Tutorial 3: Procedury

V tomto tutorialu se krok po kroku naučíte, jak vyvinout kompletní model.

Agenti a procedury

V druhém tutorialu jste se naučili, jak používat příkazový panel a monitory agentů a díky nim zkoumat a měnit agenty a zadávat jim příkazy. Teď je na čase, abyste se dozvěděli více o skutečném srdci modelu Betlova – panelu **Procedures**.

Už jsme si představili různé druhy agentů, kterým lze zadat příkazy NetLoga: políčka, želvy, spoje a pozorovatele. Políčka jsou nepohyblivá a jsou uspořádána do mřížky, po které se pohybují želvy. Dvě želvy mezi sebou spojují spoje. Pozorovatel na všechno dohlíží a dělá vše, co nemůžou sami o sobě udělat želvy, políčka a spoje.

Všechny čtyři typy agentů můžou vykonávat příkazy NetLoga. Dále můžou spouštět „procedury“ – procedura zkombinuje sérii příkazů Netloga do jediného nového příkazu, který definujete vy.

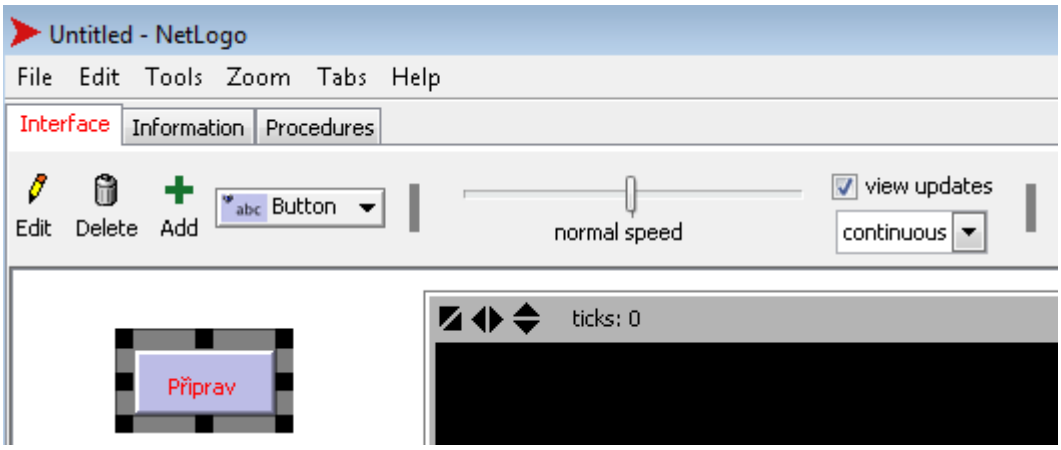
Teď se naučíte, jak psát procedury, díky kterým se budou želvy pohybovat, jíst, reprodukovat se a umírat. Také se naučíte, jak vytvořit ukazatele, posuvníky a grafy. Model, který společně vytvoříme, je založený na ekosystému, podobně jako části modelu Vlci a ovce z Tutorialu 1.

Jak vytvořit tlačítko PŘIPRAV (setup)

Pro vytvoření nového modelu zvolte položku **New** (Nový) z nabídky **File**. Následujícím způsobem vytvoříte tlačítko PŘIPRAV:

- Klikněte na ikonku **Button** (Tlačítko) na liště v horní části panelu Interface.
- Klikněte na místo v bílém okně, kam chcete umístit tlačítko.
- Otevře se dialogové okno určené pro editaci tlačítek. Napište `setup` do pole označeného **Commands** (Příkazy) a `Připrav` do pole **Display name** (Zobrazit jméno)
- Po dokončení stiskněte tlačítko OK a okno se zavře.

Nyní jste vytvořili tlačítko PŘIPRAV. Stisknutím tohoto tlačítka spustíte proceduru zvanou PŘIPRAV. Procedura je série příkazů NetLoga, které jsme označili novým jménem. Zatím jsme ještě tuto proceduru nedefinovali (učiníme tak brzy). Protože jsme vytvořili tlačítko k dosud neexistující proceduře, zobrazí se tlačítko červeně:



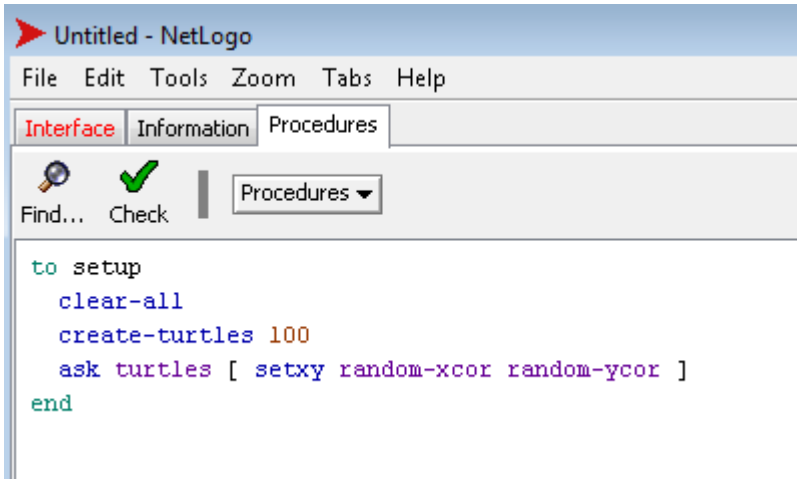
Chcete-li chybovou hlášku vidět, klikněte na tlačítko.

Teď si vytvoříme proceduru SETUP, takže nám chybová hláška zmizí:

- Přepněte okno do panelu **Procedures**.
- Napište následující:

```
to setup
  clear-all
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Po dokončení bude panel **Procedures** vypadat takto:

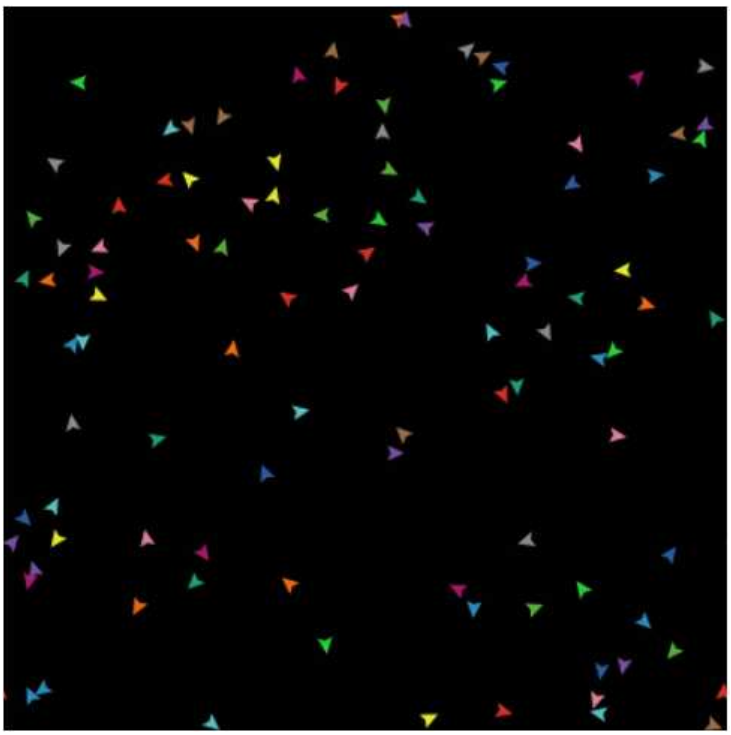


Všimněte si, že řádky jsou různě odsazeny. Většina lidí odsazení preferuje kvůli čitelnosti textu, ale není povinné. Procedura začíná slovem `to` a končí `end`. Takto bude začínat a končit každá procedura, kterou vytvoříte.

Podívejme se blíže na text, který jste napsali, a vysvětlíme si, co který řádek dělá:

- `to setup` začíná definici procedury zvané `setup` (PŘIPRAV).
- `clear-all` resetuje svět do počátečního, prázdného stavu. Všechna políčka zčernají a zmizí všechny želvy, které jste vytvořili. V podstatě smaže všechny změny a připraví okno pro běh nového modelu.
- `create-turtles 100` vytvoří 100 želv. Začínají ve výchozím bodě, tj. středu 0,0.
- `ask turtles [ ... ]` řekne všem želvám, aby nezávisle na sobě provedly příkaz uprostřed závorek. (Každý příkaz provádí v NetLogu nějaký agent.)
- `ask` je také příkaz. V tomto případě ho zadává sám sobě pozorovatel a říká, že je určen pro všechny želvy, které mají něco vykonat.
- `setxy random-xcor random y-cor` je příkaz, který používá „reportéry“ (reporters). Reportér na rozdíl od příkazu vrací výsledek. Nejdříve každá želva spustí reportér `random-xcor`, který vrátí náhodné číslo z přípustného rozmezí souřadnic želvy na ose x. Potom každá želva provede reportér `random-ycor` pro osu y. A nakonec každá želva provede příkaz `setxy`, na jehož vstupu budou dvě ohlášená čísla. Želva se posune na místo s těmito souřadnicemi.
- `end` ukončí definici procedury `setup` (PŘIPRAV).

Až dopíšete text, přepněte do panelu **Interface** a stiskněte tlačítko, které jste předtím vytvořili. Uvidíte, že se želvy náhodně rozmístily po světě:



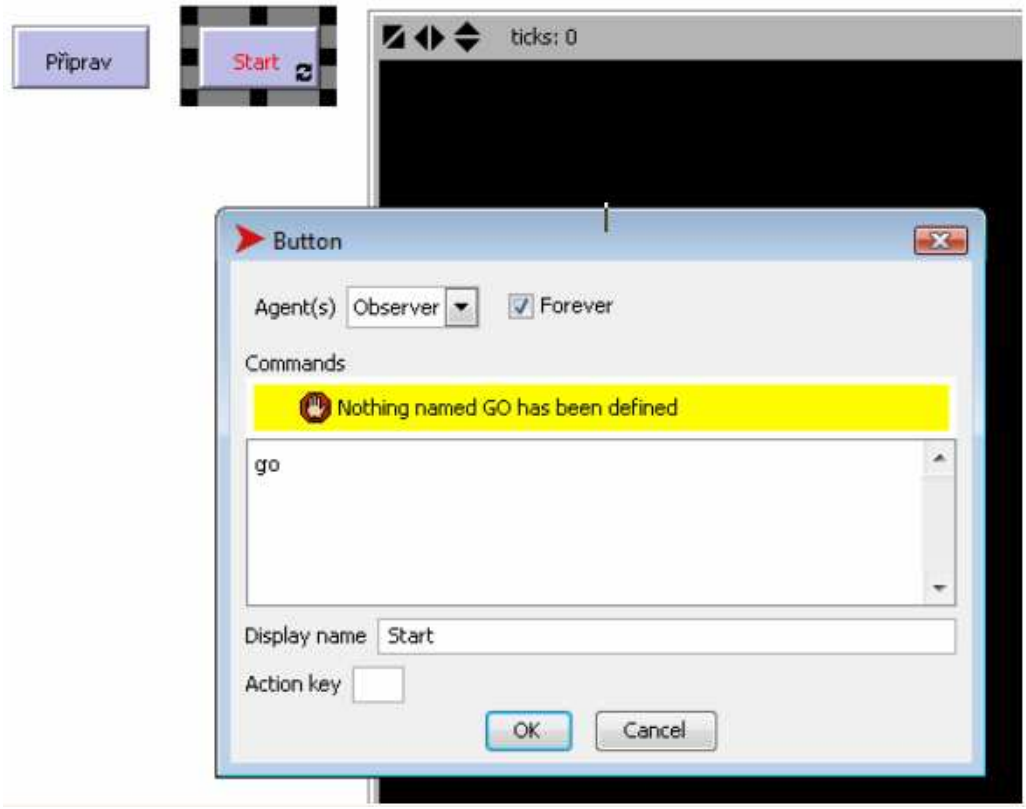
Stiskněte tlačítko PŘIPRAV ještě několikrát a sledujte, jak se rozmístění želv pokaždé změní. Všimněte si, že se některé želvy překrývají.

Přemýšlejte, jak jste k této fázi dospěli. Vytvořili jste tlačítko v uživatelském rozhraní a proceduru, která toto tlačítko používá. Tlačítko fungovalo až v momentě, kdy jste dokončili oba samostatné kroky. V průběhu tohoto tutoriálu zjistíte, že když chcete přidat do modelu novou funkci, musíte udělat dva a více podobných kroků. Pokud dokončíte krok, o kterém jste si mysleli, že je poslední, ale funkce neběží, čtěte dále, zda není třeba udělat ještě něco. Po pročtení několika odstavců se vraťte a projděte si instrukce, jestli jste něco nevynechali.

Jak vytvořit tlačítko START (go)

Nyní si vytvoříme tlačítko START. Postupujte stejně jako u tlačítka PŘIPRAV, až na:

- Do příkazového pole napište `Start` místo `Připrav`.
- Zaškrtněte rámeček **Forever** (Trvale) v editovacím okně.



Zaškrtávací rámeček **Forever** způsobí, že jakmile je tlačítko stisknuto, opakuje se příkaz stále dokola, nikoli jen jednou.

- Potom přidejte do panelu **Procedures** proceduru go (START):

```
to go
  move-turtles
end
```

Co je však `move-turtles`? Je to primitivum (jinými slovy, vestavěná součást NetLoga) podobně jako `clear-all`? Není, je to další procedura, kterou teprve přidáte. Zatím jste se naučili a vyzkoušeli, jak přidat dvě procedury: `setup` (PŘIPRAV) a go (START).

- Za proceduru go přidejte proceduru `move-turtles`:

```
to go
  move-turtles
End

to move-turtles
  ask turtles [
    right random 360
    forward 1
  ]
end
```

Všimněte si, že kolem pomlčky v `move-turtles` nejsou žádné mezery. V Tutorialu 2 jsme pro odečtení hodnoty používali `red - 2` s mezerami, ale v tomto případě chceme proceduru bez mezer. Pomlčka nám tu spojuje „move“ (pohnout) a „turtles“ (želvami) dohromady.

Následuje popis, co každý jednotlivý příkaz v proceduře `move-turtles` dělá:

- `ask turtles [ ... ]` říká, že každá želva by měla vykonat příkaz v závorkách.
- `right random 360` je další příkaz, který používá reportér. Nejdříve každá želva zvolí náhodné celé číslo od 0 do 359. (`random N` vrací náhodné číslo mezi nulou a N-1.) Želva se následovně otočí o tento počet stupňů.
- `forward 1` způsobí, že želva udělá jeden krok dopředu.

Proč jsme psali tyto příkazy do zvláštní procedury a nenapsali jsme je rovnou do START? Mohli jsme to udělat, ale během vývoje modelu se dost pravděpodobně stane, že budete muset přidat ještě další části. Proto bychom měli proceduru START nechat co nejjednodušší, aby byla dobře pochopitelná. Na konci bude obsahovat ještě hodně dalších příkazů podle toho, co budete chtít, aby se v průběhu modelování dělo, jako např. výpočty nebo zobrazení výsledků v grafu. Každá tato součást bude mít svoji vlastní proceduru a každá procedura bude mít své unikátní jméno.

Tlačítko START, jež jste vytvořili v panelu **Interface**, je trvalé, tj. příkazy jím spuštěné budou probíhat pořád dokola, dokud ho nevypnete (znovu na něj nekliknete). Po stisknutí tlačítka PŘIPRAV, kterým

jste vytvořili želvy, stiskněte tlačítko START. Sledujte, co se stane. Znovu na něj klikněte a uvidíte, že se všechny želvy zastaví.

Všimněte si, že svět je cyklický – pokud želva vypadne přes okraj světa, objeví se na druhé straně. (Toto chování je nastaveno jako výchozí, ale lze ho změnit, viz podkapitola Topologie v Průvodci programováním.)

Experimentování s příkazy

Doporučujeme vám, abyste si vyzkoušeli další příkazy pro želvy.

Příkazy pište do příkazového panelu (podobně jako `turtles> set color red`) nebo je přidejte do procedur `setup` (PŘIPRAV), `go` (START) a `move-turtles`.

Všimněte si, že když zadáváte příkazy v příkazovém panelu, musíte z pop-up menu vlevo vybrat `turtles>`, `patches>` nebo `observer>` v závislosti na tom, jací agenti mají příkaz provést. Funguje to stejně jako příkazy `ask turtles` nebo `ask patches`, ale nemusíte toho tolik vypisovat. Ještě pohodlnější způsob může být použití tabelátoru pro přepínání mezi agenty.

Zkuste si do příkazového panelu napsat `turtles> pen-down` a stiskněte tlačítko START.

Také zkuste v proceduře `move-turtles` změnit `right random 360` na `right random 45`.

Experimentujte a hrajte si. Je to jednoduché a výsledky vidíte okamžitě – což je jedna z mnoha předností NetLoga.

Máte-li pocit, že už jste si toho vyzkoušeli dost, pokračujte dále ve vyvíjení modelu.

Políčka a proměnné

V této fázi máme 100 želv, které se nám bezcílně potulují, aniž by si všímaly čehokoliv kolem. Mnohem zajímavější bude, když želvám přidáme pozadí, na němž se budou pohybovat.

- Vraťte se k proceduře `setup`. Můžeme ji přepsat následujícím způsobem:

```
to setup
  clear-all
  setup-patches
  setup-turtles
end
```

- Nová definice procedury `setup` odkazuje ke dvěma novým procedurám. Abychom definovali `setup-patches` (připravit políčka), musíme přidat toto:

```
to setup-patches
  ask patches [ set pcolor green ]
end
```

Procedura `setup-patches` nastaví pro každé políčko zelenou barvu. (Nezapomeňte, že pro

želvu máme proměnnou `color`, pro políčko `pcolor`.)

Poslední nedefinovaná část v naší nové proceduře `setup` je `setup-turtles` (připravit želvy).

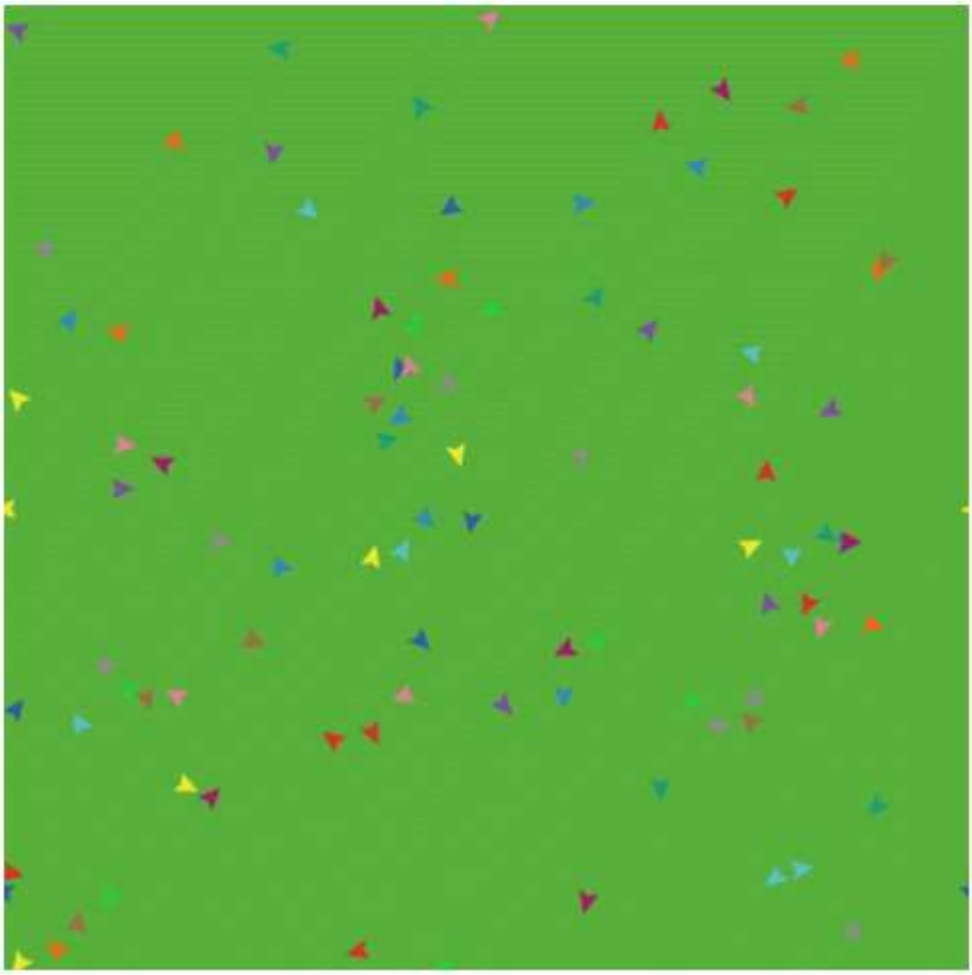
- Přidejte tuto proceduru:

```
to setup-turtles
  create-turtles 100
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Všimli jste si, že nová procedura `setup-turtles` vypadá skoro stejně jako stará procedura `setup`?

- Vraťte se do panelu Interface.
- Stiskněte tlačítko PŘIPRAV.

A vida! Objeví se krásná svěží krajina NetLoga s želvami a zelenými políčky:



Spusťte si novou proceduru `setup` tlačítkem PŘIPRAV několikrát za sebou a znovu si pročtěte její definici.

Proměnné želvy

Po krajině nám pobíhá několik želv, ale nic jiného nedělají. Přidejme trochu interakce mezi želvami a políčky.

Zařídíme, aby želvy spásaly „trávu“ (zelená políčka), množily se a umíraly. Tráva poté, co je sežrána, postupně dorůstá.

Potřebujeme zvolit způsob kontroly nad tím, kdy se želva rozmnoží a kdy zemře. To určíme pomocí energie, která každé želvě zbývá. Musíme tedy přidat novou proměnnou želvy.

Už jste měli možnost vidět vestavěné proměnné želvy jako např. `color`. Novou proměnnou vytvoříme tak, že do pole v horní části panelu **Procedures** umístíme na začátek před všechny procedury deklaraci **turtles-own**. Nazvěme ji „`energy`“ (energie):

```
turtles-own [energy]
  to go
  move-turtles
  eat-grass
end
```

Nově definovanou proměnnou `energie` použijeme k tomu, abychom želvám dovolili jíst.

- Přepněte do panelu **Procedures**.
- Přepište proceduru `go` (START) následujícím způsobem:

```
to go
  move-turtles
  eat-grass
end
```

- Přidejte novou proceduru `eat-grass` (jíst trávu):

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + 10)
    ]
  ]
end
```

Poprvé používáme příkaz `if`. Prohlédněte si pořádně kód programu. Při vykonávání příkazů porovná každá želva hodnotu barvu políčka, na kterém právě stojí (`pcolor`), s hodnotou pro zelenou (`green`). (Želva má přístup k proměnné políčka, na kterém stojí.) Je-li barva políčka zelená, výsledek porovnání se vrátí jako pravdivý (`true`) a pouze tehdy provede želva příkazy uprostřed závorek (v jiném případě je přeskočí). Příkazy způsobí, že želva změní barvu políčka na černou a zvýší si energii o hodnotu 10.

Políčko zčerná, aby bylo vidět, že na něm je tráva spasena, a želva získá víc energie tím, že se najedla.

A teď přidáme příkaz, díky kterému, když se želva pohne, spotřebuje energii.

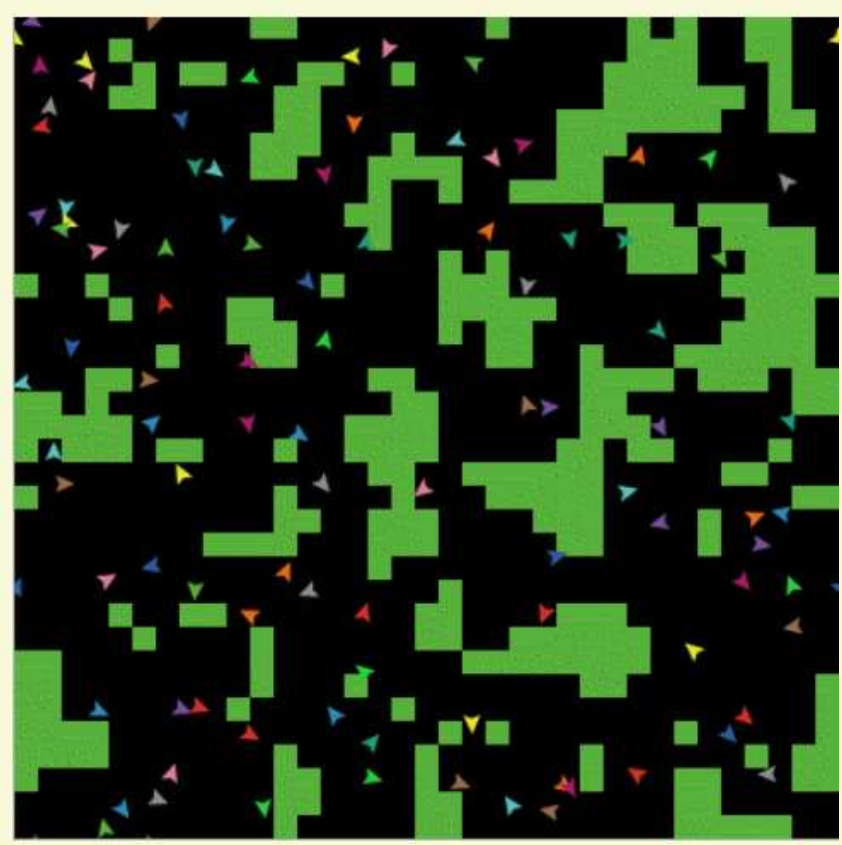
- Přepište proceduru `move-turtles` (posuň želvy) následujícím způsobem:

```
to move-turtles
  ask turtles [
    right random 360
    forward 1
    set energy energy - 1
  ]
end
```

Nyní každé želvě po každém kroku ubude jedna jednotka energie.

- Přepněte do panelu Interface a stiskněte tlačítko PŘIPRAV a START.

Uvidíte, jak políčka poté, co přes ně přejdou želvy, zčernají.



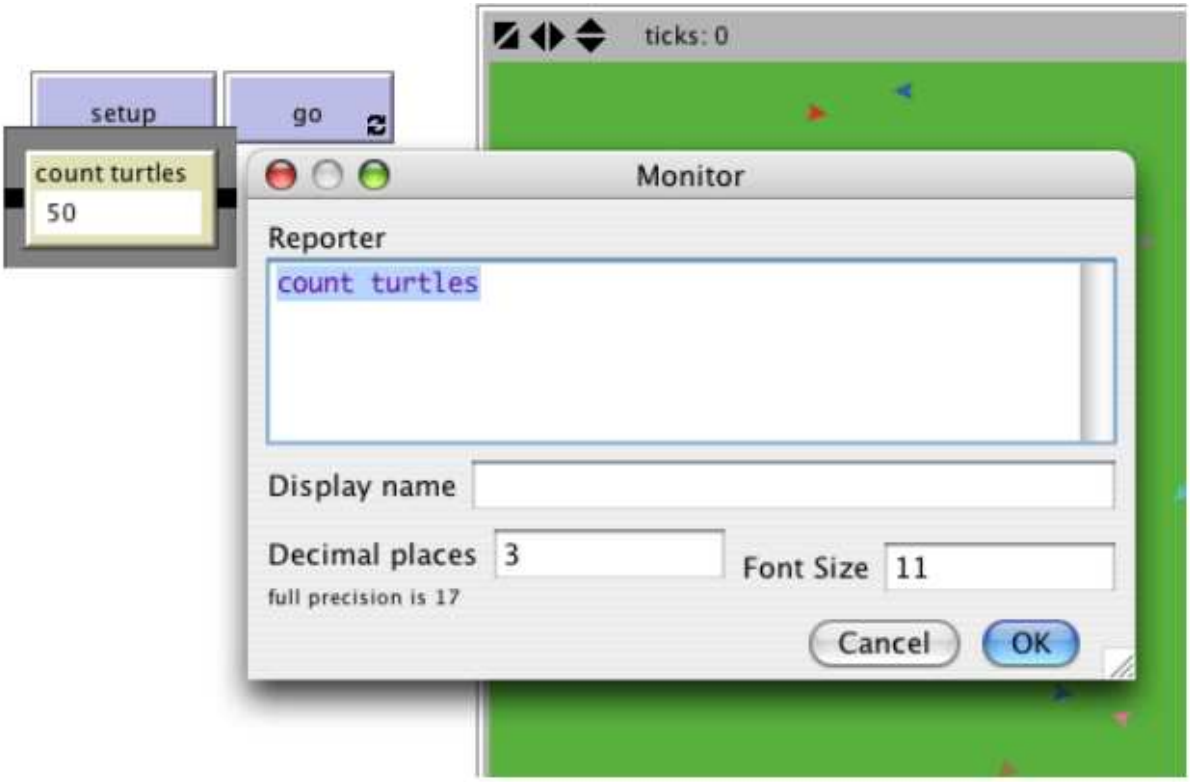
Ukazatele

V této části si vytvoříme v panelu **Interface** dva ukazatele. (Stejným postupem jako tlačítka a posuvníky, a to pomocí ikonky ukazatele na liště.) Zkusme si nyní vytvořit první.

- Vytvořte ukazatel pomocí ikonky Monitor (Ukazatel) umístěné na liště a klikněte na prázdnou plochu okna.

Otevře se dialogové okno.

- Do okna napište: `count turtles` a do pole **Display name** (Zobrazit jméno) napište: `Počet želv` (viz obrázek níže).
- Stiskněte tlačítko OK a okno se zavře.



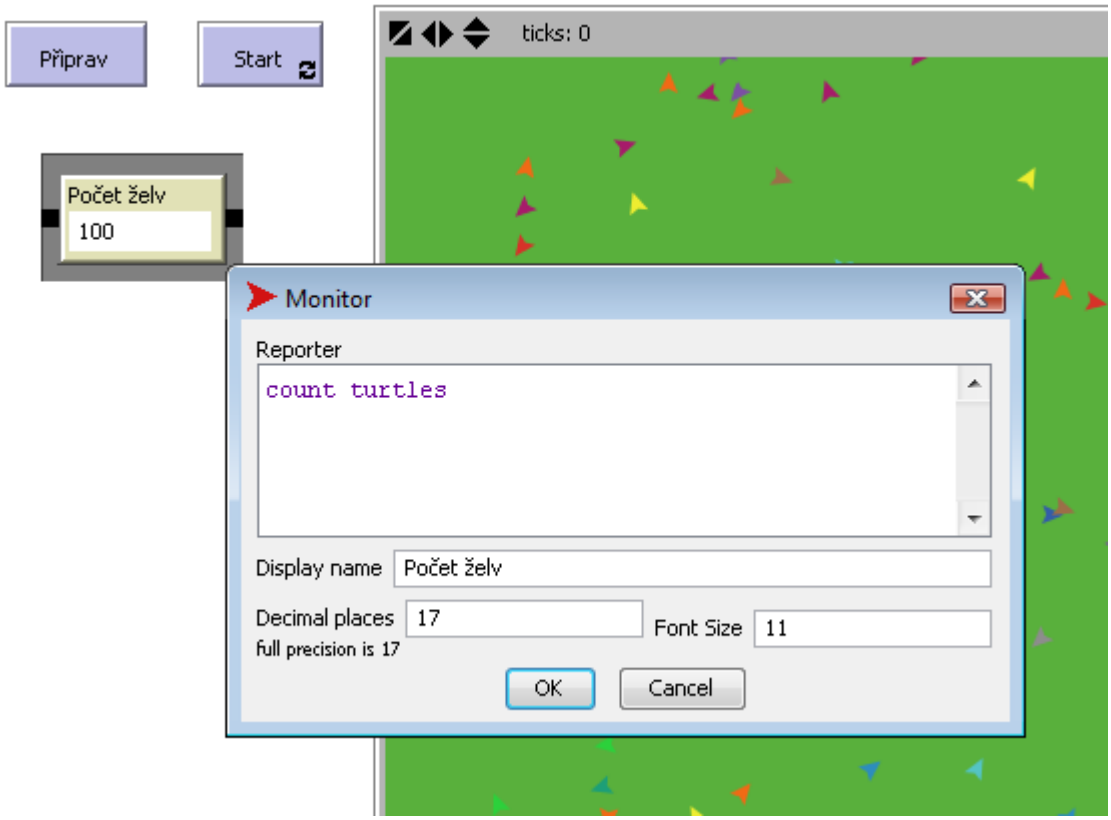
Želvy jsou „množinou agentů“, množinou želv. `count` nám řekne, kolik agentů skupina obsahuje.

Vytvoříme teď druhý ukazatel:

- Vytvořte ukazatel pomocí ikonky Monitor umístěné na liště a klikněte na prázdnou plochu okna.

Otevře se dialogové okno.

- Do pole Reporter (Reportér) dialogového okna napište: `count patches with [pcolor = green]` (viz obrázek níže).
- Do pole **Display name** (Zobrazit jméno) napište: `Zelená políčka.`
- Stiskněte tlačítko OK a okno se zavře.



Používáme zde znovu příkaz `count`, abychom viděli, kolik agentů je v dané množině. `patches` je soubor všech políček, nezajímá nás však jejich celkové množství, ale jen počet zelených. Příkaz `with` nám vybere podmnožinu pouze těch agentů, pro které platí podmínka v závorkách. Podmínka je `pcolor = green`, takže výsledkem bude počet pouze zelených políček, tj. trávy.

Nyní máme dva ukazatele, které nám řeknou, kolik želv a jaké množství trávy máme, a můžeme tak snáze sledovat, co se děje. V průběhu modelu se čísla na ukazatelích automaticky mění.

- Použijte tlačítka PŘIPRAV a START a sledujte, jak se na ukazatelích mění hodnoty.

Přepínače a popisky

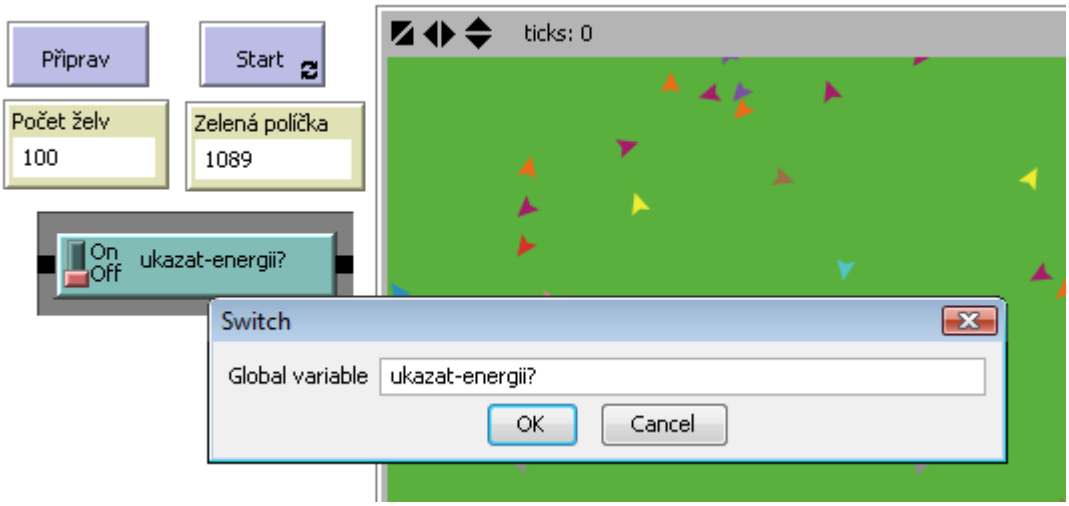
Želvy, kromě toho, že začerňují políčka, také získávají a ztrácejí energii. Sledujte v průběhu modelu monitor želvy a její energii, jak se zvyšuje a snižuje.

Bylo by hezké, kdybychom viděli energii všech želv najednou. Nyní to umožníme přidáním přepínače, který nám umožní zapínat a vypínat přídavné informace.

- Vytvořte přepínač pomocí ikonky Switchu (Přepínač) umístěné na liště (v panelu **Interface**) a klikněte na prázdnou plochu okna.

Otevře se dialogové okno.

- Do pole Global variable (Globální proměnná) dialogového okna napište: `ukazat-energii?` Nezapomeňte na otazník. (Viz obrázek níže.)



- Nyní se vraťte pomocí lišty v panelu **Procedures** do procedury go (START).
- Přepište proceduru `eat-grass` (sněz trávu) následujícím způsobem:

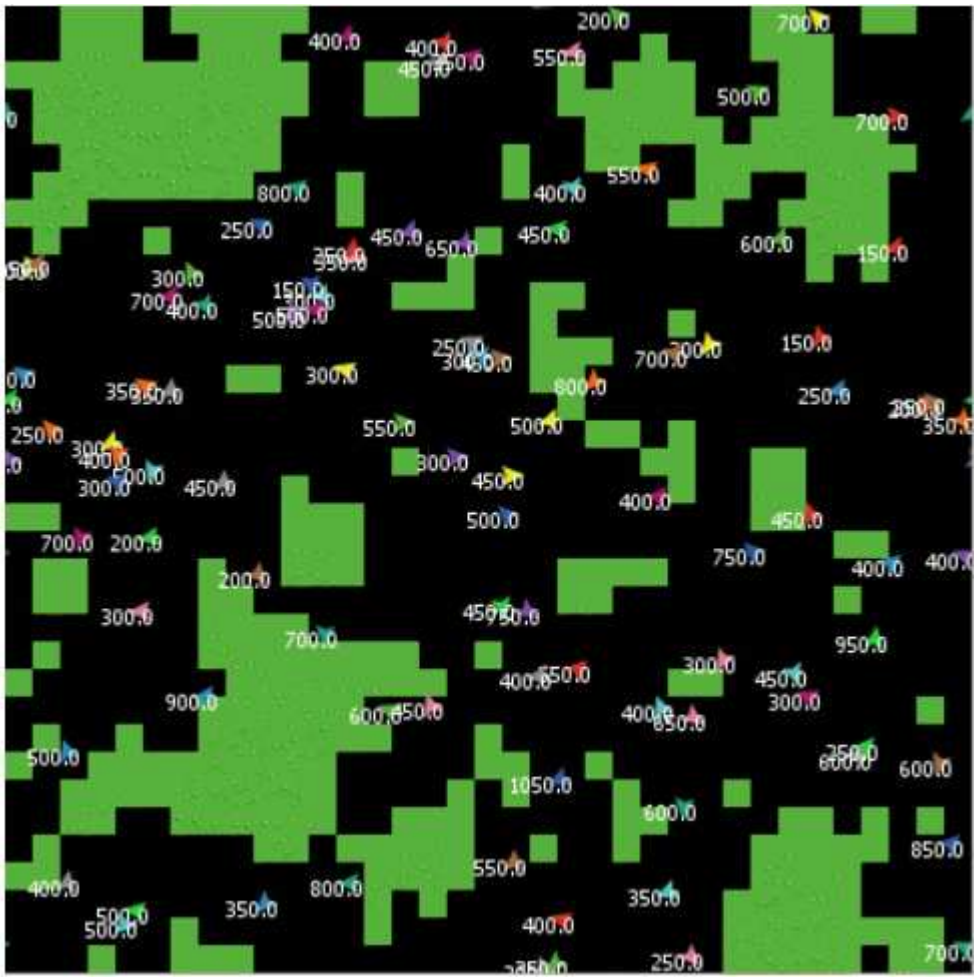
```
to eat-grass
   ask turtles [
      if pcolor = green [
         set pcolor black
         set energy (energy + 10)
      ]
   ifelse ukazat-energii?
     [ set label energy ]
     [ set label "" ]
   ]
end
```

V proceduře `eat-grass` se poprvé vyskytuje příkaz `ifelse`. Prohlédněte si pořádně kód programu. Při vykonávání těchto nových příkazů si každá želva zkontroluje hodnotu „`ukazat-energii?`“ (určenou přepínačem). Pokud je přepínač v poloze ON, je porovnání pravdivé a želva vykoná příkaz v prvních závorkách. V tomto případě se připíše hodnota energie do popisky želvy. Pokud je porovnání nepravdivé (přepínač je v poloze OFF), provede želva příkaz v druhých závorkách. V tomto případě se odstraní popiska (resp. je zadána nulová).

(V NetLogu se kus textu nazývá „řetězec“. Řetězec je sekvence písmen a dalších znaků, zapsaná mezi dvěma uvozovkami. Zde máme dvě uvozovky hned vedle sebe, což znamená, že řetězec je prázdný. V případě, že popiska želvy je prázdný řetězec, není k želvě přiřazen žádný text.)

- Ověřte to tak, že v panelu Interface spustíte model (pomocí tlačítek PŘIPRAV a START) a budete přepínat přepínač „`ukazat-energii?`“ tam a zpět.

Je-li přepínač v poloze ON, uvidíte, jak jeho hodnota stoupne pokaždé, kdy želva spase trávu. Její energie naopak klesne pokaždé, kdy vykoná pohyb.



Další procedury

Nyní jsme tedy dosáhli toho, že se nám želvy pasou, tak je ještě necháme rozmnožovat a umírat. A rovněž chceme, aby dorůstala tráva. Tato chování přidáme pomocí samostatných procedur, pro každé chování jednu.

- Přepněte do panelu Procedures.
- Přepište proceduru go (START) následujícím způsobem:

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
end
```

- Přidejte procedury reproduce, check-death a regrow-grass, jak je popsáno níže:

```
to reproduce
```

```
    ask turtles [
      if energy > 50 [
        set energy energy - 50
        hatch 1 [ set energy 50 ]

      ]
    ]
  end

  to check-death
    ask turtles [
      if energy <= 0 [ die ]
    ]
  end

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

Všechny tyto procedury používají příkaz `if`. Každá želva, když má vykonat příkaz `reproduce`, si zkontroluje proměnnou `energy`. Pokud je její hodnota větší než 50, vykoná želva příkazy v prvních závorkách, tj. její energie klesne o 50 a „vysadí“ novou želvu s počáteční energií 50. Příkaz `hatch` je primitivum NetLoga, které vypadá takto: `hatch` *počet[příkazy]*. Želva vytvoří *počet* nových želv, všechny identické s rodičem, a požádá nové želvy (novou želvu), aby provedly *příkazy*. Novým želvám můžeme přikázat, aby změnily barvu, směr otočení atd. V tomto případě jsme zadali jediný příkaz – energie nově vysazených želv je 50 jednotek.

Příkaz `check-death` způsobí, že každá želva zkontroluje, zda její energie je menší nebo rovna 0. Pokud je výsledek pravdivý, želva dostane příkaz, aby zemřela: `die` (die je primitivum NetLoga).

Když políčko provádí příkaz `regrow-grass`, zkontroluje, zda je náhodné celé číslo od 0 do 99 menší než 3. Pokud ano, políčko zezelená. Stane se tak průměrně ve 3 % případů, protože pouze tři čísla (0, 1, 2) ze 100 splňují podmínku, že jsou menší než 3.

- Přepněte do panelu Interface a stiskněte tlačítka PŘIPRAV a START.

Teď by se měl model chovat zajímavěji. Některé želvy zemřou, jiné vzniknou (jsou vysazeny) a doroste trochu trávy. Docílili jsme toho našimi příkazy.

Sledujte model a všimněte si, že ukazatele želv a trávy kolísají. Lze předvídat schéma tohoto kolísání? Je mezi proměnnými nějaký vztah?

Hodilo by se nám, kdybychom mohli sledovat, jak se chování modelu časem mění. NetLogo umožňuje znázornit data z průběhu modelu. To bude naším dalším úkolem.

Grafické znázornění

Abychom mohli sledovat dosavadní běh modelu, potřebujeme vytvořit graf v panelu **Interface** a nastavit ho. Potom přidáme další proceduru do panelu **Procedures**, která nám bude graf aktualizovat.

Vytvořme si nejdřív novou proceduru.

- Změňte proceduru setup (PŘIPRAV), aby volala proceduru `do-plots` (vykresli), již teprve přidáme:

```
to setup
  clear-all
  setup-patches
  setup-turtles
  do-plots
end
```

- Změňte také proceduru START, aby volala proceduru `do-plots` (vykresli):

```
to go
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  do-plots
end
```

- Přidejte novou proceduru. Budeme chtít vytvořit graf znázorňující počet želv a množství trávy v průběhu času. Při každém kroku – jednom kole procedury `go` (START) – se tyto hodnoty objeví v grafu.

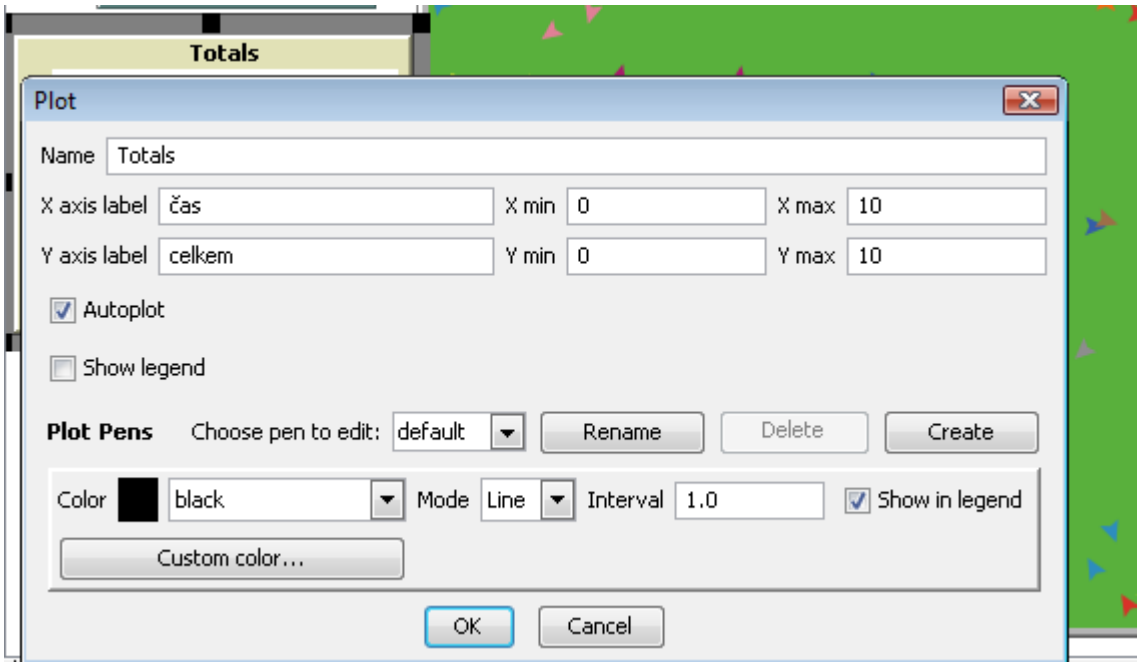
```
to do-plots
  set-current-plot "Totals"
  set-current-plot-pen "turtles"
  plot count turtles
  set-current-plot-pen "grass"
  plot count patches with [pcolor = green]
end
```

Všimněte si, že abychom přidali do grafu další bod, používáme příkaz `plot`. Předtím však ještě musíme NetLogu sdělit dvě věci. Za prvé musíme určit, jaký graf budeme používat (protože později můžeme mít grafů více), a dále musíme rozhodnout, kterým perem budeme kreslit (v tomto grafu použijeme dvě).

Příkaz `plot` posune aktuální pero (pen) bodu umístěného na souřadnici X o jeden bod dále, než je stávající pozice, a na souřadnici Y do místa, kterému odpovídá hodnota zadaná v příkazu (v prvním případě to je počet želv, v druhém případě množství trávy). Pera se posunou a nakreslí křivku.

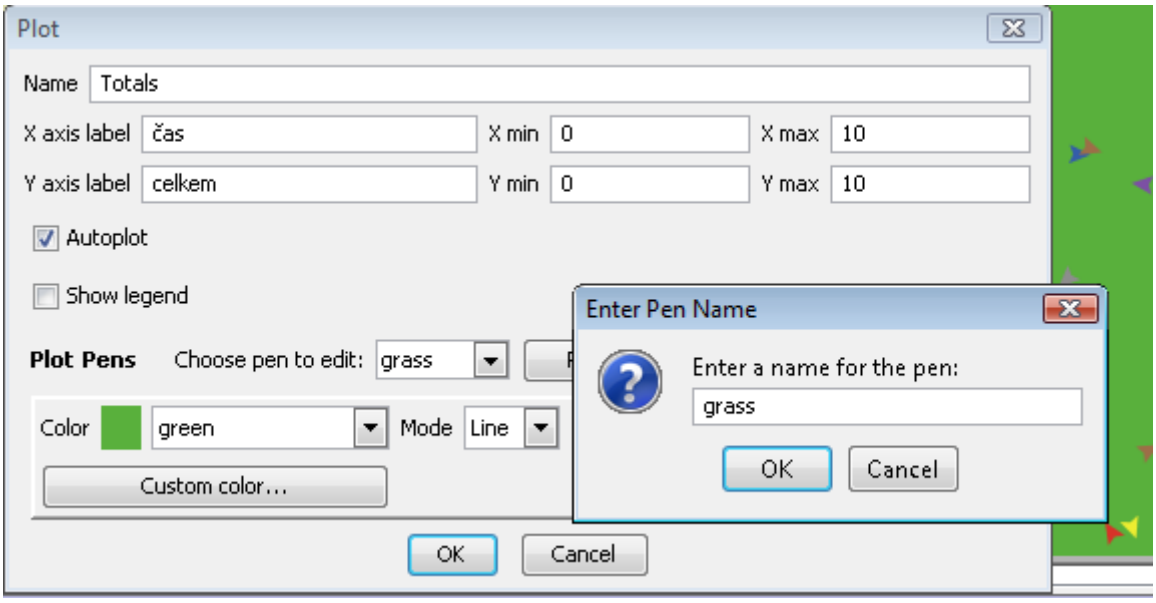
Aby nám příkaz `set-current-plot "Totals"` fungoval, musíme přidat graf do modelu pomocí panelu **Interface** a upravit ho tak, aby jeho jméno bylo shodné se jménem v procedurách. I jedna mezera navíc může způsobit nefunkčnost – název se musí shodovat na obou místech.

- Vytvořte graf pomocí ikonky **Plot** (Graf) umístěné na liště v panelu Interface a klikněte na prázdnou plochu okna.
- Pojmenujte ho `Totals` (Celkový počet) (viz obrázek níže).
- Pojmenujte osu X čas.
- Pojmenujte osu Y celkem.



Dále potřebujete vytvořit dvě pera.

- V otevřeném okně **Plot** (graf) stiskněme tlačítko **Create** (Vytvořit) a vytvoříme tak nové pero.
- Do pole „Enter pen name“ (Vložte název pera) napište `turtles` a stiskněte OK (viz níže).
- Stiskněte znovu tlačítko **Create** (vytvořit) v okně **Plot** (graf) a vytvořte druhé pero.
- Pomocí pole Enter pen name ho pojmenujte „grass“ a stiskněte OK (viz níže).
- V color zvolte barvu pera a změňte ji na zelenou (green).
- Stiskněte OK.

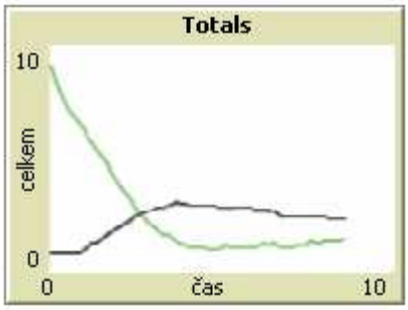


Všimněte si, že když vytváříte graf, můžete také zadat minimální a maximální hodnoty na osách X a y. Zaškrtněte rámeček **Autoplot?**, takže pokud během vykreslování grafu přerostou hodnoty os minimum a maximum, osy se automaticky zvětší a uvidíte všechna data.

- Stiskněte znovu tlačítka PŘIPRAV a START.

Nyní můžete v průběhu modelu sledovat graf. Měl by mít obecně tvar jako na obrázku níže, i když nemusí vypadat úplně stejně.

Tím, že jsme nechali zaškrtnutou funkci Autoplot?, tak si graf v případě, že mu dojde místo, sám přenastaví měřítko.



Pokud jste zapomněli, které pero je které, klikněte na popisku **Pens** v pravém horním rohu. Nechte model běžet několikrát a sledujte, v čem se graf mění a v čem ne.

Počítadlo (Tick Counter)

Někdy chceme sledovat rozdíly mezi jednotlivými běhy modelu, a proto by se nám hodilo, aby model běžel stejně dlouho. Potřebujeme vědět, jak spustit a zastavit akci v konkrétním časovém bodě, abychom mohli model zastavit ve stejném bodě. To zjistíme tak, že budeme počítat, kolikrát proběhla procedura START.

Počet kol procedury `go` (START) nám zaznamenává vestavěné počítadlo.

- Změňte proceduru `go` (START):

```
to go
  if ticks >= 500 [ stop ]
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick
  do-plots
end
```

- Stiskněte znovu tlačítka PŘIPRAV a START.

Graf a model nepoběží pořád, zastaví se automaticky, až se na počítadle na liště v panelu **Interface** objeví hodnota 500.

Příkaz `tick` posunuje počítadlo o 1. `ticks` je reportér, který vrací aktuální hodnotu počítadla. `clear-all` vynuluje počítadlo na začátku nového kola.

Všimněte se, že příkaz `tick` je umístěn před `do-plots`. Když tvoříme kód grafu, ve kterém je použita hodnota z počítadla, musíme ho nejdřív načíst, jinak by se zobrazila stará hodnota. (V tomto tutorialu sice žádný kód ve skutečnosti nepíšeme, ale obecně platí, že je dobré volat `tick` poté, co agenti dokončili své úlohy, ale než se vykreslí graf.)

Když už náš model umí počítat kroky, měli bychom změnit nastavení v horní části panelu **Interface** – přepneme aktualizaci z **continuous** (kontinuální) na **tick-based** (po jednotlivých krocích). To znamená, že zobrazení NetLoga (pohled do světa agentů) se bude aktualizovat (tj. překreslovat) mezi kroky, ale nikoliv uprostřed kroku. Výsledkem bude, že model poběží rychleji a bude se zobrazovat pravidelně (na základě pravidelných kroků). Více informací naleznete v Průvodci programováním.

Další podrobnosti

Nejprve bychom měli zmínit, že místo 100 želv můžete používat různý počet želv.

- Pomocí ikonky Monitor na liště vytvořte posuvník proměnné nazvané `number` (počet) a klikněte na prázdnou plochu okna. Zkuste si změnit nastavení minimální a maximální hodnoty.
- Do procedury `setup-turtles` (připravit želvy) napíšeme místo `create-turtles 100` např. toto:

```
to setup-turtles
  create-turtles number
  ask turtles [ setxy random-xcor random-ycor ]
end
```

Vyzkoušejte si různé počty želv a srovnejte, jak větší/menší počáteční počet želv ovlivní graf v průběhu času.

Za druhé, nelíbilo by se vám, kdybyste si mohli nastavit energii, kterou želvy získávají pasením a ztrácejí množením?

- Vytvořte posuvník nazvaný `energy-from-grass` (výnos energie z trávy).
- Vytvořte další posuvník nazvaný `birth-energy` (energie želvy při narození).
- Potom v proceduře eat-grass proveďte tuto změnu:

```
to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
      set energy (energy + energy-from-grass)
    ]
    ifelse ukazat-energii?
     [ set label energy ]
     [ set label "" ]
  ]
End
```

- A v proceduře `reproduce` (reprodukce) tuto změnu:

```
to reproduce
  ask turtles [
    if energy > birth-energy [
      set energy energy - birth-energy
      hatch 1 [ set energy birth-energy ]
    ]
  ]
End
```

Jaký další posuvník byste přidali, abyste mohli měnit i rychlost růstu trávy? Napište nějaká pravidla, která by se dala přidat k pohybu (nových) želv a která by se děla pouze v určitém časovém bodě.

Co dál?

Nyní jste vytvořili jednoduchý model ekosystému. Na políčkách roste tráva, želvy chodí po světě a spásají ji, množí se a umírají. Vyvinuli jste rozhraní s tlačítky, posuvníky, přepínači, ukazateli a grafy. Napsali jste dokonce řadu procedur, aby želvy něco dělaly.

Na tomto místě končí i tento tutorial.

Následující dokumentace k NetLogu se zabývá těmito tématy: kapitola Průvodce rozhraním vám ukáže postupně všechny části rozhraní NetLoga a vysvětlí jejich funkce. Detailní popis a specifika psaní procedur najdete v Průvodci programováním. Seznam a popis primitiv najdete ve Slovníčku NetLoga.

Pokud chcete, pokračujte v experimentování a nově vytvořený model ještě rozšiřte. Zkuste agentům nastavit různé proměnné a chování.

Také se můžete vrátit k našemu prvnímu modelu v Tutorialu 1, Vlkům a ovcím. V tomto modelu jste mohli sledovat, jak se ovce pohybují, spásají zdroje (trávu), které jsou občas znovu doplněny, za určitých podmínek se rozmnožují, a jestliže jim dojdou zdroje, tak umírají. V tomto modelu se však vyskytovali ještě další tvorové – vlci. Chcete-li přidat vlky, musíte doplnit další procedury a několik nových primitiv. Vlci a ovce jsou dva různé „rody“ (breeds) želv. Prostudujte si model Vlci a ovce a dozvíte se, jak používat rody.

Dále se můžete podívat na ostatní modely (včetně modelů v oddílu **Code Examples** v **Models Library**) nebo pokračujte a vyviňte si vlastní model. Nebo ani nic nemodelujte a jen sledujte, jak políčka a želvy vytvářejí určité struktury, a vymyslete si např. nějakou hru.

Doufáme, že jste se dozvěděli a naučili mnoho nových věcí, nejen jak si postavit vlastní model, ale i jak NetLogo funguje. Následuje kompletní řada procedur, kterou jsme vytvořili výše.

Příloha: Úplný kód modelu

Úplný model je k dispozici rovněž v **Models Library**, v oddílu **Code Examples**, najdete ho pod názvem Tutorial 3.

Následující seznam je doplněn o „komentáře“, které začínají středníky. Tyto vysvětlivky lze doplnit přímo ke kódu programu. Komentáře můžete použít, když chcete ostatním vysvětlit svůj model nebo si tak můžete dělat poznámky pro sebe.

V panelu **Procedures** jsou komentáře šedivé, takže je snadno uvidíte.

```
turtles-own [energy] ;; for keeping track of when the turtle is ready
                     ;; to reproduce and when it will die

to setup
  clear-all
  setup-patches
  setup-turtles
  do-plots
end

to setup-patches
  ask patches [ set pcolor green ]
end

to setup-turtles
  create-turtles number     ;; uses the value of the number slider to
create turtles
  ask turtles [ setxy random-xcor random-ycor ]
end

to go
  if ticks >= 500 [ stop ]  ;; stop after 500 ticks
  move-turtles
  eat-grass
  reproduce
  check-death
  regrow-grass
  tick                      ;; increase the tick counter by 1 each time
through
  do-plots
end

to eat-grass
  ask turtles [
    if pcolor = green [
      set pcolor black
           ;; the value of energy-from-grass slider is added to energy
      set energy (energy + energy-from-grass)
    ]
  ifelse ukazat-energii?
    [ set label energy ] ;; the label is set to be the value of the
energy
    [ set label "" ]     ;; the label is set to an empty text value
  ]
```

```
end

to reproduce
  ask turtles [
    if energy > birth-energy [
    set energy energy - birth-energy  ;; take away birth-energy to give
birth
    hatch 1 [ set energy birth-energy ] ;; give this birth-energy to the
offspring
    ]
  ]
end

to check-death
  ask turtles [
    if energy <= 0 [ die ] ;; removes the turtle if it has no energy left
  ]
end

to regrow-grass
  ask patches [ ;; 3 out of 100 times, the patch color is set to green
    if random 100 < 3 [ set pcolor green ]
  ]
end

to do-plots
  set-current-plot "Totals" ;; which plot we want to use next
  set-current-plot-pen "turtles" ;; which pen we want to use next
  plot count turtles ;; what will be plotted by the current pen
  set-current-plot-pen "grass" ;; which pen we want to use next
  plot count patches with [pcolor = green] ;; what will be plotted by the
current pen
end
```

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.