



**REALIZADO POR:**

## **ÍNDICE**

Carlos Prazeres – Nº 9802395  
Joaquim Barata – Nº 9800776

Objectivo.....	3
Metodologia, Implementação e Testes.....	4
Discussão.....	14
Conclusão.....	19

## **OBJECTIVO**

O objectivo deste trabalho é desenvolver um sistema pericial, utilizando a ferramenta Flex, para controlar um semáforo numa passadeira de peões.

O dispositivo do semáforo para controlar o trânsito, inclui, como normalmente, três luzes (verde, amarelo e vermelho), que servem para controlar o trânsito; uma figura de peão, em que a cor alterna entre verde, verde intermitente e vermelho, que serve para controlar a passagem dos peões na passadeira conforme a situação de trânsito e um botão que é utilizado pelos peões para pedir a autorização de atravessar a passadeira.

O funcionamento do semáforo será condicionado por duas situações: a existência de trânsito, que é detectada por um radar na proximidade do semáforo (neste caso é o utilizador que faz este papel) e a existência de peões, que é detectada quando o botão de comando é accionado (neste caso é o utilizador que também faz este papel) .

# METODOLOGIA, IMPLEMENTAÇÃO E TESTES

## Metodologia:

O programa que iremos apresentar será efectuado em Flex, utilizando um conjunto de regras, acções, perguntas e uma relação.

Os Sistemas Periciais são, na sua maioria, baseados em regras de produção do tipo *if-then*. Esta metodologia permite expressar o conhecimento de uma forma simples, intuitiva e fácil de trabalhar.

Uma regra baseia-se em factos conhecidos pelo Sistema Pericial (pré-condições), inferindo novos factos através do seu disparo (o disparo de uma regra é a sua execução, considerando válidas todas as suas pré-condições).

No *flex*, uma das formas de conseguir implementar algo que se quer ver executado, é escrever uma acção. As acções podem ser vistas da mesma forma que um predicado Prolog normal, apenas com uma sintaxe algo diferente e só existe uma definição possível para cada acção. Uma acção é, no fundo, uma colecção de directivas para execução. Uma acção pode ter um número arbitrário de argumentos.

A forma de comunicação com o utilizador que existe em *flex* é a colocação de questões. Estas questões podem ser de vários tipos: **Questões de menu** (em que o utilizador escolhe uma ou mais opções de uma lista que lhe é fornecida (sendo este o caso que iremos utilizar)), **Questões de introdução** (em que o utilizador introduz pelo teclado a informação pedida).

Uma relação não é mais do que um predicado Prolog, já que o próprio Prolog implementa um motor de pesquisa de encadeamento inverso. Como em qualquer predicado Prolog, uma relação pode ter mais do que uma definição (como mais do que uma cláusula no caso do Prolog), sendo tentada em *backtracking*. O objectivo da relação é devolver um valor *booleano* (Verdadeiro ou Falso) consoante esta sucede ou falha.

Ao implementar este sistema pericial iremos utilizar a base do conhecimento (conjunto de conhecimentos sobre um domínio específico) e um motor de inferência (que desencadeia o processo de raciocínio), sendo este utilizado de forma a funcionar pela resolução de conflito ‘first come first served’. Para além disso é de salientar que, através da utilização de perguntas e regras, o utilizador terá de interagir com o programa para simular as funções do radar e do botão de comando do semáforo.

Os valores que iremos atribuir às variáveis serão sempre transferidos para a consola pelo referido programa, de forma a permitir o acompanhamento no ecran do funcionamento do sistema pericial perante a evolução das situações de transito e de peões.

Uma vez que o Flex não imprime de forma imediata os valores no ecran, iremos utilizar a instrução ttyflush (que força o Flex a imprimir os valores de imediato) e a instrução beep(0,t) (que cria um atraso de t segundos, para abrandar a impressão dos valores).

## Implementação:

Em baixo apresentamos todo o código implementado para gerir este sistema pericial :

```
ruleset PROJECTO_DE_INTELIGENCIA_ARTIFICIAL_2
contains all rules;
select rule using first come first served;
update ruleset by removing each selected rule;
initiate by doing restart.

group situacao_do_transito existe_veiculo,nao_existe_veiculo.
group situacao_do_peao existe_peao,nao_existe_peao.

%% ESTA REGRA INICIALIZA O SEMAFORO NO CASO DE NAO HAVER NENHUMA INFORMACAO %%
%% SOBRE A SITUACAO DO TRANSITO. %%
%% ESTA REGRA PEDE AO UTILIZADOR A INFORMACAO DO TRANSITO NA AUSENCIA DESTA %%
%% E EM SEGUIDA REPORTA NA CONSOLA O VALOR ESCOLHIDO. %%

rule semaforo1
if semaforo is unknown then
iniciar_sistema.

%% ESTA REGRA PEDE AO UTILIZADOR A INFORMACAO DO TRANSITO NA AUSENCIA DESTA %%
%% E EM SEGUIDA REPORTA NA CONSOLA O VALOR ESCOLHIDO. %%

rule pergunta_transito
if transitam_veiculos is unknown then
ask transitam_veiculos and
transitam := transitam_veiculos and
nl and
write (' SITUACAO DO VEICULO - ') and
write (transitam) and
nl and
ttyflush and
beep (0,1000) and
ttyflush and
beep (0,1000).

%% ESTA REGRA PERMITE AO UTILIZADOR DEFINIR A SITUACAO DO PEAO, CONSIDERANDO %%
%% QUE EXISTEM DUAS OPCOES DE ESCOLHA POSSIVEIS E POR FIM REPORTA NA CONSOLA %%
%% A RESPECTIVA SITUACAO DO PEAO. %%

rule pergunta_peao
if situacao_peao is unknown then
```

```

ask situacao_peao and
peao := situacao_peao and
nl and
write ('          SITUACAO DO PEAO - ') and
write (peao) and
nl and
ttyflush and
beep (0,1000) and
ttyflush and
beep (0,1000)

```

```

%% ESTA REGRA PRODUZ UMA MENSAGEM NA CONSOLA PARA O PEAO NAO ATRAVESSAR
%% NAO ATRAVESSAR QUANDO ESTE ESTA A ESPERA E O TRANSITO APROXIMA-SE.

```

rule semaforo2

```

if peao is existe_peao and transitam is existe_veiculo then
  nl and
  nl and
  write (** PERIGO !!! ESPERE, NAO ATRAVESSE! **) and
  nl and
  nl and
  ttyflush and
  beep (0,1000).

```

```

%% ESTA REGRA EFECTUA A ACCAO parar_transito QUANDO EXISTE UM PEAO A
%% ESPERA.

```

rule semaforo3

```

if peao is existe_peao then parar_transito.

```

```

%% ESTA REGRA AUTORIZA O PEAO A ATRAVESSAR QUANDO O TRANSITO JA ESTA
%% PARADO, APRESENTADO UMA MENSAGEM NA CONSOLA; EM SEGUIDA POE O PEAO
%% A ATRAVESSAR ATRAVES DA ACCAO peao_a_atraversa; POR FIM ACTUALIZA
%% A SITUACAO DO PEAO CONSIDERANDO QUE JA ATRAVESSOU A PASSADEIRA E
%% REPORTA NA CONSOLA A NOVA SITUACAO DO PEAO.

```

rule semaforo4

```

if peao is existe_peao and transitam is esta_parado then
  nl and
  nl and
  write ('-> O PEAO PODE ATRAVESSAR.') and
  nl and
  ttyflush and
  beep (0,1000) and
  nl and
  peao := peao_a_atraversar and
  peao_atraversa.

```

```

%% ESTA REGRA SERVE PARA FAZER O RESET DO SEMAFORO QUANDO O PEAO ACABA
%% DE ATRAVESSAR A PASSADEIRA.

```

rule semaforo5

```

if peao is peao_atravesou then reset.

```

```

%% ESTA PERGUNTA PERMITE AO UTILIZADOR DEFINIR A SITUACAO DO TRANSITO.

```

```
question transitam_veiculos
  Escolha a situacao do transito ;
  choose from situacao_do_transito
  because Tem de escolher a situacao de transito!!!.
```

```
%% ESTA PERGUNTA PERMITE AO UTILIZADOR DEFINIR A SITUACAO DOS PEOES.
%%
```

```
question situacao_peao
  Escolha a situacao do peao ;
  choose from situacao_do_peao
  because Tem de escolher a situacao do peao!!!.
```

```
%% ESTA ACCAO INICIA O SISTEMA PERICIAL.
%%
```

```
action iniciar
  do invoke ruleset PROJECTO_DE_INTELIGENCIA_ARTIFICIAL_2.
```

```
%% ESTA ACCAO INICIALIZA O SEMAFORO FICANDO ESTE COM A COR VERDE
%% E A COR DA FIGURA DO PEAO VERMELHO E REPORTA OS VALORES PARA
%% A CONSOLA ATRAVES DA ACCAO reporta_valores.
%%
```

```
action iniciar_sistema;
  do
    semaforo := verde and
    figura := vermelho and
    reporta_valores.
```

```
%% ESTA ACCAO REPORTA OS VALORES PARA A CONSOLA.
%%
```

```
action reporta_valores;
  do
    nl and
    nl and
    write ('A SITUACAO ACTUAL DOS SEMAFOROS E :') and
    nl and
    write ('-----') and
    nl and
    nl and
    write (' Os veiculos estao com o semaforo ') and
    write (semaforo) and
    nl and
    write (' O peao esta com o semaforo ') and
    write (figura) and
    nl and
    ttyflush and
    beep (0,1000).
```

```
%% ESTA ACCAO TORNA A LUZ DO SEMAFORO AMARELO E REPORTA A COR NA
%% CONSOLA; EM SEGUIDA MUDA A COR DO SEMAFORO PARA VERMELHO E
%% REPORTA A COR NA CONSOLA; POR FIM PARA O TRANSITO E REPORTA A
%% NOVA SITUACAO DO TRANSITO NA CONSOLA.
%%
```

```
action parar_transito;
```

```

do
  nl and
  semaforo := amarelo and
  write ( ' Os veiculos estao com o semaforo ') and
  write (semaforo) and
  nl and
  ttyflush and
  beep (0,1000) and
  semaforo := vermelho and
  write ( ' Os veiculos estao com o semaforo ') and
  write (semaforo) and
  nl and
  ttyflush and
  beep (0,1000) and
  transitam := esta_parado and
  nl and
  write ( ' SITUACAO DO VEICULO - ') and
  write (transitam) and
  nl and
  ttyflush and
  beep (0,1000).

```

```

%% ESTA ACCAO TORNA A COR DA FIGURA VERDE E REPORTA A NOVA COR DA
%% NA CONSOLA; EM SEGUIDA SIMULA A TRAVESSIA DA PASSADEIRA PELO
%% PEAO, ATRIBUINDO A VARIAVEL PEAO O VALOR ADEQUADO E REPORTA A
%% NOVA SITUACAO DO PEAO NA CONSOLA.

```

```

action peao_atravessa;
do
  figura := verde and
  reporta_valores and
  peao := peao_a_atravesar and
  nl and
  write ( ' SITUACAO DO PEAO - ') and
  write (peao) and
  nl and
  ttyflush and
  beep (0,1000) and
  peao := peao_atravessou and
  nl and
  write ( ' SITUACAO DO PEAO - ') and
  write (peao) and
  nl and
  ttyflush and
  beep (0,1000).

```

```

%% ESTA ACCAO FAZ O RESET DO SEMAFORO, OU SEJA, POE A COR DA FIGURA
%% DO SEMAFORO A PISCAR EM VERDE E REPORTA NA CONSOLA O ESTADO
%% ACTUAL DO SEMAFORO; EM SEGUIDA MUDA A COR DA FIGURA DO SEMAFORO
%% PARA VERMELHO E REPORTA NA CONSOLA O NOVO ESTADO DA FIGURA; EM
%% SEGUIDA POE O SEMAFORO A PISCAR EM AMARELO E REPORTA NA CONSOLA
%% O NOVO ESTADO DO SEMAFORO; DEPOIS MUDA A COR DO SEMAFORO PARA
%% VERDE E REPORTA NA CONSOLA A NOVA COR DO SEMAFORO; POR FIM POE
%% O TRANSITO A ANDAR E REPORTA NA CONSOLA.

```

```

action reset;
do
  figura := verde_pisca and
  reporta_valores and
  figura := vermelho and

```

```

reporta_valores and
semaforo := amarelo_pisca and
reporta_valores and
semaforo := verde and
reporta_valores and
transitam := arrancou and
nl and
write (' SITUACAO DO VEICULO -') and
write (transitam) and
nl and
ttyflush and
beep (0,1000).

```

```

%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%% VERIFICA SE A PASSADEIRA ESTA SEGURA QUANDO O PEAO ATRAVESSA.      %%
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%

```

```

relation passadeira_segura
if figura is verde and semaforo is vermelho.

```

## Testes:

Em seguida iremos apresentar os teste efectuados para todas as hipóteses existentes:

### PRIMEIRO CASO:

#### Existe\_Veículo e Existe\_Peão

A SITUACAO ACTUAL DOS SEMAFOROS E :

-----

Os veiculos estao com o semaforo verde  
O peao esta com o semaforo vermelho

SITUACAO DO VEICULO - existe\_veiculo

SITUACAO DO PEAO - existe\_peao

**\*\* PERIGO !!! ESPERE, NAO ATRAVESSE! \*\***

Os veiculos estao com o semaforo amarelo  
Os veiculos estao com o semaforo vermelho

SITUACAO DO VEICULO - esta\_parado

-> O PEAO PODE ATRAVESSAR.

A SITUACAO ACTUAL DOS SEMAFOROS E :

-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo verde

SITUACAO DO PEAO - peao\_a\_atravesar

SITUACAO DO PEAO - peao\_atravessou

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo verde\_pisca

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo vermelho

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo amarelo\_pisca  
O peao esta com o semaforo vermelho

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo verde  
O peao esta com o semaforo vermelho

SITUACAO DO VEICULO -arrancou

## SEGUNDO CASO:

### Existe\_Veículo e Não\_Existe\_Peão

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo verde  
O peao esta com o semaforo vermelho

SITUACAO DO VEICULO - existe\_veiculo

SITUACAO DO PEAO - nao\_existe\_peao

## TERCEIRO CASO:

### Não\_Existe\_Veículo e Existe\_Peão

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo verde  
O peao esta com o semaforo vermelho

SITUACAO DO VEICULO - nao\_existe\_veiculo

SITUACAO DO PEAO - existe\_peao

Os veiculos estao com o semaforo amarelo  
Os veiculos estao com o semaforo vermelho

SITUACAO DO VEICULO - esta\_parado

-> O PEAO PODE ATRAVESSAR.

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo verde

SITUACAO DO PEAO - peao\_a\_atravesar

SITUACAO DO PEAO - peao\_atravessou

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo verde\_pisca

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo vermelho  
O peao esta com o semaforo vermelho

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo amarelo\_pisca  
O peao esta com o semaforo vermelho

A SITUACAO ACTUAL DOS SEMAFOROS E :  
-----

Os veiculos estao com o semaforo verde  
O peao esta com o semaforo vermelho

SITUACAO DO VEICULO –arrancou

## QUARTO CASO:

### Não\_Existe\_Veículo e Não\_Existe\_Peão

A SITUACAO ACTUAL DOS SEMAFOROS E :

-----

Os veiculos estao com o semaforo verde

O peao esta com o semaforo vermelho

SITUACAO DO VEICULO - nao\_existe\_veiculo

SITUACAO DO PEAO - nao\_existe\_peao

## DISCUSSÃO

Em torno dos resultados obtidos pelo sistema pericial podemos afirmar que estes foram apresentados de uma forma bastante explícita, uma vez que este apresentou todos os passos de forma correcta até chegar a uma solução final.

O Flex apresenta várias vantagens no desenvolvimento de sistemas periciais. Este recorre a um formalismo específico de linguagem, muito similar à linguagem natural em Inglês, permitindo assim uma programação declarativa. Para além disso este tipo de programação permite utilizar um conhecimento do tipo simbólico, permitindo assim a criação ou alteração do conhecimento.

A programação convencional não é adequada para implementar sistemas periciais porque a base do conhecimento pode ser composta por centenas ou milhares de regras, tornando bastante difícil a tarefa de colocação das regras de forma sequencial no programa.

Num sistema pericial, as regras e os factos são formulados livremente de forma ad-hoc na base do conhecimento. Cabe ao motor de inferência escolher de forma autónoma, as regras e os factos na sequência certa para desenvolver o processo de raciocínio até encontrar a solução.

Apresentação de três sistemas automáticos de controlo de tráfego:

### SCOOT

Este sistema foi desenvolvido pela Simens, TRL e pela Peek Traffic. O Scoot envia instruções para o equipamento “on-street”, utilizando linhas telefónicas especializadas. Estas instruções são interpretadas e operadas pelo equipamento do sinal de tráfego que se encontra na berma da estrada.

O equipamento reporta as instruções para o computador central confirmando a aceitação da instrução, ou reporta uma condição errada. Este sistema obtém informação acerca do fluxo de tráfego através de detectores. Estes detectores são normalmente requeridos em toda a ligação, em que a sua localização, que é muito importante, é na parte superior do fluxo próximo da ligação. Quando os veículos passam pelo detector, o Scoot converte a informação para dentro das “link profile units”. Esta unidade é usada pelo Scoot para fazer os cálculos. “Cyclic flow profiles” da unidade referida são a toda a hora construídos por cada ligação.

## **UTOPIA – SPOT Traffic optimization**

O Utopia-Spot resolve os problemas do tráfego através de sinais que controlam os entroncamentos numa área específica. É baseada em cada entroncamento recebendo informação relativamente ao fluxo de tráfego actual e envia para os outros entroncamentos na área .

O sistema vai otimizar a rede em múltiplos níveis e dá a localização óptima do tempo necessário em cada intercepção, em tempo real baseado nas condições actuais do tráfego.

## **EC-Track traffic control**

O EC-Track é um sistema “server-based” para controlar os sinais de tráfego. O desenho modular do sistema facilita a configuração para todas as especificações de utilizadores dos pequenos sistemas com um “server-based Workstation” para sistemas muito grandes com funções de um sistema, num número de servidores e “PC-based Workstations”. Continuando a comunicação com controladores de sistema estes asseguram que o EC-Track é conhecedor do status operacional de cada sinal de tráfego a qualquer momento. O EC-Track guarda a informação do tráfego através de detectores que existem no sistema e apresenta os resultados gráficamente ou em forma de tabela, apresentando detalhadamente os problemas operacionais, faltas e erros que encontra, guardando depois numa base-de-dados para estes serem consultados por outros programas.

Estes três sistemas foram retirados do site:

<http://www.peek-traffic.co.uk/urban.htm>

Apresentação e discussão dos resultados obtidos pela relação *passadeira\_segura*:

Em primeiro lugar definimos uma relação `passadeira_segura` no nosso programa, a qual apresentamos em seguida:

```
%% VERIFICA SE A PASSADEIRA ESTA SEGURA QUANDO O PEAO ATRAVESSA. %%
```

```
relation passadeira_segura
    if figura is verde and semaforo is vermelho.
```

Como o próprio nome indica, esta relação vai permitir verificar se a passadeira naquele momento se encontra segura para o peão atravessar.

Efectuamos uma consulta através da consola para saber se a passadeira estava segura e a resposta foi negativa, isto porque o semáforo do veículo nessa altura encontrava-se verde e o do peão encontrava-se vermelho.



Em seguida lançamos novamente o programa e na altura em que a figura do semáforo mudou para verde, interrompemos a inferência e efectuamos a mesma consulta verificando que a resposta foi positiva, isto porque nessa altura os veículos tinham o semáforo vermelho e o peão tinha o semáforo verde, logo a passadeira estava segura.

```
WIN-PROLOG - [Console]
File Edit Search Run Options Flex Window Help
i ?- iniciar.

A SITUACAO ACTUAL DOS SEMAFOROS E :

Os veiculos estao com o semaforo verde
O peao esta com o semaforo vermelho
SITUACAO DO VEICULO - existe_veiculo
SITUACAO DO PEAO - existe_peao

*** PERIGO !!! ESPERE, NAO ATRAVESSE! ***

Os veiculos estao com o semaforo amarelo
Os veiculos estao com o semaforo vermelho
SITUACAO DO VEICULO - esta_parado

-> O PEAO PODE ATRAVESSAR.

A SITUACAO ACTUAL DOS SEMAFOROS E :

Os veiculos estao com o semaforo vermelho
O peao esta com o semaforo verde

Aborted
i ?- passadeira_segura.
yes
i ?-
```

## CONCLUSÃO

A partir dos resultados obtidos, podemos afirmar que a maioria dos objectivos propostos foram alcançados com êxito (isto porque não conseguimos colocar o sistema a trabalhar com situações de incerteza), não encontrando na fase de testes qualquer anomalia com o sistema pericial. É desta forma que podemos dizer que este projecto serviu para nos mostrar, que os sistemas periciais desenvolvidos por esta ferramenta (Flex) que é de fácil utilização e compreensão, podem ser por vezes mais práticos e eficazes que os sistemas desenvolvidos pelas linguagens de programação convencionais.

A Inteligência Artificial é hoje um domínio do conhecimento cada vez mais ‘na moda’. Dela fala-se, escreve-se, ouve-se falar, lê-se. Mas saberemos nós o que é na verdade esta ciência, o que estuda, que aplicações práticas tem? A verdade é que muitas vezes os nossos conhecimentos sobre Inteligência Artificial (I.A.) não vão além do “isso tem qualquer coisa a ver com computadores, não é?”.

Hoje os grandes problemas colocam-se no sentido de formar e formalizar conceitos e ampliar o poder de expressão das linguagens de programação.