# Průvodce programováním

Následující kapitola vysvětlí důležité vlastnosti programování v NetLogu.

Zmiňované ukázky kódu programu modelů jsou k nalezení v sekci **Code Examples** v **Models Library**.


- Agenti
- Procedury
- Proměnné
- Barvy
- Příkaz ask
- Množiny agentů
- Rody
- Tlačítka
- Seznamy
- Aritmetika
- Náhodná čísla
- Tvary želv
- Tvary spojů
- Počítadlo kroků
- Aktualizace zobrazení
- Vykreslení grafů
- Řetězce
- Výstup
- Soubory
- Videa
- Perspektiva
- Kreslicí vrstva
- Topologie
- Spoje
- Příkaz ask-concurrent
- Vazba
- Více zdrojových souborů
- Syntax

# Agenti

Svět NetLoga je tvořen agenty. Agenti jsou tvorové, kteří se řídí instrukcemi. Každý agent provádí vlastní činnost zároveň s ostatními.

V NetLogu existují čtyři druhy agentů: želvy, políčka, spoje a pozorovatel. Želvy jsou agenti pohybující se po světě. Svět je dvourozměrný a je rozdělen do mřížky s políčky. Každé políčko je čtverec „země“, po kterém se pohybují želvy. Spoje jsou agenti spojující dvě želvy. Pozorovatel není umístěn nikde, představte si ho jako pozorovatele, který se dívá na svět želv a políček.

Na začátku po startu NetLoga ještě neexistují želvy. Nové želvy nebo políčka může vytvořit pozorovatel. (Políčka se nepohybují, ale jinak jsou stejně „naživu“ jako želvy a pozorovatel.)

Políčka mají souřadnice. Políčko se souřadnicemi (0,0) se nazývá výchozí bod a souřadnice ostatních políček vyjadřují horizontální a vertikální vzdálenost od tohoto bodu. Souřadnice políček se nazývají `pxcor` a `pycor`. Stejně jako v matematické kartézské soustavě souřadnic se `pxcor` zvyšuje směrem doprava a `pycor` směrem nahoru.

Celkový počet políček je dán nastavením hodnot `min-pxcor`, `max-pxcor`, `min-pycor` a `max-pycor`. Na začátku je nastavení následující: –16, 16, –16 a 16 v tomto pořadí. To znamená, že hodnoty obou souřadnic `pxcor` a `pycor` jsou od –16 do 16, tj. 33 x 33 políček, celkem 1 089. (Počet políček lze změnit pomocí tlačítka **Settings**.)

Želvy mají taktéž souřadnice: `xcor` a `ycor`. Souřadnice políčka jsou vždy celá čísla, souřadnice želv mohou být v desetinných číslech. Želva může být umístěna kdekoliv v políčku, nikoliv pouze ve středu.

Spoje souřadnice nemají, místo toho jsou určeny dvěma konečnými body (želvami). Spoje tyto dva body spojují nejkratší možnou cestou, i když to někdy znamená přejít přes jeden okraj světa na druhý.

Způsob, jakým je svět políček propojen, lze změnit. Výchozí nastavení je torus, tzn. že svět není ohraničený, ale zacyklený – pokud tedy želva přejde okraj světa, zmizí a objeví se na opačné straně. Každé políčko má stejný počet sousedících políček; pokud jste na políčku na okraji světa, některá sousední políčka se nacházejí na opačném okraji. Pomocí tlačítka **Settings** můžete cyklení světa (wrapping) vypnout. Když není cyklení v určitém směru povoleno, znamená to, že je svět v daném směru (x nebo y) ohraničený. Políčka podél hranice mají méně než 8 sousedících políček a želvy se nedostanou přes okraj světa. Více informací naleznete v podkapitole Topologie.

# Procedury

V NetLogu říkají příkazy a reportéry agentům, co mají dělat. **Příkaz** je akce, kterou má agent vykonat. **Reportér** vypočte výsledek a vrátí ho.

Většina příkazů začíná slovesem (`create`, `die`, `jump`, `inspect`, `clear`), zatímco většina reportérů jsou podstatná jména nebo jmenné fráze.

Příkazy a reportéry zabudované v NetLogu se nazývají **primitiva**. Kompletní seznam těchto příkazů a reportérů najdete ve Slovníčku NetLoga.

Příkazy a reportéry, které si sami definujete, se nazývají **procedury**. Každá procedura má jméno, jemuž předchází klíčové slovo `to`. Klíčové slovo `end` značí konec příkazů v proceduře. Jakmile jednou nadefinujete proceduru, můžete ji používat kdekoliv v programu.

Mnoho příkazů a reportérů vyžaduje **vstupy** – hodnoty, jež daný příkaz či reportér použije pro vykonání akce.

Příklady: Následují dvě příkazové procedury:

```
to setup
  clear-all      ;; clear the world
  crt 10         ;; make 10 new turtles
end

to go
  ask turtles
    [ fd 1             ;; all turtles move forward one step
      rt random 10     ;; ...and turn a random amount
      lt random 10 ]
end
```

Všimněte si, že do programu lze přidat komentáře oddělené středníky. Tyto komentáře usnadňují pochopení kódu.

V naší ukázce programu jsou použity následující prvky:

- `setup` a `go` jsou příkazy definované uživatelem.
- `clear-all`, `crt` (create turtles, vytvoř želvy), `ask`, `lt` (left turn, zatoč doleva) a `rt` (right turn, zatoč doprava) jsou primitivní příkazy.
- `random` a `turtles` jsou primitivní reportéry. `random` vezme ze vstupu celé číslo a vrátí náhodné celé číslo menší než vstup (v tomto případě mezi 0 a 9). `turtles` vrátí množinu agentů sestávajících ze všech želv. (Množiny agentů si vysvětlíme později.)

`setup` a `go` mohou být volány jinými procedurami. Hodně modelů NetLoga mají jednorázové tlačíko volající proceduru PŘIPRAV (setup) a trvalé tlačítko volající proceduru START (go).

V NetLogu musíte určit, kteří agenti – želvy, políčka, spoje či pozorovatel – mají příkaz vykonat. (Pokud tak neučiníte, vykoná kód pozorovatel.) Ve výše uvedeném kódu používá pozorovatel příkaz `ask`, aby všechny želvy vykonaly příkazy v hranatých závorkách.

`clear-all` a `crt` může vykonat jen pozorovatel. Naopak `fd` (forward, vpřed) mohou vykonat jen želvy. Jiné příkazy a reportéry, např. `set`, mohou vykonat různé druhy agentů.

Následují pokročilejší funkce a vlastnosti, které při definování vlastních procedur můžete využít.

## Procedury se vstupy

Stejně jako primitiva, mohou mít i vaše vlastní procedury vstupy. Proceduru se vstupem vytvoříte tak, že za jejím jménem uvedete do hranatých závorek seznam názvů vstupu. Například:

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

Jinde v programu můžete želvy požádat, aby každá nakreslila osmiúhelník o délce strany rovnající se identifikačnímu číslu:

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

## Procedury s reportérem

Stejným způsobem můžete definovat i reportéry. Postup se liší ve dvou bodech. Za prvé, místo `to` na začátku procedury použijte `to-report`. Dále v těle procedury použijte `report`, jímž vrátíte danou hodnotu.

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

# Proměnné

Proměnné jsou úložiště hodnot (například čísel). Proměnná může být globální, proměnná želvy nebo proměnná políčka.

Pokud je proměnná globální, má pouze jednu hodnotu a má k ní přístup každý agent. Naopak každá želva má svou vlastní hodnotu proměnné želvy a každé políčko vlastní hodnotu proměnné políčka.

Některé proměnné jsou do NetLoga vestavěné. Například všechny želvy mají proměnnou `color` a všechna políčka `pcolor`. (Proměnná políčka začíná na „p“, aby nebyla zaměněna s proměnnou želvy.) Pokud zadáte hodnotu, želva či políčko změní barvu. (Viz příští podkapitola.)

Další zabudované proměnné jsou `xcor`, `ycor` a `heading`. (Kompletní seznam naleznete zde.)

Můžete si definovat své vlastní proměnné. Globální proměnnou vytvoříte tak, že do modelu přidáte tlačítko nebo posuvník nebo na začátku kódu použijte klíčové slovo `globals`:

```
globals [ score ]
```

Nové proměnné želv, políček či spojů můžete také určit pomocí klíčových slov `turtles-own`, `patches-own` a `links-own`, např.:

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

Tyto proměnné pak můžete kdykoliv v modelu použít. Nastavíte je pomocí příkazu `set`. (Pokud je nenastavíte, budou mít jako výchozí hodnotu nulu.)

Globální proměnné mohou být načítány a měněny kdykoliv jakýmkoliv agentem. Želva také může načíst a změnit proměnnou políčka, na kterém stojí. Například tento kód

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

způsobí, že každá želva změní barvu políčka, na němž stojí, na červenou. (Protože jsou proměnné políčka sdíleny se želvou, musí mít jiný název než proměnné želvy.)

Chcete-li i v jiných případech po agentovi, aby přečetl proměnnou jiného agenta, použijte `of`. Příklad:

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

`of` můžete použít i v komplikovanějším výrazu, než je jméno proměnné, například:

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

### Lokální proměnné

Lokální proměnná je definována a použita pouze v rámci konkrétní procedury nebo její části. Lokální proměnnou vytvoříme pomocí příkazu `let`. Tento příkaz lze použít kdekoliv; pokud ho umístíte na začátek procedury, bude proměnná existovat v celém jejím průběhu. Použijete-li příkaz uvnitř hranatých závorek, např. v `ask`, bude existovat pouze v těchto závorkách.

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

# Barvy

NetLogo znázorňuje barvy dvěma způsoby. V prvním mají barvy číslo od 0 do 140, kromě samotné 140. Níže je uvedená tabulka s paletou barev, které v NetLogu můžete používat.

| | black = 0 | | | | | | | | | | white = 9.9 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| gray = 5 | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 9.9 |
| red = 15 | 10 | 11 | 12 | 13 | 14 | 15 | 16 | 17 | 18 | 19 | 19.9 |
| orange = 25 | 20 | 21 | 22 | 23 | 24 | 25 | 26 | 27 | 28 | 29 | 29.9 |
| brown = 35 | 30 | 31 | 32 | 33 | 34 | 35 | 36 | 37 | 38 | 39 | 39.9 |
| yellow = 45 | 40 | 41 | 42 | 43 | 44 | 45 | 46 | 47 | 48 | 49 | 49.9 |
| green = 55 | 50 | 51 | 52 | 53 | 54 | 55 | 56 | 57 | 58 | 59 | 59.9 |
| lime = 65 | 60 | 61 | 62 | 63 | 64 | 65 | 66 | 67 | 68 | 69 | 69.9 |
| turquoise = 75 | 70 | 71 | 72 | 73 | 74 | 75 | 76 | 77 | 78 | 79 | 79.9 |
| cyan = 85 | 80 | 81 | 82 | 83 | 84 | 85 | 86 | 87 | 88 | 89 | 89.9 |
| sky = 95 | 90 | 91 | 92 | 93 | 94 | 95 | 96 | 97 | 98 | 99 | 99.9 |
| blue = 105 | 100 | 101 | 102 | 103 | 104 | 105 | 106 | 107 | 108 | 109 | 109.9 |
| violet = 115 | 110 | 111 | 112 | 113 | 114 | 115 | 116 | 117 | 118 | 119 | 119.9 |
| magenta = 125 | 120 | 121 | 122 | 123 | 124 | 125 | 126 | 127 | 128 | 129 | 129.9 |
| pink = 135 | 130 | 131 | 132 | 133 | 134 | 135 | 136 | 137 | 138 | 139 | 139.9 |

Tabulka nám ukazuje, že

- některé barvy mají názvy (tyto názvy lze použít v kódu programu);
- každá pojmenovaná barva, až na černou a bílou, končí na číslici 5;
- po každé straně pojmenované barvy jsou její tmavší a světlejší odstíny;
- 0 je čistá černá; 9.9 je čistá bílá;
- 10, 20 atd. jsou tak tmavé, že vypadají jako černá;
- 19.9, 29.9 atd. jsou tak světlé, že vypadají jako bílá.

**Ukázka kódu programu**: Tabulka barev byla vytvořena v NetLogu pomocí modelu Příklad palety barev (Color Chart Example).

Použijete-li číslo mimo rozsah 0 až 140, NetLogo bude přičítat nebo odečítat 140 tak dlouho, až se bude číslo pohybovat v rozmezí 0 až 140. Např. 25 je oranžová, takže 165, 305, 445 atd. jsou také oranžová, stejně tak i –115, –255, –395 atd. Tato kalkulace se provede automaticky, kdykoliv nastavíte proměnnou želvy `color` nebo proměnnou políčka `pcolor`. Pokud byste potřebovali tuto kalkulaci provést v jiném kontextu, použijte primitivum `wrap-color`.

Jestliže chcete použít barvu, která není v tabulce, najdete ji mezi celými čísly. Například 26.5 je odstín oranžové mezi 26 a 27. To však neznamená, že v NetLogu můžete vytvořit jakoukoliv barvu; paleta barev je limitovaná. Obsahuje pouze fixní množinu diskrétních barev (jedna barva na řádek). Když zvolíte barvu, můžete buď snížit její jas (ztmavit ji) nebo saturaci (zesvětlit ji), ale nelze snížit jas i saturaci zároveň. Podstatné je pouze první desetinné místo; hodnota barvy je zaokrouhlena dolů na desetiny, např. neexistuje viditelný rozdíl mezi 26.5 a 26.52 nebo 26.58.

## Primitiva barev

Pro práci s barvami se nám bude hodit několik primitiv.

Už jsme zmínili primitivum `wrap-color`.

Primitivum `scale-color` se hodí při převádění číselných dat na barvy.

`shade-of?` vám řekne, jestli jsou dvě barvy dvěma odstíny stejné základní barvy. Například `shade-of? orange 27` je pravdivý, protože 27 je světlejší odstín oranžové.

**Ukázka kódu programu:** Příklad škály barev (Scale-color Example) ukazuje funkci zobrazovacího reportéru.

## RGB barvy

Druhým způsobem zobrazení barev v NetLogu je seznam barev RGB (red/green/blue). V režimu RGB je vám k dispozici celá paleta barev. Seznamy RGB sestávají ze tří celých čísel mezi 0 a 255; pokud číslo přesahuje 255, odečítá se 255 tak dlouho, až se číslo bude pohybovat v daném rozmezí. Do seznamu RGB můžete přidat jakoukoliv barvu (`color` pro želvy a spoje a `pcolor` pro políčka) a daný agent bude odpovídajícím způsobem zobrazen. Barvu políčka na čistě červenou nastavíte takto:

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

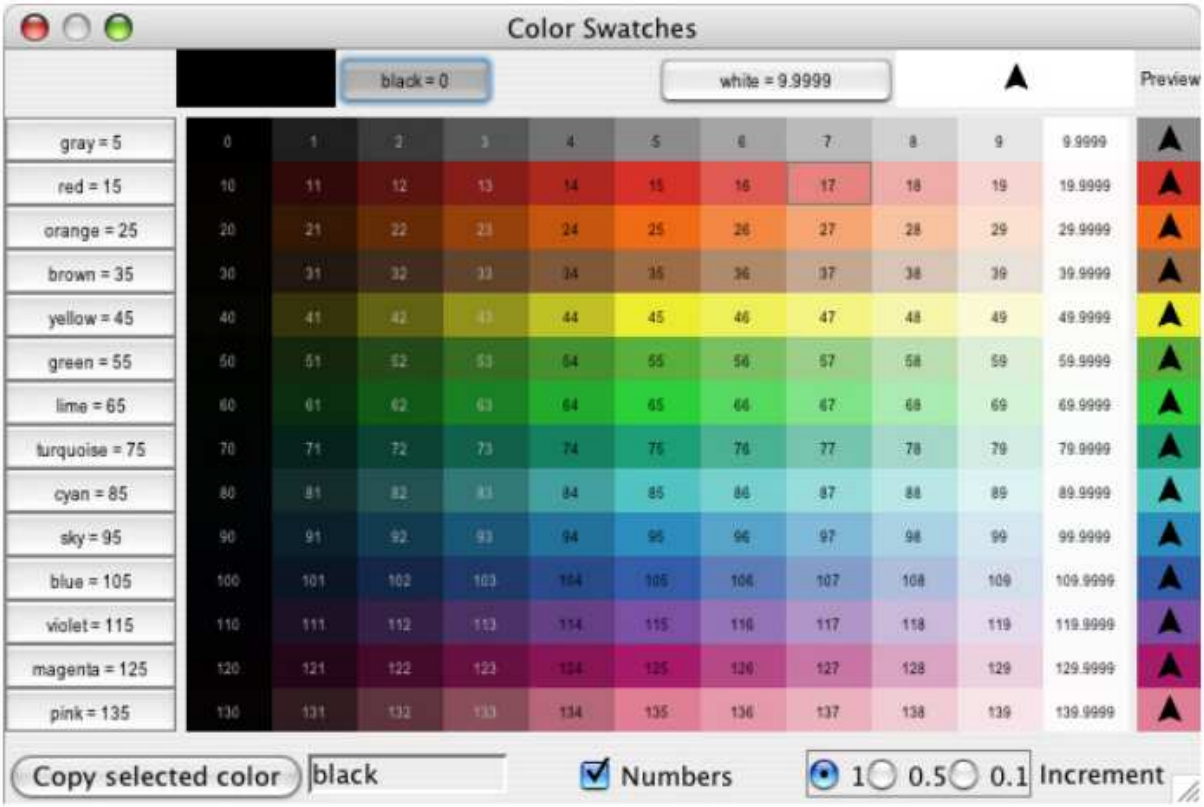
Mezi RGB, HSB (hue/saturation/brightness) a barvami NetLoga lze přepínat: z RGB/HSB do barev NetLoga pomocí příkazů `approximate-hsb` a `approximate-rgb`. `extract-hsb` a `extract-rgb`. `rgb` se používá k vytvoření rgb seznamů a `hsb` k převedení barvy HSB na RGB.

V NetLogu však mnoho barev chybí, a proto ne vždy `approximate-hsb` a `approximate-rgb` zobrazí správnou barvu, v tom případě zvolí nejbližší odpovídající.

**Ukázka kódu programu**: Model Příklad HSB a RGB (HSB and RGB Example) vám umožňuje experimentovat se systémy barev HSB a RGB.

## Paleta barev

Barvy můžete měnit a zkoušet pomocí palety barev. Okno otevřete v menu **Tools**, položka **Color Swatches**.




Když kliknete na paletu barev (nebo barevné tlačítko), daná barva se zvýrazní. V levém dolním rohu je zobrazen kód současné barvy (například `red + 2`), takže ho můžete zkopírovat a vložit do kódu programu. V pravém dolním rohu jsou tři volby – 1, 0,5 a 0,1. Tato čísla vyjadřují rozdíl mezi dvěma sousedními barevnými políčky. Když je rozdíl 1, má každý řádek 10 různých odstínů; když je rozdíl 0,1, má jich 100. 0,5 je prostřední nastavení.

# Příkaz ask

NetLogo používá příkaz `ask` pro želvy, políčka a spoje. Veškerý kód, který mají provést želvy, musí být v jejich „kontextu“. Kontext želv nastavíte třemi způsoby:

- u tlačítka vyberte z pop-up menu `turtles` (želvy). Jakýkoliv kód, který do tlačítka napíšete, bude určen pro všechny želvy;
- v příkazovém panelu vyberte z pop-up menu `turtles` (želvy). Jakýkoliv příkaz zadáte, bude určen pro všechny želvy;
- pomocí příkazu `ask turtles`

Stejně postupujeme i u políček, spojů a pozorovatele až na to, že u pozorovatele nelze použít příkaz `ask`. Jakýkoliv kód, který není uvnitř `ask`, je automaticky kód pro pozorovatele.

Následuje příklad použití `ask` v proceduře NetLoga:

```
to setup
  clear-all
  crt 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
end
```

Modely v knihovně obsahují další příklady, začněte ukázkami kódu programu.

`ask` používá obvykle pozorovatel, aby zadal všem želvám, políčkům nebo spojům, že mají vykonat příkaz. `ask` lze také použít pro příkaz určený jednotlivé želvě, políčku nebo spoji, v tomto případě použijeme reportéry `turtle`, `patch`, `link` a `patch-at`. Například:

```
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
end
```

Každá želva má identifikační číslo. První želva má číslo 0, druhá 1 atd. Identifikační číslo je vstupem pro primitivní reportér `turtle`, který vrací odkaz na identifikovanou želvu. Primitivní reportér `patch` má za vstup hodnoty pxcor a pycor a vrací políčko s těmito souřadnicemi. Primitivum `link` má dva vstupy – dvě identifikační čísla želv, které se mají spojit. Primitivní reportér `patch-at` má dva *posuny* – vzdálenosti na ose X a Y *od* prvního agenta. Ve výše uvedeném příkladu přikážete želvě s identifikačním číslem 0, aby zabarvila políčko na východ od sebe (ne na sever).

Můžete také vybrat podmnožinu želv, podmnožinu políček nebo podmnožinu spojů a něco jim přikázat. Tato problematika zahrnuje tzv. „množiny agentů“, které si dále vysvětlíme.

Když přikážete množině agentů, aby provedli více než jeden příkaz, každý jednotlivý agent je musí dokončit a pak je provádí další. Jeden agent provede všechny příkazy, pak je provede další a další atd. Napíšete-li například:

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

posune se jedna želva a zčervená, pak se posune a zčervená další želva atd.

Pokud ale příkazy napíšete tímto způsobem:

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

tak se nejdřív všechny želvy pohnou a teprve poté všechny zčervenají.

(Ještě je možno použít další podobu příkazu `ask` s rozdílným způsobem uspořádávání, viz podkapitola Příkaz ask-concurrent níže.)

# Množiny agentů

Množina agentů je přesně to, co říká název – skupinou agentů, která může obsahovat želvy, políčka či spoje, ale vždy pouze jeden druh v rámci jedné množiny.

Každá množina agentů má *vlastní* náhodné uspořádání. Při každém přístupu se navíc toto náhodné uspořádání mění. Díky tomu model nezachází s konkrétními želvami, políčky či spoji jinak než s ostatními (pokud tak sami neurčíte). Jelikož je pořadí vždy náhodné, nezačíná se pokaždé stejným agentem.

Už jste se seznámili s primitivy želv určenými množině všech želv, s primitivy políček určenými množině všech políček a s primitivy spojů určenými množině všech spojů.

Výhoda koncepce množin agentů však spočívá v tom, že můžete vytvářet množiny, jež obsahují pouze některé želvy, některá políčka či některé spoje. Například můžete vybrat všechny červené želvy, všechna políčka se souřadnicí pxcor dělitelnou pěti, želvy nacházející se v prvním kvadrantu a stojící na zelených políčkách či spoje spojující želvu 0. Těmto množinám agentů můžeme něco přikázat pomocí `ask` nebo mohou sloužit jako vstupy pro různé reportéry.

Chcete-li vytvořit množinu agentů obsahující pouze želvy na stejném či jakémkoliv jiném políčku na souřadnicích X a Y, můžete použít `turtles-here` nebo `turtles-at`. Příkazem `turtles-on` dostanete množinu želv stojících na daném políčku/daných políčkách nebo množinu želv stojících na stejném políčku jako daná želva nebo množina želv.

Následují další příklady použití množin agentů:

```
;; 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
```

Všimněte si použití `other`, čímž vyloučíte volajícího agenta.

S vytvořenou množinou agentů můžete provést například tyto jednoduché akce:

- říci agentům `ask`, aby vykonaly nějakou akci;
- pomocí `any?` zjistit, jestli množina agentů není prázdná;
- pomocí `all?` zjistit, zda všichni agenti splňují zadanou podmínku;
- pomocí `count` zjistit, kolik agentů množina přesně obsahuje.

Provést lze i následující složitější úkony:

- pomocí `one-of` vybrat z množiny jednoho náhodného agenta a potom náhodně vybranou želvu např. obarvit na zeleno:

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

  nebo říci náhodně vybranému políčku, aby vysadilo želvu pomocí příkazu `sprout`:

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

- pomocí reportérů `max-one-of` či `min-one-of` zjistit, který agent se nachází na vrcholu a který na spodu nějaké stupnice. Chcete-li např. odstranit nejbohatší želvu, napište:

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

- pomocí příkazu `histogram` (v kombinaci s `of`) vytvořit histogram množiny agentů;
- pomocí `of` vytvořit seznam hodnot, jednu pro každého agenta v množině agentů. Potom si vybrat jedno z primitiv seznamů v NetLogu a se seznamem provést nějakou akci. (Viz podkapitola Seznamy níže.) Chcete-li například zjistit, jak bohaté želvy v průměru jsou, použijte:

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

- pomocí reportérů `turtle-set`, `patch-set` a `link-set` vytvořit novou množinu agentů shromážděním z mnoha různých zdrojů;
- porovnat dvě množiny agentů pomocí `=` nebo `!=.`
- pomocí `member?` zjistit, zda konkrétní agent náleží do nějaké množiny agentů.

Výše uvedené situace jen nastiňují možnosti. Více příkladů naleznete v knihovně modelů a více informací o primitivech množin agentů ve Slovníčku NetLoga.

Další příklady použití množin agentů se nacházejí u jednotlivých vstupů primitiv ve Slovníčku NetLoga. Chcete-li poznat NetLogo, musíte začít přemýšlet, jak ve složených příkazech předává jednotlivý prvek informaci následujícímu. Součástí tohoto konceptu jsou i množiny agentů, které jsou pro vývoj dostatečně silné a pružné a více připomínají přirozený jazyk.

**Ukázka kódu programu:** Příklad uspořádávání při příkazu `ask` (Ask Ordering Example)

Dříve jsme si řekli, že množiny agentů jsou uspořádány vždy v nahodilém pořadí, a to pokaždé jiném. Pokud chcete, aby pořadí agentů bylo pevné, musíte vytvořit jejich seznam. Více viz podkapitola Seznamy níže.

# Rody

NetLogo vám umožňuje definovat rozdílné „rody“ želv a spojů. Když nadefinujete rody, můžete jim zadat, aby se chovaly rozdílně. Například máme rody nazvané `ovce` a `vlci`, přičemž vlci se snaží sníst ovce. Nebo můžeme rozlišit i spoje, jeden rod budou `silnice` a druhý `chodníky` a pěší se budou pohybovat po chodnících a auta po silnicích.

Rody želv definujete pomocí klíčového slova `breed` (rod) v horní části panelu **Procedures** ještě před procedurami:

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

Ke členu rodu želv odkazujeme jednotným číslem, stejně jako u reportérů želv. Při vypisování budou mít členové popisku rodu v jednotném čísle.

Některé příkazy a reportéry mají název rodu želv v množném čísle, např. `create-<breeds>`. Ostatní mají číslo jednotné, například `<breed>`.

Pořadí, v kterém jsou rody deklarovány, zároveň určuje pořadí, v kterém jsou vrstveny při zobrazení. Rody definované později se budou nacházet na rodech určených dříve, v našem příkladě budou ovce nakresleny přes vlky.

Když definujete rod jako např. ovce, zároveň se automaticky vytvoří množina agentů rodu, takže všechny výše popsané vlastnosti množiny agentů platí pro množinu ovcí.

Po vytvoření rodu se také nabízejí následující nová primitiva: `create-sheep`, `hatch-sheep`, `sprout-sheep`, `sheep-here`, `sheep-at`, `sheep-on` a `is-a-sheep?`.

K definování nových proměnných, které mají jen želvy určitého rodu, použijte příkaz `sheep-own`.

Množina rodu želv je uložena v proměnné rodu želvy. Můžete rod vyzkoušet například takto:

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

Všimněte si, že želvy mohou rody měnit. Vlk nemusí celý svůj život zůstat vlkem. Změňme náhodného vlka v ovci:

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

Primitivum `set-default-shape` používáme, když chceme přiřadit určité tvary želv k určitým rodům. Více viz podkapitola Tvary níže.

Následuje stručný příklad použití rodů.

```
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 ]
end
```

**Ukázka kódu programu:** Příklad rodů a tvarů (Breeds and Shapes Example)

## Rody spojů

Rody spojů jsou velmi podobné rodům želv, avšak existuje několik rozdílů.

Když deklarujete rod spojů, musíte zároveň deklarovat, zda je to rod orientovaných nebo neorientovaných spojů pomocí klíčových slov `directed-link-breed` a `undirected-link-breed`:

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

Když už jednou vytvoříte spoj patřící do nějakého rodu, nemůžete vytvořit bezrodý spoj a obráceně. (V jednom světě můžete mít orientované i neorientované spoje, ale nesmějí se vyskytovat v rámci jednoho rodu.)

Na rozdíl od rodu želv, pro rod spojů vždy používáme jednotné číslo, protože mnoho reportérů a příkazů pro práci s nimi používají singulár, např. `<link-breed>-neighbor?`.

Po vytvoření rodu orientovaných spojů se automaticky nabízejí následující primitiva: `create-street-from`, `create-streets-from`, `create-street-to`, `create-streets-to`, `in-street-neighbor?`, `in-street-neighbors`, `in-street-from`, `my-in-streets`, `my-out-streets`, `out-street-neighbor?`, `out-street-neighbors`, `out-street-to`.

Tato primitiva jsou k dispozici po vytvoření rodu neorientovaných spojů: `create-friendship-with`, `create-friendships-with`, `friendship-neighbor?`, `friendship-neighbors`, `friendship-with`, `my friendships`.

Stejně jako u rodů želv, tak i pořadí, v jakém jsou spoje nakresleny, závisí na pořadí, kdy byly rody deklarovány, takže např. přátelství bude vždy umístěno nad ulicemi (kdyby se z nějakého důvodu vyskytly tyto dva rody ve stejném modelu). Proměnné každého rodu spojů můžete také deklarovat zvlášť pomocí `<link-breeds>-own`.

Můžete měnit rod spojů, stejně jako v případě želv, avšak nelze nechat spoj bezrodý, protože nelze mít oba druhy spojů ve stejném světě.

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

Pomocí `set-default-shape` lze také přiřadit rodům spojů určité tvary.

**Ukázka kódu programu:** Příklad rodů spojů (Link Breeds Example)

# Tlačítka

Tlačítka umístěná na panelu **Interface** představují jednoduchý způsob ovládání modelu. Standardně má model minimálně tlačítko PŘIPRAV, sloužící k počátečnímu nastavení světa, a START, kterým se model spustí. Některé modely budou mít další tlačítka na spuštění dalších akcí.

Tlačítko obsahuje kód NetLoga. Kód se stisknutím tlačítka spustí.

Tlačítko může být jednorázové nebo může spustit akci trvalou. Lze ho nastavit pomocí zaškrtávacího rámečku **Forever**. Jednorázové tlačítko spustí kód jednou, pak se zastaví a vyskočí nazpátek. Naopak trvalá tlačítka spouští kód stále dokola, dokud není zastaven příkazem `stop` nebo opětovným stisknutím tlačítka. Když zvolíte stisknutí tlačítka, kód se nepřeruší, ale doběhne, a pak teprve tlačítko znovu vyskočí.

Obvykle je tlačítko označeno kódem, který spouští. Např. tlačítko START spouští kód „start“ (go), tj. „spustit proceduru START“. (Procedury jsou definovány v panelu **Procedures**, viz níže.) Tlačítko můžete upravit a vložit pomocí **display name** (zobrazit název), což je text na tlačítku místo kódu. Toto se hodí, pokud si myslíte, že označení kódem by pro uživatele mohlo být matoucí.

Když má tlačítko spouštět kód, musíme také určit, kteří agenti ho mají provést, tj. zdali pozorovatel, všechny želvy, všechna políčka nebo všechny spoje. (Pokud chcete, aby kód provedly jen některé želvy nebo některá políčka, vytvořte tlačítko pozorovatele a potom použijte příkaz `ask`, kterým nařídíte pouze některým želvám, aby akci vykonaly.)

Když upravujete tlačítko, můžete mu připsat klávesovou zkratku. Stisknutí dané klávesy/kombinace kláves na klávesnici vyvolá stejnou akci jako stisknutí tlačítka. Pokud tlačítko spouští trvalou akci, zůstane potlačeno, dokud znovu nestisknete klávesu (či nekliknete na tlačítko). Klávesové zkratky jsou vhodné zejména pro hry nebo modely vyžadující rychlé spouštění tlačítek.

## Střídání tlačítek

Najednou může být stisknuto více tlačítek. V tomto případě se tlačítka „střídají“, tj. v jeden okamžik běží pouze jedno. Každé tlačítko projde svůj kód až do konce, zatímco ostatní čekají, pak přijde na řadu další atd.

V následujících případech máme jednorázové tlačítko PŘIPRAV a trvalé tlačítko START.

Příklad 1: Uživatel stiskne PŘIPRAV a hned na to START, dříve než tlačítko PŘIPRAV znovu vyskočilo. Výsledek: Procedura PŘIPRAV je dokončena dřív, než začne START.

Příklad 2: Zatímco je spuštěno tlačítko START, stiskne uživatel PŘIPRAV. Výsledek: Tlačítko START dokončí stávající iteraci a pak se spustí procedura PŘIPRAV. Potom poběží znovu procedura START.

Příklad 3: Uživatel má stisknuta dvě trvalá tlačítka. Výsledek: První tlačítko spustí svůj kód, projde ho až do konce, pak druhé tlačítko projde svůj kód až do konce a dále se střídají.

Pozor, pokud se nějaké tlačítko zasekne v nekonečné smyčce, tak další tlačítko již nepoběží.

### Trvalá tlačítka želv, políček a spojů

Vkládání příkazů přímo do trvalého tlačítka želvy, políčka či spoje se trochu liší od stejných příkazů v tlačítku pozorovatele, který provádí `ask turtles`, `ask patches` nebo `ask links`. `ask` není dokončeno, dokud všichni agenti neprojdou příkazy v něm. Agenti vykonávají příkazy současně a nemusejí být navzájem synchronizovaní, nicméně na konci `ask` se sesynchronizují. Toto neplatí pro trvalá tlačítka želv, políček a spojů – každá želva či políčko vykonávají kód pořád dokola, takže přestanou být navzájem synchronní.

Tato vlastnost je v současných modelech v knihovně modelů využívána jen zřídka, používá ji např. model Termiti (Termites) v sekci **Biology** v **Sample Models**. Tlačítko START je trvalé tlačítko želvy, takže každý termit postupuje nezávisle na ostatních termitech; pozorovatel není vůbec zapojen. Pokud byste tedy chtěli přidat do modelu graf, potřebovali byste další trvalé tlačítko (pozorovatele) a museli byste spustit obě tlačítka najednou.

V současnosti v NetLogu nemůže jedno trvalé tlačítko spustit jiné tlačítko. Jediný způsob je stisknout je obě.

## Seznamy

V nejjednodušších modelech obsahuje každá proměnná pouze jednu informaci, obvykle číslo nebo řetězec. Seznamy vám umožňují ukládat více informací v jedné proměnné. Hodnota v seznamu může mít jakoukoliv podobu, může to být číslo, řetězec, agent, množina agentů či dokonce další seznam.

Seznamy jsou tak v NetLogu praktickým prostředkem, jak vytvářet balíčky informací. Když např. vaši agenti dělají opakované výpočty s více proměnnými, je lepší vytvořit proměnnou se seznamem než více číselných proměnných. Několik primitiv slouží ke zjednodušení procesu, ve kterém se provádí stejný výpočet každé hodnoty na seznamu.

Všechna primitiva týkající se seznamů naleznete v příslušné části Slovníčku NetLoga.

### Seznam konstant

Seznam můžete vytvořit tak, že jednoduše napíšete hodnoty, které na něm chcete mít, do hranatých závorek, takto: `set mylist [2 4 6 8]`. Všimněte si, že jednotlivé hodnoty jsou odděleny mezerou. Stejným způsobem postupujete i u seznamu obsahujícího čísla a řetězce nebo seznamy s vnořenými seznamy, např. `[[2 4] [3 5]]`.

Prázdný seznam je zapsán takto: `[]`, tj. v závorkách není uvedena žádná hodnota.

### Vytváření seznamů za pochodu

Chcete-li vytvořit seznam, v němž jsou hodnoty určeny reportéry, na rozdíl od seznamu konstant, použijte reportér `list`. Reportér `list` vezme dva další reportéry, spustí je a vrátí výsledky v podobě seznamu.

Jestliže chcete, aby seznam obsahoval dvě náhodná čísla, lze použít následující kód:

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

Pokaždé, když je tento příkaz spuštěn, nastaví se `random-list` na nový seznam dvou náhodných celých čísel.

K vytvoření kratších či delších seznamů použijte reportér `list` s méně či více než dvěma vstupy, celé volání ale musíte uzavřít do kulatých závorek, např.:

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

Více informací naleznete v podkapitole Proměnlivé množství vstupů.

Některé druhy seznamů se nejsnáze vytvoří pomocí reportéru `n-values`, který vám umožňuje sestavit seznam určité délky opakovaným spouštěním daného reportéru. Buď se v seznamu může opakovat jedna hodnota nebo seznam obsahuje čísla v daném rozsahu anebo náhodná čísla – existuje spousta možností. Detaily a příklady naleznete ve Slovníčku NetLoga.

Pomocí primitiva `of` vytvoříte seznam z množiny agentů. Vrací seznam obsahující hodnotu každého agenta pro daný reportér. (Reportér může být název jednoduché proměnné i složitější výraz – dokonce i volání procedury definované pomocí `to-report`.) Běžná spojení jsou např.:

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

Kombinovat dva a více listů lze pomocí reportéru `sentence`, který zřetězí seznamy tak, že jejich obsah spojí do jediného většího seznamu. Stejně jako `list`, i `sentence` má běžně dva vstupy, ale pokud je volání ohraničeno závorkami, může jich mít libovolné množství.

### Změna položek v seznamu

Seznamy se v podstatě nedají měnit, ale můžete vytvářet nové na základě starých. Chcete-li novým seznamem nahradit starý, použijte `set`. Např.:

```
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]]
```

Reportér `replace-item` má tři vstupy. První vstup určuje, která položka na seznamu se má změnit. 0 značí první položku, 1 druhou atd.

Chcete-li nějakou položku, např. 42, přidat na konec seznamu, použijte reportér `lput`. (`fput` přidá položku na začátek seznamu.)

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

Ale co když si to rozmyslíte? Reportér `but-last` (zkráceně `bl`) vrátí všechny položky na seznamu kromě poslední.

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

Předpokládejme, že se chceme zbavit položky 0, tj. 2 na začátku seznamu.

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

Zkusme si např. změnit třetí položku, která je zasazena uvnitř položky 3, z –2 na 9. Klíčové je uvědomit si, že název, který musíme použít k volání vloženého seznamu [3 0 –2], je `item 3 mylist`. Abychom změnili seznam v seznamu, vložíme do něj reportér `replace-item`. Kvůli přehlednosti používáme závorky.

```
set mylist (replace-item 3 mylist
                  (replace-item 2 (item 3 mylist) 9))
; mylist is now [7 10 Bob [3 0 9]]
```

## Opakování v seznamech

Chcete-li provést nějakou operaci postupně s každou položkou na seznamu, použijte příkaz `foreach` a reportér `map`.

`foreach` se používá v případě, že má každá jednotlivá položka na seznamu vykonat příkaz nebo příkazy. Použije seznam na vstupu a blok příkazů takto:

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

Proměnná `?` v bloku obsahuje aktuální hodnotu ze seznamu na vstupu.

Následují další příklady `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
```

Podobný je i reportér `map`, používá seznam na vstupu a další reportér. Na rozdíl od `foreach`, musí stát reportér na začátku, takto:

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

`map` vrátí seznam, který obsahuje výsledky aplikace reportéru na každou položku v seznamu na vstupu. Znovu použijte `?` při odkazu na aktuální položku v seznamu.

Následuje další příklad `map`:

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

`foreach` a `map` se nemusejí nutně hodit v každé situaci, kdy chcete provádět operaci s celým seznamem. V některých případech budete muset použít jinou metodu, např. vytvořit smyčku pomocí `repeat` či `while` nebo použít rekurzivní proceduru.

Primitivum `sort-by` používá podobnou syntax jako `map` a `foreach`. Reportér však potřebuje porovnat dva objekty, takže místo `?` používá zvláštní proměnné `?1` a `?2`.

Následuje příklad sort-by:

```
show sort-by [?1 < ?2] [4 1 3 2]
;; prints [1 2 3 4]
```

### Proměnlivé množství vstupů

Některé příkazy a reportéry týkající se seznamů a řetězců mohou mít různý počet vstupů. Když jim chceme předat počet vstupů, který se liší od jejich výchozího počtu, musí být primitivum a jeho vstupy uzavřeny v kulatých závorkách. Následuje několik příkladů:

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

Všimněte si, že každý z těchto speciálních příkazů má výchozí počet vstupů, u kterého nemusíte používat závorky. Primitiva s touto vlastností jsou list, word, sentence, map a foreach.

### Seznamy agentů

Dříve jsme si uvedli, že agenti v množinách mají vždy náhodné pořadí, které je pokaždé jiné. Jestliže potřebujete, aby agenti vykonávali akci v pevném pořadí, musíte vytvořit jejich seznam.

Pomohou vám dvě primitiva – sort a sort-by.

Obě primitiva sort a sort-by mohou mít jako vstup množinu agentů. Výsledkem je nový seznam obsahující stejné agenty, jako jsou v množině, ale v určitém pořadí.

Když použijete sort pro množinu želv, výsledkem bude seznam želv seřazených vzestupně podle identifikačního čísla.

Použijete-li sort pro množinu políček, výsledkem bude seznam políček seřazených zleva doprava, seshora dolů.

V případě použití sort pro množinu spojů bude výsledkem seznam spojů seřazených vzestupně nejdříve podle end1 a pak end2; zbývající vazby jsou uspořádány podle rodu v takovém pořadí, v jakém jsou deklarovány v panelu **Procedures**.

Jestliže potřebujete agenty seřadit sestupně, zkombinujte reverse a sort, například reverse sort turtles.

A pokud chcete agenty seřadit podle jiných kritérií, než nabízí `sort`, musíte použít `sort-by`.

Tento příklad:

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

vrátí seznam želv seřazených vzestupně podle proměnné želvy `size` (velikost).

### Použití ask pro seznam agentů

Když už jste vytvořili seznam agentů, můžete jim přikázat provést nějakou akci. K tomu použijte příkazy `foreach` a `ask` takto:

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

Tato kombinace prochází želvy ve vzestupném pořadí podle identifikačního čísla. Když nahradíte želvy políčky, prochází je zleva doprava nebo seshora dolů.

Použitím `foreach` způsobíte, že agenti v seznamu procházejí příkazy uvnitř `ask` jeden až po druhém, nikoliv najednou. Každý agent nejdříve dokončí příkazy a teprve potom je začne vykonávat další.

Pozor, `ask` nelze použít přímo pro seznam želv. `ask` je určeno pouze množinám želv a jednotlivým agentům.

### Efektivita seznamů

Používá-li váš model hodně dlouhých seznamů, měli byste znát rychlost operací seznamů v NetLogu, abyste napsali kód, který proběhne rychle.

Seznamy v NetLogu jsou jednosměrné spojové seznamy. Tento odborný termín z informatiky znamená, že když NetLogo potřebuje nalézt na seznamu položku, musí začít od začátku seznamu a pohybovat se od položky k položce, dokud nenalezne tu správnou. Například hledá-li stou položku, musí projít předchozích 99, jednu po druhé.

Znamená to také, že některé operace jsou velmi výkonné, a to konkrétně operace na začátku seznamu. Reportéry `first`, `but-first` a `fput` jsou velmi rychlé, zaberou stejně času bez závislosti na délce seznamu. Pokud vytváříte seznam tak, že přidáváte po jedné položce, je rychlejší použít `fput` než

`lput`. (A když bude seznam naopak, než potřebujete, pořád ještě můžete použít příkaz `reverse`, kterým seznam po dokončení převrátíte.)

Další rychlý reportér je `length`. NetLogo totiž vždy uchovává záznam o délce seznamu, takže ji nemusí skutečně měřit.

Následující reportéry jsou u dlouhých seznamů pomalé: `item`, `lput`, `but-last`, `last` a `one-of`.

# Aritmetika

Všechna čísla v NetLogu jsou uložena interně jako čísla s plovoucí řádovou čárkou s dvojnásobnou přesností, jak je určeno normou IEEE 754. 64bitová čísla sestávají z jednobitového znaménka, 11bitového exponentu a 52bitové mantisy. Více informací naleznete přímo v normě IEEE 754.

Celé číslo je v NetLogu jednoduše číslo, které nemá zlomkovou část. Mezi 3 a 3,0 neexistuje rozdíl, jsou to stejná čísla. (Stejným způsobem používají čísla lidé v každodenním životě, ale liší se od některých programovacích jazyků, které rozlišují celá čísla a čísla s desetinou čárkou.)

Celá čísla NetLogo zobrazí vždy bez koncové „.0“:

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

Pokud se na místě, kde je očekáváno celé číslo, objeví zlomek, je zlomková část jednoduše škrtnuta. Např. `crt 3.5` vytvoří tři želvy, část 0.5 je ignorována.

Rozsah celých čísel je +/–9007199254740992 (2 ^ 53, přibližně 9 biliard). Výpočty přesahující tento rozsah nezpůsobí chybu, jen budou méně přesné, protože nejméně důležité (binární) číslice jsou zaokrouhleny, aby se číslo vešlo do 64 bitů. U velkých čísel může toto zaokrouhlení vést k překvapivým nepřesným odpovědím:

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

Výpočty s menšími čísly mohou také vycházet překvapivě, pokud mají zlomky, protože ne všechny zlomky lze přesně zobrazit a dochází k zaokrouhlení. Například:

```
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
```

Jakákoliv operace, jejímž výsledkem jsou zvláštní hodnoty „nekonečno“ nebo „nečíselná hodnota“, způsobí běhovou chybu.

## Vědecký zápis

Velká nebo naopak velmi malá čísla s plovoucí desetinnou čárkou jsou v NetLogu zobrazena pomocí tzv. „vědeckého zápisu“:

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

Čísla ve vědeckém zápisu se odlišují písmenem E (pro „exponent“). Zápis znamená „krát deset na“, např. 1,0 E –12 znamená 1,0 krát 10 na minus dvanáctou:

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

Vědecký zápis tak můžete použít i ve svém kódu NetLoga:

```
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
```

Tyto příklady ukazují, že čísla se zlomky jsou zapsána vědeckým zápisem, pokud je exponent menší než –3 nebo větší než 6. Vědeckým zápisem jsou zobrazena i čísla nacházející se mimo číselný rozsah NetLoga od –9007199254740992 do 9007199254740992 (+/–2 ^ 53):

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

Když vkládáme číslo, může být písmeno E malé nebo velké. Když NetLogo číslo zobrazuje, je E vždy velké:

```
show 4.5e20
```

```
=> 4.5E20
```

### Přesnost plovoucí desetinné čárky

Čísla v NetLogu rovněž podléhají omezením, jež přináší zobrazení plovoucí desetinné čárky v binárním kódu, takže někdy můžete dostat trochu nepřesné výsledky. Například:

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

Tento problém se objevuje v každém programovacím jazyce používajícím plovoucí desetinná čísla.

Používáte-li přesné pevné částky, například dolary a centy, běžně se používají celá čísla (centy), která se pak vydělí 100, aby byl výsledek v dolarech.

Pokud používáte plovoucí desetinná čísla, musíte nahradit přímý test rovnosti jako `if x = 1 [...]` testem tolerujícím malou nepřesnost, např. `if abs (x - 1) < 0,0001 [...]`.

Na zaokrouhlování čísel při zobrazování se vám také může hodit primitivum `precision`. Ukazatele NetLoga také umějí zaokrouhlovat čísla, jež zobrazují, na nastavitelný počet desetinných míst.

## Náhodná čísla

Náhodná čísla používaná v NetLogu jsou tzv. „pseudonáhodná“ (jak je běžné v programování). Znamená to, že se jeví jako náhodná, ale ve skutečnosti jsou generována deterministickým procesem. „Deterministický“ značí, že pokud začnete stejným náhodným „semínkem“ (seed), dostanete pokaždé stejný výsledek. Výraz „semínko“ si vysvětlíme za chvilku.

Pseudonáhodná čísla jsou pro vědecké simulace v podstatě žádoucí, protože je důležité, aby se výsledek zkoumání dal zopakovat – aby si každý mohl experiment vyzkoušet a dostal stejný výsledek. Proto používá pseudonáhodná čísla i NetLogo, takže vaše experimenty si mohou zopakovat i ostatní.

Generátor náhodných čísel se v NetLogu inicializuje určitou hodnotou semínka, což může být jakékoliv celé číslo. Generátor dostane příkaz `random-seed` s konkrétním číslem a vygeneruje nadále vždy stejnou posloupnost náhodných čísel. Když například zadáte následující

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

vždy dostanete čísla 95, 7 a 54 v tomto pořadí.

Stejné pořadí čísel dostanete ale pouze v případě, že používáte stejnou verzi NetLoga. V novějších verzích bývají někdy provedeny změny v generátoru (teď momentálně používáme generátor Mersenne Twister).

Číslo vhodné jako semínko pro generátor náhodných čísel vytvoříte pomocí reportéru `new-seed`. `new seed` vytvoří semínko, pravidelně distribuované v prostoru možných semínek, v závislosti na datu a čase. Nikdy nevytvoří stejné semínko dvakrát za sebou.

**Ukázka kódu programu:** Příklad náhodného semínka (Random Seed Example)

Pokud náhodné semínko nezadáte sami, NetLogo ho vytvoří na základě současného data a času. Nelze zjistit, jaké číslo NetLogo vybralo, takže pokud chcete, aby daný běh modelu šel zopakovat, musíte náhodné semínko zadat předem.

Primitiva NetLoga, jejichž názvy obsahují „random“ (`random`, `random-float` atd.), nejsou jedinou částí používající náhodná čísla – mnoho další operací také činí náhodnou volbu. Např. množiny agentů jsou vždy v náhodném pořadí, `one-of` a `n-of` náhodně vybírá agenty, příkaz `sprout` vytvoří želvy náhodných barev a směru otočení, reportér `downhill` zvolí náhodné políčko s vazbou. Všechny tyto náhodné volby rovněž určuje náhodné semínko, takže běhy se dají reprodukovat.

Kromě rovnoměrně distribuovaných náhodných celých i desetinných čísel vytvořených pomocí `random` a `random-float` NetLogo nabízí i další náhodné distribuce, viz hesla `random-normal`, `random-poisson`, `random-exponential` ve Slovníčku NetLoga.

## Pomocný generátor

Kód spuštěný tlačítky nebo z příkazového panelu používá hlavní generátor náhodných čísel.

Kódy v ukazatelích používají pomocný generátor, takže i když ukazatel provede výpočet s použitím náhodných čísel, nemá to vliv na výsledek běhu modelu. To platí i o kódu v posuvnících.

## Lokální náhodnost

Někdy budete potřebovat, aby úsek kódu neměl vliv na stav hlavního generátoru náhodných čísel, a tím pádem na výsledek modelu. V tomto případě použijte příkaz `with-local-randomness`, více viz Slovníček NetLoga.

# Tvary želv

Želvy v NetLogu mají vektorové tvary. Spíše než jako síť pixelů jsou tvořeny základními geometrickými tvary – čtverci, kruhy a přímkami. Vektorové tvary se dají zvětšovat a otáčet. NetLogo si ukládá bitmapové obrázky vektorových tvarů velikosti 1, 1,5 a 2, aby se urychlil výpočet.

Tvar želvy je uložen v její proměnné `shape` a může být nastaven pomocí příkazu `set`.

Nové želvy mají výchozí tvar. Nastavení výchozího tvaru lze změnit pomocí primitiva `set-default-shape`, stejným způsobem lze nastavit i tvary pro jednotlivé rody želv.

Primitivum `shapes` vrací seznam aktuálně dostupných tvarů želv v daném modelu. To se nám může hodit v případě, kdy např. potřebujeme želvě přiřadit náhodný tvar:

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

Vlastní tvary vytvoříte v editoru tvarů želv (Turtle Shapes Editor). Ten můžete použít i v případě, že chcete přidat do modelu tvary z knihovny tvarů (Shapes library) nebo měnit tvary mezi modely. Více informací naleznete v kapitole Průvodce editorem tvarů této příručky.

Tloušťka čar vykreslujících vektorové tvary se nastavuje primitivem `set-line-thickness`.

**Ukázka kódu programu:** Příklad rodů a tvarů (Breeds and Shapes Example), Příklad animace tvarů (Shape Animation Example)

# Tvary spojů

Tvary spojů jsou podobné tvarům želv, k jejich vytvoření a nastavení ale používáme editor tvarů spojů (Link Shapes Editor). Celkový tvar spojů je tvořen 0 až 3 čarami různých tvarů a ukazatelem směru, jenž je vytvořen ze stejných prvků jako tvary želv. Spoje také mají proměnnou `shape`, která může být nastavena na jakýkoliv tvar spoje dostupný v daném modelu. Ve výchozím nastavení mají spoje výchozí tvar, který můžete změnit pomocí `set-default-shape`. Reportér `link-shapes` vrací všechny tvary spojů včetně aktuálního modelu.

Tloušťka čar ve tvaru spojů se nastavuje pomocí proměnné spoje `thickness`.

# Počítadlo kroků

V mnoha modelech NetLoga čas běží po diskrétních krocích (tzv. ticks). Zabudované počítadlo vám umožňuje sledovat, kolik kroků uběhlo.

Aktuální stav počítadla se nachází nad oknem zobrazení. (Pomocí tlačítka **Settings** můžete počítadlo schovat nebo změnit slovo **ticks** na jiné.)

Reportér `ticks` načte aktuální stav počítadla, příkaz `tick` posune počítadlo o 1 jednotku dopředu. Příkaz `clear-all` resetuje stav a nastaví počítadlo na 0. Chcete-li nastavit počítadlo na 0, aniž byste vše smazali, použijte příkaz `reset-ticks`.

Pokud váš model používá aktualizaci po jednotlivých krocích, příkaz `tick` také zaktualizuje zobrazení, viz následující podkapitola Aktualizace zobrazení.

### Kdy používat příkaz tick

Doporučujeme vám, abyste příkaz `tick` používali, až agenti dokončí pohyby a akce, ale než vykreslíte graf či vypočtete statistické údaje. Jedině tak totiž kód grafu či výpočtu – v případě že odkazuje na počítadlo – obdrží novou hodnotu. Například:

```
to go
  ask turtles [ move ]
  ask patches [ grow ]
  tick
  do-plots
end
to do-plots
  plotxy ticks count turtles
end
```

Tím, že `tick` umístíte před `do-plots`, dostane kód grafu správný stav počítadla od reportéru `ticks`.

### Necelé kroky

Ve většině modelů začíná stav počítadla na 0 a zvyšuje se po 1, tj. po celých číslech. Je však možné, aby počítadlo zobrazovalo zlomkové hodnoty.

Chcete-li počítadlo posunout o desetinné číslo, použijte příkaz `tick-advance`. Tento příkaz má číselný vstup, který určuje, o kolik se má počítadlo posunout.

Typicky se posun o zlomky kroků používá k aproximaci kontinuálních či zakřivených pohybů. Podívejte se např. na modely plynové laboratoře (GasLab) v **Models Library**, umístěné ve složce **Chemistry &Physics** (Chemie a fyzika). Tyto modely vypočtou přesný čas budoucí události a pak na tento čas nastaví počítadlo.

# Aktualizace zobrazení

Zobrazení vám umožňuje sledovat na obrazovce počítače agenty, každý jejich pohyb nebo změnu.

Samozřejmě, že agenty nevidíte přímo. Zobrazení je obrázek, který NetLogo nakreslí, aby vám ukázalo, jak agenti v určitý okamžik vypadají. Když tento okamžik pomine a agenti se pohnou a změní, musí se obrázek překreslit, aby odrážel nový stav světa. Toto překreslení se nazývá „aktualizace“ zobrazení.

Kdy se zobrazení aktualizuje? Následující část se zaměřuje na otázku, jak se NetLogo rozhoduje, kdy aktualizovat zobrazení, a jak toto můžete ovlivnit.

NetLogo nabízí dva způsoby aktualizace – „kontinuální“ (continuous) a „po jednotlivých krocích“ (tick-based). Přepínat mezi nimi můžete pomocí pop-up menu v horní části panelu **Interface**.

Kontinuální aktualizace jsou při spuštění NetLoga nebo nového modelu nastaveny jako výchozí. Skoro všechny modely v knihovně modelů však používají aktualizace po jednotlivých krocích.

Kontinuální aktualizace jsou nejjednodušší, avšak aktualizace po jednotlivých krocích vám dávají možnost ovládat, kdy a jak často aktualizace proběhnou.

Přesný čas aktualizace je velmi důležitý, protože určuje, co uvidíte na obrazovce. Pokud aktualizace nastane nečekaně, může se stát, že i to, co uvidíte, bude nečekané – pravděpodobně matoucí nebo zavádějící.

Důležitá je i frekvence aktualizací, protože zabírají čas. Čím více času stráví NetLogo aktualizováním zobrazení, tím pomaleji model poběží. S nižším počtem aktualizací běží model rychleji.

### Kontinuální aktualizace

Kontinuální aktualizace jsou velmi jednoduché. NetLogo aktualizuje určité zobrazení několikrát za sekundu – jako výchozí (posuvník je ve středu) jsou nastaveny aktualizace padesátkrát za sekundu.

Když posuvník rychlosti posunete na pomalejší variantu, bude se NetLogo aktualizovat častěji než padesátkrát za sekundu, výsledkem čehož bude zpomalení modelu. Zvýšíte-li rychlost, NetLogo se

bude aktualizovat méně než padesátkrát za sekundu. U nejrychlejšího nastavení se zobrazení aktualizuje jednou za několik sekund.

Při extrémně pomalém nastavení se bude NetLogo aktualizovat tak často, že uvidíte, jak se pohybují či mění jednotliví agenti.

Jestliže potřebujete dočasně vypnout kontinuální aktualizace, použijte příkaz `no-display`. Aktualizace znovu spustíte pomocí příkazu `display`, po kterém se NetLogo automaticky okamžitě aktualizuje (pokud zároveň nenastavíte model na vyšší rychlost).

## Aktualizace po jednotlivých krocích

Jak už jsme si řekli v podkapitole Počítadlo kroků, v mnoha modelech NetLoga čas běží po diskrétních krocích nazývaných „ticks“. Nejčastěji chcete zobrazení aktualizovat jednou za tento krok, a to mezi kroky. Tak vypadá i výchozí nastavení aktualizací po jednotlivých krocích.

Chcete-li přidat další aktualizace, použijte příkaz `display`. (Aktualizace však nemusejí proběhnout, pokud uživatel zrychlí model pomocí posuvníku rychlosti.)

K aktualizacím zobrazení po jednotlivých krocích nemusíte používat počítadlo. Pokud se počítadlo neposune, aktualizuje se zobrazení pouze příkazem `display`.

Když nastavíte posuvník na vyšší rychlost, NetLogo začne přeskakovat některé aktualizace, které by běžně proběhly. Nastavením na pomalejší rychlost nedocílíte vyššího počtu aktualizací, ale NetLogo se po každé bude zastavovat. Čím pomalejší nastavení, tím delší pauza.

I když je aktualizace zobrazení nastavena po jednotlivých krocích, proběhne pokaždé, když se tlačítko vrátí do původní polohy (jak jednorázové, tak trvalé) a když se dokončí příkaz zadaný v příkazovém panelu. Není tedy třeba přidávat k jednorázovým tlačítkům, která neposunují počítadlo, příkaz `display`. Naopak většina trvalých tlačítek neposunujících počítadlo příkaz `display` vyžaduje. Příkladem je model Život (Life), který naleznete v **Models Library** ve složce **Computer Science** (Informatika) pod **Cellular Automata** (Celulární automaty). Trvalá tlačítka, jež uživateli umožňují kreslit do zobrazovacího okna, používají příkaz `display`, aby uživatel viděl, co kreslí, aniž by se měnil stav počítadla.

### Výběr vhodného módu

Aktualizace po jednotlivých krocích má oproti kontinuální aktualizaci následující výhody:

1. konzistentní, předvídatelné chování, které se nemění na různých počítačích a v jednotlivých bězích;
2. kontinuální aktualizace mohou uživatele zmást, protože ukazují i stavy modelů, které by uživatel neměl vidět;
3. zvýšení rychlosti. Aktualizace zobrazení zabere čas, takže pokud stačí po každém kroku, běží model rychleji;
4. jelikož tlačítko PŘIPRAV (setup) neposunuje počítadlo, není ovlivněno nastavením rychlosti, což je žádoucí.

Jak jsme již zmínili, většina modelů v naší knihovně modelů používá aktualizace po jednotlivých krocích.

Kontinuální aktualizace se hodí pro modely, jejichž běh není rozdělen na krátké diskrétní fáze. Příkladem je model Termiti. Všimněte si ale, že model Příklad stavového stroje (State Machine Example) ukazujícího, jak změnit kód v Termitech, kroky používá.

I u modelů, jejichž aktualizace je běžně nastavena po jednotlivých krocích, se může hodit dočasně přepnout na kontinuální aktualizaci – např. když odstraňujeme chyby v kódu, protože vidíme, co se děje během kroku, nikoliv až jeho výsledek. Když přepnete na kontinuální aktualizaci, nastavte posuvník rychlosti na tak nízkou rychlost, dokud neuvidíte pohyb agentů jednoho po druhém. Až dokončíte opravu, nezapomeňte znovu přepnout na aktualizaci po jednotlivých krocích, protože volba módu aktualizace se ukládá společně s modelem.

## Vykreslení grafů

Abyste pochopili, co se v modelech děje, umožňuje vám NetLogo vytvářet grafy.

Na začátku musíte vytvořit jeden či více grafů v panelu **Interface**. Každý graf musí mít unikátní název. Tento název budete používat při odkazu v kódu v panelu **Procedures**.

Více informací o používání a úpravách grafů v panelu **Interface** naleznete v kapitole Průvodce rozhraním.

### Specifikace grafu

Jestliže v modelu máte pouze jeden graf, můžete začít s vykreslováním okamžitě. Jestliže jich máte víc, musíte určit, do kterého z nich chcete kreslit. K tomu použijte příkaz `set-current-plot` s názvem grafu v uvozovkách, takto:

```
set-current-plot "Distance vs. Time"
```

Název grafu se musí přesně shodovat s názvem uvedeným při vytvoření grafu. Pokud později změníte název grafu, musíte také pomocí set-current-plot aktualizovat název v modelu, aby volal správný graf. (Použijte funkce kopírovat a vložit.)

### Specifikace pera

Když vytváříte nový graf, má jen jedno pero. Pokud v grafu používáte i nadále pouze jedno pero, můžete graf začít vykreslovat okamžitě.

V grafu však lze mít i více per. Další pera vytvoříte v úpravách grafu pomocí **Plot Pens** v dolní části editovacího okna. Každé pero musí mít unikátní název. Tento název budete používat při odkazu v kódu v panelu **Procedures**.

U grafu s více pery musíte určit, kterým perem chcete kreslit. Pokud tak neučiníte, kreslení začne prvním perem v grafu. V případě, že chcete použít jiné pero, nastavte ho pomocí `set-current-plot-pen` s názvem pera v uvozovkách, takto:

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

### Body grafu

Pro zakreslování do grafu se používají dva základní příkazy `plot` and `plotxy`.

U příkazu `plot` musíte pouze určit hodnotu y, kterou chcete zakreslit. Hodnota x prvního bodu bude automaticky 0, druhého bodu 1 atd. (Interval má ve výchozím nastavení hodnotu 1, lze ho však změnit.)

Příkaz `plot` je užitečný zejména v případě, že chcete, aby model zakreslil nový bod po každém časovém kroku. Příklad:

```
to setup
  ...
  plot count turtles
end
```

```
to go
  ...
  plot count turtles
end
```

Všimněte si, že v tomto případě kreslíme z procedur `setup` (PŘIPRAV) i `go` (START), protože chceme zachytit počáteční stav systému. A zakreslujeme na konci procedury `go`, nikoliv na začátku, protože chceme, aby graf zobrazoval aktuální stav po dokončení procedury tlačítka START.

Potřebujete-li specifikovat hodnoty bodu grafu na ose X i Y, použijte příkaz plotxy.

**Ukázka kódu programu:** Příklad vykreslení grafu (Plotting Example)

### Další druhy grafů

Ve výchozím nastavení vykresluje pero křivku, tj. jednotlivé body jsou propojeny čárou.

Chcete-li pohnout perem, aniž by zakreslovalo, použijte příkaz `plot-pen-up`. Po tomto příkazu sice příkazy `plot` a `plotxy` pero posunou, ale nic se nezakreslí. Až bude pero umístěno v požadovaném bodě, skloníte ho pomocí `plot-pen-down`.

Jestliže chcete zakreslit jednotlivé body a ne křivku nebo preferujete sloupce, musíte změnit mód pera grafu. K dispozici jsou tři: křivka, sloupec a bod. Výchozím nastavením je křivka.

Výchozí mód pera lze změnit v úpravách grafu; pero je ale rovněž možné změnit dočasně pomocí příkazu `set-plot-pen-mode`. Jako vstup použijte číslo: 0 pro křivku, 1 pro sloupec, 2 pro bod.

### Histogramy

Histogram je speciální druh grafu, který měří, jak často se určité hodnoty nebo hodnoty v určitém rozmezí vyskytují ve skupině čísel objevujících se v modelu.

Předpokládejme například, že želvy v modelu mají proměnnou „věk“ (age). Můžeme vytvořit histogram distribuce věku mezi želvami příkazem `histogram`, takto:

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

Čísla, která se objeví v histogramu, nemusejí pocházet od množiny agentů, může to být jakýkoliv seznam čísel.

Pozor, příkaz `histogram` automaticky nemění aktuální mód pera na sloupec. Pokud chcete sloupcový mód, musíte ho nastavit sami. (Jak jsme si již říkali, výchozí mód pera změníte v nastavení grafu v panelu **Interface**.)

Šířka sloupců v histogramu je dána intervalem pera. Výchozí nastavení intervalu pera změníte v nastavení grafu v panelu **Interface**. Interval lze změnit i dočasně pomocí příkazu `set-plot-pen-interval` nebo `set-histogram-num-bars`. Použijete-li druhý příkaz, NetLogo automaticky nastaví interval tak, aby odpovídal uvedenému počtu sloupců v aktuálním rozmezí osy x.

**Ukázka kódu programu**: Příklad histogramu (Histogram Example)

### Vymazání a resetování

Aktuální graf můžete vymazat pomocí příkazu `clear-plot`, všechny grafy v modelu pomocí `clear-all-plots`. Příkaz `clear-all` vymaže vše v modelu, včetně grafů.

Pokud chcete odstranit pouze body, které do grafu zakreslilo aktuální pero, použijte `plot-pen-reset`.

Resetováním celého grafu nebo celého pera neodstraníte jenom zakreslená data, ale graf či pero se zároveň znovu nastaví do výchozího stavu tak, jak jste specifikovali v panelu **Interface** při jeho vytvoření či posledních úpravách. Příkazy jako `set-plot-x-range` a `set-plot-pen-color` jsou pouze dočasné.

### Automatické zvětšení grafu

Ve výchozím nastavení mají všechny grafy NetLogo zapnutou funkci automatického zvětšení grafu. Znamená to, že pokud se model snaží vykreslit bod mimo aktuální rozsah grafu, graf se zvětší podél jedné nebo obou os, aby nový bod byl vidět.

Aby se rozsah grafu nemusel měnit s každým přidáním nového bodu, zvětší se grafy i s nějakou rezervou – o 25 % při horizontálním růstu a 10 % při vertikálním růstu.

Chcete-li tuto funkci vypnout, odškrtněte v editovacím okně grafu rámeček **Autoplot?**. V současnosti není možné funkci vypnout či zapnout pouze pro jednu osu.

### Dočasná pera grafu

Ve většině grafů vám bude stačit pevně daný počet per, některé grafy však mají zvláštní požadavky, mohou například vyžadovat, aby se počet per měnil v závislosti na podmínkách. V těchto případech můžete v kódu programu vytvořit „dočasná“ pera a kreslit s nimi. Tato pera se nazývají dočasná, protože existují pouze do chvíle, než je graf vymazán (pomocí příkazů `clear-plot`, `clear-all-plots` či `clear-all`).

Dočasné pero vytvoříte příkazem `create-temporary-plot-pen`. Poté s ním můžete zacházet jako s klasickým perem, je nastaveno jako kreslící, má černou barvu, interval 1 a mód křivky. Tato nastavení změníte pomocí příkazů, viz sekce Vykreslení grafů ve Slovníčku NetLoga.

### Legenda

Legendu grafu zobrazíte zaškrtnutím políčka **Show legend** (Ukázat legendu) v editovacím okně. Pokud chcete, aby se v legendě nezobrazovalo konkrétní pero, odškrtněte v editovacím okně rámeček **Show in Legend** (Ukázat v legendě) pro dané pero.

### Závěr

V této podkapitole jsme si nevysvětlili celou problematiku systému grafů v NetLogu. Více informací o dalších příkazech a reportérech týkajících se kreslení grafů naleznete ve Slovníčku NetLoga.

Pokročilejší metody vytváření grafů jsou ilustrovány v ukázkových modelech v knihovně modelů. Viz také následující ukázky kódu programu:

**Ukázky kódu programu:** Příklad osy grafu (Plot Axis Example), Příklad vyhlazování grafu (Plot Smoothing Example), Příklad rolování grafu (Rolling Plot Example)

# Řetězce

Řetězcové konstanty do NetLoga vkládáme s uvozovkami.

Prázdný řetězec mezi uvozovkami neobsahuje žádný znak, zapíšeme ho takto: `""`.

Většina primitiv funguje s řetězci:

```
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"
```

Několik primitiv lze použít pouze pro řetězce – `is-string?`, `substring` a `word`:

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

Řetězce můžeme porovnat pomocí operátorů =, `!=`, <, >, <= a >=.

Potřebujete-li vložit do řetězce zvláštní symbol, použijte následující escape kód:

- `\n` = nový řádek (newline),
- `\t` = tabelátor (tab),
- `\"` = dvojité uvozovky (double quote),
- `\\` = zpětné lomítko (backslash).

# Výstup

Tato podkapitola se věnuje problematice výstupu na obrazovku, který může být rovněž uložen do souboru pomocí příkazu `export-output`. Potřebujete-li flexibilnější způsob zápisu dat do externích souborů, přečtěte si následující podkapitolu Soubory.

Základní příkazy v NetLogu, jež se používají pro vytvoření výstupu na obrazovce, jsou `print`, `show`, `type` a `write`. Tyto příkazy posílají své výstupy do příkazového panelu.

Příkazy jsou podrobněji vysvětleny ve Slovníčku NetLoga, zde uvádíme pouze základní použití:

`print` lze použít ve většině kontextů,

`show` vám ukáže, který agent co tiskne,

`type` vám umožní vytisknout několik věcí do jednoho řádku,

`write` vám umožní vytisknout hodnoty ve formátu, který lze znovu načíst pomocí `file-read`.

Model NetLoga může volitelně mít „oblast výstupu“ v panelu **Interface**, kde je oddělena od příkazového panelu. Chcete-li poslat výstup sem, místo do příkazového panelu, použijte příkazy `output-print`, `output-show`, `output-type` a `output-write`.

Oblast výstupu vymažete příkazem `clear-output` a do souboru ji uložíte pomocí `export-output`. Obsah oblasti výstupu uložíte příkazem `export-world`. Příkaz `import-world` vymaže oblast výstupu a nastaví její obsah na hodnoty importovaného souboru. Měli bychom podotknout, že velké množství dat posílaných do oblasti výstupu může zvýšit velikost exportovaných světů.

Použijete-li příkazy `output-print`, `output-show`, `output-type`, `output-write`, `clear-output` či `export-output` v modelu, který nemá zvláštní oblast výstupu, budou data zaslána do výstupu v příkazovém panelu.

# Soubory

V NetLogu existuje soubor primitiv, která uživateli umožňují pracovat s externími soubory. Všechny začínají předponou „`file-`“.

Když pracujeme se soubory, používáme v zásadě dva módy – čtení a zápis. Rozdíl mezi nimi spočívá v toku dat. Když načítáte informace ze souboru, data uložená v souboru vstupují do modelu. Na druhou stranu, zápis umožňuje proudění dat z modelu do souboru.

Máte-li NetLogo spuštěno jako applet ve webovém prohlížeči, můžete načítat pouze data ze souborů uložených na serveru ve stejném adresáři jako soubor modelu. V appletu nelze zapisovat do žádného souboru.

Vždy, když začínáte pracovat se soubory, použijte primitivum `file-open`, kterým určíte, jaký soubor budete používat. Dokud soubor neotevřete, nebudou fungovat ostatní primitiva.

Další primitiva „file-“ určují, v kterém módu soubor bude až do svého zavření – zda v módu čtení nebo zápisu.

Primitiva, při nichž jsou data načítána, zahrnují `file-read`, `file-read-line`, `file-read-characters` a `file-at-end?`. Pozor, soubor musí existovat dříve, než jej pro čtení otevřete.

**Ukázka kódu programu:** Příklad vstupu ze souboru (File Input Example)

Primitiva, při nichž jsou data zapisována, se podobají primitivům pro tisk v příkazovém panelu až na to že se výstup ukládá do souboru. Tato primitiva jsou `file-print`, `file-show`, `file-type` a `file-write`. Pozor, data nikdy nelze přepisovat, tj. pokusíte-li se zapisovat do souboru s existujícími daty, objeví se nová data na konci souboru. (Pokud chcete soubor přepsat, nejdřív jej vymaže pomocí `file-delete` a pak jej otevřete pro zápis.)

**Ukázka kódu programu**: Příklad výstupu do souboru (File Output Example)

Když skončíte práci se souborem, ukončíte jej pomocí `file-close`. Chcete-li potom soubor odstranit, použijte primitivum `file-delete`. Pokud chcete zavřít více otevřených souborů, musíte nejdříve vybrat soubor, který chcete zavřít, pomocí `file-open`.

```
;; 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
```

V případě, že chcete zavřít všechny soubory, použijte `file-close-all`.

Dvě primitiva, která si zasluhují pozornost, jsou `file-write` a `file-read`. Tato primitiva slouží k uložení a načtení konstant NetLoga, jako jsou čísla, seznamy, booleovské hodnoty a řetězce. `file-write` vždy uloží proměnnou takovým způsobem, že ji `file-read` dokáže správně interpretovat.

```
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
```

**Ukázka kódu programu:** Příklad vstupu do souboru (File Input Example) a Příklad výstupu do souboru (File Output Example)

### Uživatelský výběr

Primitiva `user-directory`, `user-file` a `user-new-file` použijete, chcete-li, aby měl uživatel možnost zvolit soubor či adresář, který kód použije.

## Videa

Tato podkapitola popisuje, jak model NetLoga zachytit jako video ve formátu QuickTime.

Nové video začnete příkazem `movie-start`. Název souboru by měl končit příponou `.mov`, což je rozšíření pro filmy v programu QuickTime.

Snímek do videa přidáte pomocí `movie-grab-view` nebo `movie-grab-interface` podle toho, jestli chcete, aby video ukazovalo pouze aktuální zobrazení nebo celý panel **Interface**. V jednom videu můžete používat pouze jedno z primitiv **movie-grab-**, nelze je míchat dohromady.

Jestliže jste dokončili s přidáváním snímků, použijte `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
```

Ve výchozím nastavení video přehrává patnáct snímků za sekundu. Snímkovou frekvenci (frame rate) upravíte tak, že voláte `movie-set-frame-rate` a zadáte jiný počet snímků. Snímkovou frekvenci musíte zadat po `movie-start`, ale předtím, než přidáte jakékoliv snímky.

Snímkovou frekvenci či počet zachycených snímků zjistíte pomocí `movie-status`, který vám vrátí řetězec s popisem stavu aktuálního videa.

Video odstraníte a soubor vymažete pomocí `movie-cancel`.

**Ukázka kódu programu:** Příklad videa (Movie Example)

Videa NetLoga jsou exportována jako nezkomprimované soubory ve formátu QuickTime. Abyste je přehráli, potřebujete přehrávač QuickTime Player, který je volně ke stažení od firmy Apple.

Jelikož nejsou videa komprimována, zabírají na disku hodně místa. Zkomprimovat je můžete pomocí dalšího softwaru, který vám umožní komprimaci bez ztráty kvality či s nižší kvalitou. Nižší kvalita znamená, že program, aby zmenšil velikost souboru, smaže nějaké detaily videa. Co zvolíte, záleží na povaze konkrétního modelu – snížení kvality se vyvarujte např. u zobrazení s vysokým rozlišením.

Jeden ze softwarů, který lze použít ke komprimaci videí ve formátu QuickTime a který běží na platformách Macintosh i Windows, je QuickTime Pro. Na Macu funguje i iMovie.

# Perspektiva

Zobrazení 2D a 3D ukazuje svět z perspektivy pozorovatele. Při výchozím nastavení shlíží pozorovatel v počátečním bodě na svět z kladné osy z. Perspektivu pozorovatele změníte pomocí příkazů pozorovatele `follow`, `ride` a `watch` a příkazů želvy `follow-me`, `ride-me` a `watch-me`. V módu sledování nebo jízdy se pozorovatel pohybuje po světě spolu s daným agentem. Rozdíl mezi sledováním (follow) a jízdou (ride) je viditelný pouze ve zobrazení 3D, kde uživatel nastavuje vzdálenost od agenta pomocí myši. V případě, že pozorovatel pronásleduje agenta z nulové vzdálenosti, tak na něm ve skutečnosti jede. V pozorovacím módu (watch) pozorovatel sleduje pohyby želvy, aniž by se sám pohnul. V obou typech zobrazení se na zaměřeném agentovi zobrazí světelný bod a ve zobrazení 3D se k němu pozorovatel otočí čelem. Který agent je právě zaměřený, lze určit pomocí reportéru `subject`.

Ukázka kódu programu: Příklad perspektivy (Perspective Example)

# Kreslicí vrstva

V kreslicí vrstvě mohou želvy zanechávat viditelné stopy.

Ve zobrazení se kreslení objeví na políčku, ale pod želvou. Na začátku je kreslicí vrstva prázdná a průhledná.

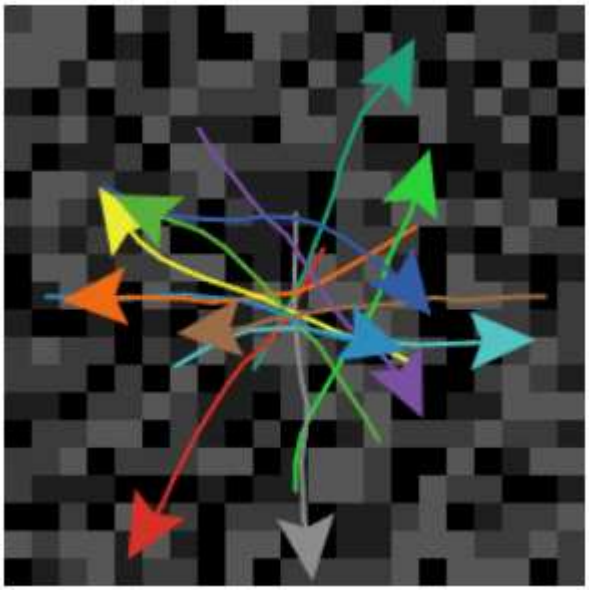
Kreslení vidíte vy, ale želvy (a políčka) nikoliv. Nevidí ho, nemohou ho vycítit ani na něj reagovat. Kreslení je určeno jen pro lidi.

Želvy kreslí a mažou čáry pomocí příkazů `pen-down` a `pen-erase`. Když má želva skloněné pero (nebo maže), kreslí za sebou čáru (nebo naopak maže), kamkoliv se pohne. Čáry mají stejnou barvu jako želva. Kreslení (či mazání) zastavíme příkazem `pen-up`.

Čáry nakreslené želvami mají tloušťku jednoho pixelu. Na jinou hodnotu tloušťku nastavíte pomocí proměnné želvy `pen-size` ještě před samotným kreslením (mazáním). U nových želv je proměnná nastavena na hodnotu 1.

Pokud se želva pohybuje způsobem, který neurčuje směr, např. `setxy` nebo `move-to`, nakreslí se čára nejkratší dráhy dodržující topologii.

Následuje ukázka obrázků několika želv, které nakreslily čáry v mřížce náhodně zbarvených políček. Všimněte si, jak želvy překrývají čáry a čáry překrývají barevná políčka. Použitá tloušťka `pen-size` byla 2:



Zadáním příkazu `stamp` za sebou želva nechá v kreslicí vrstvě svůj otisk a příkazem `stamp-erase` zase odstraní pixely pod sebou.

Celou nakreslenou vrstvu vymažeme příkazem pozorovatele `clear-drawing`. (Ke kompletnímu vymazání můžete rovněž použít `clear-all`.)

## Importování obrázku

Příkaz pozorovatele `import-drawing` vám umožňuje do kreslicí vrstvy importovat obrázkový soubor z disku.

`import-drawing` je vhodný, chceme-li přidat pozadí, aby se lidé měli na co dívat. Chcete-li, aby želvy a políčka na obrázek reagovaly, použijte `import-pcolors` nebo `import-pcolors-rgb`.

### Srovnání s ostatními jazyky Logo

Kreslení funguje v NetLogu trochu rozdílně od některých dalších jazyků Logo.

Mezi významné rozdíly patří:

- pera nových želv jsou ve výchozím nastavení nahoře, nikoliv dole;
- chcete-li želvu omezit hranicemi, nepoužíváte v NetLogu příkaz `fence`, ale vytvoříte svět a vypnete funkci cyklení;
- pozadí nelze nastavit pomocí `screen-color`, `bgcolor` či `setbg`, vytvoříte ho zabarvením políček, např. `ask patches [ set pcolor blue ]`.

Funkce kreslení, které NetLogo nepodporuje:

- neexistuje příkaz `window`. Tímto příkazem některá jiná Loga nechají želvu, aby bloumala po nekonečné rovině;
- neexistují příkazy `flood` či `fill`, jež se používají k zalití ohraničené oblasti barvou.

## Topologie

Topologie světa NetLoga má čtyři možné hodnoty – torus (torus), kostka (box), válec na výšku (vertical cylinder) a válec na šířku (horizontal cylinder). Topologii ovládáme tak, že zapneme či vypneme funkci cyklení (wrapping) ve směru osy X nebo Y. Výchozí nastavení světa je torus, stejně jako ve verzích před NetLogem 3.1.

Torus je zacyklen v obou směrech, což znamená, že horní a dolní okraj a levý a pravý okraj světa jsou propojeny. Jestliže se želva dostane za pravý okraj světa, objeví se na levém, stejně tak i „přepadne-li“ přes horní okraj, objeví se na dolním.

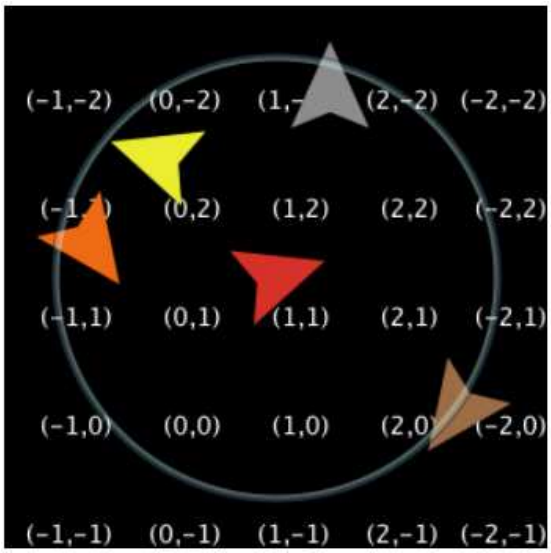
Kostka se necyklí v žádném směru. Svět je ohraničený, takže želvy, které se snaží pohybovat za okraj světa, nemohou. Všimněte si, že políčka podél okraje světa mají méně než osm sousedních políček – rohy mají tři a ostatní pět.

Válec na šířku i válec na výšku se cyklí pouze v jednom směru. Válec na šířku se cyklí vertikálně, takže horní okraj světa je spojený s dolním, avšak levý i pravý okraj je ohraničený. Válec na výšku se naopak cyklí horizontálně, takže levý a pravý okraj jsou propojeny, ale horní a spodní okraj je ohraničený.

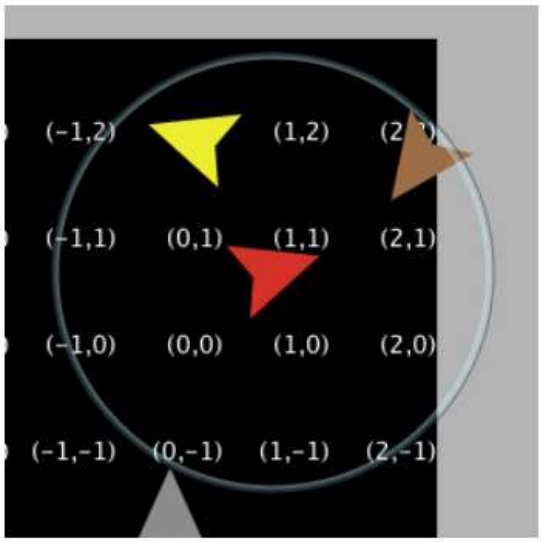
**Ukázka kódu programu:** Příklad sousedících políček (Neighbors Example)

Verze od NetLoga 3.0 umožňují cyklení světa pozorovat, takže když tvar želvy přesáhne okraj, objeví se daná část tvaru na opačném okraji zobrazení. (Želvy sami o sobě jsou body, proto nemohou existovat najednou na obou stranách světa, ale ve zobrazení jsou to tvary, jež zabírají určité místo.)

Cyklení také ovlivňuje, jak zobrazení vypadá v případě, že sledujete želvu. V toru uvidíte vždy celý svět kolem, ať už želva kráčí kamkoliv:




Oproti tomu v kostce či válci má svět hranice, takže oblasti za nimi se zobrazí šedě:



**Ukázka kódu programu:** Perspektivní demonstrace Termiti (Termites Perspective Demo) – torus, Perspektivní demonstrace Mravenci (Ants Perspective Demo) – kostka

Oproti verzi 3.0, kde se v nastavení dalo ovládat cyklení pouhého *vzhledu* světa, má NetLogo 3.1 nastavení, jímž ovládáme, zda se *skutečně* cyklí samotný svět, či nikoliv – jestli jsou opačné okraje opravdu spojeny. Nové nastavení cyklení určuje topologie světa, tj. zda je svět torus, kostka nebo válec, a má tak vliv na chování modelu, nikoliv pouze na jeho vzhled.

V minulosti museli autoři modelů psát zvláštní kód, aby simulovali svět s topologií kostky. Používali v něm speciální acyklická primitiva pro vzdálenost(xy) (distance(xy)), vnitřní poloměr (in-radius), vnitřní kužel (in-cone), otočení čelem(xy) (facy(xy)) a k(xy) (towards(xy)). Ve verzi 3.1 nejsou tato primitiva potřeba, místo toho přímo topologie rozhoduje, zda jsou primitiva cyklící či ne. Vždy se použije nejkratší vzdálenost, kterou daná topologie umožňuje. Např. vzdálenost středů políček v pravém dolním rohu (min-pxcor, min-pycor) do levého horního rohu (max-pxcor, max-pycor) bude pro minimální a maximální pxcor a pycor +/–2 v každé topologii následující.

- Torus – sqrt(2) ~ 1,414 (Hodnota bude stejná pro jakoukoliv velikost světa, protože políčka jsou v toru umístěna diagonálně od sebe.)
- Kostka – sqrt(šířka-světa^2 + výška-světa^2) ~ 7,07
- Válec na výšku – sqrt(výška-světa^2 + 1) ~ 5,099
- Válec na šířku – sqrt(šířka-světa^2 + 1) ~ 5,099

Všechna ostatní primitiva se budou chovat podobně jako vzdálenost. Jestliže jste dříve v modelu používali acyklická primitiva, doporučujeme, abyste je odstranili a změnili nastavení topologie světa.

Existuje mnoho důvodů, proč je lepší nastavit model, aby používal topologie a ne acyklická primitiva. Za prvé, předpokládáme, že pokud použijete tato primitiva, v podstatě modelujete svět, který není torus. Použijete-li topologii odpovídající světu, který modelujete, NetLogo automaticky kontroluje omezení a ulehčí vám práci, kód bude jednodušší a srozumitelnější a navíc přidá vizuální vysvětlivky, aby uživatel modelu lépe rozuměl, co právě modelujete. Uvědomte si, že s acyklickými primitivy bylo velmi obtížné vymodelovat válec, protože v případě, že v jednom ze směrů není povoleno cyklení, vrací tato primitiva buď vzdálenost či směr otočení.

Dalším důvodem je, že v modelu můžete mít chyby. Používáte-li kombinaci acyklických a cyklických primitiv, buď na tom z nějakých důvodů nezáleží, nebo máte v modelu chybu (my jsme jich několik ve svých modelech našli). Například model Vodič (Conductor) porovnával nezacyklenou vzdálenost s běžnou vzdáleností, aby zjistil, jestli se následující pozice nachází na opačné straně zacykleného světa – v tom případě elektron ze systému „vypadne“. Toto řešení je sice funkční, ale má bohužel jednu

chybu. Elektrony, které se cyklily na ose y, rovněž vypadly, ačkoliv to je v tomto případě chybné – jediný správný způsob je dosáhnout katody na levém konci drátu.

Jestliže odstraníte acyklické příkazy, není už topologie natvrdo naprogramovaná, takže lze model jednoduše testovat v rozdílných světech, aniž byste museli přidávat větší kus kódu (může se stát, že budete muset přidat několik řádků, abyste změnili torus na kostku; více viz podkapitola Jak změnit nastavení světa v modelu).

Ačkoliv jsme ze slovníčku odstranili všechna acyklická primitiva, lze je i nadále použít (chceme, aby fungovaly i starší modely, aniž by se musely změnit).

## Jak změnit nastavení světa v modelu

Když poprvé otevřete model ve verzi 3.1, NetLogo automaticky změní všechny výskyty `( -screen-edge-x )` na `min-pxcor` a všechny výskyty `screen-edge-x` na `max-pxcor` (stejně i pro y). V tomto okamžiku, i když to úplně nesouvisí s nastavením topologie, se také můžete rozhodnout, zda chcete výchozí bod posunout mimo střed. V předchozích verzích musel být svět kolem výchozího bodu symetrický, takže musel mít lichou šířku a výšku. To už teď neplatí, můžete používat jakékoliv kombinace minimální a maximální hodnoty, jedinou podmínkou je, že se ve světě musí nacházet bod (0,0). Nastavení je vhodné změnit, pokud modelujete v jednom či dvou kvadrantech či případně pokud je pro kód jednodušší používat pouze kladná čísla. Jestliže jste v minulosti vytvořili model vyžadující sudou mřížku, určitě odstraňte části kódu, jež to umožňovaly v dřívějších verzích.

**Ukázka kódu programu:** Automat mřížky plynu (Lattice Gas Automaton), Dvojčlenní králíci (Binomial Rabbits), Ragby (Rugby)

V NetLogu 3.1 jsme přidali nová primitiva, jež jsou nezbytná pro změnu topologie a hodí se i v případě, že topologii měnit nebudete. `random-pxcor`, `random-pycor`, `random-xcor` a `random-ycor` vracejí náhodné hodnoty v rozmezí minimálního a maximálního x a y. Ve starších verzích NetLoga jsme často při rozmisťování želv po světě spoléhali na jeho cyklení a napsali jsme `setxy random-float screen-size-x random-float screen-size-y`. Když však není cyklení v jednom či druhém směru povoleno, tento příkaz nefunguje (objeví se chybová hláška, že se snažíte umístit želvu mimo svět). Bez ohledu na topologii je lepší místo toho rovnou používat `setxy random-xcor rancom`.

Použití topologie u modelu dosáhnete následujícím způsobem. Nejdříve se musíte rozhodnout, jaké nastavení nejlépe odpovídá světu. Nedokážete-li to určit rovnou na základě reálného světa (např. pokoj je kostka, drát je válec), pomůže vám pár vodítek. Jestliže jsou kdekoliv v prohledávaném kódu hranice

světa nebo některá políčka na opačné straně světa nejsou považována za sousední, potom nepoužíváte torus. V případě, že jsou nastaveny hranice ve směru X i Y, má svět tvar kostky, pokud pouze ve směru X, je to válec na šířku, pokud ve směru Y, je to válec na výšku.

Pokud používáte acyklická primitiva, s největší pravděpodobností nemodelujete torus. Avšak pozor v případě, kdy používáte jak acyklická, tak cyklická primitiva. Může se stát, že použijete acyklické primitivum pro vizuální prvek, ale zbytek světa NetLoga je torus.

Máte-li určenou topologii a změnili-li jste její nastavení v editovacím okně zobrazení, jsou dalším krokem drobné změny v kódu. V případě, že jste se rozhodli, že svět je torus, nepotřebujete pravděpodobně udělat žádné změny. A jestliže váš model pracuje pouze se sousedními políčky a rovnoměrně jim rozděluje hodnoty, nebude těch změn mnoho.

Pokud se želvy v modelu pohybují po světě, musíte dále určit, co se s nimi stane, dosáhnou-li jeho okraje. Existuje několik možností: želva je vrácena zpět do světa (buď systematicky, nebo náhodně), želva vypadne ze systému (zemře) nebo je schována. Není už nutné pomocí souřadnic želvy ověřovat hranice, stačí se pouze zeptat NetLoga, zda je želva na okraji světa. Na to existuje několik způsobů, nejjednodušší je použít primitivum `can-move?`.

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

Primitivum `can-move?` pouze vrátí hodnotu **true** (pravda), nachází-li se pozice *vzdálenost* (distance) před želvou uvnitř světa NetLoga, jinak vrátí hodnotu **false** (nepravda). V tomto případě, pokud želva dorazí na konec světa, jednoduše se obrátí a jde, odkud přišla. Místo `can-move?` můžete také použít `patch-ahead 1 ! = nobody`. Chcete-li zadat želvě něco složitějšího, než aby se otočila, použijte `patch-at` s `dx` a `dy`.

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

Tentu kus kódu testuje, jestli želva narazí na horizontální či vertikální zeď a odrazí se.

V několika modelech želva, pokud se nemůže pohnout dopředu, jednoduše zemře, resp. opustí systém, jako např. ve Vodiči nebo Pastičkách (Mousetraps).

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

Pohybujete-li želvami po světě pomocí příkazu `setxy` a ne jen dopředu, měli byste se ujistit, zda políčko, kam želvu chcete posunout, existuje, protože leží-li jeho souřadnice mimo svět, způsobí

běhovou chybu. Následuje ukázka běžné situace, kdy model simuluje nekonečnou rovinu a želvy mimo zobrazení jsou jednoduše schovány.

```
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 ]
```

Tuto metodu používá několik modelů v knihovně, dobrými příklady jsou Gravitace (Gravitation), Problém N těles (N-Bodies) a Elektrostatika (Electrostatics).

Tím, že používáte topologie, můžete volně rozdělovat hodnoty (což v minulosti bylo poměrně složité). Každé políčko umí rovnoměrně rozdělit proměnnou všem sousedním políčkům; pokud má méně než 8 sousedů (nebo 4, používáte-li diffuse4), zůstane zbytek hodnoty na rozdělovacím políčku. Znamená to, že celková hodnota proměnných políček na celém světě zůstává konstantní. Jestliže jste měli v modelu na rozdělování zvláštní kód, můžete ho odstranit. Nicméně chcete-li, aby rozdělování probíhalo i přes okraj světa jako na nekonečné rovině, musíte pokaždé okraje vymazat, jak je ukázáno v Příkladu rozptýlení přes okraje (Diffuse Off Edges Example).

# Spoje

Spoj je agent spojující dvě želvy. Tyto dvě želvy se nazývají uzly (nodes). Spoj je vždy zobrazen jako čára mezi dvěma želvami. Spoje na rozdíl od želv nemají pozici, nejsou umístěny na políčku a nelze určit vzdálenost od spoje k dalšímu bodu.

Existují dva druhy spojů – neorientované a orientované. Orientovaný spoj vychází z jednoho uzlu a směruje do *dalšího/k dalšímu* uzlu. Jako orientovaný spoj lze modelovat například vztah rodiče k dítěti. Neorientovaný spoj se má k oběma uzlům stejně, každý uzel sdílí spoj s jiným uzlem. Jako neorientovaný spoj zobrazíme vztah mezi manžely či sourozenci.

Všechny spoje, stejně jako želvy či políčka, tvoří globální množinu agentů. Neorientované spoje vytvoříte pomocí příkazů `create-link-with` a `create-links-with`, orientované spoje pomocí `create-link-to`, `create-links-to`, `create-link-from` a `create-links-from`. Jakmile byl vytvořen první spoj, ať už orientovaný či neorientovaný, všechny bezrodé spoje se musejí shodovat (spoje mohou mít rody, stejně jako želvy, viz dále) – avšak nelze mít dva bezrodé spoje, z toho jeden orientovaný a druhou neorientovaný. V takovém případě nastane běhová chyba. (Když jsou všechny bezrodé spoje mrtvé, můžete vytvořit spoje rodu, který se liší od předchozích spojů.)

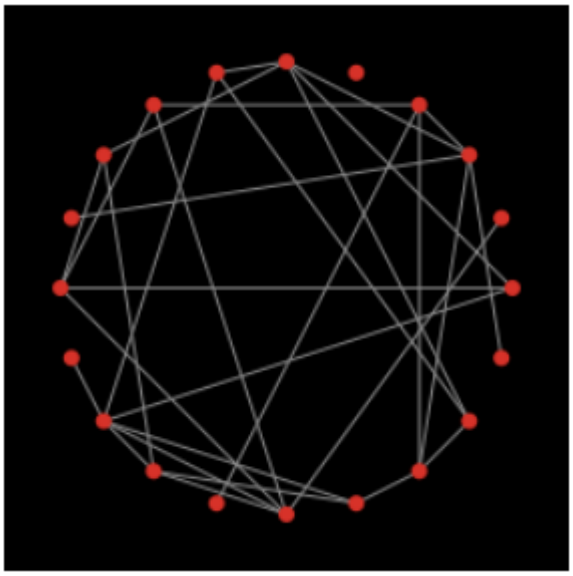
Obecně vzato obsahují názvy primitiv pracujících s orientovanými spojemi slova „in“, „out“, „to“ a „from“. Neorientovaná primitiva je neobsahují nebo používají „with“.

Rody spojů vám stejně jako rody želv umožňují v modelu definovat rozdílné druhy spojů. Rody spojů musejí být buď orientované, nebo neorientované, na rozdíl od bezrodých spojů se musí rod určit v době kompilování, nikoliv v běhu modelu. Rod u spojů deklarujete pomocí klíčových slov `undirected-link-breed` a `directed-link-breed`. Rod s neorientovanými spoji lze vytvořit pomocí příkazů `create-<breed>-with` a `create-<breeds>-with`, rod s orientovanými spoji pomocí `create-<breed>-to`, `create-<breeds>-to`, `create-<breed>-from` a `create-<breeds>-from`.

Mezi dvěma agenty nemůže existovat více než jeden neorientovaný spoj stejného rodu (či dva bezrodé spoje), což platí i pro orientované spoje – mezi dvěma agenty může být pouze jeden orientovaný spoj stejného rodu ve stejném směru. Dva orientované spoje (či dva bezrodé spoje) se mohou mezi dvěma agenty vyskytnout pouze v případě, že jsou v opačném směru.

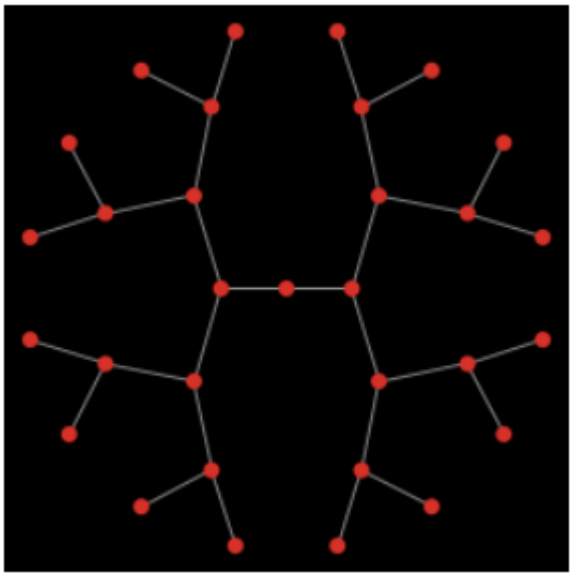
## Rozmístění v prostoru

Jako součást síťové podpory jsme přidali několik rozdílných primitiv, jež nám umožňují vizualizovat struktury sítí. Nejjednodušší je primitivum `layout-circle`, které kolem středu světa opíše kružnici o daném poloměru a rozmístí na ni v pravidelných odstupech agenty.

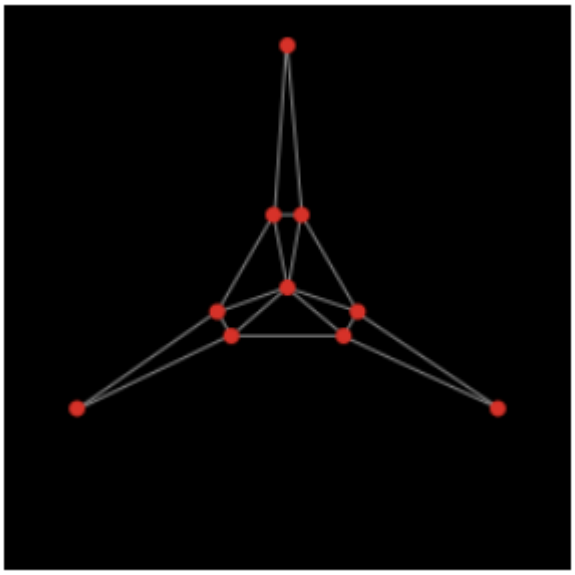


Primitivum `layout-radial` se hodí zejména pro stromovou strukturu a bude funkční i v případě, že ve stromě budeme mít nějaké cykly (pokud jich však bude hodně, nebude už struktura vypadat dobře). `layout-radial` vezme kořenového agenta, prohlásí ho za centrální uzel a umístí ho do bodu (0,0) a soustředně uspořádá připojené uzly. Uzly vzdálené od kořene jeden stupeň se uspořádají do kružnice kolem centrálního uzlu, stejným způsobem na ně navážou další a další. `layout-radial` se bude

snažit zahrnovat asymetrické grafy a dá prostor větvím, jež jsou širší. Pro vstup primitiva `layout-radial` lze také určit rod, takže můžeme vykreslit síť pouze jednoho rodu a nikoliv druhého.

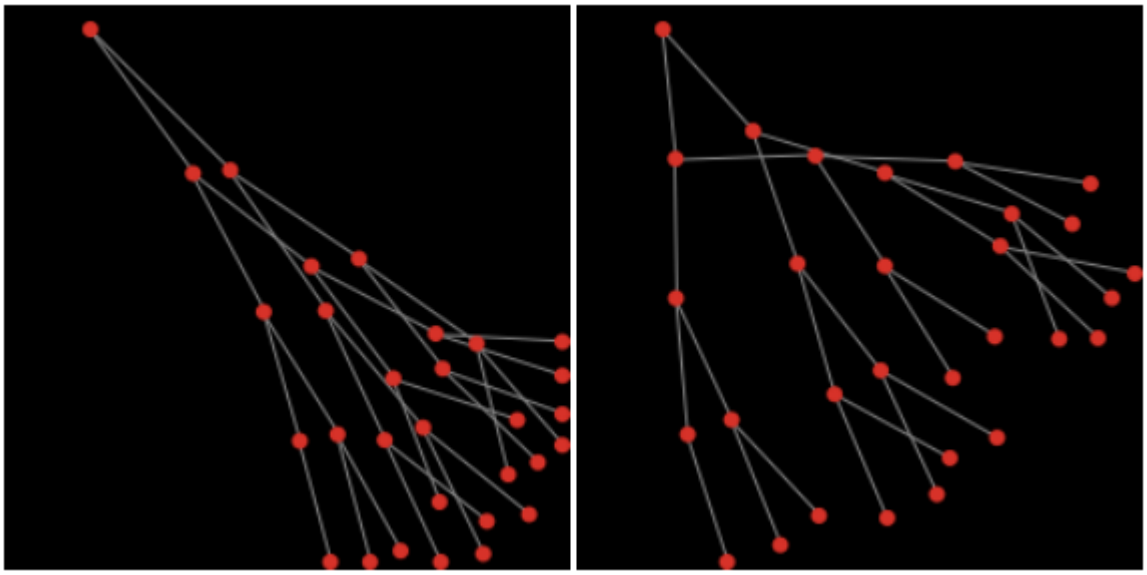


Primitivum `layout-tutte` vezme množinu zadaných kotev a umístí ostatní uzly do těžiště kotev, k nimž jsou připojeny. Množina kotev je automaticky uspořádána do kruhu, jehož poloměr určí uživatel, ostatní uzly konvergují ke svým pozicím postupně (což znamená, že musíte spustit několik kol, než je struktura stabilní).



Primitiva `layout-spring` a `layout-magspring` jsou velmi podobná a hodí se pro mnoho druhů sítí. Jejich nevýhodou je, že jsou celkem pomalá, protože ke konvergenci potřebují velký počet iterací. V obou strukturách fungují spoje jako pružiny, které stahují uzly, jež spojují, k sobě, přičemž uzly se vzájemně odpuzují. Struktura magnetické pružiny obsahuje navíc magnetické pole táhnoucí uzly ve směru, který zvolíte. Intenzita těchto sil je dána vstupy v primitivech. Tyto vstupy vždy mají hodnotu mezi 0 a 1, i malé změny ovlivní vzhled sítě. Pružiny mají také délku (vyjádřenou v políčkách), protože

je však do struktury zapojeno hodně sil, neskončí uzly od sebe na přesnou vzdálenost. Magnetická pružina má také booleovský vstup bidirectional? (volba obousměrnosti), který určuje, zda budou pružiny tlačit oběma směry paralelními s magnetickým polem – při jejím zapnutí budou sítě uspořádány ještě pravidelněji.



**Ukázka kódu programu:** Příklad sítě (Network Example), Příklad importu sítě (Network Import Example), Velká komponenta (Giant Component), Malé světy (Small Worlds), Upřednostněné připojení (Preferential Attachment)

# Příkaz ask-concurrent

V předchozích verzích NetLoga byl příkaz `ask` automaticky souběžný (concurrent). Od verze NetLoga 4.0 je příkaz sériový, tj. agenti vykonávají příkazy uvnitř `ask` postupně.

Následující část popisuje chování příkazu `ask-concurrent`, jenž se chová stejně jako příkaz `ask` ve starých verzích.

Příkaz `ask-concurrent` produkuje simulovanou souběžnost pomocí mechanismu střídání. Nejdříve vykoná první příkaz v závorkách první agent, pak druhý agent atd., dokud ho nevykonají všichni agenti v zadané množině. Potom nastoupí na řadu znovu první agent a dál se pokračuje stejně, dokud se nevykonají všechny příkazy v rámci `ask-concurrent`.

Kolo agenta končí, když provede akci, jež ovlivňuje stav světa, např. pohyb, vytvoření želvy, změnění hodnoty globální proměnné, proměnné želvy, políčka či spoje. (Nastavení lokální proměnné se nepočítá.)

Příkazy `forward (fd)` a `back (bk)` mají zvláštní průběh. Když jsou použity uvnitř `ask-concurrent`, mohou být vykonány ve vícero kolech. V jednom kole se želva může pohnout pouze o jedno políčko, takže např. `fd 20` je shodné s `repeat 20 [ fd 1 ]`, kdy každá želva skončí po jednom kole `fd`. Pokud uvedená vzdálenost není celé číslo, považuje se zlomek za celé kolo, např. `fd 20.3` je ekvivalentní `repeat 20 [ fd 1 ] fd 0.3`.

Příkaz `jump` je vždy vykonáván v jednom kole bez ohledu na vzdálenost.

Rozdílu mezi `ask` a `ask-concurrent` lépe porozumíte na základě následujících dvou příkazů:

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

U `ask` první želva udělá pět kroků dopředu, potom druhá želva udělá pět kroků dopředu atd.

U `ask-concurrent` udělají všechny želvy jeden krok, potom udělají další atd. Proto je tento příkaz ekvivalentní:

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

**Ukázka kódu programu**: Příklad ask-concurrent (Ask-Concurrent Example) ukazuje rozdíl mezi příkazy `ask` a `ask-concurrent`

Chování `ask-concurrent` nelze vždy jednoduše převést na `ask` jako ve výše uvedeném případě. Viz např. následující příkaz:

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

Abychom dostali stejné chování pomocí `ask`, museli bychom napsat:

```
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
  ]
]
```

Kolo jednoho agenta prodloužíte pomoci příkazu `without-interruption`. (Bloky příkazů uvnitř určitých příkazů jako např. `create-turtles` a `hatch` jsou obklopeny `without-interruption` automaticky.)

Je třeba si uvědomit, že chování `ask-concurrent` je zcela deterministické. Zadáme-li stejný kód a nastavíme-li stejné počáteční podmínky, dostaneme vždy stejný průběh a výsledek (v případě, že používáte stejnou verzi NetLoga a začnete modelování se shodným náhodným semínkem).

Doporučovali bychom vám, abyste svůj model napsali tak, že nebude záviset na přesných detailech fungování `ask-concurrent`. Nemůžeme vám zaručit, že v budoucích verzích NetLoga tuto sémantiku zachováme.

# Vazba

Vazba spojuje dvě želvy tak, že pohyb jedné želvy ovlivní pozici a směr otočení druhé. Vazba je vlastnost spojů, takže pokud má být mezi dvěma želvami vytvořen vztah vazby, musí mezi nimi existovat spoj.

Když je vazba spoje `tie-mode` nastavena na pevnou nebo volnou, jsou konce `end1` a `end2` spojeny. Je-li spoj orientovaný, je `end1` kořenovým agentem a `end2` agentem listovým. Když se pohne `end1` (pomocí `fd`, `jump`, `setxy` atd.), pohne se ve stejném směru a o stejnou vzdálenost i `end2`. Když se však pohne `end2`, nemá to na `end1` žádný vliv.

V případě, že je spoj neorientovaný, je vazba vzájemná, tj. ať pohyb vykoná kterákoliv z želv, je druhá ovlivněna a také se pohne. Takže záleží na tom, která želva pohybuje kterou, podle toho pak určíme kořenového a listového agenta. Kořenovým agentem je vždy želva, jež pohyb započne.

Když se kořenová želva otočí doprava nebo doleva, oběhne listová želva o stejný počet kolem ní, jako by byla připevněna ojí. Když je `tie-mode` nastaven na pevný, otočí se listová želva o stejný úhel jako želva kořenová. Když je `tie-mode` volný, zůstane směr otočení listové želvy nezměněn.

Typ vazby spoje `tie-mode` se nastavuje na pevný pomocí příkazu `tie`, na nulový (tj. želvy už nejsou spojeny) pomocí `untie`. Vazbu volnou nastavíte pomocí `set tie-mode "free"`.

**Ukázka kódu programu:** Příklad systému vazeb (Tie System Example)

# Více zdrojových souborů

Klíčové slovo `__includes` vám umožňuje použít v jednom modelu NetLoga více zdrojových souborů.

Slovo začíná dvěma podtržítky, což indikuje, že je tato funkce experimentální a může se v budoucích verzích NetLoga změnit.

Když otevřete model, jenž používá klíčové slovo `__includes` nebo když toto slovo přidáte na začátek modelu a stiskněte tlačítko Check (Zkontrolovat), objeví se lišta s menu Includes a nabídne vám soubory zahrnuté v modelu.

Uvedené soubory se otevřou v nových panelech **Procedures**. Více detailů naleznete v kapitole Průvodce rozhraním.

Do externích zdrojových souborů (`.nls`) můžete umístit cokoliv, co běžně dáváte do panelu Procedures: `globals`, `breed`, `turtles-own`, `patches-own`, `breeds-own`, definice procedur atd. Uvědomte si však, že všechny tyto deklarace sdílejí stejný jmenný prostor (namespace), tj. deklarujete-li globální proměnnou `my-global` v panelu **Procedures**, nemůžete deklarovat `my-global` ať už jako globální či jinou proměnnou v jakémkoliv souboru, který do modelu zahrnete. `my-global` bude přístupná ze všech zahrnutých souborů. To platí i pro případ, kdy je proměnná `my-global` deklarována v jednom ze zahrnutých souborů.

# Syntax

Tato podkapitola obsahuje odbornou terminologii, s níž není mnoho čtenářů obeznámeno.

### Klíčová slova

Jediná klíčová slova v jazyce jsou `globals`, `breed`, `turtles-own`, `patches-own`, `to`, `to-report` a `end`, plus `extenstions` a experimentální `__includes`. (Názvy vestavěných primitiv se nesmějí překrývat ani se měnit, takže v podstatě fungují také jako jistý druh klíčového slova.)

### Identifikátory

Všechna primitiva, názvy globálních proměnných a proměnných agentů a názvy procedur sdílejí jeden společný jmenný prostor, v němž názvy musejí být unikátní a v němž nehraje roli velikost písmen.

Lokální názvy (proměnné `let` a názvy vstupů procedur) se nesmějí překrývat mezi sebou ani nesmějí překrýt globální názvy. Identifikátory obsahují písmena, číslice a následující znaky ASCII:

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

Jiné znaky než ASCII nejsou v současné verzi jako identifikátory povoleny. (Uvědomujeme si, že to může mezinárodním uživatelům způsobit potíže a plánujeme v budoucí verzi tento problém odstranit.)

Některé názvy primitiv začínají dvojitým podtržítkem, což ukazuje, že jsou pouze experimentální a s největší pravděpodobností se v budoucích verzích NetLoga změní nebo budou odstraněny.

Identifikátory začínající otazníkem jsou rezervovány.

## Rozsah platnosti

NetLogo má lexikální rozsah platnosti. Lokální proměnné (včetně vstupů do procedur) jsou přístupné z bloku příkazů, v nichž jsou deklarovány, ale už nejsou přístupné v procedurách volaných těmito příkazy.

## Komentáře

Středník uvozuje komentář, jenž pokračuje až do konce řádku. Nelze psát komentář na více řádků.

## Struktura

Program sestává z volitelných deklarací (`globals`, `breed`, `turtles-own`, `patches-own`, `<BREED>-own`) v jakémkoliv pořadí, po kterých následuje nula a více definicí procedur. Více rodů může být určeno pomocí zvláštních deklarací `breed`, ostatní deklarace se mohou objevit pouze jednou.

Každá definice procedury začíná `to` nebo `to-report`, následuje název procedury a volitelný seznam názvů vstupů v hranatých závorkách. Každá procedura končí `end`. Mezi tím je nula a více příkazů.

## Příkazy a reportéry

Příkazy mohou mít nula a více vstupů – vstupy jsou reportéry, jež také mají nula a více vstupů. Příkazy neodděluje ani neukončuje žádná interpunkce, není ani mezi vstupy. Identifikátory musejí být odděleny

mezerou nebo kulatými či hranatými závorkami. (Takže např. `a+b` je jeden identifikátor, avšak `a(b[c]d)e` obsahuje indentifikátorů pět.)

Všechny příkazy i všechny reportéry nadefinované uživatelem jsou prefixové. Většina reportérů primitiv jsou také prefixové, jen některé (aritmetické operátory, booleovské operátory a některé operátory množiny agentů jako `with` a `in-points`) jsou infixové.

Všechny příkazy a reportéry, ať už primitivní nebo nadefinované uživatelem, mají automaticky pevný počet vstupů. (Proto syntax jazyka funguje, ačkoliv nepoužívá žádnou interpunkci k oddělení či ukončení příkazů a/nebo vstupů.) Některá primitiva jsou variadická, tj. mohou případně mít i jiný než výchozí počet vstupů, potom ale používáme závorky, např. `(list 1 2 3)` (primitivum `list` má jako výchozí dva vstupy). Kulaté závorky se rovněž používají k přepsání výchozí priority operátoru, např. `(1 + 2) * 3`, stejně jako v ostatních programovacích jazycích.

Někdy má primitivum vstup ve formě bloku příkazů (nula a více příkazů uvnitř hranatých závorek) nebo bloku reportéru (jeden reportér uvnitř hranatých závorek). Procedury nadefinované uživatelem nesmějí mít jako vstup blok příkazů nebo reportéru.

Priorita operátoru je následující, od nejvyšší po nejnižší:

- `with`, `at-points`, `in-radius`, `in-cone`
- (všechna ostatní primitiva a procedury definované uživatelem)
- `^`
- `*`, `/`, `mod`
- `+`, `-`
- `<`, `>`, `<=`, `>=`
- `=`, `!=`
- `and`, `or`, `xor`

## Porovnání s ostatními jazyky Logo

Logo je volná rodina jazyků, neexistuje pro něj žádná všeobecně uznávaná definice. Věříme, že NetLogo má s ostatními Logy dost společného na to, aby si zasloužilo jméno Logo. NetLogo se však zároveň v několika ohledech od většiny ostatních Log liší. Následují nejpodstatnější rozdíly.

**Rozdíly znatelné na první pohled**

- Liší se priority matematických operátorů. Infixy (jako +, * atd.) mají nižší prioritu než reportéry s názvy. Když např. v jazycích Logo napíšete `sin x + 1`, bude to interpretováno jako `sin (x + 1)`. NetLogo však zápis interpretuje stejně jako většina ostatních programovacích jazyků, tj. tak, jak by se dělo ve standardní matematické notaci, konkrétně `(sin x) + 1`.
- Reportéry `and` a `or` jsou zvláštní formy, nikoliv běžné funkce, a navíc provádějí neúplné vyhodnocování – pokud není třeba, nevyhodnotí už svůj druhý vstup (což se děje u `and` v případě, že první vstup je nepravdivý, a u `or` v případě, že první vstup je pravdivý).
- Procedury lze definovat pouze v panelu **Procedures**, nikoliv v příkazovém panelu.
- Procedury reportérů, tj. procedury, jež vrací hodnotu, musejí být definovány pomocí `to-report` místo `to`. Hodnotu z procedury vrátíte pomocí `report`, nikoliv `output`.
- Když definujete proceduru, musejí být její vstupy uvnitř hranatých závorek, např. `to square [x]`.
- Názvy proměnných jsou vždy bez interpunkce: píšeme `foo`, nikdy `:foo` nebo `"foo`. (Místo příkazu `make`, v kterém se vyskytuje argument v uvozovkách, používáme zvláštní podobu `set`, která nebere v potaz první vstup.) Tím pádem procedury a proměnné sdílejí jeden společný jmenný prostor.

Poslední tři rozdíly jsou ilustrovány na následujících definicích procedur:

| **většina jazyků Logo** | **NetLogo** |
|---|---|
| `to square :x` | `to-report square [x]` |
| `output :x * :x` | `report x * x` |
| `end` | `end` |

**Hlubší rozdíly**

- NetLogo má lexikální, tj. statický rozsah platnosti, nikoliv dynamický.
- NetLogo nemá žádný „slovní“ datový typ (jak Lisp nazývá „symboly“). Mohli bychom ho případně přidat, ale v modelování založeném na agentech ho potřebujeme jen velmi zřídka – ve většině situací, kdy by tradiční Logo použilo slova, jednoduše použijeme řetězce. Např. v Logu

byste mohli napsat `[see spot run]` (seznam slov), místo toho však v NetLogu musíte napsat `"see spot run"` (řetězec) nebo `["see" "spot" "run"]` (seznam řetězců).

- Příkaz NetLoga `run` funguje s řetězci, nikoliv se seznamy (protože nemáme žádný „slovní“ datový typ), a nedovoluje definování či předefinování procedur.
- Řídící struktury jako `if` a `while` nejsou obyčejné funkce, ale zvláštní formy. Zvláštní formy nemůžete sami definovat, takže nemůžete definovat ani řídící struktury. (Nepomůže zde ani příkaz NetLoga `run`.)
- Stejně jako ve většině Log není přípustné, aby hodnoty měly podobu funkce. Většina Log však poskytuje podobnou, i když méně obecnou funkcionalitu tím, že povoluje používat části zdrojového kódu ve formě seznamu. Podobné možnosti v NetLogu jsou v současné době omezené. V několika našich vestavěných zvláštních formách užíváme styl šablon UCBLoga, abychom dosáhli podobného účinku, např. `sort-by [length ?1 < length ?2] string-list`. Za určitých okolností lze použít i příkazy `run` a `runresult`, avšak na rozdíl od většiny Log tyto příkazy fungují jen s řetězci, nikoliv se seznamy.
- NetLogo má samozřejmě mnoho dalších funkcí a vlastností, které v ostatních jazycích Logo nenaleznete – nejdůležitější ze všeho jsou agenti a množiny agentů.

Copyright 1999-2009 by Uri Wilensky.
Všechna práva vyhrazena.

Aplikace NetLogo, modely i dokumentace jsou šířeny veřejnosti zdarma pro účel tvorby a studia modelů. Software, modely a dokumentaci je možné pro studijní a výzkumné účely používat a měnit, a to za podmínky, že je výsledný produkt nabízen bezplatně a s uvedením informace o autorských právech a jménem původce na všech kopiích a související dokumentaci.

Pro jiné využití - než jsou výše zmíněné nekomerční způsoby - celku i jednotlivých částí (a to jak v původní, nebo změněné podobě) je třeba předem požádat o svolení od Uri Wilensky. Software, modely ani dokumentace nesmějí být užívány, přepisovány, ani upravovány jako součást komerčního softwaru nebo hardwaru bez předchozího získání licence od Uri Wilensky. Nezaručujeme kompatibilitu tohoto systému s jakýmkoliv jiným systémem a neposkytujeme žádné záruky.

Pro účely citování v akademických publikacích používejte tento odkaz:
Wilensky, U. (1999). NetLogo. http://ccl.northwestern.edu/netlogo. Center for Connected Learning and Computer-Based Modeling. Northwestern University, Evanston, IL.