Projeto de Introdução à Arquitetura de Computadores

LEIC

IST-Taguspark

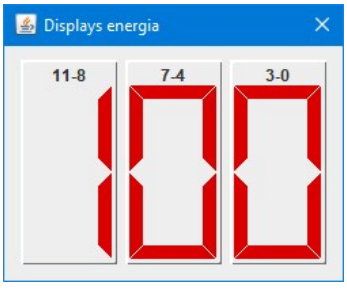
Nave guardiã mineradora

2020/2021

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 uma nave espacial que deve minerar asteroides e ao mesmo tempo destruir outras naves, inimigas. A interface consiste num ecrã, um teclado para controlo do jogo e um conjunto de displays, para mostrar a energia da nave.



O módulo MediaCenter do simulador possui variadas capacidades multimédia, permitindo definir imagens de fundo, reproduzir sons e vídeos, vários planos de imagens construídas pixel a pixel, um cenário frontal para letreiros, etc. O guião de laboratório 4 fornece mais detalhes sobre este módulo.

O teclado permite, fazendo clique em algumas das teclas, fazer o comando do jogo (*start*, *pause* e *stop*, para além de controlar a posição da nave).

Os displays permitem exibir a energia atual da nave, que vai variando com o tempo.

Cada estado do jogo (inicial, a jogar, em pausa, terminado, etc.) deve ter um cenário ou vídeo de fundo diferente, ou um letreiro (cenário frontal, uma imagem com letras e fundo transparente), de forma a dar uma indicação visual do estado do jogo. A figura da página anterior ilustra um possível cenário de entrada, em que o utilizador tem de premir a tecla C para iniciar o jogo. Também pode ser um vídeo, com as letras sobrepostas por meio de um cenário frontal.

A ideia genérica do jogo é a seguinte:

- A nave está no fundo ecrã e só pode movimentar-se (por teclas) para a esquerda e para a direita;
- Do meio do topo de ecrã "nascem" OVNIs (Objetos Viajantes Não Identificados), vindos de muito longe, pelo que quando aparecem são apenas um pixel cinzento. Estes ovnis vão descendo, ou na vertical ou nas duas oblíquas (a 45°), aumentando de tamanho à medida que se vão aproximando da nave. No segundo tamanho (2 x 2 pixels) ainda estão distantes e são cinzentos e indistintos, mas a partir daí mudam de forma e cor consoante sejam asteroides (verdes) ou naves inimigas (vermelhas), tal como ilustrado pela figura seguinte;
- O objetivo da nave é destruir as naves inimigas (disparando um míssil), para defender o seu planeta, e minerar os asteroides (permitindo que eles colidam consigo);

- A colisão de um míssil com um ovni (asteroide ou nave inimiga) implica a sua destruição e do ovni, com um efeito de explosão. Atenção, que o míssil tem um alcance limitado (não pode assim atingir ovnis distantes);
- As naves inimigas não destruídas e asteroides não minerados perdem-se pelo fundo do ecrã. Sempre que uma nave inimiga é destruída, um asteroide é minerado ou qualquer deles se perde no fundo, um novo nasce no topo, com tipo (asteroide ou nave inimiga) e direção escolhidas de forma pseudo-aleatória (25% asteroide, 75% nave inimiga; 3 direções equiprováveis);
- A nave tem uma energia inicial. Essa energia vai-se gastando ao longo do tempo, só pela nave estar em funcionamento. Disparar um míssil gasta energia adicional. No entanto, minerar um asteroide e destruir uma nave inimiga aumenta essa energia;
- O jogo termina se a nave colidir com uma nave inimiga ou se a energia chegar a zero. O objetivo do jogo consiste assim em aguentar a nave durante tanto tempo quanto possível, obtendo energia dos asteroides e naves destruídas e evitando colidir com uma nave inimiga;
- O jogador deve ter hipótese de fazer pausa ao jogo, bem como recomeçar após pausa, terminar o jogo em qualquer altura e começar novo jogo após o anterior terminar.

2 – Detalhes do projeto

2.1 – Teclado e controlo do jogo

O jogo é controlado pelo utilizador por meio de teclas num teclado (atuado por clique do rato), tal como o da figura seguinte:

A atribuição de teclas a comandos é livre e ao seu gosto. Uma possível atribuição é a seguinte:

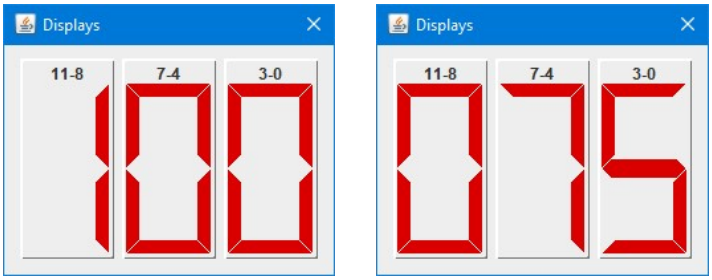
- Teclas 0 e 2: movimentar a nave para a esquerda e direita, respetivamente;
- Tecla 1: disparar o míssil;
- Tecla C: começar o jogo (deve reiniciar a energia da nave a 100%);
- Tecla D: suspender/continuar o jogo;
- Tecla E: terminar o jogo (deve manter visível a energia final da nave).

IMPORTANTE - Com exceção das teclas de movimentar a nave, cada tecla dever executar apenas um comando. Para novo comando, mesmo que igual, tem de se largar a tecla e carregar de novo. O movimento da nave pode ser "contínuo", enquanto a tecla estiver carregada.

O guião de laboratório 3 ensina a trabalhar com o teclado.

2.2 – Displays e energia da nave

Existem três displays de 7 segmentos, que devem mostrar a energia da nave em percentagem do valor inicial, em cada instante (em decimal, o que implica conversão a partir dos valores hexadecimais que o PEPE usa). As figuras seguintes ilustram a energia inicial e um possível valor após o jogo evoluir:



A energia começa com 100 (%) e deve ser decrementada de 5 % de 3 em 3 segundos.

Por cada colisão da nave com um asteroide, a energia aumenta de 10%, como resultado da mineração.

Por cada míssil disparado, a energia diminui de 5%. Por cada nave destruída, a energia aumenta de 5% (resultado da recolha da energia produzida pela explosão). Destruir um asteroide dá explosão, mas não radia energia.

Se a energia chegar a 0%, o jogo termina. A imagem de cenário de jogo terminado deve ser diferente da resultante de uma colisão com uma nave inimiga.

Se o jogo terminar por esta colisão, a energia que tinha na altura deve manter-se. Só se um novo jogo for iniciado a energia deve ser reposta a 100%.

O guião de laboratório 3 ensina a trabalhar com os displays.

2.3 – Ecrã, cenários, sons e bonecos

O ecrã de interação com o jogador tem 32 x 64 pixels (linhas x colunas). Cada pixel pode ter uma cor diferente, em 4 componentes (Alpha, Red, Green e Blue, ou ARGB), todas com 4 bits (valores sem sinal, de 0 a 15). A primeira componente define a opacidade (0 é totalmente transparente, 15 totalmente opaco). O pixel pode estar ligado (com a cor definida pelas 4 componentes) ou desligado (com tudo a zero, caso em que não se vê pois fica transparente).

Por trás deste ecrã está a imagem ou vídeo de fundo (a que se vê nas figuras anteriores), que tem uma resolução bem superior (embora deva ter um fator de forma semelhante, retangular, para que não apareça de forma distorcida).

É possível alterar a imagem/vídeo de fundo através do programa do PEPE. Por isso, espera-se que use um cenário diferente para cada situação. Nem sempre uma imagem tem de variar. Pode editá-la,

colocando texto que indique qual a situação (pausa, por exemplo), gerando assim variantes da mesma imagem. Ou pode usar um cenário frontal, com uma imagem com as letras e fundo transparente, que aparece à frente da imagem ou vídeo de fundo.

Não é fornecida nenhuma imagem nem nenhum vídeo para cenários, mas existem inúmeras imagens adequadas que pode obter de forma gratuita na Internet para este uso pessoal. O design multimédia fica ao seu gosto. A figura seguinte ilustra um possível cenário no caso de colisão da nave com uma nave inimiga.

Também devem existir efeitos sonoros, que mais uma vez podem ser obtidos na Internet. Ficheiros de som pequenos é o ideal. O módulo MediaCenter permite, no entanto, restringir a zona de reprodução de um som (ou de um vídeo), caso haja necessidade.

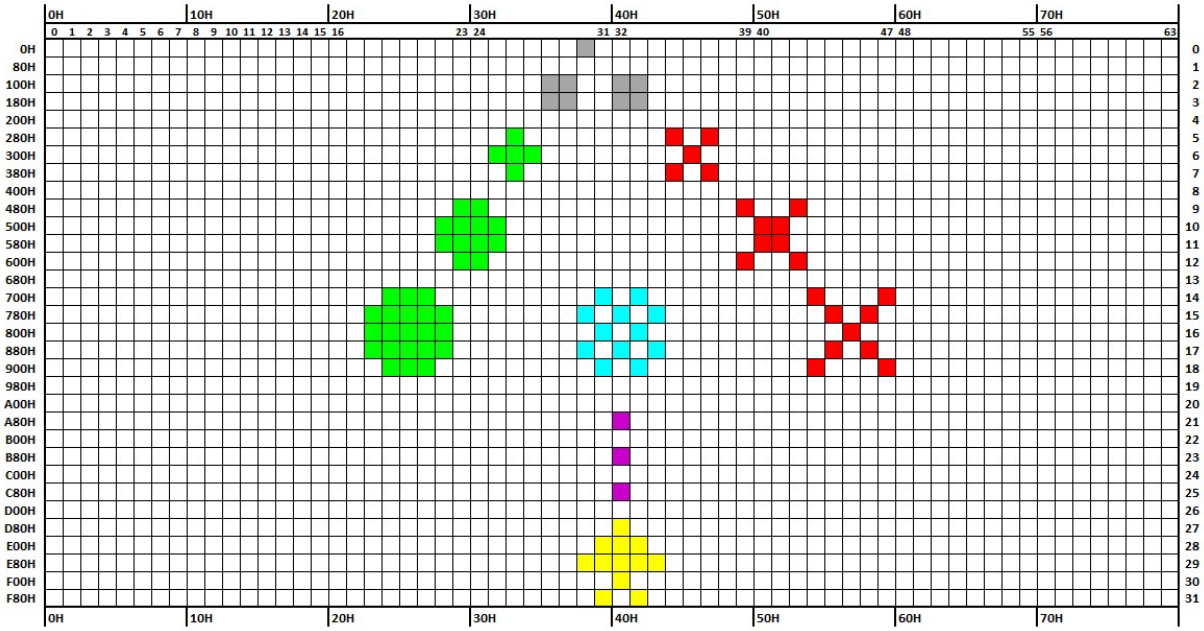
Estes efeitos sonoros devem ser reproduzidos quando é disparado um míssil, um ovni é destruído, um asteroide é minerado, o jogo termina por falta de energia e por colisão, etc. Isto é um jogo!

FUNDAMENTAL – Depois de definir os ficheiros de imagem, vídeo e som que quiser acrescentar ao módulo MediaCenter, deve salvar o circuito num diretório que contenha esses ficheiros (ou diretamente ou num subdiretório desse). Esta é a única forma de os ficheiros serem incluídos no ficheiro de descrição do circuito de forma portável. Caso contrário, é guardado o "path" absoluto do ficheiro, e depois não funciona noutro computador (nomeadamente, no do docente que vai avaliar o projeto!).

Desenhar um boneco no ecrã (à frente do cenário) é desenhar os seus pixels, com cor ou desligado (transparente), em posições (linha e coluna) adjacentes do ecrã, de acordo com a forma definida para esse objeto. Toma-se um dos pixels do objeto (canto superior esquerdo, por exemplo) como indicador da posição no ecrã desse objeto e todos os restantes pixels desse objeto são desenhados em posições relativas a esse pixel de referência.

Mover um objeto é apagá-lo na sua posição atual, mudar a sua posição e desenhá-lo na nova posição. O efeito visual é o objeto parecer mover-se.

O ficheiro Excel **ecrã-32x64.xlsx**, que é fornecido, simula as quadrículas do ecrã e mostra que o ecrã é na realidade uma memória (note os endereços de cada linha no lado esquerdo), em que cada pixel é uma palavra (2 bytes) dessa memória, com a cor do pixel. Há 64 pixels, ou 128 bytes, por cada linha do ecrã.



Pode usar este ficheiro para planear o tamanho e forma dos seus ovnis e da nave, pois não têm de ser iguais ao mostrado nas figuras anteriores.

Pretendem-se 5 tamanhos para os ovnis, desde 1 x 1 até 5 x 5 (ou algo semelhante). Cada ovni move-se um pixel de cada vez. De 3 em 3 movimentos, aumenta de tamanho, até ao máximo. Nos dois tamanhos iniciais recomenda-se uma cor neutra, até com alguma transparência, para simular a distância.

A forma e a cor dos objetos não têm de ser como ilustrado. A figura é apenas uma sugestão.

Cada pixel pode ser desenhado escrevendo diretamente na memória do ecrã ou por meio de um comando. Escolha o seu método. O guião de laboratório 4 tem os detalhes de como usar o módulo MediaCenter.

Nos seus movimentos, a nave não deve sair do ecrã. Chegada aos limites, não deve avançar mais.

O míssil (1 x 1) sai da nave no seu topo e deve ter um alcance limitado (sugere-se 12 movimentos no máximo), ao fim do qual se extingue se nada atingir. Só se pode disparar um míssil depois do anterior desaparecer (por alcance ou colisão).

Pretendem-se vários ovnis (sugerem-se 4), mas pode começar apenas por um. O módulo MediaCenter possibilita que cada um deles seja desenhado num ecrã diferente (com todos os ecrãs sobrepostos). Isto

tem a vantagem que os ovnis que seguem o mesmo caminho se sobreponham de forma graciosa (como se fossem janelas diferentes num sistema operativo), em vez de um destruir o desenho do anterior.

O guião de laboratório 4 ensina a trabalhar com o módulo MediaCenter, em termos de pixels, cenários e sons.

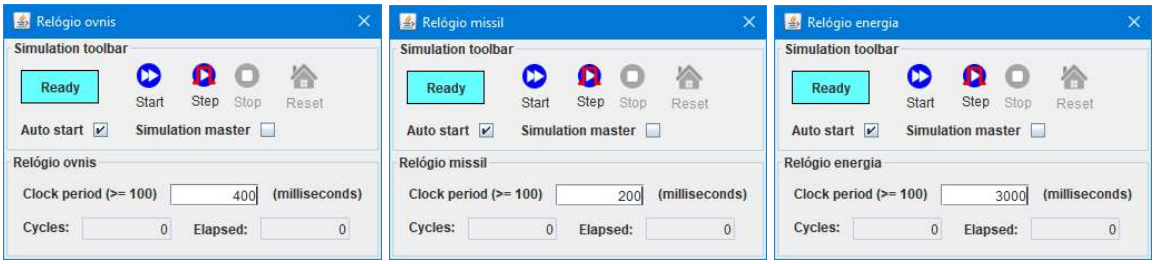
2.4 – Temporizações

A evolução do jogo requer 3 temporizações diferentes:

- Movimento dos ovnis (período de 400 milissegundos);
- Movimento do míssil (período de 200 milissegundos);
- Decréscimo periódico da energia da nave (período de 3000 milissegundos, ou 3 segundos).

Os períodos indicados, que marcam o ritmo a que cada um dos eventos ocorre, são gerados por 3 relógios de tempo real, que geram um sinal de um bit que varia periodicamente entre 0 e 1, com um dado período. Sem o tempo real marcado por estes relógios, o jogo evoluiria de forma muito mais rápida e de forma não controlável, dependendo apenas da velocidade de processamento do computador que executa o simulador.

Estes relógios estão incluídos no circuito usado neste jogo e estão pré-programados com estes tempos, mas pode alterá-los para melhorar a jogabilidade do jogo ou fazer testes.



O arranque dos relógios é automático, pelo que nem precisa de abrir as janelas de simulação deles.

O guião de laboratório 5 ensina a utilizar relógios para marcação de temporizações.

2.5 – Escolhas pseudo-aleatórias

Quando um ovni “nasce”, deve decidir se vai ser asteroide ou nave inimiga, tomando a cor forma respetiva, e qual a direção que deve tomar (vertical ou oblíqua, para a esquerda ou para a direita).

Pretende-se que cerca de 25% de ovnis sejam asteroides e 75% sejam naves inimigas. A direção que toma cada ovni, independentemente do seu tipo deve ter 33% de hipóteses para cada possibilidade.

Como o PEPE não tem mecanismos para gerar valores aleatórios, usa-se um truque simples: incrementa-se um contador de forma muito frequente (num sítio onde o programa passe muitas vezes). Quando se pretender fazer uma escolha, lê-se o valor que o contador tiver nessa altura e eliminam-se todos os bits

desse valor exceto os dois bits de menor peso, dando assim um valor entre 0 e 3. Desta forma, o 0, por exemplo, surge cerca de 25% das vezes. O valor é razoavelmente aleatório porque não se consegue controlar o ritmo a que o contador é incrementado. Da mesma forma, obtendo o resto da divisão do valor do contador por 3 (instrução MOD do PEPE) consegue-se um de 3 valores equiprováveis.

O contador funciona assim como um gerador de números aleatórios (simples, mas suficiente para o efeito aqui pretendido).

3 – Faseamento do projeto

O projeto decorrerá em duas fases, versão intermédia e final.

IMPORTANTE – Não se esqueça de identificar os ficheiros de código, .asm, em comentários, logo no início da listagem, com:

- o número do grupo;
- o número e nome dos alunos que participaram no desenvolvimento do programa.

Versão intermédia:

- Vale 25% da nota do projeto (ou 10% da nota final de IAC);
- Deve ser submetida no Fenix (Projeto IAC 2020-21 - versão intermédia) até às **23h59** do dia **6 de novembro de 2020** através de um ficheiro (**grupoXX.asm**, em que XX é o número do grupo) com o código, pronto para ser carregado no simulador e executado. Sugere-se criar uma cópia da versão mais recente do código, limpando eventual "lixo" e coisas temporárias, de modo a compilar e executar a funcionalidade pedida. Organização do código e comentários serão avaliados, tal como na versão final;
- **IMPORTANTE** - Use o circuito do projeto, **projeto.cir** (e não o de qualquer guião de laboratório). Note que o periférico dos displays é de 16 bits (deve usar MOV) e não de 8 bits (MOVB);
- **Deve cumprir os seguintes objetivos**:
    - O teclado deve reagir a 4 teclas diferentes (escolha quais), que comandam o valor de um contador decimal que deve ser mostrado nos displays (3 dígitos);
    - Duas das teclas incrementam e decrementam, respetivamente, o contador de uma unidade em regime contínuo na tecla (incrementa/decrementa repetidamente o contador enquanto a tecla estiver a ser premida). Use um ciclo simples para atrasar a velocidade de evolução do contador;
    - As outras duas teclas incrementam e decrementam, respetivamente, o contador de uma unidade por cada clique na tecla (para incrementar/decrementar de novo é preciso largar o botão do rato);
    - O contador deve contar em decimal. Internamente, deve contar em hexadecimal, num registo, que é o que o PEPE sabe fazer, mas para mostrar o valor no display deve primeiro ser convertido para decimal (ou seja, um número hexadecimal cujos dígitos sejam o do número decimal que corresponde ao valor do contador);

- Se já souber usar rotinas, pode usá-las, nomeadamente um rotina para converter o registo do contador para decimal e mostrar o valor nos displays. Se não quiser usar rotinas, faça simplesmente *copy-paste* do código respetivo tantas vezes quantas precisar. Este *copy-paste* só é admissível numa versão intermédia...

Versão final:

- Vale 75% da nota do projeto (ou 30% da nota final);
- **Deve cumprir todas as especificações do enunciado**;
- Deve ser submetida no Fenix (Projeto IAC 2020-21 - versão final) até às **23h59** do dia **11 de dezembro de 2019**, através de um ficheiro (**grupoXX.zip**, em que XX é o número do grupo) com os seguintes ficheiros:
  - Um ficheiro **grupoXX.pdf**, relatório de formato livre, mas com a seguinte informação (juntamente com este enunciado, é fornecido um possível modelo de relatório):
    - Identificação do número do grupo, números de aluno e nomes;
    - Definições relevantes, se tiverem feito algo diferente do que o enunciado pede ou indica (teclas diferentes, funcionalidade a mais, etc.);
    - Indicação concreta das funcionalidades pedidas que o código enviado NÃO satisfaz;
    - Eventuais outros comentários ou conclusões.
  - Um ficheiro **grupoXX.asm** com o código, pronto para ser carregado no simulador e executado (também deve ter o número do grupo, números de aluno e nomes);
  - Todos os ficheiros de imagem, vídeo e som usados no módulo "MediaCenter";
  - Um ficheiro **projetoXX.cir** com o circuito do projeto, mas guardado depois de definir no módulo "MediaCenter" todos os ficheiros de imagem, vídeo e som usados. Não se esqueça que estes ficheiros devem estar guardados no mesmo diretório do circuito, ou num subdiretório deste;
- Nas últimas semanas de aulas, de 14 a 18 de dezembro 2020, e de 4 a 8 janeiro de 2021, não haverá aulas de laboratório. Essas semanas estão reservadas para discussão do projeto com o docente das aulas de laboratório (durante o horário normal de laboratório do seu grupo). A data e hora aproximada em que o seu grupo terá a discussão serão anunciados no Fenix, após a data de entrega da versão final do projeto. Esta discussão será remota, por Discord.

4 – Estratégia de implementação

Os guiões de laboratório estão alinhados com 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 na versão intermédia, quer na versão final.

Devem ser usados processos cooperativos (guião de laboratório 6) 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 3);
- Nave (para controlar o movimento e o disparo do míssil);
- Ovni (para controlar as ações e evolução de cada um dos ovnis, incluindo verificação de colisão com o míssil ou com a nave. Note que vários ovnis correspondem apenas a instâncias diferentes do mesmo processo, ou várias chamadas à mesma rotina com um parâmetro que indique o número do ovni);
- Míssil (para controlar a evolução do míssil no espaço e alcance);
- Gerador (inclui um contador para gerar um número pseudoaleatório, usado para escolher o tipo e direção de um ovni);
- Controlo (para tratar das teclas de começar, suspender/continuar e terminar o jogo).

Como ordem geral de implementação do projeto, recomenda-se a seguinte estratégia (pode naturalmente adotar outra):

1. Teclado e displays (cobertos pela versão intermédia);
2. Rotinas de ecrã, para desenhar/apagar:
   - um pixel numa dada linha e coluna (de 0 a 31 e 0 a 63, respetivamente);
   - um objeto genérico, descrito por uma tabela que inclua a sua largura, altura e a cor ARGB de cada um dos seus pixels (pode também ter apenas uma palavra para dar a cor de todos os pixels do objeto e depois cada pixel pode ser descrito apenas por 0, desligado, ou 1, com a cor definida). Use um desses pixels, por exemplo o canto superior esquerdo do objeto, como referência da posição do objeto (linha e coluna) e desenhe os restantes pixels relativamente às coordenadas desse pixel de referência;
3. Desenho e movimentos da nave, de acordo com a tecla carregada;
4. Um só ovni (tem de definir tabelas com a representação de cada um dos vários tamanhos e tipos dos ovnis e uma tabela para escolher o boneco adequado consoante o tamanho e tipo do ovni);
5. Processos cooperativos (organização das rotinas preparada para o ciclo de processos, começando por converter em processos o código que já tem para o teclado e para a Nave);
6. Processos Controlo e Gerador;
7. Adaptação do processo Nave para incluir o gasto periódico de energia, por meio de uma interrupção;
8. Movimento dos ovnis, por meio de uma interrupção;
9. Processo Míssil, com o movimento linear regulado por meio de uma interrupção;
10. Deteção de colisão entre um ovni e a nave ou o míssil;

11. Resto das especificações, incluindo extensão para mais do que um ovni. Esta extensão tem algumas complicações, pelo que é melhor ter um só ovni a funcionar do que tentar logo vários e correr o risco de não conseguir nenhum.

IMPORTANTE:

- As rotinas de interrupção param o programa principal enquanto estiverem a executar. Por isso, devem apenas atualizar variáveis em memória, que os processos regulados por essas interrupções devem ir lendo. O processamento do jogo deve ser feito pelos processos e não pelas rotinas de interrupção, cuja única missão é assinalar que houve uma interrupção;
- Se usar valores negativos (por exemplo, -1 para somar à coluna de um ovni para ele se deslocar para a esquerda), as variáveis correspondentes devem ser de 16 bits (declaradas com WORD, e não STRING).

Para cada processo, recomenda-se:

- Um estado 0, de inicialização. Assim, cada processo é responsável por inicializar as suas próprias variáveis. O programa fica mais bem organizado e 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 de cada boneco no ecrã, modo, etc.).

O processo Gerador pode ser simplesmente um contador (variável de 16 bits) que é incrementado em cada iteração do ciclo de processos. Quando for preciso um número pseudoaleatório (para escolher o tipo e direção de um ovni), basta ler esse contador e obter apenas os bits menos significativos (por meio do resto da divisão desse contador por N, com a instrução MOD, obtendo-se um valor entre 0 e N-1 ou por meio de AND com uma máscara).

Como o ciclo de processos se repete muitas vezes durante a iteração dos vários processos, esse valor parecerá aleatório. Quando tiver vários ovnis, pode inclusivamente invocar o Gerador entre cada duas invocações do processo Ovni, para aumentar a aleatoriedade.

Tenha ainda em consideração as seguintes recomendações:

- Faça PUSH e POP de todos os registos que use numa rotina e não constituam valores de saída (mas não sistematicamente de todos os registos, de R0 a R11!). 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, bem como à ordem relativa;
- 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, quer de constantes, quer de variáveis, e rotinas auxiliares de implementação de cada processo junto a ele;

- Não coloque constantes numéricas (com algumas exceções, como 0 ou 1) pelo meio do código. Defina constantes simbólicas no início e use-as depois no programa;
- Como boa prática, as variáveis em memória devem ser de 16 bits (WORD), para suportarem valores negativos sem problemas. O PEPE só sabe fazer aritmética em complemento para 2 com 16 bits;
- 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) e periféricos de 16 bits devem ser acedidos com MOV;
- **ATENÇÃO!!!** Ao contrário dos guiões de laboratório, o periférico POUT-1 é de 16 bits (por causa dos 3 displays) e não de 8 (deve ser acedido com MOV, e não MOVB);
- Produza comentários abundantes, não se esquecendo de cabeçalhos para as rotinas com descrição, registos de entrada e de saída (veja exemplos nos guiões de laboratório);
- Não duplique código (com *copy-paste*). Use uma rotina com parâmetros para contemplar os diversos casos em que o comportamento correspondente é usado.

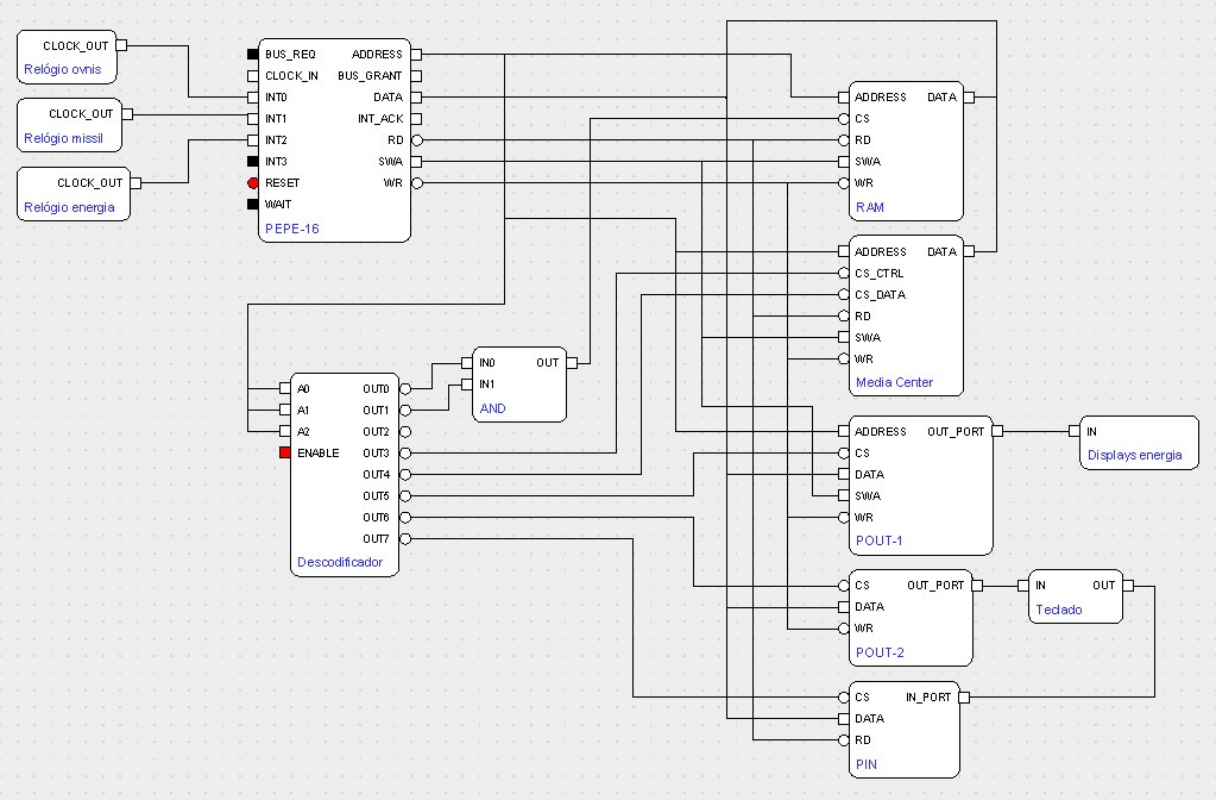
5 – Critérios de avaliação

Os critérios de avaliação e o seu peso relativo na nota final do projeto (expressos numa cotação em valores) estão indicados na tabela seguinte:

Critério	Versão intermédia	Versão final
Funcionalidade de base	1	4
Estrutura dos dados e do código	3	6
Comentários	1	2
Vários ovnis: funcionalidade, dados e código		3
Total	**5**	**15**

6 – Circuito do projeto

A figura seguinte mostra o circuito a usar (fornecido, ficheiro **projeto.cir**).



Os módulos seguintes têm painel de controlo em execução (modo "Simulation"):

- Relógio ovnis – Relógio de tempo real, para ser usado como base para a temporização do movimento dos ovnis. Está ligado ao pino de interrupção 0 do PEPE;
- Relógio míssil – Relógio de tempo real, para ser usado como base para a temporização do movimento do míssil. Está ligado ao pino de interrupção 1 do PEPE;
- Relógio energia – Relógio de tempo real, para ser usado como base para a temporização da diminuição periódica de energia da nave. Está ligado ao pino de interrupção 2 do PEPE;
- MediaCenter – módulo multimédia que inclui um ecrã de 32 x 64 pixels. Este ecrã é acedido como se fosse uma memória de 2048 pixels (ou 4096 bytes: 128 bytes em cada linha, 32 linhas), ou por comandos. Este periférico tem 2 *chip select*s, um para acesso pela memória e outro para acesso pelos comandos. Pode ver no ficheiro de excel **ecrã-32x64.xlsx** os endereços de cada byte (relativos ao endereço de base do ecrã). O guião de laboratório 4 fornece mais detalhes;

- Três displays de 7 segmentos, ligados aos bits 11-8, 7-4 e 3-0 do periférico POUT-1, para mostrar a energia da nave. **ATENÇÃO!!!**: ao contrário dos guiões de laboratório, este periférico é de 16 bits e deve ser acedido com MOV (e não MOVB);
- Teclado, de 4 x 4 teclas, com 4 bits ligados ao periférico POUT-2 e 4 bits ligados ao periférico PIN (bits 3-0). A deteção de qual tecla foi carregada é feita por varrimento. Atenção, que estes periféricos são de 8 bits e devem ser acedidos com MOVB (e não MOV);
- Memória (RAM), que tem 16 bits de dados e 14 bits de endereço, com capacidade de endereçamento de byte, tal como o PEPE e o MediaCenter;
- PEPE (processador).

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

Dispositivo	Endereços
RAM	0000H a 3FFFH
MediaCenter (acesso aos comandos)	6000H a 6063H (ver guião de laboratório 4)
MediaCenter (acesso à sua memória)	8000H a 8FFFH
POUT-1 (periférico de saída de 16 bits)	0A000H a 0A001H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H