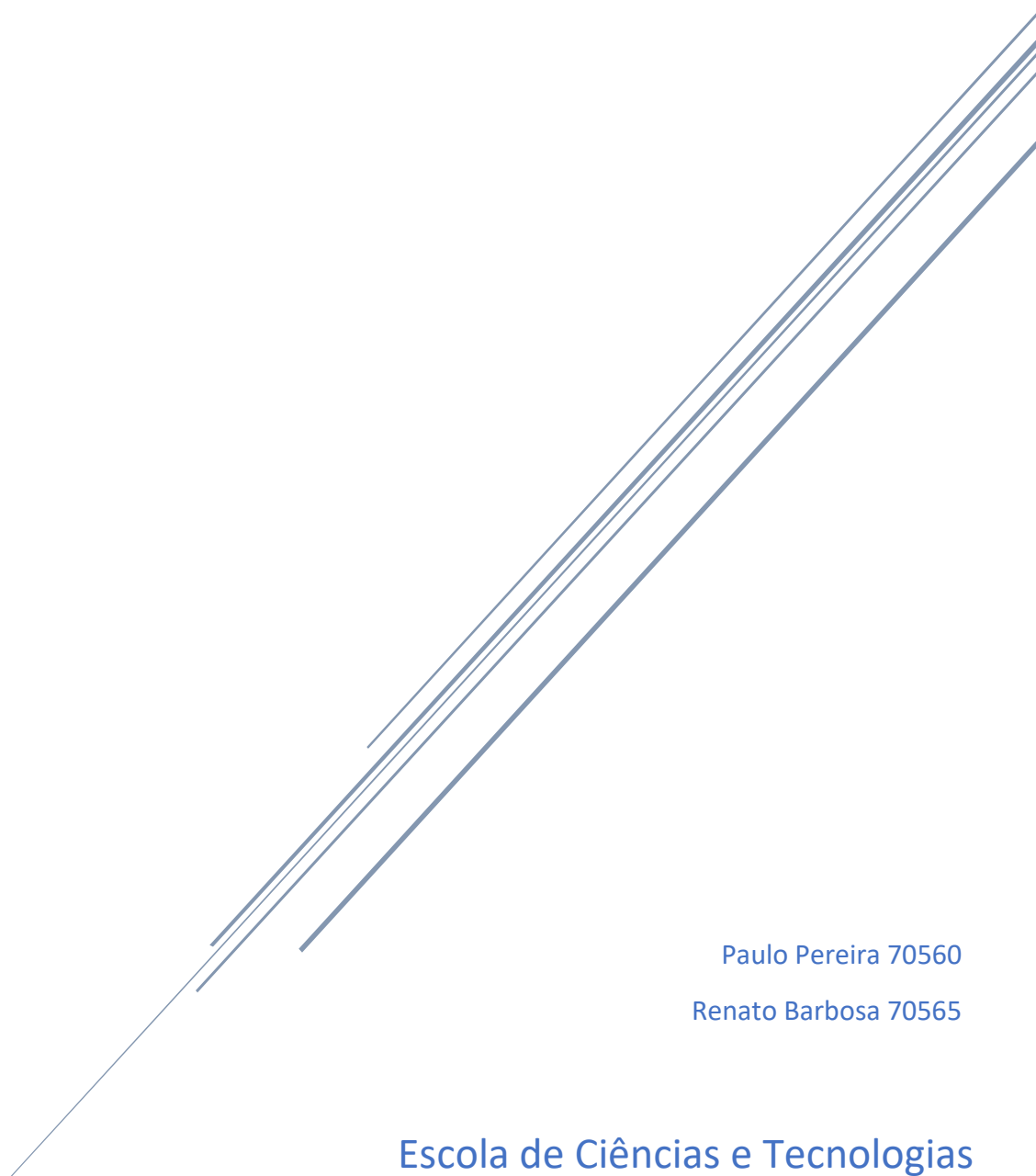


RELATÓRIO INTELIGÊNCIA ARTIFICIAL

Universidade Trás-os-Montes e Alto Douro



Paulo Pereira 70560

Renato Barbosa 70565

Escola de Ciências e Tecnologias
Licenciatura em Engenharia Informática

Introdução

No âmbito da unidade curricular **Inteligência Artificial**, foi-nos proposto a construção de um ambiente multiagente que simula um campo de relva, com o “**planta-relva**” que planta relva ao longo do campo e os parasitas, as toupeiras, que destroem a relva do campo.

Para o desenvolvimento deste projeto utilizamos o software NetLogo 6.2.0 para programar, modelar e simular sistemas com agentes racionais e Microsoft Word para escrever o relatório.

Índice

Introdução	1
Objetivos do trabalho	3
Desenvolvimento.....	4
Resposta ao Protocolo	4
A movimentação do <i>planta-relva</i>	5
• <i>Para_a_frente</i>	5
• <i>Serpentina</i>	7
• <i>Manutenção</i>	15
Gráficos:	22
População das toupeiras	22
Toupeiras comidas e procriadas	23
Qualidade da relva.....	23
Toupeiras com fome.....	24
Monitores:	24
Toupeiras comidas.....	24
Relva verde.....	25
Toupeiras Adultas	25
Movimentos Planta Relva	25
Morta por Fome.....	26
Toupeiras com Fome.....	26
Principais problemas encontrados	27
Conclusão	31
Referências	32

Objetivos do trabalho

O objetivo do projeto é efetuar alterações no tutorial do agente *planta-relva*, feito anteriormente, desenvolvendo novas funcionalidades e adicionando agentes racionais (ex.: *toupeira*) ao sistema elevando os nossos conhecimentos de sistemas multiagentes utilizando a ferramenta NetLogo.

Para o desenvolvimento destas funcionalidades não podemos considerar possível os agentes serem capazes de perceberem mais do que o patch à sua frente.

Desenvolvimento

Resposta ao Protocolo

O Projeto possui um protocolo detalhado que devemos seguir com funcionalidades que o sistema deve ter. Assim sendo, nesta fase do relatório explicamos o que é pedido em cada ponto do protocolo assim como a resposta que nós demos para resolver o problema.

Pergunta

1. Altere a localização da origem do ambiente (0,0) para o canto inferior esquerdo.

Resposta

Para alterar a localização de origem do ambiente abrimos o menu de opções da “**View**” carregando com o botão do lado direito do rato sobre a mesma e selecionamos a opção “**Editar**”. De seguida alteramos o campo “**Localização da origem**” para “**Canto**” e “**Abaixo e à esquerda**”.

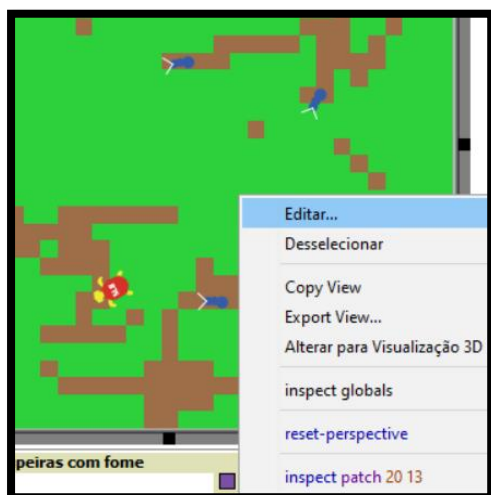


Figura 1: Carregar com o botão do lado direito para abrir o menu de opções da View

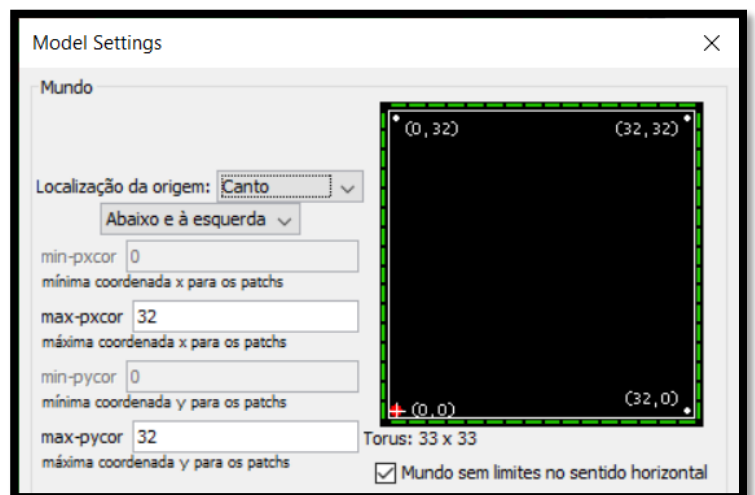


Figura 2: Definir localização de origem para Canto "Abaixo e à esquerda"

Pergunta

2. Insira um botão que permita ao planta-relva fazer a plantação de todo o campo.

Essa plantação poderá ser feita (pelo menos) de dois modos:

- ***Para_a_frente***: o planta-relva é inicializado com uma direção (heading) aleatória entre quatro posições, com ângulos de 0°, 90°, 180° e 270°, seguindo essa direção na plantação do campo.
- ***Serpentina***: o planta-relva é inicializado com uma direção ângulos de 0° seguindo essa direção até atingir o limite superior do campo, invertendo então o sentido.

Resposta

A movimentação do ***planta-relva*** têm três tipos de funcionamento: ***Frente***; ***Serpentina***; ***Manutenção***.

O modo “***Manutenção***” apenas é explicado no ponto 8).

- ***Para_a_frente***

Neste tipo de movimentação o agente procura sempre andar em frente, quando, através do sensor, vir que não é possível, escolhe virar ou para a esquerda ou para a direita, isto se um dos patches estiver disponível. Caso o patch da esquerda e da direita não estejam disponíveis, o ***planta-relva*** atualiza o modo de plantação para “***Manutenção***”.

Antes de se movimentar, é inicializada na função ***criar_planta-relva*** um agente do tipo “***planta-relva***” com uma direção (heading) aleatória, com ângulos de 0°, 90°, 180° e 270° como mostra a *Figura 3*.

```
to criar_planta-relva [x y]
  create-planta-relvas 1[
    set heading one-of[0 90 180 270]
    setxy x y
    ask patch-here[set pcolor lime]
    if modo_de_plantar = "Serpentina"[set heading 0]
    set size 2.5
  ]
end
```

Figura 3: Comando set heading na função criar_planta-relva

Depois de criado o agente, é iniciado o seu movimento.

```
to plantar_frente
  ifelse [pcolor] of patch-ahead 1 = brown      ;Verificar se a patch da frente pode ser plantada
  [
    avançar_plantar                               ;Avança e planta relva
  ]
  [
    if check-left != true [left 90]              ;Verifica se pode plantar do lado esquerdo
    if check-right != true [right 90]            ;Verifica se pode plantar do lado direito

    if[pcolor] of patch-ahead 1 = lime ;        ;Verifica se está rodeado de relva
    [
      set modo_de_plantar "Manutenção"          ;Atualiza o modo de plantação para "Manutenção"
      set contador_patch 0                      ;Inicializar variável para o modo "Manutenção"
    ]
  ]
end
```

Figura 4: Função plantar_frente

Primeiramente verifica se a cor (*pcolor*) do patch à sua frente (*patch-ahead 1*) é castanho (*brown*) e se esta condição se verificar verdadeira (*true*) o *planta-relva* avança e planta a relva através da função “*avançar_plantar*”. No caso da condição se verificar falsa (*false*), o *planta-relva* prossegue para mais verificações.

Para isto, chama a função “*check-left*” para rodar 90° para a esquerda e ver se a cor do novo patch à sua frente é lima. Se for lima, a função “*check-left*” vira o *planta-relva* 90° para a direita (*right 90*), para a sua frente “*antiga*”, e retorna verdadeiro (*true*) como podemos observar na Figura 5.

```
to-report check-left
  left 90
  ifelse [pcolor] of patch-ahead 1 = lime
    [right 90 report true]
    [right 90 report false]
end
```

Figura 5: Função check-left

Se a função check-left retornar false, então significa que o *planta-relva* pode plantar do lado esquerdo e vira-se para essa direção.

Depois de serem chamadas as duas funções de verificação “*check-left*” e “*check-right*” é feita uma última verificação onde é verificada a cor do patch à frente. Se for lima, então é porque o *planta-relva* se encontra rodeado de relva e por isso entra no modo “*Manutenção*” e inicializa o *contador_patch* a 0 (caso não seja a primeira vez que o modo de plantar seja atualizado para “*Manutenção*”, ajuda a limpar valores antigos do contador que possam vir a prejudicar o bom funcionamento do mesmo).

- *Serpentina*

Neste tipo de movimentação o agente procura andar em serpentinas, vai andando sempre em frente até encontrar um patch com relva à sua frente e de seguida vai *alternando* entre *virar à direita* e *virar à esquerda*.

Se não for possível continuar a plantar a relva em serpentina, é ativado o modo “*Manutenção*”.

Antes de ser iniciado o movimento, é inicializada a direção (heading) com o ângulo de 0° como podemos observar na *Figura 3*.

Se o modo de plantar for alterado durante a execução do programa (depois de já estar criado o *planta-relva*) ele continua com a direção que tinha até iniciar o novo movimento como mostrado na *Figura 6*.

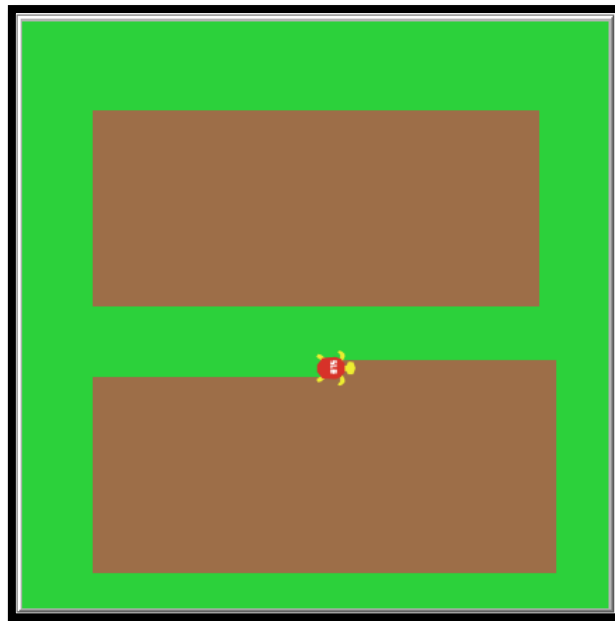


Figura 6: Alteração para modo de plantar “Serpentina” em run-time

Primeiramente verifica se a cor (*pcolor*) do patch à sua frente (*patch-ahead 1*) é castanho (*brown*) e se esta condição se verificar verdadeira (*true*) o *planta-relva* avança e planta a relva através da função “*avancar_plantar*”. No caso da condição se verificar falsa (*false*), o *planta-relva* prossegue para mais verificações.

É feita uma dupla verificação para ver se existe relva dos dois lados do *planta-relva* (lado direito e lado esquerdo) e no caso de existir o modo de plantar é atualizado para “*Manutenção*”. Esta dupla verificação é feita utilizando as funções “*check-left*” e “*check-right*” como podemos ver na Figura 7.

No caso de não existir, avançamos para a verificação da *flag* (variável global que funciona como uma lâmpada que alterna entre ligada e desligada) para sabermos se temos de fazer uma *curva* de 180° para a *direita* (*turn_flag* = 0) ou para a *esquerda* (*turn_flag* = 1). Esta curva é feita através das funções “*right180*” e “*left180*” visíveis na Figura 8.

```
to plantar_serpentina
  ifelse [pcolor] of patch-ahead 1 = brown ;Verificar se a patch da frente pode ser plantada
  [
    avançar_plantar ;Avança e planta relva
  ]
  [
    if grass-right and grass-left ;Verifica se tem relva do lado direito e do lado esquerdo
    [
      set modo_de_plantar "Manutenção" ;Atualiza o modo de plantação para "Manutenção"
      set contador_patch 0 ;Inicializar variável para o modo "Manutenção"
    ]
    ifelse turn_flag = 0 ;Para a flag "turn_flag" :
    [
      right180 ;turn_flag = 0 (False) -> Inverte o sentido pelo lado da direita
    ]
    ;Else
    [
      left180 ;turn_flag = 1 (True) -> Inverte o sentido pelo lado da esquerda
    ]
  ]
end
```

Figura 7: Função plantar_serpentina com código comentado

```
to right180
  set heading heading + 90
  avançar_plantar
  set heading heading + 90
  set turn_flag 1
end
```

```
to left180
  set heading heading - 90
  avançar_plantar
  set heading heading - 90
  set turn_flag 0
end
```

Figura 8: Funções responsáveis por virar 180º à direita e à esquerda

Pergunta

3. Insira um seletor que permita escolher o modo de funcionamento do “*planta-relva*”. De acordo com o modo selecionado o “*planta-relva*” deve agir em conformidade.

Resposta

Para introduzirmos um seletor apenas tivemos de escolher o “*Seletor*” no menu da “*Interface*” (ver *Figura 9*) e carregar com o botão do lado esquerdo do rato num lugar qualquer da interface.

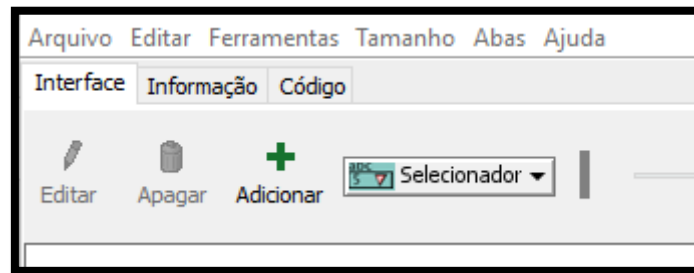


Figura9: Menu Interface selecionado

Depois disto, temos de definir o nome da variável global que vamos utilizar e adicionar as diferentes escolhas que podemos fazer “*Frente*”, “*Serpentina*” e “*Manutenção*” como podemos ver na *Figura 10*.



Figura 10: Seletor com as opções dos modos de plantar

Tendo as funções utilizadas para responder ao ponto 2), podemos agora juntar tudo numa função para simplificar a leitura do código e a compreensão do mesmo (ver *Figura 11*).

```
to andar_planta-relva  
  if modo_de_plantar = "Frente" [plantar_frente]  
  if modo_de_plantar = "Serpentina"[plantar_serpentina]  
end
```

Figura 11: Função *andar_planta-relva* para facilitar a leitura e compreensão do código

Assim, na função “*go*” podemos chamar apenas “*andar_planta-relva*”, não é preciso escrever o código todo. No futuro adicionaremos também a esta função o modo de plantar “*Manutenção*”. Este tipo de simplificação ajuda a ler e interpretar o código mais facilmente assim como para prevenir erros, como por exemplo problemas de falta de parênteses retos, funções colocadas no sítio errado...

Pergunta

4. Considerando que o relvado pode ser danificado por toupeiras, altere o modelo de forma que existam duas espécies de agentes: *planta-relva* e *toupeira*.

Resposta

Para adicionarmos uma *nova espécie* de agentes ao sistema temos de utilizar o comando “*breed* [nome-da-especie-plural nome-da-especie-singular]” (como pode observar na Figura 12).

```
breed [planta_relvas planta_relva]  
breed [toupeiras toupeira]
```

Figura 12: Código para adicionar duas espécies de agentes

Para distinguirmos uma espécie de agentes da outra, é necessário na função “*setup*” definirmos uma forma predefinida (“*default-shape*”). No nosso caso utilizamos a forma de uma *tartaruga* (“*turtle*”) para os “*planta-relvas*” e a forma *inseto* (“*bug*”) para as *toupeiras*.

```
to setup
  inicializar_campo

  set-default-shape planta_relvas "turtle"
  criar_planta-relva 0 0

  set-default-shape toupeiras "bug"
  reset-ticks
end
```

Figura 13: Função setup onde são inicializadas as formas das espécies de agentes

Pergunta

5. Insira um botão com o nome “**Inserir_Toupeira**”, cujo programa permite inserir uma toupeira no campo numa localização e com direção aleatórias. Cada vez que o botão é pressionado é inserida uma toupeira.

Resposta

Para inserirmos o botão “**Inserir_Toupeira**” primeiro temos de criar a função que o botão vai chamar. Foi dado à função o nome criativo “**Inserir_Toupeira**” para que cada vez que seja chamada, insira uma toupeira com *coordenadas aleatórias* e com uma *direção aleatória* de 0° a 360°.

```
to inserir_toupeira [x]
  let pares[[40 0.05][80 0.9][320 0.05]]
  create-toupeiras x[
    set energy first rnd:weighted-one-of-list pares [[p] -> last p]

    ( ifelse
      energy = 320 [set color black ]
      energy = 80 [set color blue ]
      energy = 40 [set color red] )
    setxy random sqrt(count patches) random sqrt(count patches)
    set size 2
    set heading random 360
  ]
end
```

Figura 14: Função inserir_toupeira que recebe o parametro para determinar o número de toupeiras inseridas

Para explicar um pouco o funcionamento deste código é preciso referir que utilizamos a extensão “*rnd*” capaz de trabalhar com **probabilidade condicionada** que nos ajudou a concretizar a ideia de **imprevisibilidade** mantendo um **padrão** que fizesse sentido.

Através desta extensão fomos capazes de, quando é introduzida uma nova toupeira, definir **diferentes energias iniciais** para diferentes toupeiras, sendo a mais comum, com uma probabilidade de **90%**, a energia ser 80 e de seguida com percentagens de **5%** a energia ser 40 ou 320. O **objetivo** deste código valores é **simular o ambiente real** onde é esperada uma cria (toupeira bebe) saudável e “**normal**”, porém pode nascer uma cria “**desnutrida**” (com fome) ou com **peso a mais** (fértil). Para saber mais sobre a extensão “*rnd*” [clique aqui](#).

Nota: A explicação do porquê de utilizarmos energia será feito na parte dos “**Extras**”

Para um entendimento mais detalhado de cada linha de código, temos as linhas comentadas no ficheiro “*.nlogo*”.

Feito isto, inserimos um botão com o nome da função para cada vez que clicarmos no botão ser chamada a função.

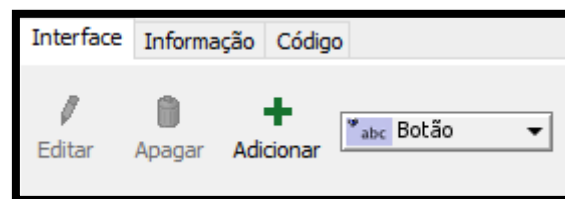


Figura 15: Selecionar a opção "Botão" no Seleccionador do menu Interface

Pergunta

6. A circulação da(s) toupeira(s) é determinada por um valor de probabilidade (**Prob_toupeira**). Quando maior a probabilidade maior atividade deve ter a toupeira. A toupeira destrói a relva nas células por onde passa.

Resposta

Para o utilizador efetuar a escolha da probabilidade (**Prob_toupeira**) escolhemos adicionar um deslizador com início em 0 e fim em 1 que incrementa 0.01 unidades.

No código foi feita a verificação se a probabilidade (**Prob_toupeira**) é superior a um número real aleatório entre 0 e 1 (**random-float 1**) como podemos ver na *Figura 16*.

```
if Prob_toupeira > (random-float 1)[  
  andar_toupeira
```

Figura 16: Verificação de Prob_toupeira

Pergunta

7. Insira um deslizador que permita definir o valor da **Prob_toupeira**.

Resposta

Para introduzirmos um deslizador apenas tivemos de escolher o “**Deslizador**” no menu da “**Interface**” (ver *Figura 12*) e carregar com o botão do lado esquerdo do rato num lugar qualquer da interface.

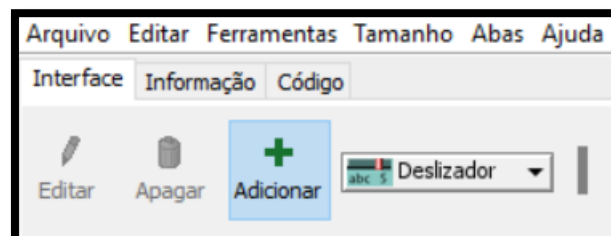


Figura 17: Menu interface selecionado

Ao criar o deslizador vai abrir o menu do deslizador onde podemos definir o valor mínimo, valor máximo, o incremento e o nome da variável global. Neste exemplo usamos os seguintes valores (ver imagem 13).

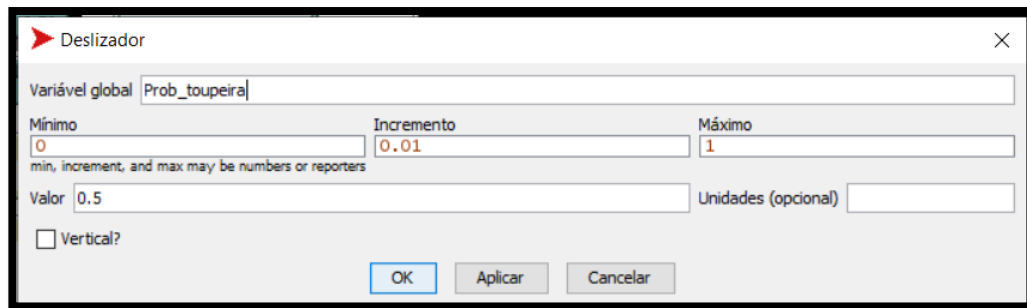


Figura18: Deslizador Prob_toupeira

Pergunta

8. Após a plantação inicial da relva no campo, o **planta-relva** é colocado no modo de **Manutenção**. Neste modo o **planta-relva** circula no campo a repor relva nas células danificadas podendo mover-se sempre em frente ou numa direção aleatória. Neste modo, o planta relvas pode passar por cima de células com relva.

Resposta

Como já foi referido anteriormente este modo já estava “criado”, apenas não estava definido.

O objetivo deste modo é manter a plantação de relva num nível estável sendo permitido ao “**planta-relva**” que passe por cima da relva, ao contrário dos outros modos (“**Frente**” e “**Serpentina**”).

- **Manutenção**

Este modo de plantação é executado através da função “**plantar_manutenção**”.

Para chegarmos a esta arrojada função “inteligente” foram precisas várias etapas de aperfeiçoamento. Idealmente esta função passa pelo campo de relva e consegue ter a visão do patch à sua frente assim como dos dois patches do seu lado direito e esquerdo (apenas usando “**patch-ahead 1**” como ficou explicito pelo professor).

O “**planta-relva**” começa por verificar se pode **plantar á direita** através do comando “**set heading heading + 90**”, neste caso se o patch da direita poder ser plantado, ele atualiza o “**contador_patch**” para 0 e chama a função “**avancar_plantar**”. Se não conseguir plantar, o “**planta-relva**”, gira a cabeça para a posição inicial (“**set heading heading – 90**”) e repete o mesmo procedimento para a **frente**. Se continuar a não

conseguir, o “*planta-relva*”, verifica o patch à sua **esquerda** girando a cabeça nessa direção (“*set heading heading – 90*”) e, novamente, se for possível avança e planta.

Por fim, se nenhum dos patches que ele verificou estiver disponível, ele gira a cabeça para a direita (“*set heading heading + 90*”), ficando assim virado para a frente e efetua uma das seguintes ações com base na variável global “**contador_patch**”.

Opção 1:

- Se $\text{Contador_patch} \neq \sqrt{\text{numero de patches}} + 3$
 1. Avança 1 patch
 2. Incrementa a variável **contador_patch**

Opção 2:

- Se $\text{Contador_patch} = \sqrt{\text{numero de patches}} + 3$
 1. Reseta a variável **contador_patch**.
 2. Vira á esquerda ou á direita dependo da flag = turn_flag
(0 – Direita) – (1 – Esquerda)

No caso de a verificação ser uma desigualdade, o “*planta-relva*” avança e incrementa o número do “**contador_patch**”. Mas se for verificada a igualdade, o contador é igualado a 0 e dependendo do valor da flag (“*turn_flag*”), 0 ou 1, o “*planta-relva*” vira à direita ou à esquerda, respetivamente.

Agora vai ser feita uma explicação um pouco mais detalhada sobre o funcionamento desta função, se não desejar ler e avançar para a próxima pergunta [carregue aqui](#)

O “segredo” desta verificação está no “+ 3” isto porque se fosse apenas o valor $\sqrt{\text{numero de patches}}$ ele estaria sempre a verificar os mesmos patches alterando apenas a direção como o fazia.

- E se utilizássemos “+ 1”, o que aconteceria?

Esta seria a solução ideal se o nosso “*planta-relva*” apenas tivesse a oportunidade de alterar o seu heading (“*set heading heading +- 90*”) quando fosse fazer a curva.

Porém, da maneira como temos o algoritmo, sendo que ele consegue virar antes de avançar esta maneira seria bastante *ineficiente* considerando as possibilidades e o potencial do algoritmo, visto que estaríamos a verificar patches já verificados.

- E se utilizássemos “+ 2”, o que aconteceria?

Esta verificação torna-se melhor que a anterior sendo que reduz a sobreposição de cobertura, ainda assim ainda há espaço para melhorias, considerando que o planta-relva tem uma visão de 3 patches (em frente, exatamente à direita e exatamente à esquerda) poderíamos aproveitar ainda melhor o algoritmo e reduzir as iterações pela mesma área de cobertura.

- E se utilizássemos “+ 3”, o que aconteceria?

Aconteceria que estaríamos a utilizar muito melhor a visão do planta-relva ao reduzir a sobreposição de patches que já foram verificados, aumentando a rapidez com que é feita a cobertura da mesma área dos exemplos anteriores.

Para demonstrar melhor isto que aqui foi dito, fizemos uns pequenos esboços onde desenhamos uma View de 8x8 e assinalamos a ordem onde era feita a mudança de direção e quantas vezes era preciso entrar na **Opção 2** para cobrir/verificar o campo na sua totalidade.

Sendo que na 1ª iteração, o “*planta-relva*” move-se da esquerda para a direita cobrindo/verificando 3 linhas e na segunda de baixo para cima cobrindo/verificando 3 colunas, temos os seguintes esboços.

							11ª	
						9ª	10ª	
					7ª	8ª		
				5ª	6ª			
			3ª	4ª				
	1ª	2ª						

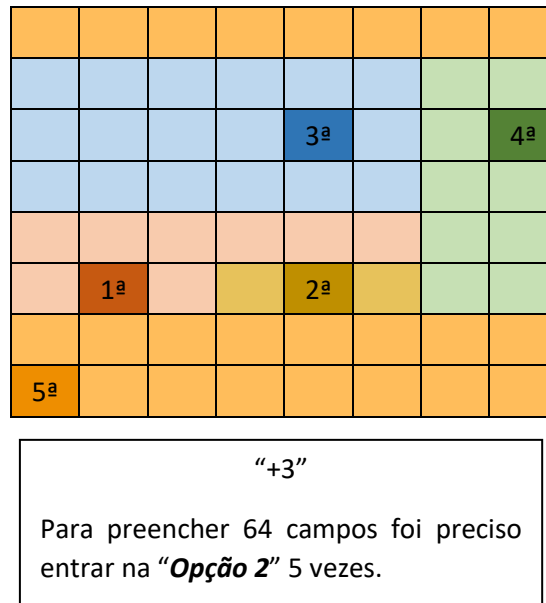
						5ª		6ª
				3ª		4ª		
	1ª		2ª					
7ª								

“+1”

Para preencher 64 campos foi preciso entrar na “**Opção 2**” 11 vezes.

“+2”

Para preencher 64 campos foi preciso entrar na “**Opção 2**” 7 vezes.



Considerando o mesmo ponto inicial em todas as situações, para uma mesma área podemos fazer uma aproximação do número de operações necessárias para serem verificados todos os patches.

A função “*andar_manutencao*” tem 10 operações se a verificação do contador_patch for false:

1. Virar à direita
2. Verifica
3. Virar à esquerda
4. Verifica
5. Vira à esquerda
6. Verifica
7. Vira à direita
8. *Verifica contador*
9. Avança
10. Incrementa contador

Na última execução da função, quando a verificação do contador for true, o programa faz mais 4 operações:

1. Igualar o contador a 0
2. Verifica turn_flag
3. Define o novo valor da flag
4. Vira (direita ou esquerda)

Número de operações por N_patches, no exemplo de uma View 8x8:

- “+1”
O código da função “*andar_manutencao*” é executado na totalidade 9 vezes (10 operações por vez) + 1 vez quando a verificação do contador for true (8 + 4 operações)
Por iteração temos um total de $90+12=102$ operações
Foram precisas 11 iterações $\Rightarrow 102 \times 11 = 1122$ operações
- “+2”
O código da função “*andar_manutencao*” é executado na totalidade 10 vezes (10 operações por vez) + 1 vez quando a verificação do contador for true (8 + 4 operações)
Por iteração temos um total de $100+12=112$ operações
Foram precisas 7 iterações $\Rightarrow 112 \times 7 = 784$ operações
- “+3”
O código da função “*andar_manutencao*” é executado na totalidade 11 vezes (10 operações por vez) + 1 vez quando a verificação do contador for true (8 + 4 operações)
Por iteração temos um total de $110+12=122$ operações
Foram precisas 5 iterações $\Rightarrow 122 \times 5 = 610$ operações

Pensamos que com base no apresentado ficou explicado, o porquê de nós termos usado o “+3” e não outro qualquer. Compreendemos e queremos que o leitor também compreenda que estes cálculos efetuados são apenas rascunhos e não podem ser usados como provas matemáticas para sustentar a nossa ideia. O nosso intuito foi apenas mostrar o nosso raciocínio e todo o processo de otimização que esta função sofreu.

Nestes cálculos não consideramos a existência de patches com relva por plantar pois o foco é testar qual a opção com maior área de cobertura e mais eficiente.

Segue-se uma figura com o código da função para os mais curiosos.

```

to plantar_manutencao
  set heading heading + 90
  ifelse [pcolor] of patch-ahead 1 = brown [set contador_patch 0 avançar_plantar]
  [
    set heading heading - 90

    ifelse [pcolor] of patch-ahead 1 = brown [set contador_patch 0 avançar_plantar]
    [
      set heading heading - 90
      ifelse [pcolor] of patch-ahead 1 = brown [set contador_patch 0 avançar_plantar]
      [
        set heading heading + 90
        ifelse contador_patch = sqrt(count patches) + 3
        [
          set contador_patch 0
          ifelse turn_flag = 0 [set turn_flag 1 right 90][set turn_flag 0 left 90]

        ]
      ]
      forward 1
      set contador_patch contador_patch + 1
    ]
  ]
end

```

Figura 19 Função andar_manutencao

Pergunta

9. Se o *planta-relva* chocar com uma toupeira na mesma célula, a toupeira morre.

Resposta

O objetivo deste procedimento é fazer com que a população das toupeiras cresça linearmente.

Para resolver esta etapa verificamos se o *planta-relva* possui toupeiras na mesma patch. Se isso acontecer todas as toupeiras da mesma morrem imediatamente.

```

to matar_toupeiras-here
  ask toupeiras-here
  [
    set Toupeiras_mortas Toupeiras_mortas + 1
    die
  ]
end

```

Figura 20: Função matar_toupeiras-here

Pergunta

10. Se duas **toupeiras** chocarem na mesma célula, nasce uma nova toupeira.

Resposta

Este procedimento ajudou a criar um sistema onde sem interferência humana (Inserir toupeiras, matar toupeiras, plantar relva, etc...) as toupeiras e o “*planta-relva*” conseguissem manter um ambiente onde ambos conseguissem coexistir. O “*planta-relva*” planta a relva para as “*toupeiras*” comerem e mata-as quando colide com elas e as toupeiras alimentam-se comendo a relva no patch onde se situam ganhando assim energia para se tornarem “adultas” (férteis), aptas para se reproduzirem.

```
ask toupeiras[
  check-energy
  if Prob_toupeira > (random-float 1)[
    andar_toupeira
    if count toupeiras-here with [energy >= Adult_energy] >= 2
      [ask toupeiras-here[hatch_toupeira]]
  ]
]
```

Figura 21: Vida da toupeira

```
to hatch_toupeira
  if energy >= Adult_energy
  [
    if Prob_hatch > random-float 1
    [
      ask toupeiras-here [set energy Adult_energy / 2]
      set color blue
      set nascimentos nascimentos + 1
      hatch-toupeiras 1
      [
        set color blue
        set energy (Adult_energy / 2 + random 30)
        set heading random 360
      ]
    ]
  ]
end
```

Figura22: Função hatch_toupeira

Pergunta

11. Insere os gráficos que ache apropriados.

Resposta

Achamos necessário criar quatro gráficos e nove monitores para conseguirmos analisar diversas variáveis e analisar os dados do sistema mais facilmente.

A utilização de gráficos deve-se ao facto de facilitar a comparação entre dados do sistema, por exemplo, o número de toupeiras com o passar do tempo(ticks).

Gráficos:

População das toupeiras

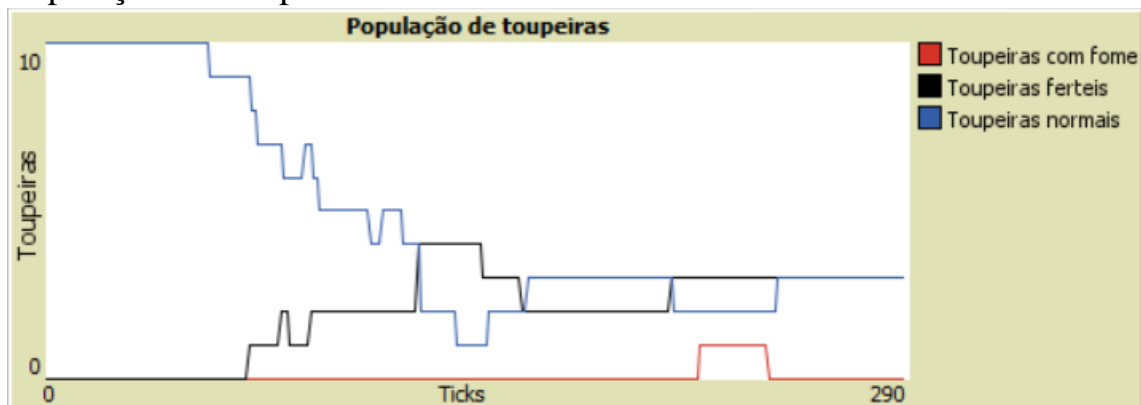


Gráfico 1 - População das toupeiras

Neste gráfico é nos fornecido a informação sobre os diversos tipos de toupeiras e a sua quantidade.

Comandos

- **Toupeiras com fome** - plot count Toupeiras with [energy < 50]
- **Toupeiras férteis** - plot count toupeiras with [energy > Adult_energy]
- **Toupeiras normais** - plot count toupeiras with [color = blue]

Toupeiras comidas e procriadas

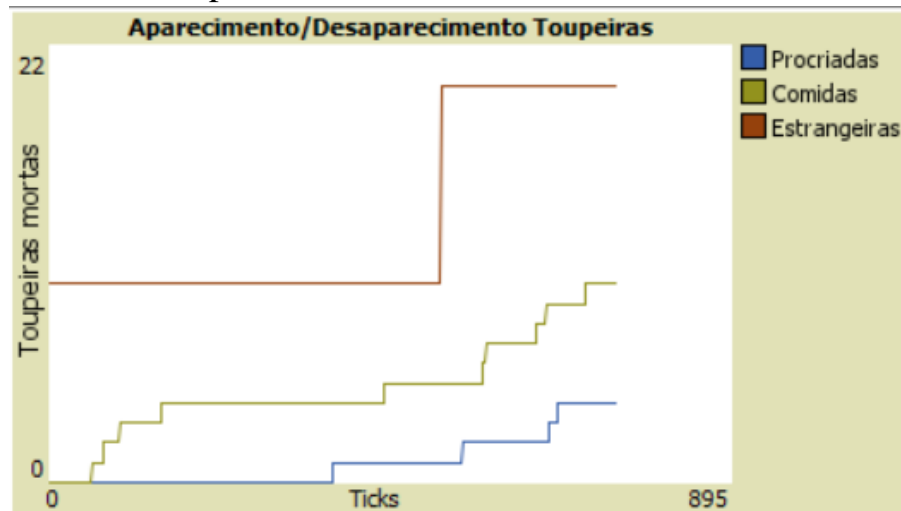


Gráfico 2 - Aparecimento/Desaparecimento Toupeiras

Neste gráfico temos a ideia de quantas toupeiras foram mortas pela “*planta-relva*” (**Comidas**), quantas toupeiras foram procriadas através da colisão (**Procriadas**) entre toupeiras e o número de toupeiras introduzidas pelo utilizador (**Estrangeiras**).

Comandos

- **Procriadas** - plot nascimentos
- **Comidas** - plot Toupeiras_mortas
- **Estrangeiras** - plot inseridas

Qualidade da relva



Gráfico 3 - Qualidade da relva

Neste gráfico é visível a **quantidade de relva** que o “*planta-relva*” plantou desde o início da aplicação (função setup), sendo o máximo de relva o número de patches da View (neste caso $33 \times 33 = 1089$).

Comandos

- **Máximo de relva** - plot count patches
- **Relva** – plot N_relva

Toupeiras com fome

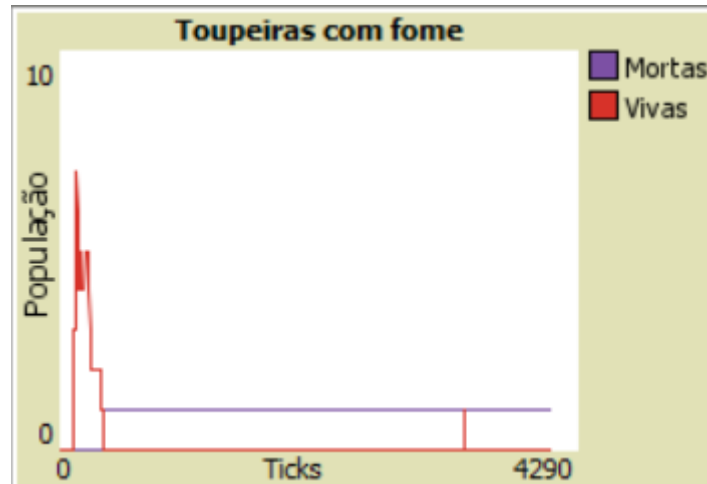


Gráfico 4 - Toupeiras com fome

Neste gráfico é apresentada as toupeiras que morreram com fome e as toupeiras que estão esfomeadas.

Comandos

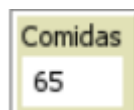
- **Mortas**- plot contador_starved
- **Vivas** – plot count toupeiras with [energy < 50]

A utilização de monitores é adequada quanto queremos saber o valor atual de alguma variável (dinâmica ou estática) de forma bastante evidente.

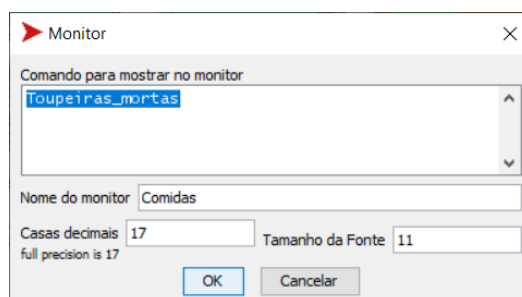
Monitores:

Toupeiras comidas

Neste monitor é visível a quantidade de toupeiras que até ao momento foram comidas pelo “*planta-relva*”.

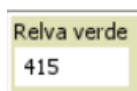


Monitor 1 - Comidas

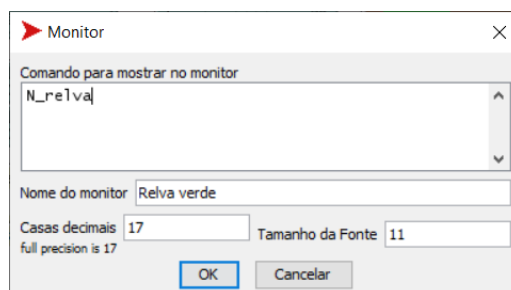


Relva verde

Apesar de já existir um gráfico que representa quanta relva o “*planta-relva*” já plantou até ao momento, este monitor ajuda na visualização do número de Relva plantada no campo.

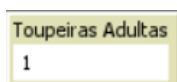


Monitor 2 - Relva verde

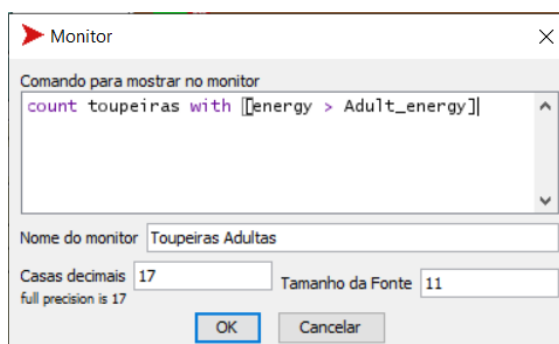
A screenshot of a "Monitor" configuration dialog box. It has a title bar with a red arrow icon and a close button. The "Comando para mostrar no monitor" field contains the text "N_relva". The "Nome do monitor" field contains "Relva verde". The "Casas decimais" field is set to "17" (with a note "full precision is 17") and the "Tamanho da Fonte" field is set to "11". There are "OK" and "Cancelar" buttons at the bottom.

Toupeiras Adultas

Para manter registo do número de toupeiras adultas/férteis, criamos este monitor que apresenta o número de toupeiras adultas no campo.

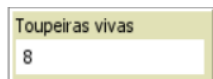


Monitor 3 - Toupeiras Adultas

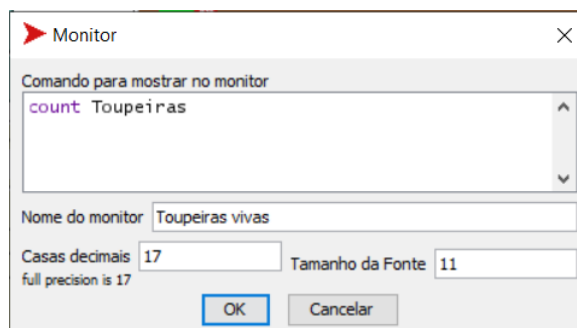
A screenshot of a "Monitor" configuration dialog box. The "Comando para mostrar no monitor" field contains the text "count toupeiras with [energy > Adult_energy]". The "Nome do monitor" field contains "Toupeiras Adultas". The "Casas decimais" field is set to "17" (with a note "full precision is 17") and the "Tamanho da Fonte" field is set to "11". There are "OK" and "Cancelar" buttons at the bottom.

Toupeiras vivas

Este monitor aponta para o número de toupeiras vivas em campo. Este monitor foi dos mais uteis para controlar o tamanho da população das toupeiras.



Monitor 4 - Toupeiras vivas

A screenshot of a "Monitor" configuration dialog box. The "Comando para mostrar no monitor" field contains the text "count Toupeiras". The "Nome do monitor" field contains "Toupeiras vivas". The "Casas decimais" field is set to "17" (with a note "full precision is 17") and the "Tamanho da Fonte" field is set to "11". There are "OK" and "Cancelar" buttons at the bottom.

Movimentos Planta Relva

De modo a testar a eficiência do planta-relvas, por exemplo com o número de movimentos que executa e a quantidade de relva plantada, implementamos este contador para facilitar na visualização do número de movimentos.

Movimentos Planta relva
110

Monitor 5 - Movimentos Planta relva

Monitor

Comando para mostrar no monitor
cont_movs|

Nome do monitor Movimentos Planta relva

Casas decimais 17 full precision is 17 Tamanho da Fonte 11

OK Cancelar

Morta por Fome

Apesar de já se saber quantas toupeiras morreram através da colisão com o planta-relvas, não se consegue saber quantas toupeiras morreram à fome, visto que nesta aplicação as toupeiras podem morrer à fome se o **GetEnergyOnlyFromFood** estiver “On”.

Mortas por fome
0

Monitor 6 - Mortas por Fome

Monitor

Comando para mostrar no monitor
contador_starved|

Nome do monitor Mortas por fome

Casas decimais 17 full precision is 17 Tamanho da Fonte 11

OK Cancelar

Toupeiras com Fome

Para finalizar, este monitor foi o último monitor que achamos necessário implementar e representa o número atual de toupeiras com fome.

Toupeiras com fome
1

Monitor 7 - Toupeiras com fome

Monitor

Comando para mostrar no monitor
count toupeiras with [[energy < 50]]

Nome do monitor Toupeiras com fome

Casas decimais 17 full precision is 17 Tamanho da Fonte 11

OK Cancelar

Pergunta

12) Proponha algumas inovações/variantes que melhorem o comportamento do modelo.

Resposta

Para responder a alguns problemas que encontramos ao desenvolver inicialmente o projeto, decidimos adicionar funcionalidades e condicionantes extra.

Principais problemas encontrados:

1. Nascimento exponencial das toupeiras
2. As toupeiras apenas desaparecerem se forem comidas
3. Dificuldade do “*planta-relva*” em manter o campo relvado
4. O “*planta-relva*” ficar sozinho depois de matar todas as toupeiras

Começamos por adicionar um sistema de *energia* para as *toupeiras*, impedindo que as mesmas se conseguissem reproduzir imediatamente depois de nascerem. Juntamente com a necessidade de energia para se reproduzirem, adicionamos também a necessidade de energia para sobreviverem. Com estas duas condicionantes conseguimos controlar o nascimento abrupto de *toupeiras* num curto espaço de tempo assim como impedir que as mesmas apenas morressem através do “*planta-relva*”.

A única forma das *toupeiras* ganharem energia é comendo patches com relva plantada.

Quando duas toupeiras férteis (com energia suficiente para procriarem) colidirem, nasce uma terceira toupeira e a energia das toupeiras férteis é reduzida para metade.

Por cada movimento que uma toupeira faça perde energia. Quando a energia for ≤ 0 a toupeira morre.

```
ask toupeiras[
  check-energy

  if Prob_toupeira > (random-float 1)[
    andar_toupeira

    if count toupeiras-here with [energy >= Adult_energy] >= 2 [ask toupeiras-here[hatch_toupeira]]
  ]
]
```

Figura 23: ask toupeiras na função to go

```
to check-energy
  if GetEnergyOnlyFromFood = false[increment-energy]

  (ifelse
    energy < 0 [set contador_starved contador_starved + 1 die]
    energy <= 50 [set color red]
    energy > 50 and energy < Adult_energy [set color blue]
    energy >= Adult_energy [set color black]
  )
end
```

Figura 24: Função check-energy para atualizar as toupeiras

```
to andar_toupeira
  ifelse [pcolor] of patch-ahead 1 = lime [avançar_toupeira]
  [
    ifelse energy < Adult_energy ; Se a toupeira não for adulta
    [
      set heading heading + 90 ;Virar à direita
      ifelse [pcolor] of patch-ahead 1 = lime [avançar_toupeira]
      [
        set heading heading - 90 ;Vira para a frente
        ifelse [pcolor] of patch-ahead 1 = lime [avançar_toupeira]
        [
          set heading heading - 90 ;Vira à esquerda
          ifelse [pcolor] of patch-ahead 1 = lime [avançar_toupeira]
          [; Se não encontrar relva para comer, vira para a frente e anda
            set heading heading + 90
            avançar_toupeira
          ]
        ]
      ]
    ]
  ]
  [avançar_toupeira]
end
```

Figura 25: Função para as toupeiras procurarem relva

Acrescentamos também um modo automático para manter a View com ação e para adicionar dinamismo e realismo ao ambiente, na medida em podem aparecer toupeiras sem qualquer ligação às presentes no ambiente em busca de relva para comer.

Para desenvolvermos este modo automático definimos a “**Prob_praga**” escolhida pelo utilizador e um interruptor para ligar ou desligar este modo. A atuação no sistema é feita através da função “**update_board_afk**” que compara a “**Prob_praga**” com um “**random-float**” de 0 a 40 (porque de 0 a 1 era quase instantâneo) e se a probabilidade de aparecer uma praga for superior ao random-float gerado, então vamos fazer uma verificação da qualidade da relva para sabermos se faz sentido ou não, aparecerem toupeiras no sistema. Se não existir relva em alguma abundância, as toupeiras não aparecem, porém, se existir relva suficiente, é selecionado o número de toupeiras, através de probabilidades condicionais e são inseridas as toupeiras no sistema.

A verificação usada para fazer o controlo da qualidade da relva foi a seguinte:

```
if N_relva > count_patches / (1.5 + (0.5 * Prob_praga))[
```

Figura 26: Condição que verifica a qualidade da relva

Deste modo conseguimos limitar o aparecimento de praga para apenas quando houver um número significativo de relva. No caso da probabilidade de aparecer praga ser igual a 1, faz com que apenas sejam inseridas toupeiras se o N_relva for superior a metade da relva total possível de plantar.

Resumidamente, quanto maior for a probabilidade de aparecer praga, menor é o número de relva necessária.

Com a adição destas funcionalidades conseguimos controlar o crescimento das toupeiras assim como equilibrar todo o ecossistema “**planta-relva**”-“**toupeiras**”.

Inserimos também a possibilidade do utilizador, plantar ou comer relva clicando na View com o botão do lado esquerdo do rato para aumentar o número de possibilidades para testar o sistema.

Conclusão

Este trabalho foi o primeiro trabalho prático na UC de Inteligência Artificial e consiste em contruir uma aplicação com dois agentes, o planta-relva que planta relva no sítio onde se encontra e que mata as toupeiras na patch atual e as toupeiras que destroem a relva e procriam quando duas toupeiras se cruzam.

Em suma, achamos um trabalho importante e uma ótima introdução à parte pratica da UC de Inteligência Artificial uma vez que esta é uma aplicação bastante dinâmica e que cativa ao seu desenvolvimento. Foi também uma maneira de aplicar conhecimentos aprendidos nas aulas teóricas como por exemplo: O que é agente? O que o agente realiza? Perceção, decisão, ação.

Referências

- [1] NetLogo User Manual version 6.2.1 October 20, 2021,
<https://ccl.northwestern.edu/netlogo/docs/> acedido em 10-11-2021
- [2] Modelo para o Relatório dos Trabalhos – José Paulo Barroso de Moura Oliveira
- [3] Agentes Inteligentes - José Paulo Barroso de Moura Oliveira
- [4] Protocolo do Trabalho 1 - José Paulo Barroso de Moura Oliveira
- [5] NetLogo Dictionary, <http://ccl.northwestern.edu/netlogo/docs/dictionary.html#ifelse> acedido em 1-11-2021
- [6] NetLogo Dictionary, <https://ccl.northwestern.edu/netlogo/docs/rnd.html> acedido em 9-11-2021