

Trabalho prático I

Inteligência Artificial

Discente:

Diogo Sousa AL73928
Francisco Gouveia AL74044

Docente

Professor José Paulo
Barroso de Moura Oliveira

Resumo

No âmbito da disciplina de Inteligência Artificial, foi realizada a etapa I do trabalho prático relacionado com o projeto de desenvolvimento de um aquário. Para este efeito, foi utilizada a ferramenta NetLogo, onde se realizou o executável requerido pelo docente, e o processador de texto, Microsoft Word 2021, para a realização do relatório de trabalho.

Este trabalho é constituído por três pilares basais, essenciais para a compreensão do trabalho. O primeiro referente à conversação e explicação entre os dois membros do grupo. Aqui através de diversas reuniões, sendo as primeiras dedicadas a brainstorming, discutiu-se o que se deveria implementar no programa, e quais seriam os métodos de trabalho e planeamento mais corretos a utilizar para este propósito. No segundo passo, o grupo separou-se em funções diferentes. Aqui há alguma divisão de tarefas, principalmente de funcionalidades, onde cada um dos membros do projeto se ocupa de diversas tarefas. Mais tarde, ainda nessa etapa, juntou-se às funcionalidades no projeto atual, e começou-se a procura por erros e melhorias que poderiam ser adicionadas. Na terceira e última etapa, existe a contextualização e redação do relatório referente, portanto, à etapa atual, pelos dois membros do grupo.

Procuramos assim explicitar e contextualizar o recetor de todos os diversos conceitos, funcionalidades e métodos. Para isso recorremos a figuras e diagramas da nossa autoria, para melhor compreensão.

Índice

Resumo	2
Introdução	6
Objetivos da Etapa.....	7
Parte 1	7
Introdução:.....	7
Desenvolvimento:	8
Parte 2.....	17
Introdução:.....	17
Desenvolvimento:	17
Conclusão e Notas Finais.....	24
Bibliografia	25
ANEXO.....	26

Índice Figuras

<i>Figura 1 - Captura de ecrã do modelo do aquário</i>	<i>7</i>
<i>Figura 2 - Captura de ecrã do aquário para visualizar as espécies dos peixes</i>	<i>8</i>
<i>Figura 3-Código usado para a criação das duas espécies</i>	<i>8</i>
<i>Figura 4-Captura de ecrã do aquário para visualizar a cor e os limites.....</i>	<i>9</i>
<i>Figura 5 - Excerto de código que mostra a água e os limites do aquário.....</i>	<i>9</i>
<i>Figura 6 - Excerto de código sobre a movimentação dos peixes</i>	<i>9</i>
<i>Figura 7 - Excerto do código da construção do obstáculo.....</i>	<i>10</i>
<i>Figura 7.1 – Visualização do Obstáculo.....</i>	<i>11</i>
<i>Figura 8 - Capturas de ecrã que representam como funciona a sujidade do aquário... </i>	<i>11</i>
<i>Figura 9 - Excerto de código que representa como funciona a sujidade no aquário</i>	<i>12</i>
<i>Figura 10 - Captura de ecrã que mostra os dois botões</i>	<i>12</i>
<i>Figura 11 - Deslizadores que geram o número dos peixes de cada espécie.....</i>	<i>13</i>
<i>Figura 12 - Captura de ecrã de como funciona o gráfico que mostra a quantidade de peixes de cada espécie</i>	<i>13</i>
<i>Figura 13 - Captura de ecrã quando o botão "Feed" é acionado e o excerto do código que cria a comida</i>	<i>14</i>
<i>Figura 14 - Captura de ecrã de comportamento da comida.....</i>	<i>14</i>
<i>Figura 15- Excerto do código que mostra o movimento da comida</i>	<i>15</i>
<i>Figura 16- Excerto de código que mostra como os peixes comem a comida</i>	<i>15</i>
<i>Figura 17- Captura de ecrã que mostra a idade dos peixes</i>	<i>16</i>
<i>Figura 18- Excerto de código que mostra como foi feita a idade dos peixes</i>	<i>16</i>
<i>Figura 19- Excerto de código que mostra como aumenta a idade dos peixes e como eles falecem</i>	<i>16</i>
<i>Figura 20- Excerto de Código, com a variável com terminação _num.....</i>	<i>17</i>
<i>Figura 21- Excerto de Código com ambos os botões para adicionar peixes de forma singular.....</i>	<i>17</i>
<i>Figura 22 -Excerto de código que tem como função a limpeza do aquário</i>	<i>17</i>
<i>Figura 23 – Excerto de código que limpa a comida perdida no aquário.....</i>	<i>18</i>
<i>Figura 24 – Excerto de código que faz a reprodução dos peixes amarelos</i>	<i>18</i>
<i>Figura 25 – Excerto de código que mostra que quando o peixe come recebe energia nos peixes vermelhos</i>	<i>18</i>
<i>Figura 26 – Excerto de código que permite escolher a quantidade de comida.....</i>	<i>19</i>
<i>Figura 27 –Excerto de código que mostra que quando o peixe come recebe energia nos peixes amarelos</i>	<i>19</i>
<i>Figura 28 – Duas linhas de código que permitem desenhar caminho onde os peixes passam.</i>	<i>19</i>

<i>Figura 29- Captura do ecrã que mostra o aquário com a resistência elétrica e um deslizador que permitiu alterar a temperatura para 79°</i>	20
<i>Figura 30- Captura do ecrã que mostra o aquário com a resistência elétrica e um deslizador que permitiu alterar a temperatura para 37°</i>	20
<i>Figura 31 - Mostra a variação da temperatura.....</i>	20
<i>Figura 32- Excerto de código que permitiu criar a resistência elétrica e trocar a sua cor consoante a temperatura</i>	20
<i>Figura 33- Excerto de código que cria a função "graus"</i>	20
<i>Figura 34 -Linha de código que permite alterar a opção de reprodução ativa ou não.</i>	21
<i>Figura 35 – Excerto de código que faz com que os peixes morrem</i>	22
<i>Figura 36 – Definições do gráfico que permite mostrar a evolução da sujidade</i>	22
<i>Figura 37 – Excerto de código que mostra a energia ou idade em uma etiqueta no peixe</i>	22
<i>Figura 38- Linha de código que tem como objetivo a possibilidade de doenças nos peixes.....</i>	23

Introdução

É verdade que não há definição consensual sobre o conceito de Inteligência Artificial. Até podemos ir mais longe e afirmar com grande causalidade que dentro da Engenharia Informática a inteligência artificial deve ser dos ramos, que, ao mesmo tempo, é tão importante quanto a sua incompreensão. Como disse o grande teórico americano, Roger Schank: “Mesmo os praticantes de IA estão de certo modo confusos em relação ao que Inteligência Artificial é realmente.”. Isto, na nossa opinião, deve-se, em grande parte, ao facto de que o conceito de Inteligência Artificial é acompanhado de dois conceitos muito complexos e trabalhosos de se definir: Inteligência, e Artificial.

Podemos afirmar que Inteligência, tem sido ao longo da nossa história definida de formas diferentes. Esta depende imenso das doutrinas a que se rege o autor que se atreve a definir inteligência. No entanto, de uma forma comum e vulgar, podemos dizer que se classifica como a capacidade de alguém/algo para lógica, abstração, compreensão, autoconhecimento e raciocínio, conceitos estes que estão também muito presentes em inteligência artificial, como iremos mais a diante ver.

Recorrendo então à definição de inteligência, e munindo-nos de diversas obras redigidas pelos teóricos e engenheiros da área, podemos chegar a algo próximo de definição, pelos termos convencionais: A Inteligência Artificial é a capacidade de dispositivos eletrônicos operarem com uma lógica que remete ao raciocínio. Ou seja, dispositivos criados pelo homem que possam desempenhar determinadas funções sem a interferência humana. Este conceito é até evidenciado no ecossistema que criamos, para esta presente etapa.

Neste trabalho propomos assim, para além do que o Docente requer, tentar compreender melhor estes conceitos complexos, de Inteligência Artificial, através deste trabalho.

Objetivos da Etapa

Parte 1

Introdução:

Neste primeiro trabalho prático foi proposto a realização de um modelo *NetLogo* que permita simular um ecossistema simplificado, onde os agentes, são representados por peixes dentro de um habitat – neste caso um aquário. Dentro deste habitat, certas condições têm de ser cumpridas, nomeadamente: os peixes comerem, deixarem lixo, e o habitat ir-se degradando. Para além disso os peixes não se podem mover livremente e necessitam de estar encapsulados dentro dos limites da janela de visualização.

Este trabalho foi dividido em duas fases. Numa primeira instância vamos mostrar e legendar todas as funcionalidades que a nossa simulação possui.

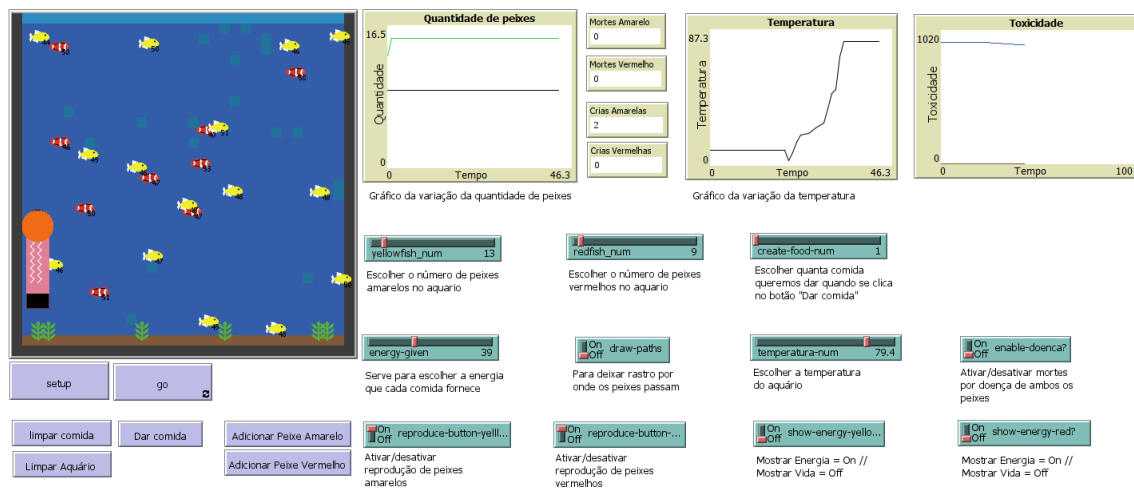


Figura 1 - Captura de ecrã do modelo do aquário

Desenvolvimento:

Como já referido em anteriormente, a primeira fase tem vários pontos na elaboração do projeto. Um dos primeiros pontos é a criação de espécies de “*peixes-age*”-variável interna referente à idade dos peixes.

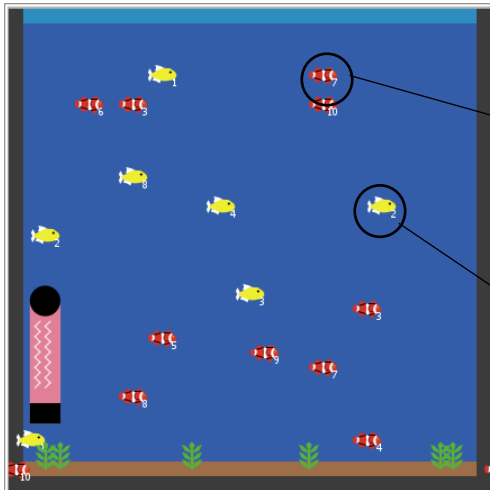


Figura 3 - Captura de ecrã do aquário para visualizar as espécies dos peixes

```
breed [redfish a-fish]
breed [yellowfish b-fish]

create-redfish redfish_num
[
  setxy random-pxcor random-pycor
  set color red
  set size 2
  set shape "fish 3"
  set age-redfish 1 + random 10 ;random idade até aos 10
  set label age-redfish
]

create-yellowfish yellowfish_num
[
  setxy random-pxcor random-pycor
  set color yellow
  set size 2
  set shape "fish"
  set age-yellowfish 1 + random 10 ;random idade até aos 10
  set label age-yellowfish
]
```

Figura 2-Código usado para a criação das duas espécies

Começamos por criar as duas raças (*breed*) que designamos com o nome de *redfish* e *yellowfish*. Este excerto de código que podemos visualizar em cima está ligado à ação do botão *setup*. Atribuímos nele as diferentes definições dos peixes: duas cores diferentes (*set color*), tamanho das duas raças (*set size 2*) e as texturas dos peixes (*set shape "fish"* ou "*fish 3*"). O resto do código por enquanto não é relevante e irá ser explicado quando for necessário, mais à frente no relatório.

Outro ponto fundamental requerido, para a nossa simulação, é o habitat ter um aspeto de aquário. Desta forma a água, que no nosso caso representa todas as *patches* de *background*, deve ter uma cor azulada. Optamos também por fazer um limite à volta do modelo sem ter o topo tapado e no fundo do aquário ter algo que aparenta ser uma camada de terra.

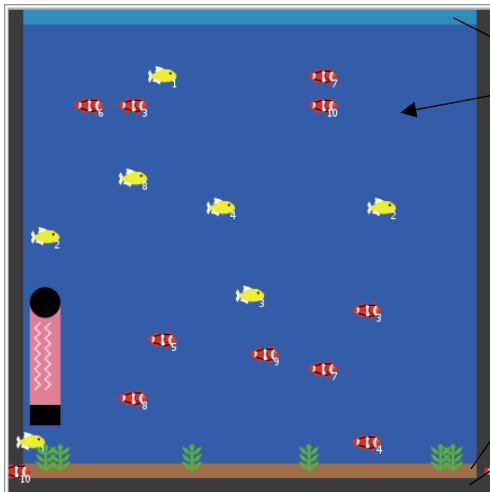


Figura 4-Captura de ecrã do aquário para visualizar a cor e os limites

```
to draw-aquarium
  ask patches [set pcolor blue]

  set top patches with [pycor = max-pycor]
  ask top [set pcolor 95]

  set ground-real patches with [pycor = -15]
  ask ground-real [set pcolor brown]

  set aquarium patches with
  [
    pxcor = max-pxcor or
    pxcor = min-pxcor or
    pycor = min-pycor
  ]
  ask aquarium [set pcolor 2]
end
```

Figura 5 - Excerto de código que mostra a água e os limites do aquário

Nomeamos a função de “draw-aquarium”. Na primeira linha de código referimo-nos a todos os *patches*, e definimos a cor de fundo de azul – esta representa a água. Definimos posteriormente a cor do topo, com a ajuda da variável global *top*. Aqui colocamos a cor mais clara, e usando só os *patches* do topo, através do comando *pycor = maxpycor*. Depois disto, utilizando a variável global, *ground-real*, fazemos o chão, com aspeto de terra. Utilizamos coordenadas, pois deve ser ligeiramente acima das bordas que colocamos para o desenho do aquário em si. O passo seguinte foi desenhar as fronteiras do aquário. Utilizando a relação entre *pxcor*, *pycor*, *max-pxcor*, *max-pycor* e os mínimos relativos fizemos o desenho. De notar que não desenhámos o topo, de forma a simular um aquário na vida real que não está preenchido no seu máximo.

Como já temos o aquário feito, só falta a movimentação dos peixes. A função *move* é talvez a mais complexa desta simulação. Isto deve-se ao facto de tentarmos executar com bastante precisão o movimento de um peixe na vida real.

```
to move [ dist ]
  let turn-angle (random-float 60) - 30

  ; if world does not wrap target patch could be nobody
  let target-patch patch-right-and-ahead turn-angle dist

  ifelse target-patch != nobody and not member? target-patch obstacle
  [ rt turn-angle ]
  [ rt turn-angle - 180 ]

  ; turtle could still wind up outside region, so then don't move
  set target-patch patch-right-and-ahead 0 dist
  if target-patch != nobody and not member? target-patch obstacle
  [ fd dist ]
end
```

Figura 6 - Excerto de código sobre a movimentação dos peixes

A função tem como parâmetro uma distância. Essa distância, no nosso caso, tem um valor *default* de um. São definidas as seguintes variáveis:

1. *Turn Angle*: refere-se a um *random float* entre 0 e 60, que é subtraído por 30.

No nosso caso imaginemos que estes valores são graus.

2. *Target Patch*: esta variável vai armazenar o patch que está a um certo "*turn-angle*" e à distância de um, ou seja, o turtle atual vai olhar na direção de um angulo que foi aleatoriamente escolhido no ponto um. Após isto, irá verificar se, a um *patch* de distância, se encontra lá algo.

No primeiro bloco *IfElse*, se o *patch* que definimos antes, *target-patch*, não for alguém (ou seja, ninguém, falando de forma vulgar) e se não for um membro do obstáculo, então ele irá fazer um *right-turn* (*rt* no código) com o ângulo que foi aleatoriamente gerado anteriormente no ponto um. Se isso não acontecer, ou seja: se for alguém ou um obstáculo, então ele irá fazer o *turn-right* só que com uma subtração de 180 graus. Ou seja, vira no sentido oposto do primeiramente definido. O segundo bloco *IfElse*, serve para no caso de existir algum peixe que dê *spawn* fora do obstáculo, assim que se detete uma border, então irá ficar parado nessa zona, não interferindo com nada que seja dentro do habitáculo.

Por fim, agora na Figura 7, é criado um obstáculo. Criou-se essa funcionalidade para que quando o peixe batesse nos limites do aquário, existisse uma entidade padronizada para podermos efetuar o controlo de comportamento após esse evento.

```
to draw-obstacle
  set obstacle patches with
  [
    pxcor = max-pxcor or
    pxcor = min-pxcor or
    pycor = 16 or
    pycor = min-pycor
  ]
  ask obstacle [set pcolor red]
  ask n-of num-turtles patches with [pcolor != red] [ sprout 1 [ set color red set heading random 360 ]]
end
```

Figura 7 - Excerto do código da construção do obstáculo

Como podemos ver aqui, a função é bastante simples, sendo que desenha uma fronteira quadrada vermelha. Esta tem lugar debaixo da fronteira preta que se vê na simulação. Escolhemos a cor vermelha para conseguirmos ver bem a sua construção do início do *setup procedure*, e conseguir observar melhor o controlo dela.



Figura 7.1 – Visualização do Obstáculo

Como todos os aquários ficam sujos, este não é exceção. Outro ponto fundamental neste trabalho foi a criação de sujidade, como mostra a figura abaixo:

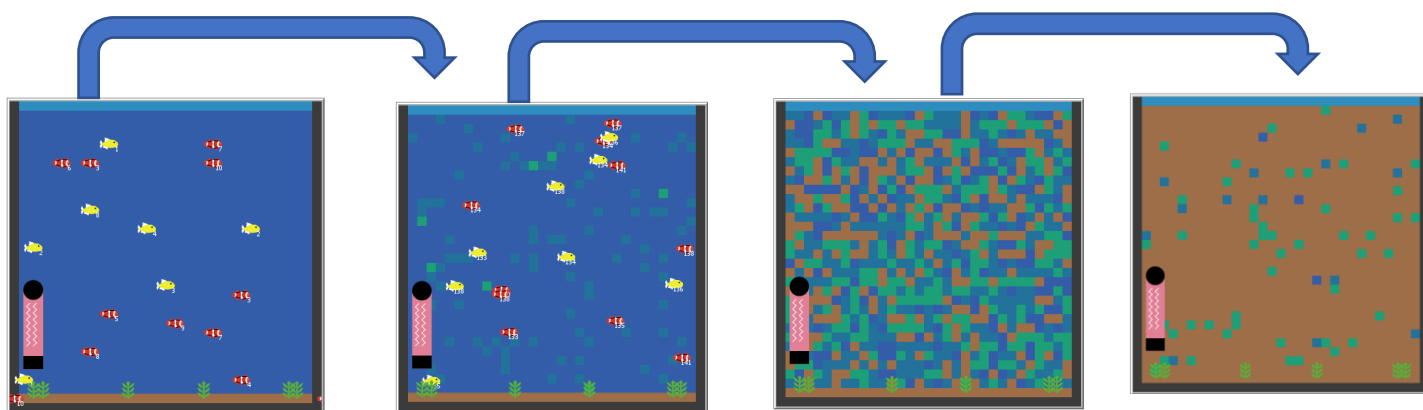


Figura 8 - Capturas de ecrã que representam como funciona a sujidade do aquário

Nesta etapa, a nossa ideia foi alterar a cor perante o aumento dos ticks que faz analogia à sujidade “ao longo do tempo”. No excerto de código criamos uma função chamada *toxicity*, que tem como objetivo uma condição que pergunta a todos os *patches* e compara se a cor que está presente é o azul. Se for verdade, troca a cor (*set pcolor 94*) e assim sucessivamente com o resto das outras cores, até todas os *patches* ficarem castanhos.

```
to toxicity
  ask one-of patches
  [
    if pcolor = blue
    [
      set pcolor 94
    ]
  ]
  ask one-of patches
  [
    if pcolor = 94
    [
      set pcolor 75
    ]
  ]
  ask one-of patches
  [
    if pcolor = 75
    [
      set pcolor brown
    ]
  ]
end
```

Figura 9 - Excerto de código que representa como funciona a sujidade no aquário

Mais um dos pontos do projeto foi a criação de botões, deslizadores e gráficos.

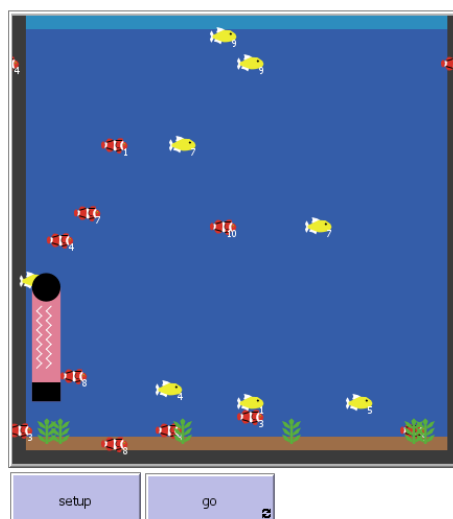


Figura 10 - Captura de ecrã que mostra os dois botões

Os primeiros botões que criamos foi o *Setup* e o *Go*. O botão *Setup* tem como objetivo limpar o ambiente, colocar a cor escolhida, que foi o azul em todas as células, criar e introduzir no mundo os agentes (como se evidencia na figura 3), etc. De uma forma resumida, todas as funções que o botão faz foram referidas em cima. Alongando o assunto, dizemos que o botão faz as *setup procedures*. O botão *Go* quando pressionado faz com que os peixes circulem no aquário de forma aleatória usando a função do movimento que já foi referida em cima (figura 6).



Figura 11 - Deslizadores que geram o número dos peixes de cada espécie

Criamos dois deslizadores que permitem definir o número de peixes de cada espécie. A implementação desta funcionalidade no código acaba por ser fácil. Bastou colocar o nome do deslizador na frente do “*create-readfish*” como designado na figura 3.

Foi feito um gráfico que permite visualizar a evolução das populações de peixes.

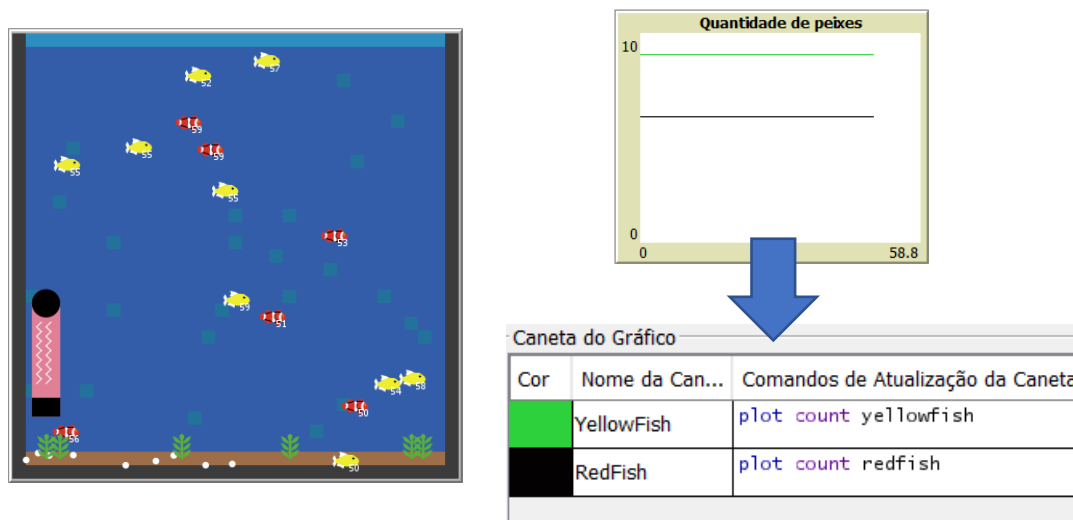


Figura 12 - Captura de ecrã de como funciona o gráfico que mostra a quantidade de peixes de cada espécie

A criação do gráfico é um processo simples. Colocamos o “*plot count*” dos peixes amarelos e dos vermelhos e através disso ele contabiliza todos os peixes. Coloca os peixes amarelos como linha verde no gráfico enquanto os peixes vermelhos ficam com a linha preta, gerando assim a contagem pretendida.

Como último ponto, criamos um botão chamado “Feed” que serve para alimentar os peixes.

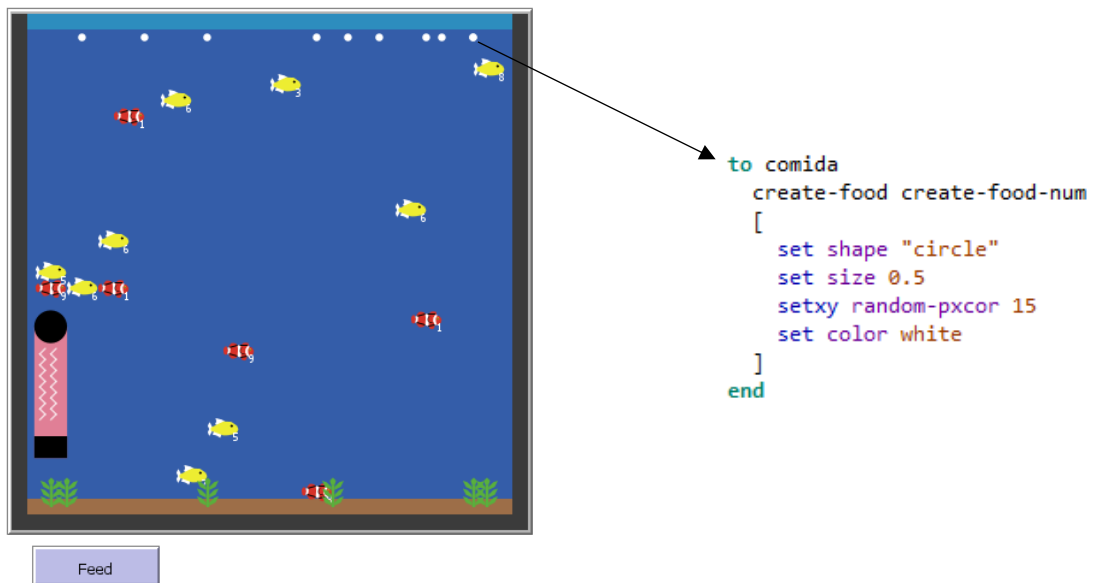


Figura 13 - Captura de ecrã quando o botão "Feed" é acionado e o excerto do código que cria a comida

O excerto de código acima cria a comida como um agente branco, redondo e em lugares aleatoriamente gerados, mas apenas com o mesmo ponto no eixo das ordenadas. Quando o botão é acionado deposita uma quantidade de agentes *comida* na parte superior do aquário.

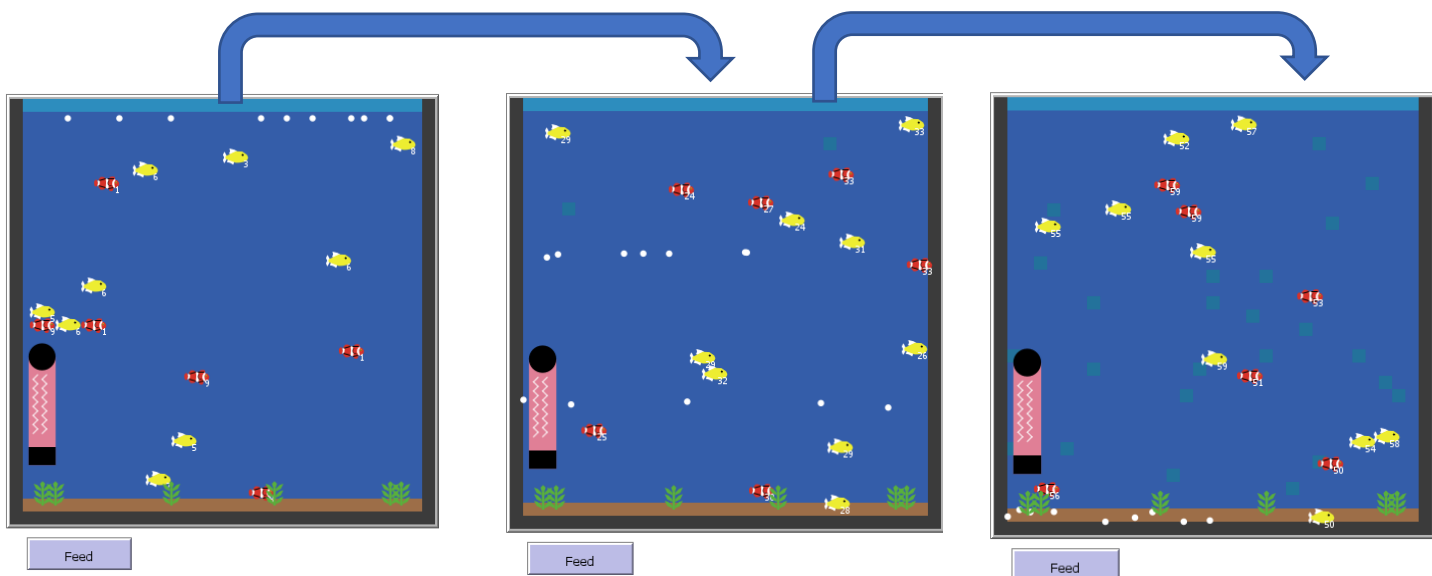


Figura 14 - Captura de ecrã de comportamento da comida

A comida deposita-se gradualmente em direção ao fundo do aquário e os peixes podem deslocar-se em direção à superfície para se alimentarem como ilustram as figuras em cima, ou aproveitar a comida que permanece no fundo do aquário.

O movimento da comida é do tipo “zig-zag” na vertical, e foi elaborada com a seguinte função

```
to move-food [dist ]
  set heading 180
  let turn-angle (random-float 60) - 30

  ; if world does not wrap target patch could be nobody
  let target-patch patch-right-and-ahead turn-angle dist

  ifelse target-patch != nobody and not member? target-patch ground
  [ rt turn-angle]
  [ rt turn-angle - 0 ]

  ; turtle could still wind up outside region, so then don't move
  set target-patch patch-right-and-ahead 0 dist
  if target-patch != nobody and not member? target-patch ground
  [ fd dist ]
end
```

Figura 15- Excerto do código que mostra o movimento da comida

A função “move-food” funciona de forma igual ao movimento dos peixes que foi referido na figura 6. A única diferença é que nos blocos *ElseIf* referimo-nos ao obstáculo como *ground*, já que é uma localização diferente. Para além disso, se bater, em vez de ir no sentido oposto, fica no mesmo. Isto tem o propósito de quando a comida chega ao fundo, permanece lá.

E assim sendo, criamos uma função que está situada no botão “GO” para que cada espécie consiga comer a comida que depositamos a partir da parte superior do aquário.

```
to yellowfish-eat
  ask yellowfish
  [
    ask food-here
    [
      die
    ]
  ]
end
```

```
to redfish-eat
  ask redfish
  [
    ask food-here
    [
      die
    ]
  ]
end
```

Figura 16- Excerto de código que mostra como os peixes comem a comida

Com estas duas funções, quando a comida e o peixe estiverem na mesma célula, usamos a função “die” para eliminar a comida, realizando assim a simulação que o peixe a comeu.

Por fim, executamos uma variável que representa a idade dos peixes, para que, quando se exceder um determinado tick, os peixes morram naturalmente.

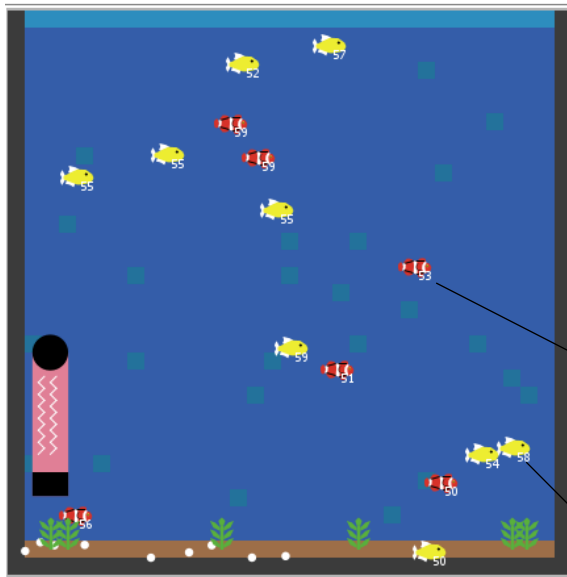


Figura 17- Captura de ecrã que mostra a idade dos peixes

```
redfish-own
[
  age-redfish
  energy-redfish
]

yellowfish-own
[
  age-yellowfish
  energy-yellowfish
]

create-redfish redfish_num
[
  setxy random-pxcor random-pycor
  set color red
  set size 2
  set shape "fish 3"
  set age-redfish 1 + random 10 ;random idade até aos 10
  set energy-redfish 10
  set label age-redfish
]

create-yellowfish yellowfish_num
[
  setxy random-pxcor random-pycor
  set color yellow
  set size 2
  set shape "fish"
  set energy-yellowfish 10
  set age-yellowfish 1 + random 10 ;random idade até aos 10
  set label age-yellowfish
]
```

Figura 18- Excerto de código que mostra como foi feita a idade dos peixes

Para esse efeito, criou-se as variáveis internas de cada *breed*, a idade e a energia (discutiremos mais à frente o propósito da variável *energy*). Usamos no botão *Setup* o “*set age-redfish/yellowfish 1 + random 10*”. Isto tem a finalidade de os peixes criados inicialmente tenham uma idade que não seja igual para todos, somando o valor 1 a um valor aleatório entre 0 e 10 que será mostrado debaixo de cada peixe, através da *label* que criamos. Por fim efetuamos o aumento de idade em relação ao aumento dos *ticks*.

```
ask redfish
[
  move 1
  set age-redfish (age-redfish + 1)
  set label round age-redfish
  if age-redfish = 300
  [
    set count_red-death count_red-death + 1
    die
  ]
]
```

Figura 19- Excerto de código que mostra como aumenta a idade dos peixes e como eles falecem

Ao colocar esta função no botão “GO” sempre que avançar um tick ele aumenta a sua idade no valor de um. Isto é feito através do “*set age-redfish (age-redfish + 1)*”. Quando os peixes chegam à idade 300 acabam por falecer.

Parte 2

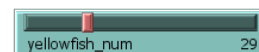
Introdução:

Na segunda fase de implementação, pretende-se aumentar o grau de dificuldade do trabalho. Aqui, ao contrário da primeira parte, pretende-se que os discentes utilizem a sua criatividade para adicionar elementos ao aquário. No nosso trabalho adicionamos alguns elementos.

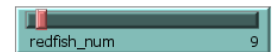
Desenvolvimento:

1. Número de peixes a dar *spawn*:

Para este efeito, utilizamos um *slider*. Cada um destes tem uma variável associada, com terminação *num*, que depois é inserida no lugar da criação dos peixes.



Escolher o número de peixes amarelos no aquário



Escolher o número de peixes vermelhos no aquário

```
create-redfish redfish_num
[
  setxy random-pxcor random-pycor
  set color red
  set size 2
  set shape "fish 3"
  set age-redfish 1 + random 10 ;random idade até aos 10
  set energy-redfish 17
]
```

Figura 20- Excerto de Código, com a variável com terminação *_num*.

2. Adicionar singularmente um peixe:

Para este efeito, criamos um botão, tanto para peixes vermelhos como para peixes amarelos, onde tem o código da Figura 20.

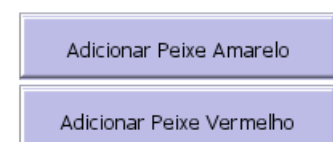


Figura 21- Excerto de Código com ambos os botões para adicionar peixes de forma singular.

3. Limpar Aquário:

Para esta funcionalidade, utilizamos *reverse-engineering*, para contradizer o efeito da toxicidade. Questionamos todos os *patches*, e os que tiverem cor que não seja azul, colocamo-la a azul.

```
to clean-tank
  ask patches
  [
    if pcolor = 94 or pcolor = 75 or pcolor = brown
    [
      set pcolor blue
    ]
  ]
]
```

Figura 22 -Excerto de código que tem como função a limpeza do aquário

4. Limpar Comida:

Para este efeito, simplesmente temos uma função associada a um botão, que pede a todos os agentes do tipo *comida*, que sejam eliminados.

```
to clean-food
  ask food [die]
end
```

Figura 23 – Excerto de código que limpa a comida perdida no aquário

5. Reprodução:

Quando dois peixes da mesma espécie se tocam, reproduzem um peixe do mesmo tipo. Isto é feito através da função *hatch*. Estes peixes precisam de um mínimo de energia para se reproduzirem. O propósito disto, será explicado no próximo ponto. Para além disto, também se utiliza uma variável para a contagem das crias – *count_yel-mating*; os peixes perdem energia por acasalarem.

```
to reproduce-yellow
  yellowfish-eat
  ask yellowfish
  [
    if any? yellowfish-on patch-ahead 1 and energy-yellowfish > 15
    [
      set energy-yellowfish energy-yellowfish - 5
      hatch 1
      [
        set count_yel-mating count_yel-mating + 1
      ]
    ]
  ]
end
```

Figura 24 – Excerto de código que faz a reprodução dos peixes amarelos

6. Energia como *internal value*:

Esta variável decidiu-se ser implementada para impedir um problema grave que se encontrou depois de alguma experimentação. Os peixes quando se tocam, reproduzem-se. Se um peixe se reproduzir então, ele olhará automaticamente para frente para verificar se é da mesma espécie e se

```
to redfish-eat
  ask redfish
  [
    if any? food-here
    [
      set energy-redfish energy-redfish + 2
    ]
    ask food-here
    [
      die
    ]
  ]
end
```

Figura 25 – Excerto de código que mostra que quando o peixe come recebe energia nos peixes vermelhos

pode se reproduzir. Isto faz com que, quando um acasalamento aconteça se tornem exponencialmente incontroláveis, dando *crash* no programa. Para isto precisamos de controlar através de uma variável. Quando um peixe nasce então terá sempre valor de energia inferior ao necessário para acasalar. Precisa também

necessariamente de comer. Isto é feito através da variável *energy-yellowfish* da Figura 24 e da função *yellowfish/redfish-eat*.

7. Selecionar quantidade de comida:

Indica-se, da mesma forma que na quantidade de peixes criados, através de um *slider*, a quantidade de comida que queremos criar na função.

```
to comida
  create-food create-food-num
  [
    set shape "circle"
    set size 0.5
    setxy random-pxcor 15
    set color white
  ]
end
```

Figura 26 – Excerto de código que permite escolher a quantidade de comida

8. Energia selecionada por comida:

Indica-se da mesma forma que a figura anterior, Figura 26, quanta energia queremos colocar por unidade de comida (*energy-given*) criada na função *yellowfish/redfish-eat*. Assim desta forma podemos experimentar ver a evolução dos nascimentos e mortes dos peixes.

```
to yellowfish-eat
  ask yellowfish
  [
    if any? food-here
    [
      set energy-yellowfish energy-yellowfish + energy-given
    ]
    ask food-here
    [
      die
    ]
  ]
end
```

Figura 27 – Excerto de código que mostra que quando o peixe come recebe energia nos peixes amarelos

9. Desenhar caminhos:

Para esta funcionalidade utilizamos um *switch*, que alterna entre colocar a *pen-down*, ou *pen-up*.

Além disso, para funcionar corretamente também adicionamos a prioridade “*pen*” à lista de variáveis internas das *turtles*. Utilizou-se um ciclo *IfElse* para o efeito.

```
turtles-own [pen]
ifelse draw-paths [ pen-down ] [ pen-up ]
```

Figura 28 – Duas linhas de código que permitem desenhar caminho onde os peixes passam.

10. Temperatura da Água:

Criou-se um agente chamado *temperatura*. Este agente é mais tarde invocado no gráfico para ser monitorizado. Usamos uma resistência elétrica que mantém a temperatura da água no valor desejado.

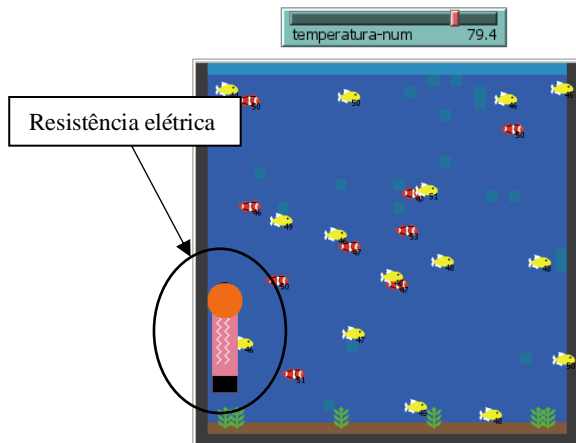


Figura 29- Captura do ecrã que mostra o aquário com a resistência elétrica e um deslizador que permitiu alterar a temperatura para 79°

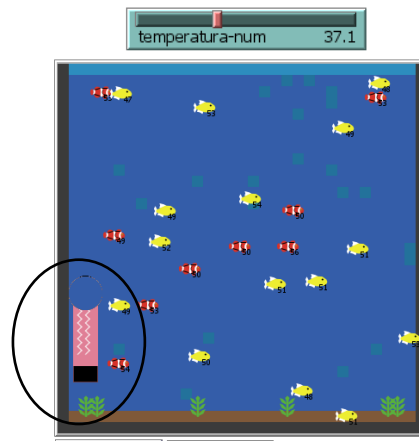


Figura 30- Captura do ecrã que mostra o aquário com a resistência elétrica e um deslizador que permitiu alterar a temperatura para 37°

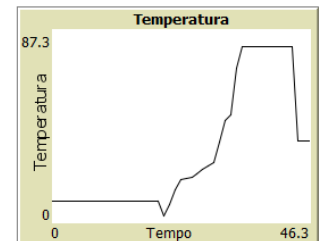


Figura 31 - Mostra a variação da temperatura

Como podemos verificar nas figuras acima, podemos através dos deslizadores a temperatura que idealizámos para o nosso aquário, trocando a cor da resistência elétrica consoante as temperaturas escolhidas. O gráfico da temperatura muda consoante alteração da mesma.

```
create-resistencial 1
[
  setxy -14 -4
  set size 3
  set shape "circle"
  set color blue

  if temperatura-num < 10 and temperatura-num >= 0 [set color 89
  ask redfish[
    set count_red-death count_red-death + 1
    die ]
  ]
  if temperatura-num < 50 and temperatura-num > 11[set color blue]
  if temperatura-num < 70 and temperatura-num > 51[set color green]
  if temperatura-num < 90 and temperatura-num > 71[set color 25]
  if temperatura-num < 101 and temperatura-num > 91[
    ask redfish[
      set count_red-death count_red-death + 1
      die ]
    ask yellowfish [
      set count_yellow-death count_yellow-death + 1
      die
    ]
    set color 12
  ]
]
```

Figura 32- Excerto de código que permitiu criar a resistência elétrica e trocar a sua cor consoante a temperatura

```
to graus
  create-temperatura temperatura-num
  [
    set shape "temperatura"
    set temperaturas-grafico temperatura-num
  ]
end
```

Figura 33- Excerto de código que cria a função "graus"

A função *drawn-resistencias* serve para colocar a resistência elétrica no aquário e escolhendo as suas características, logo abaixo dessa função temos o *create-resistencial* que faz a criação de um círculo na parte superior da resistência com o objetivo de mudar as cores perante a troca de temperatura. No final do código temos uma função que caso a temperatura da água seja superior a 91 graus todos os peixes do aquário não aguentam com o calor e acabam por falecer. Devido ao facto de os peixes amarelos terem melhores condições para viver em águas frias comparativamente aos peixes vermelhos, se o termómetro descer para baixo dos 10 graus os peixes vermelhos não irão resistir. Enquanto à figura 24, o excerto de código lá presente é uma função que faz com que o gráfico da figura 22 troque consoante a temperatura do aquário.

11. Reprodução ativada:

Para esta funcionalidade utilizou-se `ifelse reproduce-button-red[reproduce-red][]` um esquema similar ao do Ponto 9. Um bloco de código *IfElse*, de [Figura 34 -Linha de código que permite alterar a opção de reprodução ativa ou não](#) forma a podermos alternar se queremos a reprodução ativa ou não. Invoca-se também, conseqüentemente, as funções de reprodução respetivas aos dois tipos de *breed*.

12. Número de Crias:

Utilizou-se um monitor, para esta funcionalidade. Esse monitor está relacionado com a variável *count_yel-mating* ou *count_red-mating*. Assim, sempre que existir um nascimento, será contabilizado pelo monitor. Podemos ver essa variável na Figura 24.

13. Número de Mortes:

Utilizou-se a mesma metodologia que a utilizada no Ponto 12. Se o peixe tiver mais de 300 ticks, morre. Logo fazemos nesse local uma contagem para detetar a morte do peixe.

```
ask yellowfish
[
  move 1
  set energy-yellowfish (energy-yellowfish - 1)
  set age-yellowfish (age-yellowfish + 1)
  set label round energy-yellowfish
  ifelse reproduce-button-yellow[reproduce-yellow][]

  if age-yellowfish = 300
  [
    set count_yellow-death count_yellow-death + 1
    die
  ]
  ifelse draw-paths [ pen-down ] [ pen-up ]
]

ask food [move-food 1]
tick
end
```

Figura 35 – Excerto de código que faz com que os peixes morrem

14. Gráfico de Toxicidade:

O gráfico de toxicidade está associado ao seguinte comando:







Cor	Nome da Caneta	Comandos de Atualização da Caneta		
	BluePatch	plot count patches with [pcolor = blue]		
	BrownPatch	plot count patches with [pcolor = brown]		

Figura 36 – Definições do gráfico que permite mostrar a evolução da sujidade

Este serve para se contabilizar todas as *patches* da simulação. Quando elas ficarem com a cor castanha, significa que a toxicidade chegou ao topo da sua cadeia. Sendo assim, o gráfico irá contar essas *patches*.

15. Mostrar Energia ou Vida:

Para melhor visualização criou-se este *switch*. Dentro da função *go*, anteriormente explicada, temos o bloco *IfElse*, que é responsável por isto. Se o botão for acionado, irá executar um bloco. Se não se verificar, origina o bloco seguinte. Em ambos os blocos estão *labels* que são diferentes na variável que apresentam. Essas variáveis referem-se às duas *internal variables* dos peixes.

```
ifelse show-energy-red?
[ set label energy-redfish
  set label-color orange
]
[ set label age-redfish
  set label-color black
]

ifelse reproduce-button-red[reproduce-red][]
if age-redfish = 300
[
```

Figura 37 – Excerto de código que mostra a energia ou idade em uma etiqueta no peixe

16. Ativar desativar mortes por doença:

Para esta funcionalidade foi criado um switch de nome *enable-doença*. Este está ligado a um bloco *ifElse*, que se for verificado, coloca um dos peixes a meter a idade aleatória entre os 250 e 300. Ora se acontece que chega a 300, o peixe morre.

```
ifelse enable-doenca? [ask one-of redfish [set random-death-redfish random 1000]][]
```

Figura 38- Linha de código que tem como objetivo a possibilidade de doenças nos peixes

Conclusão e Notas Finais

Com o desenvolvimento terminado, pretendemos agora sintetizar os objetivos, e fornecer algumas notas adicionais.

Neste trabalho, separado em seis capítulos, podemos dizer com convicção que os objetivos colocados pelo docente foram completados com sucesso. Em primeiro lugar, obtivemos conhecimentos importantes e essenciais para a esférica teórica da Unidade Curricular. Isto graças em parte à pesquisa aprofundada que realizamos para tentar contextualizar o melhor possível o leitor, tal como, graças ao material fornecido pelo Docente da Unidade Curricular. Em segundo lugar, e em último, mas não menos importante achamos que a nossa simulação tem os elementos requeridos pelo professor:

1. Os peixes mexem-se dentro de um habitáculo, que simula um aquário;
2. O estado do habitáculo evolui, dependendo de certas condições;
3. A direção dos peixes é aleatória;

Assim sendo, concluímos este trabalho com um sentido de realização, podendo assim, todos os membros do grupo, dizer que, tanto o relatório como os assuntos discutidos nele, foram de grande importância, e que certamente contribuirão de forma positiva, para futuros projetos, tanto na Unidade Curricular presente, como no resto da vida académica que provirá.

Bibliografia

Novak, M. and Wilensky, U. (2011). NetLogo Fish Tank Genetic Drift model. <http://ccl.northwestern.edu/netlogo/models/FishTankGeneticDrift>. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Wilensky, U. (1999). NetLogo. <http://ccl.northwestern.edu/netlogo/>. Center for Connected Learning and Computer-Based Modeling, Northwestern University, Evanston, IL.

Desconhecido. (n.d.). *netlogo: asking turtles-on patch-here whether breed is the same - Stack Overflow*. Retrieved October 23, 2022, from <https://stackoverflow.com/questions/43499886/netlogo-asking-turtles-on-patch-here-whether-breed-is-the-same>

Coelho, Helder. (1995). *Inteligência artificial em 25 lições*.

Rich, Elaine., & Knight, Kevin. (1994). *Inteligencia artificial*.

Nilsson, N. J. (1998). *Artificial Intelligence : a new synthesis*. 513.

Neisser, U., Boodoo, G., Bouchard, T. J., Boykin, A. W., Brody, N., Ceci, S. J., Halpern, D. F., Loehlin, J. C., Perloff, R., Sternberg, R. J., & Urbina, S. (1996). Intelligence: Knowns and unknowns. *American Psychologist*, 51(2), 77–101. <https://doi.org/10.1037/0003-066x.51.2.77>

Inteligência – Wikipédia, a enciclopédia livre. (n.d.). Retrieved October 23, 2022, from <https://pt.wikipedia.org/wiki/Intelig%C3%Aancia>

O que é a inteligência artificial e como funciona? | Atualidade | Parlamento Europeu. (n.d.). Retrieved October 23, 2022, from <https://www.europarl.europa.eu/news/pt/headlines/society/20200827STO85804/o-que-e-a-inteligencia-artificial-e-como-funciona>

Oliveira M., Paulo. (2021). *Inteligência Artificial – Introdução*.

NetLogo 6.3.0 User Manual: NetLogo Dictionary. (n.d.). Retrieved October 23, 2022, from <https://ccl.northwestern.edu/netlogo/docs/dictionary.html>

ANEXO

```

.....
BREEDS .....
.....

```

```

breed [redfish a-fish]
breed [yellowfish b-fish]
breed [food foods]
breed [plant plants]
breed [resistencia resistencias]
breed [temperatura temperaturas]
breed [resistencia1 resistencias1]

```

```

.....
GLOBALS .....
.....

```

```

globals
[
  obstacle
  ground
  ground-real
  aquarium
  top

  num-turtles

  births-redfish
  births-yellowfish

  deaths-redfish
  deaths-yellowfish

  count_red-death
  count_yellow-death

  count_red-mating
  count_yel-mating

  count-blue-patch
  count-brown-patch

  temperaturas-grafico
]

turtles-own [pen]

temperatura-own [grafico-temperatura]

redfish-own
[
  age-redfish
  energy-redfish
  random-death-redfish
]

```

```

yellowfish-own
[
  age-yellowfish
  energy-yellowfish
  random-death-yellowfish
]

```

```

.....
.....
..... SETUP .....
.....
.....

```

```

to setup
  clear-all
  set births-redfish 0
  set births-yellowfish 0
  set temperaturas-grafico 10
  create-redfish redfish_num
  [
    setxy random-pxcor random-pycor
    set color red
    set size 2
    set shape "fish 3"
    set age-redfish 1 + random 10 ;random idade até aos 10
    set energy-redfish 17
    set random-death-redfish 0
  ]

```

```

create-yellowfish yellowfish_num
[
  setxy random-pxcor random-pycor
  set color yellow
  set size 2
  set shape "fish"
  set energy-yellowfish 19
  set random-death-yellowfish 0

  set age-yellowfish 1 + random 10 ;random idade até aos 10
]
drawn-plants
draw-obstacle
draw-ground
draw-aquarium

reset-ticks
end

```

```

.....
.....
..... BUTTONS .....
.....
.....

```

```

to go

```

```

if ticks > 20
[
  toxicity
]
drawn-resistencia
graus

ask redfish
[
  kill-random-red
  set energy-redfish(energy-redfish - 1) ;nao pode ir para função externa, Contagem dos ticks !
  set age-redfish (age-redfish + 1)
  set label round age-redfish

  ifelse show-energy-red?
  [ set label energy-redfish
    set label-color orange
  ]
  [ set label age-redfish
    set label-color black
  ]

  if age-redfish = 300
  [
    set count_red-death count_red-death + 1
    die
  ]

  ifelse draw-paths [ pen-down ] [ pen-up ]
  move 1
]
ask yellowfish
[
  move 1
  kill-random-yellow
  set energy-yellowfish(energy-yellowfish - 1)
  set age-yellowfish (age-yellowfish + 1)
  set label round energy-yellowfish

  ifelse show-energy-yellow?
  [
    set label energy-yellowfish
    set label-color orange
  ]
  [
    set label age-yellowfish
    set label-color black
  ]
]

ifelse reproduce-button-yellow[reproduce-yellow][]

if age-yellowfish = 300
[
  set count_yellow-death count_yellow-death + 1

```

```

    die
  ]
  ifelse draw-paths [ pen-down ] [ pen-up ]
]
ask food [move-food 1]
tick
end

```

```

to kill-random-red
  ifelse enable-doenca? [ask one-of redfish [set random-death-redfish random 1000]][]

```

```

  if random-death-redfish = 1
  [
    ask one-of redfish [die]
    set count_red-death count_red-death + 1
  ]
end

```

```

to kill-random-yellow
  ifelse enable-doenca? [ask one-of yellowfish [set random-death-yellowfish random 1000]][]

```

```

  if random-death-yellowfish = 1
  [
    ask one-of yellowfish [die]
    set count_yellow-death count_yellow-death + 1
  ]
end

```

```

.....
..... ACTIONS .....
.....

```

```

to adicionar-yellow
  create-yellowfish 1
  [
    setxy random-pxcor random-pycor
    set color yellow
    set size 2
    set shape "fish"
    set energy-yellowfish 19
    set age-yellowfish 1 + random 10 ;random idade até aos 10
    set label age-yellowfish
  ]
end

```

```

to adicionar-red
  create-redfish 1
  [
    setxy random-pxcor random-pycor
    set color red

```

```

    set size 2
    set shape "fish"
    set energy-redfish 19
    set age-redfish 1 + random 10 ;random idade até aos 10
    set label age-redfish
  ]
end

to clean-food
  ask food [die]
end

to reproduce-yellow
  yellowfish-eat
  ask yellowfish
  [
    if any? yellowfish-on patch-ahead 1 and energy-yellowfish > 15
    [
      set energy-yellowfish energy-yellowfish - 5
      hatch 1
      [
        set count_yel-mating count_yel-mating + 1
      ]
    ]
  ]
end

to reproduce-red
  redfish-eat
  ask redfish
  [
    if any? redfish-on patch-ahead 1 and energy-redfish > 15
    [
      set energy-redfish energy-redfish - 5
      hatch 1
      [
        set count_red-mating count_red-mating + 1
      ]
    ]
  ]
end

to yellowfish-eat
  ask yellowfish
  [
    if any? food-here
    [
      set energy-yellowfish energy-yellowfish + energy-given
    ]
    ask food-here
    [
      die
    ]
  ]
end

```

```

to redfish-eat
  ask redfish
  [
    if any? food-here
    [
      set energy-redfish energy-redfish + energy-given
    ]
    ask food-here
    [
      die
    ]
  ]
end

```

```

to graus
  create-temperatura temperatura-num
  [
    set shape "temperatura"
    set temperaturas-grafico temperatura-num
  ]
end

```

```

to drawn-resistencias
  create-resistencia 1
  [
    setxy -14 -7
    set size 7
    set shape "resistencia2"
    set color blue
  ]

```

```

create-resistencia1 1
[
  setxy -14 -4
  set size 3
  set shape "circle"
  set color blue

```

```

if temperatura-num < 10 and temperatura-num >= 0 [set color 89
ask redfish[
  set count_red-death count_red-death + 1
  die ]
]
if temperatura-num < 50 and temperatura-num > 11[set color blue]
if temperatura-num < 70 and temperatura-num > 51[set color green]
if temperatura-num < 90 and temperatura-num > 71[set color 25]
if temperatura-num < 101 and temperatura-num > 91[
  ask redfish[
    set count_red-death count_red-death + 1
    die ]
  ask yellowfish [
    set count_yellow-death count_yellow-death + 1

```



```

        die
    ]
    set color 12
  ]
]

end

to comida
  create-food create-food-num
  [
    set shape "circle"
    set size 0.5
    setxy random-pxcor 15
    set color white
  ]
end

.....
..... MOVE .....
.....
.....

to move-food [dist ]
  set heading 180
  let turn-angle (random-float 60) - 30

  ; if world does not wrap target patch could be nobody
  let target-patch patch-right-and-ahead turn-angle dist

  ifelse target-patch != nobody and not member? target-patch ground
  [ rt turn-angle ]
  [ rt turn-angle - 0 ]

  ; turtle could still wind up outside region, so then don't move
  set target-patch patch-right-and-ahead 0 dist
  if target-patch != nobody and not member? target-patch ground
  [ fd dist ]
end

to move [ dist ]
  let turn-angle (random-float 60) - 30

  ; if world does not wrap target patch could be nobody
  let target-patch patch-right-and-ahead turn-angle dist

  ifelse target-patch != nobody and not member? target-patch obstacle
  [ rt turn-angle ]
  [ rt turn-angle - 180 ]

  ; turtle could still wind up outside region, so then don't move
  set target-patch patch-right-and-ahead 0 dist
  if target-patch != nobody and not member? target-patch obstacle
  [ fd dist ]
end

```

```

to toxicity
  ask one-of patches
  [
    if pcolor = blue
    [
      set pcolor 94
    ]
  ]
  ask one-of patches
  [
    if pcolor = 94
    [
      set pcolor 75
    ]
  ]
  ask one-of patches
  [
    if pcolor = 75
    [
      set pcolor brown
    ]
  ]
end

```

```

to clean-tank
  ask patches
  [
    if pcolor = 94 or pcolor = 75 or pcolor = brown
    [
      set pcolor blue
    ]
  ]
end

```

```

.....
..... DRAWING PROCEDURES .....
.....

```

```

to drawn-plants
  create-plant 1
  [
    setxy 4 -14
    set size 2
    set shape "plant"
    set color green
  ]
  create-plant 1
  [
    setxy -4 -14
    set size 2
    set shape "plant"
    set color green
  ]

```

```

]
create-plant 1
[
  setxy 13 -14
  set size 2
  set shape "plant"
  set color green
]
create-plant 1
[
  setxy 13.9 -14
  set size 2
  set shape "plant"
  set color green
]
create-plant 1
[
  setxy -14 -14
  set size 2
  set shape "plant"
  set color green
]
create-plant 1
[
  setxy -13 -14
  set size 2
  set shape "plant"
  set color green
]
end

```

```

to draw-obstacle
  set obstacle patches with
  [
    pxcor = max-pxcor or
    pxcor = min-pxcor or
    pycor = 16 or
    pycor = min-pycor
  ]
  ask obstacle [set pcolor red]
  ask n-of num-turtles patches with [pcolor != red] [ sprout 1 [ set color red set heading random
360 ]]
end

```

```

to draw-ground
  set ground patches with [pycor = min-pycor]
  ask ground [set pcolor black]
end

```

```

to draw-aquarium
  ask patches [set pcolor blue]

  set top patches with [pycor = max-pycor]

```

```
ask top [set pcolor 95]

set ground-real patches with [pycor = -15]
ask ground-real [set pcolor 34]

set aquarium patches with
[
  pxcor = max-pxcor or
  pxcor = min-pxcor or
  pycor = min-pycor
]
ask aquarium [set pcolor 2]
end
```