Projeto de Introdução à Arquitetura de Computadores

LEIC

IST-Taguspark

Galáxia de Pulsares

2019/2020

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 do voo de uma nave espacial através de uma galáxia com um grande número de pulsares.

Um pulsar é uma estrela de neutrões, que resulta do colapso de uma grande estrela numa grande explosão (supernova), ficando apenas um núcleo com uma massa de cerca de 2 a 5 vezes a do nosso Sol mas com um diâmetro na ordem de apenas 20 Km. A gravidade é tal que os protões e eletrões fundem-se, tornando-se neutrões. A velocidade de rotação fica extremamente elevada, devido à conservação do momento angular e ao colapso do diâmetro da estrela (menor diâmetro, maior velocidade), podendo variar entre cerca de 4 segundos/rotação até mais de 1000 rotações/segundo. Esta rotação pode gerar campos magnéticos de cerca de 10^{12} vezes mais intensos que o da Terra. Os pulsares emitem uma forte radiação eletromagnética nos seus polos. Se esta radiação passar pela Terra o pulsar é detetável, mas a sua rotação faz com que pareça que está a pulsar (daí o nome), tal como acontece com um farol. A figura seguinte ilustra um pulsar a emitir radiação.

O cenário de base do jogo deve ser uma foto do espaço, para dar mais realismo. Há 4 pulsares, um em cada canto do ecrã, para ilustrar que há pulsares em todos os cantos do universo. Para além de pulsarem e emitirem raios, os pulsares tanto podem emitir radiação boa (verde, energizante), como má (vermelha, letal).

O mesmo pulsar vai mudando de tipo de radiação, de forma aleatória. Os raios são simulados por um pixel (verde ou vermelho) que navega pelo ecrã na diagonal que parte do canto onde está o pulsar. Quando o pixel chega ao topo ou ao fundo do ecrã, perde-se e o pulsar gera novo raio, que pode ser de tipo diferente do anterior.

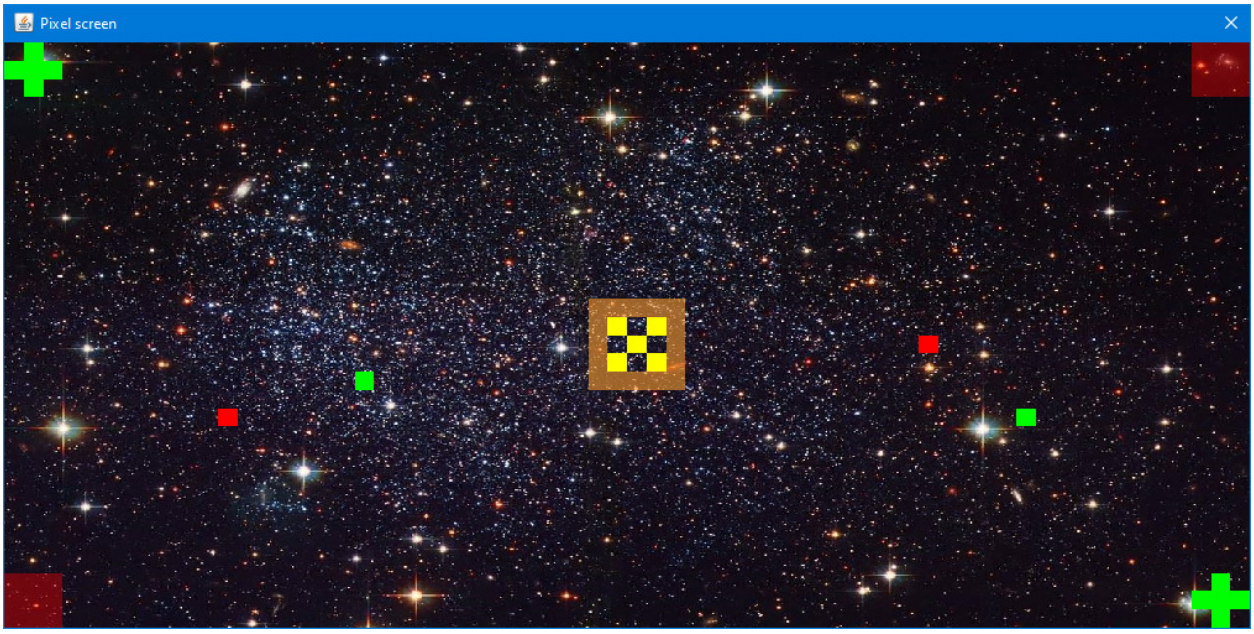
O pulsar adquire também a cor do raio (verde ou vermelho) e uma forma diferente para cada cor. A pulsação é simulada desenhando o pulsar ora totalmente opaco, ora semi-transparente. Todo o funcionamento dos pulsares é automático e autónomo do jogador.

No meio do ecrã, está uma nave, que se pode deslocar nas 8 direções da rosa-dos-ventos, sob controlo do jogador. Existem uns displays de 7 segmentos, que mostram a energia da nave. No início, a energia está a 100%, mas vai-se gastando de forma autónoma (por exemplo, 5% em cada 3 segundos).

No meio do ecrã, a nave está a salvo de ser atingida pelos raios dos pulsares. No entanto, se não fizer nada, ao fim de algum tempo a sua energia esgota-se e o jogador perde o jogo. O que pode fazer é deslocar-se para apanhar os raios verdes (energizantes). Por cada colisão destas, a energia sobe 30%. No entanto, se for apanhada por um raio vermelho, a nave explode e o jogador perde. Para o evitar, o jogador pode ligar um escudo de campo de forças, caso em que uma colisão com um raio mau quebra a energia de 20% (mas não explode). O problema é que com o escudo ligado a energia gasta-se ao dobro do ritmo (10% em vez de 5%), por isso não deve estar sempre ligado. Colisão com um raio bom com o escudo ligado não tem efeito na energia (a energia boa do raio não penetra no escudo).

A figura seguinte ilustra um aspeto do jogo. Os dois pulsares de baixo estão no brilho máximo e os de baixo no mínimo (mas são independentes). Veem-se os 4 raios e nave, sem escudo. Um dos raios é bom.

Na figura seguinte, os pulsares já evoluíram (em cada novo raio, podem mudar de bom para mau ou vice-versa) e a nave ligou o seu escudo (uma bordadura semi-transparente).



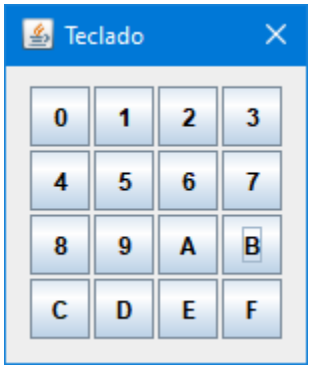
O objetivo do jogo consiste assim em aguentar a nave durante tanto tempo quanto possível, obtendo energia da radiação verde e evitando ser atingida pela radiação vermelha.

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. Como exemplo, uma possível atribuição é a seguinte:

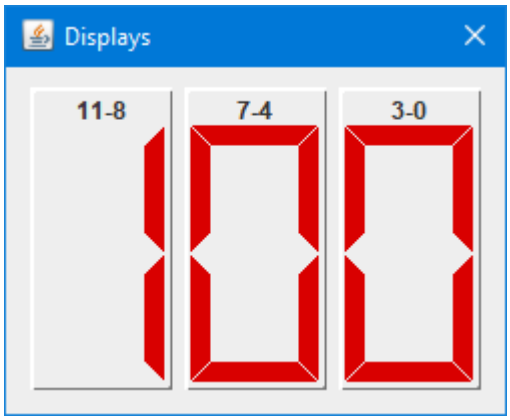
- Teclas 0, 1, 2: movimentar a nave para cima-esquerda, cima, cima-direita, respetivamente;
- Teclas 4 e 6: movimentar a nave para a esquerda e direita, respetivamente;
- Teclas 8, 9, A: movimentar a nave para baixo-esquerda, baixo, baixo -direita, respetivamente;
- Tecla B: ligar/desligar o escudo da nave;
- Tecla C: começar o jogo (deve reiniciar a 100% a energia da nave);
- Tecla D: suspender/continuar o jogo;
- Tecla E: terminar o jogo (deve manter visível a energia final da nave).

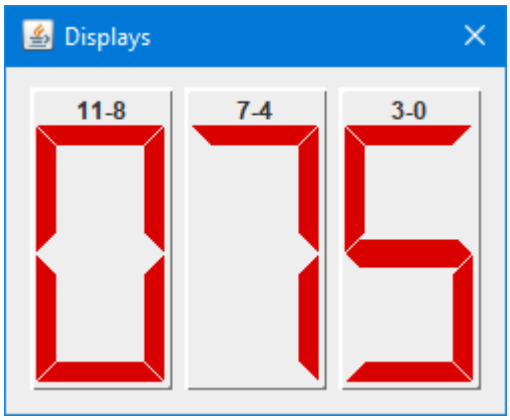
IMPORTANTE - Com exceção das teclas de movimentar a nave, cada tecla dever executar apenas um comando. Para novo comando, mesmo que igual, tem de se largar a tecla e carregar de novo. O movimento da nave pode ser em "contínuo", enquanto a tecla estiver carregada.

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 máximo, 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 % (ou 10%, se o escudo estiver ligado) de 3 em 3 segundos.

Por cada colisão da nave com um raio bom sem escudo, a energia aumenta de 30% (limitado a 100%). Se tiver escudo ligado, não tem efeito na energia.

Por cada colisão da nave com um raio mau com escudo, a energia diminui de 20% (limitado a 0%). Se não tiver escudo ligado, a nave explode e o jogo termina.

Se a energia chegar a 0%, por tempo ou colisões indesejáveis, embora com escudo ligado, o jogo termina também, mas agora por falta de energia. As imagens de cenário nas duas situações devem ser diferentes.

Se o jogo terminar, 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, bonecos, cenários e sons

O ecrã de interação com o jogador tem 32 x 64 pixels (linhas x colunas), em que cada raio é um pixel que se vai movimentando. Cada pixel pode ter uma cor diferente, em 4 componentes (Red, Green, Blue e Alpha, ou RGBA), todas com 8 bits (valores sem sinal, de 0 a 255). Esta última componente define a opacidade (0 é totalmente transparente, 255 totalmente opaco). O pixel pode estar ligado (com a cor definida pelas 4 componentes) ou desligado (caso em que não se vê).

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

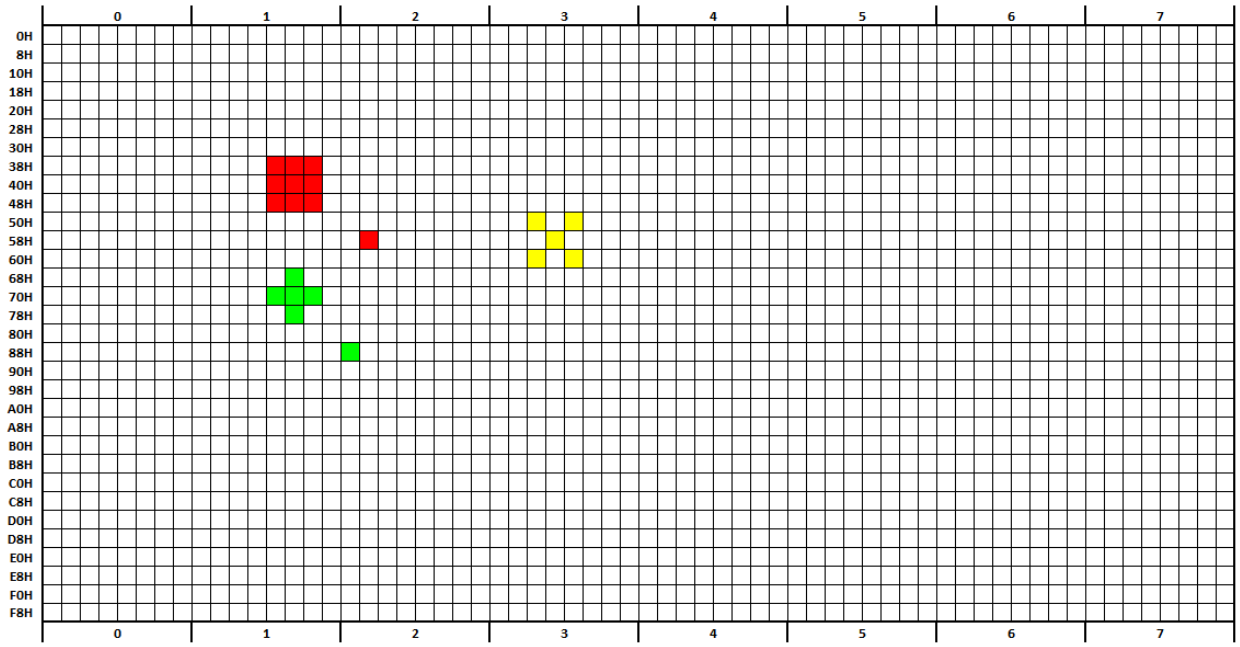
É possível alterar a imagem de fundo através do programa do PEPE. Por isso, espera-se que use um cenário diferente para cada situação. Nem sempre a imagem tem de variar. Pode edita-la, colocando texto que indique qual a situação (pausa, por exemplo), gerando assim variantes da mesma imagem. A figura seguinte ilustra esta possibilidade, com o cenário inicial.

Não é fornecida nenhuma imagem para cenário, mas existem inúmeras imagens adequadas que pode obter de forma gratuita para este uso pessoal. O design multimédia fica ao seu gosto.

Desenhar um boneco no ecrã (à frente do cenário) é ligar os seus bits a 1 e desligar os seus bits a zero, 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 apaga-lo na sua posição atual, mudar a sua posição e desenha-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 um bit dessa memória (embora internamente, de forma escondida, ainda tenha capacidade para guardar as 4 componente por cada pixel). Há 64 bits, ou 8 bytes, por cada linha do ecrã.



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

Cada pixel pode ser desenhado escrevendo diretamente na memória do ecrã ou por meio de um comando. Este último permite especificar diretamente a linha e coluna pretendidas, ao contrário do primeiro método em que a unidade mínima de leitura ou escrita é de 1 byte (8 pixels). Neste caso, para desenhar apenas um pixel tem de se determinar o endereço do byte que contém o pixel pretendido, ler esse byte, alterar o bit respetivo e a escrever o byte alterado no mesmo endereço.

Nos seus movimentos, a nave não deve sair do ecrã. Chegada ao limite, não deve avançar mais.

Um raio termina o seu movimento (sendo gerado novo raio, possivelmente de tipo diferente) quando atinge a borda superior ou inferior do ecrã ou se colidir com a nave.

Para além dos cenários e dos bonecos, o módulo do ecrã suporta ainda efeitos sonoros, ou seja, fazer ouvir ficheiros de som (apenas de formato WAVE, extensão .wav). Deve implementar pelo menos um ou dois efeitos sonoros.

Não é fornecido nenhum ficheiro de áudio, mas existem inúmeros ficheiros com efeitos sonoros, que pode obter de forma gratuita para este uso pessoal. O design multimédia fica ao seu gosto.

FUNDAMENTAL – Depois de definir os ficheiros de imagem e som que quiser acrescentar ao módulo de ecrã, deve salvar o circuito no mesmo diretório onde estão esses ficheiros. Esta é a única forma de os ficheiros serem incluídos no ficheiro de descrição do circuito de forma portável, apenas pelo nome do ficheiro. Caso contrário, é guardado o "path" absoluto do ficheiro, e depois não funciona noutro computador (nomeadamente, no do docente que vai avaliar o projeto!).

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

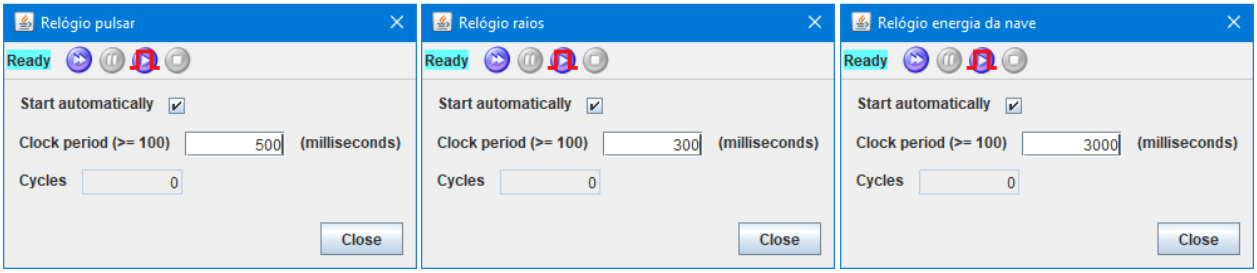
2.4 – Temporizações

A evolução do jogo requer 3 temporizações diferentes:

- Pulsação dos pulsares (período de 500 milissegundos);
- Movimento dos raios (período de 300 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 altera-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 laboratório5 ensina a utilizar relógios para marcação de temporizações.

2.5 – Escolhas pseudo-aleatórias

Quando um pulsar vai começar a emitir um novo raio, deve decidir se vai ser bom ou mau, tomando a cor forma respetiva e lançando um raio da cor que tiver escolhido.

Pretende-se que haja cerca de 25% de raios bons e 75% de maus.

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.

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 2019-20 - versão intermédia) até às **23h59** do dia **26 de outubro de 2019** através de um ficheiro (**grupoXX.zip**, em que XX é o número do grupo) com os seguintes ficheiros:
  - Um ficheiro **grupoXX.asm** 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;
  - Os ficheiros de imagem e de som usados no módulo "Pixel screen";
  - Um ficheiro **projetoXX.cir** com o circuito do projeto, mas guardado depois de definir no módulo "Pixel screen" os ficheiros de imagem e de som usados. Não se esqueça que o circuito deve ser guardado no mesmo diretório onde estão estes ficheiros, e não copiado para lá depois;
- Numa aula de laboratório seguinte, o docente poderá fazer algumas perguntas sobre o programa entregue e dar feedback com algumas sugestões;
- **IMPORTANTE** - Use o circuito do projeto, **projeto.cir** (e não o de qualquer guião de laboratório);

- **Deve cumprir os seguintes objetivos**:
    - Colocar no ecrã uma imagem de cenário de fundo adequada para este projeto;
    - Desenhar a nave no meio do ecrã (posição inicial do jogo) e pelo menos um pulsar, na sua posição. Não desenhe a nave e o pulsar manualmente, pixel a pixel. Faça uma rotina que leia uma tabela com o valor dos pixels da nave e do pulsar. Fica assim mais genérico e permite desenhar qualquer objeto, nave com e sem escudo, pulsares e raios;
    - Implemente uma rotina para executar um efeito sonoro adequado para este projeto, invocada quando se carrega numa tecla (só pode invocar novamente quando se larga a tecla e se carrega de novo!);
    - Implemente um contador **decimal** de 0 até 100, com uma rotina para aumentar e outra para diminuir (ambas mostram o novo valor do contador após a variação). O contador não deve sair do intervalo [0 .. 100];
    - Estas rotinas devem ser invocadas repetidamente enquanto se carrega no teclado, numa dada tecla (defina duas ao seu gosto). Quando se larga a tecla, o contador deve parar de evoluir. Se a evolução for muito rápida, pode atrasa-la com uma rotina que apenas faz um ciclo de um número elevado de iterações (1000 ou mais, por exemplo).

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 2019-20 - versão final) até às **23h59** do dia **22 de novembro 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;
    - Todos os ficheiros de imagem e de som usados no módulo “Pixel screen”;
    - Um ficheiro **projetoXX.cir** com o circuito do projeto, mas guardado depois de definir no módulo “Pixel screen” todos os ficheiros de imagem e de som usados. Não se esqueça que o circuito deve ser guardado no mesmo diretório onde estão estes ficheiros, e não copiado para lá depois;

- Nas últimas semanas de aulas, a partir de 2 de dezembro, 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 local em que o seu grupo deverá comparecer serão anunciados no Fenix, após a data de entrega da versão final do projeto.

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 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 a direção, ligar/desligar escudo e fazer evoluir a energia);
- Pulsar (para controlar as ações e evolução de cada um dos pulsares. Note que vários pulsares correspondem apenas a instâncias diferentes do mesmo processo, ou várias chamadas à mesma rotina com um parâmetro que indique o número do pulsar);
- Raio (para controlar a evolução dos raios no espaço. Note que vários raios correspondem apenas a instâncias diferentes do mesmo processo, ou várias chamadas à mesma rotina com um parâmetro que indique o número do raio, que na prática é o mesmo número do pulsar);
- Gerador (inclui um contador para gerar um número pseudoaleatório, usado para escolher o tipo de um pulsar, bom ou mau, bem como o tipo do respetivo raio);
- 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, displays e desenho da nave (cobertos pela versão intermédia);
2. Movimentos da nave, de acordo com a tecla carregada;
3. Rotinas de ecrã, para desenhar/apagar:
    - um pixel numa dada linha e coluna (de 0 a 31 e 0 a 63, respetivamente), se ainda o não tiver feito na versão intermédia;
    - um objeto genérico, descrito por uma tabela que inclua a sua largura, altura e o valor de cada um dos seus pixels, e ainda as componentes RGBA. 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;
4. Um só pulsar (tem de definir tabelas com a representação dos vários bonecos dos pulsares e tabelas para escolher o boneco adequado consoante o brilho e tipo do pulsar);

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. Pulsação do pulsar, por meio de uma interrupção;
9. Processo Raio, com o movimento linear regulado por meio de uma interrupção;
10. Deteção de colisão de um raio com a nave;
11. Resto das especificações, incluindo extensão para mais do que um pulsar e raio. Esta extensão tem algumas complicações, pelo que é melhor ter um só pulsar 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 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 raio 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 de um pulsar), 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 pulsares e raios, pode inclusivamente invocar o Gerador entre cada invocação do pulsar, 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 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.

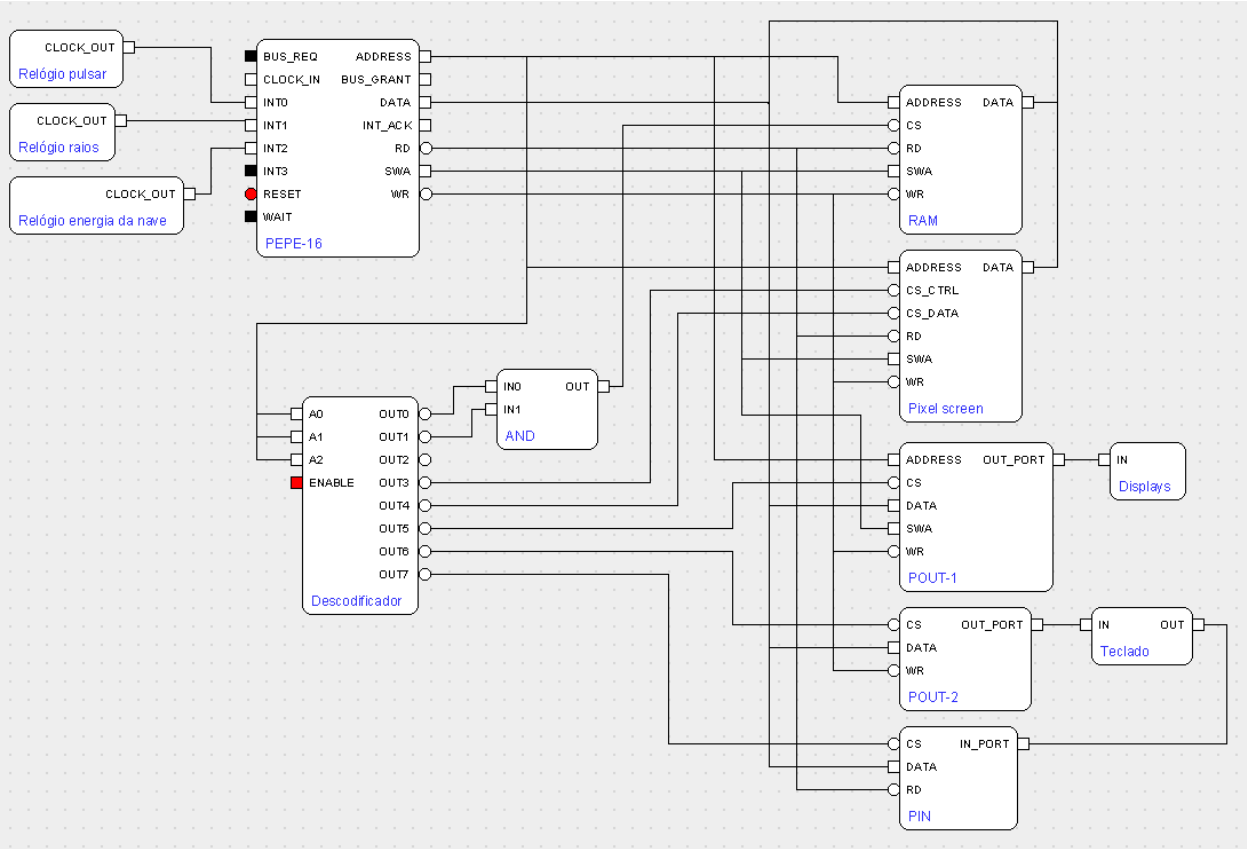
5 – Critérios de avaliação

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 pulsares: 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 Simulação):

- Relógio pulsar – Relógio de tempo real, para ser usado como base para a temporização da pulsação dos pulsares. Está ligado ao pino de interrupção 0 do PEPE;
- Relógio raios – Relógio de tempo real, para ser usado como base para a temporização do movimento dos raios. Está ligado ao pino de interrupção 1 do PEPE;
- Relógio energia da nave – Relógio de tempo real, para ser usado como base para a temporização da diminuição de energia da nave. Está ligado ao pino de interrupção 2 do PEPE;
- PixelScreen (ecrã de pixels) – ecrã de 32 x 64 pixels. É acedido como se fosse uma memória de 256 bytes (8 bytes em cada linha, 32 linhas), ou por comandos. Este periférico tem 2 *chip selects*, 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). 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 ligado (com uma dada cor), a 0 um pixel desligado. 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 acedido 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 PixelScreen;
- PEPE (processador).

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

Dispositivo	Endereços
RAM	0000H a 3FFFH
PixelScreen (acesso aos comandos)	6000H e seguintes (ver guião de laboratório 4)
PixelScreen (acesso à sua memória)	8000H a 80FFH
POUT-1 (periférico de saída de 16 bits)	0A000H
POUT-2 (periférico de saída de 8 bits)	0C000H
PIN (periférico de entrada de 8 bits)	0E000H