Resumo blocos arena

Seize

Aloca (ocupa) um recurso.

> Ex: Cliente ocupa um caixa de supermercado.

Delay

Causa um atraso (espera) no processo.

> Ex: Tempo de atendimento do cliente.

Release

Libera o recurso após o uso.

> Ex: Cliente termina o pagamento e libera o caixa.

Animações Diversas

Representam graficamente recursos, filas, processos, etc.

> Ex: Ícone de um funcionário ocupado ou livre.

Funções NQ, MR, RESUTIL

- `NQ(Queue)`: Número de entidades na fila.
- `MR(Resource)`: Número de recursos ocupados.
- `RESUTIL(Resource)`: Percentual de utilização do recurso.

Decide

Faz uma escolha condicional ou baseada em probabilidade.

> Ex: Pedido aprovado (80%) ou rejeitado (20%).

Assign

Modifica atributos, variáveis ou tipos de entidade.

> Ex: Aumentar prioridade de entrega urgente.

Label

Marca um ponto específico no modelo.

> Ex: Definir “Reprocessar Peça”.

Goto Label

Move a entidade diretamente para um Label.

> Ex: Redirecionar cliente para reavaliação.

Advanced Set

Cria um conjunto agrupando vários elementos do mesmo tipo (filas, storages, etc.).

> Exemplo: Criar um conjunto de caixas automáticos para alocar clientes em qualquer um disponível.

Record

Registra estatísticas durante a simulação. Pode contar eventos, medir tempos, registrar expressões, etc.

> Exemplo: Contar quantos clientes passaram pelo atendimento ou medir o tempo de serviço.

Element Tally

Coleta dados estatísticos contínuos (como tempos ou valores) e calcula medidas como média, desvio padrão, etc.

> Exemplo: Medir o tempo médio de espera de clientes numa fila.

Separate

Separa um grupo de entidades formadas por um Batch ou cria cópias (clones) de uma entidade.

> Exemplo: Após agrupar produtos num lote, separar para processar cada item individualmente.

Clone

Gera cópias de uma entidade de forma direta, sem agrupamento.

> Exemplo: Duplicar um cliente para tratá-lo em dois processos paralelos (financeiro e técnico).

Wait

Similar ao Hold: mantém a entidade parada até uma condição ser satisfeita.

> Exemplo: Cliente aguarda uma vaga no estacionamento.

Signal

Envia um sinal para liberar entidades que estão em Wait/Hold.

> Exemplo: Quando uma vaga é liberada, o Signal avisa os carros esperando.

Failure

Simula falhas de recursos (quebras, manutenções programadas). Pode ser baseado em tempo ou contagem de uso.

> Exemplo: Máquina quebra a cada 8 horas de operação ou a cada 50 peças produzidas.

Search

Procura uma entidade específica em uma fila ou grupo, conforme uma condição.

> Exemplo: Buscar o pedido de maior valor na fila para atendimento prioritário.

Remove

Remove uma entidade de uma fila ou storage manualmente (não necessariamente a primeira).

> Exemplo: Remover o cliente VIP da fila normal para atendimento preferencial.

Search and Remove

Procura e remove uma entidade de uma fila ao mesmo tempo, baseado em uma condição.

Exemplo: Encontrar e remover o item mais antigo de um estoque para despacho.

Element Resource

Define recursos como máquinas ou pessoas.

Ex: Máquina CNC.

Element Set

Grupo de elementos do mesmo tipo (recursos, filas, etc.).

Ex: Conjunto de caixas rápidos.

Seizable Selection Rule

Regra para escolher recurso dentro de um set.

Ex: Selecionar o caixa com menor fila.

Seizable Save Attribute

Guarda o recurso escolhido em um atributo.

Ex: Registrar qual operador atendeu.

Funções DAVG

- `DAVG(Fila)`: Calcula o tempo médio de espera ou de ocupação.

Resource Capacity

Número de unidades disponíveis de um recurso.

> Ex: 5 médicos disponíveis.

Element Queue

Cria e associa uma fila a um recurso ou processo.

> Ex: Fila de peças esperando inspeção.

Queue Type

Define a regra de atendimento da fila (FIFO, LIFO, atributo).

> Ex: FIFO — primeiro a entrar, primeiro a sair.

Element Variable

Cria variáveis para controle interno do modelo.

> Ex: Número de pedidos processados.

Element Expression

Expressão matemática ou lógica usada no modelo.

Ex: Custo total = preço base + imposto.

Funções AINT, ANINT, DISC

- `AINT(x)`: Parte inteira de x (descarta fração).
- `ANINT(x)`: Arredonda x para o inteiro mais próximo.
- `DISC(p1, v1, p2, v2, ...)`: Escolha discreta com probabilidades.

Ex: `DISC(0.6,1, 0.4,2)` = 60% chance de sair 1 e 40% de sair 2.

Função TRIA

- É uma função que gera números aleatórios no Arena

Ex: `TRIA(10, 35, 50)`

10 → valor mínimo possível,

35 → valor mais provável (o "pico" da distribuição),

50 → valor máximo possível.

Resumindo, o valor sorteado vai ficar entre 10 e 50, mas a maioria dos valores tende a se concentrar em torno de 35

Coletar tempo dentro de um intervalo

Para medir o tempo que uma entidade leva em uma atividade, crie um atributo na entidade. Antes do "Delay", use um "Assign" atribuindo `TNOW` ao atributo (registro do momento atual). Depois, ao sair do "Delay", use um "Record" para registrar a diferença entre `TNOW` atual e o valor salvo no atributo anterior.

Dica: pode usar **Expression** (cálculo de tempo) ou **Count** (quantidade de entidades).

Funcionamento do Hold e Signal

O bloco **Hold** segura a entidade até receber um **Signal**. Cada Hold pode esperar por um número de sinal diferente (um inteiro).

> Exemplo: Se os Holds A e C esperam o sinal "2", quando o **Signal(2)** é ativado, ambos são liberados.

Uso do Separate para clonar entidade

O **Separate** pode clonar uma entidade que entra no bloco.
Para evitar problemas de cópia errada, cada processo deve ter um **ID único**, criado com uma variável global. Assim cada entidade clone sabe a quem pertence.

Funcionamento do Match

O **Match** combina entidades vindas de diferentes filas (de 1 até 5 filas).

Cada entrada vai para uma fila distinta e espera formar o "match".

> Observação: estudar mais sobre a utilidade prática do Match.

Separate para separar lotes

O **Separate** também serve para separar lotes criados com o **Batch**.
No Batch, entidades são agrupadas (temporária ou permanentemente), e o Separate desfaz esse agrupamento.

Exemplo: posso separar os dois primeiros itens de um lote ou aplicar regras específicas.

Clone para duplicar entidades

Se a ideia é apenas duplicar a entidade, é melhor usar o **Clone**.

A saída do Clone normalmente é direcionada usando **Labels** para especificar os caminhos das cópias.

Fluxo:
Entidade -> Clone -> Label 1, Label 2

Dica para estudar os componentes

No Arena, dá para aprender mais sobre cada bloco clicando em **Pesquisar SMARTS**.

Os SMARTS são exemplos práticos rápidos sobre o uso de blocos.

Exercício Cap 6 Modelagem

Modele, simule e extraia resultados de cada um dos sistemas descritos a seguir.

1. Num sistema computacional, os tempos entre chegadas de programas/processos para execução segue uma distribuição exponencial com média 10 segundos. Após chegar, cada processo executa no único processador existente por um tempo que segue uma distribuição normal de média 8s e desvio 4s. Após terminar de executar no processador, o processo é excluído (desaparece) do computador.
   Mostre animações de todos os recursos filas, tempo atual no modelo, tempo de execução, gráficos da ocupação dos recursos, etc
   <seize, delay, release, animações diversas, funções nq, mr, resutil>
2. Altere o sistema 1 de modo que aproximadamente metade dos processos tenham tempo de execução que segue uma distribuição normal de média 2s e desvio 0.5s, enquanto a outra metade continua com a mesma distribuição do sistema 1.
   <decide, assign, label, goto label>

3. Altere o sistema 2 de modo que haja 4 processadores disponíveis. Um processo tenta alocar o processador menos ocupado. Se todos estiverem ocupados, o processo espera numa fila única de processos prontos a executar. Altere o tempo de execução de norm(2, 0.5) para norm(25, 5). Atualize as animações para refletir os 4 processadores.
   <element resource, element set, seizable selection rule, seizable save attribute, funções davg>
4. Altere o sistema 3 de modo que um programa só pode executar se tiver antes sido alocado em memória RAM. Ao ser criado, um processo demanda uma quantidade de memória que é no mínimo 10KB, no máximo 50KB e o valor mais provável é 35KB. O total de memória RAM disponível é 412KB. Se há memória livre suficiente para todo o programa, ele é alocado na RAM, executa e depois que terminou de executar, libera a memória usada. Se não houver memória para todo o programa, ele é colocado numa fila de espera por memória, em que prioridade é dada para os programas que precisam de menos memória.
   <resource capacity, *element* queue, queue type>
5. Altere o sistema 4 de modo que cada processo pode pertencer a uma das seguintes categorias: User_CPU_Bound, User_IO_Bound, OS_CPU_Bound, OS_IO_Bound e OS_Daemon, todos com a mesma probabilidade. Aproximadamente metade dos processos são de usuário e o restante do sistema operacional. Os tempos de execução para uma dessas categorias são, respectivamente e em segundos, lognormal(5, 2), lognormal(1, 0.3), lognormal(0.2, 0.05), lognormal(0.1, 0.05), uniforme(0.1, 0.2). Os tamanhos de memória exigidos por processos do usuário e do sistema operacional são, respectivamente e em KB (inteiros), unif(10, 50) e unif(5, 20). Além disso, cada processo possui uma prioridade, que é um número de -20 a +20. Quanto menor o número, mais prioritário é o processo. Processos do sistema operacional podem ter as seguintes prioridades (inclusive): daemon: de -20 a -10, IO bound: de -15 a -5, e CPU bound: de -10 a 0. Processos do usuário podem ter as seguintes prioridades (inclusive): CPU bound: de 0 a 15, e IO bound: de 10 a 20. Atualize as animações para que todas as informações do processo sejam apresentadas. Inclua animações mostrando a frequência de cada categoria de processos e a frequência de processos de usuário e do sistema operacional.
   <*element* variable, *element* expression, funções aint, anint, disc>
6. Altere o sistema 5 de modo que processos do sistema operacional executem apenas no processador 1, enquanto processos do usuário executam apenas nos processadores restantes, de forma cíclica. Além disso há uma fila de processos prontos para cada processador, e o escalonamento de processos (tanto do usuário quanto do SO) é por prioridade não preemptiva. Além disso, a memória RAM agora em 640KB, e 128KB são reservados para o sistema operacional e o restante é reservado para programas de usuário. Quando há processos esperando na fila por

memória, primeiro são alocados processos do sistema operacional e depois os processos de usuário. Gere estatísticas sobre o tempo em que os processos levam desde que são criados (antes mesmo de alocar memória) até o instante em que começam a executar. Atualize as animações.
<advanced set, record, *element* tally>

7. Altere o sistema 6 de modo que qualquer processo (com exceção dos *daemons*) não executa simplesmente num processador e depois termina. Na realidade cada processo realiza ciclos de CPU-IO, ou seja, executa no processador por um novo tempo aleatório que segue a expressão dada no sistema 5 para sua categoria e depois acessa um dispositivo de IO, geralmente o disco rígido (o que completa um ciclo). Qualquer processo que acessa o disco rígido realiza de 5 a 15 ciclos de CPU-IO antes de terminar. Daemons, por outro lado, sempre realizam 100 ciclos, e o dispositivo de IO é um "timer" de exatamente 1 segundo (e não o disco rígido), ou seja executam num processador e esperam um 1 segundo, o que completa um ciclo. Ou seja, deamons acessam CPU e esperam pelo timer, e o restante acessam CPU e depois o disco rígido. O disco rígido tem capacidade de atender as requisições de acesso de 16 processos simultaneamente, e o tempo de acesso de cada requisição seque uma exponencial com média 10 ms. Além disso, uma vez que um processo é alocado a um processador no primeiro ciclo de CPU-IO (por alocação cíclica), ele gera "afinidade" com aquele processador e, nos próximos ciclos de CPU-IO, ele será alocado sempre no mesmo processador. Gere estatísticas sobre o tempo em que os processos levam desde que são criados até o instante em que começam a executar (o primeiro ciclo) para cada categoria de processo, separadamente. Atualize as animações.
<apenas lógica e funções para as animações>
8. Altere o sistema 7 de modo que o escalonamento de processos nos processadores inclua uma fatia (quantum) de tempo máxima (tipo round robin) para que um processo execute. Se um processo precisa executar por menos que essa fatia de tempo ele simplesmente executa até terminar; senão, após sua fatia de tempo, verifica-se se há algum outro processo esperando na fila de prontos daquele processador. Se não houver, o processo executando pode continuar executando no processador por mais uma fatia de tempo. Se houver outro processo na fila de prontos, o processo executando libera a CPU e volta para a fila de prontos.
<apenas lógica>
9. Altere o sistema 8 de modo que o escalonamento de processos nos processadores seja preemptivo de fato, ou seja, se um processo está executado, mas outro processo mais prioritário chega na fila de prontos daquele processador, o processo sendo executado é retirado da CPU (preemptado) imediatamente e o processo mais prioritário deve ser escalonado. Colete estatísticas sobre os processos que foram

preemptados. Atualize as animações.
<separate / clone, wait, signal>

10. Altere o sistema 9 de forma que durante a execução (de uma fatia de tempo) de um processo de usuário qualquer, existe uma propabilidade de 0,00025% do processo travar e aquele processador parar de funcionar. Quando isso acontece, ele pára imediatamente e permanece inativo por 1 segundo, após o qual algum timer (watchdog) detecta esse travamento e reinicia o processador. Assuma que nenhum processo na fila de prontos daquele processador é afetado e que o processo em execução continua a executar normalmente após o processador travado voltar a funcionar.
<failure>
11. Altere o sistema 10 de modo que quando um processador trava, todos os processos na fila de prontos e o processo executando são imediatamente destruídos. Colete estatística sobre os processos destruídos.
<search, remove, search and remove>

RESPOSTAS

1.

Create -> Assign -> Process (Seize Delay Release) -> Record -> Dispose

Configuração dos blocos:

Create: Name: "Chegada de Processo", Time Between Arrivals: Exponential(10), Units: Seconds.

Assign: Add -> Type: Attribute, Attribute Name: “StartTime”, New Value: TNOW (marca o tempo que o processo chegou no sistema).

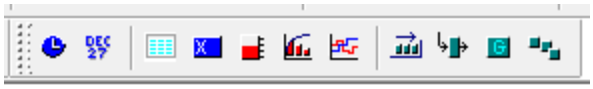
Process: Name: "Executar no Processador", Action: Seize Delay Release, Resource: “Processador” (novo recurso criado), Delay Type: Normal(8,4), Units: Seconds.

Record: Add -> Type: Expression, Expression: TNOW - StartTime, Tally Name: “Tempo de Execução” (registra o tempo que o processo gastou no processador).

Dispose: Name: "Excluir Processo" (processo desaparece após terminar o processamento).

Definir tempo de simulação: Run -> Setup -> Replication Parameters: Replication length: 3600, Time Units: Seconds

Animações e gráficos:

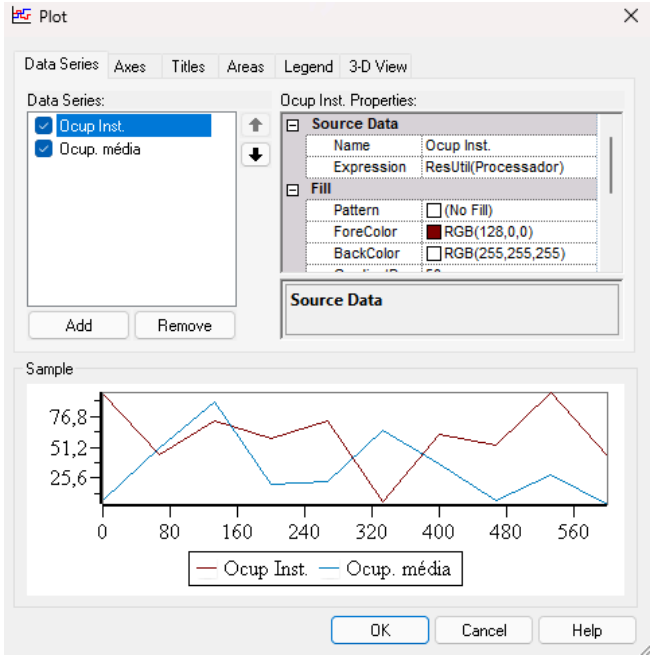

(esses são os ícones)

Resource -> Animação de estado Ocupado/Livre para o Processador.
Identifier: “Processador” -> Ok

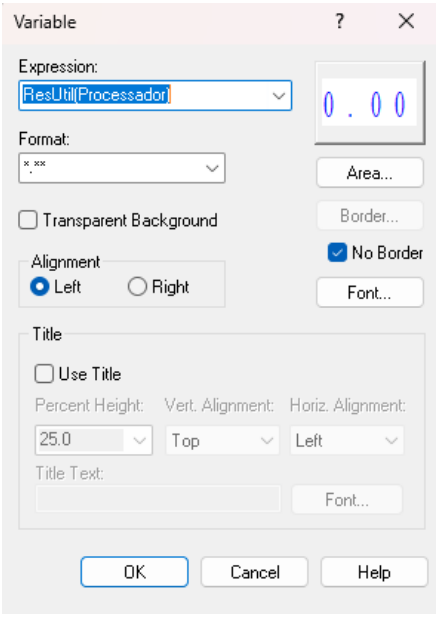
Queue -> Mostrar a fila do Processador para visualizar processos esperando.
Queue name: “Executar no processador.Queue”
Nota: Essa animação já existe, é isso:

TNOW -> Mostrar o tempo atual de simulação como texto animado.
Variable -> Expression: TNOW, Format: *.***

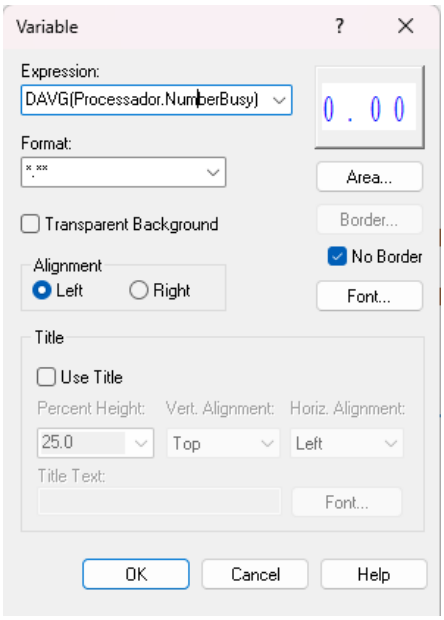
Plot -> Plotar a ocupação do Processador (Busy) e sua Utilização (Utilization).
Add: Name: Ocup. Inst., Expression: ResUtil(Processador)
Add: Name: Ocup. Méd., Expression: DAVG(Processador.NumberBusy)



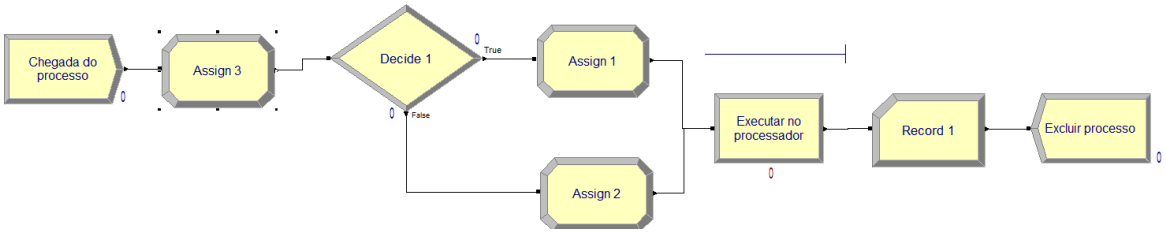
Ocupação instantânea:
Expression: ResUtil(Processador) , Format: *.**



Ocupação média:
Expression: DAVG(Processador.NumberBusy), Format: *.**



2.



Create -> **Decide** -> **Assign** -> **Process** (Seize Delay Release) -> **Record** -> **Dispose**

Configuração dos blocos:

Decide: Type: 2-way by Chance, Percent True: 50 (50% chance de seguir para cada tipo de tempo de execução).

Assign (caminho True): Add -> Type: Attribute, Attribute Name: “TempoProcessamento”, New Value: NORM(2,0.5) (tempo de execução rápido).

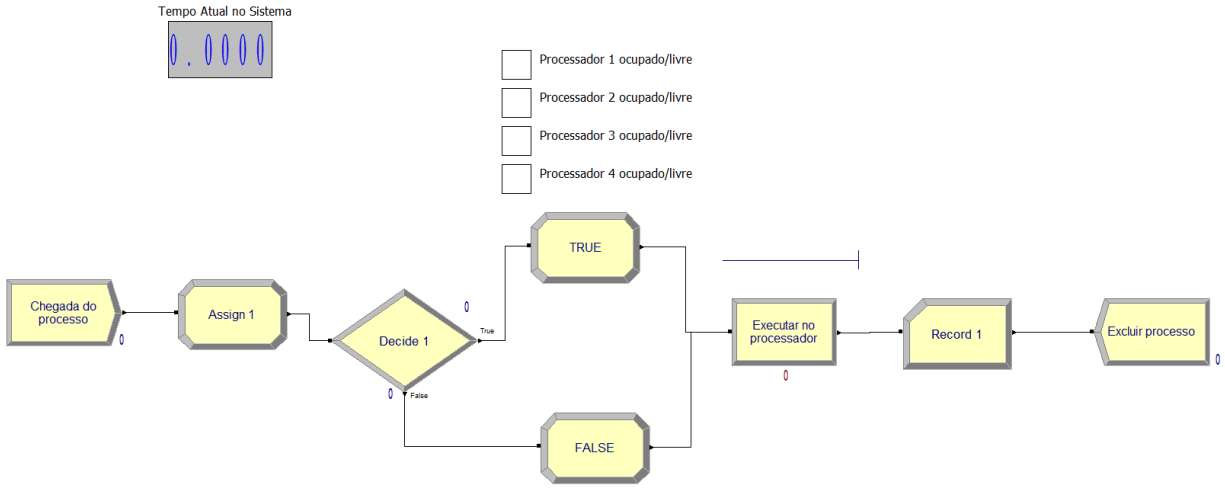
Assign (caminho False): Add -> Type: Attribute, Attribute Name: “TempoProcessamento”, New Value: NORM(8,4) (tempo de execução padrão).

Process: Name: "Executar no Processador", Action: Seize Delay Release, Resource: “Processador” (novo recurso criado), Delay Type: Expression, Expression: “TempoProcessamento”, Units: Seconds.

Record: Add -> Type: Expression, Expression: TNOW - StartTime, Tally Name: “Tempo de Execução” (registra o tempo que o processo gastou no processador).

Dispose: Name: "Excluir Processo" (processo desaparece após terminar o processamento).

3.



Create -> **Decide** -> **Assign** (TempoProcessamento) -> **Process** (usa ProcessadorSet, escolhe menos ocupado) -> **Record** -> **Dispose**

Configuração dos blocos:

Criar 4 processos:
ABA BASIC PROCESS -> **Resource** -> Criar 4 recursos: Processador 1, Processador 2, Processador 3 e Processador 4. Todos com **Capacity**: 1.

Criar um Set de processadores:
ABA BASIC PROCESS -> **Set** -> Name: “ProcessadorSet”, Type: Resource. Adicionar os 4 processadores como membros do Set.

Alterar o bloco Process existente:
Type: Set, Set Name: “ProcessadorSet”, Selection Rule: Smallest Number Busy, Save Attribute: “EscolhaProcessador” (***O Arena cria o atributo automaticamente*** *quando você escreve o nome dele na configuração do* ***Process***.)

Assign (caminho True): Add -> Type: Attribute, Attribute Name: "TempoProcessamento", New Value: NORM(25,5) (tempo de execução rápido atualizado).

Process: Name: "Executar no Processador", Action: Seize Delay Release, Resource: ProcessadorSet (novo conjunto de recursos criado com 4 processadores), Set Name: “Processadores Set”, Delay Type: Expression, Expression: "TempoProcessamento", Units: Seconds.

- **Seizable Type**: Set (não é Resource isolado, agora usa o **ProcessadorSet**).
- **Seizable Selection Rule**: Smallest Number Busy (escolhe o processador menos ocupado).O processo vai tentar alocar o processador **menos ocupado** naquele

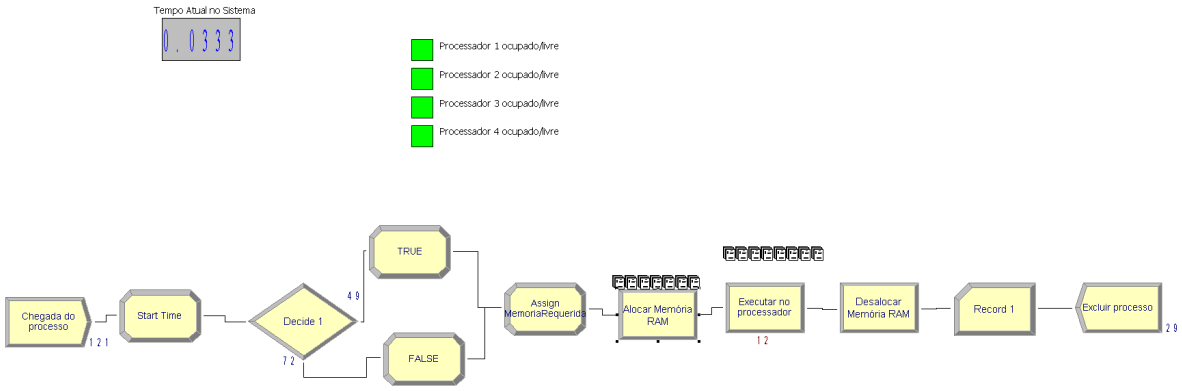
momento. Se tiver empate (vários livres), ele pega o primeiro da lista. *Quando você usa um **Set**, o Arena precisa saber **qual recurso dentro do conjunto** escolher.* O processo sempre escolhe o **processador menos ocupado** automaticamente!

- **Seizable Save Attribute**: Create Attribute "EscolhaProcessador" (salva qual processador foi alocado). Se o processo pegou o **Processador 2**, o atributo EscolhaProcessador vai guardar essa informação. *Serve para **registrar** qual processador a entidade usou.*

Atualizar animações (criar 4 animações para os 4 processos):
PS: Tem que remover tudo que diz "Processo" das animações de antes, porque senão dá erro, pois agora só temos o Set e não ele individual. Não consegui fazer funcionar o gráfico pra todos os 4 processos.

Resource -> Animação de estado Ocupado/Livre para o Processador.
Identifier: "Processador 1" -> Ok
Resource -> Animação de estado Ocupado/Livre para o Processador.
Identifier: "Processador 2" -> Ok
Resource -> Animação de estado Ocupado/Livre para o Processador.
Identifier: "Processador 3" -> Ok
Resource -> Animação de estado Ocupado/Livre para o Processador.
Identifier: "Processador 4" -> Ok

4.



Create -> **Assign** (StartTime) -> **Decide** -> **Assign** (TempoProcessamento) -> **Assign** (MemoriaRequerida) -> **Seize** (RAM) -> **Process** (Seize Delay Release no Processador) -> **Release** (RAM) -> **Record** -> **Dispose**

Configuração dos blocos:

Criar 1 novo recurso Memória RAM:
ABA BASIC PROCESS -> **Resource** -> Name: “MemoriaRAM”, Capacity: 412 *(cada unidade representa 1KB)*

Assign: Attribute Name: “MemoriaRequerida”, New value: TRIA(10, 35, 50)

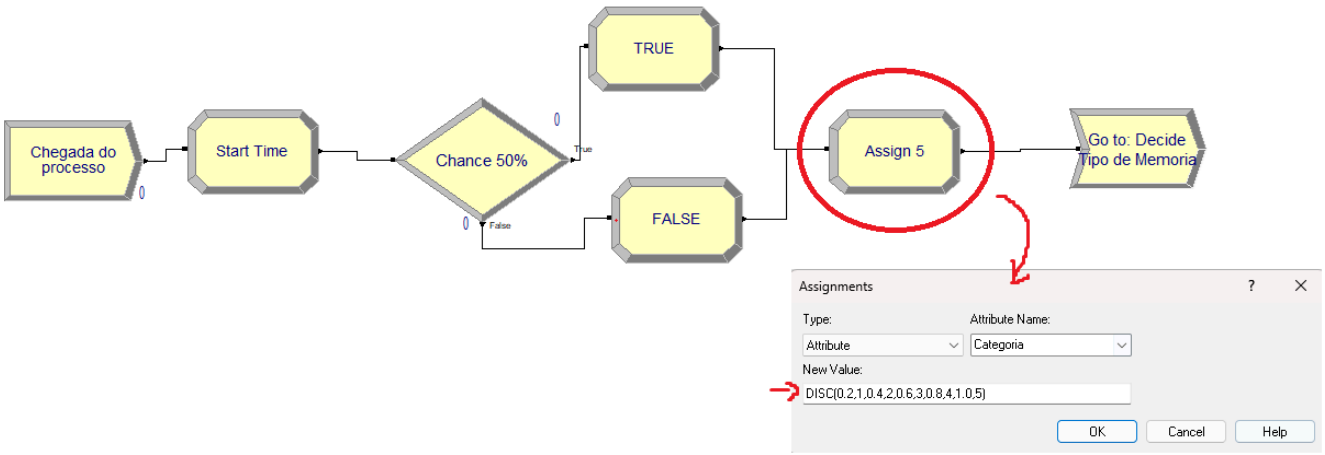
Seize (Alocar RAM): Name: “Alocar Memória RAM”, Criar antes do Process. Resource: Type: Resource, Resource Name: “MemoriaRAM”, Units to Seize: “MemoriaRequerida”, Queue Type: Queue, Attribute: “MemoriaRequerida”

Release Only (Liberar RAM): Name: “Desalocar Memória RAM”, Criar depois do Process. Resource Name: “MemoriaRAM”, Units Released: “MemoriaRequerida”

5.

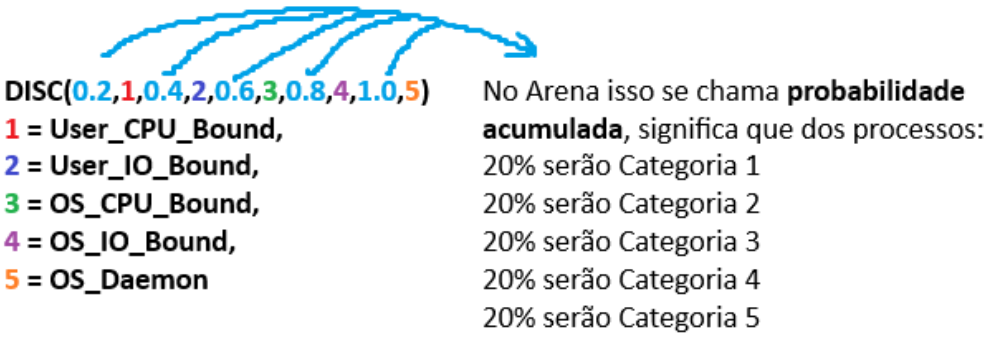
Essa resolução dividi em 5 partes. No Arena é possível usar os blocos **Go-To Label** e **Label**, assim dá pra organizar o exercício em várias partes, sem que fique poluído.

Primeira parte:



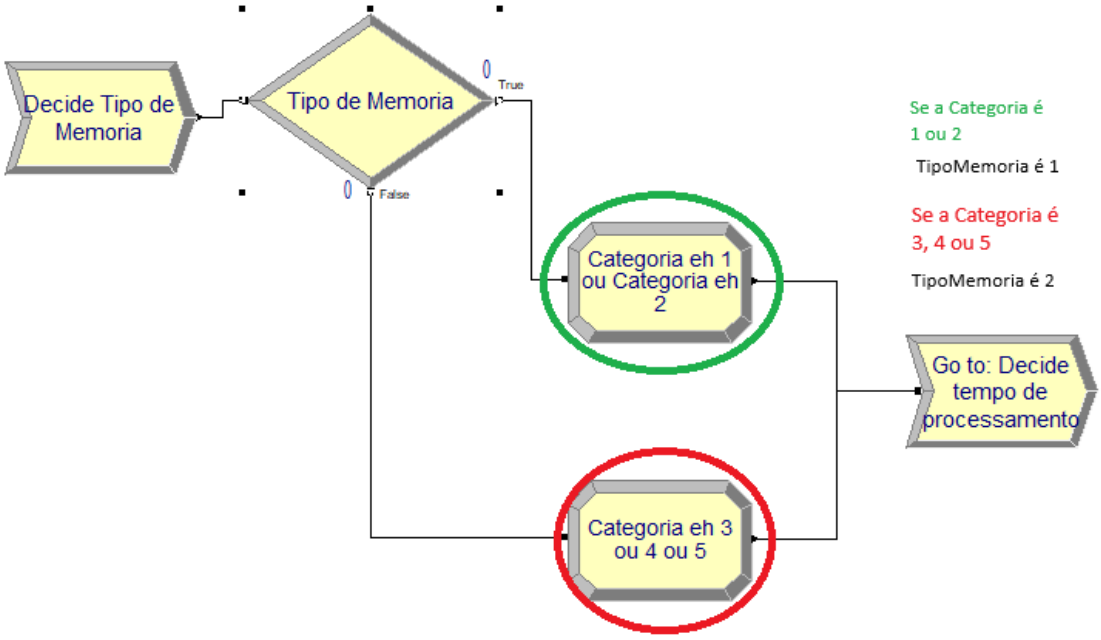
Assign -> Type: Attribute, Attribute Name: “Categoria”,
New Value: DISC(0.2,1,0.4,2,0.6,3,0.8,4,1.0,5)

Explicação do porque fiz isso abaixo:



Segunda parte:

Essa parte é pra decidir qual vai ser o tipo de memória do processo, de acordo com a Categoria (1, 2, 3, 4 ou 5) que ele recebeu. Tipo de memória 1 é Usuário e o 2 é Sistema

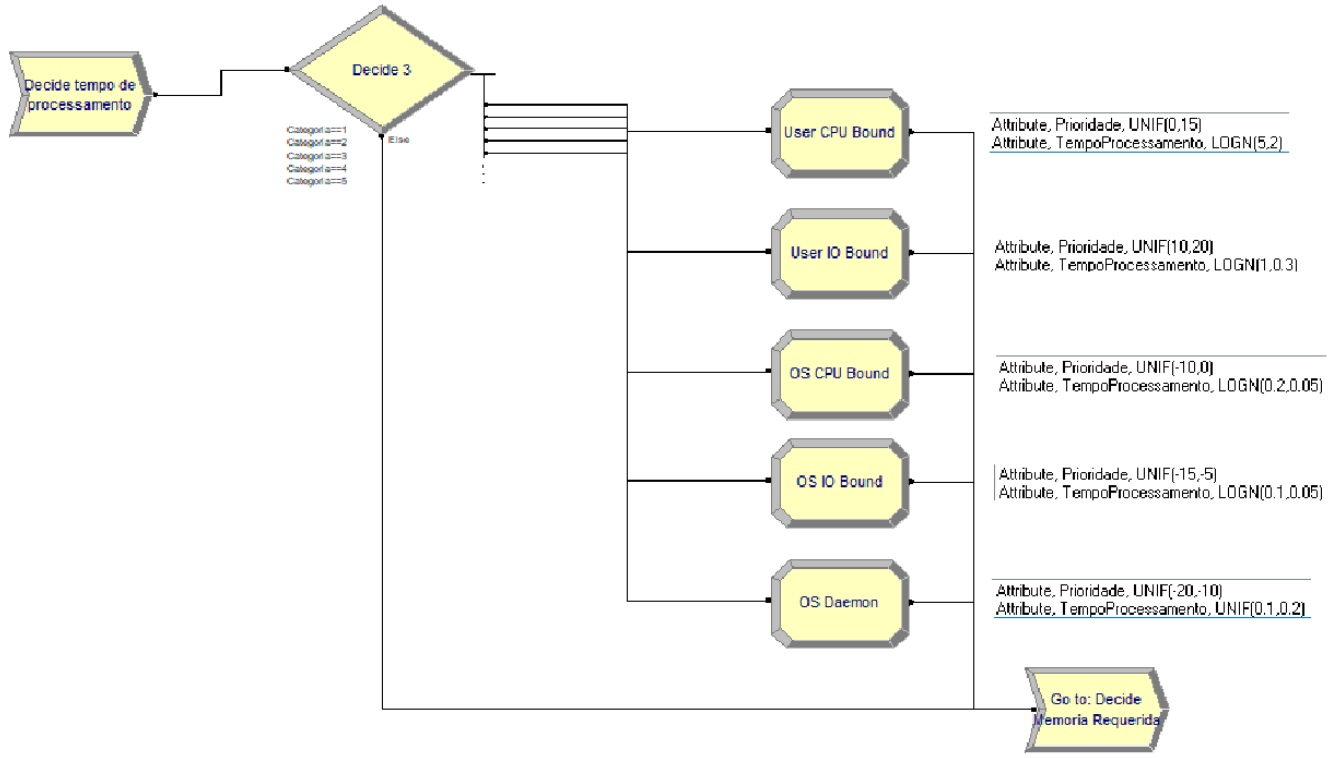


Decide -> Type: N-way by condition, If: Attribute, Named: “Categoria”, Is <=, Value: 2

Assign (caminho True): Add -> Type: Attribute, Attribute Name: “TipoMemoria”, New Value: 1

Assign (caminho False): Add -> Type: Attribute, Attribute Name: “TipoMemoria”, New Value: 2

Terceira parte:



Essa parte é pra decidir o tempo de processamento de cada processo de acordo com a categoria.

Decide -> Type: 5-way by Condition,If: Attribute, Named: "Categoria".
Assign ->
Add -> Type: Attribute, Attribute Name: "TempoProcessamento", New Value: X
Add -> Type: Attribute, Attribute Name: "Prioridade", New Value: Y

X e Y de acordo com a tabela abaixo:

Condição no Decide	TempoProcessamento (X)	Prioridade (Y)
Categoria = 1	LOGN(5,2)	UNIF(0,15)
Categoria = 2	LOGN(1,0.3)	UNIF(10,20)
Categoria = 3	LOGN(0.2,0.05)	UNIF(-10,0)
Categoria = 4	LOGN(0.1,0.05)	UNIF(-15,-5)
Categoria = 5	UNIF(0.1,0.2)	UNIF(-20,-10)

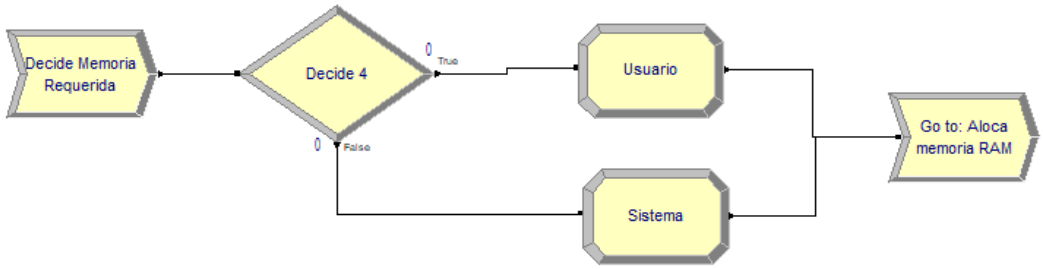
Quarta parte:

Essa parte é pra Decidir a memória requerida, baseado no TipoMemoria.

Decide -> Type: 2-way by condition, If: Attribute, Named: “TipoMemoria”, Is ==, Value: 1

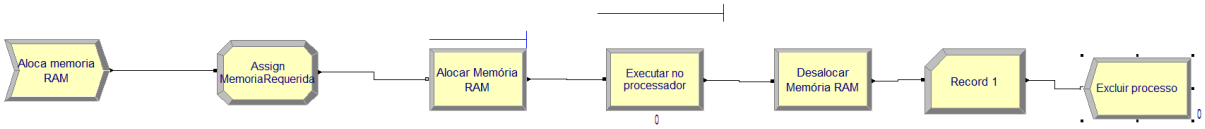
Assign (caminho True): Add -> Type: Attribute, Attribute Name: “MemoriaRequerida”, New Value: AINT(UNIF(10,50))

Assign (caminho False): Add -> Type: Attribute, Attribute Name: “MemoriaRequerida”, New Value: AINT(UNIF(5,20))



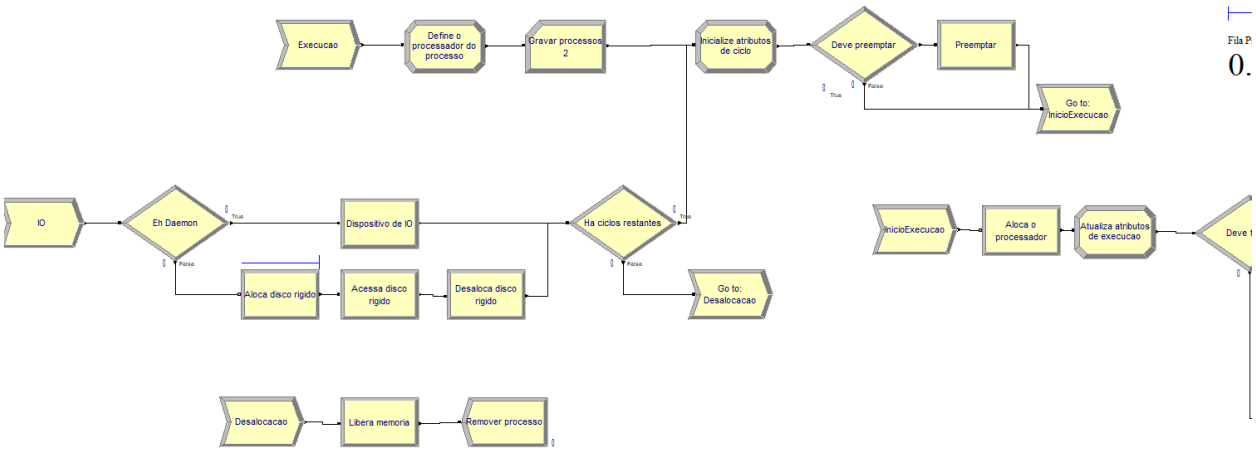
Quinta parte:

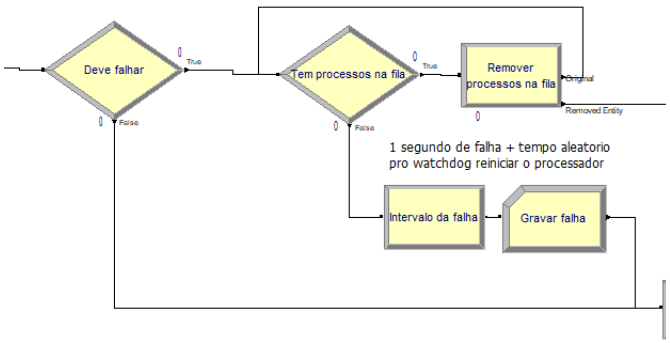
Aqui não precisa mudar nada, fica igual veio do exercício 4.



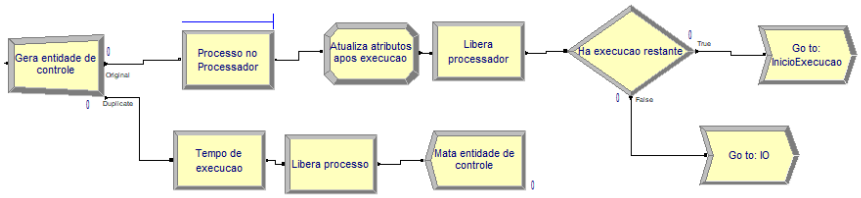
6.
7.
8.
9.
10.

11.


Execucao
Define o processador do processo
Gravar processos 2
Inicialize atributos de ciclo
Deve preemptar
Preemptar
Go to: InicioExecucao
IO
Eh Daemon
Dispositivo de IO
Ha ciclos restantes
Aloca disco rigido
Acessa disco rigido
Desaloca disco rigido
Go to: Desalocacao
InicioExecucao
Aloca o processador
Atualiza atributos de execucao
Desalocacao
Libera memoria
Remover processo


Deve falhar
Tem processos na fila
Remover processos na fila
1 segundo de falha + tempo aleatorio pro watchdog reiniciar o processador
Intervalo da falha
Gravar falha


Gravar processo destruido
Go to: Desalocacao


Gera entidade de controle
Processo no Processador
Atualiza atributos apos execucao
Libera processador
Ha execucao restante
Go to: InicioExecucao
Tempo de execucao
Libera processo
Mata entidade de controle
Go to: IO

```
proximoProcessador   =
MOD(proximoProcessador + 1, 3)*deUsuario + proximoProcessador*(1-deUsuario)

meuProcessador =
(proximoProcessador+2)*deUsuario + 1*(1-deUsuario)

tipoProcesso*100 + meuProcessador

(TNOW - tempoCriacao)*(ciclosTotal==ciclosRestantes) +
tempoExecucao*(ciclosTotal<>ciclosRestantes)

10e5 + (prioridade+21)*10e2 + meuProcessador

Preemptar = idProcessoNoProcessador(meuProcessador, 1)
```