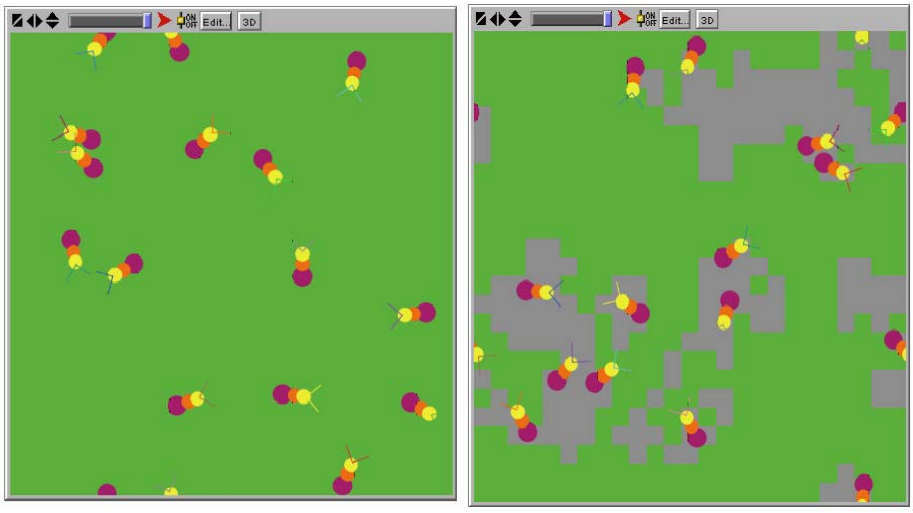
Tutorial

Exploração do ambiente NetLogo



Beatriz Corso Magdalena

(beamag@terra.com.br)

Crediné Silva de Menezes

(credine@inf.ufes.br)

Edílson Pantarolo

(edi@pb.cefetpr.br)

Lívia Lopes Azevedo

(livia@cpd.ufmt.br)

Índice

Este tutorial foi baseado no tutorial disponível na homepage do NetLogo, acesse:

http://ccl.northwestern.edu/netlogo/

1. Introdução

Um importante tipo de simulação é a "*agent-based modeling*" (modelagem baseada em agentes). Esse tipo de simulação é caracterizado pela existência de muitos agentes interagindo uns com os outros, com pouca ou nenhuma direção centralizada. A propriedade de *inteligência emergente* do modelo baseado em agentes surge durante o processo de interação, de baixo para cima (*bottom-up*)[1] e não do sentido de cima pra baixo (*top-down*).

Através da modelagem baseada em agentes é possível observar como os agentes individuais (pessoas, bactérias, insetos, nações, ou organizações) interagem entre si e com seu ambiente. A simulação no computador é então usada para descobrir propriedades do modelo e, assim, ganhar entendimento dentro de um processo dinâmico, o que seria muito difícil de modelar com técnicas matemáticas padrões.

Em função disso, é importante dispor de ferramentas capazes de criar, no computador, simulações de fenômenos complexos e modelos descentralizados. Existem vários ambientes de programação especificamente projetados para modelagem baseada em agentes, entre eles StarLogo, Microworlds, Swarm, RePast e NetLogo.

2. Ambiente de programação NetLogo

O NetLogo é um ambiente de modelagem programável para explorar comportamento de sistemas descentralizados e sistemas complexos. É particularmente bem situado para modelagem de sistemas complexos desenvolvidos no decorrer do tempo. O NetLogo fornece um modo fácil de começar a modelagem baseada em agentes, mesmo para aqueles que não têm avançada habilidade em programação e matemática. O ambiente NetLogo é composto de três tipos de agentes: *turtles, patches* e *observer*. O modelador pode dar, simultaneamente, instruções a centenas ou milhares de "agentes" independentes que trabalham paralelamente, tornando possível explorar as conexões entre o comportamento de micro-níveis e os de modelos de macro-níveis que emergem das interações de muitos indivíduos. NetLogo usa uma

[1] Cada agente desenvolve regras geralmente simples e estes se auto-organizam para formar um comportamento inteligente.

linguagem de modelagem própria, derivada da linguagem de programação Logo do Media Lab – MIT.

2.1. Programação em NetLogo

A programação em NetLogo consiste em atribuir comportamentos a três grupos de agentes: o observador (*observer*), as criaturas (*turtles*) e o ambiente (*patches*). O observador é um "criador" que especifica as condições de funcionamento e dá controle ao demais elementos.

A linguagem de programação do NetLogo apresenta vários recursos, tais como: atribuir diferentes formas (*shapes)* para as *turtles* (os modelos podem ser selecionados na biblioteca do ambiente ou criados pelo próprio usuário através de uma janela de edição de modelos); definir diferentes espécies (*breeds*) para as *turtles* que tenham comportamentos diferenciados. Por exemplo, definidas as espécies denominadas "lobo" e "ovelha", através da programação é possível escrever uma regra que faça o "lobo" comer as "ovelhas". Essa diferenciação das espécies faz, também, com que o modelo fique visualmente mais atraente e esclarecedor.

Ainda com relação à programação, o NetLogo dispõe de uma característica diferenciada dos demais ambientes da mesma categoria que é denominada *Agentset,* que permite definir um conjunto de agentes. Um *Agentset* pode conter tanto *turtles* quanto *patches*, mas não ambos ao mesmo tempo. O que é importante no conceito de *Agentset* é que ele possibilita construir conjuntos de apenas algumas *turtles* ou *patches* com regras específicas. Por exemplo, todas as *turtles* com ID divisível por cinco, ou todos os *patches* com *pxcor* (valor da coordenada x) par.

3. Instalação do ambiente NetLogo

Para baixar (*download*) os arquivos de instalação do Netlogo, basta ir ao *site* http://ccl.northwestern.edu/netlogo/. No *site* há explicações do equipamento e sistema requerido para instalação do pacote. O sistema poderá trabalhar em qualquer plataforma na qual esteja instalada a máquina virtual Java (JVM –

Java Virtual Machine), versão 1.4.1 ou superior. Na versão para Windows existe a opção de baixar o pacote que já inclui a JVM necessária, o que é mais recomendado para usuários menos experientes. Sugerimos que seja escolhida a versão 3.0 (recomendada JVM 1.4.2_08), com novas características e a mais atual até o momento.

Figura 1 – Página de download do NetLogo.

O pacote de instalação inclui todos os módulos necessários, o que permite uma rápida visualização dos elementos instalados (Figura 2) e a consulta local à documentação do programa.

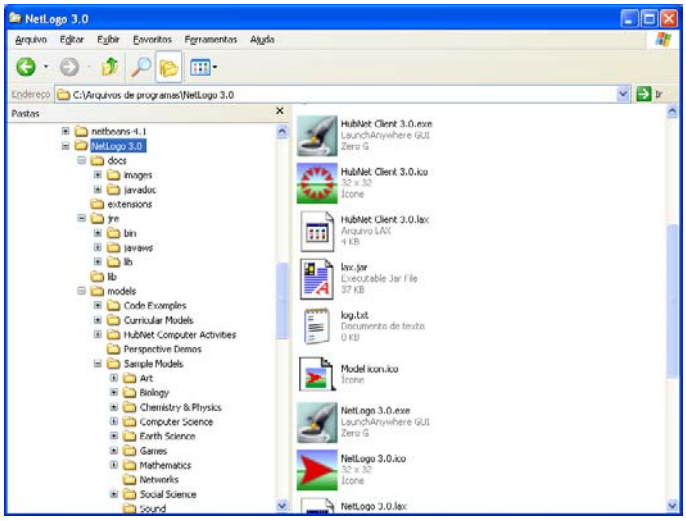


Figura 2 – Pacote NetLogo instalado.

Imediatamente após a instalação do ambiente é possível começar a trabalhar sobre ele. Quando o NetLogo é iniciado, a Interface principal é apresentada (Figura 3). Nesta janela é possível visualizar a execução dos modelos de simulação baseada em agentes. Na parte inferior da janela encontra-se a palheta ***Command Center***, que permite a edição de comandos. Para executar mais de um comando basta escrevê-los na seqüência, separados por espaço.

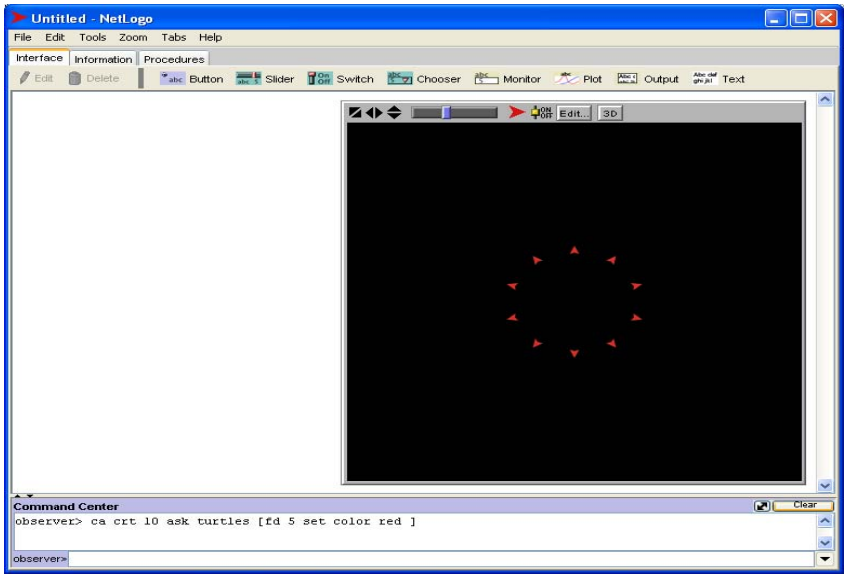


Figura 3 – Tela inicial do ambiente Netlogo.

4. Programação passo a passo

Agora vamos experimentar fazer um modelo usando comandos simples. Para fazer isso selecionamos: ***File*** seguido de ***New***. Feito isso abrirá a tela principal com a janela gráfica e a janela de comandos (***Command Center***). A janela gráfica serve para visualizar as criaturas (*turtles*) e a base do ambiente (*patches*) que elas habitam, enquanto a janela de comandos é utilizada para digitar as instruções a serem executadas pelos agentes (*turtles* e *patches*).

Command Center

Esta janela aceita os comandos em três contextos: definidos pelo *Observer;* definidos pelas *Turtles*; e definidos pelos *Patches*. Observe que a janela de comandos possibilita para o usuário uma das três opções através de um menu *drop down* (Figura 4).

Figura 4 – Janela de comandos (destaque agentes).

Segundo a instalação padrão para Windows, o ***Manual do Usuário*** fica localmente acessível a partir do arquivo ***PastaLocalNetLogo\docs\index.html***. A versão *on-line* e mais recente desse manual pode ser acessada em http://ccl.northwestern.edu/netlogo/docs.

O manual apresenta informações gerais, FAQ (*Frequently Asked Questions*), tutoriais e guias de referência sobre a interface e a programação NetLogo. Além disso, oferece um dicionário das primitivas disponíveis para interagir com o NetLogo e para a construção de programas, na forma de uma extensa relação de comandos, variáveis internas padrão dos agentes, palavras-chave e constantes.

Faremos uma breve apresentação do conteúdo deste dicionário (no Apêndice A apresentamos uma relação de comandos mais utilizados). A Figura 5 apresenta a tela inicial do dicionário de primitivas.

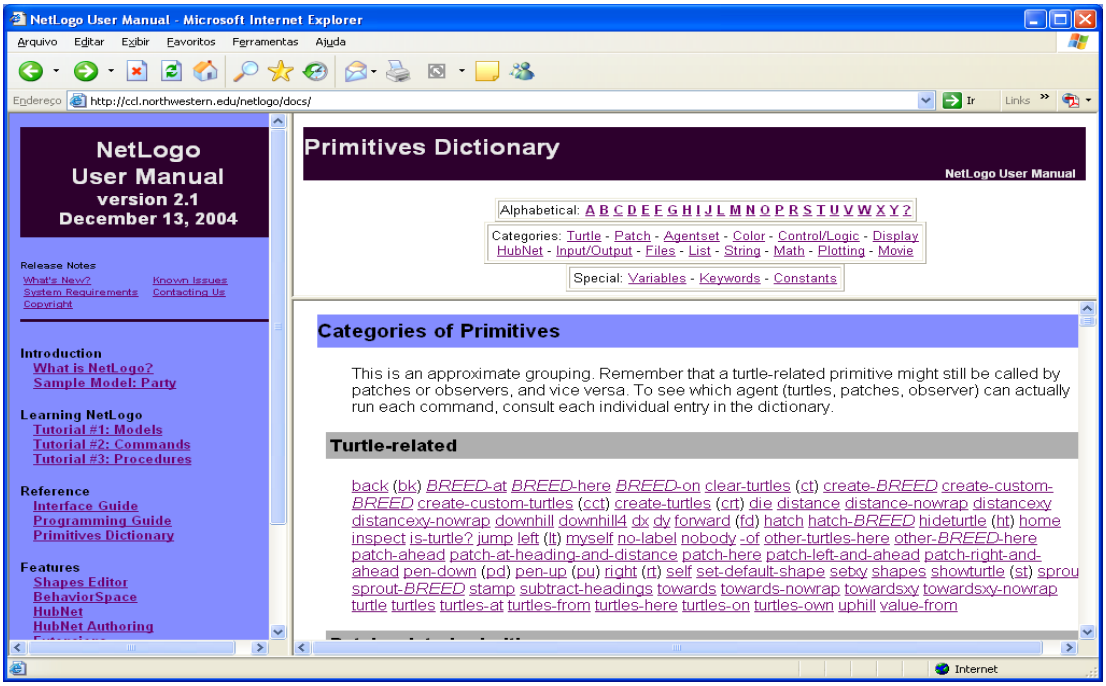


Figura 5 – Dicionário de primitivas da programação NetLogo.

Para cada comando no dicionário há uma indicação gráfica de quem pode usá-los. A indicação dos comandos de utilização do *observer* é semelhante a um olho (Figura 6). Os comandos de utilização das *turtles* são representados por uma tartaruguinha verde e os comandos de uso dos *patches* são representados por um quadradinho vermelho (Figura 7).

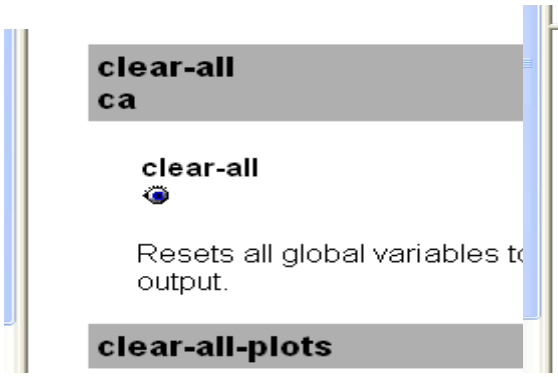


Figura 6 – Comando *observer*.

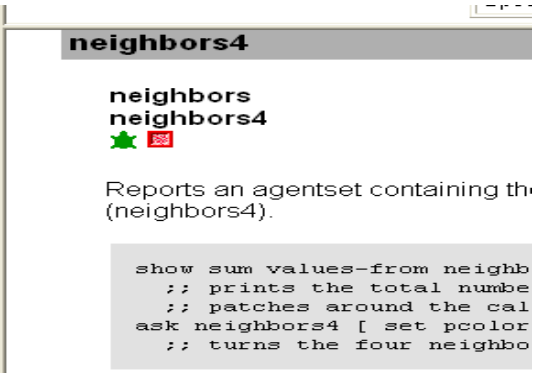


Figura 7 – Comando *turtle* / *patch*.

Vamos começar com alguns comandos. Escolha para enviar os comandos os agentes *patches*. Assim, de volta a janela principal do NetLogo, vamos a janela de comandos e digitamos: "***set pcolor (pxcor + pycor)***". Observe o que acontece (Figura 8).

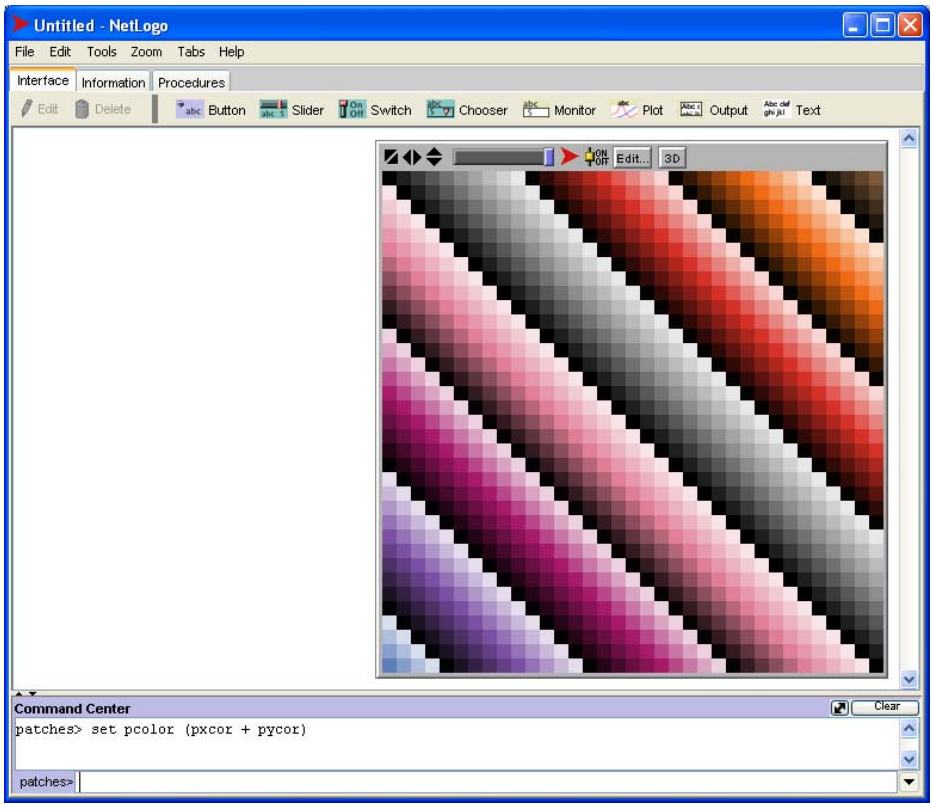


Figura 8 – Comando *patches*: cor em função da posição.

Obs. Os comandos primitivos relacionados às *turtles e patches* também podem ser chamados pelo *observer* através da solicitação
ask turtles [comandos] *ou*
ask patches [comandos],
respectivamente.

Para diferenciar os comandos e propriedades comuns a *turtles* e *patches*, os destes últimos iniciam com a letra ***p*** (exemplo: ***xcor*** é relacionado à posição da coordenada x para a *turtle*, da mesma forma ***pxcor*** é relacionado à coordenada x do *patch*).

Abaixo, recorte de um código que determina a criação de *patches* e *turtles* com suas devidas distribuições.

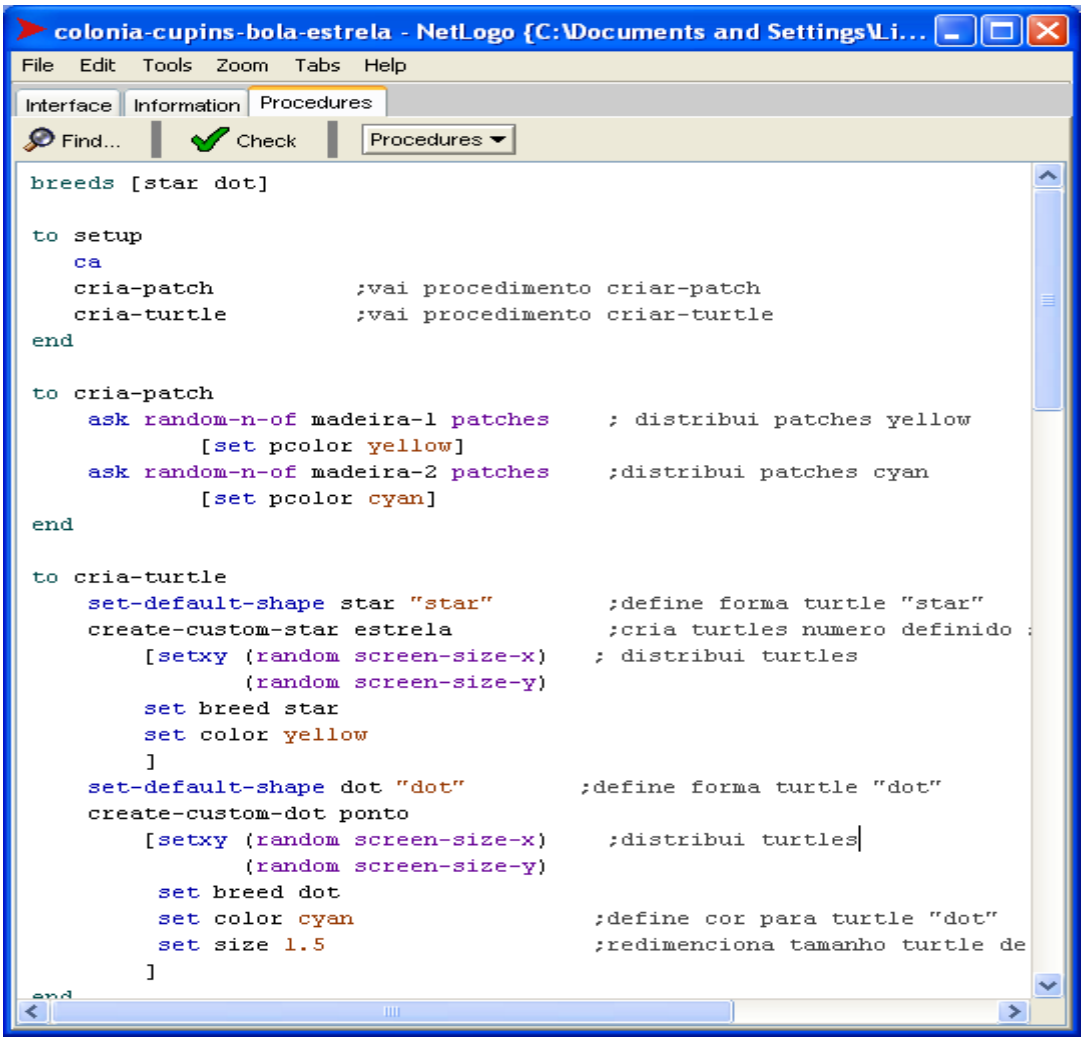


Figura 9 – Comando para definição de turtles e patches

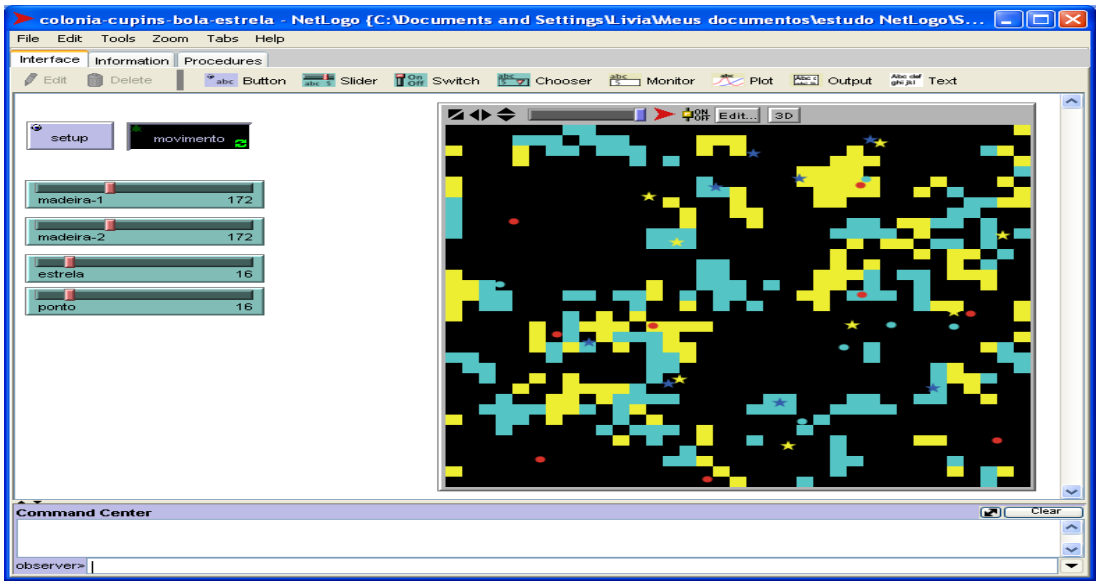


Figura 10 – Execução do código acima

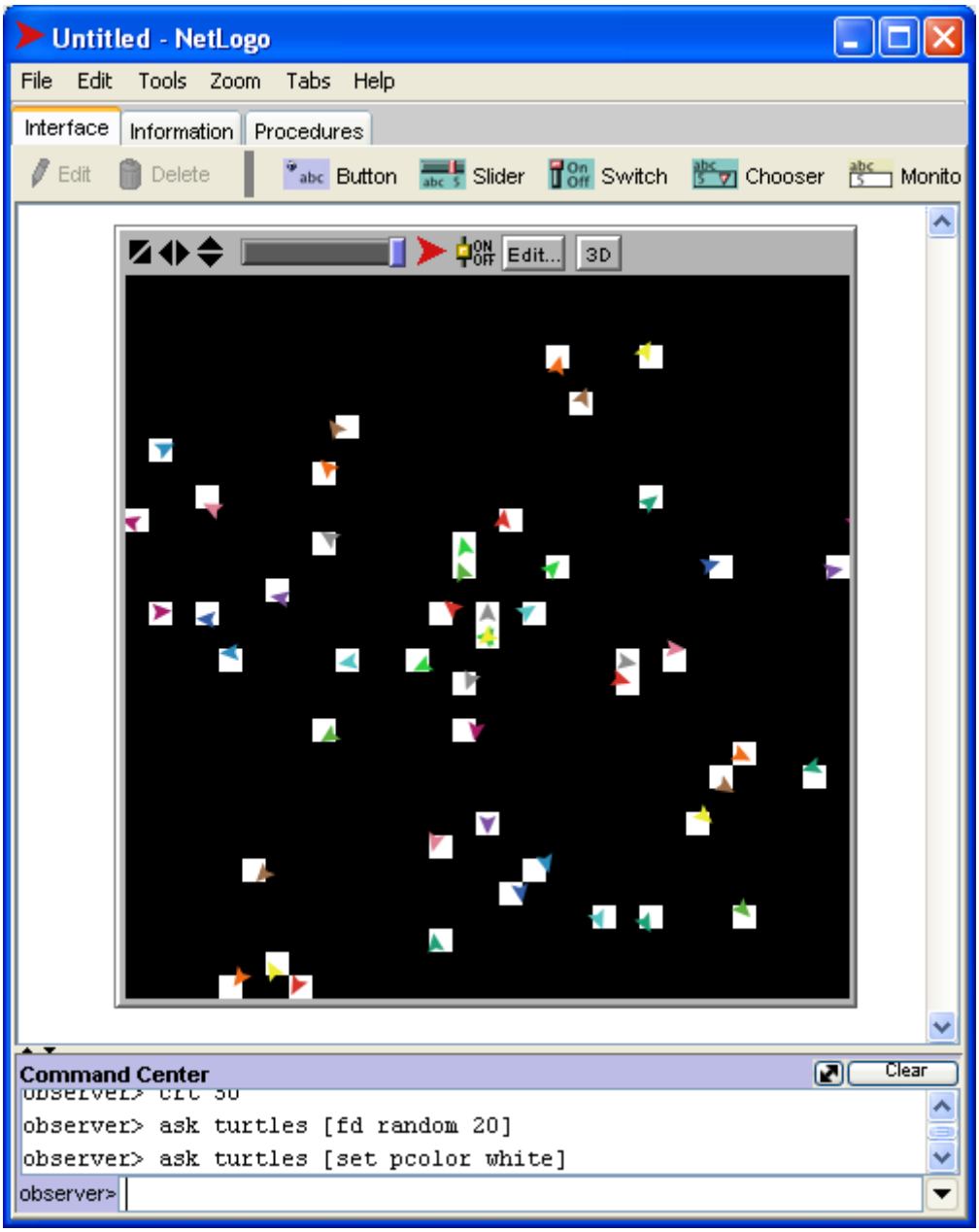


Figura 11 – Exemplo de atribuição de patches

As outras propriedades comuns a *turtles* e *patches* são: ***pcolor*** (cor), ***plabel*** (rótulo), ***plabel-color*** (cor do rótulo) e ***pycor*** (coordenada y). Além dessas, as *turtles* apresentam as seguintes propriedades exclusivas:

- ***breed***, espécie a que pertence (o padrão é ***turtles***);
- ***heading***, direção para a qual está apontada (valor entre ***0.0*** e ***360.0***);
- ***hidden?***, indica se *turtle* está invisível (***true***) ou visível (***false***);

- ***pen-down?***, indica se caneta da *turtle* está ativa (***true***) ou inativa (***false***);

- ***shape***, nome da forma aparente da *turtle* (o padrão é ***"default"***);

- ***size***, tamanho aparente da *turtle* (o padrão é ***1.0***, que equivale ao tamanho de um *patch* corrente);

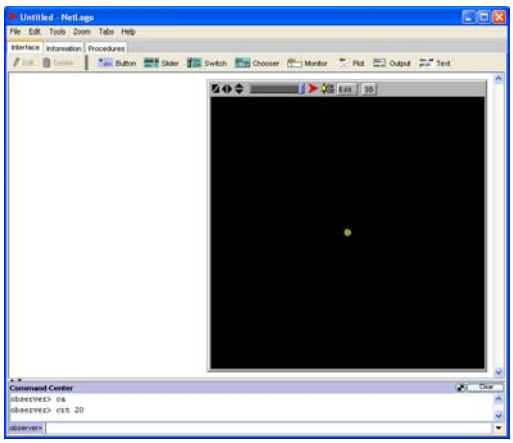
- ***who***, identificador numérico e único de cada *turtle* (ao criar as primeiras 20 *turtles*, ***who*** varia de 0 a 19); uma turtle específica pode ser identificada através do valor numérico de sua propriedade ***who***.

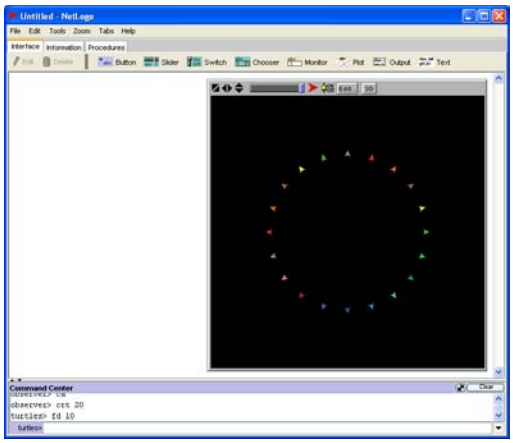
Os comandos de limpeza e inicialização do ambiente só podem ser enviados pelo *observer*. Para limpar a tela e reiniciar a execução usamos o comando ***clear-all*** (ou ***ca***), que elimina simultaneamente todas as *turtles* e *patches* existentes. Existem outros comandos relacionados a esse, são eles: ***clear-turtles*** (***ct***), ***clear-patches*** (***cp***), ***clear-drawing***, ***clear-all-plots*** e ***clear-output***.

Selecione na janela de comando *observer* e digite ***clear-all*** ou se preferir ***ca.*** Observe que a tela gráfica voltou à posição inicial. A chamada a ***clear-all*** zera todas as variáveis globais do ambiente e chama ***clear-turtles***, ***clear-patches***, ***clear-drawing***, ***clear-all-plots*** e ***clear-output***, respectivamente. O comando ***clear-turtles*** destrói todas as *turtles* existentes. O comando ***clear-patches*** reinicia todos os *patches* para seus valores iniciais, cor preta inclusive.

Vamos criar agora algumas *turtles*. Essa possibilidade é também reservada ao *observer*. Digite ***create-turtles*** (***crt***) seguido de um número (***create-turtles 20*** ou ***crt 20*** – para criar 20 *turtles*). Observe que aparecerá no meio da tela gráfica um ponto, pois todas as *turtles* criadas estão amontoadas uma sobre as outras, cada uma voltada numa direção (Figura 12).

Na seqüência, para visualizar as *turtles* vamos fazer com que elas andem para frente dez passos de *turtle*. Na janela de comando para *turtle*, digite ***forward 10*** ou ***fd 10.*** Observe que as *turtles* criadas são de várias cores e ao andarem para frente, todas juntas, formaram um círculo, o que realmente mostra que cada uma está voltada para uma direção (Figura 13).





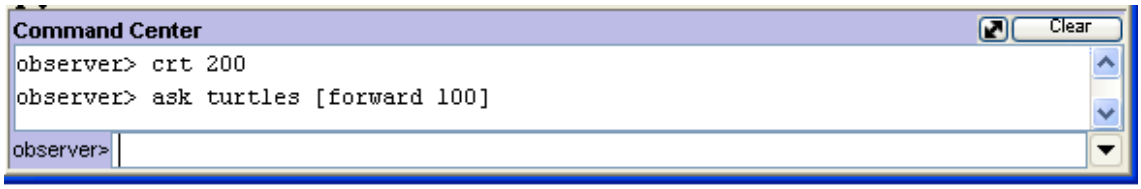


Figura 12 – Criando as *turtles*. **Figura 13 – Movendo as *turtles*.**

Caso queiramos que o *observer* defina o comando para as *turtles* executarem, basta digitarmos na janela de comando do *observer* ***ask turtles [forward 10]***, para que andem dez passos para frente. Sempre que o *observer* deseje solicitar comandos para serem executados pelas *turtles* ou *patches*, devemos seguir esta forma: a primitiva ***ask*** seguida dos agentes a que se refere (no exemplo, ***turtles***) e do bloco de comandos entre colchetes (no exemplo, ***[forward 10]***).

Cada *turtle* ou *patch* tem uma posição no seu ambiente, disposta em coordenadas cartesianas. O centro do ambiente tem coordenada (0,0). Como no plano cartesiano, o eixo horizontal é denominado eixo-x e o vertical eixo-y. No NetLogo a distância do meio até a extremidade direita ou esquerda é chamada de ***screen-edge-x***, a distância do meio aos extremos superior ou inferior é chamado de ***screen-edge-y***. Na Figura 14 temos: do centro até as extremidades direita e esquerda a distância 3, do centro aos extremos superior e inferior a distância 2.

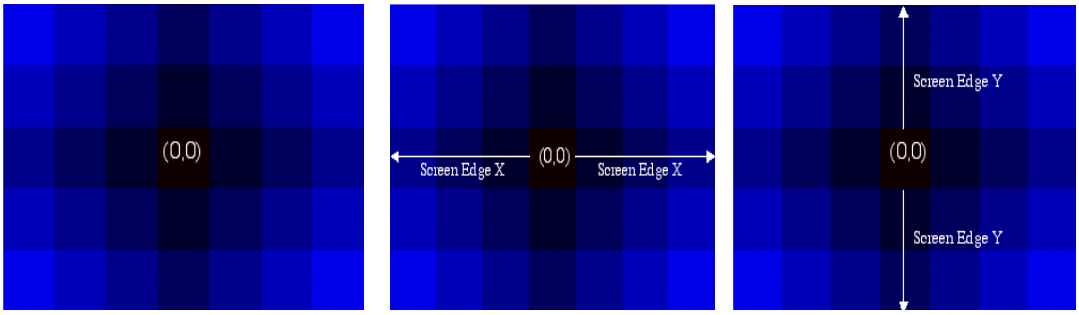


Figura 14 – Distâncias *screen-edge* da tela gráfica NetLogo.

Podemos também criar *turtles* customizadas através do comando ***create-custom-turtles N [comandos]*** ou ***cct-turtles N [comandos]***. Desta forma, são criadas ***N*** *turtles* e os ***comandos*** são executados por cada *turtle* recém criada. Por exemplo, se desejamos criar 50 *turtles* com a forma de um círculo e a cor verde, podemos fazer, como *observer*:

create-custom-turtles 50 [set shape "circle" set color green] ou

cct 50 [set shape "circle" set color green]

Neste exemplo, notamos que "***circle***" é o nome, entre aspas, de uma forma básica presente na biblioteca de formas (Figura 15), acessível pelo menu ***Tools***, opção ***Shapes Editor***. E notamos ainda que ***green*** é uma constante reconhecida pelo NetLogo para cor verde. Uma tabela de cores e respectivos valores numéricos (Figura 16) está acessível através do menu ***Tools***, opção ***Color Swatches***.

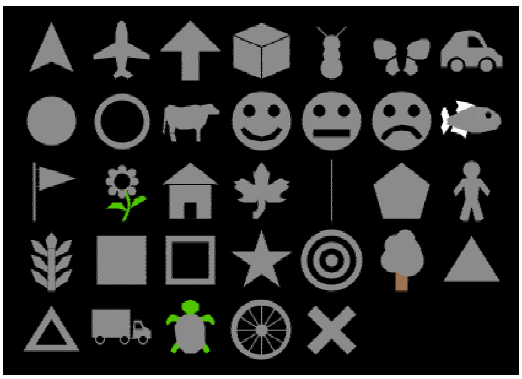


Figura 15 – Formas básicas disponíveis para as *turtles*.

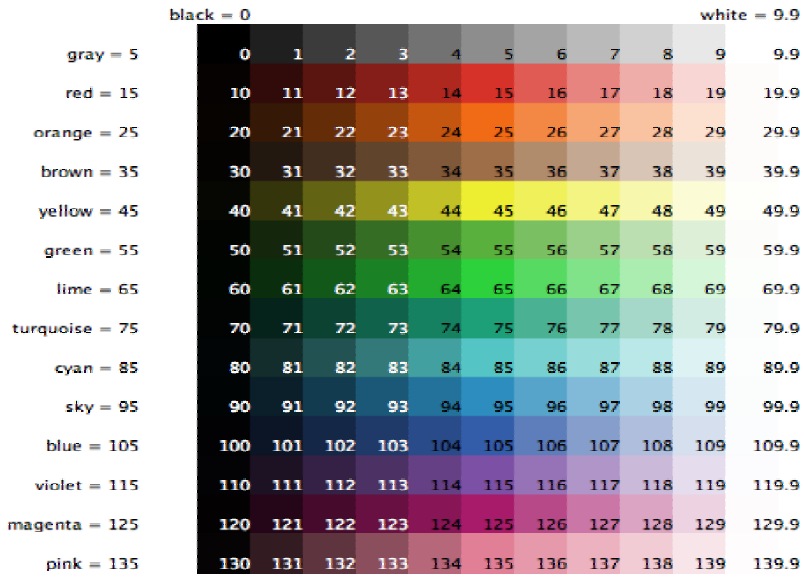


Figura 16 – Tabela de cores e respectivos valores numéricos.

Apenas o observer pode solicitar comandos ao conjunto de todas as *turtles* de uma só vez. Exemplo: ***ask turtles [set color sky]***.

Por outro lado, tanto *observer* quanto *patches* podem solicitar comandos para um conjunto reduzido de *turtles*. Assim como *observer* e *turtles* podem solicitar comandos para um conjunto reduzido de *patches*.

Exemplo com uma *turtle* específica: solicitar que a *turtle* 10 mude para vermelho, ***ask turtle 10 [set color red]***.

É possível ainda atingir conjuntos de *turtles* (ou *patches*) com propriedades comuns, usando a primitiva ***with***. Exemplo, solicitar que todas as *turtles* de cor azul céu mudem para verde: ***ask turtles with [color = sky] [set color green]***.

O mesmo vale para *patches*, respeitadas as propriedades que estes possuem.

Na prática, a primitiva ***with*** permite a definição do que o NetLogo chama de *AgentSet*. Note que *turtles* e *patches* são os *AgentSets* básicos da linguagem, mutuamente exclusivos entre si, e portanto não podem ser misturados num mesmo *AgentSet*.

NetLogo apresenta também comandos considerados “repórteres”, que na prática facilitam a contagem de elementos em um *AgentSet*. O repórter mais

básico é o comando ***count***. Por exemplo, podemos contar quantas *turtles* existem através do comando ***count turtles***, ou contar quantos *patches* são azuis com o comando ***count patches with [pcolor = blue]***.

Se você tentou estes comandos repórteres deve ter recebido mensagens de erro do tipo "***ERROR: Expected command***.", ou seja, "comando esperado". Isso porque devemos indicar de maneira explícita que o resultado deve ser mostrado. Há quatro comandos distintos para indicar ao NetLogo que algum resultado deve ser mostrado na janela de comandos:

- ***show valor*** – resulta em: ***agente-chamador valor <CR>***[2]

  Exemplo: ***observer> show count turtles***

  resulta em ***observer: 0 <CR>*** se nenhuma *turtle* foi criada ainda.

- ***print valor*** – resulta em: ***valor <CR>***

  Exemplo: ***observer> print count patches with [pcolor = blue]***

  resulta em ***20 <CR>*** (não precedido de ***observer:***) se há 20 *patches* azuis.

- ***type valor*** – resulta em: ***valor***

  Exemplo: ***observer> type shape-of turtle 10***

  resulta em ***default*** (string sem aspas) se a *turtle* 10 tem a forma padrão.

- ***write valor*** – resulta em: ***valor***

  Exemplo: ***observer> type shape-of turtle 10***

  resulta em ***"default"*** (string com aspas) se a *turtle* 10 tem a forma padrão.

Estes comandos podem ser usados com qualquer expressão que possa ser avaliada pelo NetLogo e resultar em um numeral, *string*, lista, valor lógico, constante. Experimente e observe os resultados de:

observer> show 10

observer> type sky

[2] ***<CR>*** representa *Carriage Return*, retorno de carro, ou seja, quebra de linha.

observer> print “João e Maria”

observer> type “João e Maria”

observer> write “João e Maria”

observer> show sky = blue

observer> write sky = 95.0

observer> write yellow

observer> type turtles-at 1000 1000

Experimente enviar estes mesmos comandos através dos agentes *turtles* e *patches* e observe em que diferem os resultados.

5. Biblioteca NetLogo

Quando desejamos analisar um modelo pronto, basta abrirmos a biblioteca de modelos, selecionando o menu ***File***, opção ***Models Library***, que abre uma nova janela (Figura 17), que disponibiliza uma relação de modelos em várias áreas (arte, biologia, química, ciência da computação, ciências da terra, entre outros). Cada modelo da biblioteca traz instruções sobre o mesmo, qual seu objetivo, como utilizar, o que pode ser observado, etc.

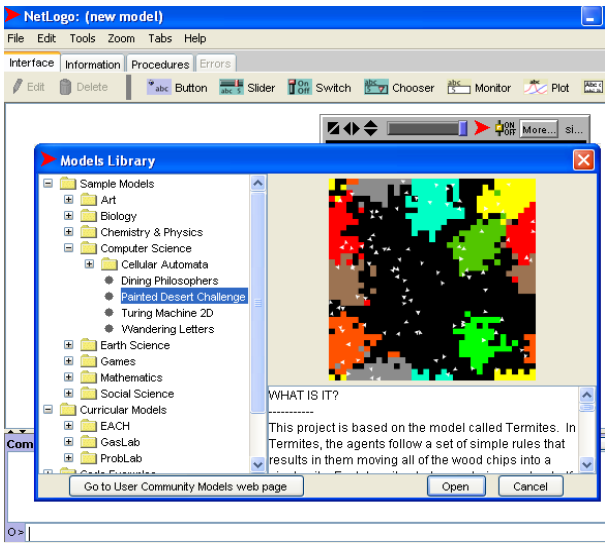


Figura 17 – Biblioteca de modelos NetLogo.

Para visualizar algum modelo, basta selecioná-lo para que este fique pronto para execução. É interessante, antes da execução, ler a pasta de

informação para saber como o modelo funciona, suas características, como usar, o que pode ser observado.

Geralmente os modelos apresentam dois botões básicos: botão ***setup***, que é utilizado para iniciar o modelo; e o botão ***go***, para que o modelo seja executado. Se tiver sido selecionada a opção *forever*, as instruções associadas com o botão são executadas repetidamente, caso contrário são executadas uma única vez por clique. Os modelos podem trazer também botões de controle e visualização de parâmetros e informações (*Slider, Switch, Chooser, Monitor, Plot, Output* e *Text*), que serão abordados a seguir (Seção 6).

Para iniciar o modelo clique no botão ***setup*** e para colocá-lo em movimento clique no botão ***go***. Após analisar o modelo, experimente variar os parâmetros e observar o que ocorre.

6. Elementos de interface

Agora que já experimentamos a interação com um modelo através dos elementos de interface, vamos entender o que eles significam e como são tratados internamente pelo NetLogo e, mais importante, como usá-los para facilitar nosso trabalho de programação. Esses elementos podem ser incluídos em nossos modelos através da barra de ferramentas (Figura 18).

Figura 18 – Barra de ferramentas dos elementos de interface NetLogo.

6.1 Button

Criamos um *button* para facilitar a execução automática de comandos através de um único clique. Para criar um *button* clique em 

e em seguida clique na área de trabalho (janela branca do NetLogo). Isso abrirá a caixa de diálogo ***Button*** (Figura 16) que permite definir:

- qual agente executará os comandos (ex. ***Observer***);
- executar repetidamente (***forever***) ou apenas uma vez por clique;
- quais os comandos associados, separados por espaços ou em linhas distintas (ex., ***ca crt 20 ask turtles [fd 5]***);

- nome que será exibido na interface (ex. ***Botão Teste***);
- tecla de atalho equivalente (ex. ***T***).

Experimente criar alguns botões diferentes com os mesmos comandos, combinando as diferentes possibilidades de agente executor, *forever* ou não, com ou sem tecla de atalho e teste-os observando como e quando funcionam. Em seguida, experimente criar botões para execução de comandos diferentes, por exemplo, alguns dos comandos que você treinou através da janela de comandos (Seção 4).

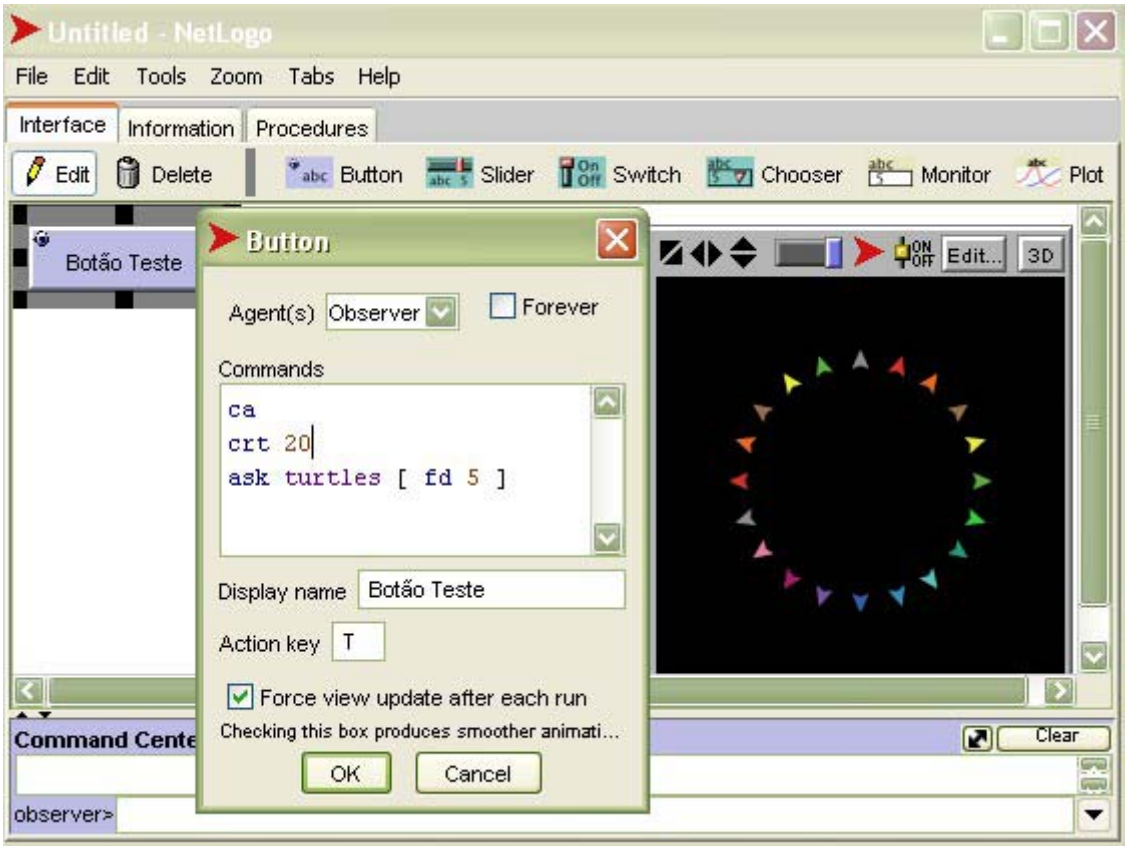


Figura 19 – Criação de botões de comando.

6.2 Slider

Um *slider* define uma **variável global**, acessível a todos os agentes do modelo. Geralmente esta variável é usada para variar parâmetros do modelo sem a necessidade de reescrever os comandos.

Vamos criar um *slider* para variar o número de *turtles* criadas. Para isso clique em 

e em seguida na área de trabalho, isso abrirá a caixa de diálogo ***Slider*** (Figura 20), que permite definir:

- nome da variável global associada ao *slider* (ex., ***NumeroDeCriaturas***);

- os valores mínimo e máximo que a variável pode assumir e o incremento ou variação (ex., mínimo de ***5*** e máximo de ***20***, variando de ***1*** em ***1***);

- o valor inicial pré-selecionado no *slider* (ex., ***Value 10***);

- unidade de medida (opcional), útil para variáveis que expressam medidas padronizadas como *kg*, *cm*, *min*, *Km*, etc.

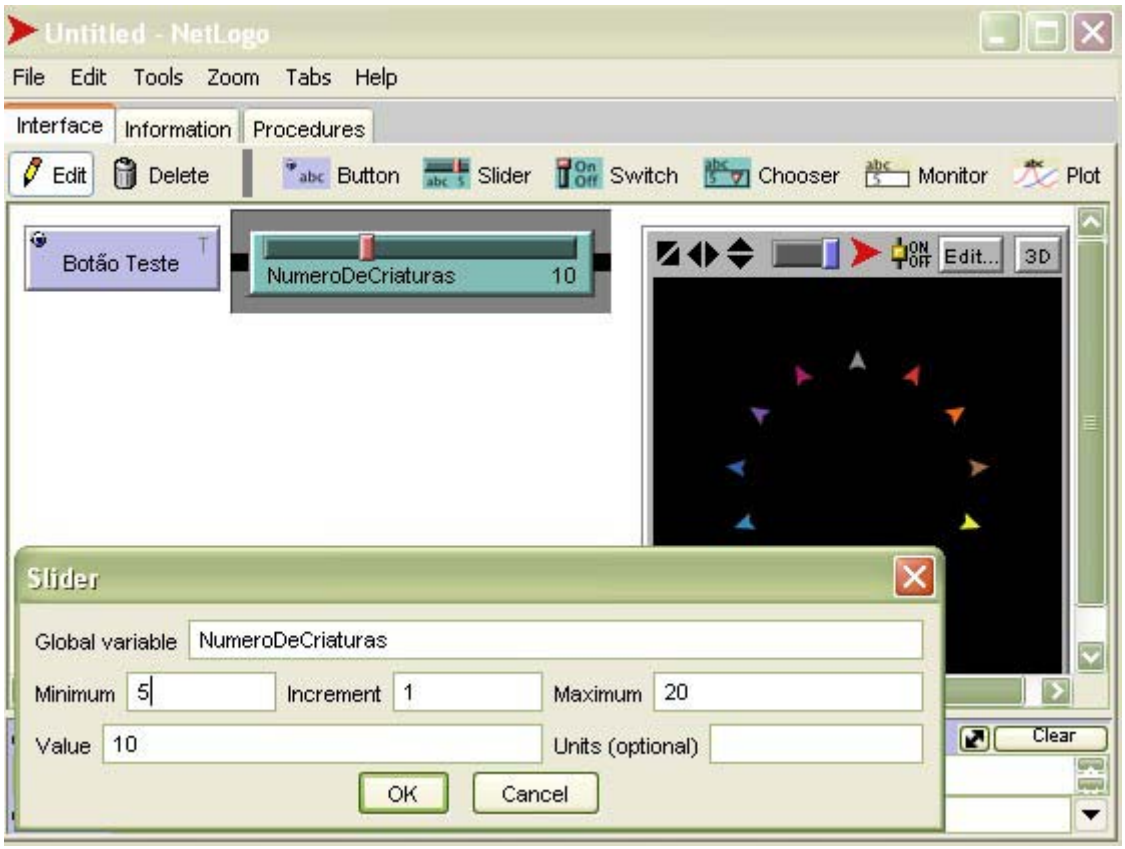


Figura 20 – Criação de *slider* para controlar parâmetros do modelo.

Após criar um *slider* como o da Figura 20, modifique o ***Botão Teste*** original (Figura 14), clicando com o botão direito, opção ***Edit...*** e substitua ***crt 20*** por ***crt NumeroDeCriaturas***. Observe o resultado ao variar os valores de ***NumeroDeCriaturas*** e executar o ***Botão Teste***.

Sugerimos que você repita o mesmo para definir parâmetros variáveis associados aos botões criados na atividade anterior (lembre-se que é necessário modificar os comandos nos botões) e teste incrementos decimais.

Parâmetros associados a *sliders* também são acessíveis através da janela de comando. Por exemplo, como *observer* digite ***show NumeroDeCriaturas***. Observe que dependendo da ordem de suas ações sobre o modelo, isso nem sempre resulta em um valor equivalente a ***show count turtles***.

6.3 Switch

Um *switch* define e representa visualmente uma **variável lógica global**, acessível a todos os agentes do modelo. Variáveis lógicas podem assumir os valores ***true*** (verdadeiro, quando *swich* ***On***) ou ***false*** (falso, quando *switch* ***Off***).

Para criar um *switch* clicamos em 

e em seguida na área de trabalho. Isso abrirá a caixa de diálogo ***Switch*** (Figura 21) que permite definir o nome da variável lógica global associada (ex. ***LimparAntesDeCriarNovas?***).

Parâmetros lógicos variáveis podem ser usados como critério de decisão sobre a **execução ou não** de partes do modelo, quando usadas em associação com o comando condicional ***if condição [comandos]***, que significa que os ***comandos*** somente serão executados se a ***condição*** for verdadeira.

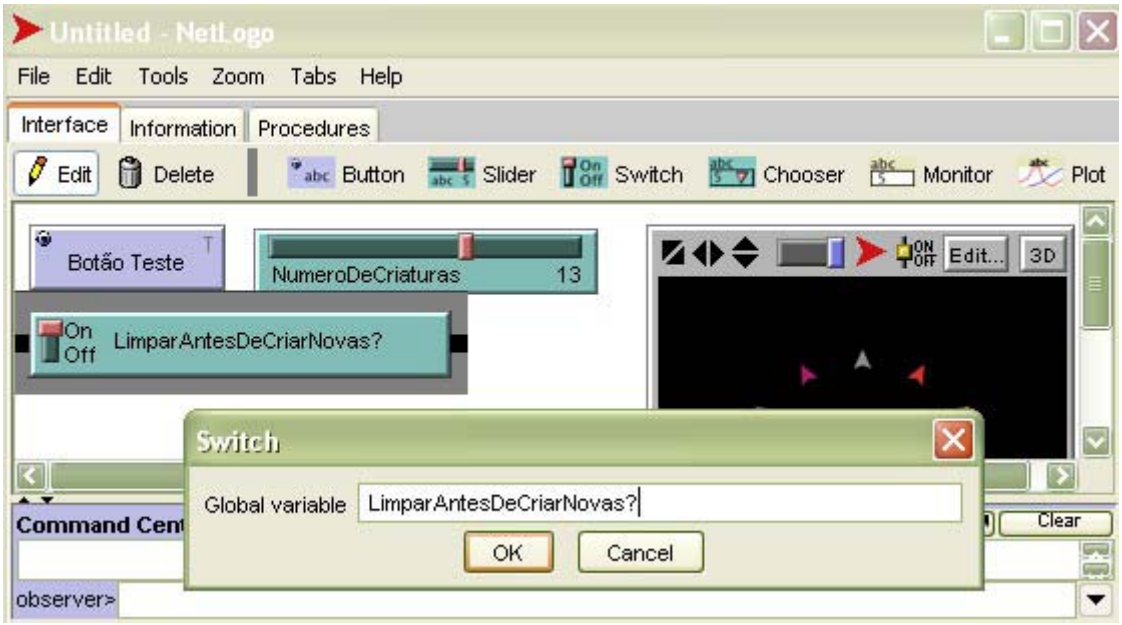


Figura 21 – Criação de *switch* para controlar parâmetros lógicos.

Após criar um *switch* como o da Figura 18, modifique o ***Botão Teste*** original (Figura 16), substituindo ***ca*** por ***if LimparAntesDeCriarNovas? [ca]***. Isso fará com que os comandos entre colchetes (no exemplo, ***ca***) sejam

executados apenas se o valor de ***LimparAntesDeCriarNovas?*** for verdadeiro (*switch* ***On***).

Experimente criar *switches* para controlar outros parâmetros lógicos do modelo (lembre-se de modificar os comandos nos respectivos botões). Parâmetros associados a *switches* também são acessíveis através da janela de comando. Por exemplo, como *observer* digite ***show LimparAntesDeCriarNovas?***.

6.4 Chooser

Um *chooser* define uma **variável global** a qual é atribuído o valor selecionado pelo usuário em um menu *drop down* (lista de seleção desdobrável). Para criar um *chooser* clicamos em 

e em seguida na área de trabalho. Isso abrirá a caixa de diálogo ***Chooser*** (Figura 22), que permite definir o nome da variável global (ex. ***Forma***) e uma lista de opções que o menu apresentará, separadas por espaços ou linhas (ex. ***"circle" "square" "triangle" "default"***).

Podemos testar o *chooser* ***Forma*** modificando no ***Botão Teste*** os comandos ***crt NumeroDeCriaturas*** para ***cct NumeroDeCriaturas [set shape Forma]***.

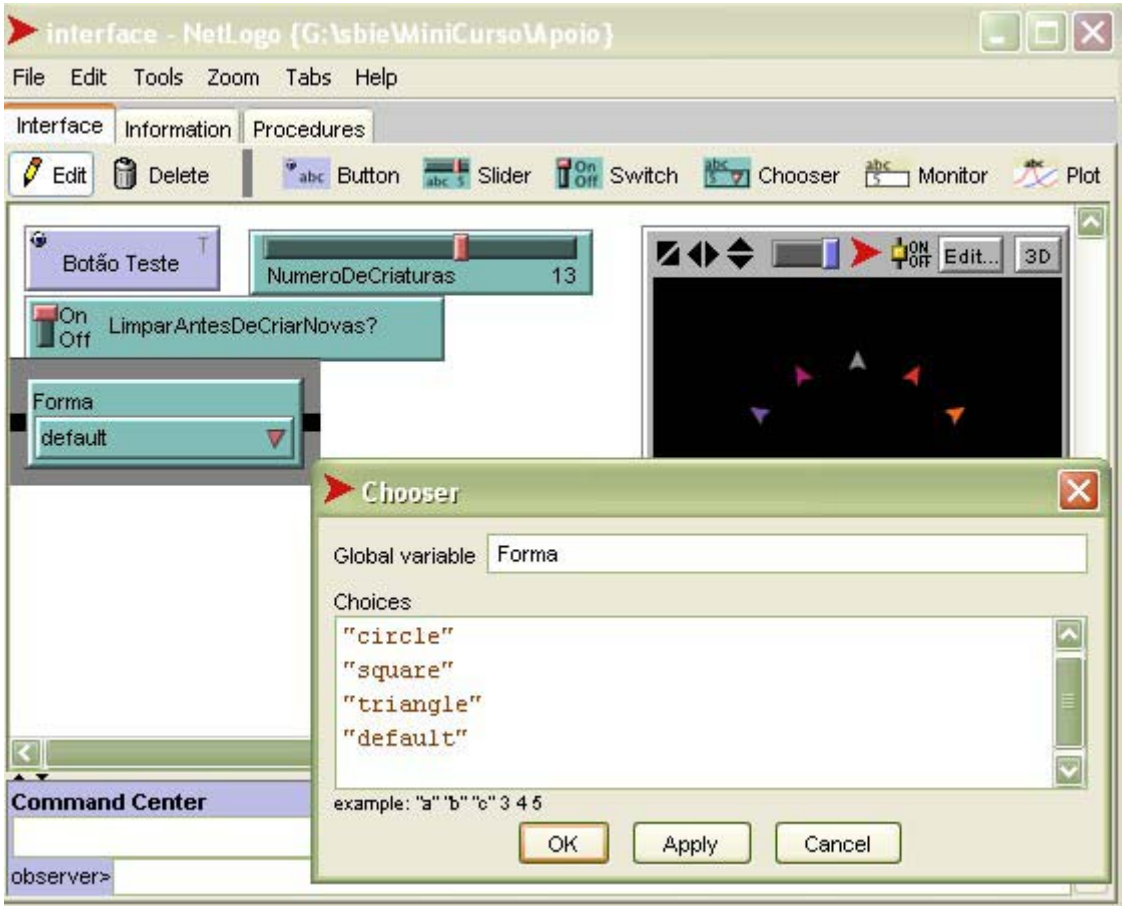


Figura 22 – Criação de *chooser* para controlar listas de parâmetros.

6.5 Monitor

Um *monitor* mostra o conteúdo dinâmico de uma variável, um *repórter* ou uma expressão composta. Para criar um *monitor* clicamos em 

e em seguida na área de trabalho. Isso abrirá a caixa de diálogo Monitor (Figura 23), que permite definir:

- a variável, repórter ou expressão que será avaliada e cujo valor será mostrado (ex. ***count turtles with [pxcor < 0]***) ;
- o nome do monitor (ex. ***Turtles na metade esquerda da tela***);
- se numérico decimal, quantas casas decimais devem ser consideradas.

Para observar a mudança dinâmica de ***Turtles na metade esquerda da tela*** podemos, por exemplo, criar um botão de comando (***go***) para os agentes *turtles*, do tipo *forever*, com os comandos ***rt random 360 fd 1***.

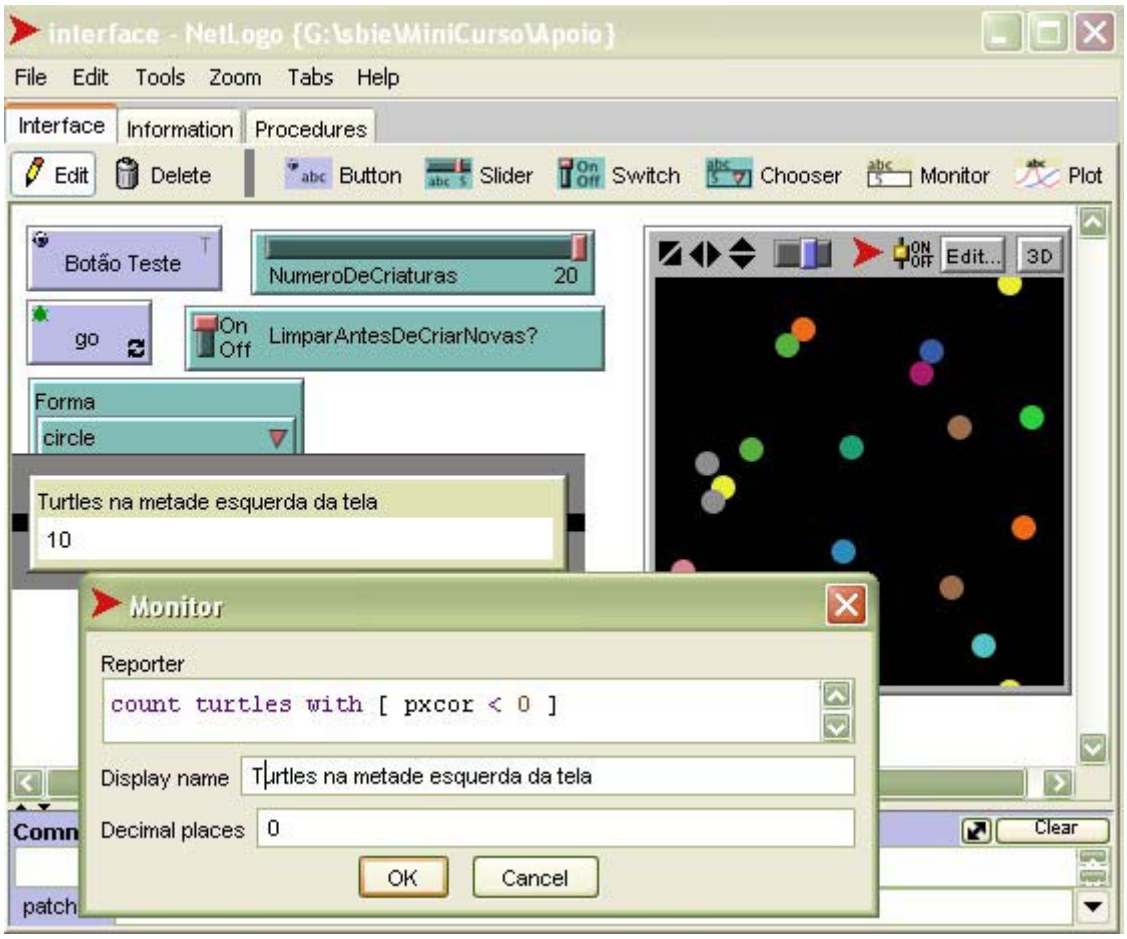


Figura 23 – Criação de *monitor* para mostrar valores de expressões.

6.6 Text

Um elemento *text* apresenta um texto estático na área de trabalho. Utilizamos esse recurso para tornar mais clara e explicativa a interface de um modelo. Para criar um *text* clicamos em Text e em seguida na área de trabalho. Isso abrirá a caixa de diálogo ***Text Box*** (Figura 24), na qual digitamos a mensagem de texto a ser apresentada.

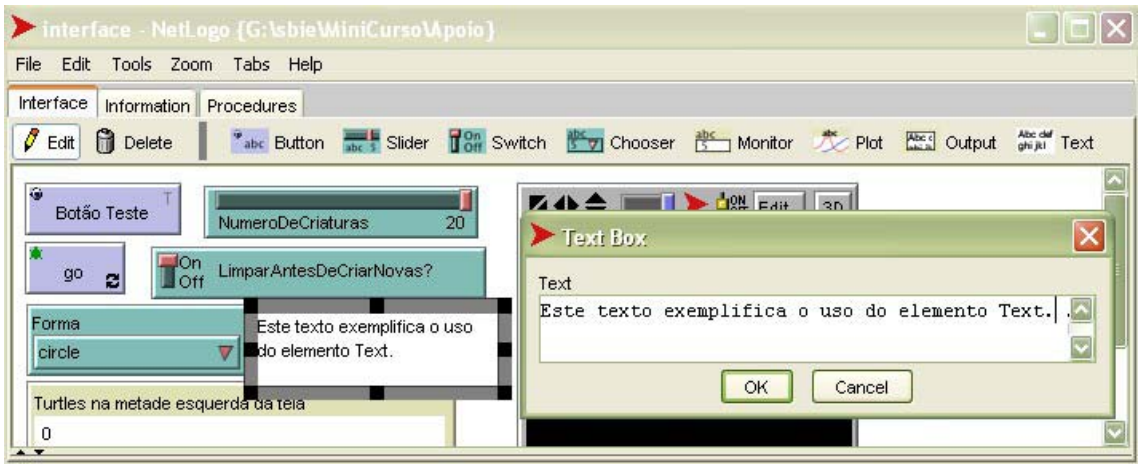


Figura 24 – Criação de *text* para dar clareza a interface do modelo.

6.7 Plot

Um elemento *plot* permite plotar um gráfico dinâmico a partir dos dados gerados pelo modelo. Trabalharemos com esse elemento quando desenvolvermos nossos modelos na forma de programas completos (Seção 7).

6.8 Output

Um elemento *output* cria uma área de texto dinâmica, com possibilidade de *scrolling*, útil para manter um registro do tipo *log*, por exemplo, quando estamos testando se um modelo funciona conforme o esperado, ou quando desejamos manter e mostrar um histórico de valores gerados ou ações executadas. Apenas um *output* por modelo pode ser criado.

Para criar um *output* clicamos em 

e em seguida na área de trabalho. Para escrever no *output* usamos os comandos ***output-print valor***, ***output-show valor***, ***output-type valor*** e ***output-write valor***, que são semelhantes aos comandos ***print***, ***show***, ***type*** e ***write*** vistos na Seção 4. A diferença é que os primeiros escrevem o ***valor*** no *output* e os últimos na janela de comandos.

Propomos que você crie um *output* e experimente escrever alguns conteúdos nele, usando os comandos ***output-??? valor*** para relembrar a diferença entre ***print***, ***show***, ***type*** e ***write***. Varie o envio pela janela de comandos ou associado a botões.

7. Desenvolvendo uma atividade com NetLogo

Vamos desenvolver uma atividade com o NetLogo para simular uma epidemia. Antes de desenvolver qualquer atividade é importante ter claro:

- a que proposta o modelo servirá?
- pretende auxiliar na exploração de um novo tópico ou entender melhor um fenômeno que esta sendo observado?
- ele servirá para mostrar alguma idéia da qual esteja pensando ou descrever sobre dados coletados?

A resposta a essas e muitas outras perguntas são válidas para se construir um modelo.

Normalmente consideramos epidemia algo ou doença que se alastra rapidamente. Como doença, essa geralmente é infecciosa e de caráter transitório, atacando simultaneamente um grande número de pessoas de uma determinada localidade. Geralmente ocorre da seguinte maneira, primeiro alguma pessoa contrai a doença e através do contato dessa com outras pessoas há uma disseminação do problema.

Através desse modelo muitas relações podem ser exploradas: o caráter interdisciplinar do problema, aspectos biológicos, matemáticos, sociais, entre outros.

7.1 Criando um exemplo - Epidemia (nome do nosso projeto)

Estamos observando nosso micro-mundo, que está sendo representado por uma comunidade cujo espaço físico é definido pelo tabuleiro do NetLogo, e os seus habitantes são os agentes que se deslocam livremente sobre este.

Em seus percursos aleatórios, os habitantes se encontram e interagem uns com os outros. Enquanto todos gozam de bom estado físico, suas interações não desencadeiam nenhuma desarmonia na comunidade. Porém, vamos supor que um dos agentes desta comunidade seja infectado por um corpo estranho a este ambiente.

Os agentes, desconhecendo o fato, continuam se deslocando livremente sobre o ambiente e interagindo uns com os outros. Entretanto, a cada encontro do agente infectado com um agente sadio, este é infectado.

Num processo contínuo, podemos prever que essa contaminação se alastrará por toda comunidade num rápido crescimento, caso não seja combatida. É esse modelo que pretendemos descrever no nosso ambiente de simulação.

7.1.1 Desenvolvendo o modelo

O espaço físico do micro-mundo já está definido, tabuleiro do ambiente NetLogo. Vamos definir a população que habita essa comunidade. Neste caso é necessário estabelecer um número de habitantes. No ambiente NetLogo esse procedimento é realizado como segue.

Na janela do comando central criamos 200 *turtles* (supondo que nossa comunidade tenha 200 habitantes);

```
Observer > crt 200
```

Todas as *turtles* criadas estão posicionadas uma sobre a outra no ponto (0,0) da tela. Elas são iniciadas com uma variedade de cores.

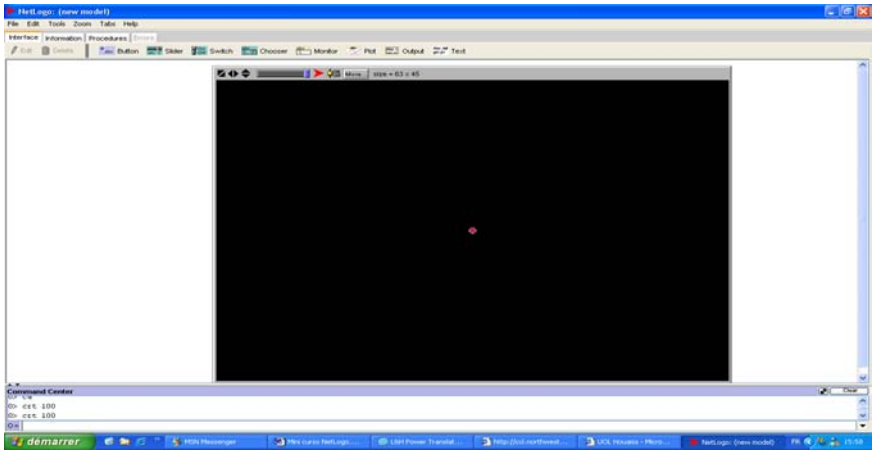


Figura 25 – Início modelo

Agora, clicando sobre o ***Observer>***, na janela de comando, lado esquerdo, e selecione ***Tutles***>; porque queremos direcionar os comandos para serem executados apenas pelas *turtles* criadas. Assim, o próximo comando é:

`Turtles> forward random 100` (fd random 100), esse comando fará com que todas as *turtles* andem para frente um número randômico (de 0 a 99). Feito isso podemos ver as todas as *turtles*. Para cada *turtle* é atribuído um valor randômico para o movimento.

Para melhor aproximar do modelo e dar uma melhor visualização do que pretendemos simular, precisamos fazer com que toda a população apresente a mesma característica. Faremos com que todos os agentes sadios tenham a mesma cor.

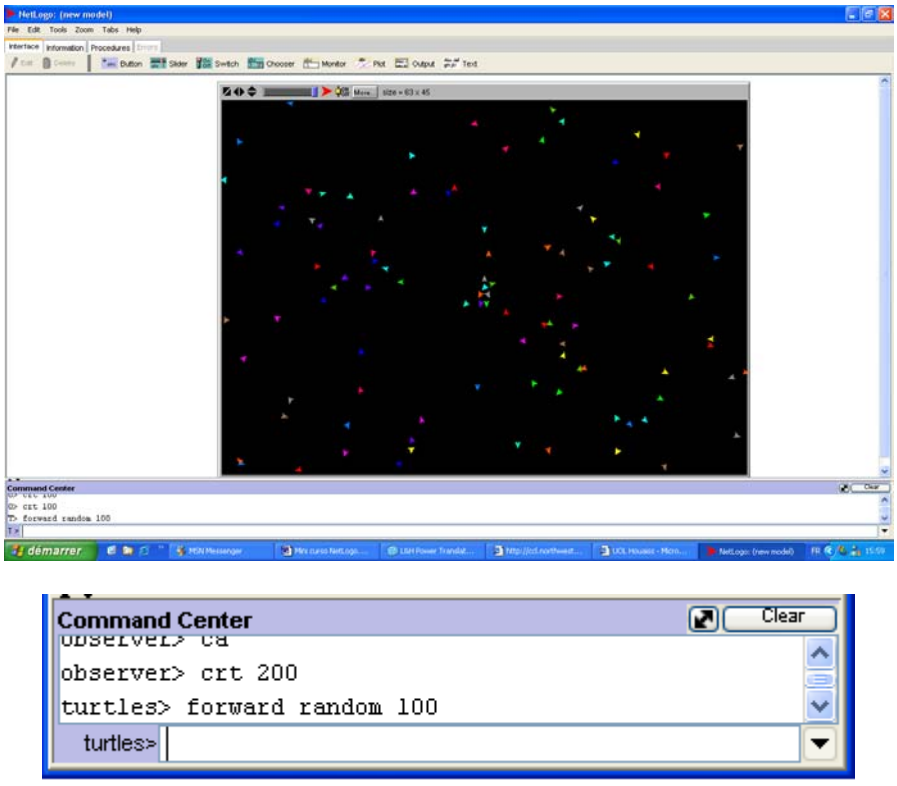
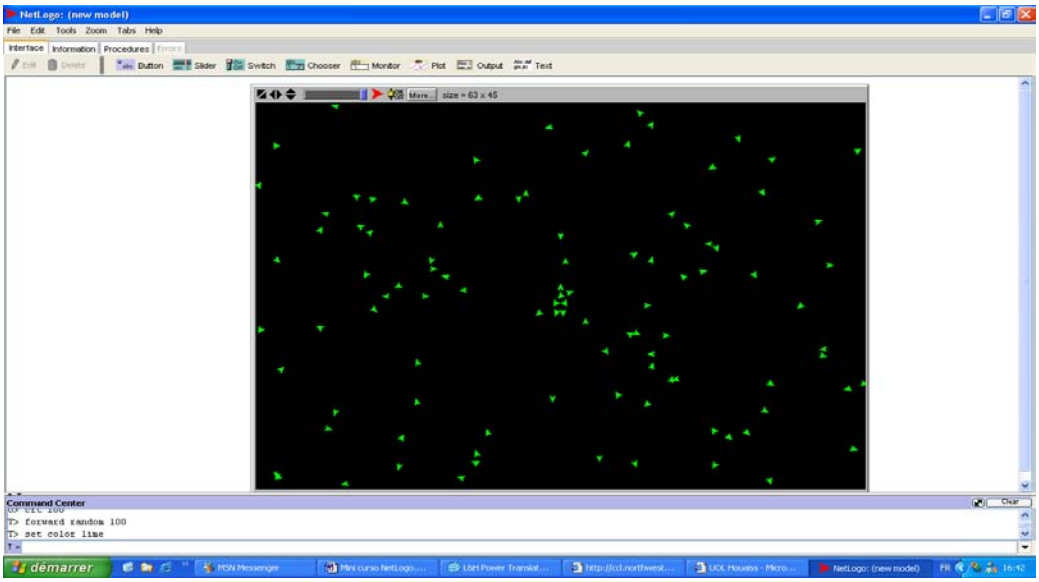


Figura 26 – Distribuição agentes

Veja o que ocorre se fizermos:

`Turtles> set color lime` (continuamos no comando de Turtle);



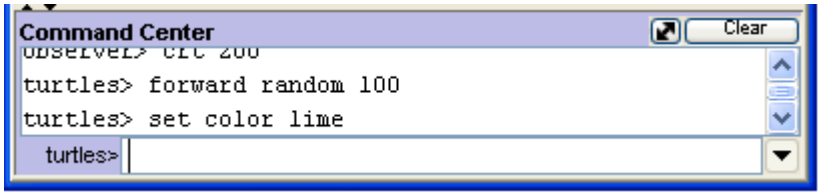
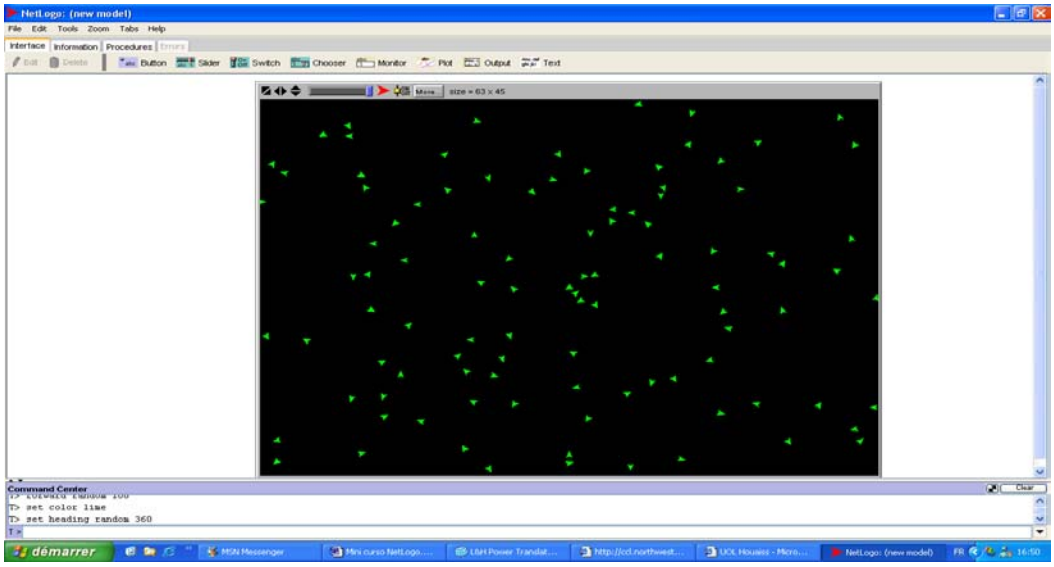


Figura 27 – Atribuição de cor

Com o comando ***set heading n°*** (N° é o grau do giro) faz com que as *turtles* girem para uma nova direção.

Turtles> `set heading random 360` (grau variando de 0° a 359°);



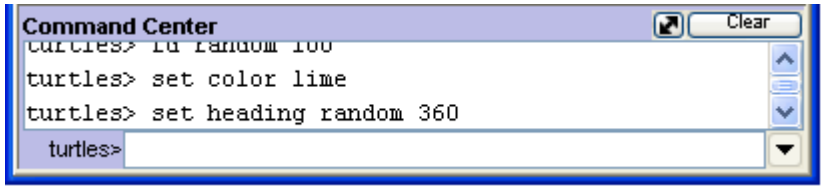


Figura 28 – Atribuição de direção

Bem, agora queremos fazer com que uma *turtle* fique doente (contaminada).

Simularemos isso mudando a cor de uma *turtle*, estabelecendo uma condição:

`if who = 1 [set color red]` (observe o espaço ante e depois do sinal de "=") (obs. Cada *turtle* tem uma ID, nesse caso específico estou chamando a *turtle* de ID = 1)

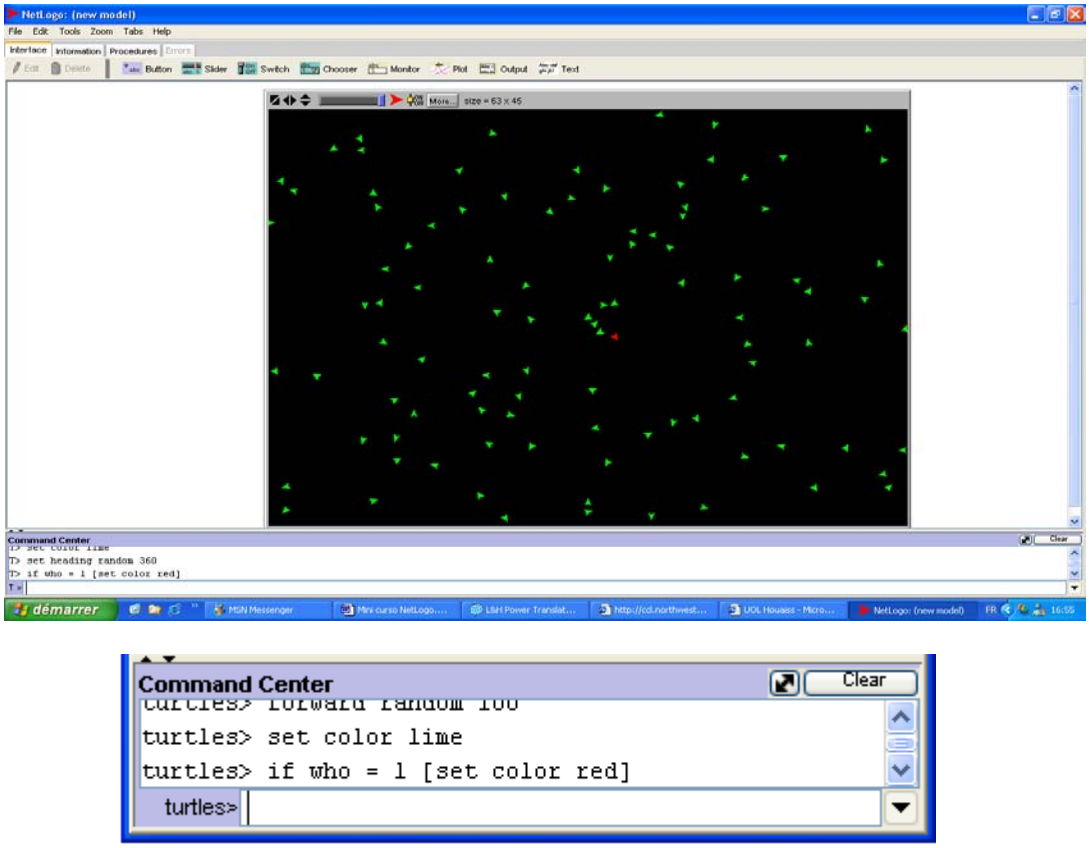


Figura 29 – Diferenciando um dos agentes: usando seu identificador

Observe que entre todas as *turtles* apenas a de ID = 1 ficou de cor vermelha.

Todos esses comandos executados até o momento podem compor o procedimento inicial (setup). Vamos recapitular o que fizemos ate agora.

```
Observer> crt 100

Turtles> forward random 100
```

```
Turtles> set color lime

Turtles> set heading random 360

Turtles> if who = 1 [set color red]
```

Na janela principal escolha na barra de ferramentas a opção **Procedures** e transportamos todos esses comandos para este local, retirando os caracteres Observer> e *Turtles*>. Para diferenciar os comandos executados pela *turtle* indicaremos eles por ask *turtles* [ ].

Entretanto todo procedimento deve ter um nome, e deve sempre iniciar com **to nome-procedimento** e terminar com **end**, faremos:

Para mais informações sobre o uso de procedimentos leia o documento complementar **Leia Mais 1**

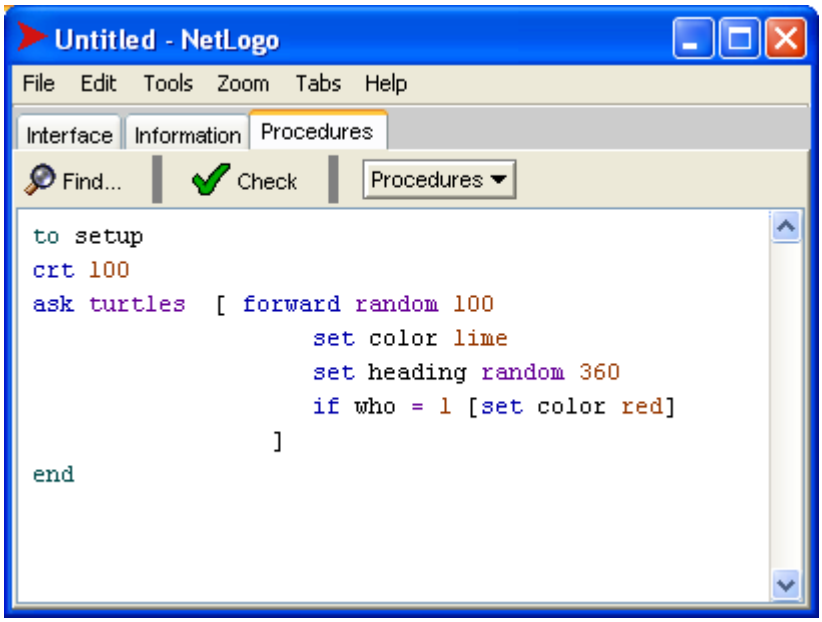


Figura 30 – Uso de procedimentos

Vamos agora testar nosso procedimento. Digite no comando central o procedimento **setup**.

Atenção: todo procedimento começa com **to nome-procedimento** e termina com **end**

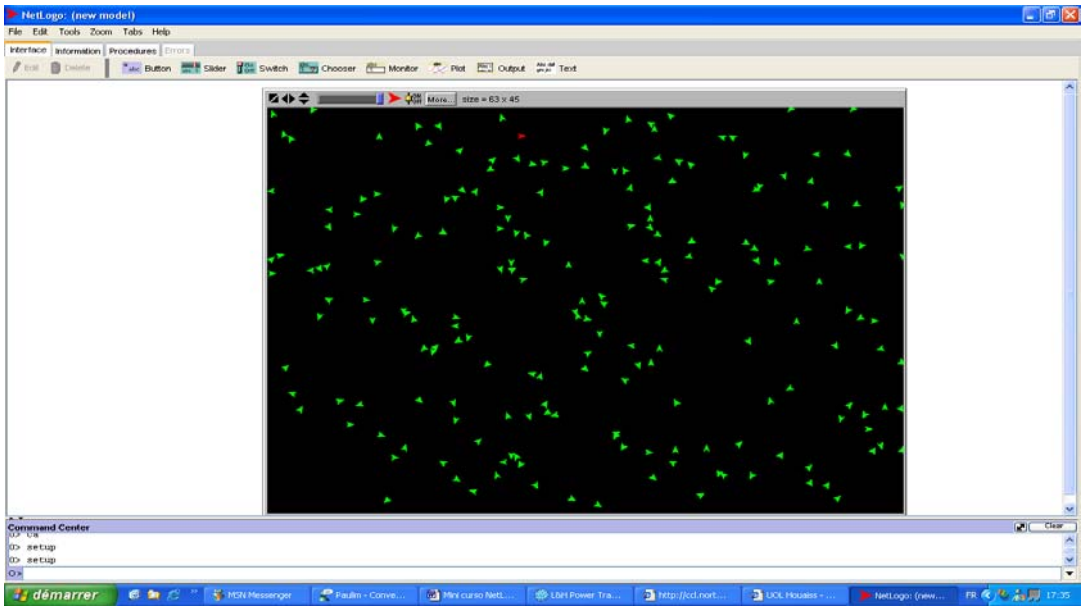


Figura 31 – Execução do procedimento

Ora, aumentaram o número de *turtles* em sua tela?

Bem, provavelmente a primeira coisa que devemos fazer antes de executar um procedimento é limpar toda a tela, para isso usamos o comando "**ca**". Um modo mais prático de fazer isso é sempre iniciar com o comando "**ca**" na primeira linha do seu procedimento **setup**, assim:

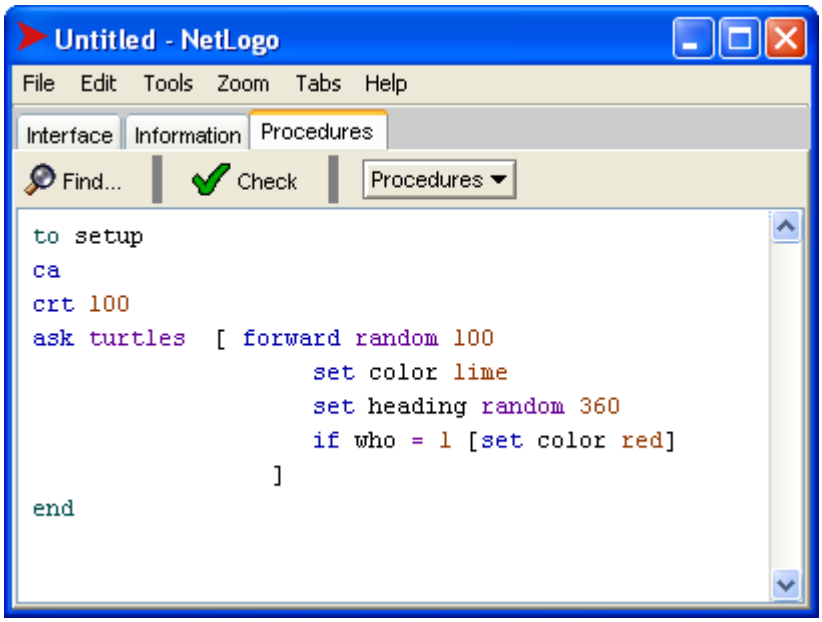


Figura 32 – Definição de procedimento

Para facilitar ainda mais nosso trabalho, ao invés de digitar todas às vezes o comando **setup** para executar o procedimento vamos criar um botão para disparar esse procedimento, conforme apresentado na seção 6.

Voltamos para a janela principal e selecione na barra de ferramentas a opção **Interface** clique sobre o **button**, em seguida clique em qualquer lugar da tela branca, aparecera o botão e uma janela para descriminação do mesmo,

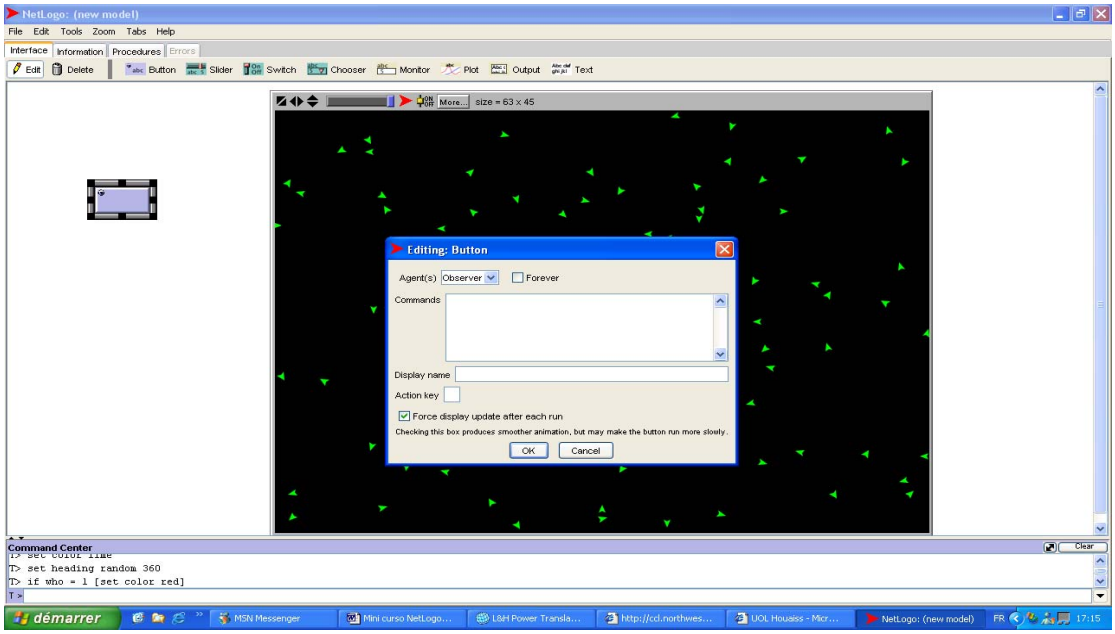


Figura 33 – Uso de botões

Escreva **setup** no box chamado comando da caixa Editing Buton, em seguida clique OK. Agora teste o procedimento para ver o que ocorre, teclando no botão **setup** . Observe que nesta caixa de dialogo você pode especificar o(s) comando(s) a serem executados e pode nomear o botão com o nome que achar mais conveniente para acionar o seu procedimento.

É possível alterar o tamanho da tela de exibição, para isso clique com o botão direito do mouse, sobre a tela preta, e selecione **Edit**. Assim é possível alterar o tamanho da tela. Para mover a tela ou qualquer um dos botões colocados na interface, basta clicar sobre ele com o botão direito do mouse e escolher **select** e arrastar para o local desejado. Procedimento idêntico para parar a seleção ou editar.

7.2 Criando novos procedimentos.

Dando seqüência ao problema proposto, mencionamos que os habitantes da comunidade movimentavam livremente e interagiam uns com os outros. Assim, vamos agora criar procedimentos para dar movimento (go) aos agentes e, se quiser, parar a execução. Queremos definir um procedimento que faça nosso modelo interagir várias vezes. Voltemos para a área de procedimentos:

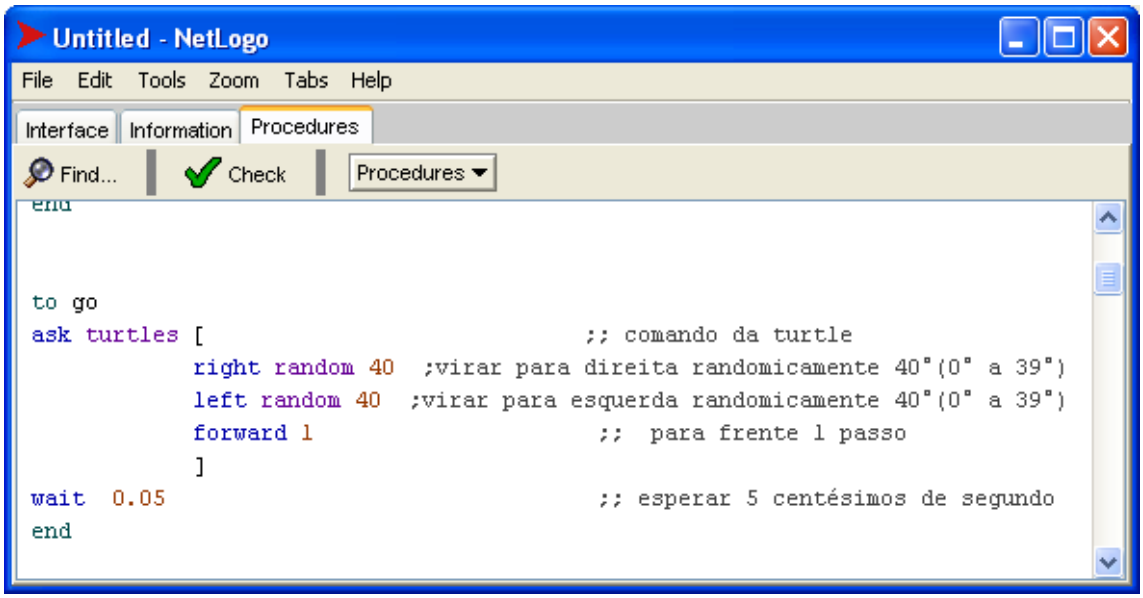


Figura 34 – Definição de procedimento de movimento

da mesma forma vamos criar o botão **go**. A única diferença é que selecionaremos o ícone **forever** para tornar o modelo numa execução sem interrupção.

Esse mesmo movimento pode ser acionado pelo agente *turtles,* com pequena modificação na forma de acionar, pois sendo um movimento descrito pelo próprio agente não há necessidade de usar o comando ask, ficando:

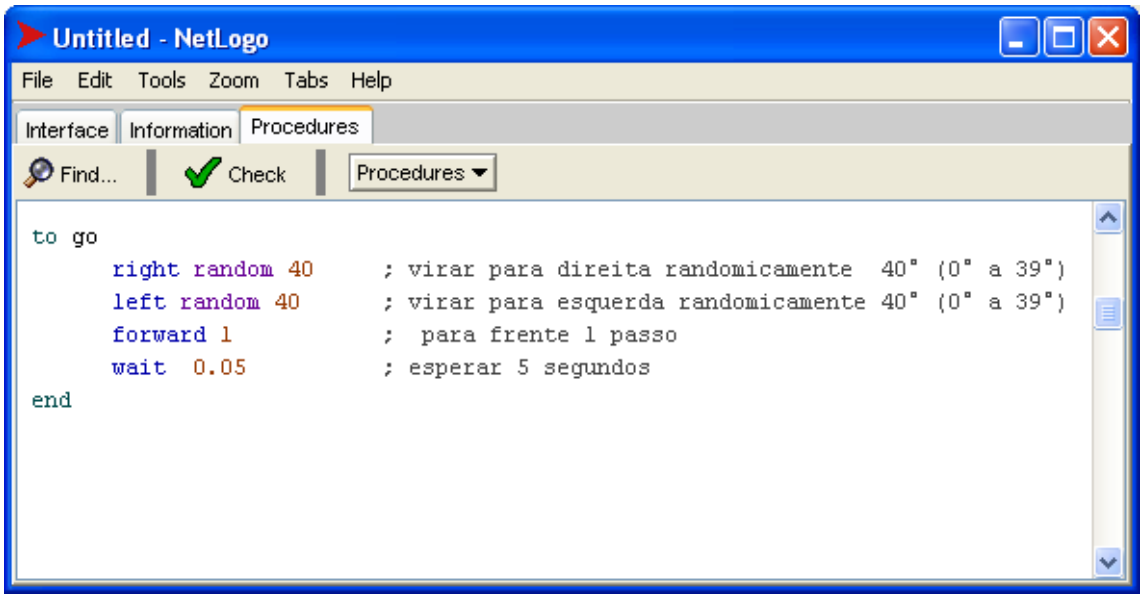


Figura 35 – Mesmo comando da janela anterior, porem chamado pelo agente turtles

Criando o botão **go**, conforme item anterior de geração de botão.

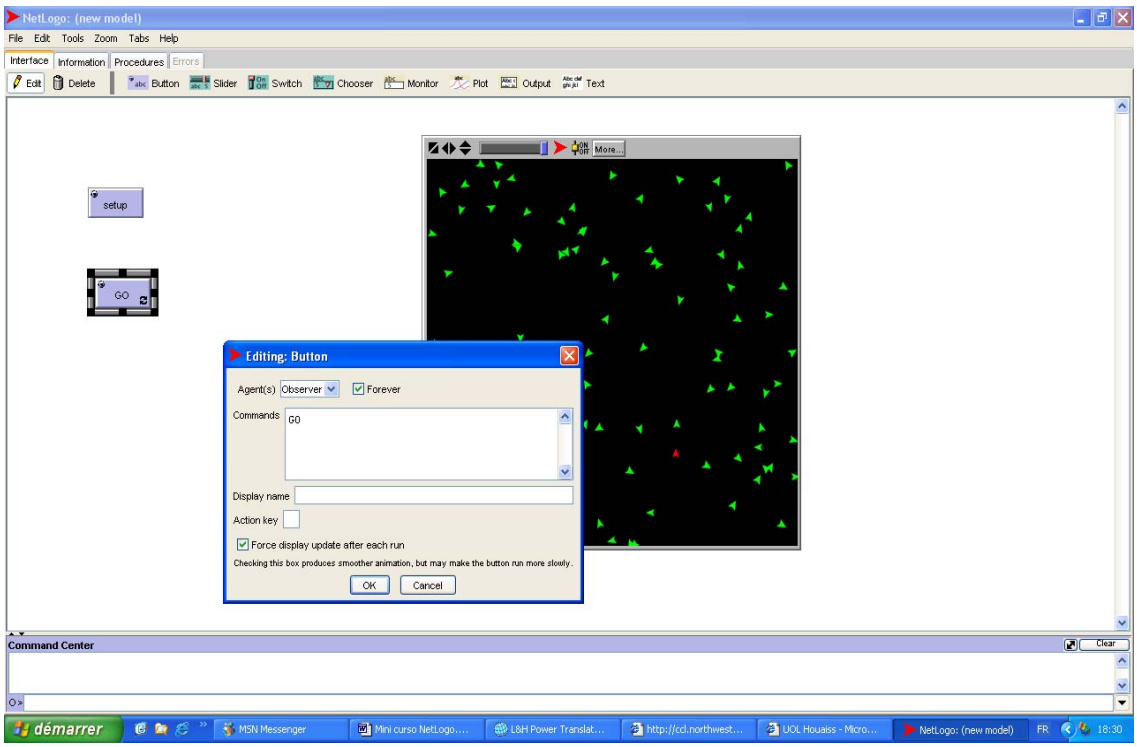


Figura 36 – Idem figura 33

Teste o modelo usando os botões **setup** e **go**, veja o que acontece. Para parar o movimento, basta clicar no botão **go** novamente.

Queremos agora criar uma *turtle* que esteja contaminada. Vamos supor que a *turtle* de ID = 1 esteja contaminada, para diferenciar essa das demais vamos mudar a sua coloração, tornando a de cor vermelha. Esse procedimento é realizado através da condição:

```
if  who = 1 [set color red]
```

Feito isso, a *turtle* vermelha apresenta "infectada" em relação as demais *turtles*.

Toda a população esta se movimentando no ambiente, inclusive a infectada. Com essa interação, simulando uma epidemia. Para que isso ocorra vamos estabelecer a seguinte condição: a *turtle* vermelha infeta outra quando essa cruzar o seu caminho; para isso usamos o comando do NetLogo para definir essa expressão:

```
if color = red [set color-of random-one-of turtles-here red]
```

colocaremos esse comando na primeira linha abaixo do comando **forward 1** do procedimento **go.**

Teste o experimento novamente e veja o que acontece.

O comando **random-one-of** escolhe um agente aleatório que esta sobre o patch ***turtles*-here**. Se o *agentset* esta vazio, retorna **nobody**, caso contrário pinta a *turtle* de vermelha.

Suponha que quiséssemos ver a expansão dessa epidemia em função do tempo. Vamos traçar um gráfico para acompanhar os valores.

Sobre a barra de ferramentas clique sobre o botão *Plot*, em seguida clique em qualquer lugar da tela branca para criar o espaço do gráfico.

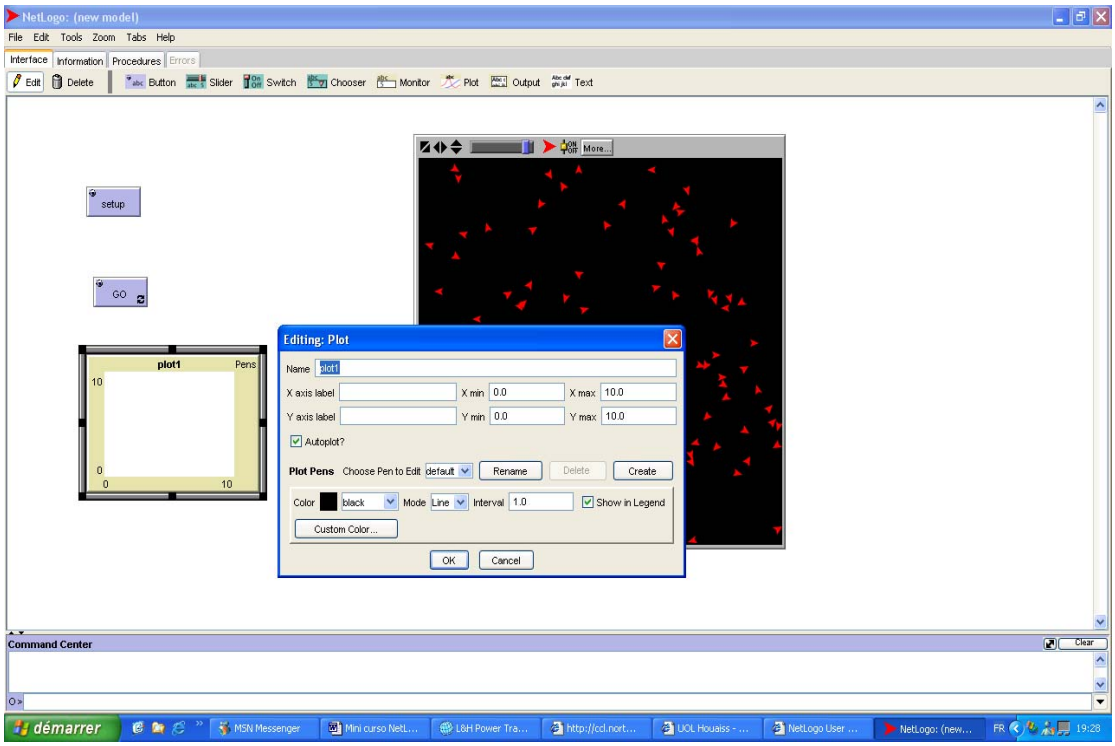


Figura 37 – Montando o botão go

vamos preencher os dados:

Vamos dar um nome para esse gráfico: Epidemia

No eixo-x colocamos o tempo; no eixo-y as *turtles* (n° de *turtles*). Escolha a cor da linha do gráfico e o modelo de gráfico (linha, barra, ponto). Escolha o nome para o qual deve aparecer na legenda através do comando "**plot pen**".

Vamos ao procedimento **go** para efetivar a construção do gráfico. O que queremos plotar no gráfico? O número de *turtles* vermelhas, então usaremos o comando do NetLogo para fazer isso:

```
plot (count turtles with [color = red])
```

como temos um número definido de *turtles*, seria interessante que o gráfico parasse quando chegasse a esse valor, certo? O comando no NetLogo para fazer isso é um comando com condição:

```
if (count turtles with [color = red]) = 200 [stop]
```

devemos colocar esses dois comandos no procedimento **GO**, na linha imediatamente abaixo do comando **wait**

O número de *turtles* é definido, então podemos estabelecê-lo no gráfico, para isso basta usar o comando do NetLogo

```
set-plot-y-range 0 100
```

no procedimento **Setup**, antes da declaração **end**. Como a dimensão do tempo é infinita, nada a fazer com ela.

Verifique o que acontece.

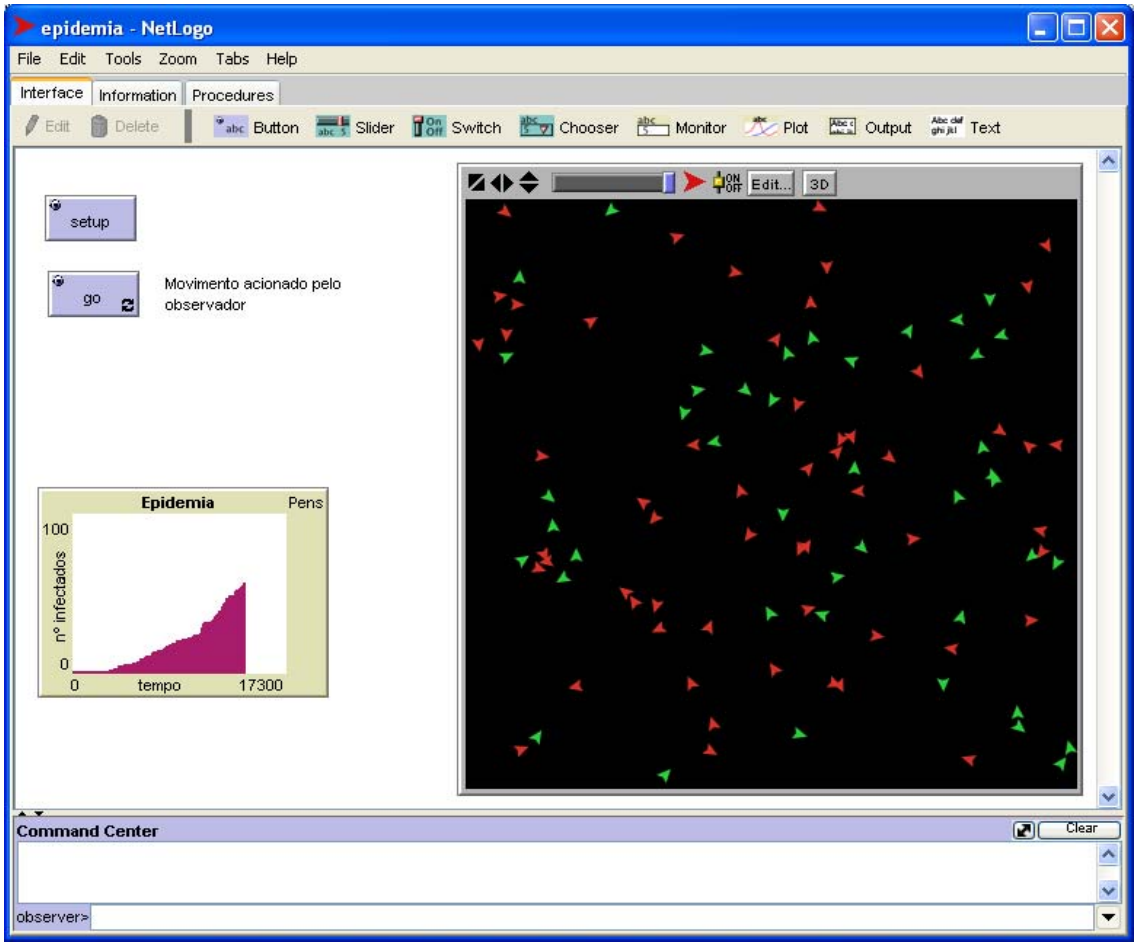


Figura 38 – Modelo complete em execuçao

Modelo Epidemia. (veja o modelo em execução)

Podemos trabalhar bem mais neste modelo com vários parâmetros (sliders), por exemplo, tornando as constantes variáveis, como o número de *turtles* iniciais, a quantidade inicial de *turtles* infectadas, usando botões de controle para isso.

Experimente!

Modelo Epidemia2 (veja o modelo em execução)

Obs. Até este momento, programamos a simulação para parar quando todos os agentes forem infectados.

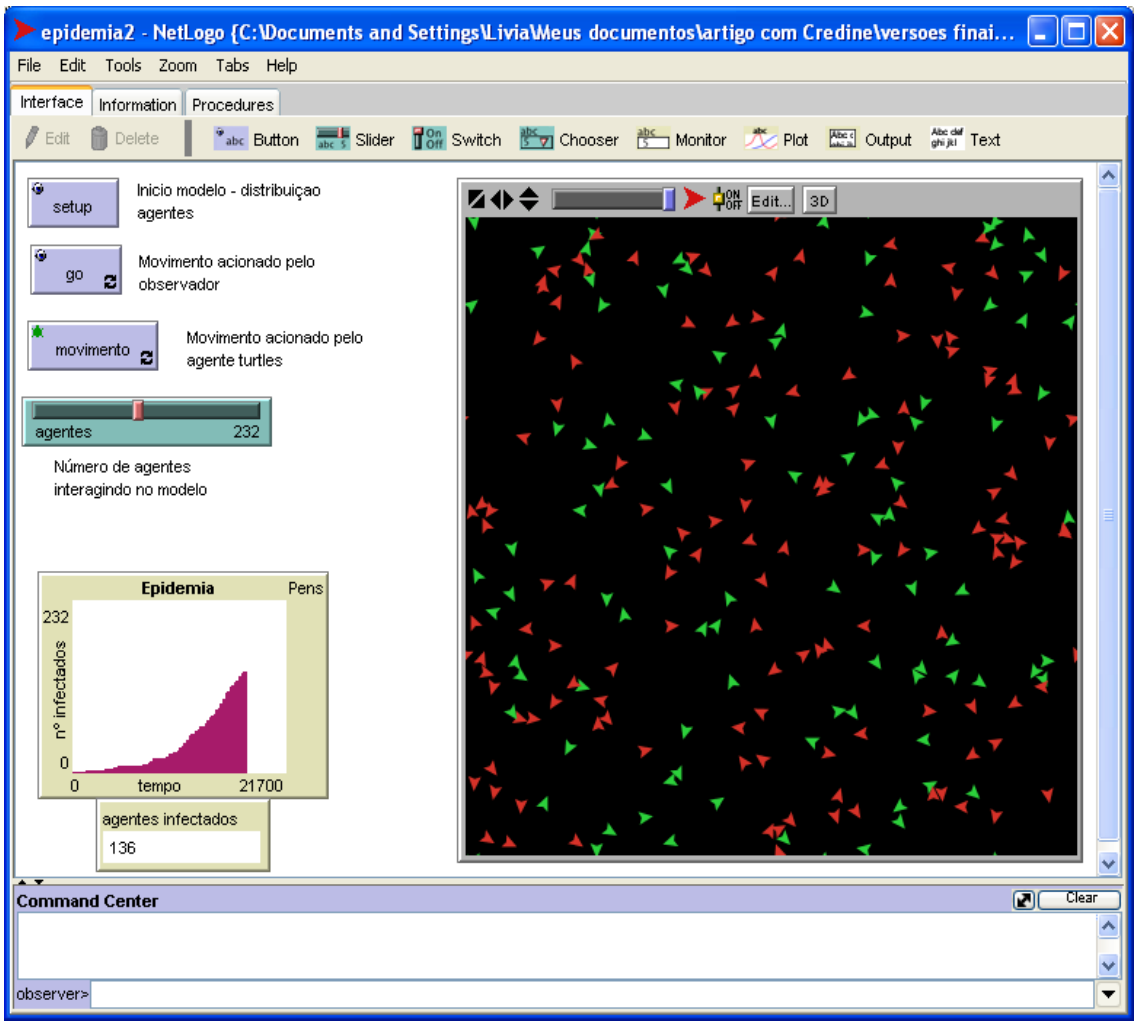


Figura 39 – Adicionar mais botões, na área de trabalho

7.3 Comentários

Bem, da forma como o problema foi abordado até o momento a tendência é que a epidemia se alastre e infecte toda a população. Entretanto, muitas outras questões podem ser levantadas baseada neste problema, por exemplo: uma vez que se detecte uma epidemia, que métodos podem ser adotados para combatê-la? É importante salientar que não pretendemos que o problema termine simplesmente com a infecção de toda a população, isto seria um modo muito simplista de encarar o problema. Algumas alternativas para combater o problema poderiam ser:

- Droga;

- Vacina;
- Controle de interação da população;
- Isolamento dos infectados;

Caso haja aumento da violência da infecção com mortes, que métodos poderiam ser adotados para preservar a espécie? Ou através das medidas adotadas, semelhantes às sugeridas acima, que mutações poderiam ocorrer com a população?

Aborde uma ou mais dessas alternativas e procure resolver o problema. Esse trabalho pode ser realizado individualmente ou em grupo.

Encaminhamentos:

- Discutir as soluções apresentadas e as estratégias de resolução;
- Analisar os caminhos adotados para programação.

8. Problemas para desenvolvimento posterior

8.1. Colônia de cupins

Desenvolvendo passo a passo modelo de colônia de cupins

Descrição e biologia - os cupins são insetos sociais organizados em castas, com funções definidas. Os operários fazem a limpeza e quase todo o trabalho do cupinzeiro. Os soldados são responsáveis pela defesa física ou química (toxinas ou substâncias pegajosas). Os reprodutores, rei e rainha, podem viver alguns anos e apresentam grande fecundidade.

Imagine que temos um cupinzeiro (cujos insetos são representados pelas turtles), vamos supor que o material para construção da colônia (no caso patches) esteja disperso e que os insetos tentem manter o ninho num faz e desfaz, isto é, re-arranjando a distribuição de material.

O que é necessário para descrever o modelo?

1. para começar crie apenas alguns cupins e o material para ser deslocado;

Implemente esse exemplo no NetLogo.Faça uso das facilidades do NetLogo para variações de parâmetros, contadores, exibição de gráficos, etc.

2. faça variações no modelo, incluindo as diferentes castas e as funções de cada uma. Dê forma diferente para cada um dos representantes da casta.

Analise o modelo. Observe o comportamento que emerge dessas interações.

comando manual - Modelo-1

comando manual - Modelo-2

com procedimento - Modelo-3

com procedimento e apenas uma espécie - Modelo-4

com procedimento e duas espécies trabalhando - Modelo-5

8.2. Aprisionamento

Suponha que exista um território livre onde está construído um castelo habitado por um monstro hostil que aprisiona todos os invasores que por lá apareça; um mago poderoso e inofensivo ao ataque do monstro, que transita livremente pelo território; e heróis destemidos que ficam desafiando o monstro, tentando entrar e sair do castelo sem o monstro perceber. O castelo tem apenas duas aberturas por onde o mago e os heróis podem passar. O mostro é cárcere permanente do território, isto é, ele não pode ir ao mundo exterior. O mago fica transitando livremente pelo território e libertando os heróis aprisionados. Após o inicio do jogo, um novo herói só surge se todos os demais estiverem aprisionados. O movimento das criaturas é aleatório.

Construa esse modelo e observe o seu desenvolvimento e as interações que emergem dos comportamentos descritos. Estime o número de heróis existentes no território passado alguns minutos de simulação; que relações podem ser observadas deste comportamento?

Modelo aprisionamento

8.3. Labirinto

Neste caso, o desafio é desenvolver uma estratégia para que os agentes (bolas vermelhas) alcancem o objetivo de sair do tabuleiro (*patches* verdes), através

das portas (*patches* pretos), contornando um labirinto fechado (*patches* azuis) de formas geradas randomicamente a cada execução do modelo.

O modelo atual tem duas estratégias pré-implementadas, chamadas de ***randômico*** e ***estratégia 1***. Em todos os casos a direção inicial é randômica.

Na estratégia ***randômico***, as *turtles* trocam de direção e procuram randomicamente a saída. Funciona "*sempre"*, mas pode demorar muito!

Na ***estratégia 1***, uma direção é escolhida e mantida o máximo possível. A troca de direção atende a um critério de prioridade pré-definido e arbitrário. Esta estratégia tende a ser mais rápida que a anterior, mas nem sempre funciona. Rode o modelo algumas vezes e descubra qual a limitação...

A ***estratégia 2*** não foi escrita, este é o seu desafio.

Modelo Labirinto

9. Recursos avançados do NetLogo

O objetivo principal deste tutorial foi guiar o aprendiz num primeiro contato com o ambiente NetLogo, portanto, exploramos com algum detalhe apenas os recursos mais básicos do ambiente e da linguagem de programação. Entretanto, à medida que você for se familiarizando com o NetLogo, poderá sentir a necessidade de explorar recursos mais avançados. Em seguida abordaremos, ainda que de maneira bastante superficial, alguns recursos e formas avançadas de utilização do NetLogo que consideramos mais importantes. Estes e outros recursos avançados estão detalhados no ***Manual do Usuário NetLogo*** http://ccl.northwestern.edu/netlogo/docs/ .

9.1. Experimentos controlados

Imagine que tenhamos um modelo de simulação com três parâmetros que possam variar de 100 valores distintos cada. Isso dá 1 milhão (100^3) de combinações. Algo praticamente impossível de testar uma a uma.

O NetLogo oferece uma ferramenta, acessível através do menu ***Tools***, opção ***BehaviorSpace***, para automatizar a construção e execução de experimentos controlados. Através deste recurso é possível determinar de que forma variar os parâmetros de um modelo, por quanto tempo insistir em cada

combinação ou outra condição de parada, ou mesmo quantas vezes o mesmo experimento deve ser repetido.

9.2. HubNet

O recurso **HubNet** possibilita a autoria e aplicação de atividades de *simulação participativa*, na sala de aula ou laboratório de informática. Esta ferramenta, acessível através do menu ***Tools***, opção ***HubNet Control Center***,[3] transforma o NetLogo em um servidor de simulações, que funciona numa rede de computadores em arquitetura cliente-servidor. Através do **HubNet** o professor pode disponibilizar simulações em que cada parte do modelo (parâmetros, agentes, etc.) seja controlada por um aluno através de seu computador (ou mesmo por um modelo específico de calculadora). A biblioteca de modelos do NetLogo 3.0 traz, na pasta ***HubNet Computer Activities***, alguns exemplos de simulações formatadas para a arquitetura **HubNet**.

9.3. Extensões

O ambiente NetLogo foi projetado de maneira que podem ser anexadas extensões ao programa básico. Desta maneira, podem ser adicionados novos recursos ao programa e comandos à linguagem de modelagem. Na prática, uma extensão é um arquivo "***jar***" (*java archive*) com determinadas características.

Usuários de NetLogo podem usar uma extensão indicando no início do programa a cláusula ***__extensions ["NomeDaExtensão.jar"]***. Programadores da linguagem Java podem escrever novas extensões para o NetLogo atendendo às especificações descritas na documentação.

Extensões podem acrescentar características interessantes, por exemplo, ***GoGo*** é uma extensão que permite conectar o NetLogo ao mundo físico através de uma placa de baixo custo, que você mesmo pode construir. Esta possui 8 portas para sensores e 4 portas para atuadores. Esse recurso tem inúmeras aplicações educacionais, como robótica, meio ambiente, automação de laboratórios, entre outras. Informações sobre a placa GoGo e projetos educacionais desenvolvidos com ela em www.gogoboard.org.

---

[3] A ***opção HubNet Control Center*** somente fica disponível se o modelo aberto estiver configurado para arquitetura HubNet.

9.4. Modelagem da dinâmica de sistemas

Na abordagem baseada em agentes, programamos as regras de comportamento individual de agentes que, ao interagirem, produzem um sistema dinâmico. Diferentemente, a Dinâmica de Sistemas é um tipo de abordagem em que programamos o comportamento de cada população como um todo para observar como duas ou mais populações se auto-influenciam.

NetLogo oferece uma ferramenta, acessível através do menu ***Tools***, opção ***System Dynamics Modeler***, que nos possibilita construir um diagrama que relaciona populações (***stocks***) dinâmicas e como elas afetam umas às outras. A biblioteca de modelos do NetLogo 3.0 traz uma nova versão do exemplo do sistema dinâmico presa / predador entre lobos e ovelhas, no qual é possível comparar a modelagem baseada em agentes com a modelagem baseada em Dinâmica de Sistemas.

Apêndice A

Relação de comandos NetLogo

Categoria	Comando	Descrição
observador	**Clear-all (ca)**	Limpa tela
	Create (crt)	Cria *turtles* e patches
Turtles **patches**	**pxcor / pycor** **xcor / ycor**	Especifica as coordenadas x e y para patches Especifica as coordenadas x e y para *turtles*
	count **count turtles with [color = blue]** **count patches with [pcolor = 45]**	Conta o número de agentes com condição solicitada
	pcolor **color**	Atributo de cor dos agentes *patches* Atributo de cor dos agentes *turtles*
	screen-edge-x **screen-edge-y** **screen-size-x** **screen-size-y**	Apresenta as distancia (nos eixos x e y, respectivamente) da origem aos extremos da tela Corresponde a 2 x screen-edge-x (y)
Turtles	**heading ângulo** **right (rt) ângulo** **left (lt) ângulo**	Indica a direção (ângulo) para a qual esta a turtle
	forward número **fd 10**	Solicita a *turtle* andar numero passos *turtle* para frente
	turtle número	Para chamar uma turtle especifica
	back número **bk número**	Solicita a *turtle* andar numero passos *turtle* para trás
Comando Geral	**set variable value**	Atribui valor para uma variável local
	random número **random 3**	Escolhe um numero randomicamente Escolhe um valor entre 0, 1 e 2. Obs.: Há variações do tipo de números a serem selecionados, inteiro, float, exponencial
	jump número	Solicita turtle saltar
	to procedure-name	Para iniciar um procedimento
	end	Finaliza um procedimento
	ask **ask turtles [comando]** **ask patches [comando]**	Solicitar que os agentes executem algum comando
	globals [var1 var2 ...]	Define variáveis globais, deve ser no inicio do programa