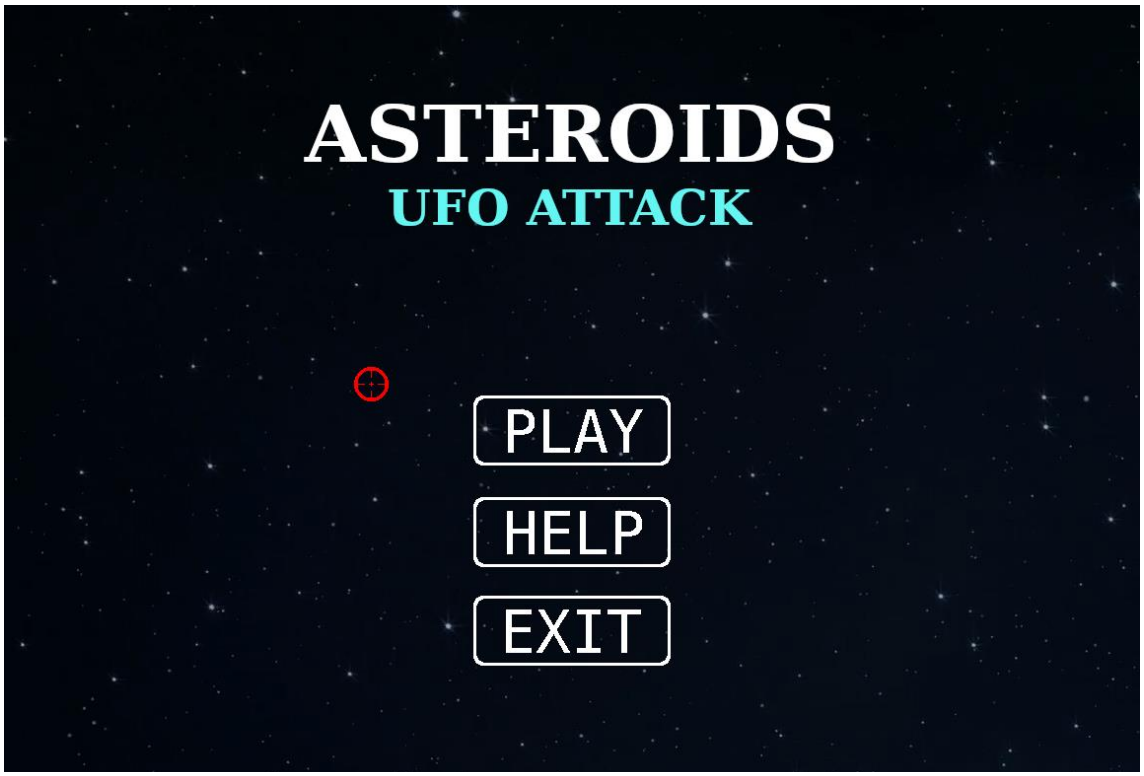
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Asteroids UFO Attack
Relatório



Laboratório de Computadores 2019/2020
2º Ano MIEIC
Turma 5 Grupo 07

André Filipe Meireles do Nascimento (up201806461)
João Nuno Diegues Vasconcelos (up201504397)

Índice

1. Descrição

O jogo desenvolvido é inspirado no jogo original Asteroids mas com algumas modificações. O objetivo do jogo é destruir os asteroides que vêm de direções aleatórias e desviar o ufo para não colidir com estes, aumentando o score à medida que o tempo passa e asteroides são destruídos. O jogo termina quando o ufo colide com um asteroide.

2. Instruções de utilização

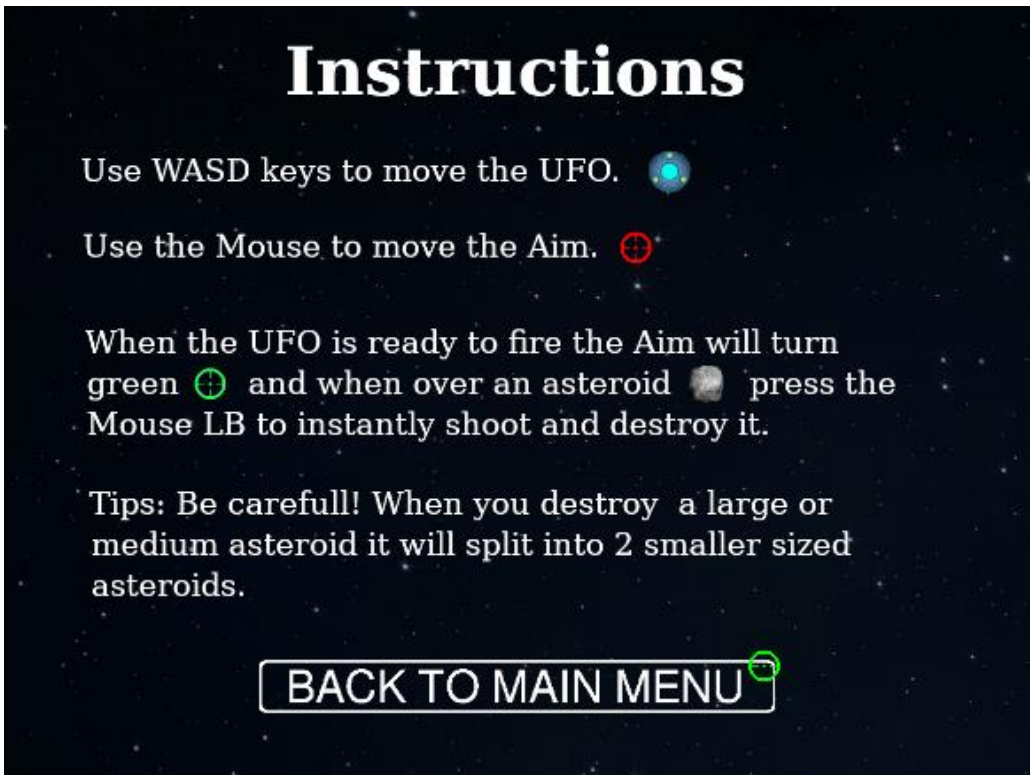
2.1) Main Menu

Ao inicializar o programa é apresentado ao utilizador o menu principal do jogo que contem 3 botões com opções que podem ser selecionadas com o rato.

- Play: Inicia um novo jogo
- Help: Instruções do jogo
- Exit: Fecha o programa

2.2) Help

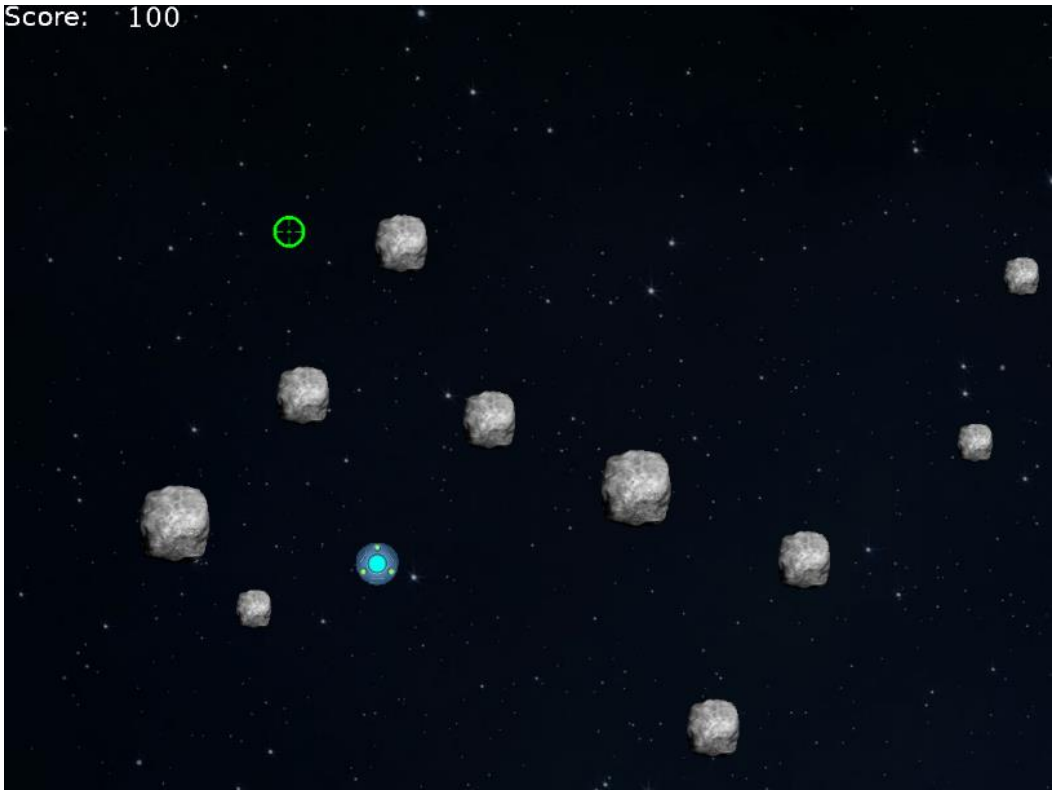
Apresenta as instruções do jogo. O utilizador movimenta o ufo com a tecla W (move-se para cima), A (move-se para a esquerda), S (move-se para baixo), e D (move-se para a direita). Tem também uma mira de cor vermelha, que fica verde quando o disparo está disponível, isto é, a cada meio segundo (para impedir que o jogador esteja sempre a disparar e tornando o jogo mais difícil), movimentando-a com o rato, e pode "disparar" premindo o botão esquerdo do rato quando está em cima de um asteroide, o que faz este destruir-se imediatamente, e caso seja de tamanho grande ou médio divide-se em dois asteroides de tamanho menor correspondente. Pode voltar ao menu inicial carregando no botão correspondente.



2.3) Play

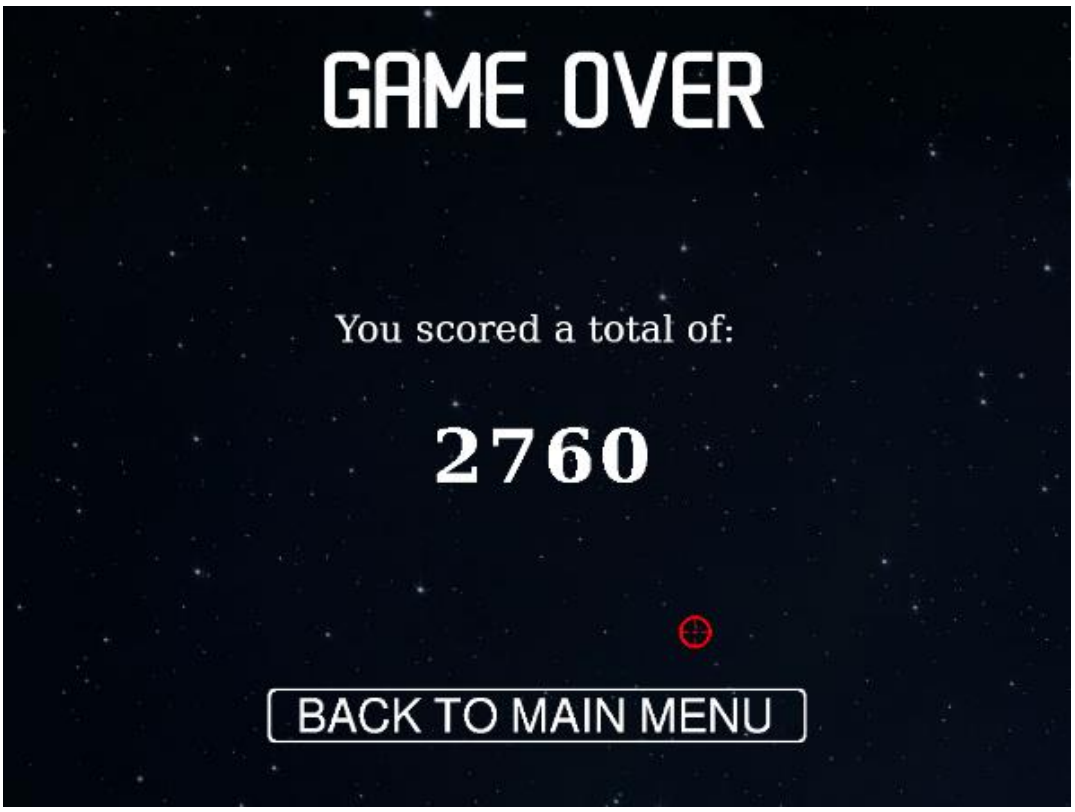
Após começar um novo jogo o utilizador depara-se com o ufo no centro do ecrã e com a mira presente também, podendo-os movimentar de acordo como já explicado. Começam a aparecer asteroides aleatórios, tanto em posição em que são adicionados como em velocidade e tamanho. Inicialmente os asteroides aparecem a uma taxa menor, mas à medida que o score aumenta estes são adicionados com mais frequência. O score do jogo atual é apresentado no canto superior esquerdo, e aumenta cada vez que um novo asteroide é adicionado e quando asteroides são destruídos, sendo que asteroides de menor tamanho dão mais pontos. O score é também diminuído cada vez que o jogador falha um "disparo".

O jogador perde quando o ufo colide com qualquer um dos asteroides. Termina assim o jogo e é encaminhado para o ecrã de Game Over.



2.4) Game Over

Quando o jogo termina aparece o ecrã de Game Over onde é apresentada a pontuação obtida no jogo que decorreu, e um botão que permite voltar ao menu principal.



3. Estado do projeto

3.1) Tabela com os dispositivos usados

Dispositivo	Funcionalidade	Interrupções
Timer	Atualizar o estado do jogo (atualizar o que aparece no ecrã, criar asteroides, incrementar o score, disponibilidade do "disparo").	Sim
Teclado	Movimento do UFO, sair do jogo com a tecla ESC quando no menu principal.	Sim
Rato	Movimento da mira, navegação dos menus, e "disparar".	Sim
Placa gráfica	Visualizar menus e todas as imagens do jogo	Não

3.2) Timer

É usado para atualizar o estado do jogo a cada interrupção. Consoante o estado em que o jogo se encontra é feito o desenho para um buffer secundário (por exemplo, no caso do menu principal: drawMainMenu() para desenhar o fundo e moveMouse() para desenhar o rato na posição atualizada), e depois efetuada a cópia para o buffer principal (swapBuffer()). É usado também para a criação de asteroides à medida que o tempo passa, aumentar o score e verificar colisões (nos menus, mouseMainMenuChange() e mouseMenuChange(), e durante o jogo, checkBulletAsteroidColision() para verificar se algum asteroide foi atingido e checkAsteroidUFOColision() para verificar se o ufo foi atingido e assim acabar o jogo).

Todas as funções fundamentais são chamadas nas suas interrupções.

3.3) Teclado

Permite sair do programa, quando no menu principal, ao carregar na tecla Esc., mas a sua principal funcionalidade é movimentar o ufo. A cada interrupção é chamada a função updateUFO() que consoante os make/break codes recebidos pelo kbd_ih() (interrupt handler do teclado) atualiza a direção do ufo.

3.4) Rato

É usado durante todo o programa. A cada interrupção é chamada função updateMouse() que trata de atualizar a sua posição no ecrã consoante o movimento e verifica se o botão esquerdo foi premido ou não. Nos menus quando se encontra em cima de um botão e é premido o botão esquerdo do rato o programa passa para o menu correspondente, e quando em jogo, se o rato se encontrar em cima de asteroide, e for premido o botão esquerdo, o asteroide é destruído.

3.5) Placa gráfica

Responsável por tudo o que aparece no ecrã. Foi usado o modo gráfico 0x14C, cuja resolução é 1152x864, com 32 bits por pixel. Foi usado “double buffering” que consiste em desenhar num buffer secundário e quando tudo estiver pronto copia o conteúdo desse buffer para a memória principal (através da função swapBuffer()), o que nos permite evitar o flickering, que tornaria a experiência desagradável.

Foram usadas XPMs para as imagens, e para fazer com que os objetos se movessem usamos Sprites cujas coordenadas de x e y mudam consoante o necessário. A deteção de colisões entre dois sprites é feita através das coordenadas e dimensões das imagens, verificando se ficariam sobrepostos, e nesse caso ocorre uma colisão (a função checkColision() faz essa verificação e serve de base para outras funções).

4. Estrutura e organização do código

4.1) Proj

Contem o main loop onde é tratado tudo do programa sendo a raiz do projeto. Todas as funções são aqui chamadas, consoante o estado em que se encontra o programa (menus ou dentro do jogo), que é ditado pela state machine incluída no loop.

Peso no projeto: 10% Autor: André Nascimento (100%)

4.2) Game

Módulo responsável por inicializar o jogo e as respetivas variáveis, fazer reset quando se perde um jogo e destruir as variáveis criadas quando se sai do programa. Contém também as funções para desenhar os menus e o in-game score.

As variáveis do jogo são todas guardadas numa struct Game que contem: uma struct GameSps com todos os Sprites usados no jogo, uma struct Images com todas as imagens usadas no programa, uma enum game_state que guarda o estado atual do programa, e o score da sessão atual.

Peso no projeto: 15% Autor: André Nascimento (90%)

4.3) Video_graph

Importado do lab5 com algumas alterações feitas. Contem a função para inicializar em modo gráfico e funções auxiliares para obter informações do modo atual, função para desenhar os XPMs no buffer secundário e a função para fazer a copia para o buffer principal.

Peso no projeto: 5% Autor: André Nascimento (70%)

4.4) Sprite

Contém a struct Sprite fornecida para o lab5, que serve de base aos objetos animados no jogo. Funções de criar, destruir, desenhar e apagar sprites estão aqui incluídas.

Peso no projeto: 10% Autor: André Nascimento (100%)

4.5) Timer

Importado do lab2 e sem alterações realizadas.

Peso no projeto: 5% Autor: André Nascimento (60%)

4.6) Keyboard

Importado do lab3 e sem alterações realizadas.

Peso no projeto: 5% Autor: André Nascimento (60%)

4.7) Mouse

Importado do lab4 e sem alterações realizadas.

Peso no projeto: 5% Autor: André Nascimento (60%)

4.8) Aim

Módulo responsável pela mira. Contém a função criar o Sprite da mira, função para receber os pacotes do rato e atualizar a mira com essa informação, função para desenhar a mira na posição atualizada, função para tratar disponibilidade do “disparo” e função para dar reset ao “disparo”.

Peso no projeto: 10% Autor: André Nascimento (80%)

4.9) UFO

Módulo responsável pelo ufo. Contém a função criar o Sprite do ufo, função para atualizar a posição do ufo consoante os interrupts recebidos pelo teclado e funções para movimentar o sprite e desenhar na posição atualizada.

Peso no projeto: 10% Autor: André Nascimento (100%)

4.10) Asteroids

Módulo responsável pelos asteroides e o array onde são guardados. Contém as funções que permitem criar um array Sprite* de asteroides vazio, criar asteroides aleatórios, movimentá-los e desenhá-los.

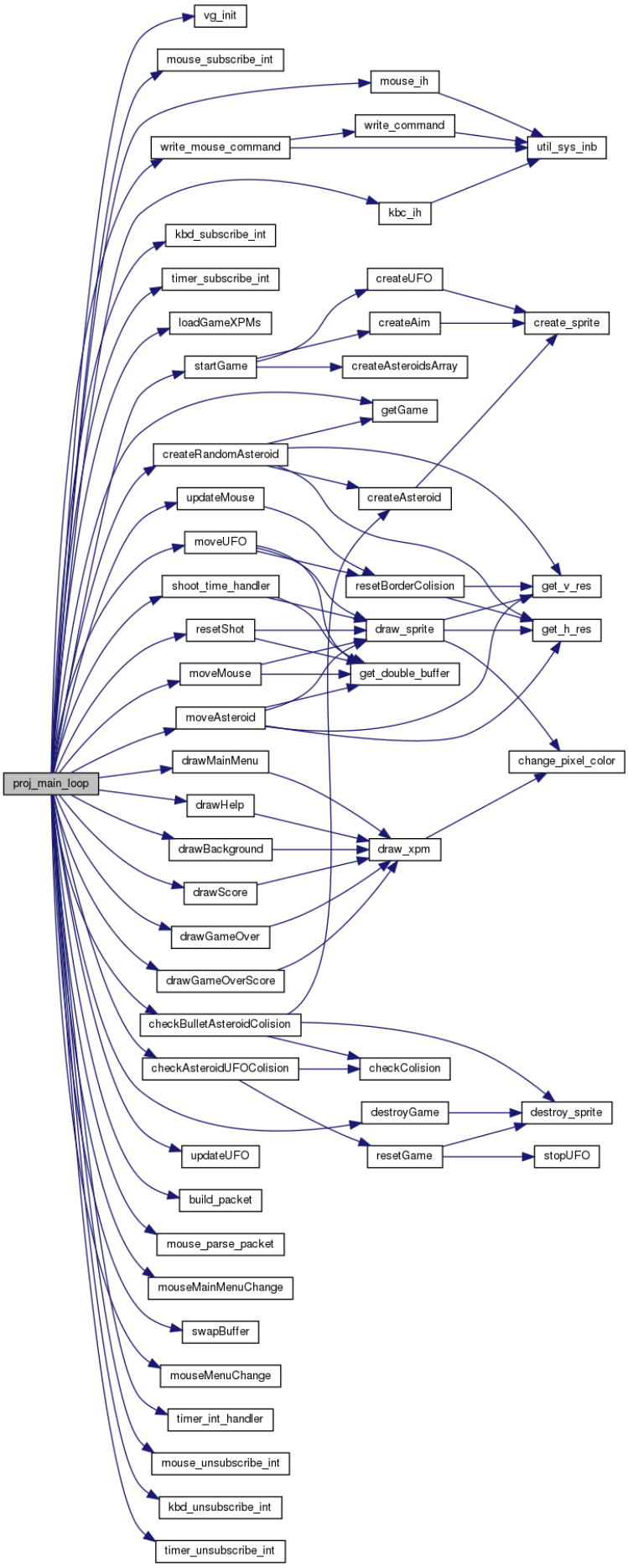
Peso no projeto: 10% Autor: André Nascimento (100%)

4.11) Collision

Módulo onde são tratadas as colisões entre sprites e alternar de menu consoante os botões carregados. Contém funções para detetar colisões entre a borda da janela do programa e o ufo e a mira para não os permitir sair do ecrã, entre ufo e asteroides, entre “disparos” e asteroides, e funções que mudam de menu.

Peso no projeto: 15% Autor: André Nascimento (80%)

4.12) Call graph



5. Detalhes de implementação

Um detalhe a destacar é a struct Game que guarda todas as variáveis necessárias para o jogo, tal como as sprites, todas as imagens usadas, score do jogo e o estado atual e único em que o programa se encontra (menu principal, instruções, em jogo, game over e estado final). Isto facilita a utilização destas variáveis noutros ficheiros, sem ter de recorrer a vários métodos get e set, usando apenas getGame() que retorna um apontador para essa struct permitindo-nos assim aceder e alterar as variáveis livremente.

Outro destaque é a maquina de estados, implementada no main loop do proj.c, que trata de chamar as funções necessárias para o respetivo estado do jogo e mostrar no ecrã apenas o necessário.

Foi também implementada a técnica de “double buffering” que permite uma melhor experiencia de utilização do programa sem flickering.

6. Conclusões

LCOM foi sem dúvida a unidade curricular mais desafiadora este semestre e a que exigiu mais trabalho.

Houve uma dificuldade inicial em adaptar-nos ao trabalho semanal que a cadeira exige, não estávamos habituados a entregas todas as semanas e ter de trabalhar tanto fora de aula. Os handouts são bastante extensos e achamos que podem ser mais resumidos e concisos, pois torna-se confuso e complicado ter de processar tanta informação que por vezes acabava por nem se revelar necessária.

O projeto foi bastante interessante e sem dúvida o melhor ponto da unidade curricular, e permitiu perceber melhor toda a integração dos periféricos e a sua utilização.