Faculdade de Engenharia da Universidade do Porto

Pong 2. 0.

Projeto de Laboratório de Computadores

Licenciatura em Engenharia Informática e Computação

2022/2023

2º Ano - 2º Semestre

Turma 8 – Grupo 3

André Dantas Rodrigues	up202108721@fe.up.pt
Ian da Nóbrega Beltrão	up202102360@fe.up.pt
Lucas Borborema Nakajima	up202001337@fe.up.pt
Manuel Ramos Leite Carvalho Neto	up202108744@fe.up.pt

Índice

Índice de Figuras

1. Instruções de Utilização

1.1. Menu Principal

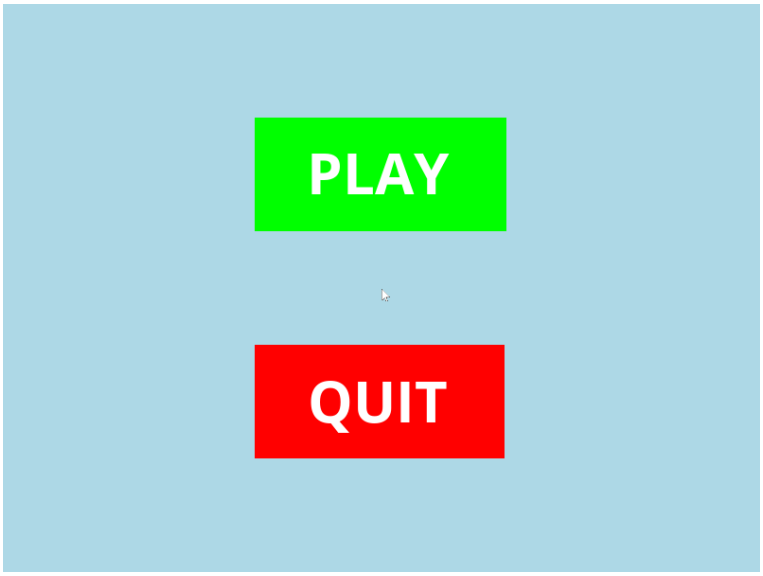

Figura 1 - Menu Principal

Ao iniciar o projeto, surge o ecrã com o menu principal do jogo. Neste ecrã, é possível selecionar uma de duas opções: *play* – para passar para o ecrã de seleção do nível/dificuldade do jogo – e *quit* – para terminar a execução do programa.

A seleção da opção pretendida pode ser realizada utilizando o rato, movendo o cursor para a opção a selecionar e carregando no botão do lado esquerdo, ou utilizando o teclado, através das teclas P ou 1 (para selecionar a opção *play*) e Q ou 2 (para selecionar a opção *quit*). É também possível terminar a execução do programa através da tecla ESC.

Conforme é possível verificar nas imagens abaixo, quando uma opção é selecionada, o botão correspondente é destacado.

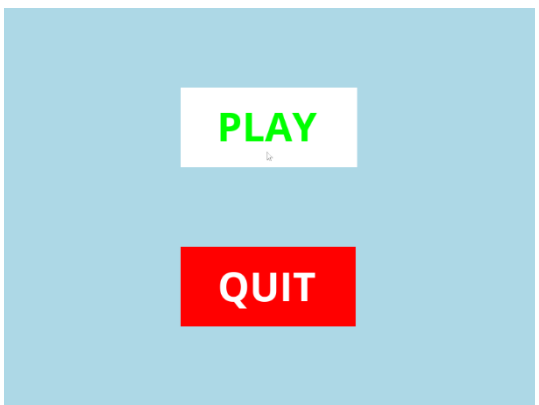

Figura 2 - Play

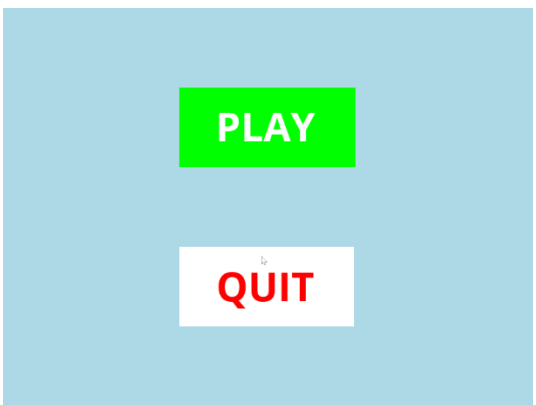

Figura 3 - Quit

1.2. Escolha do Nível/Dificuldade

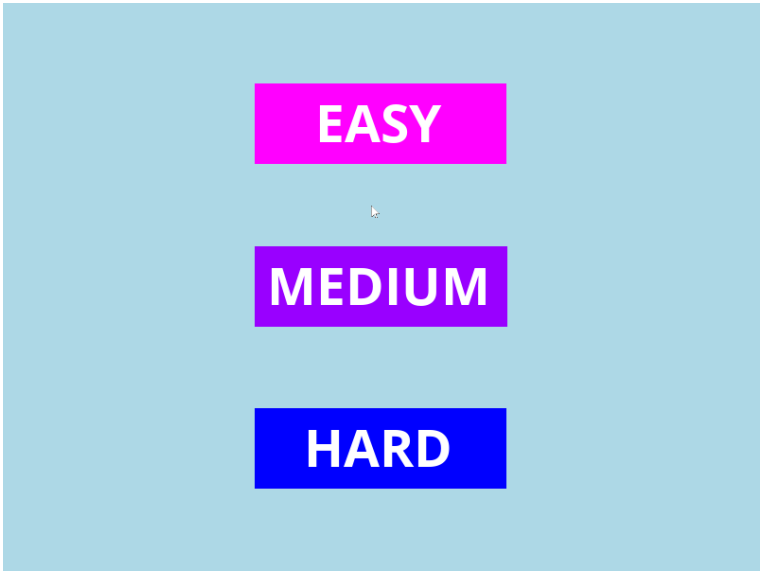

Figura 4 - Escolha do Nível/Dificuldade

Selecionada a opção *play* no menu principal, surge o ecrã para escolha do nível/dificuldade do jogo. Neste ecrã, é possível escolher uma de três opções: *easy* – para um jogo fácil –, *medium* – para um jogo de dificuldade média – e *hard* – para um jogo difícil. A explicação das diferenças entre cada dificuldade consta da próxima subsecção.

Tal como no menu principal, a seleção da dificuldade pretendida pode ser realizada através do rato ou através do teclado. Através do rato, a opção desejada é selecionada movendo o cursor para cima dela e premindo o botão do lado esquerdo. Através do teclado, cada dificuldade é selecionada através da letra inicial ou do número correspondente, isto é, a opção *easy* é selecionada carregando na tecla E ou na tecla 1, a opção *medium* é selecionada carregando na tecla M ou na tecla 2 e a opção *hard* é selecionada carregando na tecla H ou na tecla 3. É igualmente possível voltar ao menu principal (ecrã anterior), carregando na tecla ESC.

De forma semelhante ao que sucede no menu principal, assim que uma dificuldade é selecionada, a opção correspondente é salientada visualmente.

Figura 5 - Easy

Figura 6 - Medium

Figura 7 - Hard

1.3. Jogo

Finalmente, escolhida uma dificuldade, inicia-se o jogo propriamente dito. O jogo consiste numa versão – adaptada para um jogador – do tradicional *Pong* da *Atari*. Nesta recriação do jogo original, o objetivo é aguentar o máximo de tempo possível com a bola dentro dos limites do ecrã, tendo, para isso, uma "parede" móvel e controlável na qual a bola pode embater e ressaltar sem sair do ecrã. Esta "parede" aparece do lado esquerdo do ecrã, sendo este o único lado por onde a bola pode sair do campo de jogo, dado que nos outros três lados o movimento da bola é limitado pela fronteira do ecrã, ocorrendo uma colisão perfeitamente elástica em cada choque.

Tal como consta na subsecção anterior, existem três dificuldades possíveis para o jogo. A variação da dificuldade tem duas implicações: a velocidade inicial da bola e o comprimento da "parede". Assim sendo, quanto maior for a dificuldade escolhida, maior será a velocidade inicial da bola e menor será o comprimento da "parede", conforme é possível observar nas imagens constantes do final desta subsecção.

Para além da bola e da "parede", existem mais dois elementos presentes no ecrã do jogo: o multiplicador (no canto superior esquerdo) e a pontuação atual (no canto superior direito). O multiplicador corresponde à velocidade atual da bola, que aumenta a partir de 1 (a velocidade mais lenta possível – velocidade inicial no modo de jogo fácil). A pontuação atual corresponde ao número de vezes que a bola já embateu na "parede" móvel, de acordo com o multiplicador no instante da colisão. Isto é, uma colisão entre a bola e a "parede" com o multiplicador a 1 soma um ponto à pontuação total acumulada, enquanto uma colisão com o multiplicador a 2 soma 2 pontos, e assim sucessivamente (uma colisão com o multiplicador a x soma x pontos).

A "parede" móvel deve ser controlada pelo utilizador através do teclado, em particular das teclas das setas (para cima ou para baixo, tendo a "parede" o movimento no sentido correspondente) ou W (para um movimento da "parede" para cima) ou S (para um movimento da "parede" para baixo).

A velocidade da bola pode variar por dois motivos: ou ao fim de um determinado tempo, ou por decisão do utilizador. Por um lado, de maneira que a dificuldade do jogo aumente progressivamente ao longo do tempo, a velocidade da bola vai aumentando de acordo com o tempo passado – a velocidade 1 demora 10 segundos até passar para a velocidade 2, a velocidade 2 demora 20 segundos até passar para a velocidade 3, e assim por diante (a velocidade x demora $10x$ segundos até transitar para a velocidade $x + 1$). Por outro lado, se o jogador já se sentir confortável com a velocidade atual da bola e pretender tentar melhorar a sua pontuação, pode carregar no botão do lado direito do rato para que a velocidade aumente uma unidade. Evidentemente, em qualquer caso, o aumento da velocidade da bola é acompanhado por um aumento do valor do multiplicador na mesma medida.

Quando o jogo terminar, isto é, quando o jogador não conseguir impedir que a bola saia dos limites do ecrã pelo lado esquerdo, o programa regressa ao menu principal.

Se, a qualquer momento, o utilizador pretender voltar ao ecrã de escolha do nível/dificuldade, basta carregar na tecla ESC.

Figura 8 - Jogo Fácil

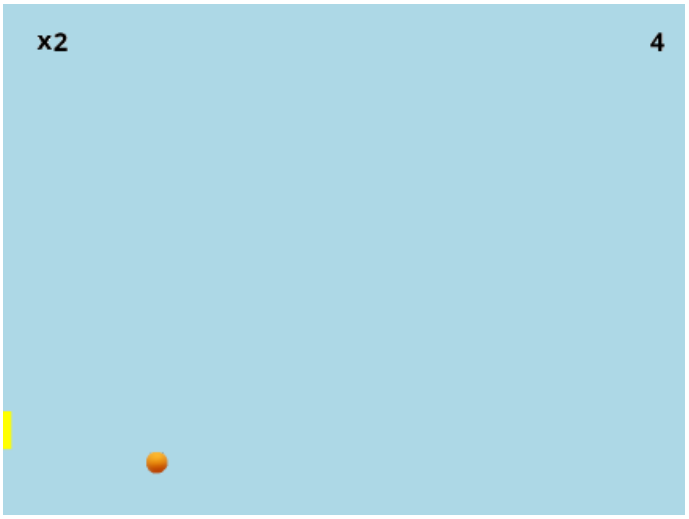

Figura 9 - Jogo de Dificuldade Média

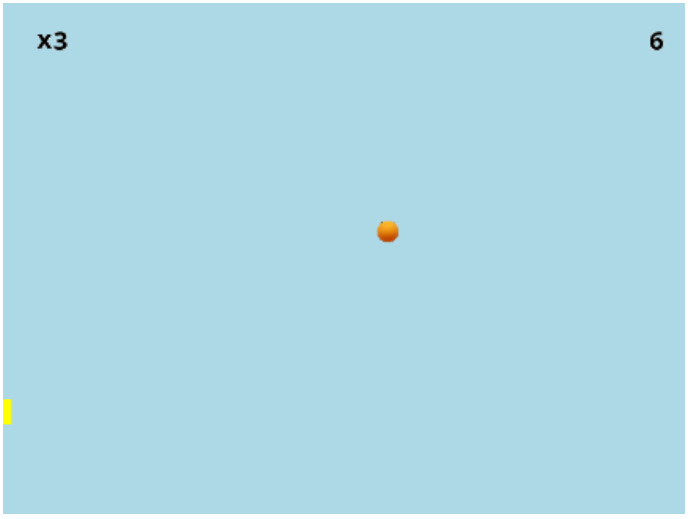

Figura 10 - Jogo Difícil

2. Estado do Projeto

2.1. Funcionalidades Implementadas

Todas as funcionalidades apresentadas na secção anterior foram implementadas, mas, por limitações de tempo, nem todas as descritas na proposta do projeto conseguiram ser realizadas. Em particular, acabaram por não ser integrados o RTC e a porta série, o que impossibilitou o desenvolvimento de um modo para dois jogadores e do registo da data/hora de recordes (pontuações máximas obtidas no modo para um jogador) anteriores.

Deste modo, procurou-se acrescentar mais algumas funcionalidades (como a possibilidade de escolher o nível/dificuldade do jogo, de aumentar manualmente a velocidade da bola e o multiplicador para cálculo da pontuação) para valorizar o trabalho, ainda que não na mesma medida em que a implementação do RTC e da porta série o fariam. A par destas funcionalidades, tentou-se também aumentar a robustez do produto desenvolvido, isto é, a inexistência de falhas ou comportamentos inesperados e, principalmente, permitir ao utilizador realizar as mesmas ações de múltiplas maneiras diferentes, nomeadamente através do rato ou do teclado (até com teclas distintas para a mesma função).

Em síntese, como já foi descrito, foi implementado um menu principal, um menu de escolha do nível/dificuldade e um jogo propriamente dito. Em qualquer um dos três ecrãs/estados possíveis, o rato e o teclado têm sempre alguma função, bem como o *timer*, tal como será explicado de forma mais detalhada nas subsecções seguintes.

Dispositivo	Propósito	Interrupções?
Timer	Controlar a frequência do jogo – a troca dos *buffers* Controlar a (alteração da) velocidade da bola e o tempo do jogo	Sim
Teclado	Navegar nos menus e selecionar uma opção Possibilitar voltar ao estado/ecrã anterior Mover a “parede” para cima ou para baixo	Sim
Rato	Navegar nos menus e selecionar uma opção Possibilitar o aumento da velocidade da bola manualmente	Sim
Placa Gráfica	Desenhar/Mostrar todos os ecrãs, botões/opções, cursor e elementos do jogo (bola, “parede”, multiplicador e pontuação atual)	Não

2.2. *Timer*

O *timer* foi utilizado para gerar interrupções periódicas (com uma frequência de 60 vezes por segundo, $60\ Hz$), que determinam o momento de troca dos *buffers* e, por isso, o *frame-rate* do projeto. Para além disto, as interrupções do *timer* serviram também para definir os momentos de incremento da velocidade da bola no jogo, através da contagem do tempo do jogo a decorrer.

Este dispositivo é, assim, utilizado ao longo de toda a execução do programa, desde a configuração da frequência (*timer_set_frequency()*) e subscrição das interrupções (*timer_subscribe_int()*) em *start()* (*proj.c*), passando pelo lidar com as interrupções em *loop()* (*proj.c*), *timer_interrupt_handler()* (*model.c*) e *timer_game_handler()* (*game.c*), até anular a subscrição das interrupções (*timer_unsubscribe_int()*) em *end()* (*proj.c*).

O módulo *timer.c* contém a implementação das funções mencionadas (*timer_set_frequency()*, *timer_subscribe_int()* e *timer_unsubscribe_int()*), bem como outra função útil (*timer_get_conf()*).

2.3. Teclado

O teclado foi utilizado para controlo do jogo e da aplicação. Para controlo do jogo, são utilizadas as teclas da seta para cima e da letra W para mover a “parede” para cima e as teclas da seta para baixo e da letra S para mover a “parede” para baixo. Para controlo da aplicação, a tecla ESC é utilizada para, em qualquer momento, voltar ao estado/ecrã antecessor (do jogo para a escolha do nível/dificuldade, da escolha do nível/dificuldade para o menu e do menu para o fim da execução do programa), e as teclas dos números (1, 2 ou 3) ou de algumas letras (P, Q, E, M ou H) são utilizadas para selecionar a opção correspondente (por ordem ou por letra inicial) apresentada no ecrã do menu principal e de escolha do nível/dificuldade.

Este dispositivo é, por isso, utilizado durante toda a execução do projeto, desde a subscrição das interrupções (*keyboard_subscribe_int()*) em *start()* (*proj.c*), passando pelo lidar com as interrupções em *loop()* (*proj.c*) e *keyboard_interrupt_handler()* (*model.c*), interpretação dos *scancodes* recebidos em *keyboard_menu_handler()* (*menu.c*), *keyboard_levels_handler()* (*levels.c*) e *keyboard_game_handler()* (*game.c*), até anular a subscrição das interrupções em (*keyboard_unsubscribe_int()*) em *end()* (*proj.c*).

O módulo *keyboard.c* contém a implementação das funções referidas diretamente (*keyboard_subscribe_int()* e *keyboard_unsubscribe_int()*) e indiretamente (*kbc_ih()* e *keyboard_read_scancode_byte()*), bem como outras funções úteis (*keyboard_restore()* e *keyboard_enable_interrupts()*).

2.4. Rato

Quanto ao rato, foi utilizada a posição e os botões. A posição do rato foi usada no menu principal e na escolha do nível/dificuldade, para avaliar se o cursor se encontra

em cima de uma opção, selecionando-a. Os botões do rato foram utilizados com dois propósitos diferentes: no menu principal e na escolha do nível/dificuldade, o botão do lado esquerdo (quando premido uma vez) é utilizado para confirmar a seleção da opção sobre a qual o cursor se encontra; no jogo propriamente dito, o botão do lado direito (quando premido uma vez) é utilizado para aumentar a velocidade da bola.

O rato é, por isso, utilizado ao longo de todo o programa, desde a subscrição das interrupções (*mouse_data_reporting()* e *mouse_subscribe_int()*) em *start()* (*proj.c*), passando pelo lidar com as interrupções em *loop()* (*proj.c*) e *mouse_interrupt_handler()* (*model.c*), interpretação dos pacotes recebidos em *mouse_menu_handler()* (*menu.c*), *mouse_levels_handler()* (*levels.c*) e *mouse_game_handler()* (*game.c*), até anular a subscrição das interrupções em (*mouse_unsubscribe_int()* e *mouse_data_reporting()*) em *end()* (*proj.c*).

O módulo *mouse.c* contém a implementação das funções mencionadas diretamente (*mouse_subscribe_int()*, *mouse_unsubscribe_int()* e *mouse_data_reporting()*) e não diretamente (*mouse_ih()* e *mouse_read_packet_byte()*), bem como outras funções úteis (*mouse_restore()* e *mouse_write_command()*).

2.5. Placa Gráfica

No que concerne à placa gráfica, foi utilizado o modo *0x115*, com resolução de 800 pixels por 600 pixels (800×600), modo de cor direta e 24 bits por pixel (RGB 8:8:8), o que se traduz em $2^8 \times 2^8 \times 2^8 = 2^{24} = 16777216$ cores. Foi também utilizado *double buffering*, via *page flipping*. Foram utilizados objetos em movimento com *sprites* e, pese embora não tenham sido utilizadas fontes, foram utilizados os dígitos desde 0 até 9. No que concerne a funções VBE, foram utilizadas as funções *0x02* – "Set VBE Mode" (para definir o modo gráfico pretendido) e *0x07* – "Set/Get Display Start", subfunções *0x00* – "Set Display Start" e *0x01* – "Get Display Start" (para implementar *page flipping*).

Evidentemente, a placa gráfica foi utilizada para desenhar o *frame* atual, isto é, o estado atual do programa, pintando/desenhando os pixels desde o fundo do ecrã até aos elementos mais anteriores. Conforme já foi referido, foi implementado *double buffering* via *page flipping* e desenho de *sprites*.

Assim sendo, a placa gráfica é utilizada durante todo o projeto, desde a inicialização em modo gráfico (*video_init()*) em *start()* (*proj.c*), passando pelo desenho de cada *frame* em *draw_frame()* (*view.c*), do menu em *draw_menu()* (*menu_view.c*), da escolha do nível/dificuldade em *draw_levels()* (*levels_view.c*), do jogo propriamente dito em *draw_game()* (*game_view.c*), dos botões em *draw_button()* (*button_view.c*), do cursor em *draw_cursor()* (*cursor_view.c*), da bola em *draw_ball()* (*ball_view.c*), da "parede" em *draw_wall()* (*wall_view.c*), do multiplicador em *draw_times()* (*sprite_view.c*) e *draw_digit()* (*sprite_view.c*), da pontuação em *draw_number()* (*sprite_view.c*) e de um qualquer *sprite* em *draw_sprite()* (*sprite_view.c*), incluindo a troca dos *buffers* via *page flipping* (*video_swap_buffers()*) em *timer_interrupt_handler()* (*model.c*), até ao regresso ao modo de texto (*vg_exit()*) em *end()* (*proj.c*).

O módulo *video.c* contém a implementação das funções descritas de forma direta (*video_init()* e *video_swap_buffers()*) e não direta (*video_draw_pixmap()*), bem como outras funções úteis (*video_draw_pixel()*, *video_draw_hline()*, *video_draw_rectangle()* e *video_draw_background()*).

3. Organização e Estrutura do Código

3.1. Arquitetura

O código está organizado/estruturado segundo uma arquitetura baseada em MVC – "Model-View-Controller". Deste modo, os módulos em *model* são responsáveis por representar os dados e a informação, conhecendo os objetos e os estados. Por sua vez, os módulos em *view* apenas tratam de desenhar a informação modelada. Finalmente, os módulos em *controller* são os *drivers* dos dispositivos de entrada e saída, que interpretam as ações do utilizador e alteram o modelo/estado do programa.

3.2. *proj.c*

Este módulo inicia o programa (subscreve as interrupções e configura os dispositivos, nomeadamente o modo gráfico da placa de vídeo), recebe e lida com as interrupções dos dispositivos enquanto o programa corre e, no final, termina-o, anulando a subscrição das interrupções e regressando ao modo de texto.

Peso no Projeto: 5 %

Neste módulo, não existem estruturas de dados a descrever.

3.3. *Model*

3.3.1. *model.c*

Este módulo é responsável por armazenar a informação de mais "alto-nível" do programa, nomeadamente o estado atual, e, de acordo com esse estado atual, delegar a gestão da interrupção de cada dispositivo para o módulo apropriado. O ficheiro contém código para iniciar cada estado do programa (menu principal, escolha do nível/dificuldade e jogo propriamente dito).

Peso no Projeto: 5 %

A única estrutura de dados relevante é a *enum* com o estado do jogo, que pode tomar um de quatro valores: MENU, LEVELS, GAME e END, considerados autoexplicativos (os três primeiros correspondem às três primeiras subsecções do relatório e o quarto ao término da execução do programa).

3.3.2. *ball.c*

Este módulo é responsável pela bola, ou seja, sabe construí-la, movê-la (incluindo após uma colisão com o limite do ecrã ou com a "parede" móvel) e destruí-la.

Peso no Projeto: 3 %

A bola é uma *struct* com cinco campos: a coordenada horizontal (*x*), a coordenada vertical (*y*), a velocidade horizontal (*vx*), a velocidade vertical (*vy*) e o *sprite*.

3.3.3. *button.c*

Este módulo é responsável por cada botão enquanto objeto, ou seja, sabe construir um botão, verificar se dadas coordenadas (*x*, *y*) se encontram sobre um botão e destruir um botão.

Peso no Projeto: 3 %

Um botão é uma *struct* com sete campos: as coordenadas horizontal (*x*) e vertical (*y*) do canto superior esquerdo do botão, a sua largura (*w*) e altura (*h*), um atributo (*selected*) que indica se o botão está selecionado ou não, e dois *sprites*, um para o estado "normal" do botão e outro para quando selecionado.

3.3.4. *cursor.c*

Este módulo é responsável pelo cursor, ou seja, sabe construí-lo, movê-lo para umas dadas coordenadas e destruí-lo.

Peso no Projeto: 3 %

O cursor é uma *struct* com três campos: as suas coordenadas horizontal (*x*) e vertical (*y*), bem como o seu *sprite*.

3.3.5. *game.c*

Este módulo é responsável pelo jogo propriamente dito, pelo que sabe iniciá-lo (num dado ecrã e para uma dada dificuldade, construindo a bola e a parede), lidar com as interrupções de cada dispositivo (*timer*, teclado e rato, conforme explicado anteriormente), verificar se o jogo deve terminar (quando a bola ultrapassa o limite esquerdo do ecrã) e terminá-lo (destruir a bola e a parede).

Peso no Projeto: 5 %

Este módulo não define novas estruturas de dados, apenas inclui uma bola e uma parede, definidas nos respetivos módulos.

3.3.6. levels.c

Este módulo é responsável pela escolha do nível/dificuldade, pelo que sabe iniciar esse estado (num dado ecrã, construindo os botões com as opções e o cursor) lidar com as interrupções de cada dispositivo (teclado e rato, conforme explicado anteriormente), verificar se algum botão está selecionado e terminar o estado de escolha do nível/dificuldade (destruir os botões e o cursor).

Peso no Projeto: 4 %

Este módulo não define novas estruturas de dados, apenas inclui três botões e um cursor, definidos nos respetivos módulos.

3.3.7. menu.c

Este módulo é responsável pelo menu principal, pelo que sabe iniciá-lo (num dado ecrã, construindo os botões com as opções e o cursor), lidar com as interrupções

de cada dispositivo (teclado e rato, conforme explicado anteriormente), verificar se algum botão está selecionado e terminar o menu principal.

Peso no Projeto: 4 %

Este módulo não define novas estruturas de dados, apenas inclui dois botões e um cursor, definidos nos respetivos módulos.

3.3.8. sprite.c

Este módulo é responsável pelos *sprites*, isto é, por construí-los (dado um XPM e uma cor de fundo para transparência) e destruí-los, bem como por construir e destruir todos os *sprites* com dígitos e com o sinal de multiplicação (para o multiplicador).

Peso no Projeto: 5 %

Um *sprite* é uma *struct* com quatro campos: a altura (*height*), a largura (*width*), um *array* com as cores (*colors*) e a cor de fundo para transparência (*background*).

3.3.9. wall.c

Este módulo é responsável pela "parede" móvel, ou seja, sabe construí-la, movê-la (para cima ou para baixo) e destruí-la.

Peso no Projeto: 3 %

A "parede" é uma *struct* com quatro campos: as coordenadas horizontal (*x*) e vertical (*y*) do seu canto superior esquerdo, a largura (*w*) e a altura (*h*).

3.4. *View*

Uma vez que os módulos do *view* não armazenam/representam qualquer informação que não seja importada do *model*, em nenhum dos módulos desta subsecção existem estruturas de dados a assinalar.

3.4.1. *view.c*

Este módulo é responsável por representar o jogo no ecrã, isto é, desenhar o fundo e o *frame* de acordo com o estado atual. Na verdade, de acordo com o estado, deve delegar o desenho do *frame* para o módulo apropriado.

Peso no Projeto: 4 %

3.4.2. *ball_view.c*

Este módulo é responsável por desenhar uma bola nas suas coordenadas. Na verdade, delega o desenho do *sprite* da bola para o módulo *sprite_view*.

Peso no Projeto: 2 %

3.4.3. *button_view.c*

Este módulo é responsável por desenhar um botão nas suas coordenadas, de forma diferente dependendo do seu estado interno (selecionado ou não selecionado). Na verdade, delega o desenho do *sprite* do botão para o módulo *sprite_view*.

Peso no Projeto: 2 %

3.4.4. *cursor_view.c*

Este módulo é responsável por desenhar um cursor nas suas coordenadas. Na verdade, delega o desenho do *sprite* do cursor para o módulo *sprite_view*.

Peso no Projeto: 2 %

3.4.5. *game_view.c*

Este módulo é responsável por desenhar o estado atual do jogo: desenha o multiplicador, a pontuação, a bola e a parede, delegando cada elemento para o módulo adequado.

Peso no Projeto: 3 %

3.4.6. levels_views.c

Este módulo é responsável por desenhar o ecrã de escolha do nível/dificuldade: desenha os botões e o cursor, delegando cada elemento para o módulo adequado.

Peso no Projeto: 3 %

3.4.7. menu_view.c

Este módulo é responsável por desenhar o menu principal: desenha os botões e o cursor, delegando cada elemento para o módulo adequado.

Peso no Projeto: 3 %

3.4.8. sprite_view.c

Este módulo é responsável por desenhar um *sprite* (seja qualquer *sprite*, seja o símbolo de multiplicação, seja um dígito, seja um número) em dadas coordenadas.

Peso no Projeto: 4 %

3.4.9. wall_view.c

Este módulo é responsável por desenhar uma “parede” nas suas coordenadas, desenhando um retângulo com a largura e a altura da parede e uma dada cor.

Peso no Projeto: 2 %

3.5. *Controller*

3.5.1. *kbc.c*

Este módulo contém o código para lidar com a interface programática do KBC: ler o seu estado, ler o *output*, ler o *byte* emitido em resposta a um comando do rato, escrever um comando e escrever um argumento do comando. Os *drivers* estão preparados para lidar com falhas do KBC e até para varrimento, pese embora o programa só use interrupções.

Peso no Projeto: 6 %

Neste módulo, não existem estruturas de dados a descrever.

3.5.2. *keyboard.c*

Este módulo contém o código para lidar com o teclado: subscrever interrupções e anular a sua subscrição, lidar com interrupções, lendo o *scancode* e interpretando-o (tamanho, *bytes* e *makecode* vs. *breakcode*), apagar o último *scancode* lido e armazenado e ativar interrupções. Este módulo interage diretamente com o módulo do KBC.

Peso no Projeto: 6 %

Neste módulo, define-se a *struct scancode* que contém três campos: um booleano que indica se o *scancode* é um *makecode* (tecla premida) ou *breakcode* (tecla libertada), o tamanho do *scancode* (um ou dois *bytes*) e o(s) *byte*(s) do *scancode*. Como o teclado só emite um *byte* por interrupção, o tamanho do *scancode* na struct é zero até ele estar completo (momento em que o tamanho registado passa de zero para um ou para dois).

3.5.3. *mouse.c*

Este módulo contém o código para lidar com o rato: subscrever interrupções e anular a sua subscrição, lidar com interrupções, lendo o pacote e interpretando-o (*bytes*, botões, variação de coordenadas e *overflow* em cada eixo), apagar o último pacote lido e armazenado, escrever um comando para o rato e ativar ou desativar o reporte de informação pelo rato (note-se que esta função não é a fornecida pela LCF, mas foi desenvolvida por nós, para o projeto). Este módulo interage diretamente com o módulo do KBC.

Peso no Projeto: 6 %

Neste módulo, define-se a *struct packet* que contém oito campos: um array com os três *bytes* do pacote do rato, três booleanos (*rb*, *mb* e *lb*) – cada um correspondente a cada botão do rato, indicando se foi premido ou não –, duas variáveis inteiras (*delta_x* e *delta_y*) – correspondendo cada uma ao deslocamento do rato em cada eixo, desde o último pacote emitido – e outros dois booleanos (*x_ov* e *y_ov*) – que indicam se houve, ou não, *overflow* no eixo correspondente.

3.5.4. *timer.c*

Este módulo contém o código para lidar com o *timer*: definir a frequência de operação, subscrever interrupções e anular a sua subscrição, lidar com as interrupções (incrementar um contador global) e ler a configuração atual.

Peso no Projeto: 6 %

Neste módulo, não existem quaisquer estruturas de dados a descrever.

3.5.5. *utils.c*

Este módulo contém apenas três funções úteis para os outros módulos do *controller*: uma função para obter o byte menos significativo (LSB) de um inteiro de 16 bits, uma função para obter o byte mais significativo (MSB) de um inteiro de 16 bits e uma função que atua como *wrapper* de *sys_inb()*, para facilitar a leitura de informação de um porto para um valor de 8 bits (pois *sys_inb()* recebe como segundo parâmetro um apontador para uma variável de 32 bits, cujo valor depois é convertido em 8 bits).

Peso no Projeto: 4 %

Este módulo, enquanto ficheiro de *utils*, não define quais estruturas de dados para descrever.

3.6.6. *video.c*

Este módulo contém o código para lidar com a placa de vídeo e tudo o que com ela se relaciona: inicializar o modo gráfico pretendido (obtendo a informação do modo para inicializar variáveis globais estáticas, mapeando a VRAM para o espaço de endereçamento do processo e interagindo com a interface programática da VBE para definir o modo gráfico pretendido), trocar os *buffers* para a implementação de *double buffering* via *page flipping* (recorrendo à função VBE necessária – 0x07 – e subfunções 0x00 e 0x01 para, respetivamente, obter e definir o início do *display*), converter cores nas codificações pretendidas, desenhar um pixel, uma linha horizontal, um retângulo e um *pixmap* dada uma imagem XPM, bem como limpar uma área do ecrã e pintar todo o fundo de uma cor.

Peso no Projeto: 7 %

Este módulo apenas recorre à *struct vbe_mode_info_t*, fornecida pela LCF e que contém todas as informações sobre um determinado modo gráfico, preenchidas por *vbe_get_mode_info()*.

3.6. Diagrama de Chamadas a Funções

Na próxima página encontra-se o diagrama de chamadas a funções do projeto, gerado automaticamente pelo Doxygen. A única função que chama *driver_receive()* é *loop()* (*proj.c*), que é a função responsável por lidar com as interrupções subscritas, delegando essa gestão para o *model* que, por sua vez, a delega para o *handler* correspondente no estado do programa no momento.

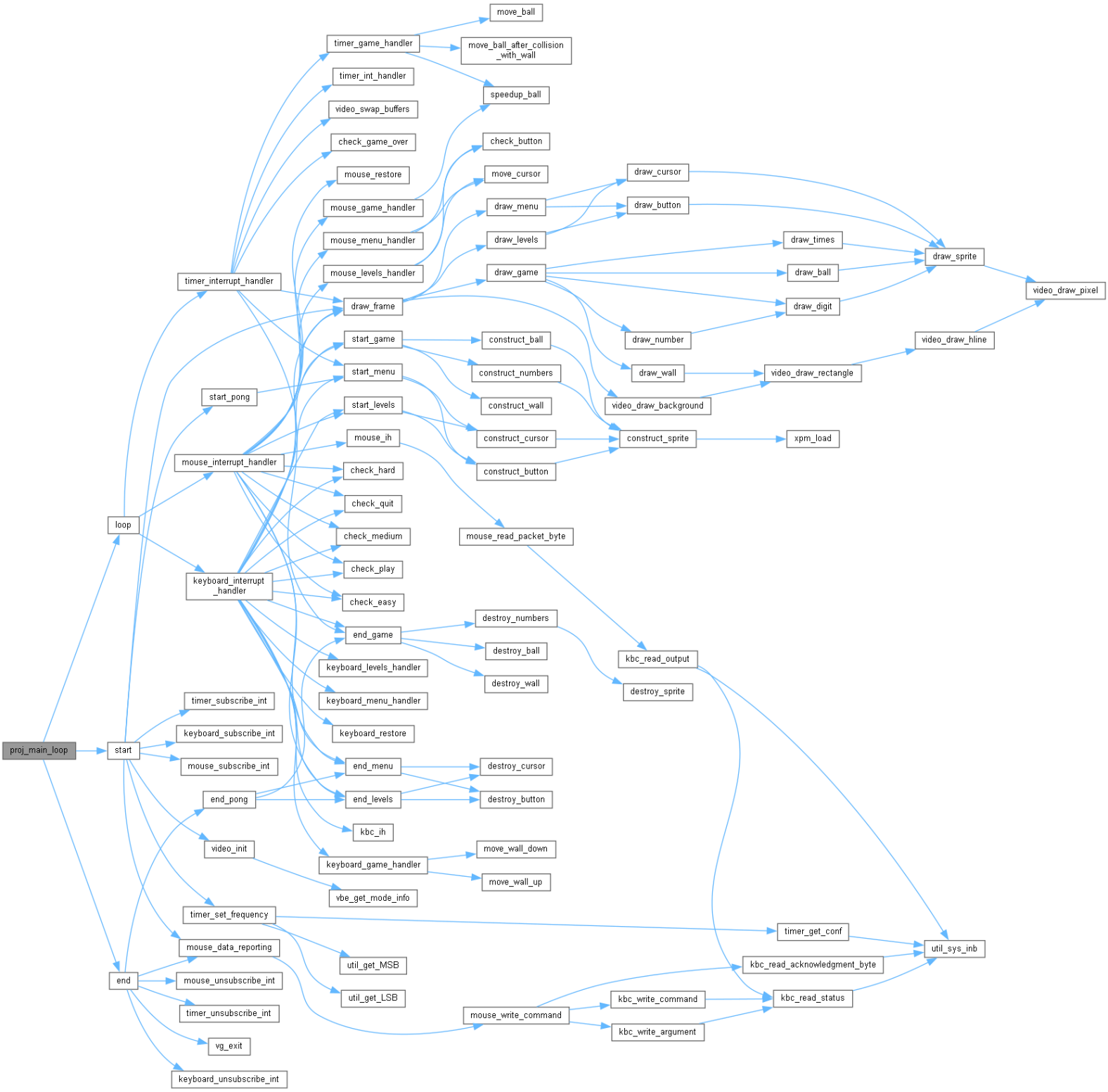

Figura 11 - Diagrama de Chamadas a Funções

4. Detalhes da Implementação

Nesta penúltima secção, devem constar os tópicos lecionados nas aulas teóricas, mas que, para a sua aplicação no projeto, foi necessária alguma ingenuidade, bem como os tópicos não abordados nas aulas teóricas que foram aplicados no projeto.

Em primeiro lugar, consideramos que a programação orientada a objetos em C foi essencial para facilitar o desenvolvimento do projeto e, em simultâneo, melhorar a qualidade do código e a arquitetura. Tendo em conta que, ao longo do curso, a linguagem com que mais nos fomos familiarizando foi C++, é natural que tenhamos uma maior predisposição para desenvolver código inserido no paradigma da orientação a objetos. Como tal, procuramos fazê-lo também neste trabalho, pese embora esse conteúdo não tenha sido lecionado nas aulas teóricas desta edição da Unidade Curricular. Assim, definimos cada classe (como a bola, a parede, o cursor e o botão) num ficheiro *.c* com funções cujo primeiro argumento é um apontador para o estado do objeto – armazenado numa *struct* – sobre o qual o método vai operar, o que nos permitiu melhorar a modularidade e organização do código.

Em segundo lugar, procuramos desenvolver um código estruturado em camadas – principalmente para o *controller* – por estarmos habituados a fazê-lo nos *Labs*. Deste modo, estruturamos os módulos do teclado e do rato para, em vez de interagirem diretamente com o KBC, fazerem-no através de um outro módulo (o módulo para o KBC), de maneira a uniformizar o código e melhorar a sua qualidade, reduzindo código que, eventualmente, seria repetido.

Em terceiro lugar, criamos uma máquina de estados para, num código dirigido a eventos, gerir e lidar com os diferentes estados possíveis do programa. Assim sendo, definimos uma *enum* com quatro valores/estados possíveis: MENU, LEVELS, GAME e END. Os módulos que lidam com esta *enum* são *model.c*, *view.c* e *proj.c*. *Model.c* fá-lo para, através de uma estrutura de controlo *switch-case*, permitir que o programa opere de forma diferente consoante o estado em que se encontra e, desse modo, direcionar o tratamento das interrupções de cada dispositivo para o módulo adequado. A par disto, *model.*c manipula também o estado atual e trata de transitar entre estados diferentes. De forma análoga, o estado é também relevante em *view.c* para saber qual o *frame* a desenhar em cada momento. Finalmente, em *proj.c*, o valor do estado só tem significado para saber quando é END, momento em que o programa deve terminar.

Por último, podemos detalhar o processo de geração de *frames*. Aquando da inicialização do programa, o *Timer* 0 é configurado para gerar 60 interrupções por segundo (em princípio isto não seria necessário, dado que esta deve ser a frequência normal de operação do *Timer* 0, mas esta abordagem permite-nos ter a certeza do *frame-rate* do projeto e, para além disso, tornar o código mais maleável para uma futura alteração do *frame-rate*). A cada interrupção do *Timer* 0, os buffers devem ser trocados e o frame desenhado no *back buffer* deve passar para o *buffer* principal, permitindo a visualização do seu conteúdo no ecrã. Ora, como os *frames* são desenhados aquando das interrupções do teclado e do rato (e, no modo de jogo, até do *timer*), esta abordagem permite-nos que o programa corra de forma fluída visualmente, prevenindo eventuais artefactos visuais.

Em suma, acreditamos que, no projeto, aprofundamos estes tópicos conforme necessário e adequado, procurando não só saber porquê aplicá-los, mas também aplicá-los da forma correta.

5. Conclusões

Finalmente, cumpre-nos tecer considerações sobre a globalidade do projeto, nomeadamente no que diz respeito a quatro aspetos principais: problemas, “like to haves”, principais conquistas e lições aprendidas.

Quanto ao primeiro ponto, consideramos que o maior entrave a uma melhor realização do projeto foi a falta de tempo, derivada, por um lado, da elevada carga de trabalho a que fomos sujeitos durante este semestre e, por outro lado, a alguma autogestão da nossa parte que deixou a desejar, isto é, que, se fosse feita agora, seria feita de outra forma, provavelmente melhor. Para além disto, entendemos que, sendo esta Unidade Curricular tão específica, talvez pudesse existir documentação mais acessível e não tão verbosa, que nos auxiliasse e guiasse melhor em alguns aspetos relativos à elaboração do projeto (por exemplo, na implementação de *double buffering* com *page flipping*). Ainda assim, consideramos que, com mais tempo e com uma melhor organização da nossa parte, o impacto deste segundo problema teria sido mitigado.

Em segundo lugar, existem, efetivamente, alguns aspetos e funcionalidades que, não tendo sido definidos nem planeados, poderiam acrescentar valor ao nosso projeto. A título de exemplo, podemos destacar não só um modo multijogador, não através da porta série, mas sim através do teclado e do rato em simultâneo, no qual um jogador controlava a sua “parede” móvel com o rato e o outro jogador controlava a respetiva “parede” com o teclado, como também maior variedade no modo para um jogador, por exemplo através do aparecimento de obstáculos aleatórios que alterassem a trajetória da bola.

Na nossa opinião, este projeto permitiu-nos alcançar alguns feitos dos quais nos orgulhamos. Particularmente, podemos destacar o facto de conseguirmos observar o resultado de toda a nossa aprendizagem ao longo do semestre em Laboratório de Computadores, isto é, desde o desenvolvimento de módulos de baixo-nível – os *drivers* dos dispositivos – até à elaboração do código de mais alto-nível. A par disto, realçamos também a aquisição de competências essenciais para um engenheiro informático, em específico a organização do código, a programação estruturada, a capacidade de fazer *debugging*, bem como de consultar (e elaborar) documentação. A funcionalidade que mais nos sentimos realizados em implementar no projeto foi o *double buffering* com *page flipping*, dado que não seguiu a abordagem tradicional para o *double buffering* – através de cópia de um *buffer* para a memória de vídeo –, obrigando-nos, assim, a algum trabalho de pesquisa que, no final, veio a dar resultados.

Ora, com a realização deste projeto aprendemos várias lições. Para além das competências já referidas, percebemos a importância de desenvolver bons *drivers* para dispositivos de entrada e saída, de saber como interagir com eles e o cuidado com que é necessário fazê-lo. Alargamos, também, o nosso leque de competências, nomeadamente a programação (estruturada) em C e o desenvolvimento de software de baixo nível, bem como o uso da interface de *hardware* dos periféricos do computador e de ferramentas de desenvolvimento de software.

Em suma, não sendo este um projeto perfeito – nenhum é –, acreditamos que, no final, realizamos um bom trabalho (que poderia, certamente, ter sido melhor, mas os erros ficam como aprendizagem para o futuro) e cumprimos com sucesso os objetivos da Unidade Curricular.