilor `no-turtles`, `no patches` şi `no-links` pentru crearea de seturi de agenţi goale.
- Verificarea egalităţii dintre două seturi de agenţi, folosind = sau !=.
- Folosirea `member?` Pentru a afla dacă un anumit agent este membru al unui set de agenţi.

Acestea sunt doar lucruri minore. A se consulta Models Library pentru multe alte exemple, şi Dicţionarul NetLogo pentru mai multe informaţii despre toţi primitivii seturilor de agenţi.

Mai multe exemple de utilizare a seturilor de agenţi sunt furnizate în rubricile individuale pentru aceste primitive din Dicţionarul NetLogo.

Seturi de agenţi speciale

Seturile de agenţi `turtles` şi `links` au un comportament special deoarece ele deţin întotdeauna setul tuturor turtles link-urilor. Prin urmare, aceste seturi de agenţi pot creşte.

Următoarea interacţiune demonstrează un comportament special al acestora. Să presupunem că tab-ul Code are `globals [g]`. Atunci:

```
observer> clear-all observer>
create-turtles 5 observer>
set g turtles observer> print
count g
5
observer> create-turtles 5 observer>
print count g
10
observer> set g turtle-set turtles observer>
print count g
10
observer> create-turtles 5 observer>
print count g
10
observer> print count turtles
15
```

Setul de agenţi turtles creşte când se nasc noi turtles, dar alte seturi de agenţi nu cresc. Dacă se scrie `turtle-set turtles`, se obţine un set de agenţi nou, ce conţine doar turtles care există şi în prezent. Turtles noi nu se vor adăuga după ce vor apărea.

Seturi de agenţi şi liste

Anterior, s-a specificat faptul că fiecare set de agenţi este sortat aleator. Dacă este necesar ca agenţii să execute ceva într-o ordine prestabilită, trebuie făcută o anumită listă a acestor agenţi. A se vedea secţiunea Liste de mai jos.

Exemplu de cod: Ask Ordering Example

1.8. Rase

NetLogo permite utilizatorului să definească diferite “rase” de turtles sau rase de link-uri. După definirea raselor, se poate trece mai departe şi face ca diferite rase să se comporte în mod diferit. De exemplu, poţi avea rase denumite `sheep` şi `wolves`, şi să faci ca wolves să încerce să mănânce sheep, sau poţi avea rase de link-uri denumite `streets` şi `sidewalks` în care traficul pietonal este direcţionat spre sidewalks iar cel rutier spre streets.

Rasele de turtles se pot defini folosind cuvântul-cheie *breed*, din capătul tab-ului Code, înaintea oricărei alte proceduri:

```
breed [wolves wolf]
breed [sheep a-sheep]
```

Referirea la un membru al rasei se face folosind forma singular, exact ca la un raportor `turtle`. Când sunt editaţi, membrii rasei vor fi etichetaţi cu numele singular.

Unele comenzi şi raportori primesc numele rasei în varianta plurală, cum ar fi `create-<breeds>`. Altele primesc numele singular al rasei, cum ar fi `<breeds>`.

Ordinea în care rasele sunt declarate este și ordinea în care ele sunt afișate. Așadar rasele definite mai târziu vor apărea deasupra celor definite mai devreme; în acest exemplu, `sheep` vor fi desenate peste `wolves`.

Când se defineşte o rasă `sheep`, se creează automat un set de agenţi pentru această rasă, astfel că toate proprietățile seturilor de agenți descrise mai sus sunt valabile și pentru `sheep`.

Alte primitive disponibile odată definită o rasă: `create-sheep`, `hatch-sheep`, `sprout-sheep`, `sheep-here`, `sheep-at`, `sheep-on`, şi `is-a-sheep?`.

De asemenea, se poate utiliza `sheep-own` pentru a defini noi variabile , doar pentru turtles ale acelei rase. (Se permite ca mai multe rase să deţină aceeaşi variabilă.)

Setul de agenţi al unei rase turtle este stocat în variabila turtle `breed`. În consecinţă, rasa unui turtle poate fi verificată astfel:

```
if breed = wolves [ ... ]
```

De asemenea, un turtle își poate schimba rasa. Un wolf nu trebuie să rămână wolf întreaga sa viaţă. Un wolf aleator poate fi schimbat într-un sheep astfel:

```
ask one-of wolves [ set breed sheep ]
```

Primitivul set-default-shape este util pentru asocierea diferitelor forme de turtle cu anumite rase. A se consulta secţiunea forme de mai jos.

Iată un exemplu rapid de utilizare a raselor:

```
breed [mice mouse]
breed [frogs frog]
mice-own [cheese] to
setup clear-all
create-mice 50
              [ set color white set cheese
                     random 10 ]
       create-frogs 50
              [ set color green ] reset-ticks
end
```

Exemplu de cod: Breeds and Shapes Example

Rasele link

Rasele link sunt foarte asemănătoare cu rasele turtle, existând totuși câteva diferenţe:

Când se declară o rasă link, trebuie să se specifice dacă este o rasă de link-uri direcționate sau nedirecționate, folosind cuvintele cheie directed-link-breed şi undirected-link-breed.

```
directed-link-breed [streets street]
undirected-link-breed [friendships friendship]
```

După ce s-a creat un link de rasă, nu se mai pot crea link-uri fără rasă sau invers. (Se poate totuşi să existe link-uri direcționate sau nedirecționate în aceeaşi lume, doar că nu în cadrul aceleaşi rase.)

Spre deosebire de rasele turtle, numele singular al rasei este necesar pentru rasele de link-uri, de vreme ce multe dintre comenzile sau raportorii link folosesc numele singular, ca de exemplu `<link-breed>-neighbor?`.

Primitive disponibile odată definită o rasă de link-uri direcționate:
`create-street-from` `create-streets-from` `create-street-to` `create-streets-to` `in-street-neighbor?` `in-street-neighbors` `in-street-from` `my-in-streets` `myout-streets` `out-street-neighbor?` `out-street-neighbors` `out-street-to`

Primitive disponibile odată definită o rasă de link-uri nedirecționate: `create-friendship-with` `create-friendships-with` `friendship-neighbor?` `friendship-neighbors` `friendship-with` `my-friendships`

Rasele de link-uri multiple pot declara aceeaşi variabilă `own`, dar o variabilă nu poate fi împărţită între o rasă turtle şi o rasă link.

La fel ca şi la rasele turtle, ordinea în care rasele link sunt declarate devine ordinea în care ele sunt reprezentate; astfel, `friendships` vor fi întotdeauna deasupra `streets` (în cazul în care dintr-un motiv sau altul aceste rase au făcut parte din acelaşi model). De asemenea, se poate folosi <linkbreed>-own pentru a declara variabilele fiecărei rase de link-uri separat.

Se poate schimba rasa unui link cu `set breed` (Cu toate acestea, nu se poate schimba un link cu rasă cu unul fără, pentru a împiedica existenţa link-urilor cu rasă şi fără rasă în aceeaşi lume.)

```
ask one-of friendships [ set breed streets ]
ask one-of friendships [ set breed links ] ;; produces a runtime error
```

`set-default-shape` poate fi de asemenea folosită şi pentru a asocia rasele link cu un anumit profil link.

Exemplu de cod: Link Breeds Example

2.1. Butoane

Butoanele din interfaţă oferă o modalitate uşoară de a controla modelul. De obicei, un model va avea cel puţin un buton “setup”, pentru a stabili starea iniţială a lumii, şi un buton “go” pentru a face modelul să ruleze continuu. Unele modele au butoane suplimentare care realizează alte acţiuni.

Un buton conţine cod NetLogo. Acel cod este rulat când se apasă butonul.

Un buton poate fi fie “buton Once”, fie “buton Forever”. Acest lucru poate fi controlat prin editarea butonului şi bifarea sau debifarea casetei “Forever”. Butoanele Forever continuă să-şi execute codul în mod repetat, până când fie se ajunge la comanda `stop`, fie utilizatorul apasă butonul din nou pentru a-l opri. Dacă butonul e oprit, codul nu se întrerupe, ci se așteaptă încheierea execuției codului său.

De obicei, un buton este etichetat cu codul pe care-l execută. De exemplu, un buton pe care scrie “go” conţine de obicei codul “go”, care înseamnă “execută procedura go”. (Procedurile sunt definite în tab-ul Code). De asemenea, se poate edita un buton şi introduce un “nume afişat” pentru buton, care e un text ce apare pe buton în loc de cod. Această funcţie ar putea fi utilizată dacă se crede că în realitate codul este prea confuz pentru utilizatori.

Când s-a asociat o secvență de cod unui buton, trebuie de asemenea specificaţi agenţii care vor executa acel cod. Se poate alege ca observatorul să ruleze codul, sau toţi turtles, toate patch-urile sau toate link-urile. (Dacă se intenţionează ca un cod să fie executat doar de anumiţi turtles sau anumite patch-uri, se poate crea un buton de observator, care apoi să utilizeze comanda ask pentru a cere doar unora dintre turtles sau patch-uri să facă ceva.)

Când se editează un buton, există opţiunea de a desemna o tastă asociată butonului. Acest lucru face ca acea tastă de pe tastatură să se comporte la fel ca o apăsare a butonului. Dacă butonul respectiv este un buton Forever, acesta va rămâne activ până când tasta este apăsată din nou (sau este apăsat butonul). Tastele sunt deosebit de utile pentru jocuri sau orice model în care este necesară declanşarea rapidă a butoanelor.

Butoane pe rând

La un moment dat, mai mult de un buton poate fi apăsat. Dacă se întâmplă acest lucru, butoanele merg "pe rând", ceea ce înseamnă că doar un singur buton execută o acţiune la un moment dat. Fiecare buton îşi rulează codul până la capăt, o singură dată, în timp ce celelalte butoane aşteaptă, apoi următorului buton îi vine rândul.

În următoarele exemple, “setup” este un buton Once iar “go” este un buton Forever.

Exemplul # 1: Utilizatorul apasă butonul “Setup”, apoi apasă “go” imediat, înainte ca “setup” să apară iar. Rezultat: acţiunea “setup” se termină înainte de a începe “go”.

Exemplul # 2: În timp ce butonul “go” este apăsat, utilizatorul apasă “setup”. Rezultat: butonul de “go” îşi termină iterația curentă, apoi se activează butonul “setup”, apoi începe să ruleze din nou “go”.

Exemplul # 3: Utilizatorul are două butoane Forever apăsate în acelaşi timp. Rezultat: primul buton îşi derulează codul său până la capăt, apoi celălalt îşi execută codul său până la final, şi aşa mai departe, alternativ.

A se reţine că dacă un buton intră într-o buclă infinită, atunci nici un alt buton nu va rula.

Butoanele Forever turtle, patch, si link

Există o diferenţă subtilă între a plasa comenzi într-un buton Forever turtle, patch sau link , şi de a plasa aceleaşi comenzi într-un buton de observator care execută `ask turtles`, `ask patches` sau `ask links`. O acţiune “ask” nu se va finaliza până când toţi agenţii nu vor termina de rulat toate comenzile din “ask”. Deci agenţii, pentru că toţi execută comenzile în acelaşi timp, pot fi desincronizaţi între ei, dar toţi se vor sincroniza din nou la sfârşitul cerinţei. Acest lucru nu este valabil în cazul butoanelor Forever turtle, patch sau link. Din moment ce `ask` nu a fost folosit, fiecare turtle sau patch rulează codul dat în mod repetat, deci ei pot deveni (şi rămâne) desincronizaţi între ei.

În prezent, această capacitate este foarte rar folosită în modelele din Models Library. Un model care poate utiliza această capacitatea este modelul Termite, în secţiunea Biologie din Sample Models. Butonul “go” este un buton Forever turtle, astfel că fiecare termită continuă treaba independent de oricare altă termită, iar observatorul nu este deloc implicat. Acest lucru înseamnă că dacă, de exemplu, s-a dorit adăugarea de căpuşe şi/sau o acţiune modelului, va trebui adăugat un al doilea buton Forever (un buton observator Forever), şi să se acţioneze ambele butoane Forever în acelaşi timp. De remarcat că, de asemenea, un model ca acesta nu poate fi folosit cu BehaviorSpace.

Exemplu de cod: State Machine Example arată cum Termite pot fi recodate înr-un mod bazat pe bifare, fără a mai folosi un buton Forever turtle.

În momentul de faţă, NetLogo nu permite ca la apăsarea unui buton Forever să se activeze altul. Butoane sunt pornite numai atunci când sunt apăsate.

2.2. Liste

În modelele cele mai simple, fiecare variabilă are doar un fragment de informaţie, de obicei un număr sau un şir. Listele permit stocarea mai multor părţi de informaţie într-o singură valoare prin

colectarea acelor informaţii într-o listă. Fiecare valoare din listă poate fi de orice tip: un număr, sau un şir de caractere, un agent sau un set de agenţi, sau chiar o altă listă.

Listele permit încorporarea convenabilă de informaţii în NetLogo. În cazul în care agenţii efectuează un calcul repetitiv pentru mai multe variabile, ar putea fi mai uşor să existe o listă de variabile, în loc de mai multe variabile numerice. Câţiva primitivi simplifică procesul de realizare a aceluiaşi calcul pentru fiecare valoare dintr-o listă.

Dicţionarul NetLogo are o secţiune care specifică toţi primitivii asociaţi listelor.

Liste de constante

Se poate crea o listă punând pur şi simplu valorile dorite în listă între paranteze, astfel: `set mylist [2 4 6 8]`. De remarcat faptul că valorile individuale sunt separate de spaţii. Se pot crea astfel liste care conţin numere sau şiruri, dar şi liste în interiorul altor liste, ca de exemplu: `[[2 4] [3 5]]`.

O listă goală este creată nepunând nimic între paranteze: `[]`.

Crearea de liste în mod dinamic

Dacă se doreşte crearea unei liste în care valorile sunt determinate de raportori, şi nu ca o serie de constante, se va folosi raportorul `list`. Raportorul `list` acceptă alţi doi raportori, îi rulează şi raportează rezultatele sub forma unei liste.

Dacă se dorește ca o listă să conţină două valori aleatoare, se poate folosi următorul cod:

```
set random-list list (random 10) (random 20)
```

Aceasta va stabili o `random list` ca pe o nouă listă de două numere întregi aleatoare de fiecare dată când rulează.

Pentru a face liste mai lungi sau mai scurte, se poate folosi raportorul `list` cu mai puţin sau mai mult de două intrări, dar pentru a putea realiza acest lucru, trebuie inclusă întreaga cerinţă în paranteze, ca de exemplu :

```
(list random 10)
(list random 10 random 20 random 30)
```

Unele tipuri de liste sunt cel mai uşor de construit folosind raportorul `n-values`, care permite crearea unei anumite liste prin executarea repetată a unui raportor dat. Se poate face o listă cu aceeaşi valoare repetată, sau cu toate numerele dintr-un şir, sau o mulţime de numere aleatorii, sau multe alte posibilități.

Primitiva `of` permite crearea unei liste pornind de la un set de agenţi. Aceasta raportează o listă

care conţine valoarea fiecărui agent pentru raportorul dat. (Raportorul poate fi un simplu nume de variabilă, sau o expresie mai complexă -- chiar şi o cerinţă pentru o procedură, definită prin utilizarea `to-report`.) Un idiom comun este

```
max [...] of turtles sum
[...] of turtles
```

şi aşa mai departe.

Se pot combina două sau mai multe liste cu ajutorul raportorului `sentence`, care leagă listele combinând conţinutul lor într-o singură listă mai mare. Ca şi `list`, `sentence` primeşte în mod normal două intrări, dar poate accepta orice număr de intrări în cazul în care cerinţa este între paranteze.

Schimbarea elementelor din listă

Din punct de vedere tehnic, listele nu pot fi modificate, dar se pot construi noi liste bazate pe listele vechi. Dacă se doreşte ca noua listă să o înlocuiască pe cea veche, se foloseşte `set`. De exemplu:

```
set mylist [2 7 5 Bob [3 0 -2]] ;
mylist is now [2 7 5 Bob [3 0 -2]]
set mylist replace-item 2 mylist 10 ;
mylist is now [2 7 10 Bob [3 0 -2]]
```

Raportorul `replace-item` primeşte trei intrări. Prima specifică ce element din listă va fi modificat. 0 desemnează primul element, 1 următorul, şi aşa mai departe.

Pentru adăugarea unui element, de exemplu 42, ca fiind sfârşitul unei liste, se utilizează raportorul `lput` .(`fput` adaugă un element la începutul unei liste.)

```
set mylist lput 42 mylist
; mylist is now [2 7 10 Bob [3 0 -2] 42]
```

Dar dacă utilizatorul se răzgândeşte? Raportorul `but-last` (prescurtat `bl`) raportează toate elementele din listă cu excepţia ultimului.

```
set mylist but-last mylist
; mylist is now [2 7 10 Bob [3 0 -2]]
```

Să presupunem că se doreşte eliminarea elementului 0, de la începutul listei:

```
set mylist but-first mylist
; mylist is now [7 10 Bob [3 0 -2]]
```

Să presupunem că se doreşte modificarea celui de-al treilea element care este în interiorul elementului 3, de la -2 la 9. Numele ce poate apela lista integrată [3 0 -2] este `item 3 mylist`. Apoi raportorul `replace-item` poate schimba o listă din interiorul altei liste. Parantezele sunt adăugate pentru claritate.

Iterarea peste liste

Dacă se intenţionează efectuarea unor de operaţii pentru fiecare element dintr-o listă pe rând, comanda `foreach` şi raportorul `map` pot fi de ajutor.

`foreach` este folosit pentru a rula o comandă sau mai multe comenzi pentru fiecare element dintro listă. Este nevoie de o listă de intrări şi un nume de comandă sau un bloc de comenzi, cum ar fi:

```
foreach [1 2 3] show
=> 1
=> 2 => 3
foreach [2 4 6]
       [ crt ?
              show (word "created " ? " turtles") ]
=> created 2 turtles
=> created 4 turtles
=> created 6 turtles
```

În bloc, variabila `?` păstrează valoarea curentă din lista de intrări.

Aici sunt mai multe exemple pentru `foreach`:

```
foreach [1 2 3] [ ask turtles [ fd ? ] ] ;;
turtles move forward 6 patches
foreach [true false true true] [ ask turtles [ if ? [ fd 1 ] ] ]
;; turtles move forward 3 patches
```

`map` este similar cu `foreach`, dar este un raportor. Este necesară o listă de intrări şi un nume de raportor sau un bloc de raportori. Spre deosebire de `foreach`, raportorul acţionează primul, de exemplu:

```
show map round [1.2 2.2 2.7]
;; prints [1 2 3]
```

`map` raportează o listă care conţine rezultatele aplicării raportorului la fiecare element din lista de intrări. Din nou, se foloseşte `?` pentru a se face referire la elementul curent din listă.

Aici sunt câteva exemple de ulilizare a `map`:

```
show map [? < 0] [1 -1 3 4 -2 -10]
;; prints [false true false false true true] show
map [? * ?] [1 2 3]
;; prints [1 4 9]
```

În afară de map şi foreach, alte primitive utilizate în procesarea listelor întregi într-un mod configurabil includ filter, reduce, şi sort-by.

Aceste primitive nu reprezintă întotdeauna o soluţie pentru fiecare situaţie în care se doreşte operarea cu o listă întreagă. În anumite situaţii, este posibil să fie necesară folosirea altor tehnici, cum ar fi utilizarea unor bucle repeat sau while, sau o prodecudă recursivă.

Blocurile de cod furnizate la map şi foreach în aceste exemple sunt în realitate **sarcini**. Sarcinile sunt explicate mai detaliat in Sarcini, mai jos.

Numărul variabil de intrări

Unele comenzi şi raportori care implică liste şi şiruri de caractere pot avea un număr variabil de intrări. În aceste cazuri, pentru a le transmite un număr de intrări diferit de cel implicit, primitivele şi intrările lor trebuiesc încadrate de paranteze. Iată câteva exemple:

```
show list 1 2 =>
[1 2]
show (list 1 2 3 4)
=> [1 2 3 4] show
(list)
=> []
```

A se reţine faptul că fiecare dintre aceste comenzi speciale are un număr implicit de intrări pentru care nu sunt necesare parantezele. Primitivele ce au această capacitate sunt list, word, sentence, map, şi foreach.

Listele de intrări

Anterior, s-a specificat faptul că seturile de agenţi sunt într-o ordine aleatoare, de fiecare dată alta. Dacă se solicită ca agenţii să efectueze o acţiune înt-o ordine fixă, atunci aceştia trebuie ordonaţi într-o listă.

Există două primitive ce se pot utiliza în acest sens, sort şi sort-by.

Atât sort cât şi sort-by pot primi un set de agenţi ca intrare. Întotdeauna, rezultatul este o nouă listă, ce conţine aceeaşi agenţi ca şi setul de agenţi, dar într-o anumită ordine.

Dacă se foloseşte sort pe un set de agenţi turtle, rezultatul este o listă de turtles sortați ascendent după numărul ~~who~~.

Dacă se foloseşte ~~sort~~ pe un set de agenţi patch, rezultatul este o listă de patch-uri ordonate de la stânga la dreapta şi de sus în jos.

Dacă se foloseşte `sort` pe un set de agenţi link, rezultatul este o listă de link-uri ordonate ascendent mai întâi după `end1` şi apoi după `end2` ; orice legături rămase sunt gestionate de rasă în ordinea în care sunt declarate în tab-ul Code.

Dacă în schimb e nevoie de o ordine descendentă, se poate combina `reverse` cu `sort`, de exemplu `reverse sort turtles`.

Dacă se doreşte ordonarea agenţilor după un alt criteriu decât cel standard folosit de `sort`, va fi nevoie de utilizarea lui `sort-by`.

Iată un exemplu:

```
sort-by [[size] of ?1 < [size] of ?2] turtles
```

Aceasta returnează o listă de turtles sortaţi în ordine crescătoare după variabila proprie `size`.

Cererea pentru o listă de agenţi

Odată ce există o listă a agenţilor, se poate cere fiecăruia să execute ceva. Pentru a face acest lucru, se folosesc combinat comenzile `foreach` şi `ask`, astfel:

```
foreach sort turtles [ ask
       ? [
              ...
       ]
]
```

Astfel, se cere fiecărui turtle ordonarea ascendentă după numărul `who`. Pentru a cere patch-uri ordonate de la stânga la dreapta şi de sus în jos, se va înlocui “turtles” cu “patches”.

A se reţine că nu se poate utiliza ask direct pentru o listă de turtles. ask funcţionează numai pentru seturi de agenţi şi cu agenţi individuali.

Performanţa listelor

Structura de date care stă la baza listelor NetLogo este un tip de arbore, unde cele mai multe operaţiuni rulează în timp aproape constant. Aceasta include `fput`, `lput`, `butfirst`, `butlast`, `length`, `item`, şi `replace-item`.

O excepţie de la regula performanţei rapide este că înlănţuirea a două liste cu `sentence` necesită parcurgerea şi copierea în întregime a celei de-a doua liste. (Acest lucru va fi poate îmbunătăţit întro versiune viitoare.)

Din punct de vedere tehnic, “timpul aproape constant” este, de fapt, timp logaritmic, proporţional cu adâncimea arborelui; însă aceşti arbori au noduri mari şi un factor de ramificare ridicat, astfel încât

ei nu au niciodată mai mult de câteva nivele. Acest lucru înseamnă că modificările pot fi făcute în cel mult câţiva paşi. Arborii sunt invariabili, dar pot avea o structură comună unul cu celălalt, astfel încât nu este necesar să fie copiat întreg arborele pentru a crea o versiune modificată.

2.3. Matematică

Toate numerele din NetLogo sunt stocate intern ca numere de dublă precizie în virgulă mobilă, aşa cum sunt definite în standardul IEEE 754. Acestea sunt numere pe 64 biţi, constând dintr-un un bit de semn, un exponent de 11 biţi, şi o mantisă de 52 de biţi. A se vedea standardul IEEE 754 pentru detalii.

Un "număr întreg" în NetLogo este pur şi simplu un număr care se întâmplă să nu aibă nici o parte fracţionară. Nu se face nici o distincţie între 3 şi 3.0; ele reprezintă acelaşi număr. (Acest lucru este similar cu modul în care majoritatea oamenilor folosesc numerele în viaţa de zi cu zi, dar diferit de anumite limbaje de programare. Unele limbaje tratează numerele întregi şi cele în virgulă mobilă ca fiind de tipuri distincte.)

Întregii sunt întotdeauna tipăriţi de NetLogo fără coada "0.0":

```
show 1.5 + 1.5
observer: 3
```

În cazul în care un număr cu o parte fracţionară apare într-un context în care se aşteaptă un număr întreg, partea fracţionară este pur şi simplu îndepărtată. Deci, de exemplu, `crt 3.5` creează trei turtles, 0.5 fiind ignorat.

Gama de numere întregi este de +/-9007199254740992 (2^53, aproximativ 9 cvadrilioane). Calculele care depăşesc acest interval nu vor provoca erori de execuție, dar precizia va avea mult de suferit în cazul în care cifrele cel mai puţin semnificative (binare) sunt rotunjite, pentru a se încadra într-un număr pe 64 de biţi. În cazul numerelor foarte mari, această rotunjire poate duce la comportamente imprecise care pot fi surprinzătoare:

```
show 2 ^ 60 + 1 = 2 ^ 60
=> true
```

Calculele cu numere mai mici pot de asemenea produce rezultate surprinzătoare dacă implică părți fracţionare, întrucât nu toate fracţiile pot fi reprezentate precis şi pot apărea rotunjiri. De exemplu:

```
show 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6 + 1 / 6
=> 0.9999999999999999
show 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9 + 1 / 9
=> 1.0000000000000002
```

Orice operaţiune care generează valorile speciale ”infinity” sau ”no number”, va provoca eroare de execuţie.

Notaţie ştiinţifică

Numerele cu virgulă mobilă foarte mari sau foarte mici sunt afişate de NetLogo folosind notaţia ştiinţifică. Exemple:

```
show 0.000000000001 =>
1.0E-12
show 50000000000000000000
=> 5.0E19
```

Numerele din notaţia ştiinţifică se disting prin prezenţa literei E (“exponent”). Aceasta înseamnă “la puterea a zecea”. De exemplu, 1.0E-12 reprezintă 1.0 ori 10 la puterea -12:

```
show 1.0 * 10 ^ -12
=> 1.0E-12
```

De asemenea, oricine poate folosi notaţia ştiinţifică în codul NetLogo:

```
show 3.0E6 =>
3000000 show
8.123456789E6 =>
8123456.789 show
8.123456789E7 =>
8.123456789E7 show
3.0E16 => 3.0E16
show 8.0E-3 =>
0.0080 show 8.0E-4
=> 8.0E-4
```

Numerele cu părţi fracţionare sunt afişate folosind notaţia ştiinţifică dacă exponentul este mai mic decât -3 sau mai mare decât 6. Numerele aflate în afara intervalului întreg NetLogo de la 9007199254740992 la -9007199254740992 (+/-2 ^ 53) sunt, de asemenea, reprezentate întotdeauna prin notaţia ştiinţifică:

```
show 2 ^ 60
=> 1.15292150460684698E18
```

Litera E poate fi majusculă sau nu. Când se afișează un număr, NetLogo utilizează întotdeauna un E mare:

```
show 4.5e20 =>
4.5E20
```

Acurateţea virgulei

Deoarece numerele din NetLogo sunt supuse limitărilor legate de modul în care numerele cu virgulă sunt reprezentate binar, se pot obţine răspunsuri uşor inexacte. De exemplu:

```
show 0.1 + 0.1 + 0.1 =>
0.30000000000000004
show cos 90
=> 6.123233995736766E-17
```

Aceasta este o problemă inerentă aritmeticii fracţionare, ea apare în toate limbajele de programare care utilizează numerele cu virgulă.

Dacă se lucrează cu cantităţi de precizie fixe, de exemplu cu dolari şi cenţi, o tehnică comună este folosirea doar a numerelor întregi (cenţi) pe plan intern, apoi valoarea se împarte la 100 pentru a obţine un rezultat în dolari pentru a fi afişat.

Dacă este necesară folosrea numerelor cu virgulă, atunci în unele situaţii este trebuie înlocuit un test de egalitate simplă, cum ar fi `if x = 1 [ ... ]` cu un test ce permite uşoare imprecizii, de exemplu `if abs (x - 1) < 0.0001 [ ... ]`.

De asemenea, primitiva precision este utilă pentru rotunjirea numerelor în scopuri de afişare. De asemenea, monitoarele NetLogo rotunjesc numerele pe care le afişează la un numar configurabil de zecimale.

2.4. Numere aleatoare

Numerele aleatoare folosite de NetLogo sunt ceea ce se numeşte "pseudo-aleatoare". (Acest lucru este tipic în programare.) Asta înseamnă că, deși par aleatoare, ele sunt, de fapt, generate de un proces determinist. “Determinist” înseamnă că se obţin aceleaşi rezultate de fiecare dată, dacă se pornește de la același seed.

În contextul unei modelări ştiinţifice, numerele pseudo-aleatoare sunt de fapt preferabile celor aleatoaret. Aceasta deoarece este important ca un experiment ştiinţific să poată fi reprodus - ca oricine să-l poată încerca şi să obţină acelaşi rezultat. Deoarece NetLogo utilizează numere pseudo-aleatoare, “experimentele” făcute cu el pot fi reproduse şi de către alte persoane.

Iată cum funcţionează: Generatorul NetLogo de numere aleatoare poate fi pornit cu o anumită valoare seed, care trebuie să fie un număr întreg în intervalul de la -2147483648 la 2147483647. Odată ce generatorul a primit un seed cu comanda `random-seed`, acesta va genera întotdeauna aceeaşi secvenţă de numere aleatoare. De exemplu, dacă se execută aceste comenzi:

```
random-seed 137 show
random 100 show
random 100 show
random 100
```

Se vor obţine întotdeauna numerele 79, 89, şi 61, în această ordine.

A se reţine, totuşi, că este garantată obţinerea aceloraşi numere doar dacă se utilizează aceeaşi versiune de NetLogo. Uneori, atunci când se creează o nouă versiune a NetLogo generatorul de

numere aleatoare se schimbă. (În prezent, se foloseşte un generator cunoscut ca Mersenne Twister.)

Pentru a obține un număr adecvat pentru generatea unui seed, se foloseşte raportorul `new-seed`. Acesta creează un seed folosind ora și data curentă. Acesta nu returnează același seed de două ori la rând.

Exemplu de cod: Random Seed Example

Dacă nu se setează sămânţa aleatoare, NetLogo îi stabileşte o valoare pe baza datei si orei curente. Nu există nici o cale de a afla care seed aleator a fost ales, deci, dacă se doreşte ca execuţia modelului să fie reproductibilă, trebuie setat seed-ul în prealabil.

Primitivele NetLogo cu “random” în numele lor (random, random-float, şi aşa mai departe), nu sunt singurii care utilizează numerele pseudo-aleatoare. Multe alte operaţiuni fac, de asemenea, alegeri aleatoare. De exemplu, seturile de agenţi sunt întotdeauna în ordine aleatoare, `one-of` şi `n-of` îşi aleg la întâmplare agenţii, comanda `sprout` creează turtles cu culori şi poziţii aleatoare, iar raportorul `downhill` alege un patch la întâmplare atunci când există o egalitate. Toate aceste alegeri aleatoare sunt reglementate de asemenea de seed-uri aleatoare, astfel încât rularea modelului poate fi reprodusă.

În plus față de numerele întregi aleatoare uniform distribuite şi numerele cu virgulă generate de `random` şi `random-float`, NetLogo oferă de asemenea câteva alte distribuţii aleatoare. A se consulta intrările din dicţionar pentru `random-normal`, `random-poisson`, `randomexponential`, şi `random-gamma`.

Generatorul auxiliar

Codul executat de către butoane sau din centrul de comandă foloseşte generatorul de numere aleatoare principal.

Codul din monitoare foloseşte un generator aleator auxiliar, astfel încât, chiar dacă un monitor face un calcul care utilizează numere aleatoare, rezultatul modelului nu este afectat. Acelaşi lucru este valabil pentru codurile din cursoare.

Dezordinea pe loc fix

Este de dorit să se specifice în mod explicit că o secţiune a codului nu afectează starea generatorului principal de numere aleatoare, astfel încât rezultatul modelului să nu fie afectat. Comanda `with-local-randomness` este prevăzută în acest scop. A se vedea intrarea sa în NetLogo Dictionary pentru mai multe informaţii.

2.5. Forme turtle

În NetLogo, formele de turtle sunt vectoriale. Ele sunt construite din forme geometrice de bază: pătrate, cercuri şi linii, mai degrabă decât dintr-o reţea de pixeli. Formele vectoriale sunt complet scalabile şi rotabile. NetLogo utilizează imagini bitmap ale formelor vectoriale de dimensiuni de 1, 1.5 și 2, pentru a accelera execuţia.

Forma unui turtle este stocată în variabila `shape` şi poate fi setată folosind comanda `set`.

Turtles noi au o formă implicită. Primitiva `set-default-shape` modifică forma implicită a turtle.

Primitiva `shapes` raportează lista a formelor disponibile la acel moment în model. Acest lucru este util, de exemplu, dacă se intenţionează atribuirea unei forme aleatoare unui turtle:

```
ask turtles [ set shape one-of shapes ]
```

Se utilizează Turtle Shapes Editor pentru a crea forme proprii de turtle, sau pentru a adăuga forme pentru modele din biblioteca de forme, sau pentru a transfera forme între diferite modele. Pentru mai multe informații, consultaţi secţiunea Shapes Editor din acest manual.

Grosimea liniilor folosite pentru a desena formele vectoriale poate fi controlată de către primitiva `set-line-thickness`.

Exemplu de cod: Breeds şi Shapes Example, Shape Animation Example

2.6. Forme link

Formele link sunt similare cu formele turtle, doar că se foloseşte Link Shape Editor pentru a le crea şi edita. Formele link sunt alcătuite din 0 până la 3 linii care pot avea diferite tipare, şi dintr-un indicator de direcţie, care este compus din aceleași elemente ca şi formele turtle. Link-urile au, de asemenea, o variabilă ~~`shape`~~ care poate fi setată oricărei forme link din model. Prin link-urile implicite se obţine forma "default", deşi există posibilitatea de a modifica acest lucru utilizând ~~`set-default-shape`~~. Raportorul ~~`link-shapes`~~ raportează toate formele link incluse în modelul actual.

Grosimea liniilor din formele link este controlată de către variabila link `thickness`.

2.7. Vizualizarea actualizărilor

Meniul "View" în NetLogo permite vizualizarea agenților modelului existent pe ecranul computerului observând cum agenții se mișcă sau își schimbă starea.

NetLogo creează o imagine care va permite să vedeți cum arată agenții la un anumit moment dat. Odată cu trecerea timpului, agenții își modifică starea și astfel imaginea trebuie să fie actualizată.

NetLogo oferă două moduri de actualizări, "continue" și "bazate pe tick-uri". Se poate comuta între cele două tipuri de actualizări, folosind un meniu pop-up din partea de sus a tabului Interface.

Actualizările continue sunt implicite la pornirea NetLogo sau când se creează un nou model. Aproape fiecare model din biblioteca de modele folosește actualizări bazate pe tick-uri.

Actualizările continue sunt mai simple, dar cele bazate pe tick-uri oferă mai mult control asupra momentului în care se realizează actualizările și asupra frecvenței cu care acestea se produc.

Este important momentul în care se realizează o actualizare, deoarece atunci se determină ceea ce se afișează pe ecran. Dacă o actualizare se efectuează într-un moment neașteptat, s-ar putea vedea ceva ce poate crea confuzie sau ceva ce poate induce în eroare.

Este important cât de des se realizează actualizările, deoarece acestea necesită timp. Cu cât NetLogo petrece mai mult timp pentru actualizarea imaginii, cu atât va rula mai lent modelul creat.
Cu cât sunt mai puține actualizări, cu atât modelul rulează mai repede.

Actualizări continue

Actualizările continue sunt foarte simple. NetLogo actualizează imaginea de un anumit număr de ori pe secundă - implicit de 30 de ori pe secundă atunci când slider-ul de viteză este în poziția implicită, la mijloc.

Dacă slider-ul de viteză este mutat la o setare mai lentă, NetLogo va actualiza mai mult de 30 de ori pe secundă, încetinind efectiv modelul. Pe o setare mai rapidă, NetLogo va actualiza mai puțin de 30 de ori pe secundă. Pe setări rapide, actualizările succesive vor fi separate de mai multe secunde.

La setări extrem de lente, NetLogo va actualiza atât de des încât se vor vedea agenții mișcându-se (sau schimbându-și culoarea, etc), unul câte unul.

Dacă este nevoie să se dezactiveze temporar actualizările continue, se va utiliza comanda `nodisplay`. Comanda `display` pornește din nou actualizările și forțează, de asemenea, o actualizare imediată (dacă utilizatorul nu setează modelul pe fast forward, folosind slider-ul de viteză).

Actualizările bazate pe tick-uri

Așa cum s-a discutat mai sus în secțiunea Tick Counter, în unele modele NetLogo, timpul trece în pași discreți, numiți "tick"-uri. De obicei, se dorește vizualizarea actualizărilor o dată pe tick, între tick-uri. Acesta este comportamentul implicit al actualizărilor bazate pe tick-uri.

Dacă se doresc actualizări suplimentare, se poate forța o actualizare folosind comanda `display` (actualizarea poate fi omisă în cazul în care utilizatorul forțează modelul pe fast forward, folosind slider-ul de viteză).

Nu trebuie să se utilizeze „tick counter”-ul pentru a folosi actualizări bazate pe tick-uri. În cazul în care „tick counter”-ul nu avansează niciodată, imaginea se va actualiza numai atunci când se utilizează comanda `display`.

Dacă slider-ul de viteză va fi mutat la o setare mai rapidă, în cele din urmă NetLogo va sări peste o parte din actualizările care s-ar fi realizat în mod normal. Mutarea cursorului de viteză la o setare mai lentă nu produce actualizări suplimentare, NetLogo face o pauza după fiecare actualizare. Cu cât viteza este mai lentă, cu atât pauza va fi mai lungă.

Chiar și în cazul actualizărilor bazate pe tick-uri, imaginea se actualizează ori de câte ori se declanșează un buton din interfață (atât butoanele "once", cât și cele "forever"), și atunci când o comandă dată în Command Center se termină de executat. Nu este necesară adăugarea comenzii `display` pentru butoanele "once" care nu incrementează „tick counter”-ul. Multe butoane permanente care nu incrementează „tick counter”-ul trebuie să folosească comanda `display`. Un exemplu din Libraria de modele este modelul Life (Computer Science - > Cellular Automat). Butoanele permanente, care permit utilizatorului să traseze linii în imagine, utilizează comanda `display`, astfel încât utilizatorul să poată vedea ceea ce se trasează, chiar dacă „tick counter”-ul nu se incrementează.

Alegerea unui mod

Avantajele actualizărilor bazate pe tick-uri față de actualizările continue includ:

1. Un comportament consecvent, previzibil, al actualizărilor care nu variază de la calculator la calculator sau de la rulare la rulare.
2. Actualizările continue pot crea confuzie, afișând stări nedorite ale modelului, ceea ce poate conduce la o eroare.
3. Viteza crescută. Actualizarea imaginii necesită timp; astfel, dacă o actualizare pe tick este suficientă, atunci modelul va rula mai rapid.
4. Deoarece butoanele de `setup` nu incrementeaza „tick counter”-ul, ele nu sunt afectate de slider-ul de viteză; acesta este, de cele mai multe ori, comportamentul dorit.

După cum s-a menționat mai sus, cele mai multe modele din biblioteca de modele folosesc actualizările bazate pe tick-uri.

Actualizările continue sunt utile pentru modelele a căror execuție nu este împărțită în faze scurte și discrete. Un exemplu din biblioteca de modele sunt Termitele (Termites). (A se vedea, de asemenea, exemplul cu modelul State Machine, care arată cum să recodăm Termitele, folosind tick-urile).

Comutarea temporară la actualizări continue ar putea fi utilă pentru scopuri de depanare chiar si pentru modelele care în mod normal sunt setate pe actualizări bazate pe tick-uri. Vizualizarea a ceea ce se întâmplă într-un tick, în loc de rezultatul final, ar putea ajuta la depanare. După trecerea la actualizări continue, s-ar putea utiliza slider-ul pentru a încetini viteza modelului până când se vor vedea agenții mișcându-se unul câte unul. Este necesară revenirea la actualizările bazate pe tickuri, deoarece alegerea modului de actualizare este salvată odată cu modelul.

Frame rate

Una dintre setările modelului din NetLogo este "rata cadrelor" (frame rate), care este implicit setată la 30 de cadre pe secundă.

Setarea de cadre afectează atât actualizările continue cât și actualizările bazate pe tick-uri.

În cazul actualizărilor continue, setarea determină în mod direct frecvența de actualizări. În cazul celor bazate pe tick-uri, setarea este un plafon pentru cât de multe actualizări pe secundă se efectuează. Dacă rata este de 30, atunci NetLogo va asigura faptul că modelul nu rulează mai rapid atunci când slider-ul de viteză este în poziția implicită. Dacă procesarea oricărui cadru durează mai puțin de 1/30 secunde, NetLogo va face o pauză și va aștepta până când vor trece cele 1/30 secunde înainte de a continua.

3.1 . Trasarea (Plotarea)

Înainte de plotare, trebuie create una sau mai multe parcele în tab-ul Interface. Pentru mai multe informații cu privire la utilizarea și editarea parcelelor, se va consulta Ghidul de interfață (Interface Guide).

Puncte de plotare

Cele două comenzi de bază pentru plotare sunt `plot` și `plotxy`.

Pentru `plot` trebuie specificată doar valoarea y care va fi reprezentată. Valoarea x va fi în mod automat 0 pentru primul punct trasat, 1 pentru al doilea, și așa mai departe. (Asta dacă "intervalul" de trasare are valoarea implicită 1, se poate schimba intervalul dacă se dorește acest lucru).

Comanda `plot` este deosebit de utilă atunci când se dorește ca modelul să traseze un nou punct la fiecare pas. Exemplu:

```
plot count turtle
```

Dacă se dorește specificarea atât a valorilor lui x, cât și celor ale lui y ale punctului dorit, se va utiliza `plotxy`. Acest exemplu presupune că există o variabilă globală numită `time`:

```
plotxy time count-turtles
```

Comenzile plot

Fiecare parcelă și stilourile (pens) sale de trasat au domenii de configurare și actualizare care pot conține comenzi (de obicei, `plot` sau `plotxy`). Aceste comenzi sunt rulate automat când sunt declanșate de alte comenzi în NetLogo.

Comenzile pentru configurarea plotului și comenzile pentru configurarea instrumentelor de trasat sunt rulate atunci când sunt lansate comenzile `reset-ticks` sau `setup-plots`. În cazul în care comanda `stop` se execută în corpul de configurare al plotării atunci comanda pentru configurarea instrumentelor de trasare nu va fi rulată.

Comenzile pentru actualizarea plotării și cele pentru actualizarea instrumentelor de trasat sunt rulate atunci la execuția uneia din următoarele comenzi: `reset-ticks`, `tick` sau `updateplots`. În cazul în care comanda `stop` se execută în corpul de actualizare al plotării, comanda pentru actualizarea instrumentelor de trasare nu va fi rulată.

In continuare vor fi prezentate patru comenzi care declanșează trasarea, explicate mai detaliat.

- `setup-plots` execută comenzile pentru un plot la un moment dat, comanda rulând pentru fiecare plot în parte. În cazul în care comanda de oprire nu este întâlnită în timp ce rulează acele comenzi, fiecare dintre instrumentele de trasat vor avea codul de configurare executat.
- `update-plots` este foarte similar cu `setup-plots`. Pentru fiecare plot, comenzile de actualizare ale plotului sunt executate. În cazul în care comanda de oprire nu este întâlnită în timp ce execută aceste comenzi, fiecare dintre instrumentele de trasat vor avea codul de actualizare executat.
- `tick` este exact la fel ca `update-plots` cu excepția faptului că tick-counter-ul este incrementat înainte de a executa comenzile de plotare.
- `reset-ticks` resetează întâi tick-counter-ul la 0, și apoi face echivalentul lui `setupplots` urmat de `update-plots`.

Un model tipic va folosi `reset-ticks` și `tick` astfel:

```
to setup   clear-
all
  ...
reset-ticks
end to go
```

```
...   tick
end
```

De reținut că în acest exemplu s-au plotat procedurile `setup` și `go` (deoarece `reset-ticks` rulează comenzile pentru configurarea și actualizarea plotului). S-a făcut acest lucru deoarece plotul trebuie să includă starea inițială a sistemului dupa procedura `setup`. S-a plotat la sfârșitul procedurii `go`, și nu la început, deoarece plotul trebuie să fie întotdeauna actualizat după ce se oprește butonul Go.

Modelele care nu utilizează tick-uri, dar doresc să facă trasarea, vor folosi în schimb `setup-plots` și `update-plots`. În codul anterior, înlocuiți `reset-ticks` cu `setup-plots update-plots` și înlocuiți `tick` cu `update-plots`.

Exemplu de cod: Plotting Example

În mod implicit, instrumentul de trasare în NetLogo este setat pe modul linie, astfel încât punctele trasate sunt conectate printr-o linie.

Dacă se dorește mutarea stiloului (pen-ului) fără a trasa, se va folosi comanda `plot-pen-up`. După ce această comandă este apelată, comenzile `plot` și `plotxy` mută stiloul, fără a trasa nimic. Odată ce pointer-ul este acolo unde se dorește, se va utiliza comanda `plot-pen-down` pentru a activa din nou stiloul.

Dacă se dorește plotarea unor puncte individuale în loc de linii, sau se dorește trasarea barelor în loc de linii sau puncte, trebuie schimbat "modul" stilou al plotului. Sunt disponibile trei moduri: linie (implicit), bară și punct. Este posibilă schimbarea temporară a modului stilou folosind comanda `set-plot-pen-mode`. Comanda primește un număr drept intrare: 0 pentru linie, 1 pentru bară, 2 pentru punct.

Histograme

O histogramă este un tip special de plotare care măsoară cât de frecvent apar anumite valori într-o colecție de numere din model.

De exemplu, se presupune că agenții turtle din model au o vârstă variabilă. Se va putea crea o histogramă a distribuției vârstelor agenților turtle folosind comanda:

```
histogram [age] of turtles
```

Numerele din histogramă nu trebuie neapărat să provină dintr-un set, acestea ar putea fi din orice listă de numere.

Utilizarea comenzii `histogram` nu schimbă automat modul stilou în modul bare. Pentru aceasta, trebuie setat acest mod manual. (Așa cum am spus înainte, se poate schimba modul implicit stilou prin editarea plotului în tab-ul Interface.)

Lățimea barelor într-o histogramă este controlată de intervalul stiloului din plot. Se poate seta intervalul implicit prin editarea plotului în tab-ul Interface. De asemenea, se poate schimba temporar intervalul folosind comenzile `set-plot-pen-interval` sau `set-histogram-num-bars`. Dacă se utilizează comanda din urmă, NetLogo va seta intervalul corespunzător, astfel încât numărul specificat de bare să se potrivească în gama axei x.

Exemplu de cod: Histogram Example

Ștergerea și resetarea

Se poate șterge plot-ul curent folosind comanda `clear-plot`, sau se pot sterge toate plot-urile din model folosind `clear-all-plots`. Comanda `clear-all` elimină, de asemenea, toate ploturile, și șterge orice altceva din model.

Dacă se dorește eliminarea punctelor create cu un anumit stilou, se va utiliza comanda `plot-penreset`.

Atunci când un întreg plot este șters, sau atunci când un stilou este resetat, nu se elimină doar datele care au fost reprezentate grafic, ci se restabilește, de asemenea, plot-ul sau stiloul la setările implicite, așa cum au fost specificate în tab-ul Interface atunci când plot-ul a fost creat sau editat ultima dată. Prin urmare, efectele comenzilor precum `set-plot-x-range` și `set-plot-pencolor` sunt doar temporare.

Limite și scalare automată

Limitele implicite x și y ale unui plot sunt numere fixe, dar ele pot fi modificate în timpul configurării sau pe măsură ce modelul rulează.

Pentru a modifica limitele în orice moment, se utilizează `set-plot-x-range` și `set-plot-yrange`. Sau, se pot lăsa intervalele să crească în mod automat. În orice caz, atunci când plotul este șters, limitele vor reveni la valorile lor implicite.

În mod implicit, toate ploturile din NetLogo au funcția de autoscalare activată. Aceasta înseamnă că, dacă modelul încearcă să ploteze un punct care este în afara limitelor de afișare, limita de plotare va crește de-a lungul uneia sau a ambelor axe, astfel încât noul punct să fie vizibil.

În ipoteza că intervalele nu vor trebui schimbate de fiecare dată când se adaugă un nou punct, atunci când limitele cresc, ele lasă un spațiu suplimentar: 25% dacă cresc orizontal, 10% în cazul în care cresc pe verticală.

Dacă se dorește dezactivarea acestei caracteristici, se va edita plotul și se va debifa Auto Scale. În prezent, nu este posibilă activarea sau dezactivarea aceastei funcții doar pe o singură axă, se aplică întotdeauna la ambele axe.

Folosirea legendei

Se poate afișa legenda unui plot bifând casuța "Show legend" în caseta de dialog edit. Dacă nu se dorește ca un anumit stilou să apară în legendă, se poate debifa casuța "Show legend" pentru stiloul respectiv, de asemenea, în setările avansate pentru plot (setările avansate pentru plot pot fi deschise dând click pe butonul cu iconița creion pentru stiloul respectiv în tabelul de stilouri în caseta de dialog plot edit).

Stilouri temporare

Cele mai multe ploturi se pot obține cu un număr fix de stilouri. Dar unele ploturi sunt mai complexe; acestea pot avea nevoie de mai multe stilouri în funcție de condiții. În astfel de cazuri, se pot crea stilouri "temporare" din cod și apoi se pot plota cu ele. Aceste stilouri dispar atunci când plotul este eliminat (de catre `clear-plot`, `clear-all-plots` sau `clear-all`).

Pentru a crea un stilou temporar, se va utiliza comanda `create-temporary-plot-pen`. De obicei, acest lucru ar putea fi făcut în tab-ul Code, dar este, de asemenea, posibil să se utilizeze această comandă din configurarea plotului sau din codul de actualizare al plotului (în caseta de dialog Edit). În mod implicit, noul stilou este activat, are culoarea neagră, un interval de 1, și ploteaza în modul linie.

Înainte de a putea utiliza stiloul, vor trebui lansate comenzile `set-current-plot` și `setcurrent-plot-pen`.

set-current-plot și set-current-plot-pen

Înainte de NetLogo 5, nu a fost posibilă specificarea comenzilor direct în plotul de sine stătător. Tot codul plotului era scris în tab-ul Code, împreună cu restul codului. Pentru compatibilitatea cu modelele mai vechi, și pentru ploturile temporare, această caracteristică este în continuare valabilă. Modelele din versiunile anterioare ale NetLogo (și cele care folosesc plotările cu stilourile temporare) trebuie să precizeze în mod explicit care plot este cel curent folosind comanda `setcurrent-plot` și care stilou este cel curent folosind comanda `set-current-plot-pen`.

Pentru a seta plotul curent folosim comanda `set-current-plot` cu numele plotului scris în ghilimele: `set-current-plot "Distance vs. Time"`

Numele plotului trebuie să fie exact asa cum a fost introdus atunci când a fost creat. Ulterior, dacă se schimbă numele plotului, vor trebui actualizate apelurile `set-current-plot` în model pentru a folosi noul nume. (Copy-paste poate fi de ajutor aici.)

Pentru un plot cu mai multe stiluri, se poate specifica manual care stilou să fie folosit pentru plotare. Dacă nu este specificat nici un stilou, plotarea va avea loc cu primul stilou din plot. Pentru a plota cu un stilou diferit, se folosește comanda `set-current-plot-pen` cu numele stiloului între ghilimele:

```
set-current-plot-pen "distance"
```

Odată ce stiloul curent este stabilit, comanda `plot count turtle` poate fi executată pentru acel stilou.

Modelele mai vechi cu ploturi au, de obicei, propria lor procedură `do-plotting` care arată astfel:

```
to do-plotting
   set-current-plot "populations"
set-current-plot-pen "sheep"    plot
count sheep    set-current-plot-pen
"wolves"    plot count wolves
set-current-plot "next plot"
   ... end
```

Acest lucru nu mai este necesar în NetLogo 5, cu excepția cazului în care se utilizează stilouri de plotare temporare.

Concluzii

Nu au fost explicate toate aspectele sistemului de plotare în NetLogo. Pentru mai multe informații despre comenzile adiționale și reporteri relaționari cu plotarea vizitați secțiunea Plotting din dicționarul NetLogo.

Majoritatea modelelor din biblioteca de modele ilustrează diverse tehnici avansate de plotare. De asemenea, analizați și următoarele exemple de cod:

Exemplu de cod: Plot Axis Example, Plot Smoothing Example, Rolling Plot Example

3.2 . Șiruri de caractere

Șirurile de caractere pot conține orice caracter Unicode.

Pentru a introduce o constantă de tip șir de caractere în NetLogo, aceasta se va încandra între ghilimele.

Șirul gol este scris astfel: " ".

Majoritatea primitivelor de listă funcționează și cu șiruri de caractere, precum:

```
but-first "string" => "tring"
but-last "string" => "strin"
empty? "" => true empty?
"string" => false first
"string" => "s" item 2
"string" => "r"
last "string" => "g" length "string" =>
6 member? "s" "string" => true member?
"rin" "string" => true member? "ron"
"string" => false position "s" "string"
=> 0 position "rin" "string" => 2
position "ron" "string" => false remove
"r" "string" => "sting" remove "s"
"strings" => "tring" replace-item 3
"string" "o" => "strong" reverse
"string" => "gnirts"
```

Câteva primitive sunt specifice șirurilor de caractere, cum ar fi `is-string?`, `substring`, `word`:

```
is-string? "string" => true is-
string? 37 => false substring
"string" 2 5 => "rin"
word "tur" "tle" => "turtle"
```

Șirurile de caractere pot fi comparate cu ajutorul operatorilor: =, !=, <, >, <=, >=.

Dacă se dorește încorporarea unui caracter special într-un șir, se vor utiliza următoarele secvențe:

- \n = linie nouă
- \t = tab
- \" = ghilimele
- \\ = backslash

3.3 . Ieșire (Output)

Această secțiune tratează problema afișării pe ecran. Ieșirea la ecran poate fi, de asemenea, salvată într-un fișier folosind comanda `export-output`. Dacă este nevoie de o metodă mai flexibilă de a scrie date în fișierele externe, se va consulta secțiunea următoare, File I/O.

Comenzile de bază pentru generarea ieșirii pe ecran în NetLogo sunt `print`, `show`, `type` și `write`. Aceste comenzi trimit ieșirea lor la Centrul de Comanda. Pentru informții complete legate de aceste patru comenzi, vezi detaliile lor în dicționarul NetLogo. Iată cum sunt acestea utilizate de obicei:

- `print` este utilă în majoritatea situațiilor.
- `show` permite să vedem ce anume afișează fiecare agent.
- `type` permite afișarea mai multor elemente pe aceeași linie.
- `write` permite afișarea valorilor într-un format care poate fi citit apoi folosind `file-read`.

Un model NetLogo poate avea, opțional, o "zonă de ieșire", în tab-ul său de interfață, pe lângă Command Center. Pentru a trimite dateel de ieșire în acea zonă în loc de Centrul de Comandă, se vor utiliza comenzile `output-print`, `output-show`, `output-type` si `output-write`.

Zona de ieșire poate fi ștearsă folosind comanda `clear-output` și salvată într-un fișier folosind comanda `export-output`. Conținutul zonei de ieșire va fi salvat prin comanda `export-world`. Comanda `import-world` șterge zona de ieșire și setează conținutul său la valoarea din fișierul importat. Trebuie remarcat faptul că o cantitate mare de date trimisă la zona de ieșire poate mări dimensiunile fișierelor exportate.

Dacă se utilizează `output-print`, `output-show`, `output-type`, `output-write`, `clearoutput` sau `export-output` într-un model care nu are o zonă de ieșire separată, atunci comenzile se aplică pentru porțiunea de ieșire din centrul de comandă.

3.4 . Fișiere I/O (input/output)

În NetLogo, există un set de primitive care oferă posibilitatea de a interacționa cu fișierele din exterior. Toate încep cu prefixul file-.

Există două moduri principale atunci când se lucrează cu fișiere: citire și scriere. Diferența este direcția fluxului de date. Atunci când se citesc informațiile dintr-un director, datele care sunt stocate în fișier sunt transferate în model. Pe de altă parte, scrierea permite ca datele să se transfere din model către un fișier.
Atunci când un model NetLogo rulează ca un applet într-un browser web, acesta va avea doar posibilitatea de a citi datele din fișierele care sunt în același director cu modelul. Applet-urile nu pot scrie în toate fișierele.

Atunci când se lucrează cu fișiere, întotdeauna se utilizează mai întâi primitiva `file-open`. Aceasta specifică fișierul cu care se va interacționa. Nici una din celelalte primitive nu vor funcționa decât dacă se deschide mai întâi un fișier.

O altă primitivă pentru fișiere specifică modul în care se va găsi fișierul până când acesta va fi închis: citire sau scriere. Pentru a comuta între cele două moduri, vom închide și apoi vom redeschide fișierul.

Primitivele pentru citire includ: `file-read`, `file-read-line`, `file-read-characters` și `file-at-end?`. Fișierul trebuie să existe deja, înainte de a putea fi deschis pentru citire.

Exemplu de cod: File Input Example

Primitivele pentru scriere sunt similare cu cele de afișare în Command Center, cu excepția faptului că ieșirea se salvează într-un fișier. Acestea includ `file-print`, `file-show`, `file-type` și `file-write`. De reținut este faptul că nu se pot "suprascrie" date. Cu alte cuvinte, dacă se încearcă scrierea într-un fișier cu date existente, toate datele noi vor fi adaugate la sfârșitul fișierului. (Dacă se dorește suprascrierea într-un fișier, se va utiliza `file-delete` pentru a-l șterge, apoi trebuie deschis pentru scriere.)

Exemplu de cod: File Output Example

Când s-a încheiat lucrul cu un fișier, se poate folosi comanda `file-close` pentru terminarea sesiunii de utilizare a fișierului. Dacă ulterior se dorește ștergerea fișierului, se va utiliza primitiva `file-delete`. Pentru a închide mai multe fișiere deschise, trebuie selectat mai întâi fișierul utilizând `file-open` înainte de a-l închide.

```
;; Open 3 files file-open
"myfile1.txt" file-open
"myfile2.txt" file-open
"myfile3.txt"

;; Now close the 3 files
file-close file-open
"myfile2.txt" file-close
file-open "myfile1.txt"
file-close
```

Sau, dacă se vrea închiderea tuturor fișierelor, se va utiliza `file-close-all`.

Două primitive demne de remarcat sunt `file-write` și `file-read`. Acestea sunt salveze și preiau constante din NetLogo, cum ar fi numere, liste, constante de tip boolean și șiruri de caractere. Primitiva `file-write` va afișa întotdeauna variabilele astfel încât primitiva `file-read` să le poată interpreta corect.

```
file-open "myfile.txt" ;; Opening file for writing ask
turtles
   [ file-write xcor file-write ycor ] file-close

file-open "myfile.txt" ;; Opening file for reading ask
turtles
   [ setxy file-read file-read ] file-close
```

Exemplu de cod: File Input Example și File Output Example

Primitivele `user-directory`, `user-file` și `user-new-file` sunt utile atunci când se lasă libertatea utilizatorului de a alege un fișier sau director de lucru.

3.5 . Capturarea de filme

Această secțiune descrie modul în care se poate captura un film QuickTime al unui model NetLogo.

Mai întâi, se va utiliza comanda `movie-start` pentru a începe un film nou. Numele fișierului va trebui să se termine cu extensia .mov, specifică filmelor QuickTime.

Pentru a adăuga un cadru la film, folosim fie `movie-grab-view` sau `movie-grab-interface`, în funcție de ceea ce dorim să apară în film: doar vizualizarea curentă sau întregul tab Interface. Într-un singur film, trebuie utilizată o singură dată primitiva `movie-grab-` sau oricare alta; nu pot fi amestecate. Când am terminat de adăugat cadre, utilizăm `movie-close`.

```
;; export a 30 frame movie of the view setup
movie-start "out.mov"
movie-grab-view ;; show the initial state
repeat 30 [ go
   movie-grab-view ] movie-close
```

În mod implicit, un film va rula cu 15 cadre pe secundă. Pentru a crea un film cu un alt frame rate, se va apela `movie-set-frame-rate`, cu un număr diferit de cadre pe secundă. Trebuie setat numărul de cadre pe secundă, după `movie-start`, dar înainte de a capta vreun cadru.
Pentru a verifica frame rate-ul filmului sau pentru a vedea numărul de cadre capturate, se va apela `movie-status`, care raportează un șir de caractere ce descrie starea filmului curent.

Pentru a renunța la un film și pentru a-l șterge, se folosește `movie-cancel`.

Exemplu de cod: Movie Example

Filmele pot fi generate doar de NetLogo GUI, dar nu atunci când ruleaza fără header sau când în fundal se execută un model în paralel.

Filmele NetLogo sunt exportate ca fișiere QuickTime necomprimate. Pentru a reda un film QuickTime, se poate folosi QuickTime Player, un soft care se descarcă gratuit de la Apple.

Având în vedere că filmele nu sunt arhivate, ele pot ocupa mult spațiu pe disc. Cel mai probabil se va dori comprimarea filmelor cu un software third-party. Software-ul poate oferi diferite tipuri de compresie. Unele tipuri de compresie sunt fără pierderi, în timp ce altele au pierderi. "Pierderile" înseamnă că, pentru reducerea dimensiunii fișierelor, unele detalii din film se pierd. În funcție de natura modelului, se poate dori evitarea compresiei cu pierderi, de exemplu, în cazul în care filmul conține detalii de finețe, la nivel de pixel.

Un pachet software care poate comprima filmele QuickTime atât pe Mac cat și pe Windows este QuickTime Pro. Pe Mac, iMovie funcționeaza la fel de bine. Compresia cu PNG este un bun exemplu de compresie fără pierderi.

3.6. Perspectiva

Vizualizările 2D și 3D arată modelul din punctul de vedere al observatorului. În mod implicit, observatorul privește modelul din originea spre sensul pozitiv al axei Z. Se poate schimba punctul de vedere al observatorului, prin utilizarea `follow`, `ride` si `watch` (comenzi pentru observator) și `follow-me`, `ride-me` si `watch-me` (comenzi pentru agentul turtle). În modul follow sau modul ride, observatorul se deplasează odată cu agentul. Diferența dintre follow și ride este vizibilă doar în vederea 3D. În vederea 3D utilizatorul poate modifica distanța din spatele agentului folosind mouseul. Atunci când observatorul urmează agentul de la distanța zero el de fapt se afla în modul ride. În cazul în care observatorul este în modul watch, el urmărește mișcările unui agent turtle fără să se miște. În ambele vizualizări se va vedea un prim-plan, iar în vederea 3D observatorul se va întoarce cu fața către subiect. Pentru a determina agentul care este folosit se va utiliza reporterul `subject`.

Exemplu de cod: Plot Axis Example, Plot Smoothing Example, Rolling Plot Example

3.7 . Spațiul de desen

Spațiul de desen este un layer unde agenții turtle pot face semne vizibile.

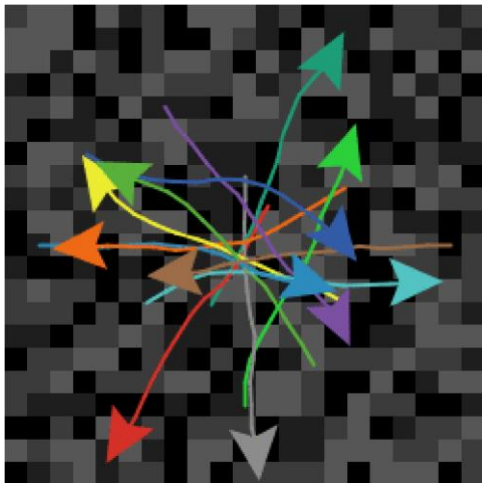
El apare deasupra patch-urilor, dar sub agenții turtle. Inițial, este gol și transparent. Se poate observa acest spațiu, însă agenții turtle (și patch-urile) nu pot percepe desenul și nu pot reacționa la el.

Agenții turtle pot desena și șterge liniile din desen utilizând comenzile `pen-down` și `pen-erase`. Când stiloul unui agent turtle este jos (sau pe modul de ștergere), agentul turtle trasează (sau șterge) o linie în spatele său ori de câte ori se mișcă. Liniile sunt de aceeași culoare ca și agentul. Pentru a opri desenarea (sau ștergerea), se folosește `pen-up`.

Liniile trasate de agenții turtle au în mod normal grosimea de un pixel. Dacă se dorește o grosime diferită, se va seta variabila agentului `pen-size` la un numar diferit înainte de a trasa (sau șterge). Pentru agenții noi, variabila este setată la 1.

Liniile trasate atunci când un agent se deplasează într-o direcție imprecisă, cum rezultă din utilizarea `setxy` sau `move-to`, vor fi trasate pe cea mai scurtă direcție care satisface topologia curentă.

Iată câtiva agenți turtle care au realizat un desen pe o grila cu patch-uri colorate aleator. Observați cum agenții acopera liniile și liniile acoperă culorile patch-ului. Pen-size-ul folosit aici a fost 2:



Comanda `stamp` permite unui agent turtle să lase o imagine a sa în urma în desen, iar `stamperase` permite eliminarea pixelilor de dedesubt în desen. Pentru a șterge întregul desen se va utiliza comanda `clear-drawing`. (De asemenea, se poate folosi `clear-all`, care șterge orice altceva.)

Importarea unei imagini

Comanda `import-drawing` permite importarea unui fișier imagine de pe disc.

Este utilă doar pentru a furniza un fundal. Daca se dorește ca agenții turtle și patch-urile să reacționeze la imagine, ar trebui utilizate comenzile `import-pcolors` sau `import-pcolorsrgb`.

Comparație cu alte Logo-uri

Desenarea funcționează oarecum diferit în NetLogo decât în alte Logo-uri.

Diferențele semnificative includ:

- Stilourile noilor agenți sunt sus, nu jos.
- În loc de comanda `fence` pentru limitarea agentului turtle în interiorul granițelor, în NetLogo se editează lumea și se setează wrapping off.
- Nu există `screen-color`, `bgcolo`r sau `setbg`. Background-ul poate fi făcut solid prin colorarea patch-urilor, de exemplu `ask patches [ set pcolor blue ]`.

Caracteristici care nu sunt acceptate de către NetLogo:

- Nu există comanda `window`. Aceasta este utilizată în alte Logo-uri pentru a lăsa agentul turtle să se deplaseze pe un plan infinit.
- Nu există comenzile `flood` sau `fill` pentru a umple un spațiu închis cu o culoare.

4.1 . Topologie

Modul în care mulțimea de patch-uri este conectată se poate schimba. În mod implicit lumea este un tor, ceea ce înseamnă că este nemărginită, dar poate să „înfășoare”, adică atunci când un agent turtle trece de marginea lumii, el dispare și reapare pe marginea opusă și fiecare patch are același număr de patch-uri "vecine". În cazul unui un patch de la marginea lumii, o parte dintre "vecinii" săi sunt pe marginea opusă.

Cu toate acestea, se pot modifica setările de înfășurare folosind butonul Settings. În cazul în care înfășurarea nu este permisă într-o direcție dată, atunci în acea direcție (x sau y) lumea este mărginită. Patch-urile de-a lungul acestei limite vor avea mai puțin de 8 vecini și agenții turtle nu se vor muta dincolo de marginea lumii.

Topologia lumii NetLogo are patru valori posibile: tor, cutie, cilindru vertical sau cilindru orizontal. Topologia este controlata prin activarea sau dezactivarea teleportarii pe direcția x sau y. Lumea implicită este tor-ul.

Un tor înfășoară în ambele direcții, ceea ce înseamna că marginile de sus și cele de jos ale lumii sunt conectate, la fel și marginile din stânga și din dreapta. Dacă un agent turtle se mută dincolo de marginea din dreapta a lumii, el va apărea din nou pe stânga, același lucru fiind valabil pentru partea de sus și de jos.

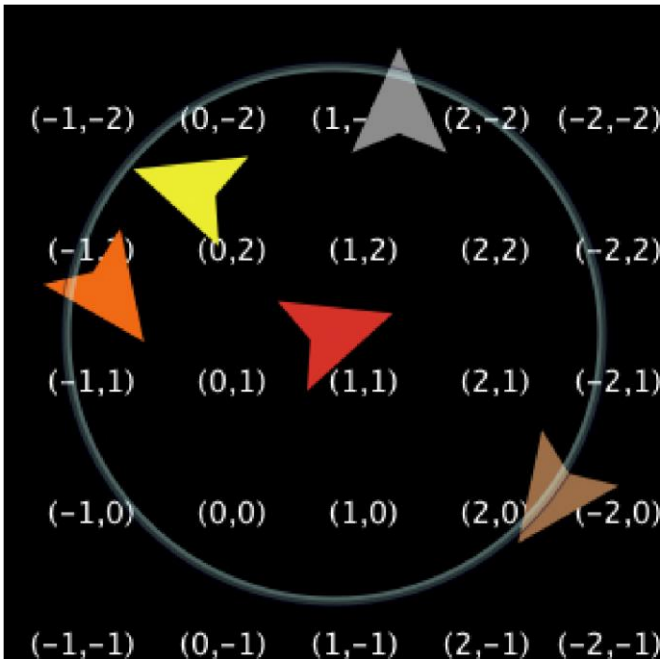
O cutie nu înfășoară în orice direcție. Lumea este delimitată astfel încât agenții turtle care încearcă să se mute dincolo de marginea lumii nu pot face acest lucru. Patch-urile din jurul marginii au mai puțin de opt vecini, colțurile au trei, iar restul au cinci.

Cilindrii orizontali și verticali înfășoară într-o singură direcție, dar nu și în cealaltă. Un cilindru orizontal înfășoară în plan vertical, astfel încât partea de sus a lumii este conectată cu partea de jos, iar marginile din stânga și din dreapta sunt delimitate. Un cilindru vertical înfășoară în plan orizontal astfel încât marginile din stânga și din dreapta sunt conectate, dar marginile de sus și de jos sunt delimitate.

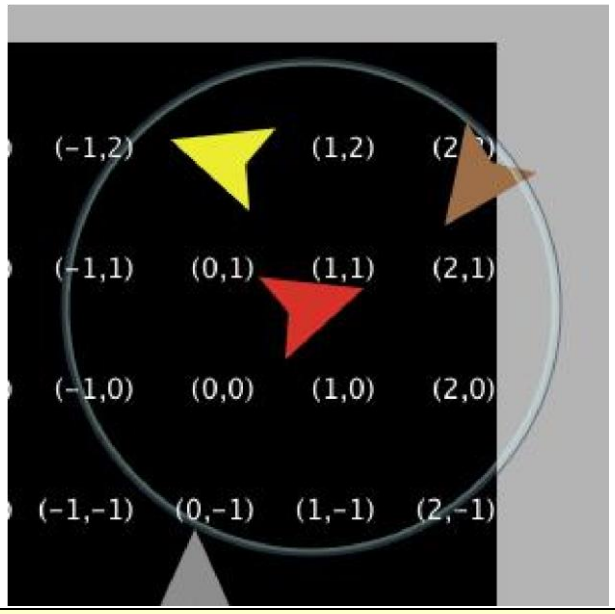
Exemplu de cod: Neighbors Example

Când coordonatele se înfășoară, agenții turtle și link-urile se înfășoară și ei vizual. În cazul în care o formă de agent turtle sau link se extinde peste o margine, o parte din ea va apărea la cealaltă margine. (Agenții turtle propriu-ziși sunt puncte care nu au acces la spațiu, astfel încât aceștia nu pot fi pe ambele părți ale lumii în același timp, dar în view ei par să aibă acces la spațiu, deoarece forma lor conține mai mult de un punct.)

Înfășurarea, afectează modul în care arată view-ul atunci când se află în modul follow al unui agent turtle. Pe un tor, oriunde se deplasează agentul turtle, se va vedea întotdeauna toată lumea din jurul său:



Într-o cutie sau cilindru lumea are margini, astfel încât zonele dincolo de acele margini apar în vedere cu gri:



Exemplu de cod: Termites Perspective Demo (torus), Ants Perspective Demo (box)

Setările de topologie pot controla comportamentul primitivelor `distance(xy)`, `in-radius`, `incone`, `face(xy)` și `towards(xy)`. Topologia controlează dacă primitivele înfășoară sau nu. Ele folosesc mereu calea cea mai scurtă permisa de topologie. De exemplu, distanța de la centrul patch-urilor până în colțul din dreapta jos (`min-pxcor`, `min-pycor`) și până în colțul din stânga sus (`max-pxcor`, `max-pycor`) va fi după cum urmează, pentru fiecare topologie, având în vedere că min și max pxcor si pycor sunt +/- 2:

- tor - sqrt(2) ~ 1.414 (va fi același pentru toate dimensiunile lumii fiindcă patch-urile sunt direct diagonale într-un tor.)
- box - sqrt(world - lățime^2 + world – înălțime^2) ~ 7.07
- cilindru vertical - sqrt(world – înălțime ^2 + 1) ~ 5.099
- cilindru orizontal - sqrt (world – lățime^2 + 1) ~ 5.099

Toate celelalte primitive vor acționa în mod similar cu `distance`. Dacă s-au folosit deja primitive `nowrap` în model se recomandă eliminarea lor și schimbarea topologiei lumii.

Dacă modelul are agenți turtle care se deplasează, va trebui luat în considerare ceea ce se întâmplă cu ei atunci când aceștia ajung la marginea lumii, în cazul în care topologia utilizată are unele margini fără înfășurare (no-wrapping). Sunt câteva cazuri frecvente: agentul este reflectat înapoi în lume (fie sistematic, fie aleatoriu), agentul iese din sistem (moare) sau agentul este ascuns. Nu mai este necesară verificarea limitelor folosind coordonatele agenților, în schimb se poate verifica în NetLogo dacă un agent este la marginea lumii. Exista câteva moduri de a face acest lucru, cel mai simplu este utilizarea primitivei `can-move?`.

```
if not can-move? distance [ rt 180 ]
```

`can-move?` returnează true dacă poziția primitivei `distance` în fața agentului este în interiorul NetLogo, și false în caz contrar. În acest caz, dacă agentul este la marginea lumii, el pur și simplu merge înapoi în același mod în care a ajuns acolo. Se poate folosi `patch-ahead 1 ! = nobody` în loc de `can-move?`. Dacă vrem să facem ceva mai interesant decât să întoarcem pur și simplu agentul ar putea fi utilă folosirea `patch-at` cu `dx` și `dy`.

```
if patch-at dx 0 = nobody [
set heading (- heading)
]
if patch-at 0 dy = nobody [
set heading (180 - heading)
]
```

Aceasta testează dacă agentul lovește un perete orizontal sau vertical și dacă este respins de perete.

În unele modele, în cazul în care un agent nu poate merge mai departe, el pur și simplu moare (iese din sistem).

```
if not can-move? distance[ die ]
```

Dacă agenții sunt mutați folosind `setxy` în loc de `forward`, ar trebui testat dacă patch-ul unde va fi mutat există, deoarece `setxy` returnează o eroare de execuție în cazul în care coordonatele date sunt în afara lumii. Aceasta este o situație frecventă în cazul în care modelul simulează un plan infinit, caz în care și agenții din afara sa ar fi pur și simplu ascunși.

```
let new-x new-value-of-xcor let
new-y new-value-of-ycor

ifelse patch-at (new-x - xcor) (new-y - ycor) = nobody
   [ hide-turtle ]
[ setxy new-x new-y
      show-turtle ]
```

Numeroase din biblioteca de modele folosesc această tehnică: gravitația și electrostatica sunt cele mai bune exemple.

Comenzile `diffuse` si `diffuse4` se comportă corect în toate topologiile. Fiecare patch se răspândește și o cantitate egală a variabilei de difuzie se transmite la fiecare dintre vecinii săi, în cazul în care are mai puțin de 8 vecini (sau 4 dacă utilizați `diffuse4`), restul ramân împrăștiați pe patch-uri. Aceasta înseamnă că suma totală a variabilelor patch-urilor în întreaga lume rămâne constantă. Cu toate acestea, dacă se dorește ca împrăștierea să cadă de pe marginile lumii așa cum ar fi pe un plan infinit, trebuiesc șterse marginile la fiecare pas ca în exemplul dispersiei pe margini.

4.2 . Link-uri (legături)

O legatură este un agent care conectează doi agenți de tip turtle. Acești agenți turtle sunt uneori numiți noduri.

Link-ul este întotdeauna trasat ca o linie între cei doi agenți turtle. Link-urile nu au o locație, aceștia nu sunt considerați a fi pe vreun patch și nu se poate determina distanța de la o legatură la un alt punct.

Există două variante de link-uri, orientate și neorientate. Un link orientat este de la un nod către un alt nod. Relația dintre un parinte și un copil poate fi modelată ca un link orientat. O legătură neorientată este la fel pentru ambele noduri, fiecare nod are o legatură cu celalalt nod. Relația dintre soți, sau frați, ar putea fi modelată ca o legatură neorientată.
Exista un „agentset" global al tuturor link-urilor, la fel ca pentru agenții turtle si pentru patch-uri. Se pot crea link-uri neorientate utilizând comenzile `create-link-with` și `create-links-with`; și link-uri orientate folosind comenzile `create-link-to`, `create-links-to`, `create-link-from` și `create-links-from`. După ce primul link a fost creat (orientat sau neorientat), toate celelalte link-uri fără rasă trebuie să se potrivească, este imposibil să existe două link-uri fără rasă dintre care unul este orientat, iar celălalt este neorientat. O eroare de execuție apare dacă se încearcă acest lucru.

În general, primitivele care lucrează cu link-uri orientate conțin în numele lor "in", "out", "to" și "from". Cele neorientate fie omit acestea, fie folosesc "with".

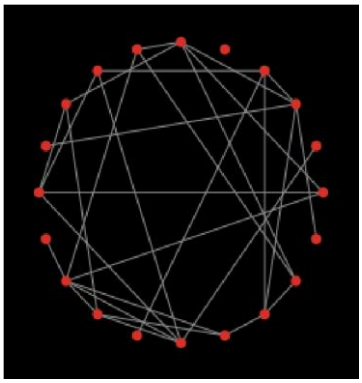
Variabilele `end1` și `end2` ale unei legături conțin cei doi agenți turtle pe care link-ul îi conectează. În cazul în care link-ul este orientat, direcția este de la `end1` la `end2`. În cazul în care link-ul este neorientat, `end1` este întotdeauna cel mai vechi dintre cei doi agenți turtle, adică agentul turtle cu numărul mai mic.

Rasele de link-uri vor permite definirea diferitelor tipuri de link-uri în model. Ele trebuie să fie orientate sau neorientate; spre deosebire de link-urile fără rasă, acestea sunt definite în timpul compilării si nu în timpul rulării. Declararea rasei unei legături se face folosind: `undirectedlink-breed` si `directed-link-breed`. Link-urile cu rasă pot fi create folosind comenzile: `create-<breed>-with` și `create-<breeds>-with` pentru rasele neorientate iar pentru cele orientate se vor folosi comenzile `create-<breed>-to`, `create-<breeds>-to`, `create<breed>-from` și `create-<breeds>-from`.

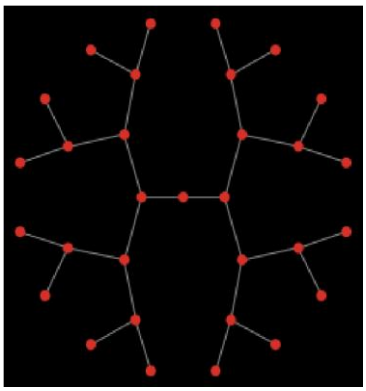
Nu pot exista mai multe link-uri neorientate din aceeași rasă (sau două legături fără rasă) între o pereche de agenți, nici mai mult de o legătură orientată a aceleiași rase în aceeași direcție între o pereche de agenți. Pot exista două link-uri orientate din aceeași rasă (sau două legături fără rasă) între o pereche de agenți în cazul în care sunt în direcții opuse.

Layout-uri

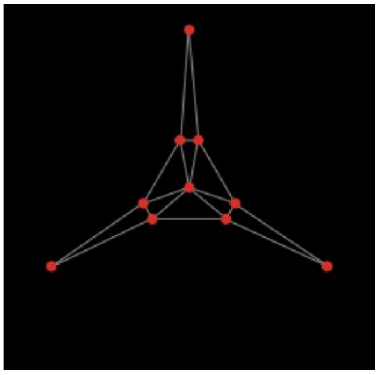
Există o serie de primitive care ajută la vizualizarea rețelelor. Cel mai simplu este `layout-circle` care poziționează uniform agenții în jurul centrului lumii la o anumită rază.



`layout-radial` este un util dacă lucrați cu ceva asemănător unei structuri arborescente; chiar dacă exista bucle în arbore, acesta va funcționa în continuare. `layout-radial` are un agent rădăcină pentru nodul central care este plasat la (0,0) și aranjează nodurile conectate la acesta întrun model concentric. Nodurile la distanță de un grad de rădăcină vor fi aranjate într-un model circular în jurul nodului central și la nivelul urmator în jurul acestor noduri și așa mai departe. `layout-radial` va încerca să țină seama de graficele asimetrice și oferă mai mult spațiu ramurilor care sunt mai mari. `layout-radial` are, de asemenea, o rasă ca intrare astfel încât se utilizează o anumită rasă de link-uri pentru afișarea unei anumite rețele și nu a alteia.



Se are în vedere un set de noduri de ancorare, astfel `layout-tutte` plasează toate celelalte noduri în centrul masei nodurilor la care este legat. Setul de ancoră este aranjat în mod automat într-un aspect de cerc cu rază definită de utilizator și celelalte noduri vor converge la acesta (desigur, modelul va trebui rulat de mai multe ori înainte ca aspectul să fie stabil.)



`layout-spring` este util pentru multe tipuri de rețele, dezavantajul fiind că este relativ lent, deoarece este nevoie de mai multe iterații pentru a converge. În acest aspect, link-urile acționează ca niște arcuri care trag nodurile conectate unele cu altele și de nodurile care se resping reciproc. Puterea forțelor arcurilor este controlată de intrările primitivelor. Aceste intrări vor avea întotdeauna valori între 0 și 1; chiar și schimbările foarte mici pot afecta în continuare aspectul rețelei. Arcurile au, de asemenea, o lungime (în unități de patch-uri); cu toate acestea, din cauza tuturor forțelor implicate, nodurile nu vor ajunge exact la aceeași distanță unul față de celălalt.

Exemplu de cod: Termites Network Example, Network Import Example, Giant Component, Small Worlds, Preferential Attachment

4.3 . Sarcini (task-uri)

Task-urile vă permit să stocați cod pentru a fi rulat mai târziu. Există două tipuri de task-uri: de comandă și de reporter.

Task-urile sunt valori, ceea ce înseamnă că un task poate fi specificat ca intrare, raportat ca și rezultat sau poate fi stocat într-o variabilă.

Un task poate fi executat o singură dată, de mai multe ori, sau deloc.

În alte limbaje de programare task-urile sunt cunoscute ca funcții de primă clasă, de închidere, sau lambda.

Primitive de task-uri

Noile primitive asociate cu task-urile sunt `task`, `is-command-task?` și `is-reporter-task?`.

Comanda `run` acceptă task-uri de comandă precum și șiruri de caractere.

Reporterul `runresult` acceptă task-urile de reporter precum și șiruri de caractere.

Aceste primitive acceptă de asemenea și task-uri: `foreach`, `map`, `reduce`, `filter`, `n-values`, `sort-by`. (Sintaxa este compatibilă cu codul existent de la versiunile NetLogo anterioare.)

Primitiva `task` creează un task. Task-ul la care se raportează ar putea fi un task de comandă sau un task de reporter, în funcție de tipul de bloc pe care îl conține. De exemplu, `task [ fd 1 ]` raportează un task de comandă, deoarece `fd` este o comandă, în timp ce `task [ count turtles ]` raportează un task de reporter deoarece `count` este un reporter.

Intrările task-urilor

Un task poate avea zero sau mai multe intrări. Intrările sunt referite folosind variabilele speciale `?` `(aka ?1), ?2, ?3`, etc. Orice intrări suplimentare sunt ignorate.

Task-uri și șiruri de caractere

Crearea și rularea unui task se face rapid. Utilizarea `run` sau `runresult` pe un șir nou pentru prima dată este de 100 de ori mai lentă decât rularea unui task. De aceea, ar fi de preferat utilizarea taskurilor în locul rulării șirurilor de caractere, cu excepția cazului șirurilor de caractere introduse de utilizator.

Sintaxa concisă

Utilizările simple ale: `foreach`, `map`, `reduce`, `filter`, `n-values` și `sort-by` pot fi acum scrise cu o sintaxă mai concisă. Astfel:

```
map abs [1 -2 3 -4] => [1 2 3 4]
reduce + [1 2 3 4] => 10 filter is-
number? [1 "x" 3] => [1 3] foreach
[1 2 3 4] print
```

În versiunile NetLogo mai vechi, acestea erau scrise astfel:

```
map [abs ?] [1 -2 3 -4] => [1 2 3 4]
reduce [?1 + ?2] [1 2 3 4] => 10 filter
[is-number? ?] [1 "x" 3] => [1 3]
foreach [1 2 3 4] [ print ? ]
```

Sintaxa veche ramâne valabilă.
Sintaxa concisă poate fi, de asemenea, utilizată cu primitiva `task`. De exemplu, `task die` este forma scurtă pentru `task [ die ]`, `task fd` este forma scurtă pentru `task [fd ?]`, iar `task`

`+` este forma scurtă pentru `task [?1 + ?2]`.

Task-uri de încheiere

Există și task-uri "de încheiere"; ceea ce înseamnă că ele capturează sau "închid" legăturile (nu doar valorile curente) de variabile locale și intrările de procedură. Ele nu includ variabilele de agent și nu captează identitatea agentului curent.

Ieșirile nelocale

Comenzile `task` și `report` ies din procedura de închidere dinamică, nu din task-ul de închidere. (Acest lucru este compatibil cu versiunile mai vechi de NetLogo.)

Task-uri și extensii

Extensiile API suportă primitive de scriere care acceptă task-uri ca intrare.

Limitări

Vom aborda cel puțin câteva din următoarele limitări în versiunile viitoare NetLogo:

- `import-world` nu acceptă task-uri.
- Intrările task-urilor nu pot avea nume. Acest lucru poate face ca returnarea task-urilor de raport sa fie dificilă. (Task-urile de comandă pot utiliza `let` pentru a da nume intrărilor, dacă este necesar.)
- Task-urile nu pot fi variadice (adică nu pot accepta un număr variabil de intrări).
- Task-urile de reporter nu pot conține comenzi, doar o singură expresie de reporter. De exemplu trebuie utilizat `ifelse-value` în loc de `if`, și nu trebuie utilizat deloc `report`. În cazul în care codul este prea complex pentru a fi scris ca un reporter, va trebui mutat codul într-o procedură de reporter separată, și apoi apelată acea procedură din task, dându-i un număr de intrări necesare.
- Task-urile nu sunt interschimbabile cu blocuri de comandă și blocuri de reporter. Doar primitivele enumerate mai sus acceptă task-uri ca intrări. Primitivele de control, cum ar fi `ifelse` și `while` și primitivele de agenți, cum ar fi `of` și `with` nu acceptă task-uri. De exemplu, dacă avem un task de raport `r` și două task-uri de comandă `c1` si `c2`, nu putem scrie `ifelse r c1 c2`, ci trebuie să scriem `ifelse runresult r [ run c1 ] [ run c2 ]`.

Exemplu de cod: State Machine Example

4.4 . ask-concurrent

În versiunile foarte vechi de NetLogo, `ask` a fost concurent în mod implicit. De la NetLogo 4.0 (2007) încoace, `ask` este serial, adică agenții rulează pe rând comenzile în interiorul unui ask.

Următoarele informații descriu comportamentul comenzii `ask-concurrent`, care se comportă la fel cu vechiul `ask`.

`ask-concurrent` produce concurență simulată printr-un mecanism de turn-taking. Primul agent ia o tură, devine activ o tura, („takes a turn”), apoi al doilea agent devine activ și asa mai departe până când fiecare agent din agentset-ul cerut a fost activ, după care se revine la primul agent. Acest proces continuă pâna când toți agenții au terminat de rulat toate comenzile.

„Tura” unui agent se încheie în momentul în care acesta efectuează o acțiune care afectează starea lumii, cum ar fi mișcarea sau crearea unui agent turtle, sau schimbarea valorii unei variabile globale, turtle, patch sau link. (Setarea unei variabile locale nu se ia in calcul.)

Comenzile `forward(fd)` și `back(bk)` sunt tratate special. Atunci când sunt utilizate în interiorul `ask-concurrent`, aceste comenzi se pot executa în decursul mai multor ture. În timpul unei „ture”, agentul turtle se poate deplasa doar cu un singur pas. De exemplu, `fd 20` este echivalent cu `repeat 20 [fd 1]`, unde tura agentului turtle se termină dupa fiecare rulare a `fd`. Dacă distanța specificată nu este un număr întreg, ultima fracțiune de pas durează o tură completă. De exemplu `fd 20.3` este echivalent cu `repeat 20 [fd 1] fd 0.3`.

Comanda jump durează întotdeauna exact o singură tură, indiferent de distanță.

Pentru a înțelege diferența dintre `ask` și `ask-concurrent`, luăm în considerare următoarele două comenzi:

```
ask turtles [ fd 5 ]
ask-concurrent turtles [ fd 5 ]
```

Cu `ask`, primul agent turtle are nevoie de cinci pași înainte, apoi al doilea agent turtle are nevoie de cinci pași înainte și așa mai departe.

Cu `ask-concurrent`, toți agentii turtle iau un pas înainte. Apoi toți iau un al doilea pas și așa mai departe. Astfel, ultima comandă este echivalentă cu:

```
repeat 5 [ ask turtles [ fd 1 ] ]
```

Exemplu de cod: State Machine Example

Comportamentul lui `ask-concurrent` nu poate fi întotdeauna atât de simplu reprodus cu ajutorul ask. Considerăm următoarea comandă:

```
ask-concurrent turtles [ fd random 10 ]
```

Pentru a obține același comportament cu ajutorul `ask`, ar trebui să scriem:

```
turtles-own [steps]
ask turtles [ set steps random 10 ]
while [any? turtles with [steps > 0]] [
ask turtles with [steps > 0] [     fd 1
    set steps steps - 1
  ]
]
```

Pentru a prelungi „tura” unui agent vom utiliza comanda `without-interruption`. (Blocurile de comandă din interiorul altor comenzi cum ar fi `create-turtle` și `hatch`, au un `withoutinterruption` implicit în jurul lor.)

Comportamentul `ask-concurrent` este complet determinist. Având în vedere același cod și în aceleași condiții inițiale, același lucru se va întâmpla întotdeauna (dacă se utilizează aceeași versiune de NetLogo și se rulează modelul cu aceleași seed-uri).

În general, trebuie ca modelul să fie scris astfel încât acesta să nu depindă de particularitățile de funcționare ale `ask-concurrent`. Nu există garanția că semantica sa va ramâne la fel și în versiunile viitoare de NetLogo.

4.5 . „Tie” (conexiune)

„Tie” conectează doi agenți turtle, astfel încât deplasarea unui agent afectează poziția și orientarea celuilalt. „Tie” este o proprietate a link-urilor, astfel încât trebuie să existe un link între doi agenți turtle pentru a crea o relație „tie”.

Când `tie-mode`-ul unei legături este setat pe "fixed" sau "free" `end1` și `end2` sunt legate împreună. În cazul în care link-ul este orientat, `end1` este "agentul rădăcină" și `end2` este "agentul frunză". Aceasta înseamnă că atunci când `end1` se mută (folosind `fd`, `jump`, `setxy`, etc), `end2` se mută cu aceeași distanță și în aceeași direcție. Cu toate acestea, atunci când se mută `end2`, acesta nu afectează `end1`.

În cazul în care link-ul este neorientat, există o relație de reciprocitate, egalitate, ceea ce înseamnă că dacă oricare agent turtle se mută, se va deplasa și celălalt. În funcție de agentul turtle care se deplasează, fiecare agent poate fi considerat rădăcină sau frunză. Agentul rădăcină este întotdeauna agentul care inițiaza mișcarea.

Când agentul rădăcină vireaza spre dreapta sau stânga, agentul frunză se rotește în jurul acestuia cu aceeași distanță ca si cum un corp rigid a fost atașat de cei doi agenți. Când `tie-mode`-ul este setat pe "fixed" antetul agentului frunză se schimbă cu aceeași valoare. Dacă `tie-mode`-ul este setat pe "free" antetul agentului frunză rămâne neschimbat.

`Tie-mode`-ul unui link poate fi setat pe "fixed", folosind comanda `tie` și setând pe "none" (în sensul că agenții turtle nu mai sunt legați); pentru folosirea comenzii `untie` pentru setarea pe modul "free" trebuie utilizată comanda: `set tie-mode "free"`.

Exemplu de cod: Tie System Example

4.6 . Fișierele sursă multiple

Cuvântul cheie `__include` permite folosirea mai multor fișiere sursă într-un singur model NetLogo.

Cuvântul cheie începe cu două underscore-uri pentru a indica faptul că funcția este experimentală și se poate schimba în versiunile viitoare NetLogo.

Când se deschide un model care utilizează cuvantul cheie `__include`, sau dacă este adăugat la partea de sus a unui model și apoi este apăsat butonul de verificare, meniul include va apărea în bara de instrumente. Din meniul include se pot selecta fișierele incluse în model.

Când se deschid, fișierele incluse apar în tab-uri suplimentare. Consultați Ghidul de interfață (Interface Guide) pentru mai multe detalii.

În fișierele din sursa externă (.nls) poate exista orice element care poate fi adăugat în mod normal în tab-ul Code: `globals`, `breed`, `turtles-own`, `patches-own`, `breeds-own`, definiții de proceduri, etc. Aceste declarații împărtășesc același namespace, adică, dacă se va declara un global `my-global` în tab-ul Code, nu mai poate fi declarat înca un global (sau orice altceva) cu numele `my-`

`global` în nici un alt fișier inclus în model. `my-global` va fi accesibil din toate fișierele incluse. Același lucru este valabil dacă `my-global` este deja declarat în unul dintre fișierele incluse.

4.7 . sintaxa

Culori

În tab-ul Code și în orice alta parte în interfața cu utilizatorul NetLogo, codul programului este colorat astfel:

- Cuvintele cheie sunt verzi
- Constantele sunt portocalii
- Comentariile sunt gri
- Comenzile de primitive sunt albastre
- Raporturile de primitive sunt violet
- Orice altceva este negru

Cuvinte cheie

Singurele cuvinte cheie în acest limbaj sunt `globals`, `breed`, `turtles-own`, `patches-own`, `to`, `to-report` si `end` plus `extensions` și cuvântul cheie încă experimental `__include`. (Numele de primitive încapsulate nu pot fi umbrite sau redefinite, astfel încât acestea sunt efectiv un fel de cuvinte cheie)

Identificatori

Toate primitivele, numele de variabile globale, numele agenților și numele procedurilor împart un singur namespace, case-insensitive, la nivel global; numele locale (variabilele `let` și numele intrărilor procedurilor) pot să nu afecteze numele globale sau pe ele însele. Identificatorii pot conține orice caracter Unicode sau cifre și următoarele caractere ASCII:

```
.?=*!<>:#+/%$_^'&-
```

Unele nume de primitive încep cu două underscore-uri pentru a indica faptul că acestea sunt încă experimentale și foarte probabil se vor schimba sau vor fi eliminate în versiunile viitoare NetLogo.

Identificatorii care încep cu un semn de întrebare sunt rezervati.

Scope

Variabilele locale (inclusiv intrările procedurilor) sunt accesibile în blocul de comenzi în care sunt declarate, dar nu sunt accesibile în procedurile apelate de aceste comenzi.

Comentarii

Caracterul punct și virgulă introduce un comentariu, care ține până la sfârșitul liniei. Nu există nici o sintaxă pentru comentariu multi-linie.

Structura

Un program este format din declarațiile opționale (`globals`, `breed`, `turtles-own`, `patchesown`, `<BREED>-own`) în oricare ordine, urmat de zero sau mai multe definiții de proceduri. Rasele multiple pot fi declarate separat; alte declarații pot să apară o singură dată.
Fiecare definiție de procedură începe cu `to` sau `to-report`, numele procedurii și o listă de paranteze opțională cu nume de intrare. Fiecare definiție de procedură se termina cu `end`. Între acestea sunt zero sau mai multe comenzi.

Comenzi și reporteri

Comenzile au zero sau mai multe intrări, intrările sunt reporteri, care pot lua, de asemenea zero sau mai multe intrări. Comenzile nu sunt separate sau terminate prin punctuație; la fel, intrările nu sunt separate prin punctuație. Identificatorii trebuie să fie separați prin spațiu, paranteze rotunde sau paranteze pătrate. (De exemplu, `a + b` este un singur identificator, dar `a(b[c]d)e` contine cinci identificatori.)

Toate comenzile sunt prefixe. Toti repoarterii definiți de utilizator sunt prefixe. Reporterii-primitive sunt prefixe, dar unii (operatori aritmetici, operatori booleni și unii operatori agentset cum ar fi like și in-points), sunt infixe.

Toate comenzile și reporterii, atât primitivele cât și cele definite de utilizator, au un număr fix de intrări implicit. (Acesta este motivul pentru care limbajul poate fi analizat, deși nu există nici o punctuație care separă sau termină comenzile și/sau intrările.) Unele primitive pot avea un numar diferit de intrări decât cel implicit; parantezele sunt utilizate pentru a indica acest lucru, de exemplu, `(list 1 2 3)` (primitiva `list` are numai două intrări în mod implicit). Parantezele sunt de asemenea folosite pentru a suprascrie implicit operatorul de prioritate, de exemplu, `(1 + 2) * 3`, ca și în alte limbaje de programare.

Uneori, intrarea unei primitive este un bloc de comandă (zero sau mai multe comenzi în interiorul parantezelor pătrate) sau un bloc de reporter (o singură expresie de reporter în interiorul parantezelor pătrate). Procedurile definite de utilizator nu pot avea un bloc de comandă sau un bloc de reporter ca intrare.

Prioritatea operatorilor, de la cel mai important la cel mai puțin important:

- with, at-points, in-radius, in-cone
- (toate celelalte primitive și proceduri definite de utilizator)
- ^

- *, /, mod
- +, -
- <, >, <=, >=
- =, !=
- and, or, xor

Comparația cu alte Logo-uri

Nu există nici o definiție standard a logo-ului, este o familie liberă de limbaje. Noi credem că NetLogo are suficiente lucruri în comun cu alte logo-uri pentru a-și câștiga numele de logo. Totuși, NetLogo diferă în unele privințe de cele mai multe alte Logo-uri. Cele mai importante diferențe sunt după cum urmează.

Diferențele de la suprafață

- Prioritatea operatorilor matematici este diferită. Operatorii matematici infix (cum ar fi +, *, etc) au prioritate mai mică decât reporterii cu nume. De exemplu, în multe Logo-uri, dacă scriem `sin x + 1`, va fi interpretat ca `sin (x + 1)`. NetLogo interpretează așa cum cele mai multe alte limbaje de programare ar face-o, totuși aceeași expresie ar putea fi interpretată în notația matematică standard, și anume `(sin x) + 1`.
- Reporterii `and` și `or` sunt forme speciale, nu sunt funcții obișnuite, și "scurtcircuitează", adică, evaluează doar două intrări, dacă este necesar.
- Procedurile pot fi definite doar în fila Code, nu interactiv în Centrul de comandă.
- Procedurile reporter, care sunt, proceduri ce returnează o valoare, trebuie să fie definite cu `to-report` în loc de `to`. Comanda pentru a raporta o valoare de la o procedură de raport este `report`, nu `output`.
- La definirea unei proceduri, intrările procedurilor trebuie să fie încadrate în paranteze pătrate, de exemplu `to square [x]`.
- Numele variabilelor sunt utilizate întotdeauna, fără semne de punctuație: întotdeauna `foo`, nu `:foo` sau `"foo`. (Pentru a face acest lucru, în loc de comanda `make` care are un argument între ghilimele vom oferi un formular special `set` care nu se evaluează la prima intrare.) Prin urmare, procedurile și variabilele ocupă un singur namespace comun.

Ultimele trei diferențe sunt ilustrate în următoarele definiții de proceduri:

cele mai multe Logo-uri	**NetLogo**

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

Diferențe în adâncime

- Variabilele și intrările procedurilor locale NetLogo sunt analizate lexical, nu dinamic.

- NetLogo nu are nici un tip de date "cuvânt" (ceea ce Lisp numeste "simboluri"). În cele din urmă, s-ar putea adăuga unul, dar din moment ce este rar solicitat, se poate ca nevoia să nu decurgă prea mult în modelarea bazată pe agenți. Avem însă șiruri de caractere. În cele mai multe situatii în care Logo-ul tradițional ar folosi cuvinte, vom folosi pur și simplu șiruri de caractere. De exemplu, în Logo am putea scrie `[ see spot run ]` (o listă de cuvinte), dar în NetLogo trebuie să scriem `"see spot run"` (un șir de caractere) sau `["see" "spot" "run"]` (o listă de șiruri de caractere).
- Comanda `run` din NetLogo lucrează pe sarcini și șiruri de caractere, nu liste (deoarece nu avem nici un tip de date "cuvânt"), și nu permite definirea sau redefinirea procedurilor.
- Structurile de control, cum ar fi `if` și `while` sunt forme speciale, nu funcții obișnuite. Nu se pot defini propriile forme speciale, astfel încât nu se pot defini structuri de control proprii. (Se poate face ceva similar folosind task-uri, dar trebuiesc utilizate primitivele `task`, `run` și `runresult`, nu există posibilitatea să fie implicite.)
- Task-urile (valorile funcției sau lambda) sunt adevărate închideri analizate lexical. Această caracteristică este disponibilă în NetLogo si în lisps-urile moderne, dar nu în logo-ul standard.

Desigur, limbajul NetLogo conține alte caracteristici care nu se găsesc în cele mai multe Logouri, cele mai importante fiind agenții și agentset-urile.