

LCOM - Project Report

Color Shooter

Eduardo Ribeiro —> up201705421
Eduardo Macedo —> up201703658

Turma 1, Grupo 04

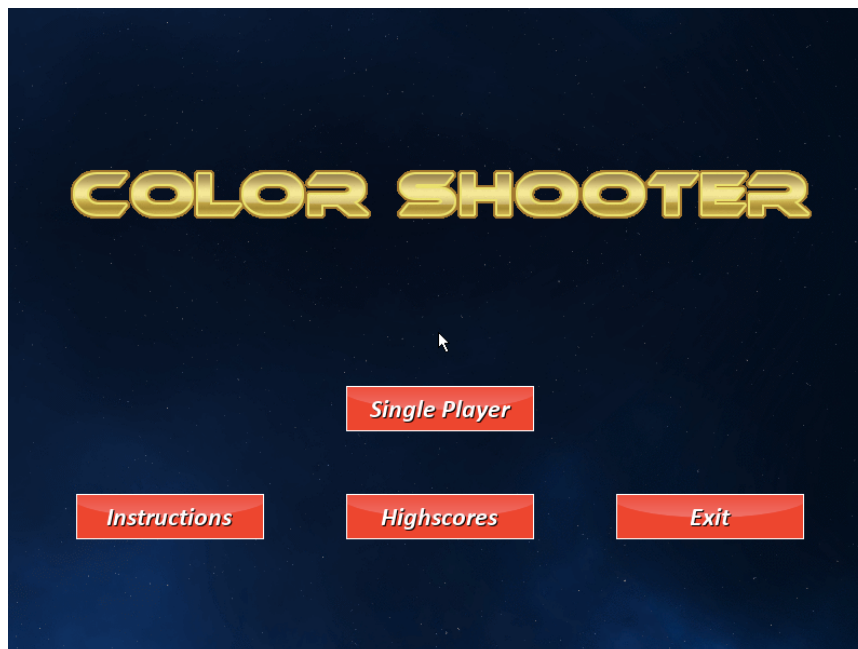
LCOM 18/19
4 de Janeiro de 2019

Capítulo 1 —> Instruções de Utilização

1.1 —> Menu Principal

Ao iniciar o jogo, é apresentado ao jogador o menu principal, em que se pode observar o título do jogo, “Color Shooter”, um fundo espacial “em movimento”, 4 botões, que representam as opções disponíveis no menu, e finalmente um cursor, que o jogador deve utilizar para escolher a opção pretendida. Isto é feito, naturalmente, com o rato. As opções disponibilizadas pelos botões são as seguintes:

- Single Player, que permite o começo de um novo jogo;
- Instructions, que permite ao utilizador ver um ecrã com as instruções do jogo;
- Highscores, que permite a visualização das 5 melhores pontuações obtidas;
- Exit, que termina o programa.



1.2 —> Instruções

Aproveitaremos este tópico para explicar o funcionamento e objetivo do jogo em si. O jogo consiste em destruir as naves inimigas, que vão aparecendo no topo do ecrã, de modo a que estas não se aproximem da nave do jogador, que se encontra na parte de baixo do ecrã. O jogador ganha 1 ponto por cada nave destruída.

Para destruir as naves inimigas, é necessário apontar para elas com a mira da nave do jogador, e disparar para elas, escolhendo a bala de cor certa (vermelha, verde, azul ou amarela), de modo a corresponder com a cor da nave inimiga. Apenas uma bala de cor certa irá destruir a nave; as balas de cor errada não irão causar nenhum dano.

Para apontar e disparar, o jogador deve movimentar o rato, e clicar no botão esquerdo do mesmo, respetivamente. A escolha da cor da bala é feita através das teclas WASD, cada uma correspondendo a uma cor, como se pode observar na imagem.

Por fim, em algumas ocasiões, o jogador irá ter disponível um ataque especial, que pode utilizar para criar um “feixe de luz”, disparando balas brancas a uma velocidade e ritmo maior, sendo que estas destroem naves de qualquer cor. Este ataque especial demora poucos segundos, sendo que o jogo retomará o seu funcionamento normal após o fim deste período.


O jogador pode ativar este ataque especial clicando no botão direito do rato.

Focando agora a atenção no menu das instruções, podemos observar uma breve explicação do funcionamento do jogo, acompanhada de algumas imagens; é também apresentado um botão Back, que permite o regresso ao menu principal.


> Instructions:


> The goal of the game is to destroy the enemy spaceships, in order to keep your ship safe. If the enemies get too close to you, you lose the game.


> The bullet used to destroy a certain enemy needs to match the color of that enemy. Other bullets won't affect them! You can change the color of the bullet using the WASD keys.

> From time to time, your ship will charge a light beam special attack. When the progress bar is full, use it to destroy all enemy spaceships in the way using the mouse's right button. 

Good luck!







1.3 —> Single Player

Como já tinha sido referido, durante o jogo irão aparecer naves inimigas de diferentes cores, como podemos observar na imagem. A nave do jogador encontra-se na parte de baixo do ecrã, acima da qual podemos observar a cor atual da bala. Podemos também observar a mira da nave do jogador, que se pode deslocar horizontalmente. Tal como no menu principal (e no menu de Game Over e dos Highscores, como iremos ver mais adiante), é apresentado um fundo espacial “em movimento”, para dar ao jogador uma sensação mais real de deslocamento das naves.

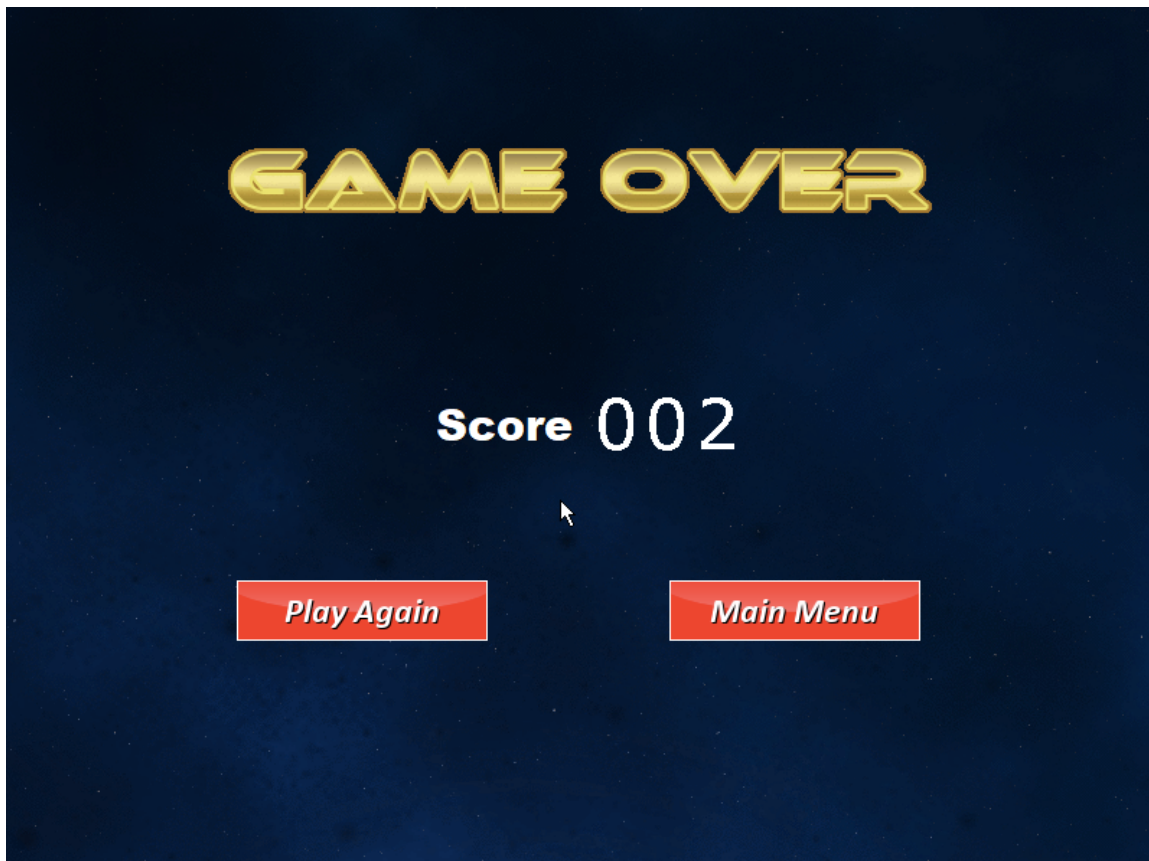
No canto inferior esquerdo, podemos observar a pontuação atual do jogador que, tal como foi dito, irá ser incrementada por 1 sempre que uma nave inimiga for destruída.

No canto inferior direito, é visível a barra de progresso do ataque especial: só quando a barra se encontrar totalmente cheia é que o jogador ganhará a capacidade de o utilizar.



1.4 —> Game Over

Quando o jogador “é apanhado” pelas naves inimigas, irá perder o jogo. Irá, portanto, ser encaminhado para o ecrã de Game Over, onde se pode observar a pontuação que o utilizador conseguiu obter durante o jogo, e dois botões, Play Again e Main Menu, que permitem o começo de um novo jogo, e o retorno para o menu principal, respetivamente.



1.5 —> Highscores

Finalmente, o utilizador do programa tem a possibilidade de visualizar as 5 melhores pontuações obtidas no jogo. Como se pode observar na imagem, estas encontram-se num ranking. Para cada uma, é também apresentada a data e a hora em que foi obtida.

É apresentado mais uma vez um botão Back, tal como nas instruções, que permite o regresso ao menu principal.



Capítulo 2 —> Estado do Projeto

2.1 —> Tabela dos dispositivos utilizados

Dispositivo	Funcionalidades	Interrupts?
• Timer	Controlo do frame rate, animações dos objetos, acontecimentos periódicos como o aparecimento de uma nave, duração do ataque especial, etc	Sim
• Keyboard	Deteção de certos keycodes que levam à mudança da cor atual da bala	Sim
• Mouse	Direção da mira e disparo de balas; utilização do ataque especial; navegação em menus	Sim
• Placa Gráfica	Desenho/display de imagens no ecrã	Não
• RTC	Extração da data e hora, sempre que é obtida uma pontuação que irá ficar presente nos Highscores	Não

2.2 —> Timer

Tal como foi mencionado, o timer assume a funcionalidade de controlo do frame rate; a cada interrupt do timer, é gerada uma nova frame, em que a posição do fundo é atualizada, bem como as posições das balas, inimigos, etc; também permite a animação das naves, balas e explosões, sendo que cada um dos objetos possui várias imagens, que quando apresentadas em sucessão rápida resulta na animação do mesmo.

Para além disto, permite controlar o ritmo de disparo da nave do jogador: para impedir a mesma de estar constantemente a disparar, cada vez que uma bala é disparada, a capacidade de disparo da nave é “desativada” durante um muito curto intervalo de tempo, ao fim do qual é “reativada”, permitindo ao utilizador disparar outra vez.

Permite também o controlo de todos os acontecimentos periódicos, ou seja, tudo o que acontece ao fim de um determinado intervalo de tempo. Isto inclui o aparecimento (“spawning”) de uma nave inimiga, que é feito a um certo ritmo, e o controlo da duração do ataque especial, ao fim do qual o jogo retoma o seu funcionamento normal.

Em cada interrupt do timer, é chamado o seu IH em assembly, timer_asm_ih(), que incrementa um contador. De seguida, é invocada a função timeManager(), que controla tudo o que necessário ser feito quando ocorre uma interrupção do timer: tendo em conta o estado do programa, chama várias outras funções, executando as tarefas acima descritas.

2.3 —> Keyboard

O teclado do computador é utilizado na lógica do jogo, para além do rato. Permite modificar a cor atual da bala, de modo a que da próxima vez que ocorrer um disparo, essa bala irá ter a cor escolhida (isto não afeta as balas que já foram disparadas). Para tal, o utilizador deve premir a tecla W, para selecionar a cor vermelha, A, para a cor verde, S, para a amarela, e D, para azul. Esta funcionalidade é desempenhada pelas funções change_current_ammo() e change_ammo_array().

Sempre que ocorre uma interrupção vinda do keyboard, é invocado o seu IH em assembly, kbc_asm_ih(), e se nenhum erro ocorreu, é invocada a função kbdManager(), que trata de todos os acontecimentos relacionados com o teclado. Se o estado do jogo o permitir, invocará outras funções (as mencionadas acima) que irão executar a funcionalidade descrita.

2.4 —> Mouse

O rato é o dispositivo diretamente utilizado pelo jogador que apresenta mais funcionalidades. Permite, como foi descrito anteriormente, a utilização/movimentação de um cursor para navegar nos menus, clicando (com o botão esquerdo) em cima de um botão para escolher a opção pretendida (mouse_check_clicks()).

Relativamente ao jogo em si, movimentar o rato horizontalmente resulta na deslocação da mira da nave do jogador (update_aim()); ao clicar no botão esquerdo, é disparada uma bala na direção da mira (launch_bullet_array()). Quando for possível a utilização do ataque especial, tal pode ser feito, clicando no botão direito do rato.

Em todas as interrupções do rato, é chamado o seu IH em assembly, mouse_asm_ih(), que extrai o “packet byte” atual. Cada vez que é formado um packet completo, é atualizado um campo de um objeto da struct Game, que guarda esse packet; de seguida, é invocada a função mouseManager(), que gere tudo o que está relacionado com eventos/acontecimentos desencadeados por interrupções do rato. Esta função, dependendo do estado atual do jogo, chama várias outras funções, que executam as funcionalidades acima descritas.

2.5 —> Placa Gráfica

Relativamente à placa gráfica, foi utilizado o modo 0x115, que oferece uma resolução de 800x600; o modo de cor associado é o modo direto, sendo que a distribuição RGB de bits por pixel é 8:8:8 (dando um total de 24 bits por pixel). Para ativar este modo, são utilizadas no início e fim da execução do jogo, as funções vg_init() e vg_exit(), respetivamente.

Este dispositivo é a base de todo o jogo. Através dele conseguimos apresentar no ecrã os diferentes fundos, objetos, etc.

De modo a tornar o jogo fluído, foi utilizada a técnica de double buffering, que permite o “desenho” das imagens num buffer secundário (feito por update_frame()), sendo que a cada interrupt do timer, o conteúdo do double buffer é passado todo de uma vez para a video RAM (feito por DBtoVM()).

Em relação às imagens utilizadas, foi não só implementado o movimento de objetos no ecrã, como por exemplo das balas e dos inimigos (funções update), mas também a animação dos objetos: as explosões, inimigos e balas possuem não uma imagem, mas várias imagens que quando apresentadas em sucessão rápida, resultam na animação destes. Ao fim de um certo número de interrupts do timer, são chamadas

funções (funções animate) que mudam o pixmap atual destes objetos, de modo a serem apresentados no ecrã sequencialmente.

Relativamente à verificação de colisões entre as balas e os inimigos, é efetuado dois tipos de verificação: um relativamente às coordenadas, que verifica se o rectângulo da imagem da bala se sobrepõe ao do inimigo (feito por check_pixmap_collision()); se existir colisão ao nível das coordenadas, é efetuado um outro tipo de verificação, ao nível do pixel (feito por check_pixel_collision()), que verifica, na área ocupada pelos dois objetos, se existem pixels correspondentes não transparentes, ou seja, verifica se houve realmente colisão entre os dois objetos. Durante a execução do jogo, estas funções são aplicadas a todas as balas e a todos os inimigos (check_all_collisions()).

Para “desenhar” no ecrã números e caracteres, como acontece no display dos highscores e da pontuação obtida pelo jogador, são utilizadas fontes; tendo as imagens de todos os algarismos, do carácter de dois pontos e do carácter da barra, é chamada a função num_to_sprite(), que associa cada número (de 0 a 9) ao seu sprite correspondente; de seguida apenas é necessário o display desse sprite nas coordenadas desejadas.

2.6 —> RTC

O RTC é utilizado para extrair a data (dia do mês, mês, e ano) e a hora (horas e minutos) sempre que uma alta pontuação é obtida pelo jogador, associando os valores extraídos à mesma, de modo a poder mostrar essa informação ao utilizador quando este se encontrar no menu dos highscores. Sempre que um jogo acaba, a função update_highscores() é chamada; se a pontuação for alta o suficiente para obter uma posição no “leaderboard”, extract_date_and_hour() é invocado, que por sua vez chama as funções do RTC get_date() e get_hour().

Capítulo 3 —> Estrutura e Organização do Código

3.1 —> Bullet.c / Bullet.h

Neste módulo encontra-se a struct que representa os objetos do tipo Bullet, que são as balas disparadas pela nave do jogador, bem como diversas funções que permitem a criação, modificação e destruição das mesmas. A struct contém, entre outros aspetos, as dimensões e coordenadas da bala, a sua velocidade, as suas imagens/pixmaps, e a sua cor atual. Algumas funções incluídas são, por exemplo, a que permite movimentar as balas, animar as balas, e disparar as mesmas. Foram também feitas algumas funções get, de modo a ser possível o acesso aos campos da struct.

Contribuição:

Eduardo Ribeiro: 55% —> Funções que mudam a cor e a posição da bala, e que disparam a bala;

Eduardo Macedo: 45% —> Criação da struct, função de criação de balas, ...

Peso relativo no projeto: 7%

3.2 —> Button.c / Button.h

Nestes ficheiros foi feita a criação e desenvolvimento da struct Button, que como o nome indica permite a criação de “botões” para serem apresentados nos menus, de forma a dar ao utilizador a capacidade de navegação entre eles. A struct contém, entre outros campos, as coordenadas e dimensões do botão, as suas imagens/pixmaps (uma imagem normal - unhighlighted, e uma “acesa” - highlighted), e um bool indicando se o cursor se encontra em cima do botão. Encontram-se também nestes ficheiros as típicas funções de criação, destruição, e modificação dos objetos desta struct, bem como alguns métodos get.

Contribuição:

Eduardo Ribeiro: 50% —> Criação da struct, funções de highlight/unhighlight, ...

Eduardo Macedo: 50% —> Funções de criação e destruição, de desenho, ...

Peso relativo no projeto: 3%

3.3 —> Cursor.c / Cursor.h

Nestes ficheiros está presente a struct Cursor, que representa um objeto “cursor”, que dá ao utilizador a capacidade de navegar nos menus, clicando em cima de um botão para selecionar a opção desejada. A struct contém as dimensões e coordenadas do cursor, bem como a sua imagem/pixmap e um bool indicando se o botão esquerdo está a ser premido. Mais uma vez, também podemos encontrar funções de criação, desenho, modificação e destruição de objetos desta struct.

Contribuição:

Eduardo Ribeiro: 55% —> Funções de desenhar no ecrã, update da posição, ...

Eduardo Macedo: 45% —> Criação da struct, e algumas funções

Peso relativo no projeto: 3%

3.4 —> Enemy.c / Enemy.h

Neste módulo foi feita a criação da struct Enemy, que caracteriza uma nave inimiga, bem como diversas funções para criação e destruição de inimigos e alteração do comportamento dos mesmos. Entre outros campos, a struct Enemy contém as dimensões e coordenadas do inimigo, a sua cor, velocidade, e imagens/pixmaps. Mais uma vez, também foram construídas funções get, de modo a facilitar o acesso à informação dos inimigos.

Contribuição:

Eduardo Ribeiro: 55% —> Criação da struct, e funções de criação de objetos, e animação dos mesmos;

Eduardo Macedo: 45% —> Funções get e de movimentação dos inimigos, ...

Peso relativo no projeto: 4%

3.5 —> Explosion.c / Explosion.h

Nestes ficheiros foi criado e implementado o funcionamento da struct Explosion, que caracteriza as explosões que ocorrem quando o jogador consegue destruir uma nave inimiga. Esta struct contém, entre outros aspetos, as coordenadas e dimensões da explosão, e as suas imagens/pixmaps. Funções de criação, destruição, desenho, e animação das explosões também podem ser encontradas nestes ficheiros.

Contribuição:

Eduardo Ribeiro: 0%

Eduardo Macedo: 100%

Peso relativo no projeto: 2%

3.6 —> Game.c / Game.h

Esta é a “classe”/struct principal do nosso jogo, bem como a mais complexa. Nestes ficheiros também está presente uma grande variedade de funções, que se dividem em duas partes: as funções relacionadas com o funcionamento do jogo, que vão desde as funções principais de “handling” de interrupções (funções Manager) até a funções de verificação de colisões (check_all_collisions(), entre outras) e desenho de objetos no ecrã (update_frame(), entre outras); e as funções relacionadas com o array de balas, este que caracteriza e engloba todas as balas presentes no ecrã, numa certa altura. Deste último tipo de funções fazem parte funções de adicionar e remover uma bala do array, de animação e movimentação de todas as balas, entre muitas outras.

De um modo geral, pode-se resumir o funcionamento do jogo e desta struct Game a estas funções:

- start_game() —> como o nome indica, faz tudo o que é necessário para o arranque do jogo/programa: cria a estrutura, inicializa vários campos da mesma, carrega as imagens necessárias, configura as máquinas de estado, subscreve os dispositivos necessários, etc.;
- play_game() —> é a função principal do programa. Contém o ciclo de interrupções que mantém o jogo em funcionamento, sendo que quando se dá uma interrupção de um periférico, na altura correta, esta função chama os “handlers” necessários, timeManager(), kbdManager() e mouseManager(), de modo a atualizar o jogo;
- end_game() —> termina o programa, basicamente fazendo o “inverso” do que a função start_game() fez: liberta toda a memória ocupada, faz unsubscribe aos dispositivos, etc.

Relativamente à struct em si, esta contém vários campos:

- irq_sets, para o timer, teclado e rato (são os dispositivos que são subscritos);
- o objeto que representa o cursor;
- o array das balas, bem como várias variáveis necessárias para o funcionamento e alteração deste;
- as imagens para as balas;
- o nível atual de dificuldade do jogo, que vai aumentando à medida que o utilizador progride no jogo;
- um objeto da struct Ships, que engloba a informação sobre os inimigos e explosões (irá ser discutida mais a frente);
- variáveis relacionadas com o ataque especial;
- todos os botões e sprites necessários para o funcionamento do jogo;
- um objeto da struct Highscores, que engloba toda a informação sobre as melhores pontuações (também irá ser discutida mais a frente);
- variáveis relacionadas com os mouse packets;
- variável englobando a informação das máquinas de estado (mais um assunto que irá ser discutido mais a frente);
- e por fim, uma variável que indica a pontuação atual.

Contribuição:

Eduardo Ribeiro: 60% —> Maior parte das funções “auxiliares”, funções Manager que controlam e gerem as interrupções dos periféricos;

Eduardo Macedo: 40% —> Funções de começo e término do jogo, algumas funções “auxiliares”.

Peso relativo no projeto: 20%

3.7 —> Highscores.c / Highscores.h

Nestes ficheiros encontram-se duas structs, Date e Highscores, que permitem gerir e guardar as melhores pontuações, e quando é que estas ocorreram (data e hora). Neste módulo, para além das funções de criação, destruição e atualização dos highscores, foram implementadas funções de escrita e leitura de ficheiros de texto (ficheiro hs.txt), de modo a guardar esta informação mesmo depois de o programa ser desligado, sendo que quando for ligado de novo a informação irá ser extraída do ficheiro. Também foram feitas algumas funções get.

Contribuição:

Eduardo Ribeiro: 100%

Eduardo Macedo: 0%

Peso relativo no projeto: 2%

3.8 —> i8042.h

Código proveniente dos labs 3 e 4, contendo macros para o teclado e rato.

Peso relativo no projeto: 1%

3.9 —> i8254.h

Código proveniente do lab 2, contendo macros para o timer.

Peso relativo no projeto: 1%

3.10 —> kbc_ih.S

Código proveniente do lab 3, contendo o IH em assembly para o teclado.

Peso relativo no projeto: 1%

3.11 —> keyboard.c / keyboard.h

Código proveniente do lab 3, contendo diversas funções relacionadas com o teclado.

Peso relativo no projeto: 1%

3.12 —> mouse_ih.S

Ficheiro contendo o IH em assembly para o rato.

Contribuição:

Eduardo Ribeiro: 0%

Eduardo Macedo: 100%

Peso relativo no projeto: 1%

3.13 —> mouse.c / mouse.h

Código proveniente do lab 4, contendo diversas funções relacionadas com o rato.

Peso relativo no projeto: 3%

3.14 —> proj.c

É o “ficheiro base” do nosso projeto, onde se encontra a função main (proj_main_loop()). Nele, são chamadas funções que preparam o começo do jogo, como a função que permite a utilização de operações em código assembly, e a função que ativa o modo gráfico. As funções start_game(), play_game() e end_game() são também chamadas para desencadear o funcionamento do jogo. No fim, retorna-se ao modo de texto.

Contribuição:

Eduardo Ribeiro: 100%

Eduardo Macedo: 0%

Peso relativo no projeto: 5%

3.15 —> projMacros.h

Ficheiro que contém várias macros relativas ao funcionamento do projeto, como por exemplo as posições iniciais para as balas e inimigos, a velocidade inicial dos inimigos, a duração, em segundos, do ataque especial, entre muitas outras.

Contribuição:

Eduardo Ribeiro: 50% —> macros relativas as balas e highscores, entre outras;

Eduardo Macedo: 50% —> macros relativas aos inimigos e botões, entre outras.

Peso relativo no projeto: 2%

3.16 —> rtc.c / rtc.h

Este é o módulo que contém as funções que estão relacionadas diretamente com o Real Time Clock. Estão presentes as funções de extração da data e hora, bem como de verificação de updates, e distinção entre valores em BCD ou binário. Acrescentamos também uma função de conversão de valores, de BCD para binário.

Contribuição:

Eduardo Ribeiro: 60% —> funções de extração da data, de verificação de update, e de distinção entre BCD e binário

Eduardo Macedo: 40% —> funções de extração da hora e conversão BCD para binário.

Peso relativo no projeto: 4%

3.17 —> Ships.c / Ships.h

Este é, provavelmente, o segundo módulo mais importante do nosso projeto (sendo que Game é o primeiro). Estes ficheiros contém uma struct, Ships, que controla toda a informação não só dos inimigos, mas também das explosões. Ela tem vários campos, sendo os principais um array de inimigos, e um array de explosões, que contém todos os inimigos e explosões presentes no ecrã, num determinado momento. A struct possui também todas as imagens necessárias para a geração de naves inimigas e explosões, bem como algumas variáveis relacionadas com o comportamento das naves inimigas, como por exemplo a sua velocidade vertical.

Algumas funções que podemos encontrar neste módulo são: funções de criação e destruição de objetos do tipo Ships, função que adicionam e removem inimigos e explosões dos seus respetivos arrays, funções de desenho, movimentação e animação, entre outras. Algumas funções get também foram implementadas.

Contribuição:

Eduardo Ribeiro: 55% —> Criação da struct, e funções de manipulação do array de inimigos, entre outras;

Eduardo Macedo: 45% —> Funções de manipulação do array de explosões, entre outras.

Peso relativo no projeto: 11%

3.18 —> sprite.c / sprite.h

Este módulo contém a struct Sprite, utilizada para caracterizar, de um modo geral, objetos que não pertencem a nenhuma outra struct, como, por exemplo, os números e caracteres, a nave do jogador e a mira, as barras que mostram o progresso do ataque especial, os fundos, etc. A struct contém, como atributos, a posição e dimensões do sprite, a velocidade do mesmo, e a imagem associada. Os ficheiros contém também as típicas funções de criação, destruição, alteração e desenho de objetos; para além disso, está presente uma função exclusiva ao sprite da mira, que recebe como argumento um mouse packet, e atualiza a posição da mesma (update_aim()). Por fim, algumas funções get foram implementadas.

Contribuição:

Eduardo Ribeiro: 30% —> Função de desenho do sprite, e funções get;

Eduardo Macedo: 70% —> Funções de criação e destruição de sprites, e função de update da mira da nave do jogador.

Peso relativo no projeto: 3%

3.19 —> StateMachine.c / StateMachine.h

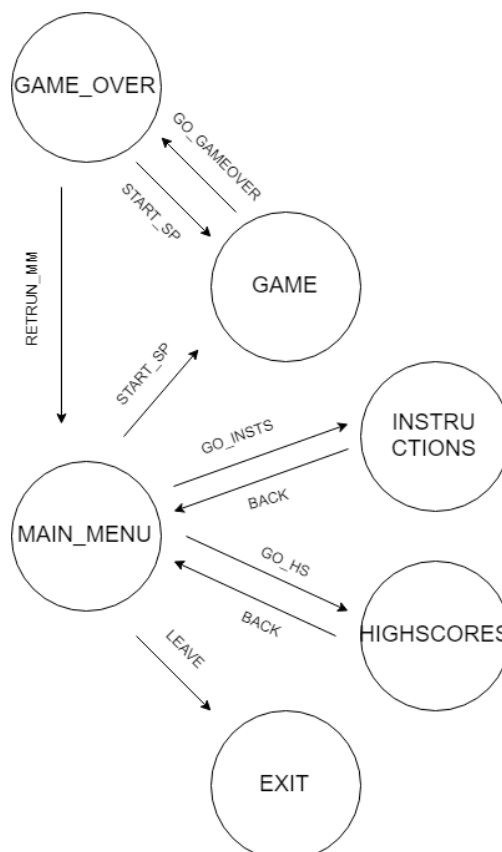
Este é um módulo muito importante, pois permitiu-nos a simplificação do código e uma melhor compreensão de como iria ser a estrutura final do jogo. No nosso projeto, existem duas máquinas de estado, uma relativa ao funcionamento geral do jogo, ou seja, relativamente ao menu em que o jogo se encontra num determinado momento, e outra relativa às ações do jogador, que indica, entre outras coisas, se este perdeu o jogo e se deve ser redirecionado para o Game Over screen.

Contém uma struct, StateMachines, que tem 4 campos: o estado atual da máquina de estado do jogo, o estado atual da máquina de estado do jogador, o último evento relativo ao jogo, e o último evento relativo ao jogador.

Relativamente à máquina de estado principal, relacionada com o jogo, temos:

Estados:

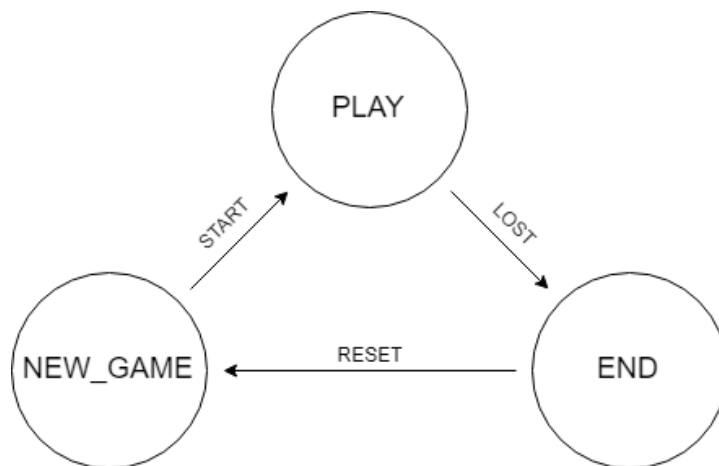
- MAIN_MENU: quando o jogo está no menu principal;
- GAME: quando um jogo está a decorrer;
- GAME_OVER: quando o jogo acaba, e nos encontramos no menu de Game Over;
- INSTRUCTIONS: quando o jogo está no menu das instruções;
- HIGHSCORES: quando o jogo está no menu dos highscores;
- EXIT: estado final, que indica a saída e o fecho do programa.



Relativamente à máquina de estado relacionada com o jogador, temos:

Estados:

- NEW_GAME: quando é começado um novo jogo, em que se dá “reset” a várias variáveis de modo a efetuar a preparação para o início do mesmo;
- PLAY: enquanto o jogador está a jogar, e ainda não perdeu;
- END: quando o jogador perde.



Em termos de funções, implementamos métodos que retornam o estado atual de cada uma das máquinas de estado, bem como funções que permitem modificar os eventos atuais. Temos, depois, as funções de “handle” de eventos, que são as máquinas de estado em si, que consoante o evento e o estado atual, efetuam (ou não) uma transição de estado. Existem também as funções de destruição e criação de objetos da struct, esta última que inicia as máquinas de estado, ficando nos respetivos estados iniciais, MAIN_MENU e NEW_GAME.

Contribuição:

Eduardo Ribeiro: 100%

Eduardo Macedo: 0%

Peso relativo no projeto: 8%

3.20 —> timer_ih.S

Ficheiro que contém o IH em assembly para o timer.

Contribuição:

Eduardo Ribeiro: 0%

Eduardo Macedo: 100%

Peso relativo no projeto: 1%

3.21 —> timer.c / timer.h

Código proveniente do lab 2, contendo diversas funções relacionadas com o timer.

Peso relativo no projeto: 6%

3.22 —> v_macros.h

Código proveniente do lab 5, contendo macros relacionadas com a placa gráfica.

Peso relativo no projeto: 1%

3.23 —> video.c / video.h

Código proveniente do lab 5, contendo diversas funções relacionadas com a placa gráfica; foram também acrescentadas algumas funções relacionadas com a técnica de double buffering.

Peso relativo no projeto: 3%

3.24 —> xpms.h

Ficheiro que contém os “includes” de todos os ficheiros xpms de imagens utilizadas no nosso projeto.

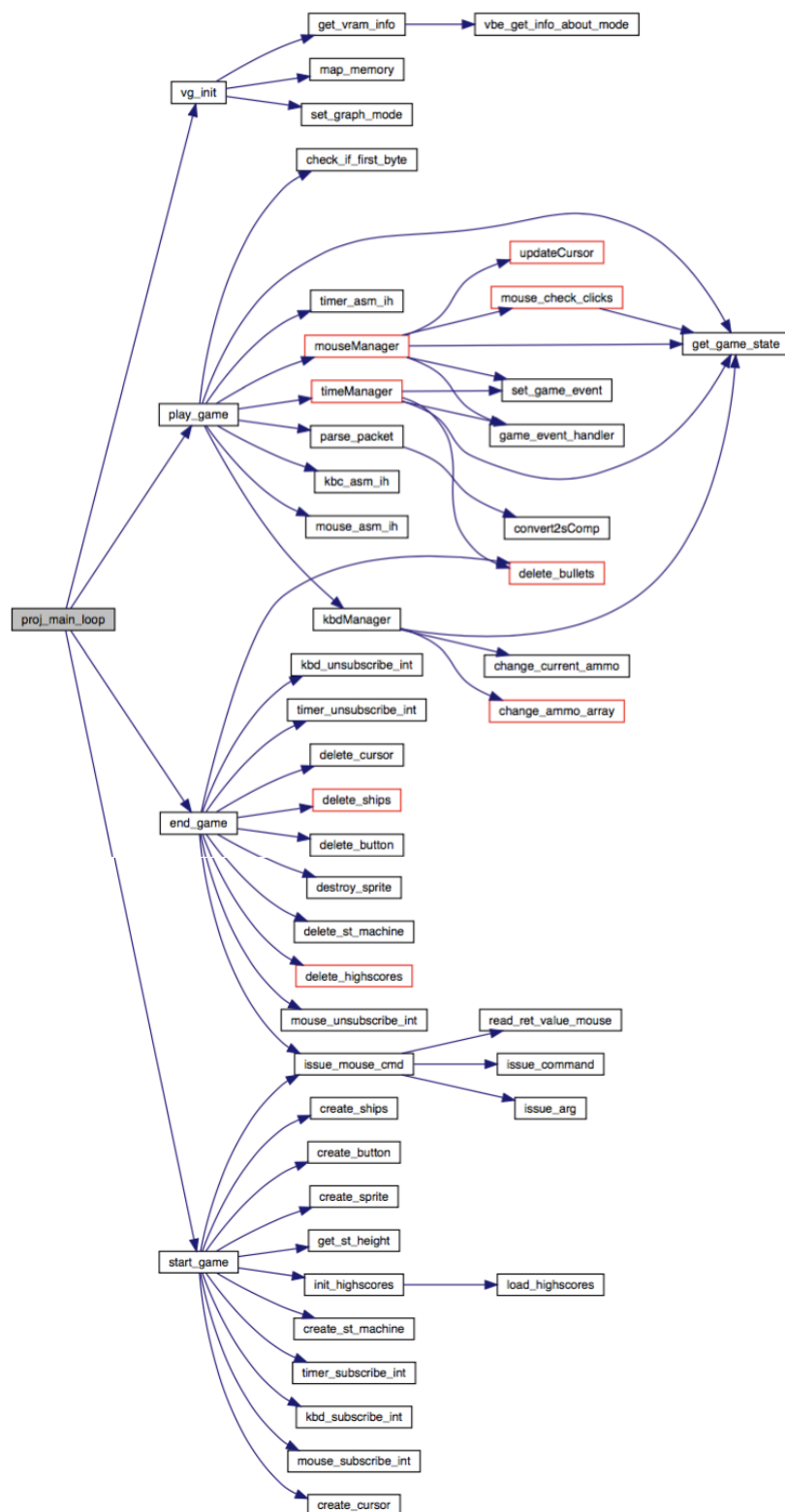
Peso relativo no projeto: 7%

Contribuição de cada estudante para a totalidade do projeto:

Eduardo Ribeiro: 58%

Eduardo Macedo: 42%

3.25 —> Call Graph



NOTA: Este gráfico pode não apresentar toda a informação necessária; para mais informação, por favor consultar os gráficos presentes na documentação de cada função.

NOTA: As funções principais do programa já foram descritas ao longo do relatório, como por exemplo na descrição dos ficheiros Game.

Capítulo 4 —> Detalhes de Implementação

Achamos relevante começar esta secção por mencionar a estrutura geral do nosso código, e a robustez do mesmo. Um detalhe importante relativo a este tema é o facto de todas as structs estarem implementadas nos ficheiros .c respetivos, e não nos header files; em vez disso, nos ficheiros .h apenas se declaram as structs, e faz-se um typedef para melhor legibilidade e uma utilização mais prática do nome das structs. A razão para se fazer a implementação das structs nos ficheiros .c é esconder os detalhes sobre as mesmas, fazendo com que os campos de cada struct deixem de ser acessíveis fora dos ficheiros .c respetivos: é o chamado encapsulamento de informação. Como o objetivo deste conceito é o acesso à informação das structs exclusivamente através das suas funções/métodos, foram feitas funções get para cada struct para tal fim.

Como recomendado, e ainda continuando no tema da estrutura do código, foi feita a construção do código por camadas, separando as funções e as structs por ficheiros, resultando assim num código menos denso e mais fácil de compreender e interpretar.

Outro aspeto importante e que merece ser mencionado é a utilização de máquinas de estado e o handling de eventos que é feito no nosso projeto. Ao invés de optarmos pela construção e utilização de apenas uma máquina de estado, decidimos que seria uma melhor opção a utilização de duas: uma relativa ao jogo (se estamos no modo das instruções, no menu principal, a jogar, etc), e a outra relativa às ações do jogador (se está a jogar, se perdeu o jogo, etc). Isto permitiu-nos simplificar o código e criar um jogo fluído e consistente.

Mais um detalhe importante (e que não foi visto nas aulas...), já acima mencionado, é a utilização de dois tipos diferentes de verificação de colisões: ao nível das coordenadas, e ao nível dos pixels. O uso desta última faz com que o jogo se torne mais autêntico, apenas detectando uma colisão quando realmente esta acontece; o uso da primeira, no entanto, é essencial, pois a verificação dos pixels pode ser uma tarefa pesada e apenas deve ser feita quando se deteta uma colisão ao nível das coordenadas.

O nosso fundo/background dinâmico é outro aspeto que gostaríamos de mencionar. Embora a sua implementação não seja necessariamente complicada, achamos que é uma parte essencial do nosso trabalho e do nosso tema, dando a sensação que as naves realmente se deslocam.

Embora não tenha sido abordado nas aulas, achamos que seria correta a utilização de ficheiros de texto para guardar as informações sobre os highscores, sempre que o nosso programa é desligado. Desta maneira, estes não se perdem, sendo que quando o programa for ligado de novo, a informação do ficheiro é carregada para o jogo.

Terminaremos este capítulo por falar na maneira como acedemos o RTC, apenas quando este não está a realizar um update, o que poderia interferir com os valores extraídos, e na construção de funções assembly para os interrupt handlers, que fazem operações de input e output de baixo nível, e na utilização dos mesmos, “misturados” com o resto do código, em C.

Capítulo 5 —> Conclusão

Gostaríamos de começar por dizer que a cadeira de LCOM foi, até agora, a cadeira mais difícil e desafiante que tivemos que fazer; no entanto, também foi uma das mais interessantes e entusiasmantes.

Apesar disso, na nossa opinião, há uma grande diferença entre a dificuldade dos labs e a dificuldade e complexidade que o projeto exige. Existe uma grande variedade de tópicos importantes para o projeto final que não foram abordados nas aulas (pelo menos não o suficiente) e, embora sabendo que o tempo não permite cobrir toda a matéria, achamos que alguns deles deveriam ter sido mais discutidos, como é o caso das state machines e handle de eventos, colisões, etc.