SIMULAÇÃO
FÁBRICA DE SKATES

Realizado por:
José Dias Simões UC2017252705
Leandro Borges Pais UC2017251509

Introdução

No âmbito da cadeira de *Simulação e Computação Científica* desenvolvemos, utilizando o módulo *Salabim* – ambiente de simulação discreto baseado na linguagem *Python* – um programa cujo objetivo é simular o funcionamento de uma fábrica de skates, neste ambiente existem variados recursos (p.e as Prensas, os Cortadores, a Forja, os Trabalhadores, etc…) e também a matéria prima necessária (Pranchas e Rodas).

O objetivo principal desta simulação é estudar conceitos fundamentais de simulação criando, para isso, um modelo funcional que permite observar, neste caso, o normal funcionamento de uma fábrica de skates, o impacto que algumas alterações nos recursos, quantidade de trabalhadores, entre outros, têm na simulação retirando para esse efeito estatísticas que nos permitem de forma mais precisa estudar maneiras de conseguir uma melhor otimização.

Modelo de Simulação

O modelo foi concebido, inicialmente, de modo a cumprir o esperado (de acordo com o enunciado). Foi designado o horário de trabalho das 10:00h ás 18:00h; cumprindo as 8 horas de trabalho durante os 22 dias úteis de um mês.

Os processos na linha de montagem encontram-se encapsulados por etapas que ocupam um dia inteiro. Quando um processo receber uma matéria prima (ou resultado de um processo anterior), deverá verificar, antes de se permitir a saída da fila de chegada e subsequente processamento, se há ou não tempo para concluir a ação associada ao processo. Se não houver tempo este deverá ser reposto na fila; e será feito assim que possível, no dia seguinte. Cada processo dentro da fábrica tem, associado, um tempo de espera específico que simula o funcionamento e processamento da matéria-prima; e tem também incluído um elemento variável que pretende simular pequenas variações no tempo de processamento, como por exemplo, algum fator humano que provoque a alteração no tempo que demora a concluir uma tarefa (notando que esta variação associada ao erro humano é maior do que a variação associada ao erro de uma máquina).

Associados às linhas de produção estão vários armazéns. No modelo, os armazéns estão em proximidade às máquinas, não havendo a necessidade de criar protocolos específicos para o transporte de uma etapa para a outra.

Mais tarde, no desenvolvimento do modelo, foi efetuada a decisão de implementar um sistema de preços e lucros, de modo a ganhar uma visão mais real da simulação. Com este sistema pode, por exemplo, aumentar-se a quantidade de um recurso (para simular a compra de uma nova máquina, ou a contratação de um novo trabalhador (pagando-lhe um mês inteiro de serviço); se tal mudança for justificada por um aumento da produtividade e eficiência da fábrica. Para este efeito foram efetuadas algumas pesquisas relativas a ordenados, para construir um modelo de despesas que permitisse simular, de um modo parametrizável, os custos de manutenção e dos salários de uma fábrica habitual (usando dados médios recolhidos em várias fábricas Europeias, e dados de salário recolhidos no site PORDATA).

Implementação em Salabim

1. Visão Global

Para simular o funcionamento de uma fábrica, e o trabalho realizado por cada uma das componentes da linha de montagem, foram criadas classes que simulam o comportamento real do procedimento. Os trabalhadores, neste caso, são representados por recursos de simulação, que são reservados no momento do início do processo; e o tempo do procedimento é simulado com uma espera no programa; correspondente ao tempo que demoraria a efetuar a tarefa. Os recursos/matérias-primas utilizados na simulação são puramente abstratos, sendo representados no decurso como contadores; decrementando da fase em que estão e incrementando na fase seguinte no momento da transição.

Salienta-se o cuidado tomado para iniciar a simulação às 10:00h do dia, na função que inicializa a cadeia de classes associada a um material.

2. Tempo e Horários

A contabilização e gestão dos tempos na simulação é efetuada com base numa "*timescale*", que é utilizada no cálculo de todos os tempos do programa. É utilizada uma função auxiliar para obter o dia atual, com recurso à variável global "currDay" que obtém o dia atual da simulação com base na escala de tempo definida. São calculados com base no valor 1 da timescale, e correspondem ao valor definido no enunciado, convertido para horas. Temos também penalties que existem para repetir os testes nos semáforos que foram implementados nas fases de montagem e embalagem.

```
# TIMES-------------------------------------------------------------------
# Time it takes for each process to complete successfuly
# (CTIME = completion time)
# Deck operation times
CTIME_press            = 1.66    / LOT_nBoard
CTIME_cut              = 1       / LOT_nBoard
CTIME_finish           = 0.25    / LOT_nBoard
CTIME_paint            = (1 / 3) / LOT_nBoard
# Wheel operation times
CTIME_foundry          = 0.92    / LOT_nWheel
CTIME_machine          = 1       / LOT_nWheel
CTIME_printer          = (1 / 3) / LOT_nWheel
# Assembler times
CTIME_packingDecks     = 0.17    / LOT_nBoardBox
CTIME_packingWheels    = 0.5     / LOT_nWheelBox
CTIME_assembly         = 0.5     / LOT_skates
# Penalties - Fail penalty time
PENALTY_assemble       = 6
PENALTY_packDeck       = 6
PENALTY_packWheel      = 6
# Time - Specifications of time (scale, start, end, etc)
TIME_scale             = 1
TIME_total             = 24 * TIME_scale * MODEL_nDays
TIME_day               = 24 * TIME_scale
TIME_startWork         = 10 * TIME_scale
TIME_endWork           = 18 * TIME_scale
TIME_work              = 6  * TIME_scale
TIME_storageWait       = 16 * TIME_scale
```

3. Ambiente de Simulação

O ambiente de simulação é definido com uma time_unit de "horas", e os recursos são iniciados com capacidades baseadas nas variáveis definidas no início. São definidas *queues* para cada etapa, que serão responsáveis por aguardar a disponibilidade dos recursos, já depois do material ter "chegado" à etapa. A simulação, neste caso, será efetuada durante 24 dias. Definem-se os recursos, com a respetiva capacidade e os simStates (semáforos).

```
#Simulation Environment, Resources, Queues and overall setup ------------------
env = sim.Environment(time_unit = "hours", trace = False)

# ALLOWANCE ---------------------------------------------------------------------
#To use with the sempahores
assembAllowed          = sim.State("assembAllowed")
packWheelAllowed       = sim.State("packWheelAllowed")
packDeckAllowed        = sim.State("packDeckAllowed")

# RESOURCES ---------------------------------------------------------------------
#DECK
press                  = sim.Resource("press", capacity = RSC_nPress)
cutter                 = sim.Resource("cutter", capacity = RSC_nCutter)
finisher               = sim.Resource("finisher", capacity = RSC_nFinisher)
painter                = sim.Resource("painter", capacity = RSC_nPainter)
#WHEEL
foundry                = sim.Resource("foundry", capacity = RSC_nFoundry)
machine                = sim.Resource("machine", capacity = RSC_nMachine)
printer                = sim.Resource("printer", capacity = RSC_nPrinter)
#FINAL
deckPacker             = sim.Resource("deckPacker", capacity = RSC_nPackingDecks)
wheelPacker            = sim.Resource("wheelPacker", capacity = RSC_nPackingWheels)
assembler              = sim.Resource("assembler", capacity = RSC_nSkateAssembler)

# QUEUES ------------------------------------------------------------------------
#DECK
pressQueue             = sim.Queue("pressQueue")
cutterQueue            = sim.Queue("cutterQueue")
finisherQueue          = sim.Queue("finisherQueue")
painterQueue           = sim.Queue("painterQueue")
#WHEEL
foundryQueue           = sim.Queue("foundryQueue")
machineQueue           = sim.Queue("machineQueue")
printerQueue           = sim.Queue("printerQueue")
#FINAL
packDeckQueue          = sim.Queue("packDeckQueue")
packWheelQueue         = sim.Queue("packWheelQueue")
assemblyQueue          = sim.Queue("assemblyQueue")
```

```
# SETUP ------------------------------------------------------------------------#
factory(nDecks = MODEL_nDecks, nWheels = MODEL_nWheels)
env.run(till = TIME_total)
```

4. *Cadeia de Processos (Linha de Montagem)*

No funcionamento do programa, é chamada a função Factory, que instancia as classes que representam os primeiros processos, quer para as rodas, quer para as pranchas. A partir daqui as classes instanciar-se-ão em cadeia, passando por armazéns e pelas várias fases da produção até se chegar à fase de decisão, onde, com ajuda de variáveis auxiliares, as instâncias pares serão encaminhadas para a fase de embalagem, e as instâncias ímpares irão para a montagem de skates. De seguida irá concluir-se o procedimento e será passado para um armazém final o resultado da decisão.

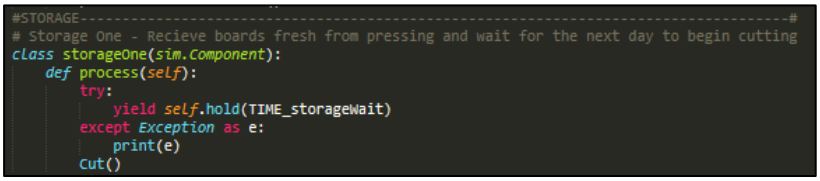
---

5. *Processo, em detalhe*

Em detalhe, o funcionamento de uma etapa (como por exemplo, a prensa) é dividido da seguinte forma:

- Entrada na queue - onde o "material" é posto na fila de espera e efetua *o request do worker*.
- Verificação do tempo restante - que permite saída da queue se houver um recurso disponível e tempo restante no dia de trabalho. Este passo utiliza o Pseudorandom (*sim.Uniform*) para calcular uma variação associada ao recurso, que será adicionada ao tempo de trabalho.
- Saída da respetiva queue e início de trabalho - recurso é reservado e efetua-se um *hold* correspondente à etapa com a adição da variação aleatória.
- Contabilização final - onde o material é ajustado (decrementado da etapa de onde vai sair) e as estatísticas são atualizadas,
- Espera até ao final do dia de trabalho
- Transição, que é especifica para cada etapa, e instancia o procedimento seguinte.

Esta estrutura base é utilizada para todos os processos na linha de montagem; apenas mudando os tempos de espera, a estatística atualizada e o destino final. A passagem efetuada para um Armazém, por exemplo, utiliza, após a "chegada ao armazém" (ou seja, após a instanciação da classe Storage()), uma espera que corresponde ao tempo entre o tempo de chegada ao armazém e o tempo de início do dia seguinte, após a qual será instanciada a classe que corresponde ao processo seguinte.



6. *Semáforos (SimStates)*

Depois da decisão, são utilizados *simStates* para poder verificar se é possível efetuar uma montagem completa de skate (1 prancha e 4 rodas), um pack completo de pranchas (8 pranchas) ou um pack completo de rodas (4 rodas). Este *simState* é verificado e atualizado nas fases de decisão; e o resultado é verificado, com recurso a uma nativa do *Salabim*, que provoca uma espera (controlada por uma variável), cujo final implica a reverificação do *simState*, funcionando este como um semáforo.

```
# ALLOWANCE ------------------------------------------
#To use with the sempahores
assembAllowed           = sim.State("assembAllowed")
packWheelAllowed        = sim.State("packWheelAllowed")
packDeckAllowed         = sim.State("packDeckAllowed")
```

```
# SimState Semaphores - See if there is enough material to pack or assembl
def updateStates():
    #Assembly
    if((storageFinal.nDecks >= 1) and (storageFinal.nWheels >= 4)):
        assembAllowed.set()
    else:
        assembAllowed.reset()

    # Packing Decks
    if(storageFinal.nDecks >= 8):
        packDeckAllowed.set()
    else:
        packDeckAllowed.reset()

    # Packing Wheels
    if(storageFinal.nWheels >= 4):
        packWheelAllowed.set()
    else:
        packWheelAllowed.reset()
```

```
class Assemble(sim.Component):
    def process(self):
        updateStates()
        global PROFIT_GLOBAL
        yield self.wait(assembAllowed, fail_delay = PENALTY_assemble)
```

---

7. *Pequenas Variações (Pseudorandom)*

Além disto, são utilizadas distribuições probabilísticas para simular variação nos tempos de trabalho. No momento da verificação do tempo restante, é utilizada uma distribuição (com os *upper* e *lowerbounds* definidos para a simulação) para criar uma variação, que será adicionada ao tempo. Demonstra-se aqui a simulação dos pequenos atrasos, relativamente aos tempos estáticos que cada etapa demora; das duas maneiras que pode ocorrer (por fonte humana ou artificial). Nos vários testes elaborados para procurar a otimização, foi utilizada a mesma *seed*.

```
# DISTRIBUTIONS ------------------------------------------
# Machine bounds for distribution
MACHINE_low             = 0
MACHINE_high            = 0.5 / 60 / 60
# Human bounds for distribution
HUMAN_low               = 0
HUMAN_high              = 1 / 60 / 60
```

```
#Check if the remaining time is enough to do the job
#if not wait till next day
now = env.now()
getCurrDay()
rem = (currDay * TIME_day) - (TIME_work + now)
extra = sim.Uniform(HUMAN_low, HUMAN_high).sample()
if(rem  + extra < CTIME_assembly):
    now = env.now()
    getCurrDay()
    wait = TIME_storageWait + rem
    try:
        yield self.hold(wait + extra)
```

---

8. *Estrutura de Decisão*

No momento da decisão, é efetuada uma verificação da paridade do material ser encaminhado, e os pares serão encaminhados, de acordo com uma probabilidade (representada por um contador decrescente pré calculado, o LIMIT_skates, calculado com base nas percentagens definidas)

```
class deckBranch(sim.Component):
    def process(self):
        #Gettins global values
        global LIMIT_packDeck
        global LIMIT_skates
        global DECISION_deck
        global FACTORY_expenses
        FACTORY_expenses+=FACTORY_prodDeckCost

        DECISION_deck += 1
        if(((DECISION_deck % 2) == 0) and (LIMIT_packDeck > 0)):
            LIMIT_packDeck -= 1
            packDecks()
        elif(LIMIT_skates > 0):
            LIMIT_skates -= 1
            Assemble()
        else:
            updateStates()
```

9. *Preços, salários e lucros*

No código, são calculados os preços/lucros com base em dados recolhidos na internet. A remoção dos preços ocorre no fim da simulação (simulando o pagamento das contas/salários ao final do mês).
Resta ainda a classe de compra de novas máquinas; que verifica tanto o dinheiro feito atualmente como as quantidades de máquinas atuais, e depois aumenta a capacidade do recurso, modificando a capacidade deles.

```
# PROFIT REDUCTORS ----------------------------------------------------------------
# Machine Workers - Press, Foundry, Machine, Printer, Packing Wheels
# Manufacture Workers - Cutter, Finisher, Painter, Deck Packer, Assembler
# Extra workers - 2 Supervisors, 5 Warehouse Worker
SALARY_machineWorker     = ( 750  / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_manufactureWorker = ( 800  / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_warehouseWorker   = ( 648  / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_painter           = ( 950  / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_forgeOperator     = ( 1100 / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_supervisor        = ( 1400 / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays
SALARY_factoryOwner      = ( 3000 / 30 / 8 )*(TIME_endWork-TIME_startWork)*MODEL_nDays

SALARY_wages    = 0
SALARY_wages    += SALARY_machineWorker         * RSC_nPress
SALARY_wages    += SALARY_manufactureWorker     * RSC_nCutter
SALARY_wages    += SALARY_manufactureWorker     * RSC_nFinisher
SALARY_wages    += SALARY_painter               * RSC_nPainter
SALARY_wages    += SALARY_forgeOperator         * RSC_nFoundry
SALARY_wages    += SALARY_machineWorker         * RSC_nMachine
SALARY_wages    += SALARY_machineWorker         * RSC_nPrinter
SALARY_wages    += SALARY_manufactureWorker     * RSC_nPackingDecks
SALARY_wages    += SALARY_machineWorker         * RSC_nPackingWheels
SALARY_wages    += SALARY_warehouseWorker       * 5
SALARY_wages    += SALARY_supervisor            * 3
SALARY_wages    += SALARY_factoryOwner          * 1

pressPrice      = 25000
machinePrice    = 8500
printerPrice    = 3000
foundryPrice    = 50000

FACTORY_avgExpenses         = 30000
FACTORY_prodDeckCost        = 10
FACTORY_prodWheelCost       = 2
FACTORY_expenses            = 0 + FACTORY_avgExpenses
```

```
class buyPress(sim.Component):
    def process (self):
        global GLOBAL_PROFIT

        global RSC_nPress
        global RSC_nMachine
        global RSC_nPrinter
        global RSC_nFoundry

        global pressPrice
        global machinePrice
        global printerPrice
        global foundryPrice

        # Buy 2 new presses, at most, should the profits be enough
        if (GLOBAL_PROFIT)>(GLOBAL_PROFIT)/0.1 and (GLOBAL_PROFIT)>pressPrice and RSC_nPress<6:
            GLOBAL_PROFIT -= pressPrice
            RSC_nPress+=1
            press.set_capacity(RSC_nPress)
        # Buy some machines to fix the 2 machine lockup
        if (GLOBAL_PROFIT)>(GLOBAL_PROFIT)/0.1 and (GLOBAL_PROFIT)>machinePrice and RSC_nPress>=6:
            if RSC_nMachine<4:
                GLOBAL_PROFIT -= machinePrice
                RSC_nMachine+=1
                press.set_capacity(RSC_nMachine)
        # Biggest expense
        if (GLOBAL_PROFIT)>foundryPrice and RSC_nMachine>=4 and RSC_nFoundry<2:
            GLOBAL_PROFIT -= foundryPrice
            RSC_nFoundry+=1
            press.set_capacity(RSC_nFoundry)
```

Funcionamento do Programa

Ao iniciar a simulação, são definidos todos os recursos e os materiais. Após estar tudo adequado e pronto para começar, as prensas e as forjas são iniciadas, recebendo a ordem através das classes BoardStartup() e WheelStartup(). Estas decrementam o contador inicial de pranchas e rodas, respetivamente, e instanciam as classes Press() e Foundry(), respetivamente.

Como foi descrito acima, as classes irão efetuar o seu procedimento efetuando um request ao respetivo recurso. Irão então verificar se de facto podem efetuar o seu procedimento nesse dia, e se não o puderem fazer, esperarão até ao dia seguinte. A fórmula utilizada têm em conta o dia atual (obtido através do getCurrDay(), que atualiza a variável global), o tempo/duração de um dia; o intervalo entre o fim do trabalho e a meia-noite; e o tempo atual. Através da fórmula (Dia*TempoDia)-(RestanteAposFim+Agora) calcula-se o tempo restante no dia de trabalho. Se este for suficiente, efetua-se o print auxiliar e segue-se, saindo da Queue e efetuando o trabalho (que é representado por um hold).

Após as primeiras fases, os “materiais” são decrementados nestas classes, e incrementados no armazém respetivo (correspondente à primeira etapa). Aqui esperarão até ao dia seguinte, onde irão ser encaminhados para a próxima classe.

Segue-se a segunda etapa, onde se repete o procedimento descrito; desta vez entrando apenas num armazém no final da etapa. Este armazém encaminhará os materiais para a estrutura de decisão.

Em detalhe, a estrutura de decisão assenta em probabilidades predefinidas, que se utilizam para calcular o número de rodas que serão encaminhadas para a montagem, e as de embalagem (efetuando o mesmo processo para as pranchas). Com este cálculo, são criados contadores para estes materiais, que serão decrementados conforme a decisão efetuada na respetiva estrutura. Além deste método, usa-se também um sistema que alterna as pares e inpares, encaminhando as matérias pares para a embalagem, e as inpares para a montagem.

O cálculo dos lucros é efetuado no final da simulação, para simular o pagamento das contas. Nesse momento poderá verificar-se o impacto financeiro eventual (não totalmente implementado ainda) da compra ou contratação de uma nova máquina ou trabalhador.

Análise dos Resultados

*Random_seed=1234567

QUEUE RESULTS	*Resources*	*Length*			*Time*			*Resource Output*	*Resource Output*
	n	mean	std.dev.	median	mean	std.dev.	median	(maximum daily lots)	(maximum daily units)
Press Queue	4	3384	2837	3279	221	132	220	19	456
Cutter Queue	3	40	94	0	2,6	1,5	2,6	24	576
FInisherQueue	1	0,15	0,47	0	0,01	0,008	0,01	32	768
PainterQueue	1	0,38	0,86	0	0,025	0,01	0,025	24	576
PackDeckQueue	2	0,2	3,31	0,4	0,27	0,171	0,277	96	1152
FoundryQueue	1	12271	10202	11934	223	133	221	8	1536
MachineQueue	2	90	261	0	1,64	0,97	1,64	16	3072
PrinterQueue	1	0,02	0,14	0	0	0,001	0	24	4608
PackWheelQueue	1	67	180	0,533	14,6	9	19	16	768
Assembly Queue	2	361	685	0,652	39,43	24	42	32	768

Podemos verificar que, conforme o esperado, não ocorreu produção (de skates ou qualquer pack) durante os primeiros dois dias, pois o tempo não foi suficiente para chegar a qualquer destas etapas. Ao fim do terceiro dia foram registados vários produtos concluídos; em quantidades de acordo com o que seria de esperar. Pode considerar-se que estes dois dias iniciais correspondem a todo o processo de inicialização de máquinas/aquecimento ou *warmup*.

Após 22 dias de simulação, verifica-se que toda a matéria-prima foi processada de acordo com o esperado, e foram produzidas as quantidades necessárias de skates/packs para satisfazer as necessidades especificadas.

Nas filas de espera, podemos verificar que os primeiros processos sofrem de Overflow, pela sua posição na ordem das filas; tanto a Press como a Foundry sofrem de excesso de entradas, e portando, têm um tempo de espera enorme. Como tal, não as consideraremos como bottlenecks, pois num cenário onde o material fosse lentamente adicionado às máquinas, estes tempos de espera não seriam tão grandes.

No entanto, há dois processos que apresentam um tempo de espera muito elevado, nomeadamente, o Machining, e o Cutting. Estes dois processos possuem tempos de processamento muito elevado (80 minutos para o Cutting, e 60 para o Machining), e portanto, podem causar uma interrupção significativa na linha (o que se pode verificar pela quantidade de unidades que eles são capazes de produzir, num dia, no caso máximo). O processo de Painting, apesar de ser mais curto (apenas 20 minutos) sofre por falta de recursos; só tendo um trabalhador; o que o equipara ao Cutting, cuja duração é muito mais extensa; mas é importante salientar que, no caso ótimo, o Painting terá sempre algo na sua fila de espera; enquanto que o Cutting poderá não ter, visto que produz mais do que o passo anterior na linha da fábrica.

Pode-se assumir que, nestes casos, um trabalhador a mais, ou uma máquina a mais poderiam aumentar a velocidade do processo. Neste caso, não se veriam alterações na quantidade de artigos produzidos, pois a simulação de 24 dias faz a produção completa a tempo; mas ver-se-ia alguma alteração a nível dos tempos de espera médios da queue, o que numa situação real teria consequências positivas no fluxo de matérias ao longo da linha.

Aspetos Relevantes

A simulação tem os seguintes aspetos a salientar:

- Utilização de distribuições uniformes para simular atrasos e adiamentos momentâneos, com valores diferentes para humanos e máquinas;

- Utilização de valores modulares, que permite efetuar mais testes face a condições com uma variedade de cenários possíveis;

- Utilização de unidades em vez de lotes, o que implica uma divisão adicional aos tempos de processamento do lote, dividindo o tempo de espera (em horas) pelo tamanho do respetivo lote;

- Impressão informativa dos processos em efeito;

- Sistema de custos e lucros *semi-implementado*, permitindo contabilizar os lucros da fábrica, tendo em conta os diversos custos que se poderão encontrar numa fábrica real;

- Possibilidade de efetuar um "upgrade" modular à fábrica, que corrige progressivamente os sítios com maior possibilidade de criar um bottleneck (como visto na imagem em baixo), utilizando o método *set_capacity*, após alterar o valor global da quantidade de cada recurso, simulando a compra e aquisição de novos recursos..

```
#////////////////////////////////////////////////////////////////////////////////
# MAKEPURCHASE BLOCK ////////////////////////////////////////////////////////////
# BUY A PRESS  ---------------------------------------------------------------#
class buyNewMachine(sim.Component):
    def process (self):
        global GLOBAL_PROFIT

        global RSC_nPress
        global RSC_nMachine
        global RSC_nPrinter
        global RSC_nFoundry

        global pressPrice
        global machinePrice
        global printerPrice
        global foundryPrice

        # Buy 2 new presses, at most, should the profits be enough
        if (GLOBAL_PROFIT)>(GLOBAL_PROFIT)/0.1 and (GLOBAL_PROFIT)>pressPrice and RSC_nPress<6:
            GLOBAL_PROFIT -= pressPrice
            RSC_nPress+=1
            press.set_capacity(RSC_nPress)
        # Buy some machines to fix the 2 machine lockup
        if (GLOBAL_PROFIT)>(GLOBAL_PROFIT)/0.1 and (GLOBAL_PROFIT)>machinePrice and RSC_nPress>=6:
            if RSC_nMachine<4:
                GLOBAL_PROFIT -= machinePrice
                RSC_nMachine+=1
                press.set_capacity(RSC_nMachine)
        # Biggest expense
        if (GLOBAL_PROFIT)>foundryPrice and RSC_nMachine>=4 and RSC_nFoundry<2:
            GLOBAL_PROFIT -= foundryPrice
            RSC_nFoundry+=1
            press.set_capacity(RSC_nFoundry)
```

Propostas de Otimização/Alteração

Ao concluir a análise dos resultados experimentais, verificou-se que a única alteração observável seria uma mudança dos tempos de espera médios nas filas; visto que a simulação não foi construída desde o ínicio com todas as considerações que possuímos atualmente para implementar.

Alterações funcionais que efetuaríamos estariam focadas na abordagem unidade/lote, e faríamos a alteração para o funcionamento e processamento de lote, em vez de processamento unidade-a-unidade; na dimensão abstrata da fábrica (simulando com mais detalhe o funcionamento real dos armazéns, e criando trabalhadores nos armazéns, de modo a criar mais variáveis, que seriam úteis num caso real, onde tivéssemos de gerir uma fábrica destas); na componente aleatória, que poderia ter sido explorada com variáveis como a eventualidade da avaria de uma máquina, diminuindo o recurso, e provocando uma paragem na linha de montagem, ou um trabalhador a ficar doente (que teria uma componente adicional de reduzir os lucros da empresa sem contribuir para a produção, recebendo pagamento de doença), e algumas outras alterações adicionais que poderiam conferir uma maior exploração do cenário de simulação que nos foi dado.

A nível da funcionalidade de comprar máquinas, poderia ter sido melhor implementada, permitindo um cálculo dinâmico por parte do programa para apurar onde valeria mais a pena investir o dinheiro feito até agora, ou até mesmo, considerando ficar em dívida, caso a perspetiva de lucros futuros justificasse tal gasto.

Para além da simulação proposta, seria interessante utilizar o *framework* criado para efetuar a simulação por mais dias, e averiguar com mais cuidado qual o efeito da introdução de novas máquinas, por exemplo, no lucro anual da fábrica, para verificar se de facto seria justificado investir, por exemplo, em 10 prensas novas.