LINGUAGEM DE PROGRAMAÇÃO AQUA

Uma linguagem de inútil sobre relações entre peixes e rios

Motivações

- Essa linguagem foi pensada de maneira que seja necessário pensar na curva de crescimento de populações e vazões de rios.
- Para realizar loops e condicionais é necessário pensar em como as populações interagem e curvas de crescimento as vezes exponenciais.

Características

- Existem 2 tipos de variáveis > Rios e Peixes
- As variáveis não suportam valores negativos
- É possível operar com números racionais se desejado, maioria das operações trabalham apenas com inteiros.
- Variáveis de rios armazenam o fluxo e as de peixe o população E o quanto cada unidade consome por ciclo (tupla).
- Rios e peixes uma vez criados existem até serem extintos e até lá não podem ser redeclarados

Características

- Os loops representados pelo sustains funcionam da seguinte maneira:
- Grafia :
- X sustains Y
- ...
- pass_time
- X é um peixe ou um rio
- Y é um peixe
- Y[0] é a população ou fluxo e Y[1] é o consumo por unidade
- Loop enquanto $X[0] > 0$
- No começo de cada iteração:
- $dif = X[0] - (Y[0] \times Y[1])$
- $sub = dif\ se\ dif > 0\ caso\ contrário\ sub = 0$
- $Y[0] = Y[0] + (Y[0] - Y[0]\%2) + sub$
- $X[0] = X[0] - (Y[0] \times \mathrm{Y}[1])$

Explicando o loop

- O loop é feito de forma a simular o fata de uma população ser o alimento de outra e por isso uma deve consumir a outra para crescer
- Se não houver alimento suficiente o loop encerra e também a população predadora não cresce completamente se não tiver alimentado toda a sua população (função de sub).
- Toda vez que uma população cresce são levados em consideração apenas as duplas, uma vez que peixes não se multiplicam sozinhos (por isso a subtração por pop%2).

Exemplos

- No README do repositório, existe um código comentado, mas nesta apresentação seram colocados alguns pontos principais.

Declaração de variáveis, flow e Discover.

```
river x create 10
river y create 10
x >> 5 >> y
discover(x)
discover(y)
```

- Neste exemplo são declarados os rios x e y com fluxo de 10 e em seguida x transfere 5 de seu fluxo para y.
- Por fim são imprimidos no terminal os fluxos de x e y.

Branch e acumulate

```
river x create 10
river y create 10


x branch 5
x acumulate 5
discover(x)
```

- Neste exemplo, após as declarações são realizadas as operações de branch e accumulate, equivalentes as operações de divisão e multiplicação.
- Ambas as operações armazenam o resultado no primeiro termo (x no caso).

Declaração de variáveis, e consumo

```
fish z create 2,1
fish f create 2,1
f -> z
```

- Neste exemplo são declarados dois peixes z e f com suas populações e consumes por unidade respectivamente.
- Logo em seguida é realizada a operação de consume que realiza exatamente a mesma operação do sustains, mas sem loopar e no formato f consome z (diferente ordem das variáveis).

Loop e condicional

```
river x create 10

fish z create 2,1
fish f create 2,1
river o create 10

x sustains z

event z inf f
discover(o)
conclude

pass_time
```

- Este exemplo realiza o loop do sustains como já explicado
- E caso z seja inferior a z, imprime o.
- Possiveis condições são inf, sup, ig (inferior, superior e igual). Em português pois assim foi feito uma vez.

Rain e dry

```
river x create 10
river y create 10


rain(y)
dry(x)
```

- Operação de rain(rio) soma ao fluxo de todos os rios o valor do rio passado como argumento.
- Operação dry(rio) subtrai do fluxo de todos os rios o valor do rio passado como argumento.