Projeto de Introdução à Arquitetura de Computadores

LEIC

IST-Taguspark

RoboTenis – Robô para treino de ténis

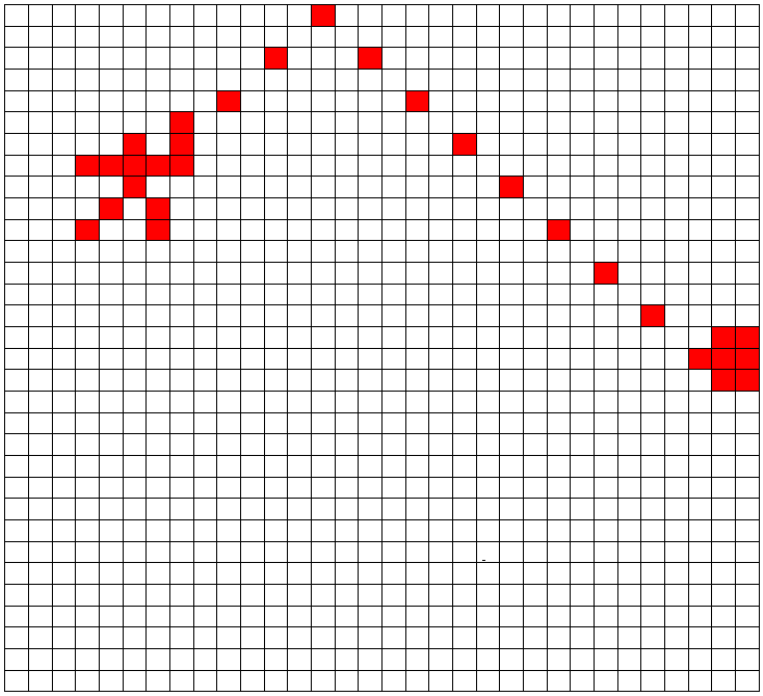
2014/2015

1 – Objetivos

Este projeto pretende exercitar os fundamentos da área de Arquitetura de Computadores, nomeadamente a programação em linguagem *assembly*, os periféricos e as interrupções.

O objetivo deste projeto consiste em concretizar um jogo de simulação de treino de ténis. Um robô vai disparando bolas em diversas direções e o tenista terá de as apanhar com a raquete. Se conseguir ganha um ponto por cada bola, caso contrário perde um ponto por cada bola que deixar escapar. O jogo tem duas especificidades face a um treino real: as bolas fazem ricochete nas paredes (topo e fundo do ecrã) e as bolas desaparecem quando são apanhadas pela raquete. O jogo termina quando se quiser e a pontuação final, mostrada nuns displays de 7 segmentos, medem a qualidade do tenista.

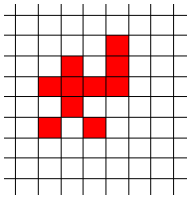
A figura seguinte exemplifica o cenário, com o tenista (e raquete), o robô do lado direito e uma possível trajetória de bola, mostrada aqui em sucessivas posições apenas para ilustração.



2 – Especificação do projeto

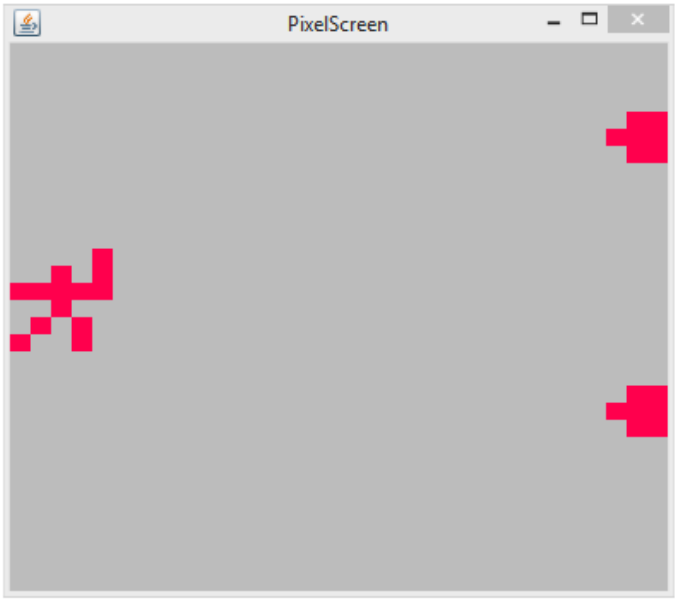
O ecrã de interação com o jogador tem 32 x 32 pixels, tantos quantas as quadrículas da figura anterior. Há três tipos de objetos no ecrã:

- Tenista, com a sua raquete. Neste enunciado exemplifica-se um boneco de 5 x 6 pixels, incluindo uma raquete de 1 x 3 pixels. Mas deve ser fácil mudar o tamanho e aspeto do boneco, sem alterar as instruções do programa (deve ser especificado por uma tabela de pixels). A figura seguinte ilustra um boneco com apenas 4 x 5 pixels. Pode usar o boneco que entender (mesmo maior), mas a raquete deve ser sempre 1 x 3 pixels, na posição vertical. Só há um tenista no jogo;



- Bola, que consiste em apenas um pixel;
- Robô, que dispara as bolas. Neste enunciado exemplifica-se um boneco de 3 x 3 pixels, mas as suas dimensões e aspeto devem também ser definidos por uma tabela de pixels e não diretamente por instruções. Pode começar por testar com apenas um robô, mas na versão final devem existir dois robôs.

A figura seguinte mostra um possível aspeto do ecrã, com o tenista e os dois robôs nas suas posições iniciais. Pode usar outras posições iniciais (embora cada robô só tenha duas posições possíveis).



O tenista pode mover-se em qualquer direção (cima, baixo, esquerda, direita e direções a 45°), sob comando do jogador. O objetivo é posicionar a raquete de modo a apanhar a bola. Não pode sair do ecrã nem invadir as colunas da direita, onde os robôs se movimentam.

Os robôs deslocam-se verticalmente apenas, cada um na sua metade (superior ou inferior) do ecrã, sempre encostados à direita do ecrã. Cada robô só pode estacionar em duas posições, a cerca de 1/3 e 2/3 da altura da sua metade.

De cada vez que a bola de um robô é apanhada ou perdida (bem como no início do jogo), cada robô deve:

- Mover-se para a outra posição de estacionamento dentro da sua metade do ecrã (sempre à direita);
- Disparar uma bola numa das seguintes direções, com igual probabilidade:
  - Horizontal (da direita para a esquerda);
  - 45° ascendente (tal como ilustrado na primeira figura);
  - 45° descendente.

A escolha da direção deve ser baseada num gerador de números aleatórios (ver secção 4, Estratégia de Implementação).

Cada robô deve mover-se a uma dada velocidade (um pixel de cada vez) e não de forma instantânea. Também as bolas se devem mover a uma dada velocidade no ecrã, podendo fazer e ricochete. O objetivo é simular os movimentos reais.

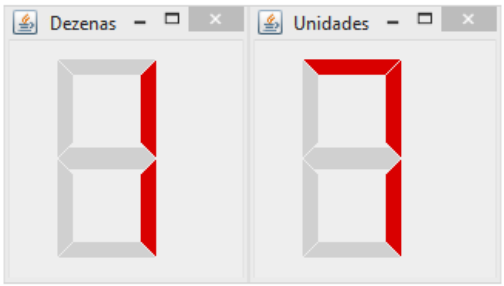
A velocidade do movimento dos robôs deve ser independente da das bolas. As velocidades podem ser ajustadas a seu gosto, para afinar a jogabilidade.

Cada robô só dispara nova bola após a bola que disparou por último ter sido perdida ou apanhada. Os robôs tomam ações de forma independente um do outro.

Bola em movimento a 45° que bata no topo ou no fundo do ecrã deve mudar a direção do movimento (de ascendente para descendente, ou vice-versa, sempre a 45°), tal como ilustrado pela primeira figura.

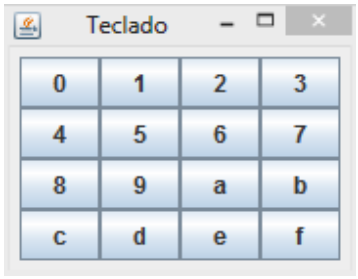
Considera-se que uma bola é apanhada se no seu percurso a sua posição no ecrã coincidir com um dos pixels da raquete. Considera-se perdida se a sua coluna ultrapassar (para a esquerda) a coluna da raquete. Sendo apanhada ou perdida, desaparece do ecrã e a pontuação é atualizada.

Existem dois displays de 7 segmentos, que mostram o conteúdo de um contador de dois dígitos decimais (hexadecimal não), que é incrementado sempre que o tenista apanha uma bola e decrementado sempre que perde (deixa passar) uma. O valor inicial (e mínimo) do contador é zero, com máximo 99. A figura seguinte ilustra um possível valor desse contador.



Se o tenista apanhar uma bola com a raquete a pelo menos 1/4 do ecrã a contar da esquerda, simulando uma subida à rede, obtém 3 pontos em vez de apenas 1.

O comando do jogo é feito por um teclado, tal como o da figura seguinte:



São necessários os seguintes comandos:

- Movimentar o tenista nas 8 direções (cima, baixo, esquerda, direita e direções a 45°);
- Começar o jogo (ou recomeçar, em qualquer altura, mesmo durante um jogo);
- Terminar o jogo (para recomeçar, carrega-se na tecla de Começar).

A escolha de que tecla faz o quê é à escolha do grupo. Teclas sem função devem ser ignoradas quando premidas.

A funcionalidade de cada tecla é executada quando essa tecla é premida, mas só depois de ser largada é que pode funcionar novamente.

Um jogo terminado deve mostrar o ecrã em branco (todos os pixels apagados) ou outro ecrã específico à sua escolha (exemplo: GAME OVER). No entanto, o valor do contador das bolas apanhadas deve ser mantido e só colocado a zero quando novo jogo for iniciado.

Juntamente com este documento, encontrará um ficheiro Excel (**ecra.xlsx**), que reproduz os pixels do ecrã. Pode usá-lo para desenhar novos bonecos para o tenista e robôs, bem como algumas letras grandes (exemplo: GAME OVER), e traduzi-los para uma tabela, de forma a produzir um jogo mais personalizado. Este aspeto é opcional.

NOTA - Cada grupo é livre de mudar as especificações do jogo, desde que não seja para ficar mais simples e melhore o jogo em si. A criatividade e a demonstração do domínio da tecnologia são sempre valorizadas!

3 – Faseamento do projeto

O projeto decorrerá em duas fases, versão intermédia e final. A versão intermédia vale 20% da nota do projeto.

Na semana de 10 a 14 de novembro de 2014, deverá mostrar ao docente do seu turno de laboratório, na aula respetiva, uma versão intermédia que cumpra (pelo menos) os passos 1 a 3 da estratégia de implementação descrita na secção seguinte. Tal implica:

- Submeter no Fenix (Projeto IAC 2014-15 - versão intermédia) um ficheiro (grupoXX.asm, em que XX é o número do grupo) com o código do programa, tal como ele estiver na altura, até ao dia 9 de novembro, às 23h59;
- Sugere-se criar uma cópia da versão atual do código, de modo a compilar e executar a funcionalidade pedida, limpando eventual "lixo" e coisas temporárias. Organização do código e comentários serão avaliados, tal como na versão final;
- Mostrar ao docente o código a funcionar, no laboratório. O docente poderá fazer algumas perguntas sobre o programa e dar algumas sugestões.

IMPORTANTE – Não se esqueça de identificar o código com o número do grupo e número e nome dos alunos que participaram na construção do programa (em comentários, logo no início da listagem).

A versão final do projeto deverá ser entregue até ao dia 3 de dezembro de 2014, 23h59. A entrega, a submeter no Fenix (Projeto IAC 2014-15 - versão final) deve consistir de um zip (grupoXX.zip, em que XX é o número do grupo) com dois ficheiros:

- Um relatório (modelo já disponível no Fenix);
- O código, pronto para ser carregado no simulador e executado.

4 – Estratégia de implementação

Alguns dos guiões de laboratório contêm objetivos parciais a atingir, em termos de desenvolvimento do projeto. Tente cumpri-los, de forma a garantir que o projeto estará concluído nas datas de entrega, quer a versão inicial quer a versão final.

Devem ser usados processos cooperativos para suportar as diversas ações do jogo, aparentemente simultâneas. Recomendam-se os seguintes processos:

- Teclado (varrimento e leitura das teclas, tal como descrito no guião do laboratório 5);
- Tenista (para controlar os movimentos do tenista);
- Robô (para controlar as ações dos robôs);
- Bola (para controlar o movimento das bolas e atualizar o contador e os displays);
- Gerador (para gerar um número aleatório, que os robôs usam para escolher uma das suas quatro ações possíveis);
- Controlo (para tratar das teclas de começar e terminar).

Como ordem geral de implementação, recomenda-se a seguinte:

1. Rotinas de ecrã (desenhar/apagar um pixel numa dada linha e coluna, desenhar/apagar tenista/robô – represente os objetos pelas coordenadas de um determinado pixel (canto superior esquerdo, por exemplo, e desenhe-os relativamente às coordenadas desse pixel);
2. Teclado (um varrimento de cada vez, inserido num ciclo principal onde as rotinas que implementam processos vão ser chamadas);

3. Tenista (desenho do boneco e raquete, com deslocamentos de um pixel por cada tecla carregada no teclado);
4. Processos cooperativos (organização das rotinas preparada para o ciclo de processos);
5. Bola (pode começar só com uma, com deslocamento de um pixel numa dada direção quando se carrega numa dada tecla; mais tarde, de forma autónoma com interrupções);
6. Robô (pode começar só um, com deslocamento de um pixel num dado sentido vertical quando se carrega numa dada tecla; mais tarde, de forma autónoma com interrupções);
7. Interrupções (há duas, uma para animar o movimento das bolas e outra para animar o movimento dos robôs);
8. Controlo;
9. Resto das especificações.

Para cada processo, recomenda-se:

- Um estado 0, de inicialização. Assim, (re)começar um jogo é pôr todos os processos no estado 0, em que cada um inicializa as suas próprias variáveis. Fica mais modular;
- Planeie os estados (situações estáveis) em que cada processo poderá estar. O processamento (decisões a tomar, ações a executar) é feito ao transitar entre estados;
- Veja que variáveis são necessárias para manter a informação relativa a cada processo, entre invocações sucessivas (posição, direção, modo, etc.)
- Quer o processo Bola quer o processo Robô trata de dois objetos independentes. Para cada um deles, recomenda-se que se invoque duas vezes a mesma rotina, passando como argumento o endereço da zona de dados onde está o estado relativo a cada objeto (ou o nº da bola ou robô que depois indexa uma tabela com esses dados).

O processo Gerador é mais simples e pode ser simplesmente um contador (variável de 16 bits) que é incrementado em cada iteração do ciclo de processos. Quando um robô precisa de um número aleatório entre 0 e 2 (para escolher uma das três direções possíveis), basta ler esse contador, usar apenas os seus dois bits menos significativos e ignorar se der 3. Como o ciclo de processos executa muitas vezes durante a execução de uma ação de um robô, esses dois bits parecerão aleatórios.

Pode invocar o processo Gerador mais do que uma vez no mesmo ciclo de processos (por exemplo, entre a invocação de um robô e outro) para aumentar a aleatoriedade das ações.

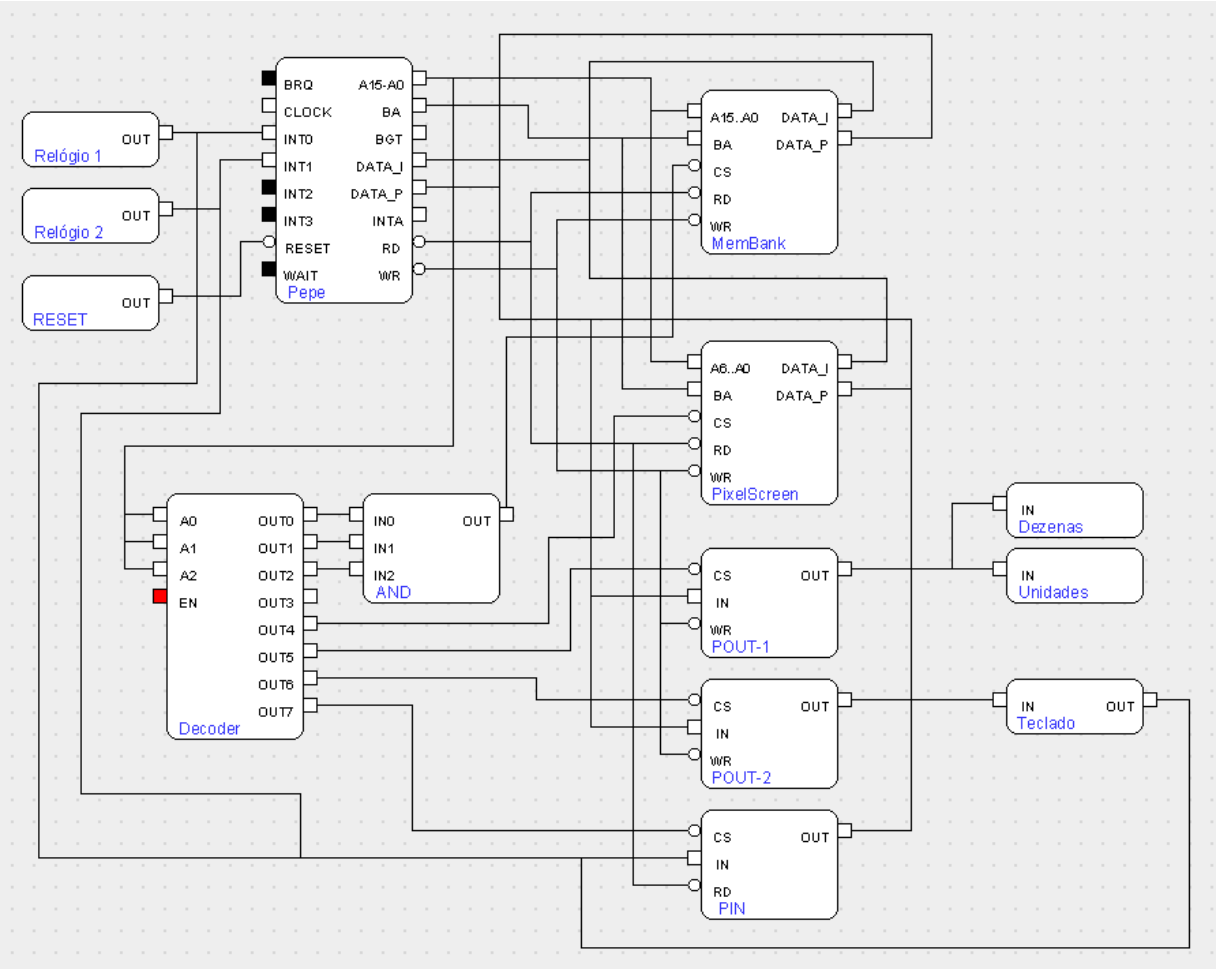
Finalmente:

- Faça PUSH e POP de todos os registos que use numa rotina e não constituam valores de saída. É muito fácil não reparar que um dado registo é alterado durante um CALL, causando erros que podem ser difíceis de depurar. Atenção ao emparelhamento dos PUSHs e POPs;
- Vá testando todas as rotinas que fizer e quando as alterar. É muito mais difícil descobrir um bug num programa já complexo e ainda não testado;
- Estruture bem o programa, com zona de dados no início e rotinas auxiliares de implementação de cada processo junto a eles;

- Não coloque constantes numéricas (com algumas exceções, como 0 ou 1) pelo meio do código. Defina constantes simbólicas e use-as depois no programa;
- Produza comentários abundantes, não se esquecendo de cabeçalhos para as rotinas com descrição, registos de entrada e de saída;
- Não duplique código (com copy-paste). Use uma rotina com parâmetros para contemplar os diversos casos em que o comportamento correspondente é usado.

5 – Implementação

A figura seguinte mostra o circuito a usar (fornecido, ficheiro **jogo.cmod**).



Podem observar-se os seguintes módulos, cujo painel de controlo deverá ser aberto em execução (modo Simulação):

- Relógio 1 – Relógio de tempo real, para ser usado como base para a temporização do movimento das bolas. Em versões intermédias poderá ser útil lê-lo num ciclo de instruções. Por isso, também liga ao bit 4 do periférico de entrada PIN;

- Relógio 2 – Relógio de tempo real, para ser usado como base para a temporização do movimento dos robôs. Em versões intermédias pode ser útil lê-lo num ciclo de instruções. Por isso, também liga ao bit 5 do periférico de entrada PIN;
- Matriz de pixels (PixelScreen) – ecrã de 32 x 32 pixels. É acedido como se fosse uma memória de 128 bytes (4 bytes em cada linha, 32 linhas). Atenção, que o pixel mais à esquerda em cada byte (conjunto de 8 colunas em cada linha) corresponde ao bit mais significativo desse byte. Um bit a 1 corresponde a um pixel a vermelho, a 0 um pixel a cinzento;
- Dois displays de 7 segmentos, ligados aos bits 7-4 e 3-0 do periférico POUT-1, para mostrar o contador de bolas apanhadas;
- Teclado, de 4 x 4 botões, com 4 bits ligados ao periférico POUT-2 e 4 bits ligados ao periférico PIN (bits 3-0). A deteção de qual botão está carregado é feita por varrimento.

O mapa de endereços (em que os dispositivos podem ser acedidos pelo PEPE) é o seguinte:

Dispositivo	**Endereços**
RAM (MemBank)	0000H a 5FFFH
PixelScreen	8000H a 807FH
POUT-1 (periférico de saída de 8 bits)	0A000H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H

Notas **MUITO IMPORTANTES**:

- Os periféricos de 8 bits e as tabelas com STRING devem ser acedidos com a instrução MOVB. As variáveis definidas com WORD (que são de 16 bits) devem ser acedidas com MOV;
- A quantidade de informação mínima a escrever no PixelScreen é de um byte. Por isso, para alterar o valor de um pixel, tem primeiro de se ler o byte em que ele está localizado, alterar o bit correspondente a esse pixel e escrever de novo no mesmo byte;
- Os relógios que ligam às interrupções do PEPE e o teclado partilham o mesmo periférico de entrada, PIN (bits 4-5 e 3-0, respetivamente). Por isso, terá de usar uma máscara ou outra forma para isolar os bits que pretender, após ler este periférico;
- As rotinas de interrupção param o programa principal enquanto estiverem a executar. Por isso, devem apenas atualizar uma variáveis em memória, que os processos sensíveis a essas interrupções devem estar a ler. O processamento deve ser feito pelos processos e não pelas rotinas de interrupção, cuja única missão é assinalar que houve uma interrupção.