Simulação com Agentes

Fernando dos Santos[1,2]

[1]Univ. Federal do Rio Grande do Sul (UFRGS)
[2]Univ. do Estado de Santa Catarina (UDESC)

Minicurso Prático

Sobre o Autor

Fernando dos Santos
(fernando.santos@udesc.br)

- Natural de Blumenau/SC
- Formação
  - Graduação: Ciência da Computação – FURB (2001–2005)
  - Mestrado: Ciência da Computação – UFRGS (2007–2009)
    - Sistemas multiagente: busca e resgate em desastres
  - Doutorado: Ciência da Computação – UFRGS (2015–atual)
- Atuação
  - Desenvolvedor/analista: 2006 e 2009–2010
  - Professor FURB: 2010–2011
  - Professor UDESC: 2011–atual

Sobre a Tese de Doutorado

Questão de Pesquisa

Como tornar mais eficiente o desenvolvimento de simulações com agentes?

Hipótese

Adotar abordagem de desenvolvimento dirigido a modelos (*model-driven development*)

Artefatos da Abordagem

- Linguagem de modelagem
- Ferramenta para modelagem e geração de código

Objetivo do minicurso para o autor

- Avaliar a eficiência da abordagem dirigida a modelos (*model-driven*)

Sobre o Minicurso

Objetivos do minicurso para os participantes:

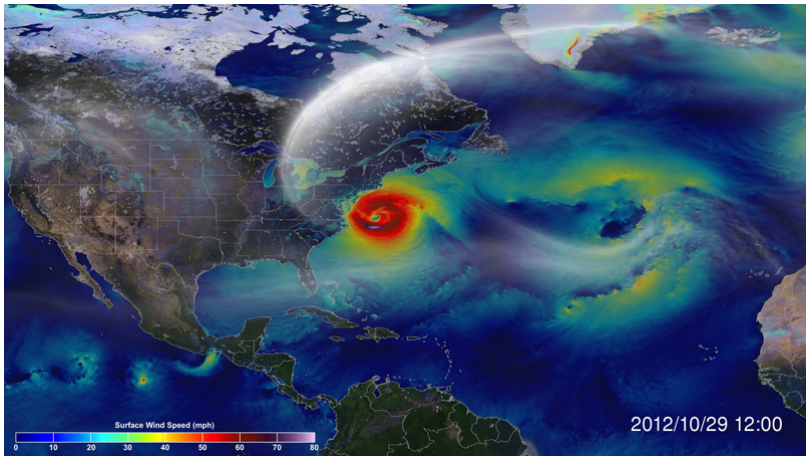
- Conhecer o paradigma de simulações com agentes
- Praticar o desenvolvimento de simulações com agentes

Conteúdo

- Fundamentos de simulação com agentes
- Ferramenta de simulação com agentes
  - NetLogo e sua linguagem programação
- Abordagem *model-driven* para desenvolver simulações
  - MDD4ABMS e sua linguagem de modelagem DSL4ABMS
- Práticas em desenvolvimento de simulações e estudo de usabilidade/efetividade das abordagens utilizadas
- Fechamento: alternativas e perspectivas

Parte 1: Fundamentos

Simulação

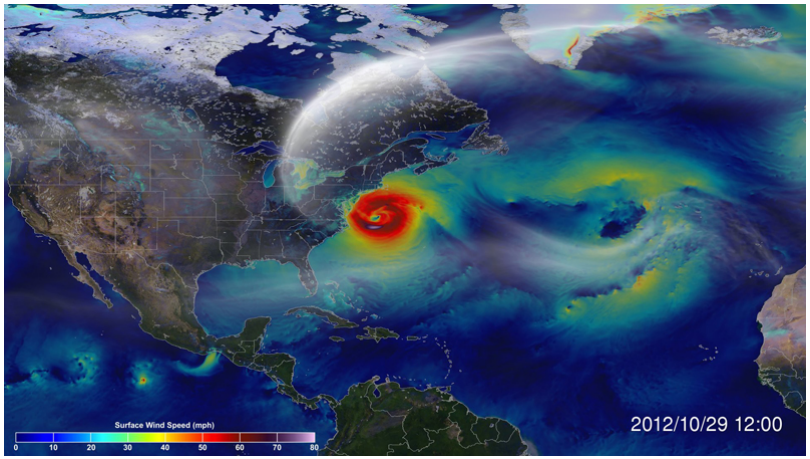


Simulação NASA: Furação Sandy 2012

Simulação de Tráfego

O que é uma **simulação**?

Simulação

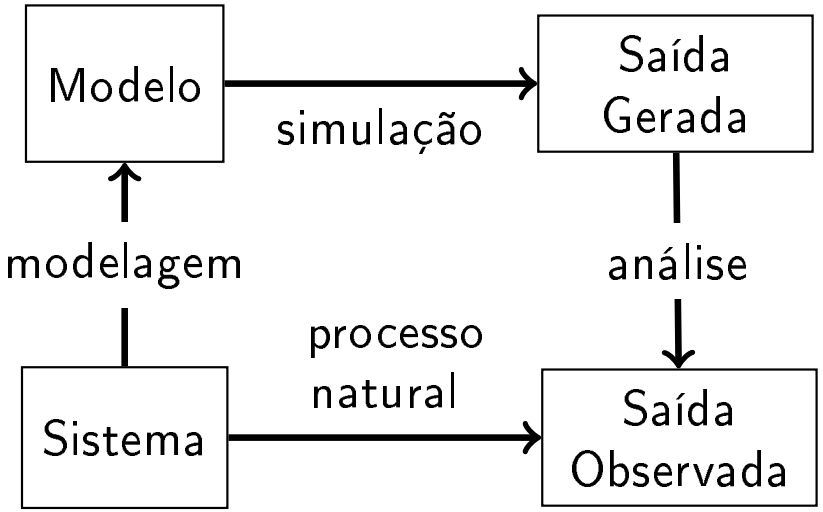


Simulação NASA: Furação Sandy 2012

Simulação de Tráfego

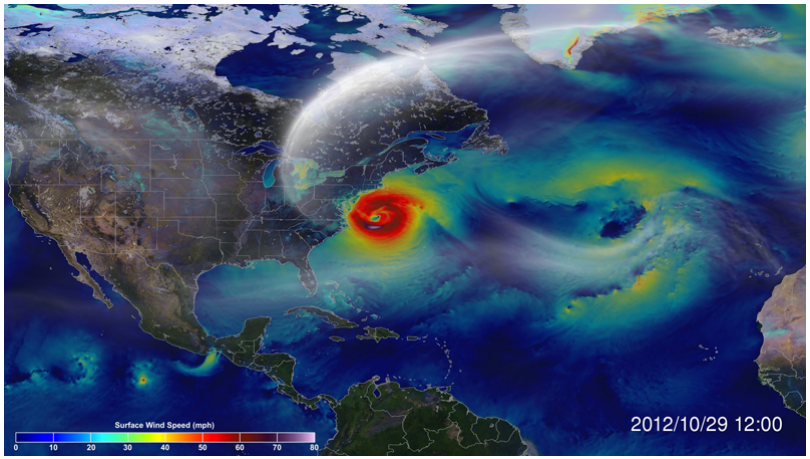
O que é uma **simulação**?

- **Execução** do **modelo** de um **sistema**



Simulação Computacional: Benefícios?

Simulação

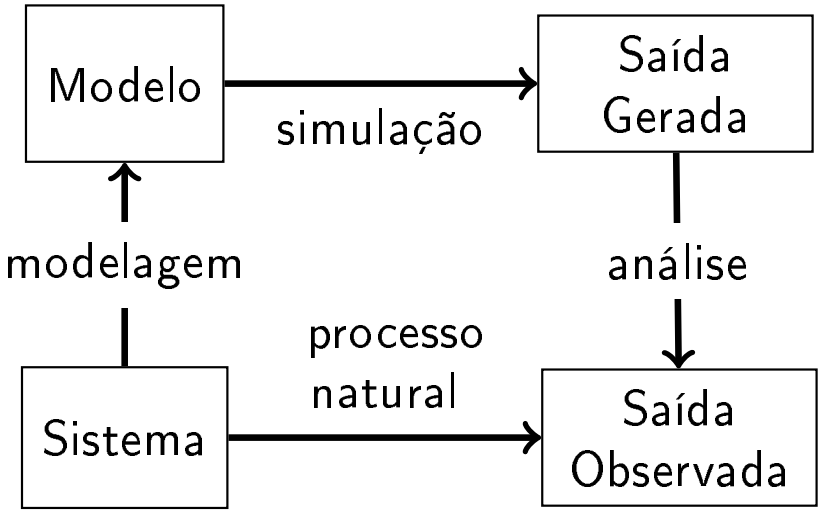


Simulação NASA: Furação Sandy 2012

Simulação de Tráfego

O que é uma **simulação**?

- **Execução** do **modelo** de um **sistema**



Simulação Computacional: Benefícios?

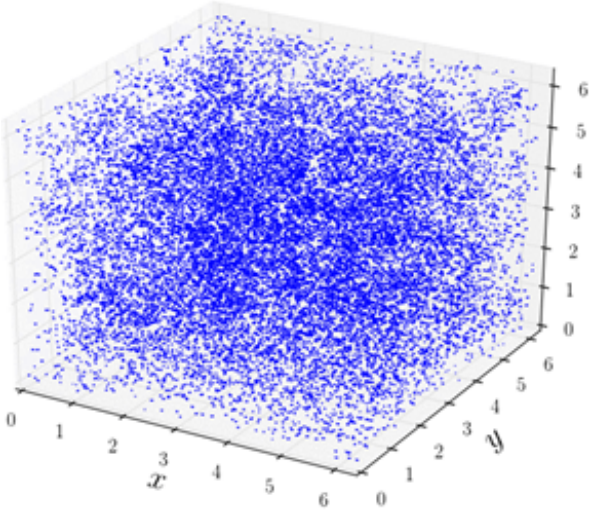
- Livre de riscos
- Baixo custo
- Insights → Tomada de Decisão

Simulação com Agentes

Usa **agentes simulados** para reproduzir um sistema

Bando de pássaros

Cardume de peixes



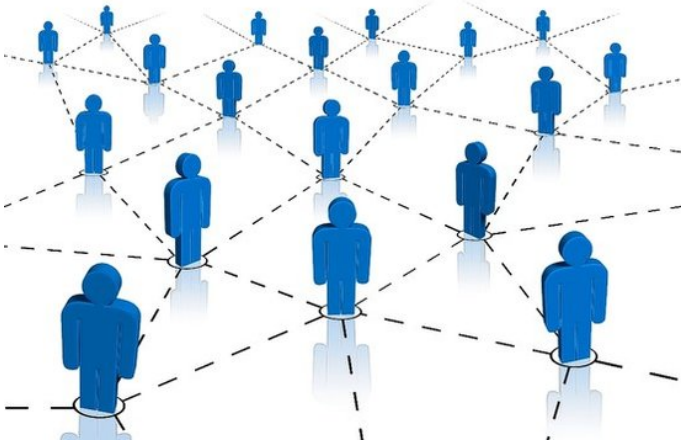
Agentes simulados

Diferenciais da Simulação com Agentes

Em relação à outros paradigmas de simulação

Foco nos **indivíduos** (agentes):

- **Heterogeneidade** dos indivíduos afeta o sistema
  - Indivíduos com características e comportamentos diferentes
- **Interações** entre os indivíduos afetam o sistema
  - Sujeitas (ou não) à restrições espaciais/topológicas



Incorporar técnicas de Inteligência Artificial nos indivíduos

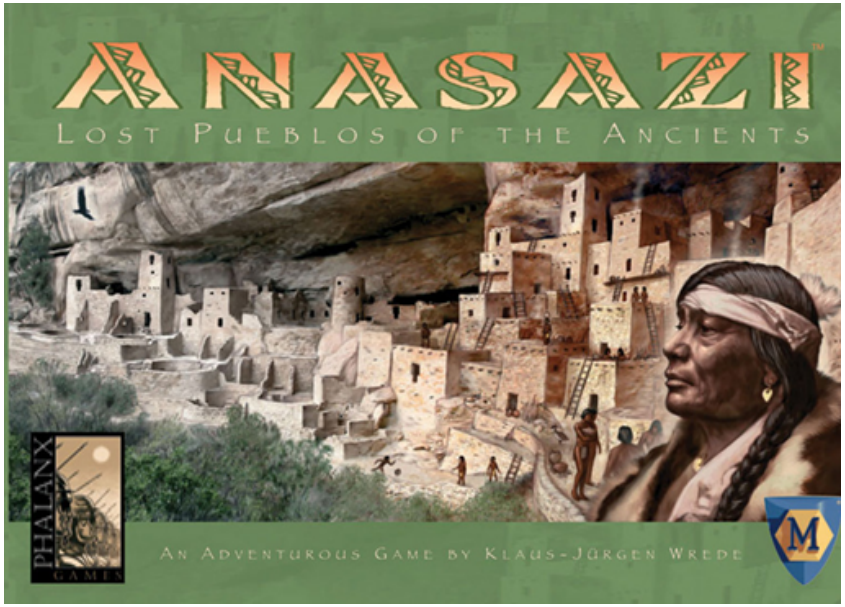
- Comportamento inteligente (aprendizagem, redes neurais, etc)

Simulação com Agentes

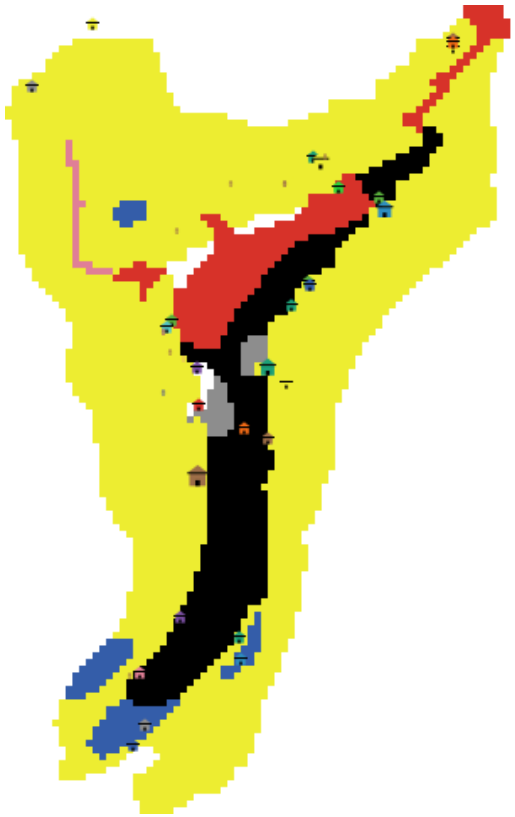
Exemplos

Anasazi Civilization (Arizona/USA–1200 A.C.)

- Mudanças climáticas, sozinhas, não explicam a extinção



Civilization



Simulation

Simulação com Agentes

Exemplos

Intelligent Transportation Systems

- Controle semafórico, escolha de rotas (motoristas), ...

Porto Alegre/RS

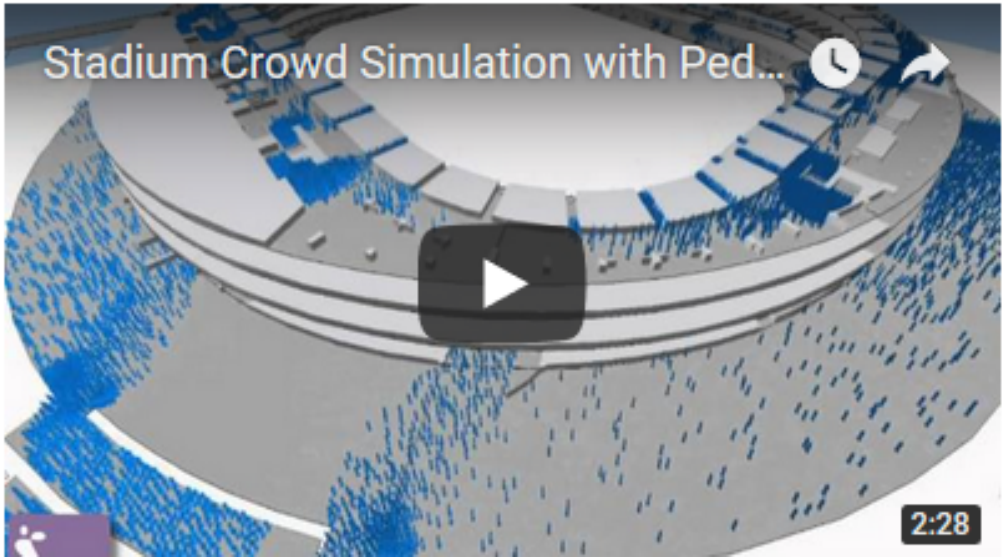
Simulações de Tráfego

Simulação com Agentes

Exemplos

Evacuação em emergências

- Projeto do Estádio Arena do Grêmio (Porto Alegre)

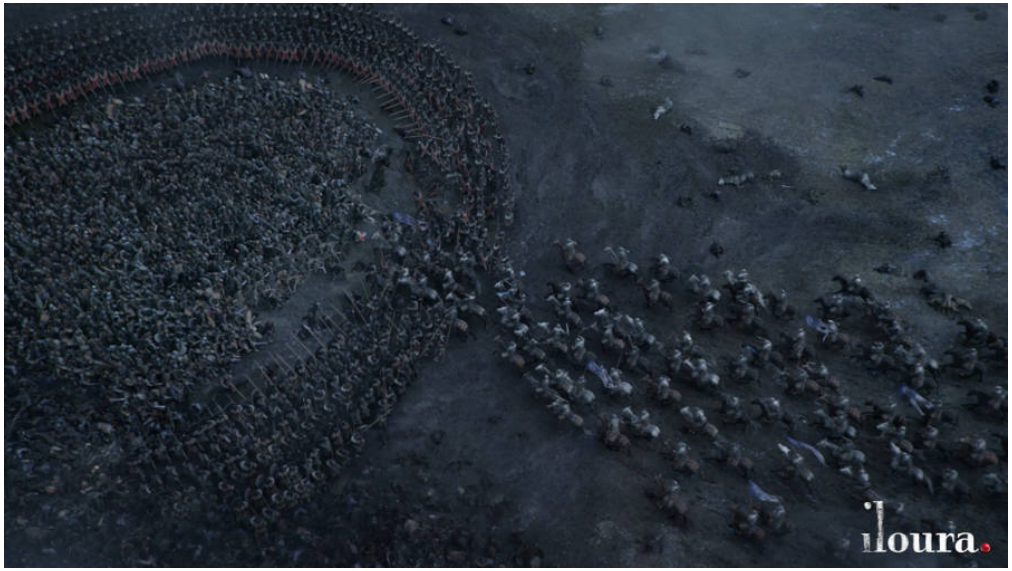


Simulação com Agentes

Exemplos

Simulação de Multidões

- Game of Thrones: Battle of the Bastards



Massive Software Demo Reel

25,509 views

71 7 SHARE

Simulação com Agentes (ABS)

Elementos

Elementos **primários** de uma Simulação com Agentes (ABS[1]):

- Agentes, Ambiente (*environment*), e Interações

[1] Agent-based Simulation

Simulação com Agentes (ABS)

Elementos

Elementos **primários** de uma Simulação com Agentes (ABS[1]):

- Agentes, Ambiente (*environment*), e Interações

Agentes

- Entidades autônomas
- Tomam decisões e atuam conforme seus objetivos
- Podem adotar técnicas de I.A. (ex.: aprendizagem, raciocínio)

Ambiente

- Espaço (paisagem) onde os agentes estão situados
- Engloba também todas as outras entidades não autônomas (ex.: objetos, casas, ruas, árvores)

Interações

- Contatos e relações entre os agentes (ex.: troca de mensagens)

[1] Agent-based Simulation

Simulação com Agentes (ABS)

Elementos Auxiliares

Representação do **Tempo**

- Discreto: o tempo é discretizado em *timesteps*
  - Convenciona-se qual *extensão temporal* é representada por um *timestep* (ex.: segundo, hora, dia, semana, etc)
  - A cada *timestep*, um ciclo de *percepção-decisão-ação* é executado por cada agente
- Continuo: o tempo é contínuo
  - Agentes *percebem-decidem-agem* de forma assíncrona

Saídas (Outputs)

- Dados coletados para analisar o resultado da simulação.

Como desenvolver ABS?

Usando linguagens de propósito geral

- Frequentemente, são mais conhecidas por desenvolvedores
- Implementar agentes, ambiente e interações "do zero"

Como desenvolver ABS?

Usando linguagens/plataformas específicas para ABS

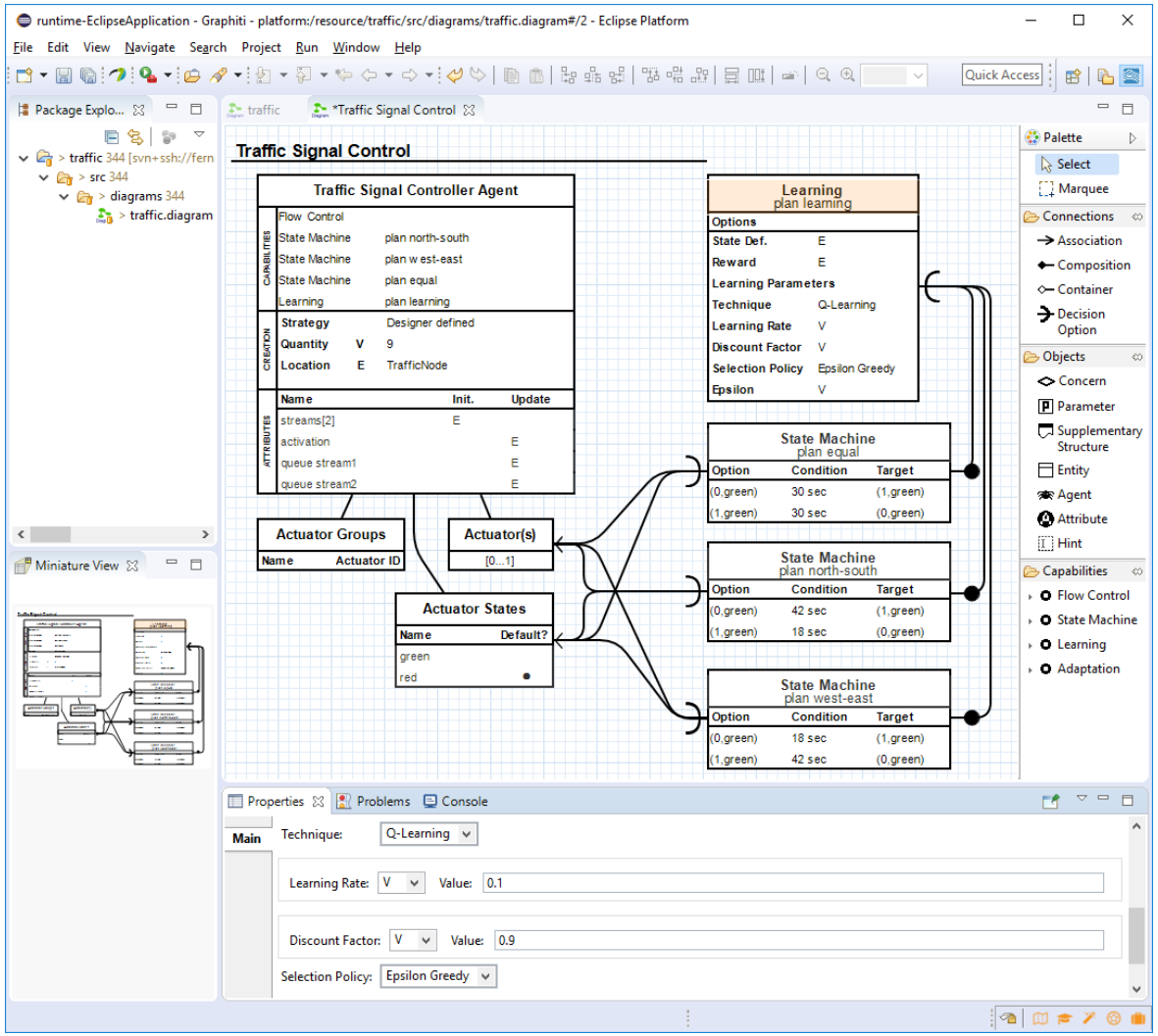
- Abstrações e comandos de mais alto nível
- Requerem treinamento e, muitas vezes, conhecimento de programação

M A S O N

Como desenvolver ABS?

Usando abordagem **model-driven**

- Modelagem usando elementos expressivos
- Geração de código automatizada



MDD4ABMS

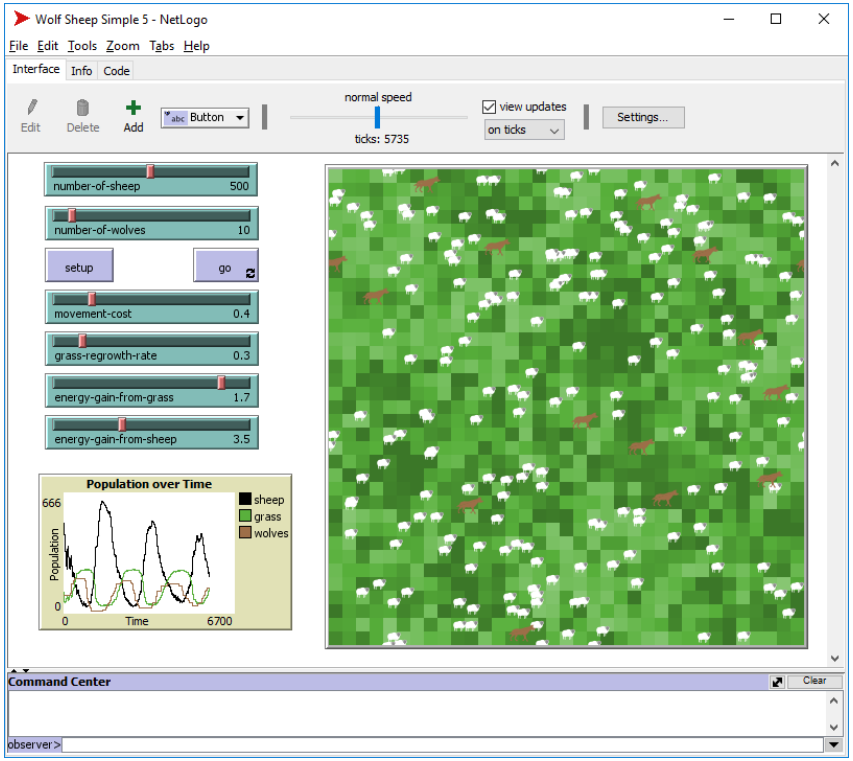
Parte 2. Práticas em Desenvolvimento de ABS

(NetLogo e MDD4ABMS)

2.1.NetLogo

A Ferramenta NetLogo

Ambiente programável para desenvolver simulações com agentes



Possui linguagem de programação própria

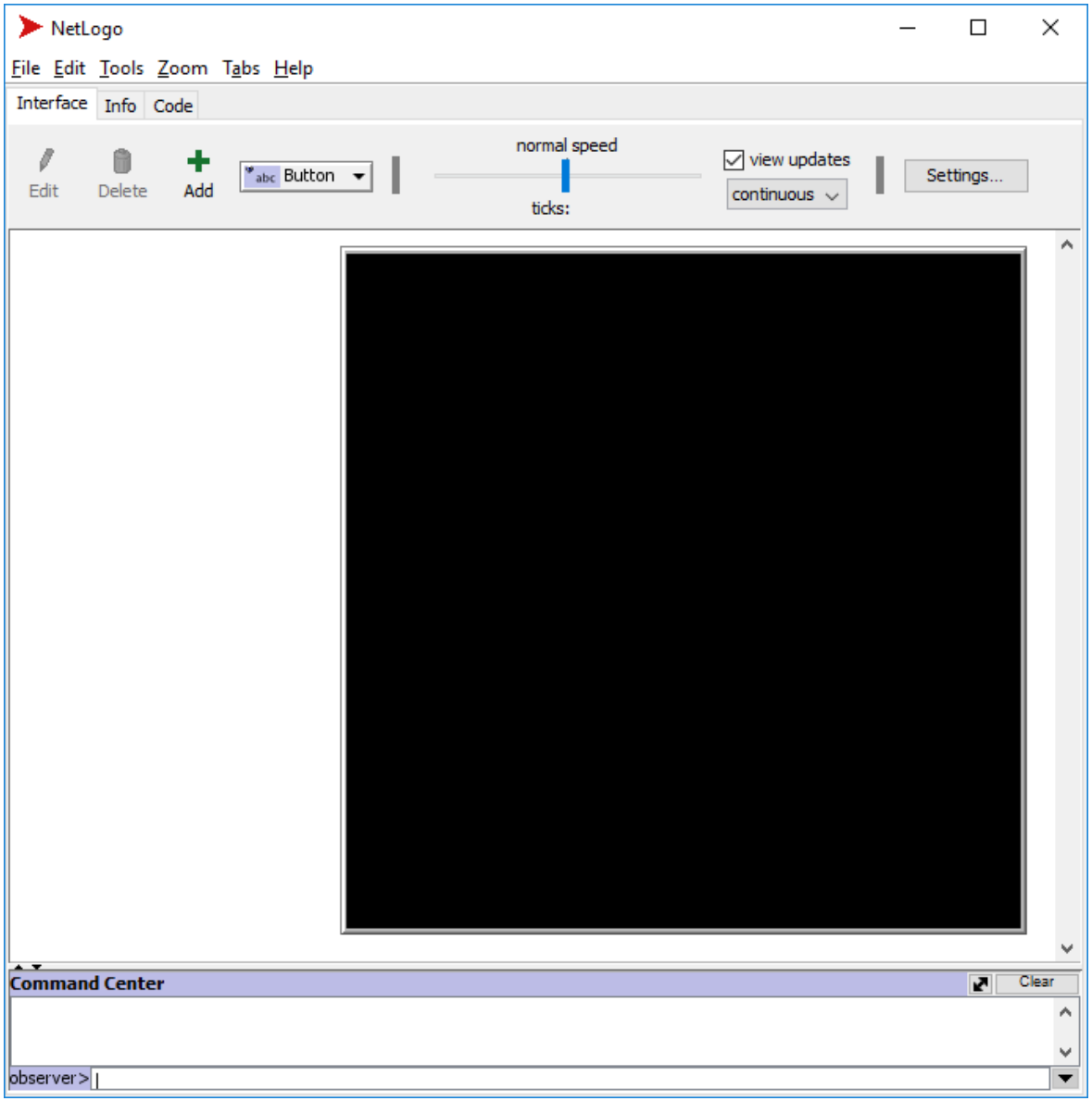
- Comandos facilitam manipulação dos objetos da simulação
- Sintaxe específica (é diferente de Java, C++, etc)

`https://ccl.northwestern.edu/netlogo/`

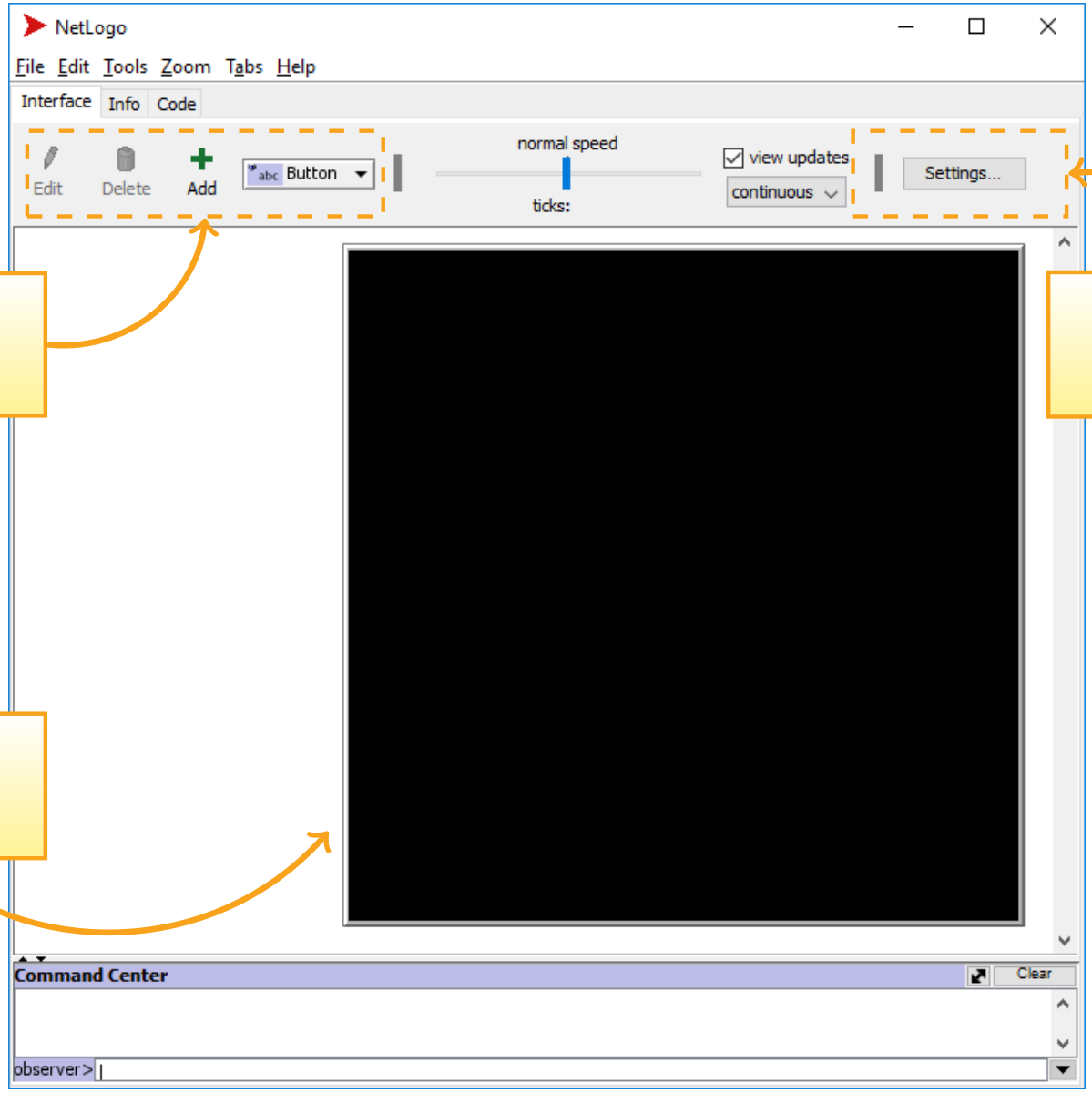
NetLogo: tela principal

O NetLogo é formado por **abas**:

- Interface
- Info
- Code



NetLogo: aba **Interface**



Componentes Visuais

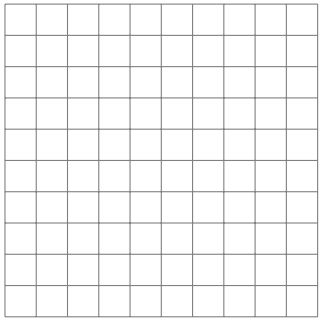
Config. do Ambiente

Visualização do Ambiente

NetLogo: Configurações do Ambiente

Ambiente é formado por ***patches***

- "células"



Origin:

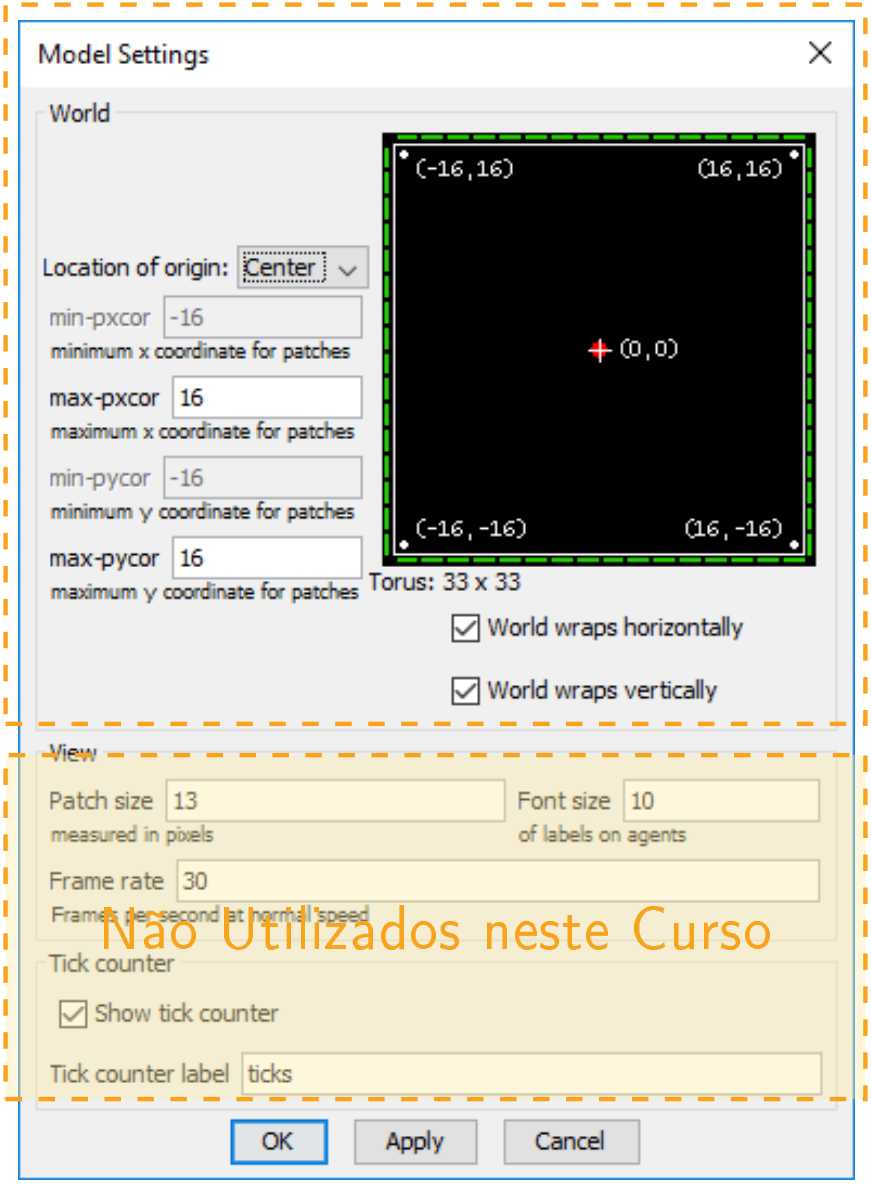
- posição do *patch* (0,0)

Min/max pxcor/pycor:

- dimensões do ambiente

World wraps:

- se o ambiente "se dobra" na horizontal/vertical



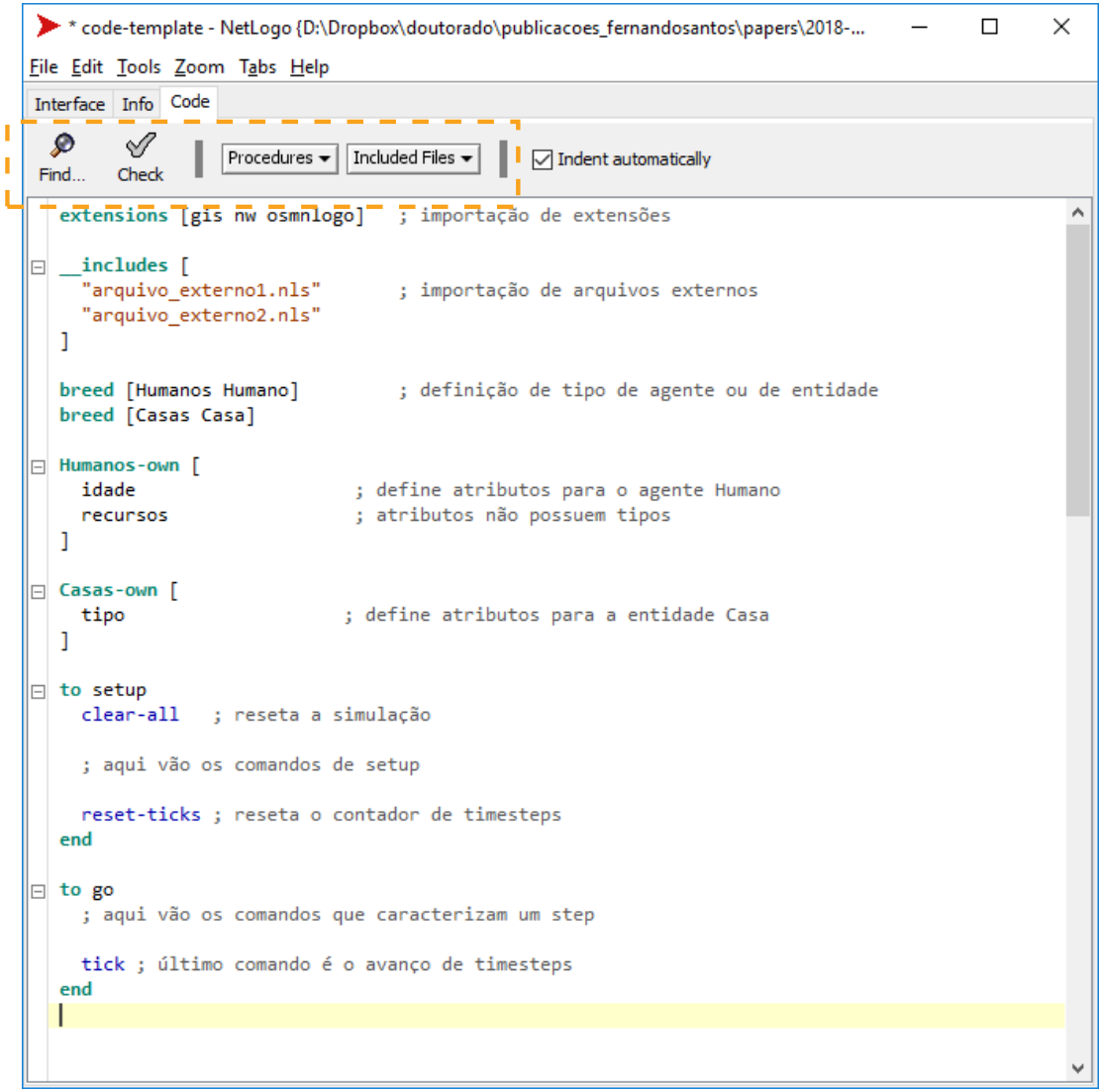
NetLogo: aba **Code**

Botão **Check**:

- Compila o código

Botões para acesso a:

- Procedures
- Included files



NetLogo: A linguagem de programação (1)

Emprega conceitos similares à **Orientação a Objetos (OO)**:

OO	**NetLogo**
Classe	Breed ("espécie"): `breed[Humanos Humano]`

Attributo	Breed-own ("característica da espécie"):
	`Humanos-own [`
	`  idade`
	`]`

Método
(sem retorno)
	Procedure:
	`to criarAgentes`
	`  comandos`
	`end`

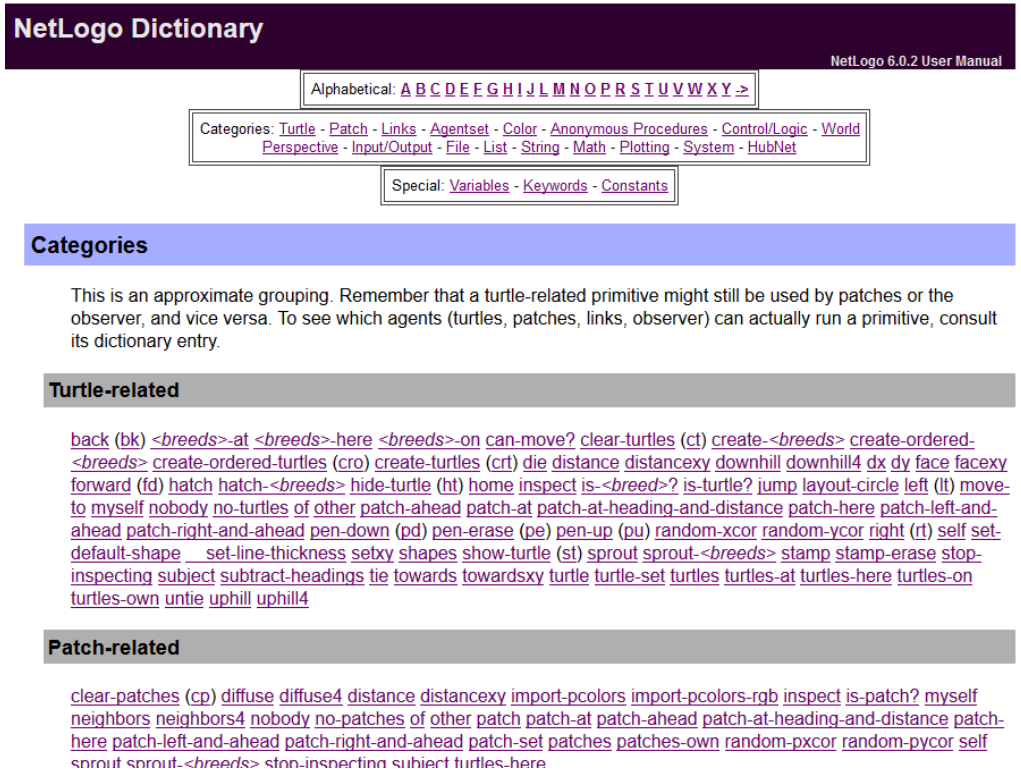
Método
(com retorno)
	Reporter:
	`to-report calcularTamanho`
	`  comandos`
	`  report valor`
	`end`

NetLogo: A linguagem de programação (2)

Onde encontrar a lista de comandos disponíveis?

NetLogo Dictionary:

- `https://ccl.northwestern.edu/netlogo/docs/dictionary.html`



NetLogo Dictionary

NetLogo 6.0.2 User Manual

Alphabetical: A B C D E F G H I J L M N O P R S T U V W X Y ->

Categories: Turtle - Patch - Links - Agentset - Color - Anonymous Procedures - Control/Logic - World Perspective - Input/Output - File - List - String - Math - Plotting - System - HubNet

Special: Variables - Keywords - Constants

Categories

This is an approximate grouping. Remember that a turtle-related primitive might still be used by patches or the observer, and vice versa. To see which agents (turtles, patches, links, observer) can actually run a primitive, consult its dictionary entry.

Turtle-related

back (bk) *<breeds>*-at *<breeds>*-here *<breeds>*-on can-move? clear-turtles (ct) create-*<breeds>* create-ordered-*<breeds>* create-ordered-turtles (cro) create-turtles (crt) die distance distancexy downhill downhill4 dx dy face facexy forward (fd) hatch hatch-*<breeds>* hide-turtle (ht) home inspect is-*<breed>*? is-turtle? jump layout-circle left (lt) move-to myself nobody no-turtles of other patch-ahead patch-at patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead pen-down (pd) pen-erase (pe) pen-up (pu) random-xcor random-ycor right (rt) self set-default-shape __set-line-thickness setxy shapes show-turtle (st) sprout sprout-*<breeds>* stamp stamp-erase stop-inspecting subject subtract-headings tie towards towardsxy turtle turtle-set turtles turtles-at turtles-here turtles-on turtles-own untie uphill uphill4

Patch-related

clear-patches (cp) diffuse diffuse4 distance distancexy import-pcolors import-pcolors-rgb inspect is-patch? myself neighbors neighbors4 nobody no-patches of other patch patch-at patch-ahead patch-at-heading-and-distance patch-here patch-left-and-ahead patch-right-and-ahead patch-set patches patches-own random-pxcor random-pycor self sprout sprout-*<breeds>* stop-inspecting subject turtles-here

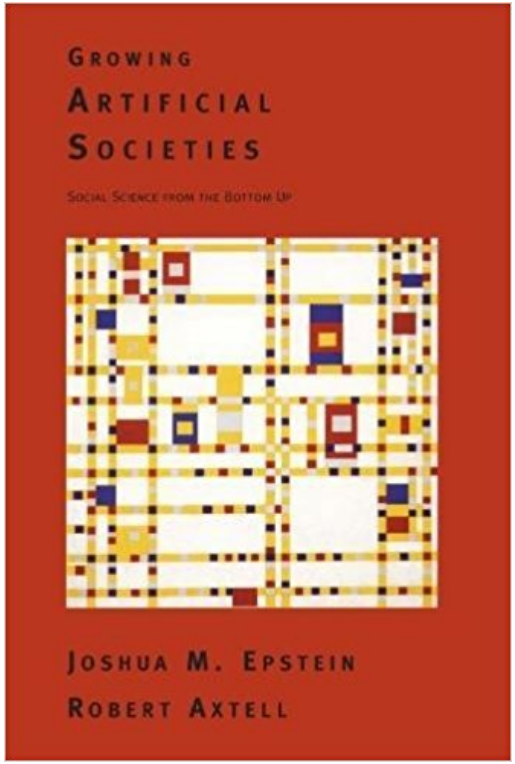
2.2. Desenvolvendo Simulações com NetLogo

Simulação Sugarscape

Conceitos

Sugarscape (ou, *"Paisagem de Açúcar"*)

- Simulação de sociedade artificial / vida artificial
- Pioneira na área de simulação com agentes
  - [Epstein and Axtell, 1996]
- Propõe mudança na forma de analisar um fenômemo:
  - ~~Como explicá-lo?~~
  - Como replicá-lo?



Simulação Sugarscape

Conceitos (continuação)

Metáfora para sociedades que dependem de **recursos**

- **Açúcar** (sugar) é o recurso simulado
  - Agentes consomem açúcar para sobreviver
- Recursos no mundo real: água, **metais**, calorias, dinheiro...

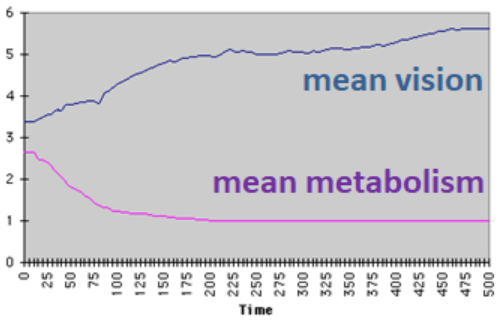
Simulação Sugarscape

Conceitos (continuação)

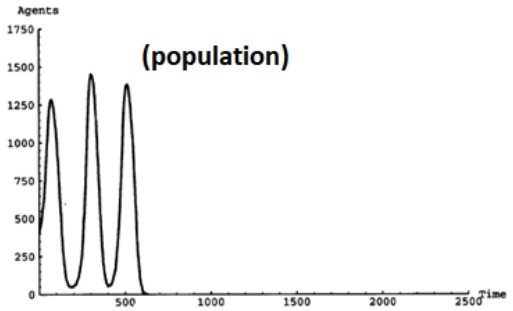
Comportamento dos Agentes:

- Percebem açúcar nas redondezas, através da **visão**.
- Coletam açúcar, que se transforma em **energia**.
- Possuem **metabolismo**: consome energia para manter **vida**.
- Se movem pelo ambiente, buscando açúcar para coletar.

Exemplos de fenômenos replicáveis com esta simulação:



(a) Evolução



(b) Extinção

(c) Ondas migratórias

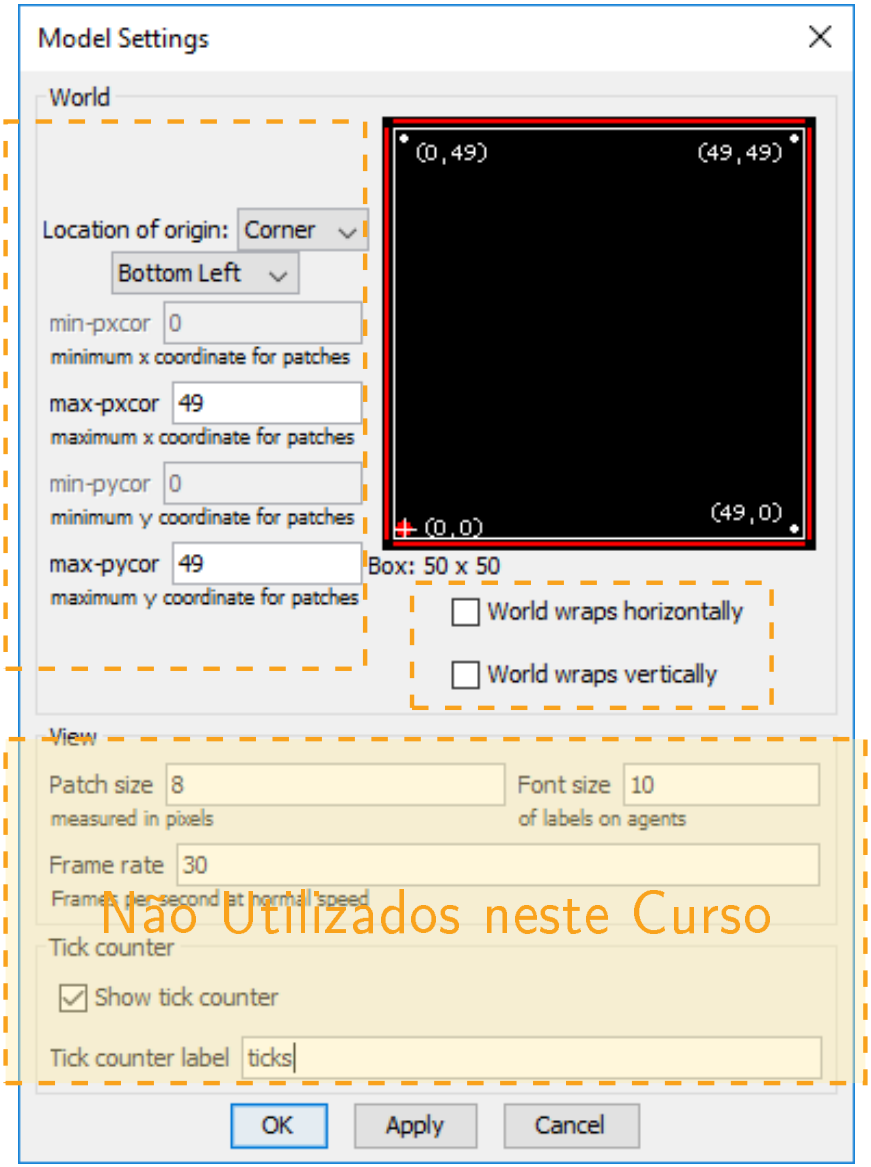
Sugarscape NetLogo: o Ambiente

Ambiente

- Grid, de 50x50 unidades
- Cada *patch* dispõe de uma quantia de açúcar.

No **NetLogo**:

- Grid, 50x50 *patches*
- Origin: *corner*, *bottom left*
- Sem *world wrap*



Sugarscape NetLogo: o Agente (primeira versão)

Comportamento a ser desenvolvido:

- Se move pelo ambiente
  - Para um *patch* não ocupado dentro de um **raio de visão**
- Permanece vivo enquanto **nível de energia** for maior que zero
- Possui um **metabolismo**
  - Faz a energia diminuir a cada passo da simulação.

Quantos atributos do agente são derivados a partir disto?

Sugarscape NetLogo: o Agente (primeira versão)

Comportamento a ser desenvolvido:

- Se move pelo ambiente
  - Para um *patch* não ocupado dentro de um **raio de visão**
- Permanece vivo enquanto **nível de energia** for maior que zero
- Possui um **metabolismo**
  - Faz a energia diminuir a cada passo da simulação.

Quantos atributos do agente são derivados a partir disto?

- raioVisao
- energia
- metabolismo

Sugarscape NetLogo: o Agente

Implementação do tipo de agente e seus atributos

```
extensions [gis nw osmnlogo]

breed [Individuos Individuo]

Individus-own [
   raioVisao
   energia
   metabolismo
]
```

Detalhes dos comandos NetLogo:

`extensions[ ext1 ...extN]`

- Carrega extensões do NetLogo

`breed [NomePlural NomeSingular]`

- Define um tipo de agente ou entidade

`NomePlural-own [ ...]`

- Define atributos do agente ou entidade

Sugascape NetLogo: o Setup

Setup da simulação

- É a rotina que prepara a simulação
- Inicialização do ambiente (quando necessário)
- Criação e inicialização dos agentes

Em NetLogo, o padrão é criar um procedimento **setup**:

```
to setup
  ; comandos que fazem o setup, exemplos:
  ;   inicialização do ambiente
  ;   criação e inicialização dos agentes
end
```

Sugascape NetLogo: o Setup

Setup da simulação: criação de agentes (1)

Criar 100 agentes **Individuos** em posições aleatórias:

```
to setup
  clear-all
  reset-ticks
  ask n-of 100 patches [
    sprout-Individuos 1 [
      set raioVisao 5
      set energia 0
      set metabolismo 1
    ]
  ]
end
```

`clear-all`

- Remove qualquer entidade/agente existente

`reset-ticks`

- Reinicia o contador de timesteps

`ask` "quem" ["faça isto"]

- Pede para que certos agentes (ou entidades ou patches — o "quem") execute os comandos
- `n-of valorN entidade`:
  - seleciona `valorN` entidades aleatorias

O comando **ask** é muito usado. Lembrar: "obj.metodo()"

Sugascape NetLogo: o Setup

Setup da simulação: criação de agentes (2)

Criar 100 agentes **Individuos** em posições aleatórias:

```
to setup
  clear-all
  reset-ticks
  ask n-of 100 patches [
    sprout-Individuos 1 [
      set raioVisao 5
      set energia 0
      set metabolismo 1
    ]
  ]
end
```

`sprout-NomePlural quantidade`

- `sprout` é executado apenas por *patches*
- Faz "brotar" uma `quantidade` de agentes no *patch*
- Cada agente executa os comandos que estão entre `[...]`

Sugascape NetLogo: sobre Tipos de Dados

Quais são os **tipos de dados** no NetLogo?

```
Humanos-own [
  raioVisao
  energia
  metabolismo
]
to setup
  clear-all
  reset-ticks
  ask n-of 100 patches [
    sprout-Individuos 1 [
      set raioVisao 5
      set energia 0
      set metabolismo 1
    ]
  ]
end
```

- **boolean**: `true / false`
- **numérico**: `0.5, 1, 99.4, -3, -5.5`
  - Indiferente se inteiro ou decimal
- **agente**: instância de agente, ou entidade, ou patch
  - `set vizinho Individuo 25`
- **agentset**: **conjunto** de agentes, ou entidades, ou patches
  - `nomePlural` identifica *agentset* com todos da mesma espécie
  - `n-of 200 patches`
    - *agentset* com 200 patches
  - os elementos **não são ordenados**

Sugascape NetLogo: os Comportamentos

Os comportamentos (ou habilidades) dos agentes são implementadas como *procedures*.

Procedure para movimentar o agente:

```
to moverAgente



end
```

Sugascape NetLogo: os Comportamentos

Os comportamentos (ou habilidades) dos agentes são implementadas como *procedures*.

Procedure para movimentar o agente:

```
to moverAgente

  ; identificar os patches visiveis (dentro do raio de visao)
  ; dentre os patches visiveis, identificar os não ocupados
  ; escolher aleatoriamente um dos patches visiveis livres e mover

end
```

Sugascape NetLogo: os Comportamentos

Movimentação do agente (1)

```
to moverAgente
  ; identificar os patches visiveis (dentro do raio de visao)
  let patchesVisiveis patches in-radius (raioVisao + 0.5)

  ; dentre os patches visiveis, identificar os não ocupados
  ; escolher aleatoriamente um dos patches visiveis livres e mover
end
```

`let <variavel> <valor>`

- Declara uma **variável local** e define seu valor

`<agentset> in-radius <valorRaio>`

- Recupera os itens do *agentset* situados no raio
  - Acréscimo de 0.5 é para também incluir *patches* em diagonal

Sugascape NetLogo: os Comportamentos

Movimentação do agente (2)

```
to moverAgente
  ; identificar os patches visiveis (dentro do raio de visao)
  let patchesVisiveis patches in-radius (raioVisao + 0.5)

  ; dentre os patches visiveis, identificar os não ocupados
  let patchesVisiveisLivres patchesVisiveis with [ not any? other individuos-here ]

  ; escolher aleatoriamente um dos patches visiveis livres e mover
end
```

`<agentset> with [<restrição>]`

- Especifica um **filtro** para os elementos do *agentset*
- O filtro é uma **expressão lógica**

`any? <agentset>`:

- Retorna `true` caso o *agentset* **não esteja vazio**

Sugascape NetLogo: os Comportamentos

Movimentação do agente (2)

```
to moverAgente
  ; identificar os patches visiveis (dentro do raio de visao)
  let patchesVisiveis patches in-radius raioVisao

  ; dentre os patches visiveis, identificar os não ocupados
  let patchesVisiveisLivres patchesVisiveis with [ not any? other Individuos-here ]

  ; escolher aleatoriamente um dos patches visiveis livres e mover
end
```

`<agentset>-here`:

- Retorna todas as entidades que estão no *patch* em que o agente está situado

`other <agentset>`:

- Retorna os elementos do <agentset> **sem** o agente que está executando o comando

Sugascape NetLogo: os Comportamentos

Movimentação do agente (3)

```
to moverAgente
  ; identificar os patches visiveis (dentro do raio de visao)
  let patchesVisiveis patches in-radius raioVisao

  ; dentre os patches visiveis, identificar os não ocupados
  let patchesVisiveisLivres patchesVisiveis with [ not any? other Individuos-here ]

  ; escolher aleatoriamente um dos patches visiveis livres e mover
  move-to one-of patchesVisiveisLivres
end
```

`move-to` <agente>:

- Move para a mesma posição do <agente>
  - O <agente> pode ser um *patch*, uma entidade, ou um agente

`one-of` <agentset>:

- Retorna um elemento aleatório do <agentset>

Sugascape NetLogo: a Execução da Simulação

É a suscessiva ativação dos comportamentos dos agentes

Por padrão, fica em uma *procedure* chamada go:

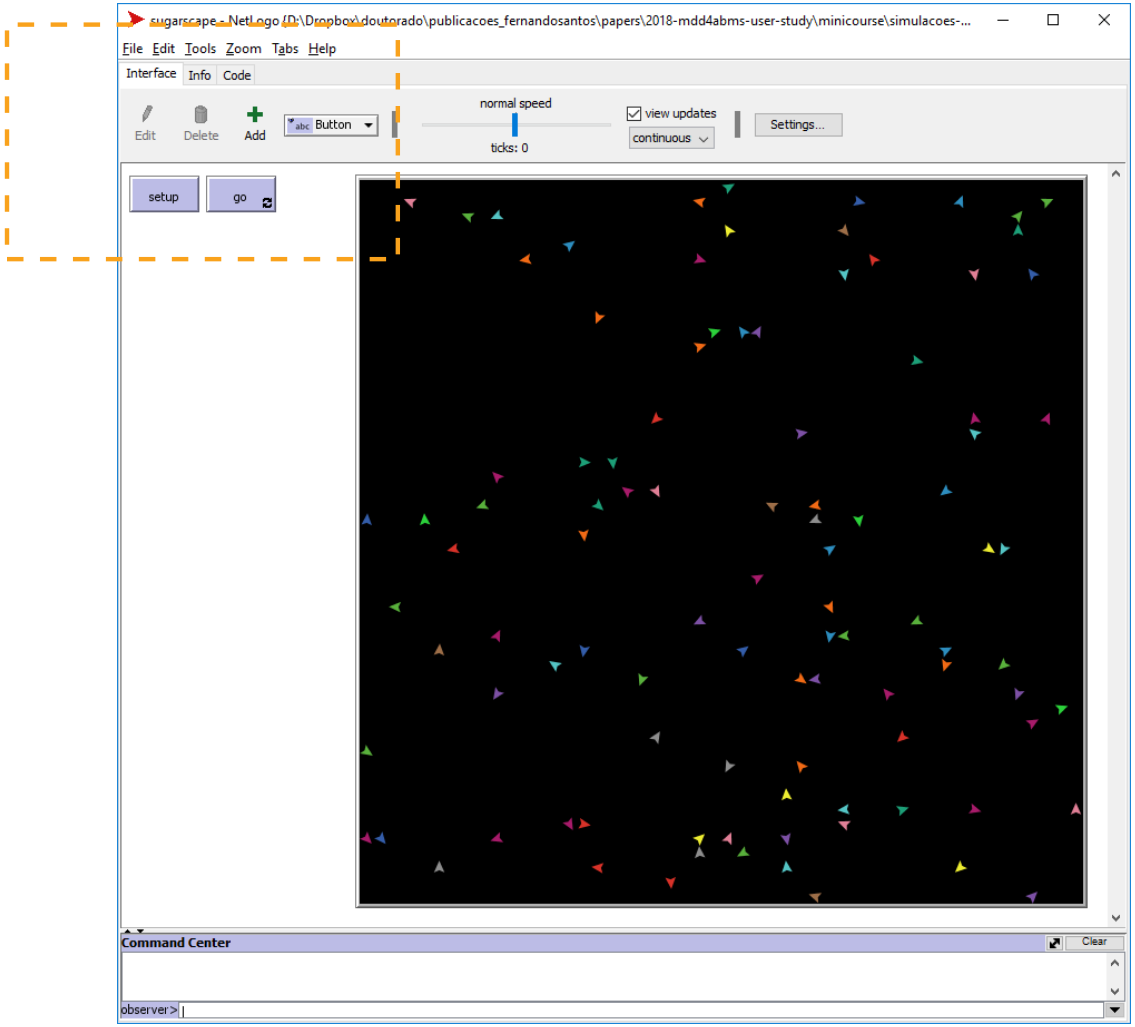
```
to go
  tick

  ; ativar comportamentos dos agentes
  ask Individuos [
    moverAgente
  ]
end
```

`tick`:

- Avança o contador de timesteps da simulação

Sugascape NetLogo: a Execução da Simulação

✍ Crie botões para ativar as *procedures* `setup` e `go`



Execute `setup` e `go`

- **Resultado esperado**: agentes são criados e se movem

Sugarscape NetLogo: o Açúcar nos Patches

Definir um atributo na entidade *Patch*:

```
patches-own [
  acucar
]
```

`patches-own [ ... ]`

- Define atributos para a entidade *Patch*

✍ Estenda o `setup` para inicializar o açúcar nos patches

- Inicialize com valor aleatório entre 0 e 4
- Dica: consulte comando `random` no NetLogo Dictionary
  - Está na categoria *Mathematical*

Sugarscape NetLogo: o Açúcar nos Patches

Definir um atributo na entidade *Patch*:

```
patches-own [
  acucar
]
```

`patches-own [ ... ]`

- Define atributos para a entidade *Patch*

✍ Estenda o `setup` para inicializar o açúcar nos patches

- Inicialize com valor aleatório entre 0 e 4
- Dica: consulte comando `random` no NetLogo Dictionary
  - Está na categoria *Mathematical*

```
to setup [
  clear-all
  reset-ticks

  ; código da criação dos agentes omitido aqui

  ask patches [
    set acucar random 5
  ]
]
```

NetLogo: Bibliotecas de Comportamentos

NetLogo suporta a "inclusão" de arquivos externos

- Similar a outras linguagens (ex.: `include`, `import`...)
- Útil para criar **bibliotecas de comportamentos**

```
__includes [
"arquivo1.nls"
"arquivo2.nls"
]

extensions [...]

breed [...]
```

`__includes ["arq1.nls" ...]`

- Inclui/importa arquivos `.nls`

✍ Incluir a seguinte biblioteca:

`color-cells-by-sugar-netlogo.nls`

NetLogo: Bibliotecas de Comportamentos

Padronização das bibliotecas adotadas neste curso

- O arquivo `.nls` pode conter várias *procedures*
- A primeira *procedure* é a que **desempenha** o comportamento
  - Deve ser ativada no `go`
- A *procedure* terminada com `Init` inicializa o comportamento
  - Se existir, deve ser ativada no `setup`

```
; arquivo 'exemplo.nls'

; aqui podem ficar definições de atributos ou var. globais

to comportamento ; primeira procedure do arquivo
   ; desempenha o comportamento
   ; esta procedure deve ser ativada no 'go'
end

to comportamentoInit [ parametro1  parametroN ] ; nem sempre há parâmetros
   ; inicialização do comportamento
   ; esta procedure deve ser ativada no 'setup'
end

; outras procedures
```

Sugarscape NetLogo: Visualização do Açúcar

✍ Inicializar e ativar a bilioteca:

```
color-cells-by-sugar-netlogo.nls
```

Sugarscape NetLogo: Visualização do Açúcar

✍ Inicializar e ativar a bilioteca:

color-cells-by-sugar-netlogo.nls

```
to setup
  ; outros comandos omitidos

  ; último comando: inicializar biblioteca 'color-cells-by-sugar-netlogo.nls'
  ask patches [
    colorCellsBySugarInit
  ]
end

to go
  tick
  ; ativar biblioteca 'color-cells-by-sugar-netlogo.nls'
  ask patches [
    colorCellsBySugar
  ]

  ; comportamentos dos agentes omitidos
end
```

Sugarscape NetLogo: Coletar Açúcar

✍ Implemente um comportamento que faz o agente coletar **todo** o açúcar do seu *patch* e transformá-lo em energia.

```
to coletarAcucar



end
```

`[<atributo>] of <agente>`

- Lê o valor do atributo do agente
- Analogia OO: *objeto.atributo*

Dica: A única forma de modificar o valor do atributo de **outro agente ou patch** é usando `ask`.

Sugarscape NetLogo: Coletar Açúcar

✍ Implemente um comportamento que faz o agente coletar **todo** o açúcar do seu *patch* e transformá-lo em energia.

```
to coletarAcucar

  ; incrementar a energia do agente
  set energia energia + [acucar] of patch-here
  ; zerar o acucar no patch
  ask patch-here [
    set acucar 0
  ]

end
```

`[<atributo>] of <agente>`

- Lê o valor do atributo do agente
- Analogia OO: *objeto.atributo*

Dica: A única forma de modificar o valor do atributo de **outro agente ou patch** é usando `ask`.

Sugarscape NetLogo: Metabolizar Açúcar

✍ Implemente um comportamento para **metabolizar** a energia

- Diminuir a energia de acordo com o valor do `metabolismo`

```
to metabolizarEnergia



end
```

Sugarscape NetLogo: Metabolizar Açúcar

✍ Implemente um comportamento para **metabolizar** a energia

- Diminuir a energia de acordo com o valor do `metabolismo`

```
to metabolizarEnergia

  ; decrementar a energia do agente
  set energia energia - metabolismo

end
```

Sugarscape NetLogo: Morte por Inanição

✍ Implemente um comportamento que faz o agente **morrer** por inanição (quando ficar sem energia)

```
to morrerPorInanicao



end
```

```
if <condição> [ <comandos> ]
if-else <condição> [ <comandos-if> ] [ <comandos-else> ]
```

- Condicionais: funcionam igual a outras linguagens

```
die
```

- Faz o agente morrer

Sugarscape NetLogo: Morte por Inanição

✍ Implemente um comportamento que faz o agente **morrer** por inanição (quando ficar sem energia)

```
to morrerPorInanicao

  ; morre se energia chegou a zero
  if energia <= 0 [
    die
  ]

end
```

```
if <condição> [ <comandos> ]
if-else <condição> [ <comandos-if> ] [ <comandos-else> ]
```

- Condicionais: funcionam igual a outras linguagens

```
die
```

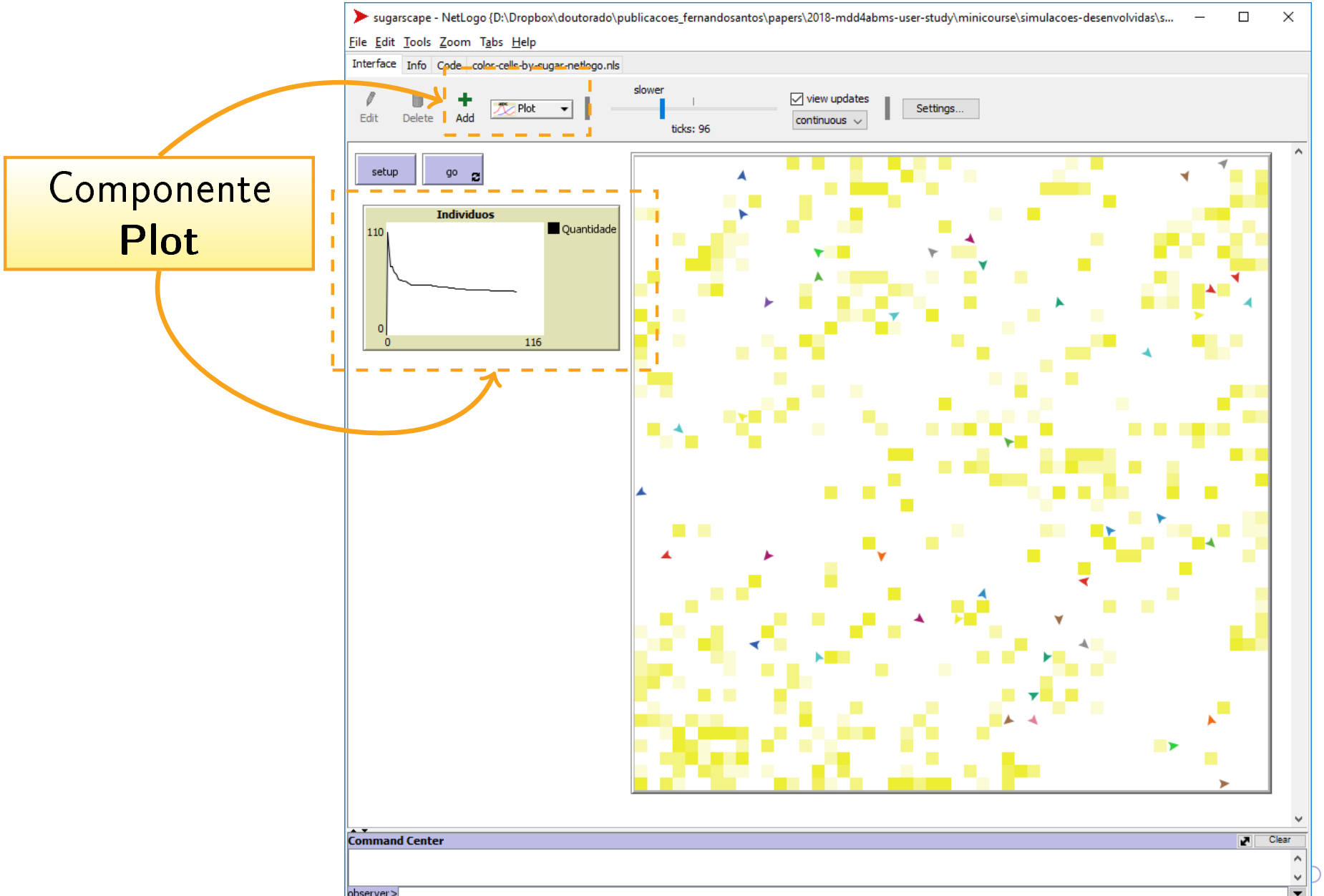
- Faz o agente morrer

Sugarscape NetLogo: o Procedimento 'go' Completo

```
to go
  tick

  ask patches [
    colorCellsBySugar
  ]

  ask Individuos [
    moverAgente
    coletarAcucar
    metabolizarEnergia
    morrerPorInanicao
  ]

end
```

Sugarscape NetLogo: as Saídas

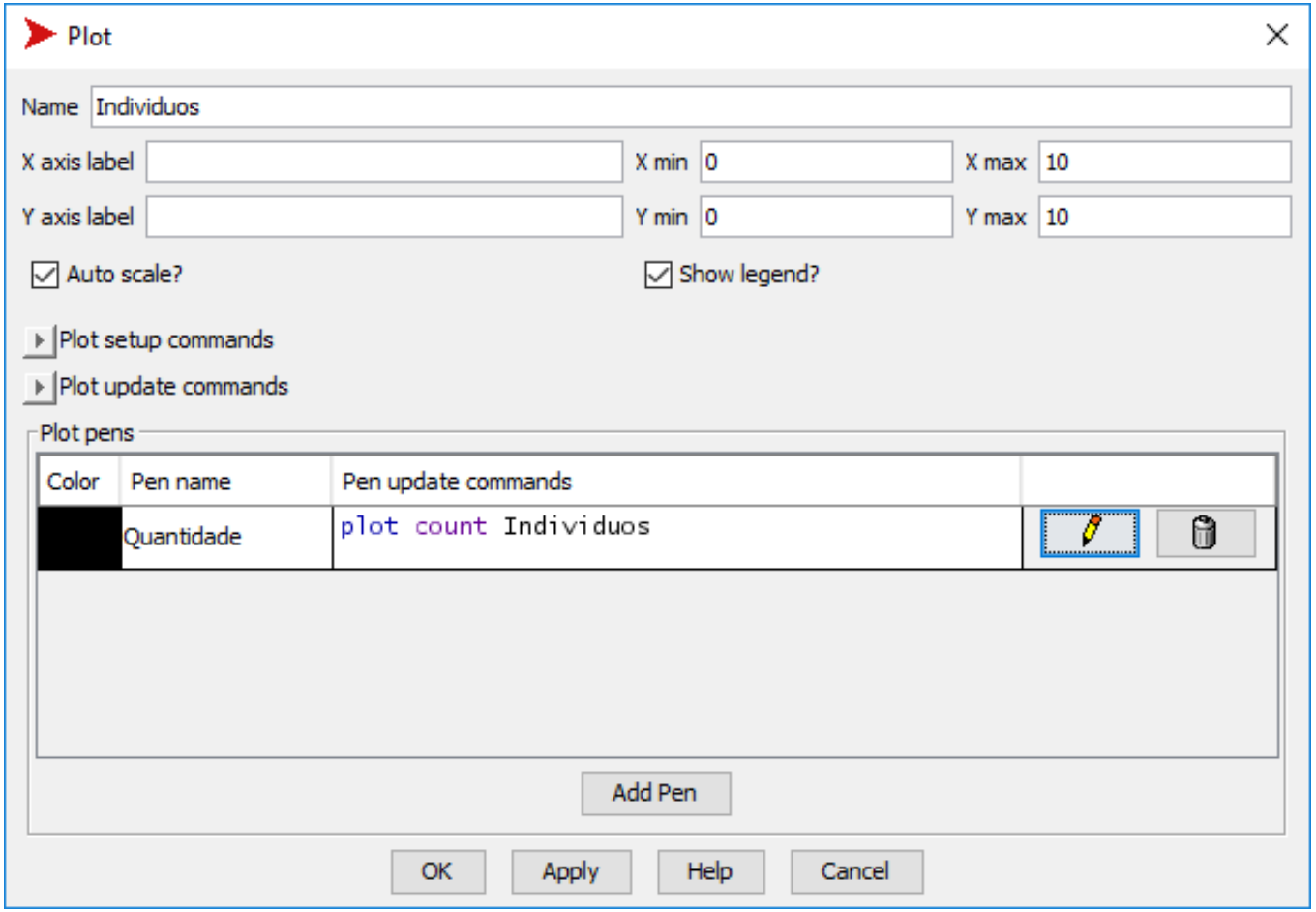
NetLogo oferece gráficos para observar o resultado da simulação



Sugarscape NetLogo: Plots

Configurações básicas (eixos, intervalos, etc)

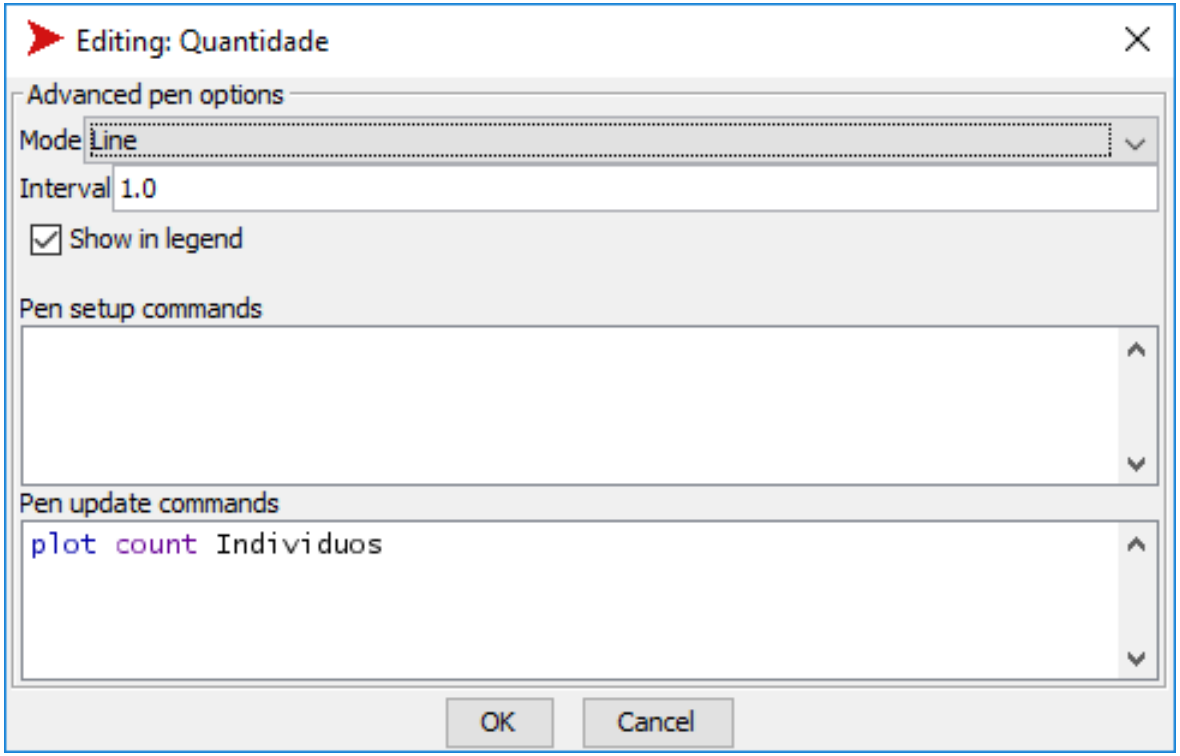
Um **Plot** é composto por **Pens**



Sugarscape NetLogo: Pens

Uma **pen** é um dado (ou linha) que será plotado

- Mode: tipo (linha, barra)
- Update commands: comandos NetLogo que produzem o dado



Pen **Quantidade**:

```
plot count Individuos
```

`plot`

- Plota um dado no gráfico

`count <agentset>`

- Quantidade de elementos no `agentset`

Sugarscape NetLogo: Estenda a implementação...

✍ Ao criar Individuo, inicializar atributos com valores aleatórios:

Atributo	**Min**	**Max**
raioVisao	1	6
metabolismo	1	4

- Ver documentação do comando `random` no NetLogo Dictionary

✍ Estenda o comportamento `moverAgente`:

- Mover o agente para o *patch* com a **maior quantidade de açúcar** (dentre os visíveis e livres)
- **Resultado esperado**
  - Agentes sobrevivem mais tempo
  - Células com mais açúcar são consumidas primeiro

Sugarscape NetLogo: Estenda a implementação...

Soluções

✍ Ao criar Individuo, inicializar atributos com valores aleatórios:

Atributo	Min	Max
raioVisao	1	6
metabolismo	1	4

```
to setup
  ; comandos omitidos
  ask n-of 100 patches [
    sprout-Individuos 1 [
      set raioVisao 1 + random 6
      set metabolismo 1 + random 4
      set energia 0
    ]
  ]
  ; comandos omitidos
```

Sugarscape NetLogo: Estenda a implementação...

Soluções

✍ Estenda o comportamento `moverAgente`:

- Mover o agente para o *patch* com a **maior quantidade de açúcar** (dentre os visíveis e livres)

```
to moverAgente
  ; identifica os patches dentro do raio de visao e armazena em variável
  let patchesVisiveis patches in-radius raioVisao

  ; identifica os patches visiveis e não ocupados por outro agente
  let patchesVisiveisLivres patchesVisiveis with [ not any? other Individuos-here ]

  ; escolhe o patch com maior quantidade de açúcar
  move-to max-one-of patchesVisiveisLivres [ acucar ]
end
```

`max-one-of <agentset> <critério>`

- Retorna o elemento do <agentset> que maximiza o <critério>
- O <critério> pode ser um atributo, expressão, ou *procedure* com retorno

Sugarscape NetLogo: o que Desenvolvemos até agora?[2]

Ambiente

- Com açúcar nos *patches*
- Visualização do açúcar (coloração)

Agentes: comportamentos

- Se move pelo ambiente
  - Para um *patch* dentro de um **raio de visão** que esteja **livre** e que tenha a **maior quantidade de açúcar**
- Coleta o açúcar disponível no *patch* e transforma em energia
- Permanece vivo enquanto **nível de energia** for maior que zero
- Possui um **metabolismo**
  - Faz a energia diminuir a cada passo da simulação.

[2] Se a sua implementação está incompleta, baixe **(#TODO)**aqui a implementação feita até agora.

Sugarscape NetLogo: Próximos Passos

Ambiente

- Inicializar o açúcar a partir de arquivo (no *setup*)
- Fazer crescer (reaparecer) açúcar nos *patches*

Agentes

- Comportamento de reprodução (simples)
  - Recriar o fenômeno de evolução / seleção natural

Saídas

- Histograma do metabolismo dos agentes
- Metabolismo e visão médio dos agentes

Sugarscape NetLogo: o Setup com Arquivo

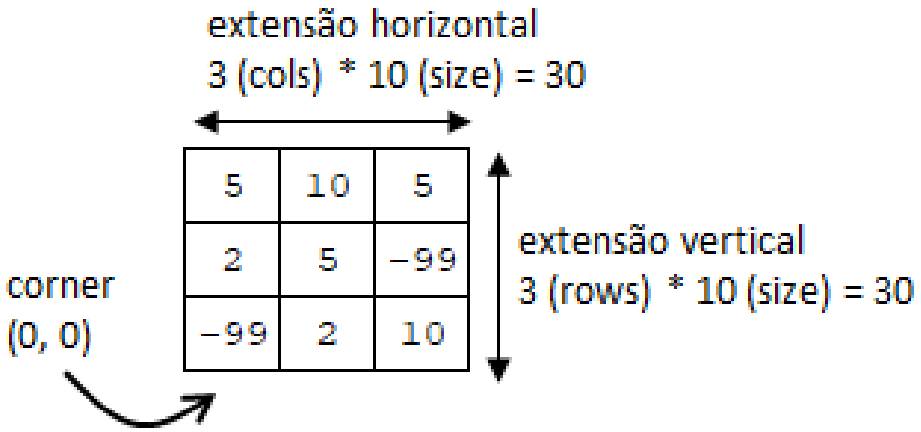
Uso de arquivo GIS[3] para inicializar atributos dos *patches*

GIS ASCII FILE

CONTEÚDO

```
ncols          3
nrows          3
xllcorner      0.0
yllcorner      0.0
cellsize       10.0
NODATA_value   -99
5     10   5
2     5    -99
-99   2    10
```

VISUALIZAÇÃO



Formato de Arquivo GIS Raster

- Arquivo: `sugar-map.asc`
  - Valores são a **quantidade inicial** de açúcar por *patch*
  - Variam de 0 a 4

[3] Geographic Information System

Sugarscape NetLogo: o Setup com Arquivo

Inicializar açúcar usando dados GIS

Modificar o setup para obter dados do GIS

```
to setup
  clear-all
  reset-ticks
   ; aqui fica a criação dos agentes
   ; lendo o açúcar do arquivo e inicializando atribudo dos patches
  let dadosAcucar gis:load-dataset "sugar-map.asc"
  gis:set-world-envelope-ds gis:envelope-of dadosAcucar
  gis:apply-raster dadosAcucar acucar
  ; aqui fica a inicialização da biblioteca de coloração dos patches
end
```

`let` <varArquivo> `gis:load-dataset` `<arquivo>`:

- Abre o <arquivo> GIS na variável local <varArquivo>

`gis:set-world-envelope-ds`
`gis:envelope-of <variavelArquivo>`

- Compatibilizam dimensões do arquivo e do *world* (coordenadas)

`gis:apply-raster <varArquivo> <atributoPatch>`:

- Copia valores do arquivo para atributo dos *patches*, conforme coordenadas

Sugarscape NetLogo: o Setup com Arquivo

Inicializar açúcar usando dados GIS

Procedimento `setup` completo:

```
to setup
  clear-all
  reset-ticks
  ask n-of 100 patches [
    sprout-Individuos 1 [
      set raioVisao 1 + random 6
      set metabolismo 1 + random 4
      set energia 0
    ]
  ]
  ; lendo o açúcar do arquivo e inicializando atribudo dos patches
  let dadosAcucar gis:load-dataset "sugar-map.asc"
  gis:set-world-envelope-ds gis:envelope-of dadosAcucar
  gis:apply-raster dadosAcucar acucar

  ; inicializando a biblioteca de coloração dos patches
  ask patches [
    colorCellsBySugarInit
  ]
end
```

Sugarscape NetLogo: Reaparecimento de Açúcar

Especificação:

- A cada passo da simulação, **uma unidade** de açúcar é adicionada em cada *patch*
- A quantidade de açúcar no *patch* não pode ultrapassar a quantidade inicial (lida do arquivo GIS)
  - Ou seja, a quantidade inicial é também a quantidade **máxima**

✍ Implemente o reaparecimento de açúcar

- Dica: definir um atributo no *patch* para armazenar a quantidade máxima de açúcar durante o `setup`

Sugarscape NetLogo: Reaparecimento de Açúcar

Solução (parte 1)

Atributo no *patch*:

```
patches-own[
   acucar
   max_acucar
]
```

Procedimento `setup`

```
to setup
   ; aqui fica a criação dos agentes
   ; lendo o açúcar do arquivo e inicializando atribudo dos patches
   let dadosAcucar gis:load-dataset "sugar-map.asc"
   gis:set-world-envelope-ds gis:envelope-of dadosAcucar
   gis:apply-raster dadosAcucar acucar
   ; ***** inicializar também a quantidade máxima de açúcar *****
   gis:apply-raster dadosAcucar max_acucar

   ; aqui fica a inicialização da biblioteca de coloração dos patches
end
```

Sugarscape NetLogo: Reaparecimento de Açúcar

Solução (parte 2)

Procedimento go

```
to go
  tick

  ask patches [
    if acucar < max_acucar [
      set acucar acucar + 1
    ]
    colorCellsBySugar
  ]

  ; aqui fica o comportamento dos agentes
end
```

Sugarscape NetLogo: Explorando a Simulação (1)

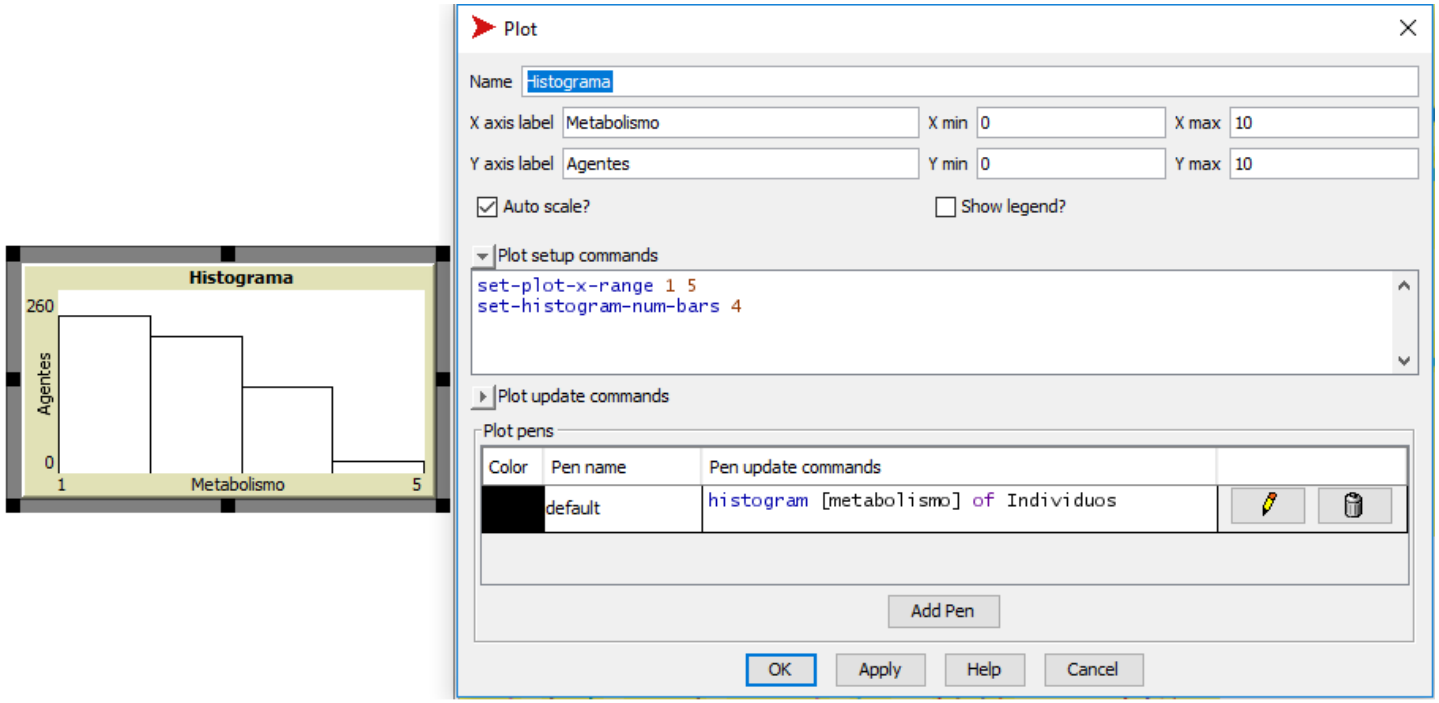
✍ Execute a simulação, observe as saídas, e responda:

- Por que há agentes que morrem no início da simulação?

- O que acontece ao modificar a energia inicial dos agentes?
  - Por exemplo, inicializar com valor aleatório entre 1 e 5

Sugarscape NetLogo: Explorando a Simulação (2)

- O que acontece se aumentar a quantidade inicial de agentes?
  - 250, 500, 1000, 2000, 2500
- Qual o metabolismo dos agentes sobreviventes?
  - Criar um *plot* de histograma do metabolismo



`set-histogram-num-bars` <quantidade>

- Quantidade de barras do histogram. Atenção: *pen* deve ser *mode* **Bar**

`set-plot-x-range` <lower> <upper>

- Define o *range* do histograma. Atenção: deve acomodar qtd. barras

Sugarscape NetLogo: Reprodução

Características da Reprodução:

- Cada agente pode gerar um descendente
  - Simulação de reprodução simples, sem acasalamento
- É necessário ter um *patch* livre na vizinhança imediata
  - Ou seja, algum *patch* no raio 1.5 deve estar livre
- O descendente herda:
  - a visão e metabolismo do genitor
  - metade da energia do genitor

Biblioteca de Reprodução: `reproduction-netlogo.nls`

✍ Importe a biblioteca e ative a reprodução.

- A *procedure* `reproduceAgentInit` possui parâmetro. Verifique

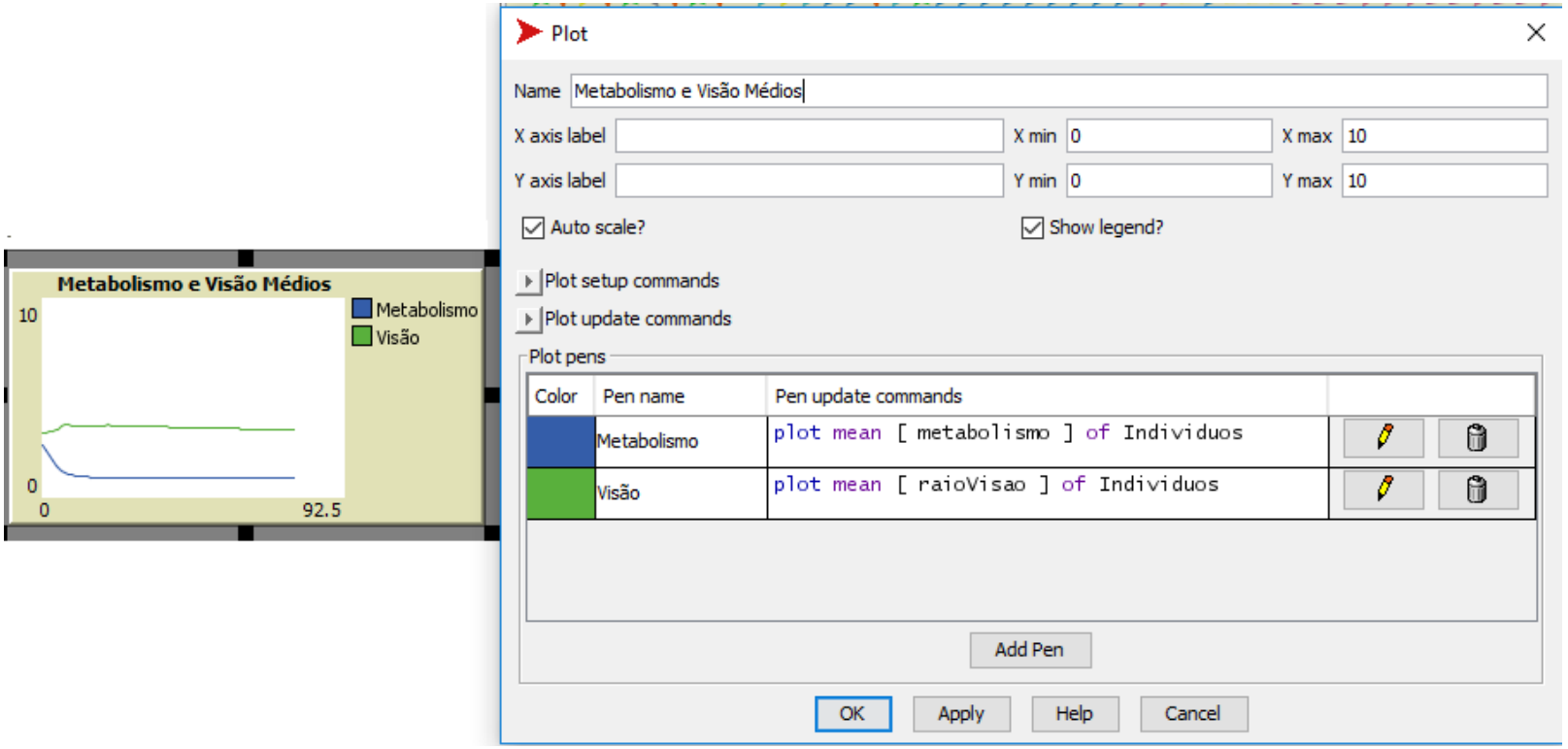
Sugarscape NetLogo: Explorando a Simulação

✍ Execute a simulação, observe as saídas, e responda:

- Quantos agentes permanecem vivos após 100 timesteps?
  - Há diferença se usar diferentes quantidades iniciais de agentes?
  - Fenômeno: capacidade de carga do ambiente
    - (*carrying capacity*)

Sugarscape NetLogo: Evolução

✍ Criar um *plot* com as médias do metabolismo e do raio de visão



`mean [ <atributo> ] of <agentset>`

- Calcula a média dos valores do <atributo> de todos os elementos no <agentset>

Sugarscape NetLogo: Explorando a Simulação

✍ Execute a simulação, observe as saídas, e responda:

- Visão/metabolismo médios aumentam/diminuem?
  - Fenômeno: Evolução da Espécie por Seleção Natural?

- Qual o efeito de alterar o raio de visão e metabolismo iniciais?

Atributo	**Min**	**Max**
raioVisao	1	10
metabolismo	1	10

  - Qual a possível explicação para a "pouca" evolução da visão?

- Qual o efeito de alterar o reaparecimento do açúcar?
  - A cada passo, apenas **0.2** de açúcar é adicionada no *patch*
  - Fenômenos:
    - Evolução da visão
    - Extinção da espécie?

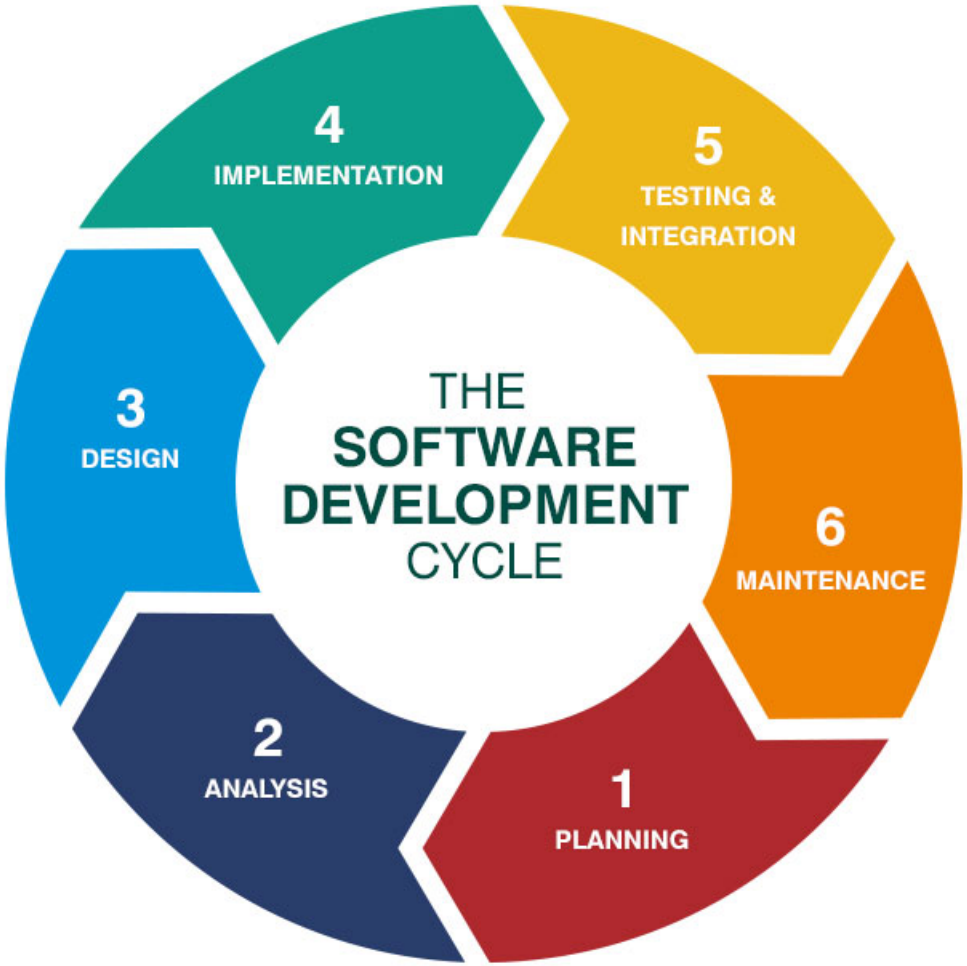
Simulações com Agentes: Conclusões Preliminares

- Sucesso em reproduzir fenômenos
  - Evolução / Seleção natural
  - Capacidade de carga
  - Extinção da espécie
- Ferramenta NetLogo traz eficiência no desenvolvimento da simulação?
  - Comparado com linguagens de propósito geral (ex.: Java)
- E se pudesse construir simulações sem implementar código?

2.3. Desenvolvimento Dirigido a Modelos de Simulações com Agentes

Sobre o Desenvolvimento de Sistemas

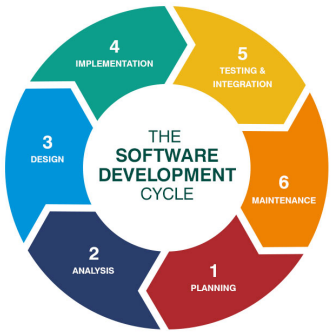
Um exemplo de ciclo de desenvolvimento de sistemas:



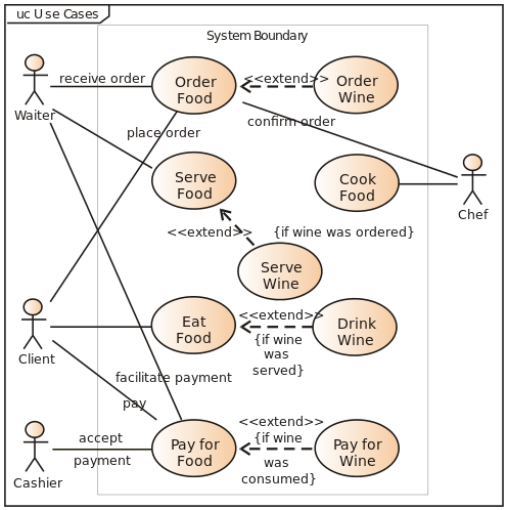
Artefatos produzidos e por quem nos estágios...

- Análise?
- Design?
- Implementação?

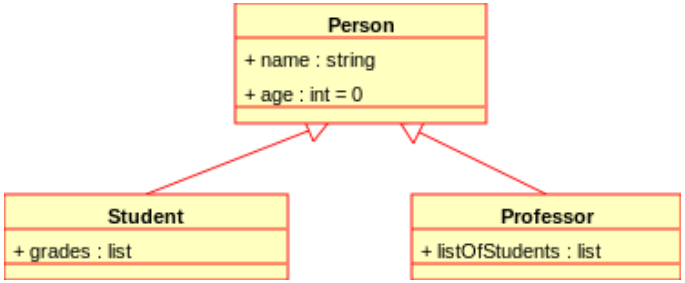
Sobre o Desenvolvimento de Sistemas



Analista →



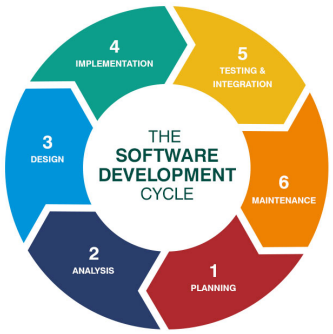
Projetista



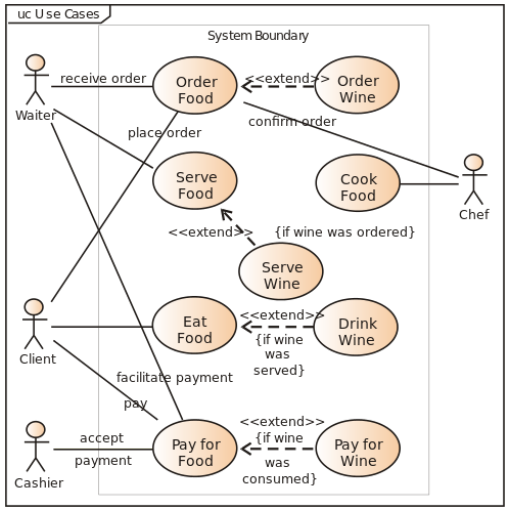
Desenvolvedor

```java
// Java bean for Person
public class Person {
    // Private variable
    private String fullName;
    // Constructor
    public Person(String fullName) {
        setFullName(fullName);
    }
    // Getter and setter for variable
    public String getFullName() {
        return fullName;
    }
    public void setFullName (String fullName) {
        this.fullName=fullName;
    }
}
```

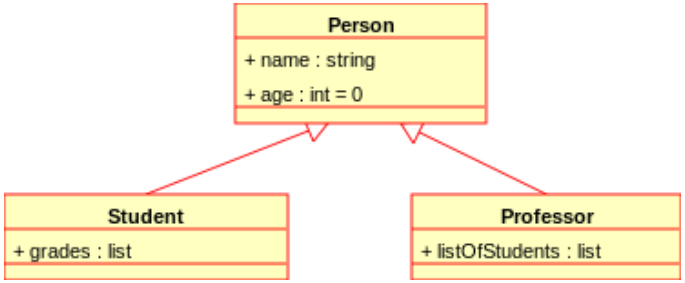
Sobre o Desenvolvimento de Sistemas



Analista →



Projetista →



Desenvolvedor →

```java
// Java bean for Person
public class Person {
    // Private variable
    private String fullName;
    // Constructor
    public Person(String fullName) {
        setFullName(fullName);
    }
    // Getter and setter for variable
    public String getFullName() {
        return fullName;
    }
    public void setFullName (String fullName) {
        this.fullName=fullName;
    }
}
```

Desenvolvimento Dirigido a Modelos

Desenvolvimento Dirigido a Modelos: Introdução

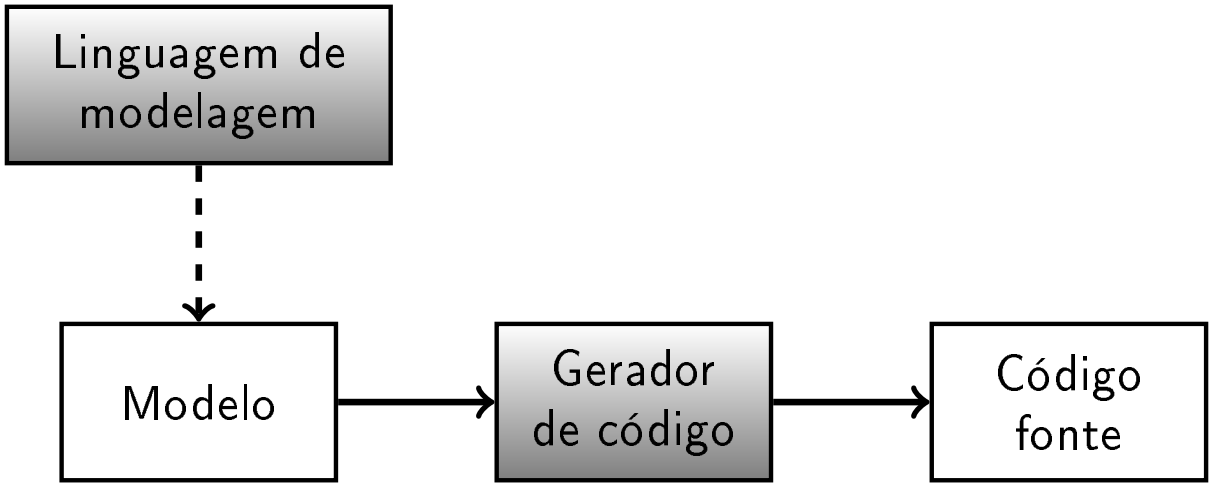
Model-driven Development (MDD)

Abordagem para desenvolvimento de sistemas. Filosofia:

- Geração de código a partir de modelos do sistema
  - Artefatos de *baixo nível* gerados por artefatos de *alto nível*

Elementos de uma abordagem MDD (preliminares):

- Linguagem de modelagem
- Geração (automática) de código



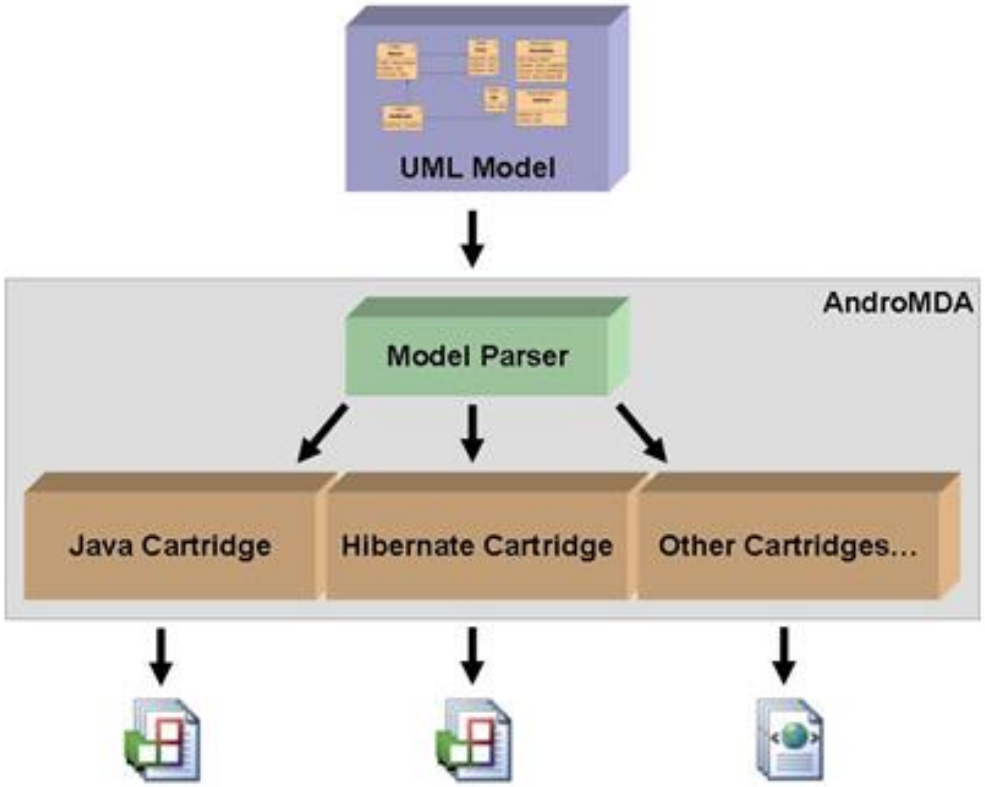
Benefícios (preliminares):

- Produtividade

Desenvolvimento Dirigido a Modelos: Exemplo

AndroMDA: http://andromda.sourceforge.net

- Geração de código a partir de modelos UML



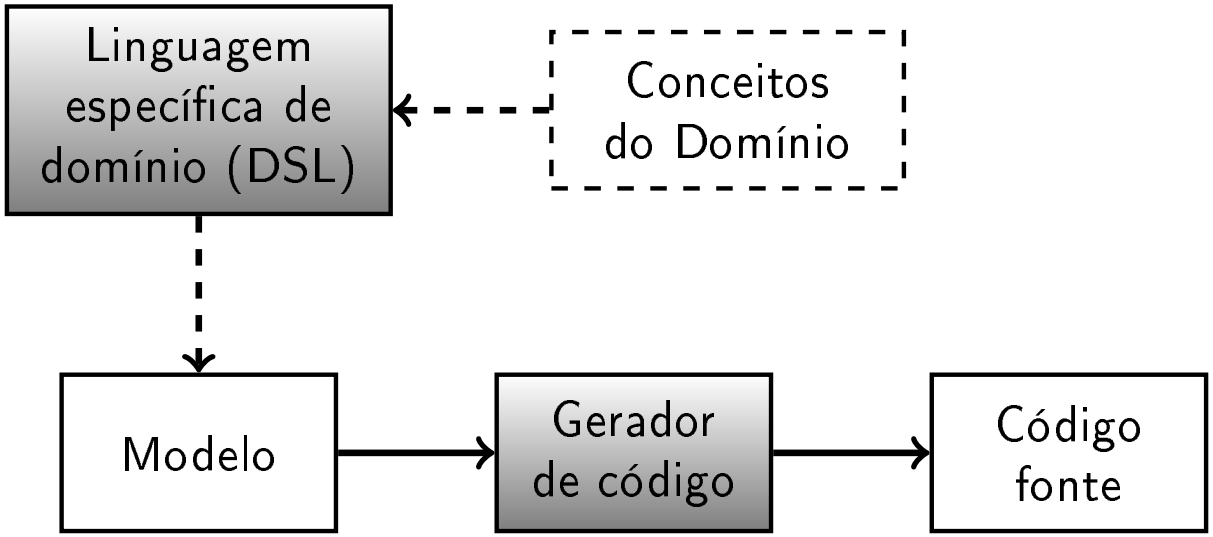
Quanto código é possível gerar a partir da UML?

- Código executável ou "templates"?

Desenvolvimento Dirigido a Modelos: + Produtividade?

Melhorar produtividade com linguagem de modelagem **expressiva**

- Fornece conceitos (abstrações) do domínio de negócio
  - Ex.: Contrato, Processo de Aprovação de Pedido, ...
- Linguagem de modelagem **específica de domínio**
  - Domain-specific Language (DSL)
- Geração de código para estes conceitos



Efeito colateral

- Aplicação restrita ao domínio de negócio
- Avaliar custo-benefício para desenvolver DSL

Desenvolvimento Dirigido a Modelos: Exemplos

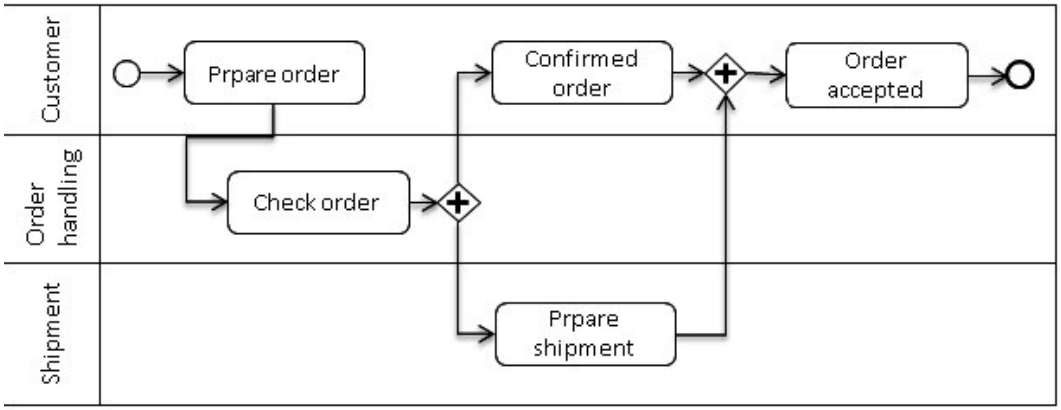
Domínio: bancos de dados relacionais

`select * from tb_pessoa`

Domínio: matemática / matrizes

`A * B`: multiplica matriz A por B

Domínio: processos

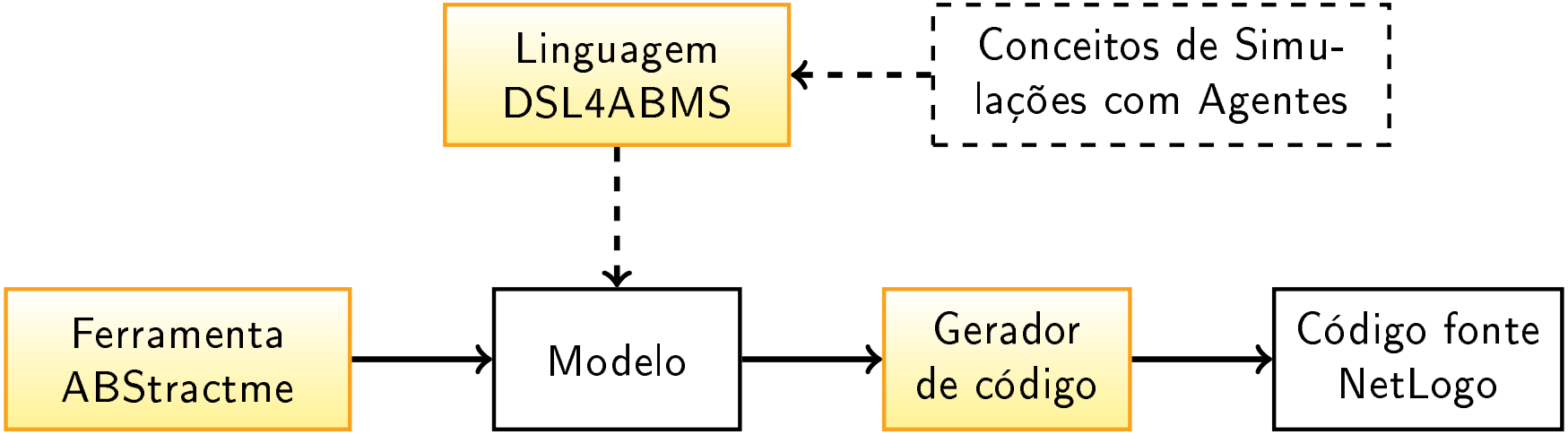


MDD de Simulações com Agentes: MDD4ABMS

Abordagem MDD para o domínio de **simulações com agentes**

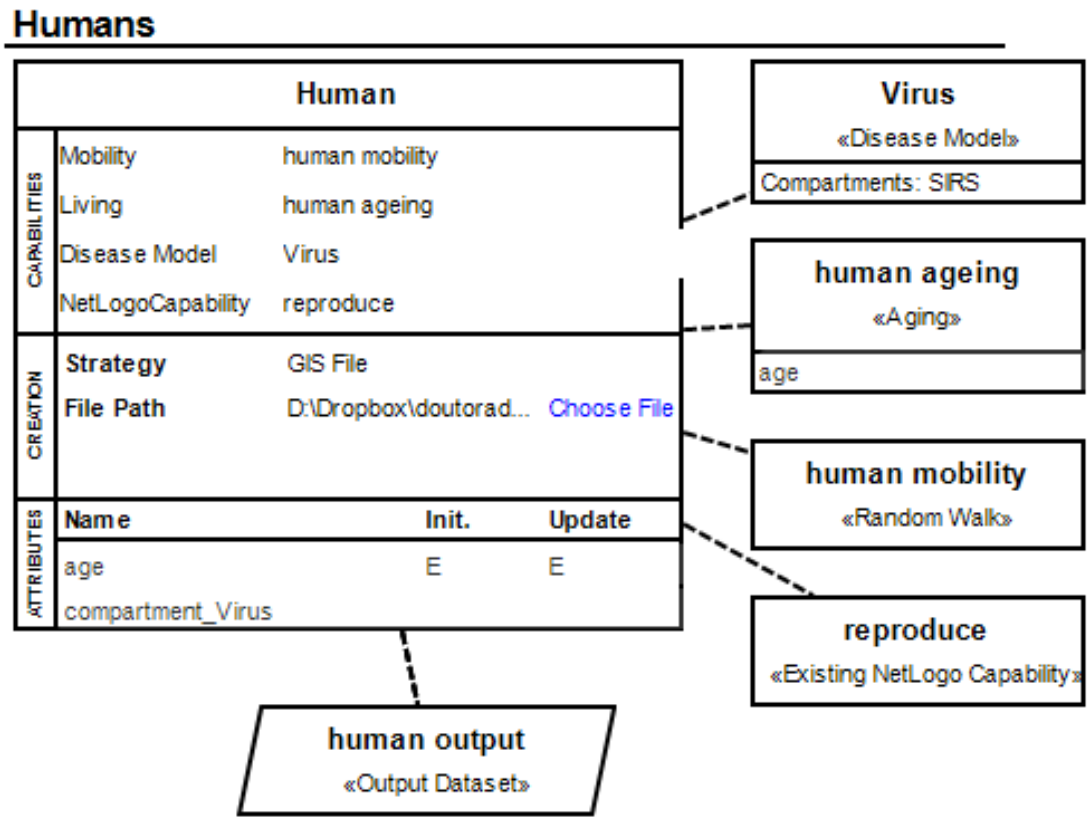
Elementos:

- DSL para modelagem: **DSL4ABMS**
- Ferramenta para modelagem: **ABStractme**
- Geração automática de código fonte NetLogo



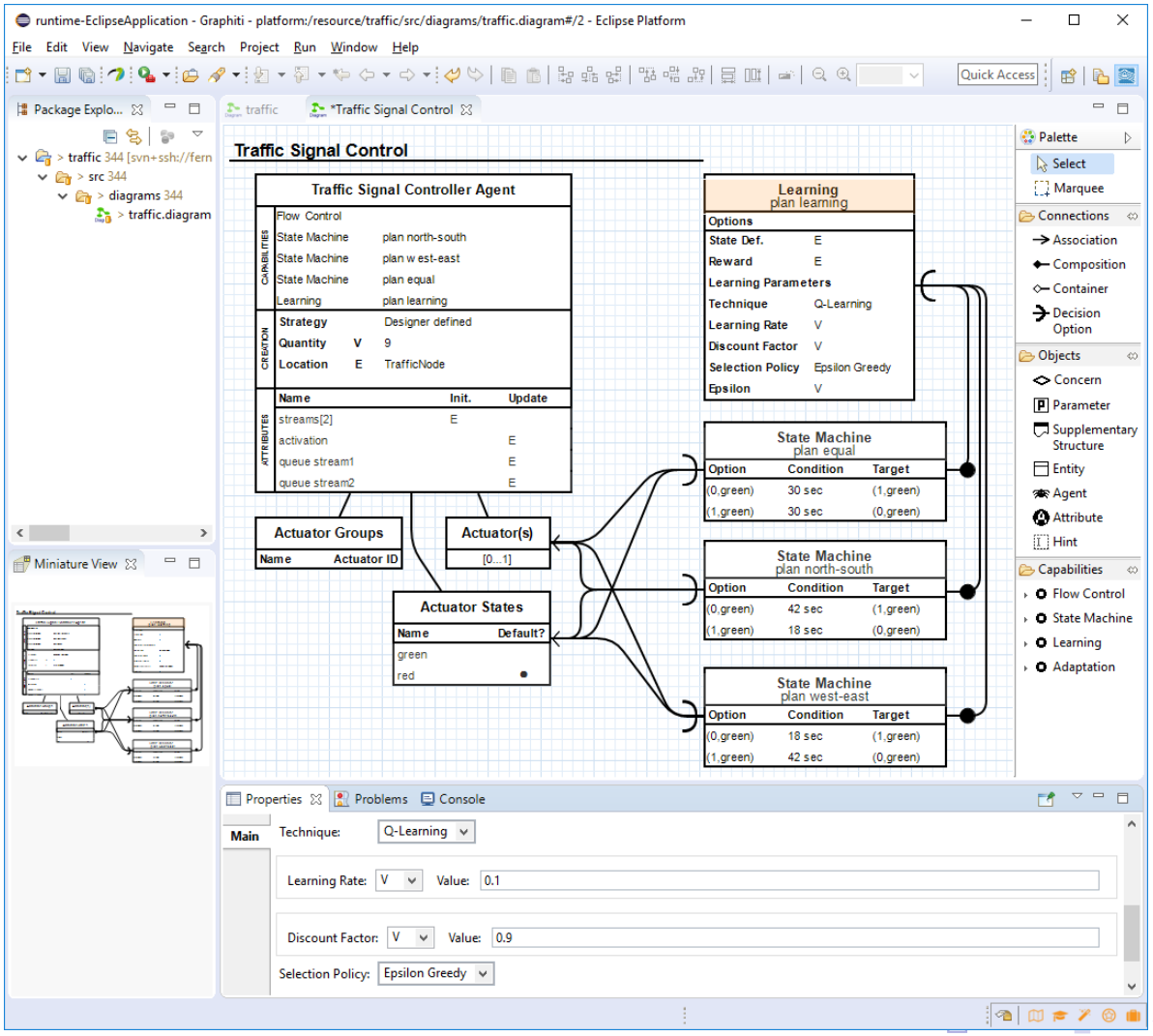
MDD de Simulações com Agentes: Linguagem DSL4ABMS

- Linguagem gráfica para modelar simulações com agentes
- Oferece abstrações para elementos recorrentes:
  - Ambiente, entidades e agentes
  - Habilidades dos agentes
    - doenças, movimentação, aprendizado, ...



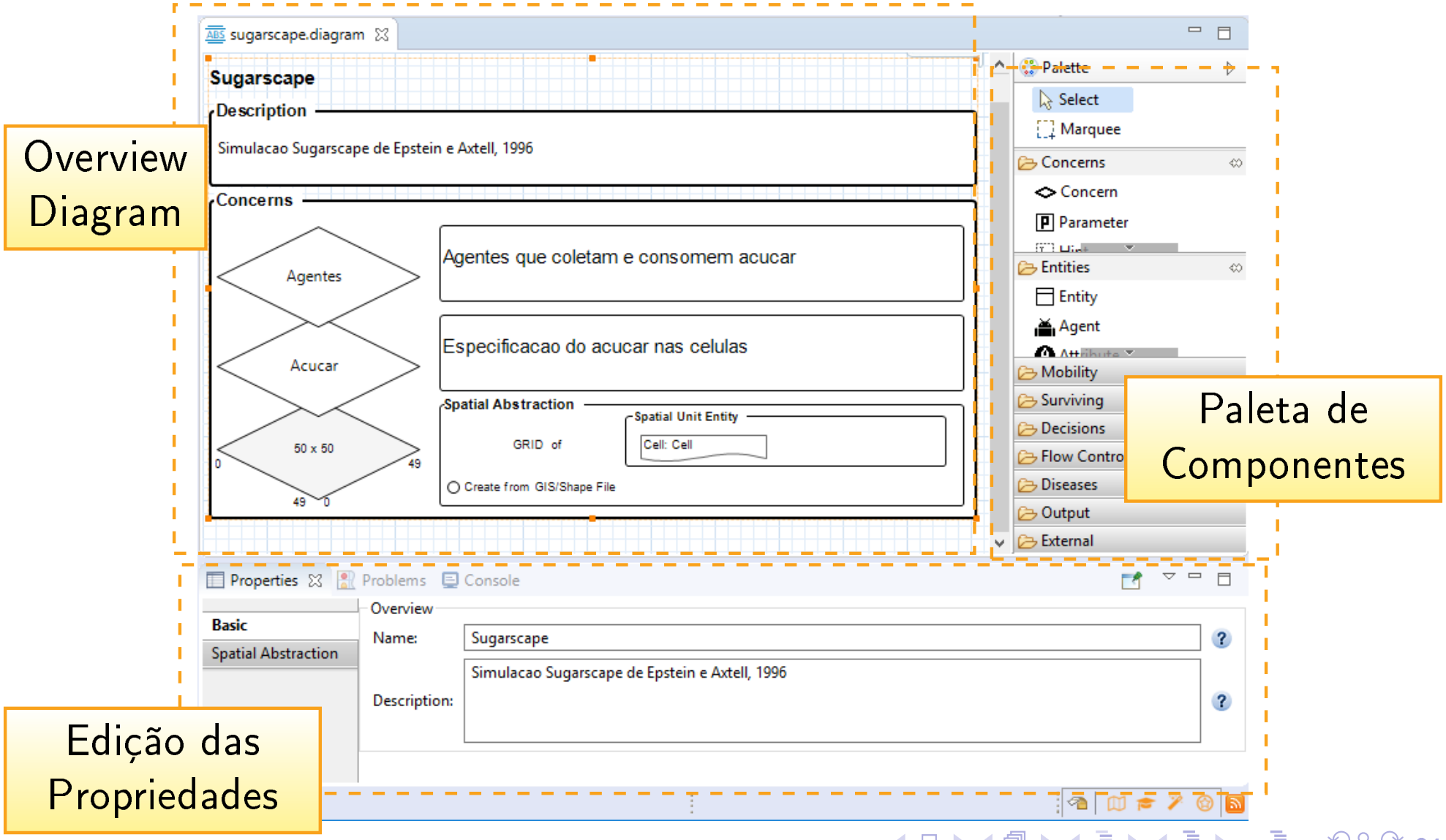
MDD de Simulações com Agentes: Ferramenta ABStractme

- Ferramenta para modelagem usando a linguagem DSL4ABMS
  - Plug-in para o ambiente de desenvolvimento Eclipse
- Oferece gerador de código NetLogo



MDD4ABMS: Diagrama de Visão Geral

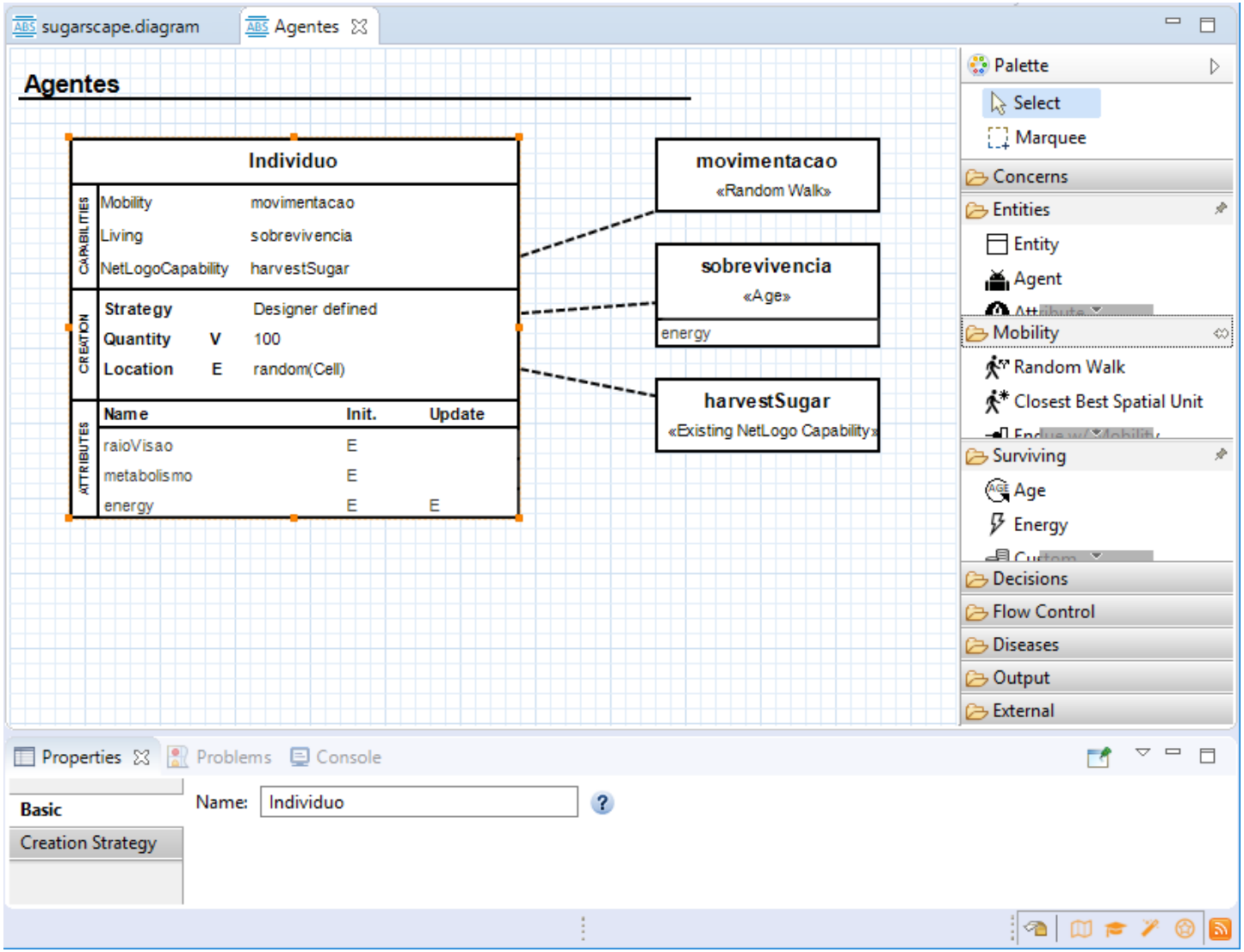
- Especificação dos elementos básicos da simulação
- **Concern**: agrupa elementos relacionados



MDD4ABMS: Diagramas de Concern

Concern: grupo de elementos relacionados

- Entidades, agentes, habilidades dos agentes, saídas, ...



2.4. Desenvolvendo Simulações com MDD4ABMS

Sugarscape MDD4ABMS

Modelar a mesma simulação Sugarscape

Ambiente

- Com açúcar nos *patches*
- Visualização (coloração) e reaparecimento do açúcar

Agentes: comportamentos

- Se move pelo ambiente
  - Para um *patch* dentro de um **raio de visão** que esteja **livre** e que tenha a **maior quantidade de açúcar**
- Coleta o açúcar disponível no *patch* e transforma em energia
- Permanece vivo enquanto **nível de energia** for maior que zero
- Possui um **metabolismo**
  - Faz a energia diminuir a cada passo da simulação.
- Se reproduz

MDD4ABMS: Criação de projetos e diagramas

Projeto:

New Project → **ABStractme Project**

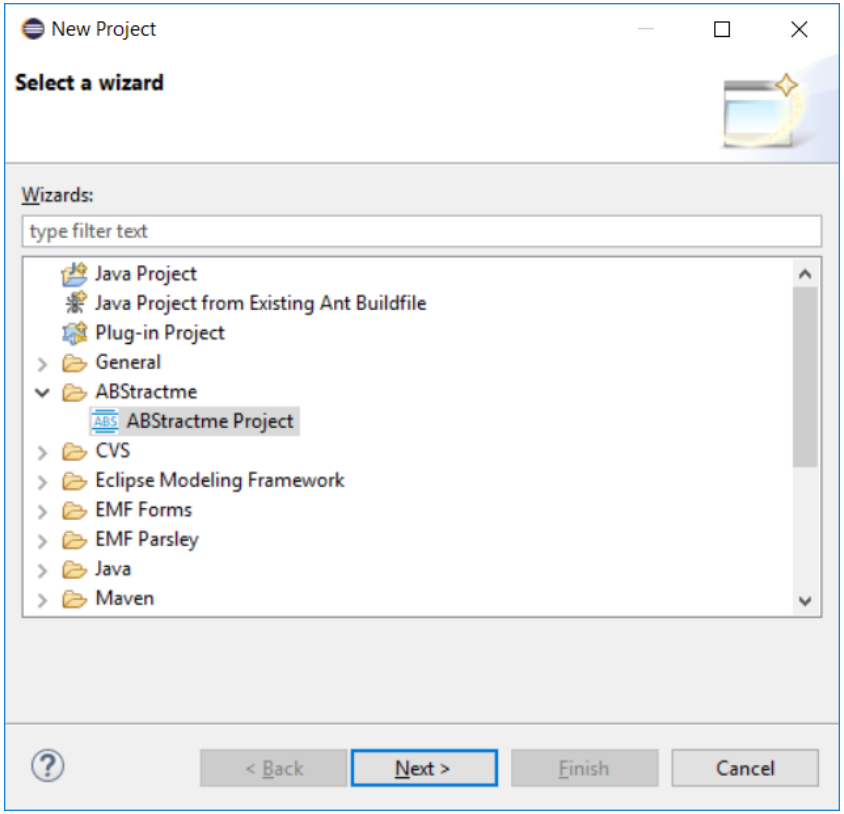
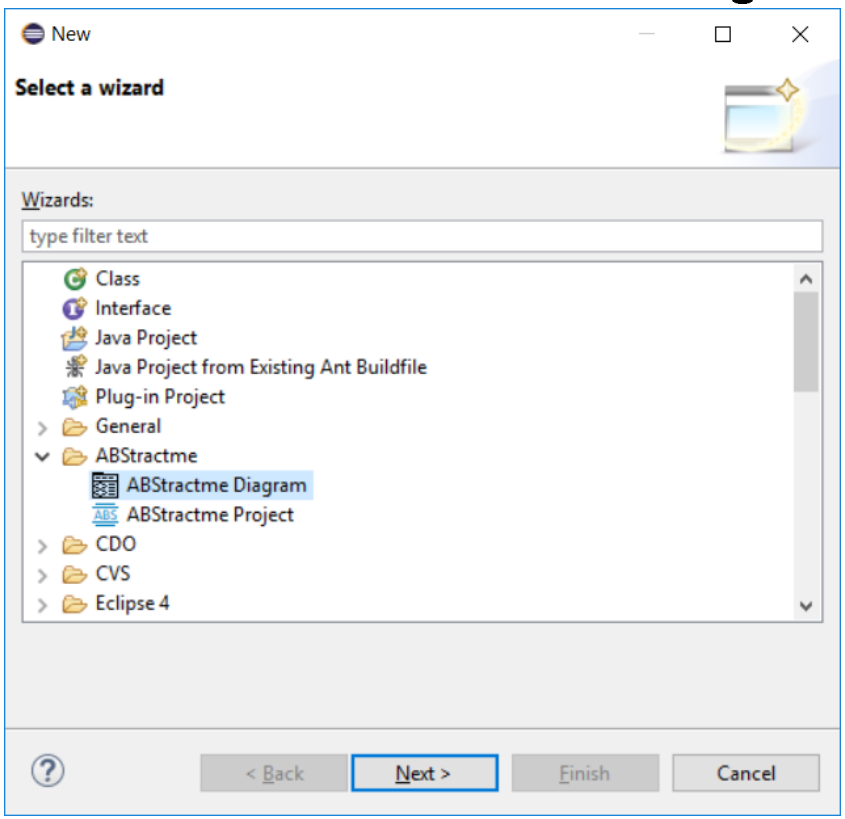


Diagrama (modelo):

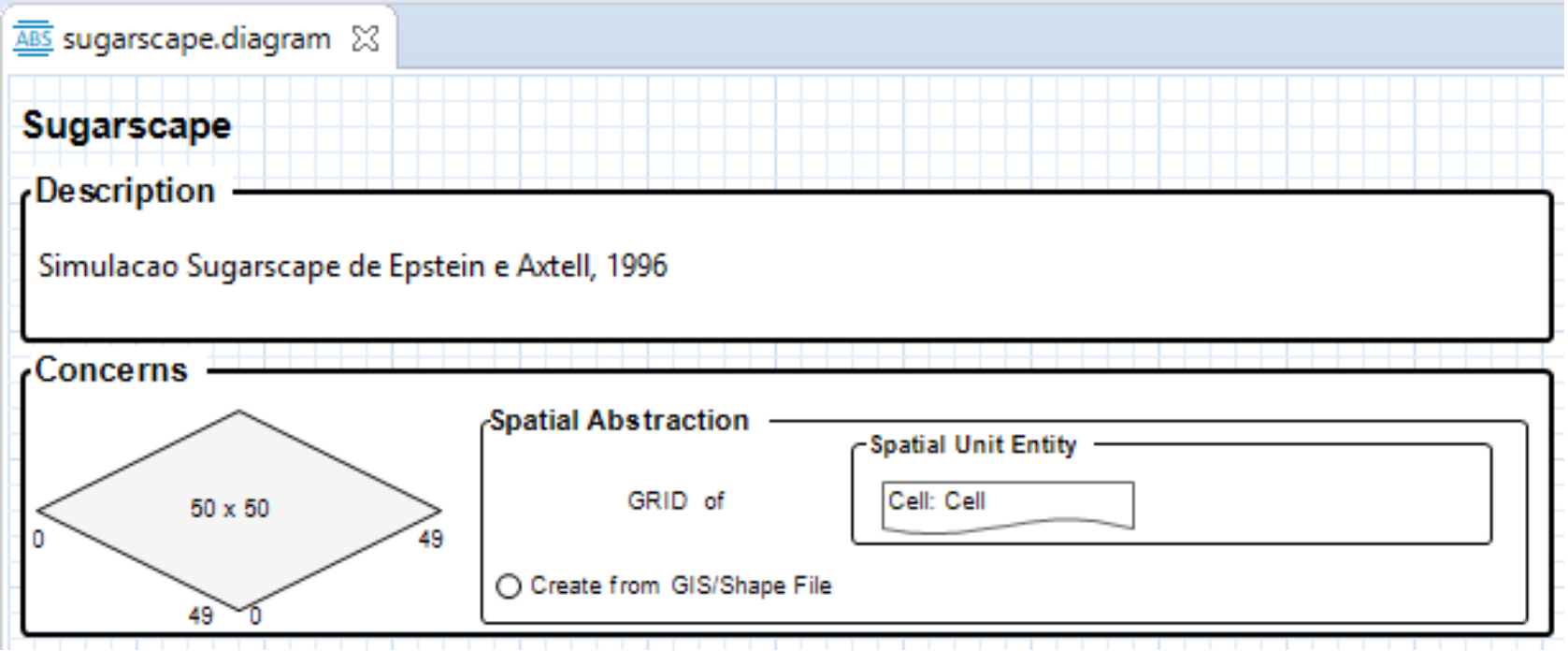
New → Other → **ABStracme Diagram**



Sugarscape MDD4ABMS: Overview (1)

Título e descrição do modelo

- Não usar acentuação/caracteres especiais



- Primeiro *concern* é o **ambiente**: *spatial abstraction*

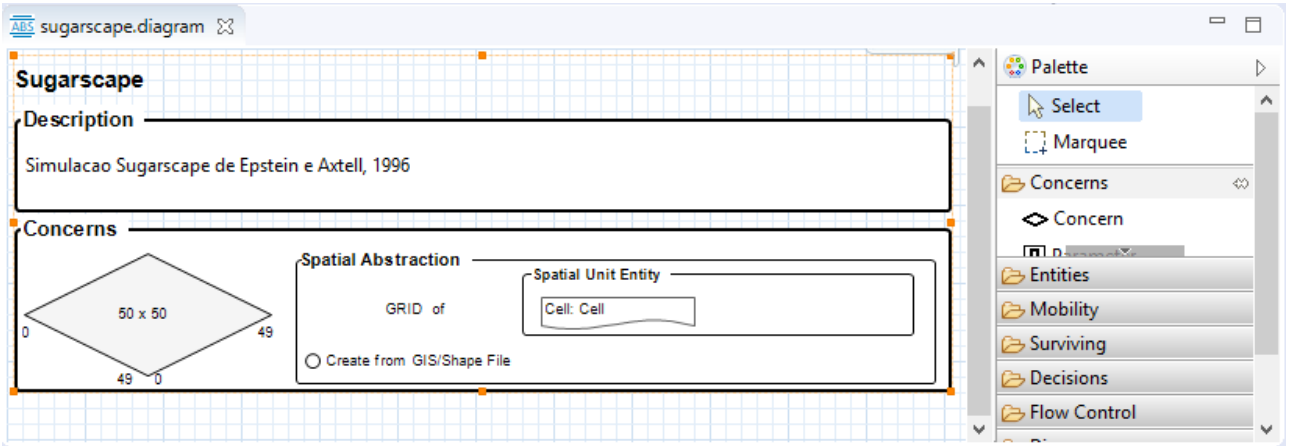
Sugarscape MDD4ABMS: o Ambiente

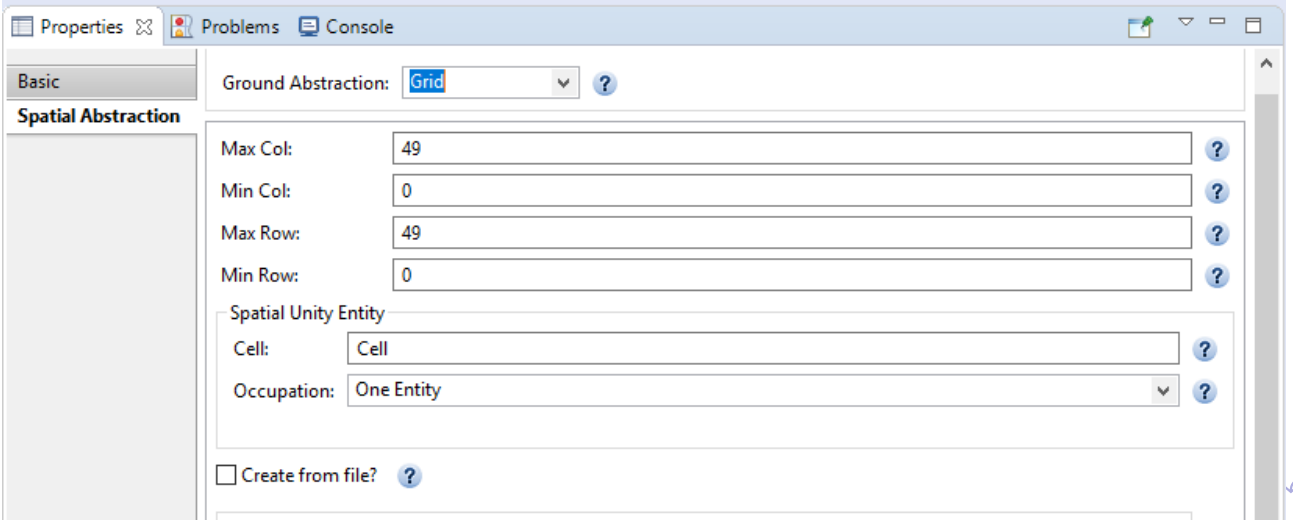
Spatial Abstraction: tipos de ambiente suportados

- Grid, Cartesian Space, e Graph
- Formado por unidades espaciais (**spatial units**)
- **Occupation**: sobreposição de entidades/agentes
  - One Entity; One Entity per Type; Many Entities

Sugarscape: grid de 50x50 células

Occupation: One Entity

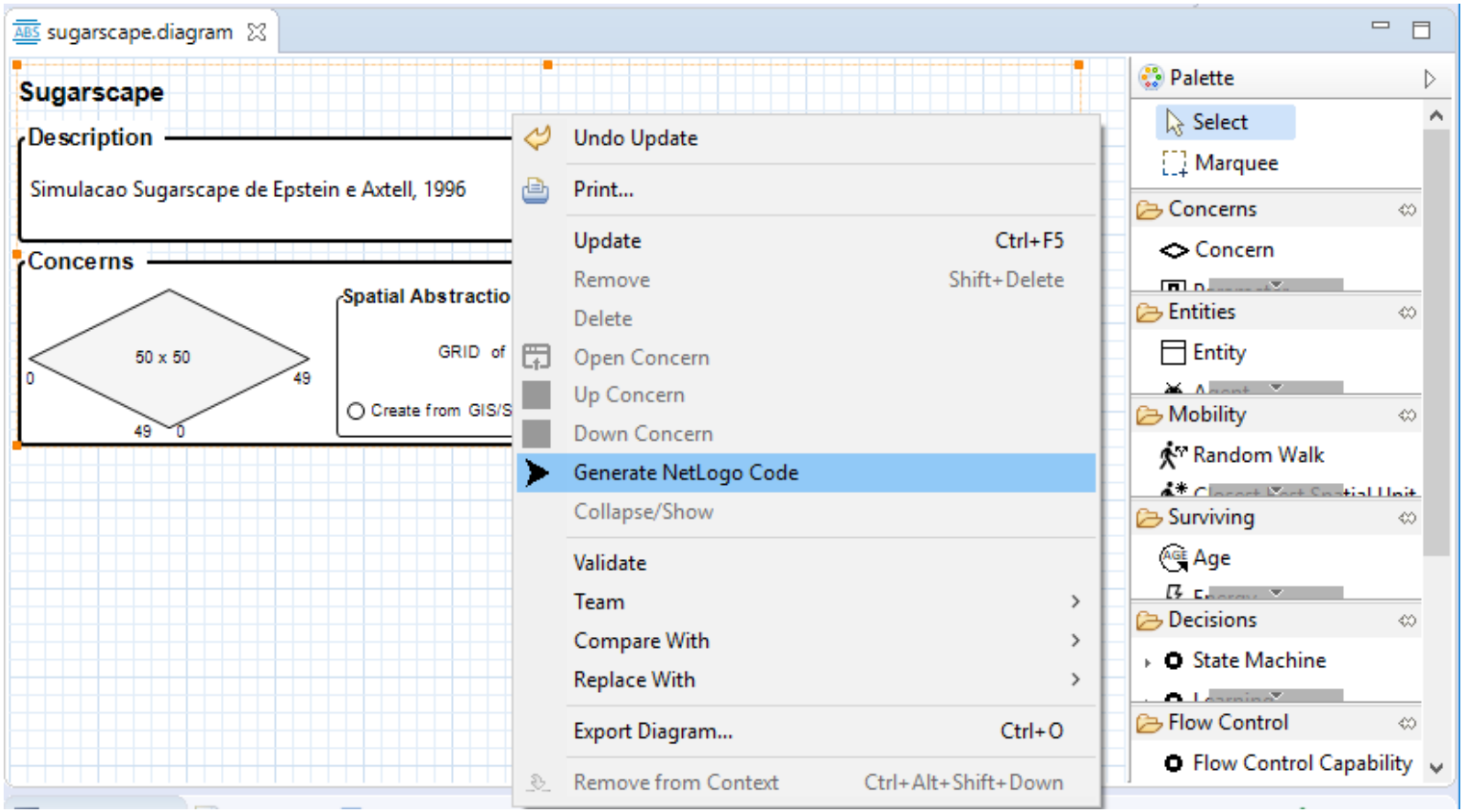




Sugarscape MDD4ABMS: Geração de Código

Botão direito → Generate NetLogo Code

- Opção disponível em qualquer diagrama



Sugarscape MDD4ABMS: Geração de Código

Status da geração de código

sugarscape.diagram

Sugarscape

Description

Simulacao Sugarscape de Epstein e Axtell, 1996

Status of the source code generation process

The source code was successfuly generated.

OK

Palette

Select

Marquee

Concerns

Concern

Entities

Entity

Mobility

Surviving

Saída do Gerador de Código

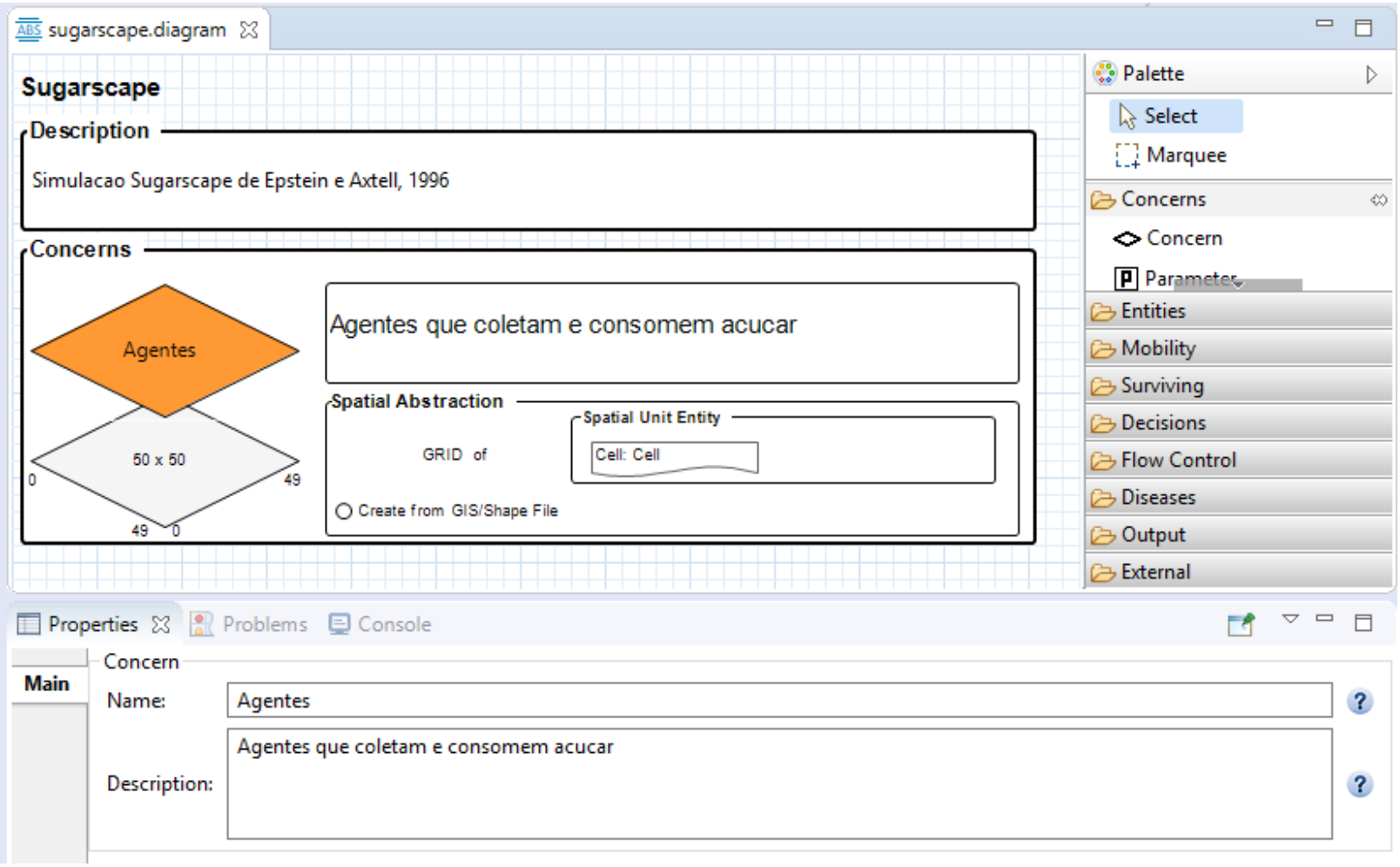
Properties | Problems | Console

ABStractme Tool

```
[INFO] Generator - Written 1 files to outlet [default](D:\Dropbox\doutorado\publicacoes_fernandosantos\papers\2018
[INFO] CompositeComponent - Generator: generating 'org::mdd4abms::m2c::xpand::NetLogoStateMachine::main FOR model'
[INFO] CompositeComponent - Generator: generating 'org::mdd4abms::m2c::xpand::NetLogoAdaptation::main FOR model' =
[INFO] CompositeComponent - Generator: generating 'org::mdd4abms::m2c::xpand::NetLogoReinforcementLearning::main F
[INFO] CompositeComponent - Generator: generating 'org::mdd4abms::m2c::xpand::NetLogoDisease::main FOR model' => D
[INFO] WorkflowEngine - workflow completed in 852ms!
```

Sugarscape MDD4ABMS: Overview (2)

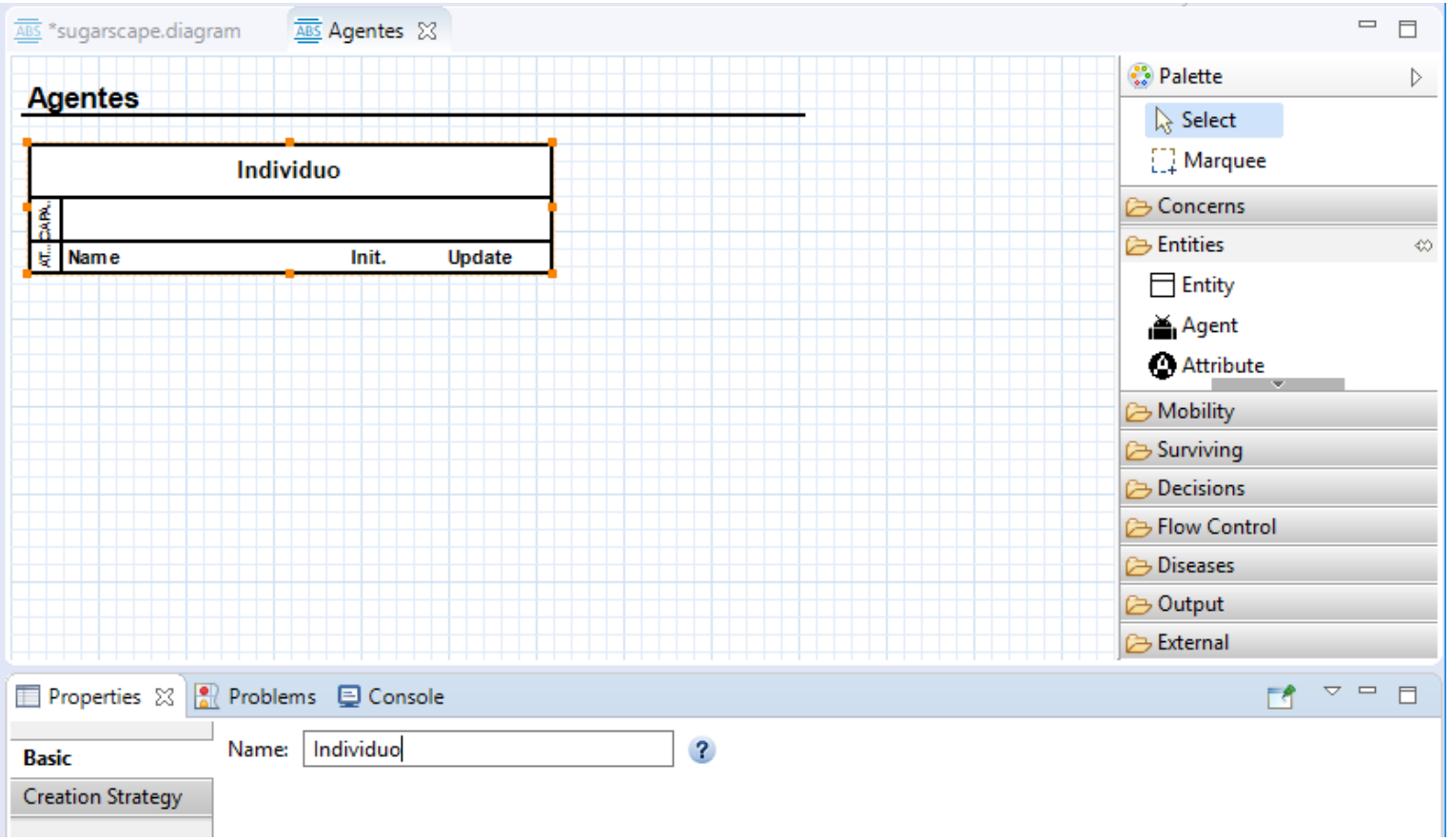
O *concern* **Agentes**



- Duplo clique no *concern* para abrir seu diagrama

Sugarscape MDD4ABMS: o Agente

O agente **Individuo**



Sugarscape MDD4ABMS: Criação do Agente

Creation Strategy: modo pré-definido para criar entidades e agentes

Estratégias disponíveis:

- **Designer defined**: usuário informa quantidade e localização
- CSV[4] file: um entidade/agente criado por linha
- **GIS file***: quantidade e localização conforme GIS
- **Line shape file***: quantidade e localização conforme "grafo"
  - Formato SHP: nós e linhas formam figuras
  - Formato OSM: nós e linhas formam malha viária

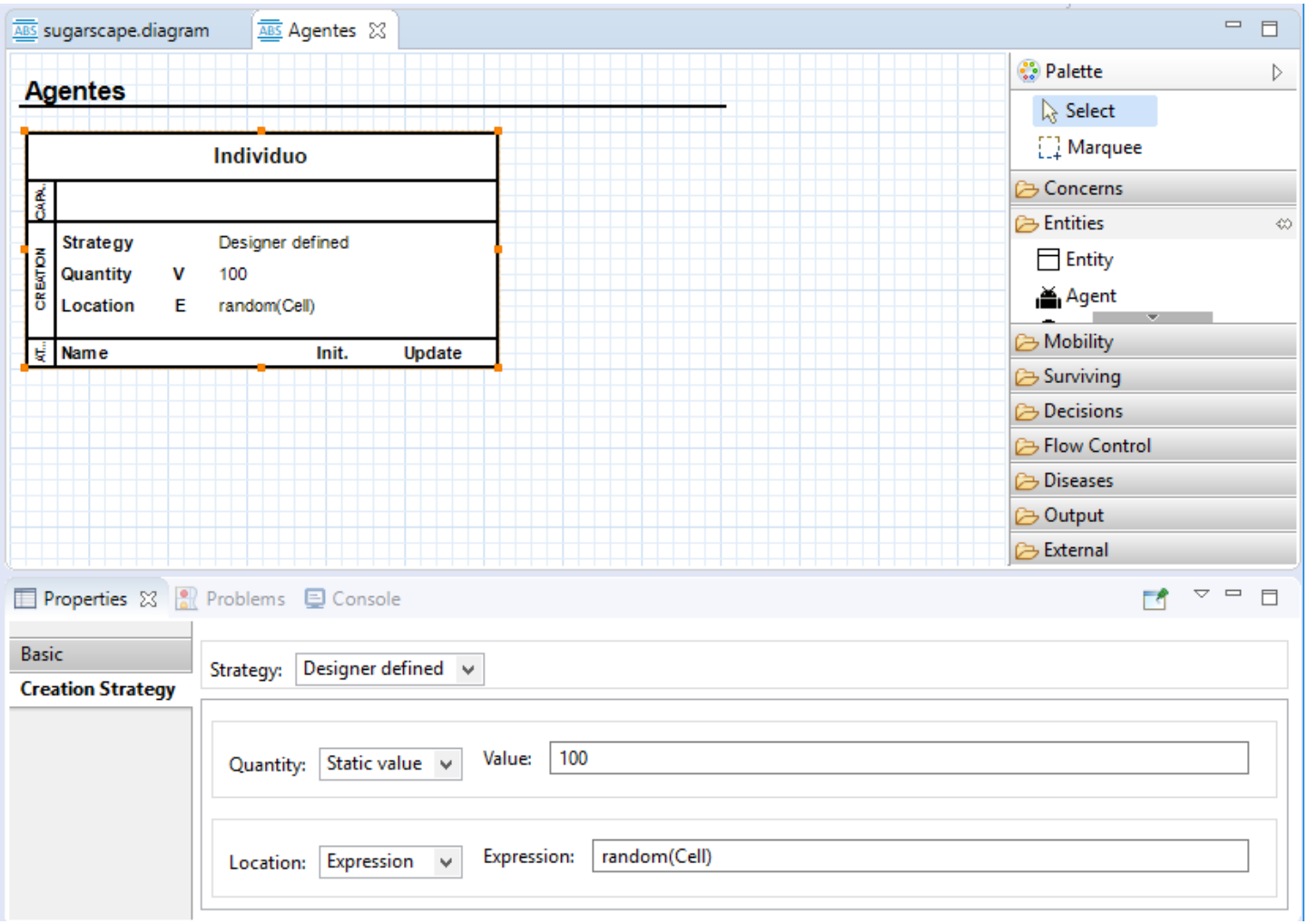
* Serão explicados nas próximas simulações que desenvolveremos

[4] *comma separated values*

Sugarscape MDD4ABMS: Criação do Agente

Agente **Individuo** com estratégia **Designer defined**

- Quantidade: 100 (*static value*)
- Localização: `random(Cell)` (*expression*)



Sugarscape MDD4ABMS: Valores

Source: modo pré-definido para prover valores ao modelo

Sources disponíveis:

Source	**Descrição**
Static value	Valor informado diretamente no modelo
Expression	O resultado da expressão define o valor
Parameter	Valor obtido de um parâmetro
Manual input	Valor provido por um componente de entrada (gerado no código)
Decision making	Uma *decision capability* provê o valor

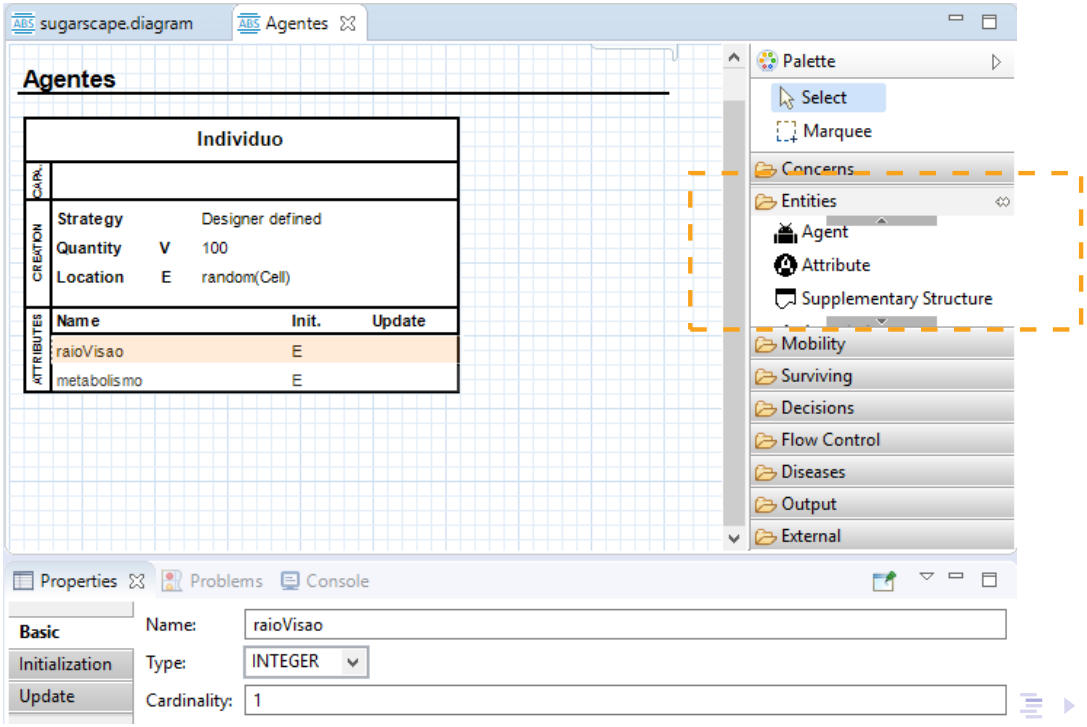
* Outros sources não são explicados por não serem pertinentes ao curso

✍ Gerar código, abrir no NetLogo, e testar

Sugarscape MDD4ABMS: Atributos

Atributos: especificações

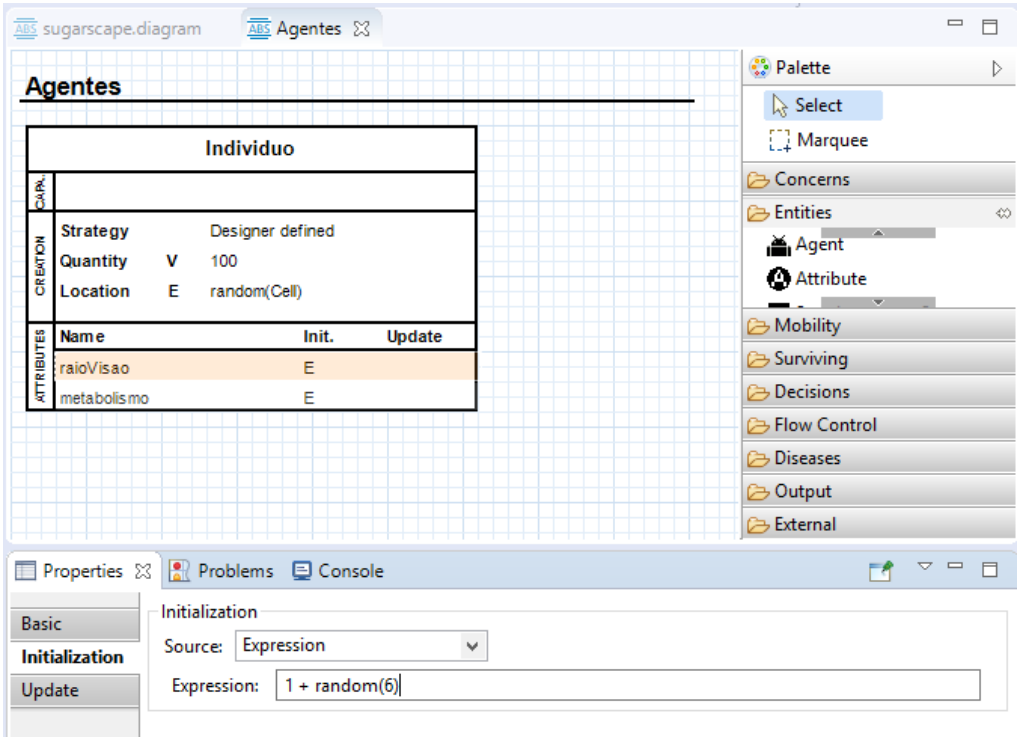
- Name: o nome do atributo (evitar espaços e acentuação)
- Type: o tipo do atributo
  - Tipos elementares: *string*, *boolean*, *integer*, *decimal*
  - Tipos complexos: entidades ou agentes
- Cardinality: a cardinalidade do atributo
- Initialization e Update: opcionais
  - Definem como atributo será inicializado / atualizado
  - Uso de **sources** para especificação



Sugarscape MDD4ABMS: Atributos do Agente

Atributos do agente **Individuo**:

- raioVisao: *integer*, cardinalidade: 1
  - Inicialização: `1 + random(6)`
- metabolismo: *decimal*, cardinalidade: 1
  - Inicialização: `1 + random(4)`
- ~~energia~~: desnecessário, usaremos habilidade de sobrevivência



✍ Gerar código, observar atributos gerados e valores inicializados

Sugarscape MDD4ABMS: as Habilidades (1)

MDD4ABMS oferece **habilidades** (comportamentos) pré-definidos

- Domínio de simulação com agentes — filosofia MDD

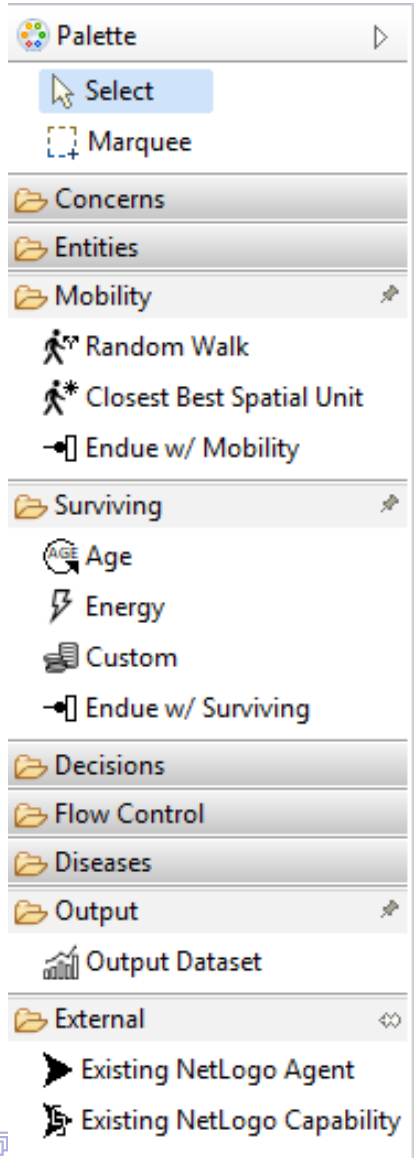
Disponíveis via **paleta**:

Mobility: movimentação do agente

- **Random Walk**: movimentação aleatória
- **Closest Best Spatial Unit**: melhor unidade espacial próxima
- **Endue w/ Mobility**: associa ao agente uma mobilidade existente

Surviving: recursos que mantém vida

- **Age**: idade/envelhecimento
- **Energy**: consumo de energia
- **Custom**: recurso customizado
- **Endue w/ Surviving**: associa ao agente um recurso existente



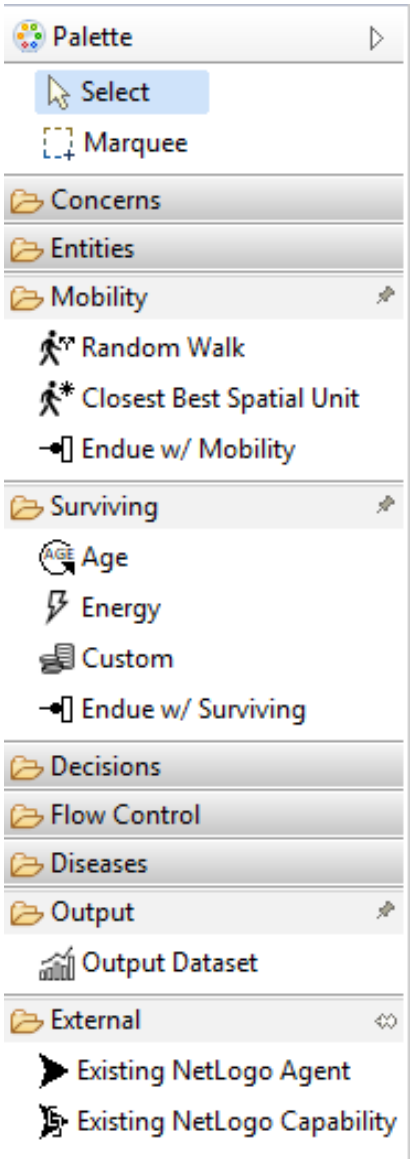
Sugarscape MDD4ABMS: as Habilidades (2)

Output: saídas da simulação

- **Output Dataset**: conjunto de saídas

External: agentes e comportamentos existentes

- **NetLogo Agent**: importa agente existente (.nls)
- **NetLogo Capability**: importa habilidade existente (.nls)



* Outras habilidades serão explicadas nas próximas simulações

Sugarscape MDD4ABMS: a Movimentação (1)

Random Walk: movimento aleatório, com ou sem distância limite

*sugarscape.diagram | *Agentes

Agentes

Individuo

CAPA. Mobility movimentacao

CREATION
Strategy Designer defined
Quantity V 100
Location E random(Cell)

ATTRIBUTES

Name	Init.	Update
raioVisao	E	
metabolismo	E	

movimentacao
«Random Walk»

Palette
Select
Marquee
Concerns
Entities
Mobility
Random Walk
Closest Best Spatial Unit
Endue w/ Mobility
Surviving
External

Identificação da Habilidade

Properties | Problems | Console

Basic
Mobility
Mobility Id.: movimentacao

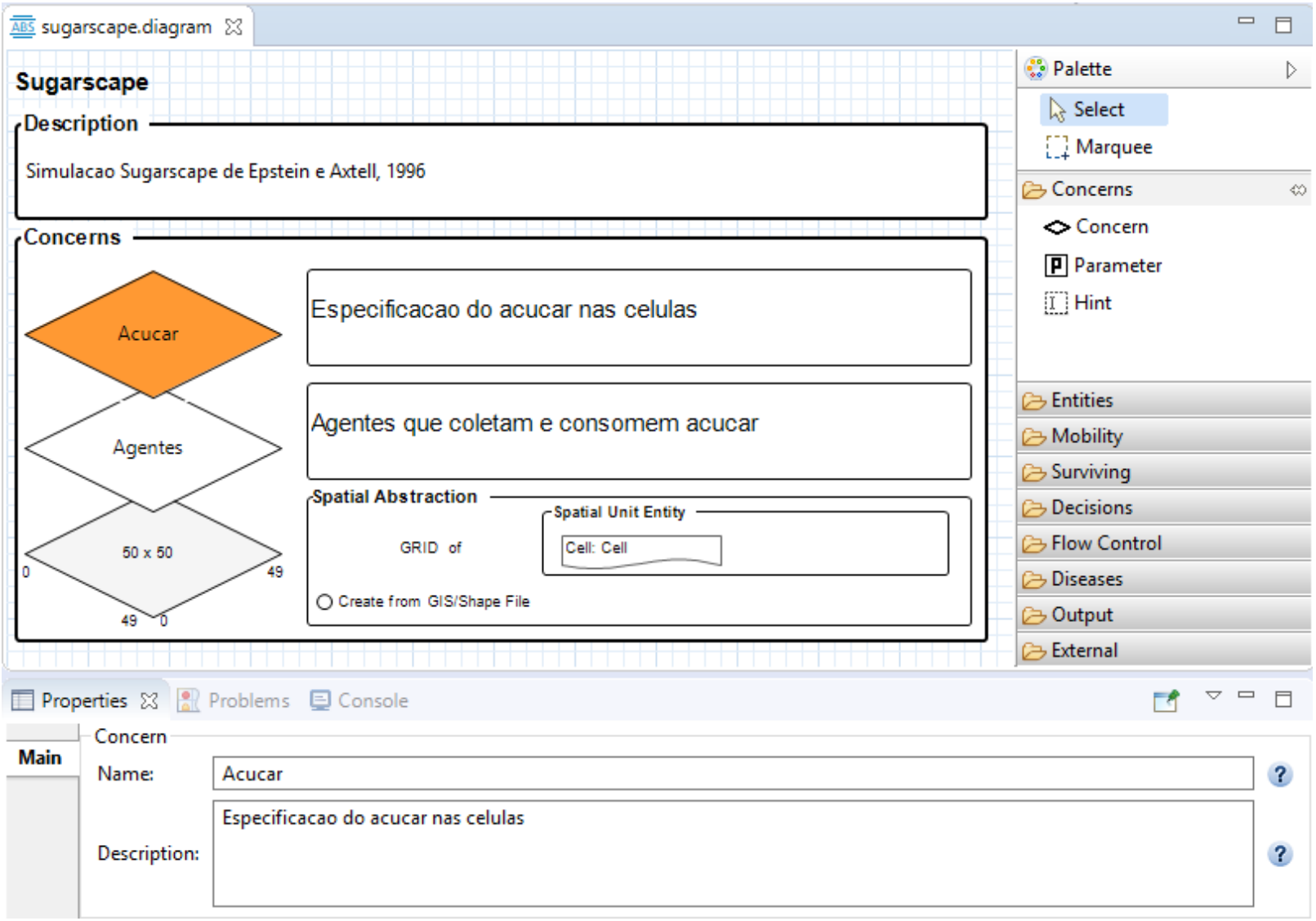
Range:
Unlimited
Limited
Lower:
Upper: raioVisao

Limites de Movimentação (em unidades espaciais)

✍ Gerar código e executar

Sugarscape MDD4ABMS: o Concern Açúcar

Agrupar a especificação do açúcar em outro *concern*: **Acucar**

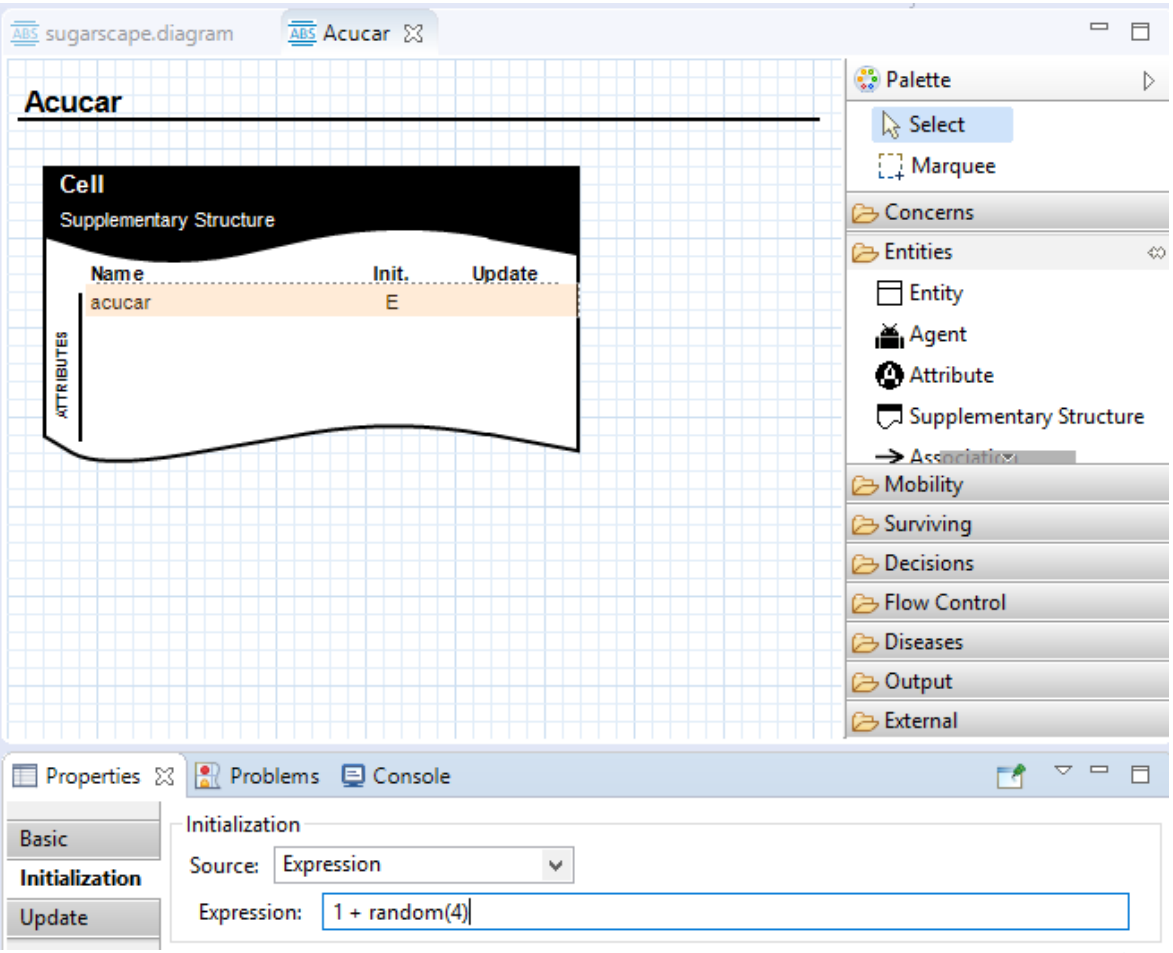


Sugarscape MDD4ABMS: o Açúcar nas Células

Definir atributo para a unidade espacial (cell)

- acucar: *decimal*, cardinalidade: 1
  - Inicialização: *expression* `1 + random(4)`

Atributos de unidades espaciais → **Supplementary Structure**



- Suplementa (complementa) a unidade espacial

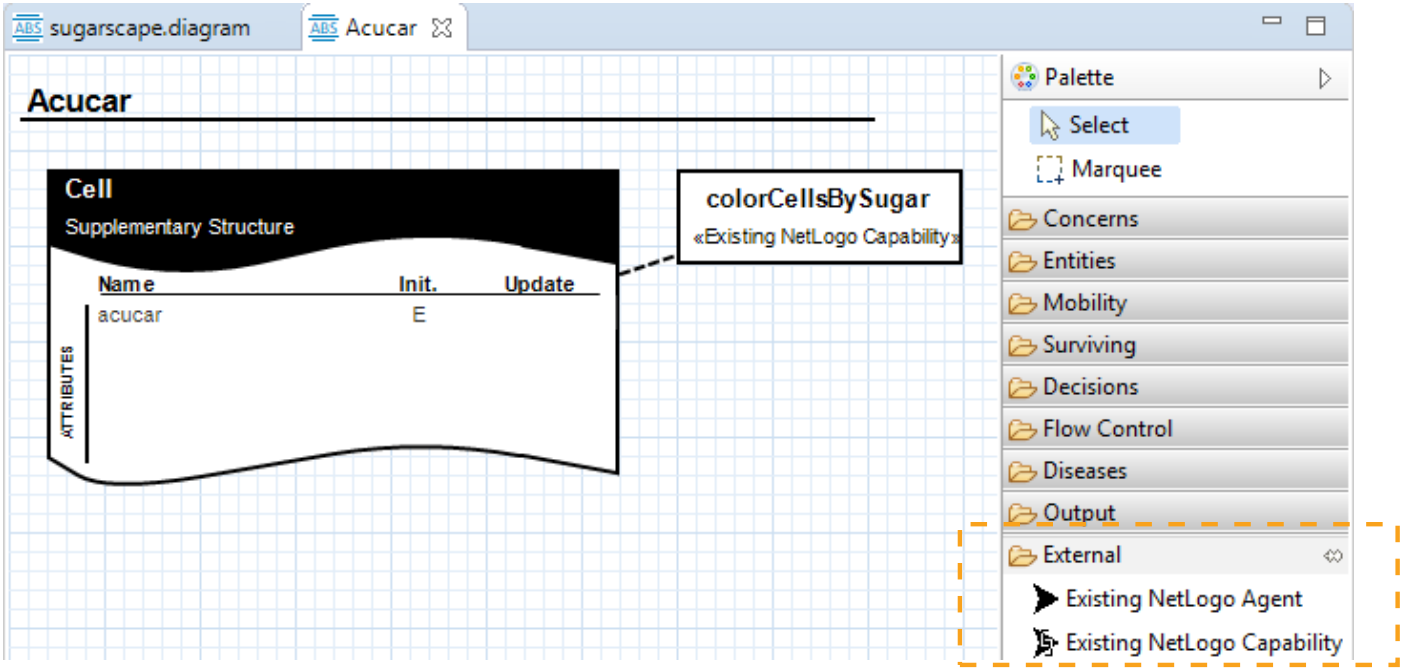
✍ Gerar código, executar, inspecionar *patches*

MDD4ABMS: Biblioteca de Habilidades

É possível *importar* habilidades externas

- Habilidades de **agentes** ou de **supplementary structures**
- Definidas em arquivo NetLogo `.nls`

Capability: `color-cells-by-sugar-mdd4abms.nls`



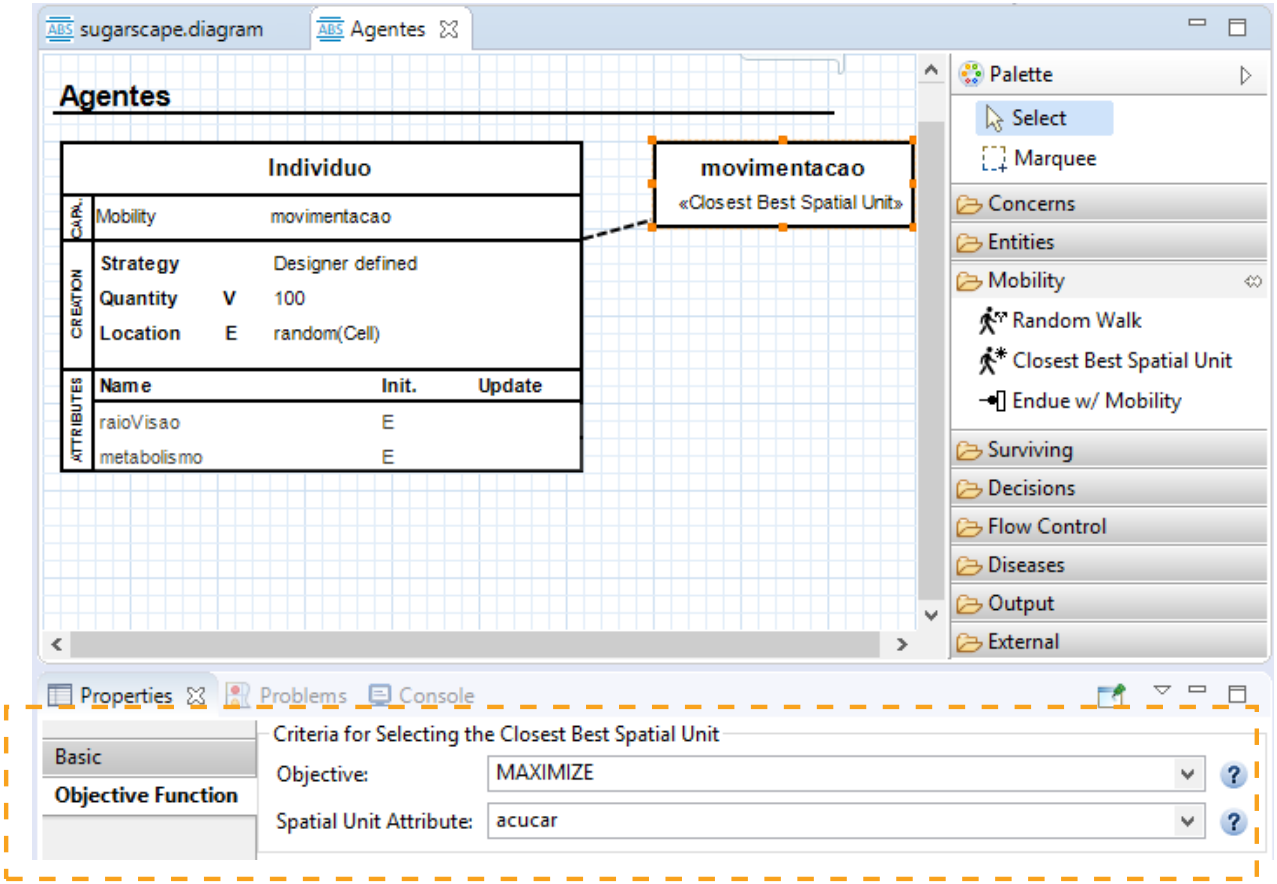
Lembrete: padronização das bibliotecas adotadas neste curso

- O arquivo `.nls` pode conter várias *procedures*
- A primeira *procedure* é a que **desempenha** o comportamento
- A *procedure* terminada com `Init` inicializa o comportamento

Sugarscape MDD4ABMS: a Movimentação (2)

Closest Best Spatial Unit: melhor unidade espacial próxima

- Range é o mesmo: *limited*, *upper*=`raioVisao`



Objective

- Selecionar a unidade espacial com valor máximo ou mínimo

Attribute

- O atributo da unidade espacial que serve de critério

✍ Gerar código e executar

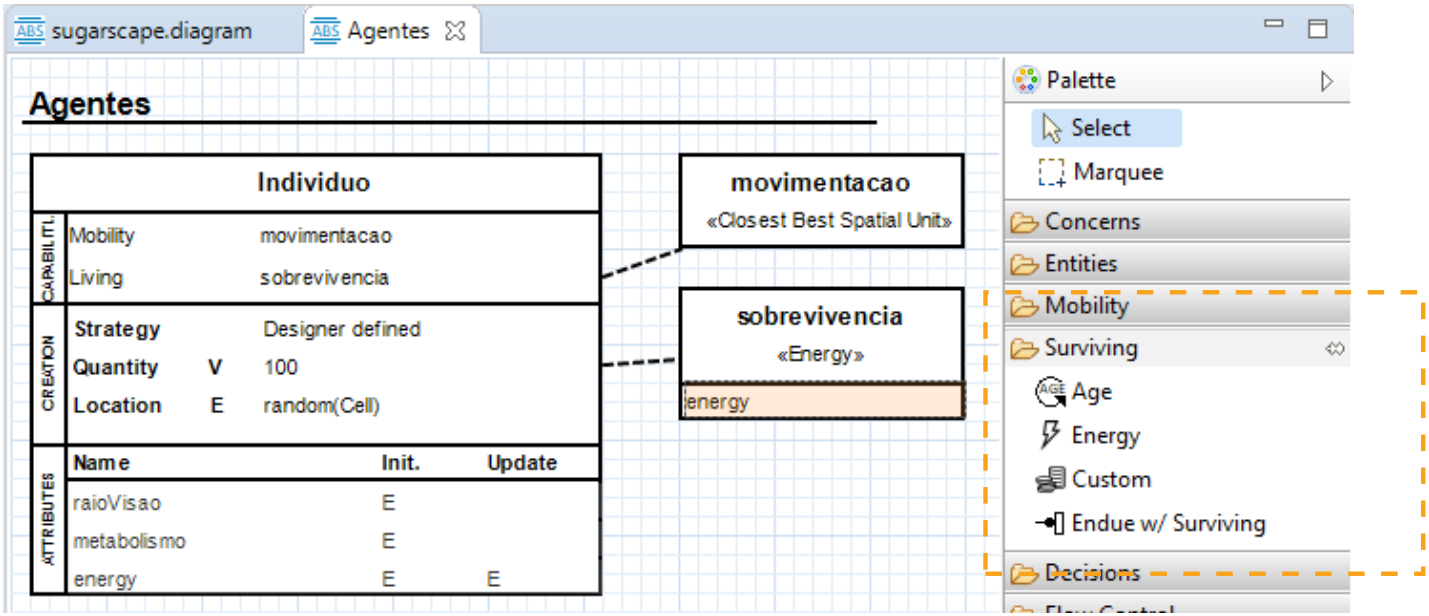
Sugarscape MDD4ABMS: Sobrevivência / Energia (1)

Surviving (sobrevivência): recursos que mantém o agente vivo

- Agente **morre** se recurso atinge certo valor (ex: 100 anos)

Exemplos:

- **Age**: idade → morte por envelhecimento
- **Energy**: energia → morte por inanição



Sugarscape MDD4ABMS: Sobrevivência / Energia (2)

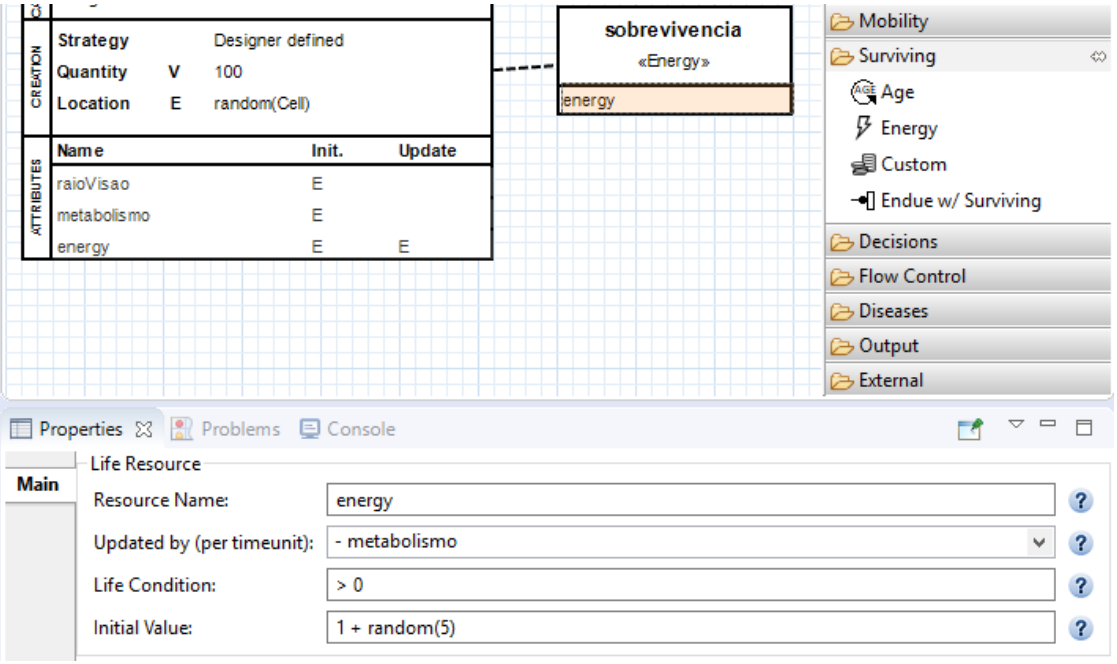
Especificação do recurso de sobrevivência: **energy**

Resource Name: `energy` (*default*)

- Nome do recurso (para criar atributo no agente)

Updated by: `- metabolismo`

- Valor de atualização do recurso a cada timestep



Life Condition: `> 0`

- Condição que mantém o agente vivo

Initial value: `1 + random(5)`

- Valor inicial do recurso

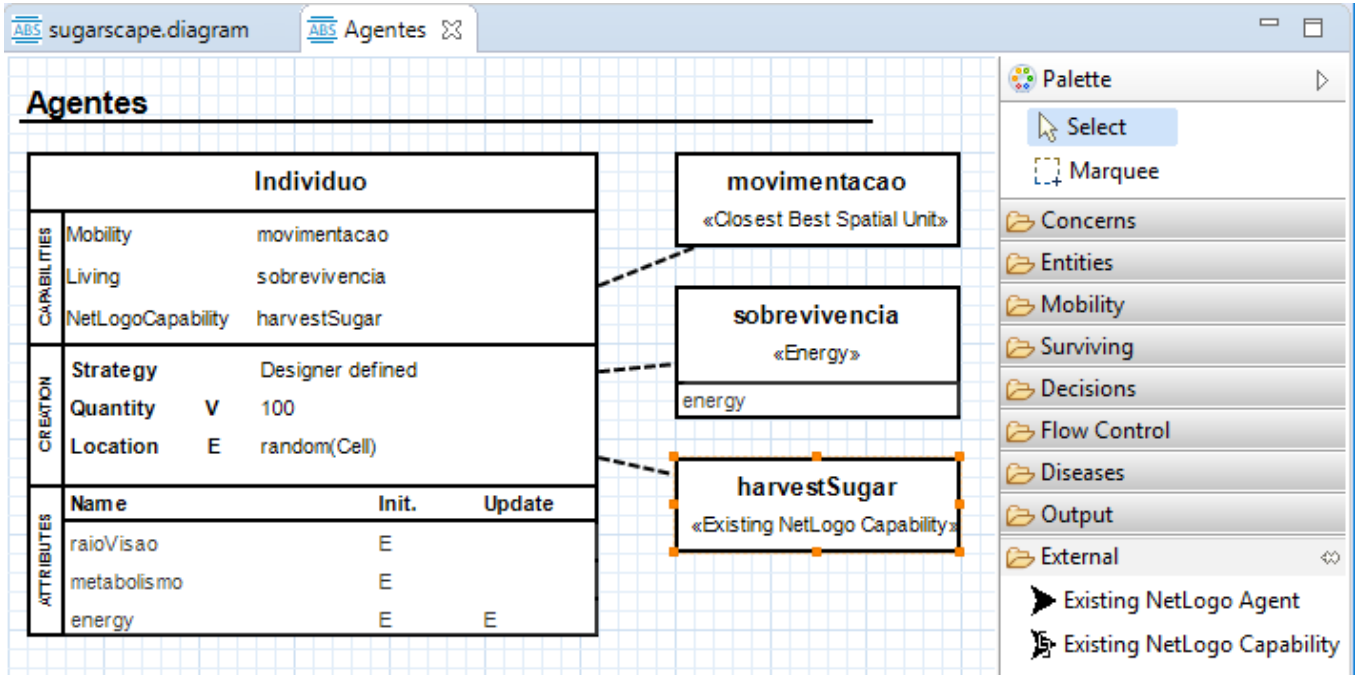
✍ Gerar código e executar

Sugarscape MDD4ABMS: Coletar Açúcar

Habilidade definida externamente

`harvestSugar.nls`

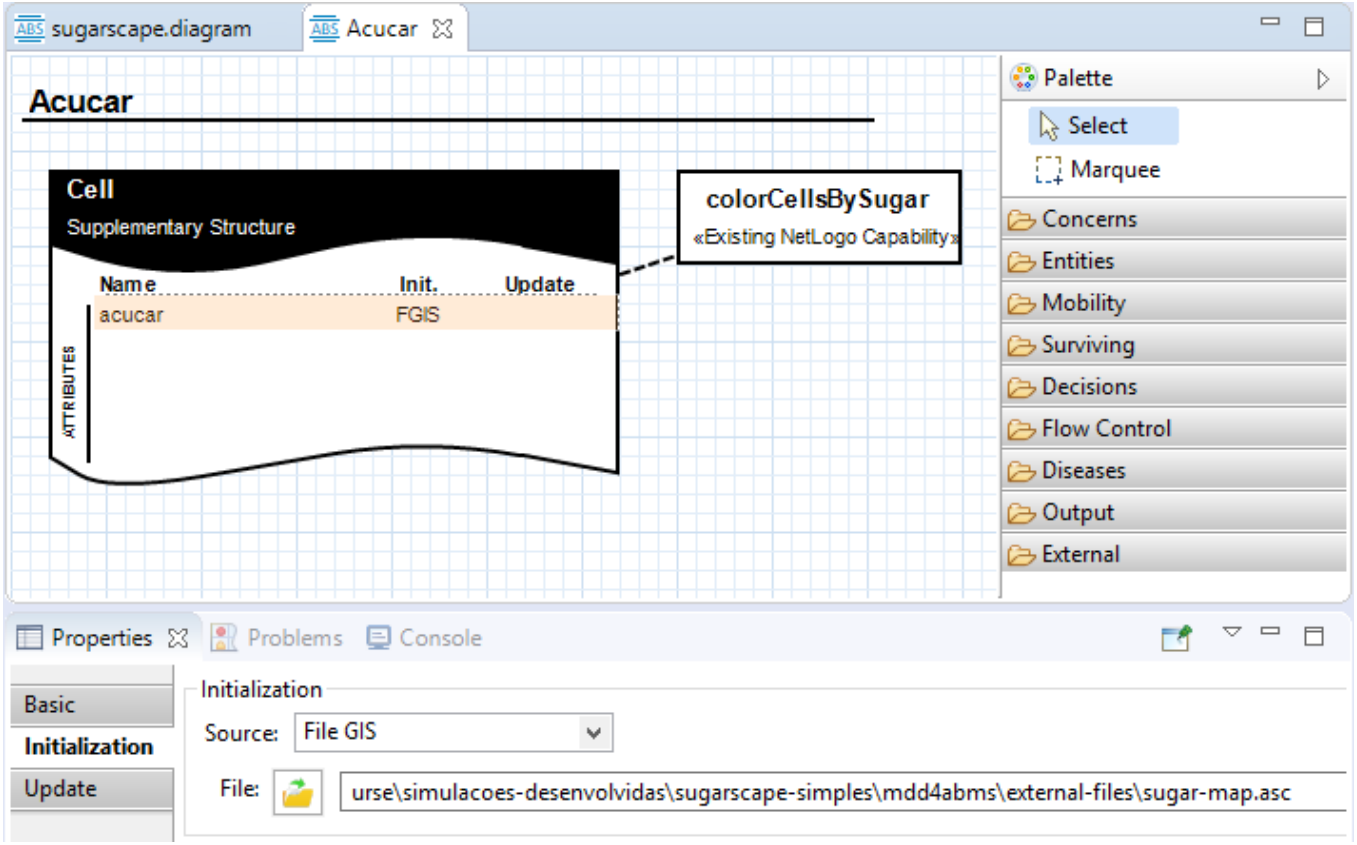
✍ Incorporar este comportamento externo no agente



✍ Gerar código e executar

Sugarscape MDD4ABMS: Açúcar de Arquivo GIS

Uso de arquivos para **inicializar** atributos



- **Source**: File GIS
- **File**: `sugar-map.asc`

✍ Gerar código e executar

Sugarscape MDD4ABMS: Reaparecimento de Açúcar

✍ Especifique o reaparecimento de açúcar

- Atributo `maxAcucar` na unidade espacial (célula)
  - Inicializado com o valor do arquivo `sugar-map.asc`
- Reaparecimento: *Update* do atributo `acucar`
  - `min (maxAcucar, (acucar + 1) )`

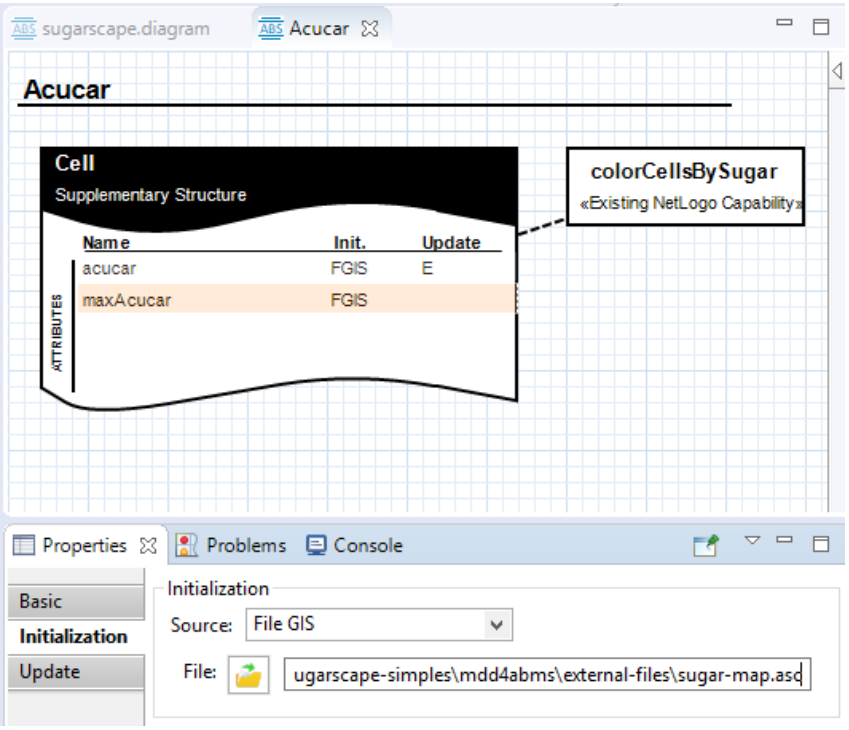
✍ Gerar código e executar

Sugarscape MDD4ABMS: Reaparecimento de Açúcar

Solução

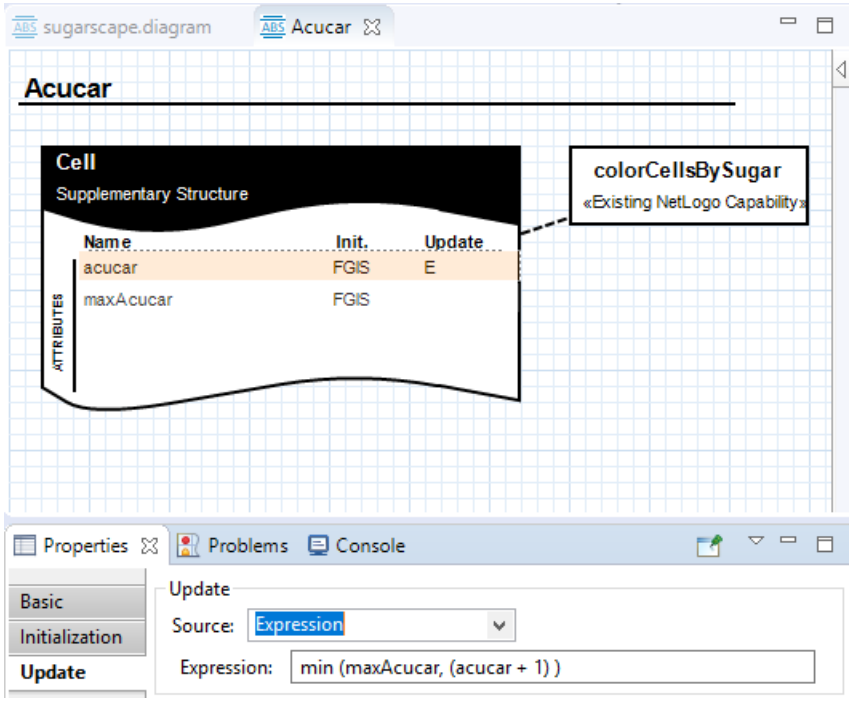
Atributo `maxAcucar` na unidade espacial (célula)

- Inicializado com o valor do arquivo `sugar-map.asc`



Reaparecimento: *Update* do atributo `acucar`

- `min (maxAcucar, (acucar + 1) )`

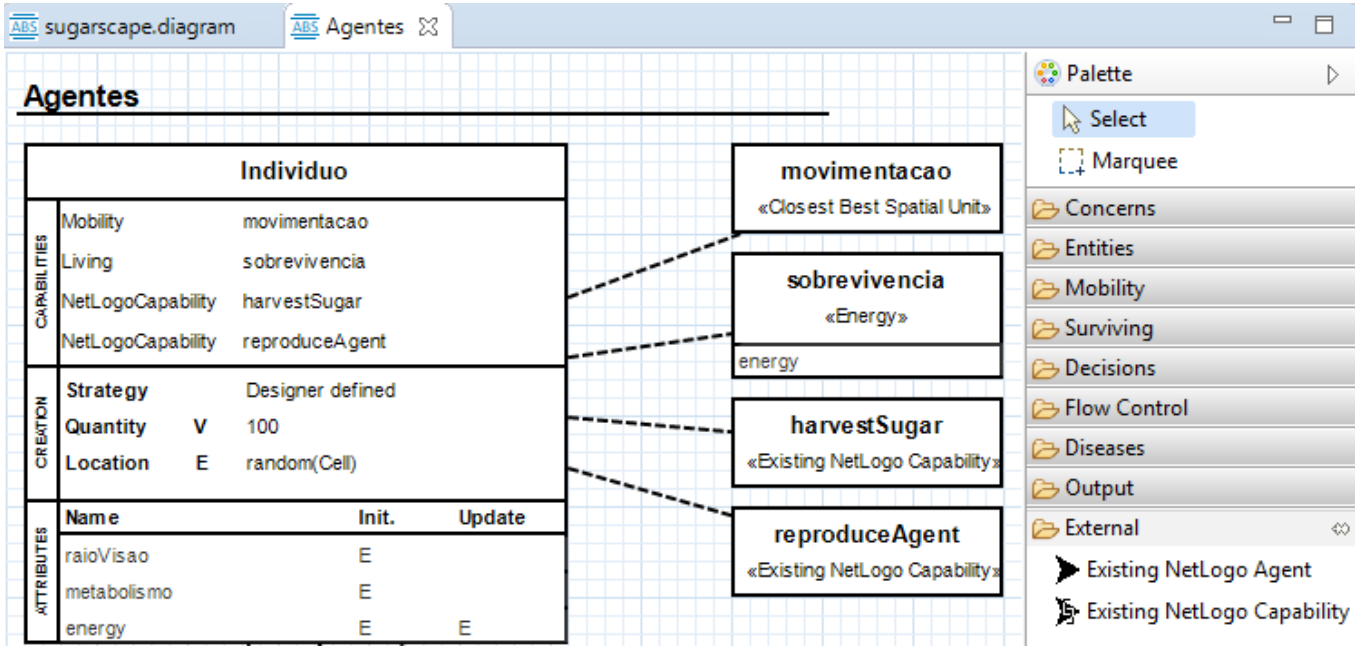


Sugarscape MDD4ABMS: Reprodução

Habilidade definida externamente

`reproduction-mdd4abms.nls`

✍ Incorporar este comportamento externo no agente



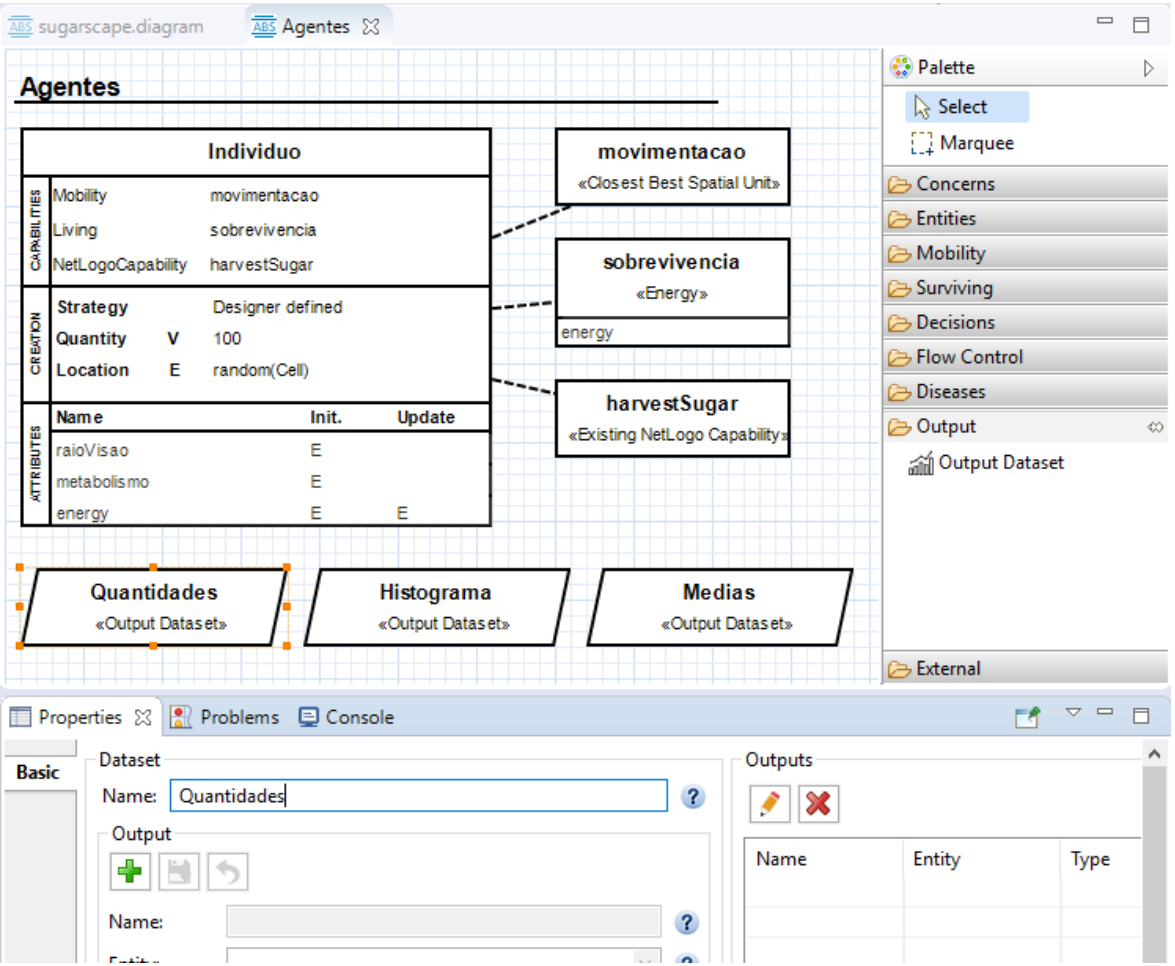
✍ Gerar código e executar

- Reprodução com mesmas características usadas anteriormente

Sugarscape MDD4ABMS: Saídas (1)

Output Dataset: conjunto de saídas da simulação

- Ao criar, não precisa "soltar" em cima de um agente
- As saídas de um *dataset* devem estar relacionadas. Exemplos:
  - Diferentes visões de um dado (ex: qtd. de agentes por tipo)
  - Diferentes dados, escalas similares (ex: visão/metabolismo)

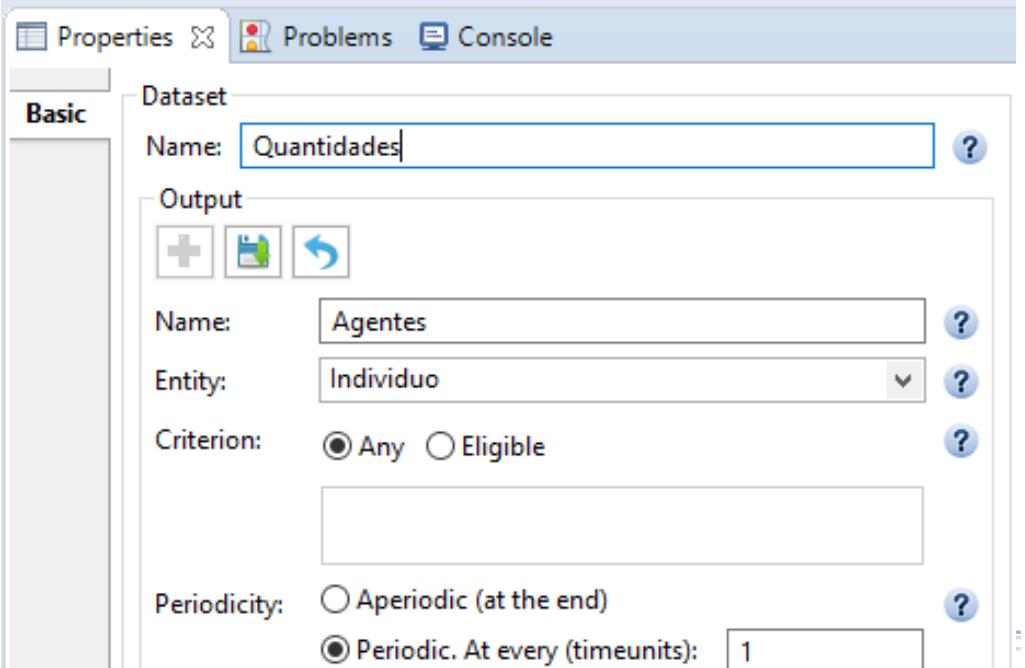


Ao criar, não precisa "soltar" sobre um agente

Sugarscape MDD4ABMS: Saídas (2)

Output: Dado que será incluso no *output dataset*

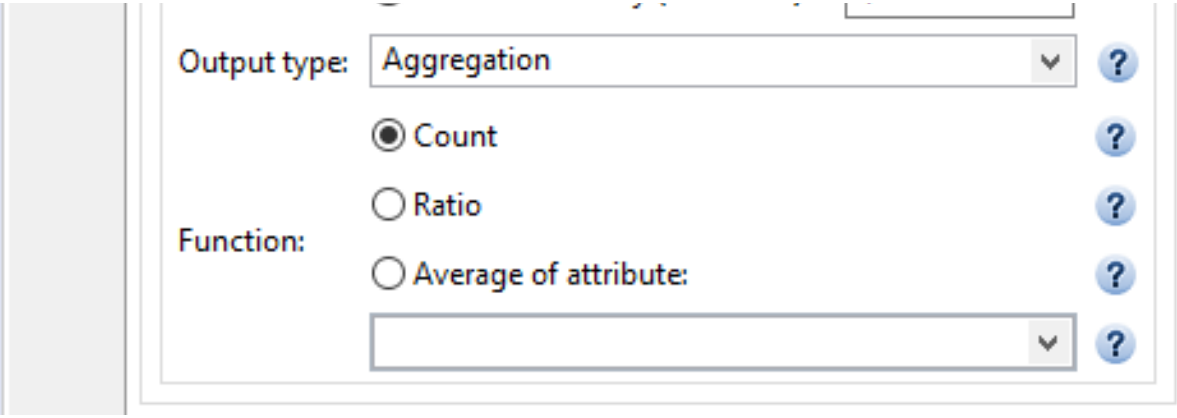
- **Name**: o nome da saída
- **Entity**: entidade ou agente considerado na saída
- **Criterion**: quais instâncias da entidade/agente serão considerados
  - **Any**: todas as instâncias
  - **Eligible**: apenas as que passam no criterio
- **Periodicity**: quando a saída será gerada/atualizada
  - **Aperiodic**: apenas no fim da simulação
  - **Periodic**: a cada X timesteps



Sugarscape MDD4ABMS: as Saídas (3)

Output (continuação)

- **Output type**: o tipo de saída
  - **Aggregation**: função de agregação
    - **Count**: quantidade de instâncias
    - **Ratio**: percentual de instâncias que atendem o critério em relação ao total de instâncias
    - **Average of attribute**: a média do valor do atributo informado
  - **Histograma**
    - **Attribute**: o atributo que fornece os valores
    - **Bins**: quantidade de barras
    - **Range**: intervalo de valores das barras



Sugarscape MDD4ABMS: as Saídas (4)

✍ Especificar os seguintes **datasets** e **outputs**

Dataset	**Output name**	**Entity**	**Criter.**	**Periodicity**	**Type**
Quantidades	Agentes	Individuo	Any	Periodic (1)	Aggregation • count
Histograma	Metabolismos	Individuo	Any	Periodic (1)	Histogram • attrib.: metabolismo • bins: 4 • range: 1...5
Medias	Visao	Individuo	Any	Periodic (1)	Aggregation • average attrib: raioVisao
	Metabolismo	Individuo	Any	Periodic (1)	Aggregation • average attrib: metabolismo

✍ Gerar código e executar

Sugarscape MDD4ABMS: Explorando a Simulação

✍ Os fenômenos anteriores ocorrem na simulação gerada?

- Evolução da Espécie por Seleção Natural?
  - Observar: média da visão/metabolismo da população
    - Efeito de alterar o raio de visão e metabolismo iniciais?

Atributo	**Min**	**Max**
raioVisao	1	10
metabolismo	1	10

  - Efeito de alterar o reaparecimento do açúcar?
    - A cada passo, apenas **0.2** de açúcar reaparece

Apêndices

NetLogo: Template de Código Básico

```
extensions [gis nw osmnlogo]  ; importação de extensões
__includes [
   "arquivo_externo1.nls"     ; importação de arquivos ex-ternos
   "arquivo_externo2.nls"
]

breed [NomePlural NomeSingular] ; define um tipo de agente ou entidade

NomePlural-own [
   atributo1   ; define atributos para o agente/entidade
   atributoN   ; atributos não possuem tipos
]

to setup
   clear-all   ; reseta a simulação
   reset-ticks ; reseta o contador de timesteps
   ; aqui vão os comandos de setup
end

to go
    ; aqui vão os comandos que caracterizam um step
    tick ; último comando é o avanço de timesteps
end
```

Referências Bibliográficas

Epstein, J. and Axtell, R. (1996).
Growing Artificial Societies Social Science From The Bottom Up.
MIT Press.