PONG 2.0

Projeto de LCOM 2022/2023

Turma 13 – Grupo 03

Carolina Viana (up202108802@fe.up.pt)

Guilherme Monteiro (up202108668@fe.up.pt)

Sérgio Peixoto (up202108681@fe.up.pt)

Sofia Sá (up202108676@fe.up.pt)

ÍNDICE

INTRODUÇÃO

Como projeto da Unidade Curricular de Laboratório de Computadores, criamos um Pong 2.0, com tema "Néon", inspirado no famoso e antigo jogo "Pong", mas com algumas features novas e originais.

O Pong 2.0 tem os dois modos de jogo: single-player e multi-player (Jogador contra computador e Jogador contra Jogador, respetivamente). No primeiro modo existe a possibilidade de utilizar dois powerups (movimento horizontal e aumento do tamanho do jogador), o que o distingue do jogo tradicional. Simultaneamente, os paddles, a bola e a arena do jogo tradicional traduzem-se, respetivamente, para jogadores, bola e campo de futebol néon, no nosso jogo.

A área de jogo é dividida a meio. Cada jogador situa-se numa das metades e existe uma bola movimentada pelos utilizadores, que se movimenta entre um e o outro lado do campo. O objetivo é projetar a bola para o lado oposto do campo e marcar pontos, impedindo justamente que a bola do adversário acerte na nossa parede. Para marcar ponto, a bola deve passar dos limites do campo do adversário e atingir a "sua" parede (parede vertical atrás de cada jogador).

Assim que um jogador atinge os 7 pontos, é então declarado vencedor e a partida termina. Os modos de funcionamento do jogo serão abordados na próxima secção.

MANUAL DO UTILIZADOR

➔ Menu inicial

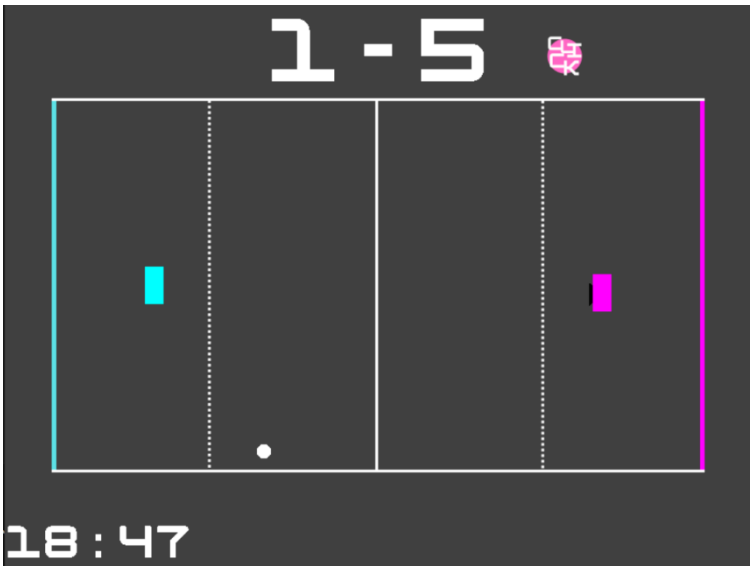
Após executar o código do projeto, aparecerá o menu inicial com as opções "1 Player", "2 Players" e "Instructions/Credits". As duas primeiras opções correspondem aos modos de jogo e a última opção mostra uma breve secção de instruções e os nomes dos criadores do jogo. Seleciona-se uma opção pressionando a tecla ENTER, podendo alternar entre elas usando as setas ('↑" e '↓"), do teclado, ou, simplesmente, utilizando o rato. A opção selecionada estará visualmente sombreada.

Menu inicial.

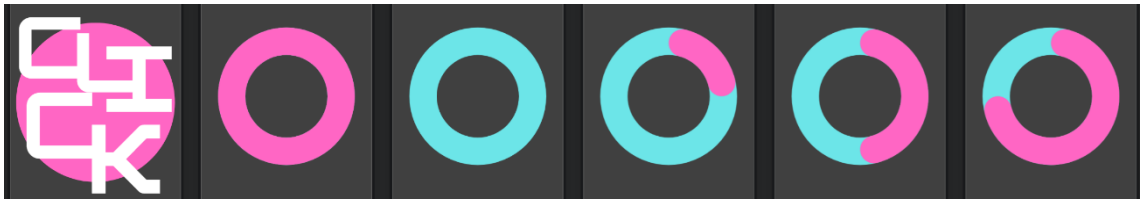
➔ 1 player e 2 players

No modo de jogo "1 Player", o jogador joga contra o computador. O objetivo é acertar com a bola na parede do lado adversário. Para este fim, o jogador deve ser movido com as teclas '↑' e '↓', para subir e para descer, respetivamente. Ao acertar na parede do lado oposto, o jogador pontua e o score é atualizado. O score é apresentado em cima do campo. Neste modo de jogo, ao clicar com o

botão esquerdo do rato no ecrã, é ativado um powerup: possibilidade do jogador se mover horizontalmente para defender a bola, com as teclas '←' e '→'. O outro powerup é ativado com o botão direito do rato, e tem como vantagem aumentar o tamanho do jogador. A duração de ambos os powerups é de 20 segundos, e têm um período de espera de 1 minuto. O movimento do jogador controlado pelo computador é calculado a partir da posição da bola, após esta ter passado para a metade direita do campo. Aqui, verifica-se se a bola está acima ou abaixo do centro do jogador e este é movido para cima ou para baixo, respetivamente.



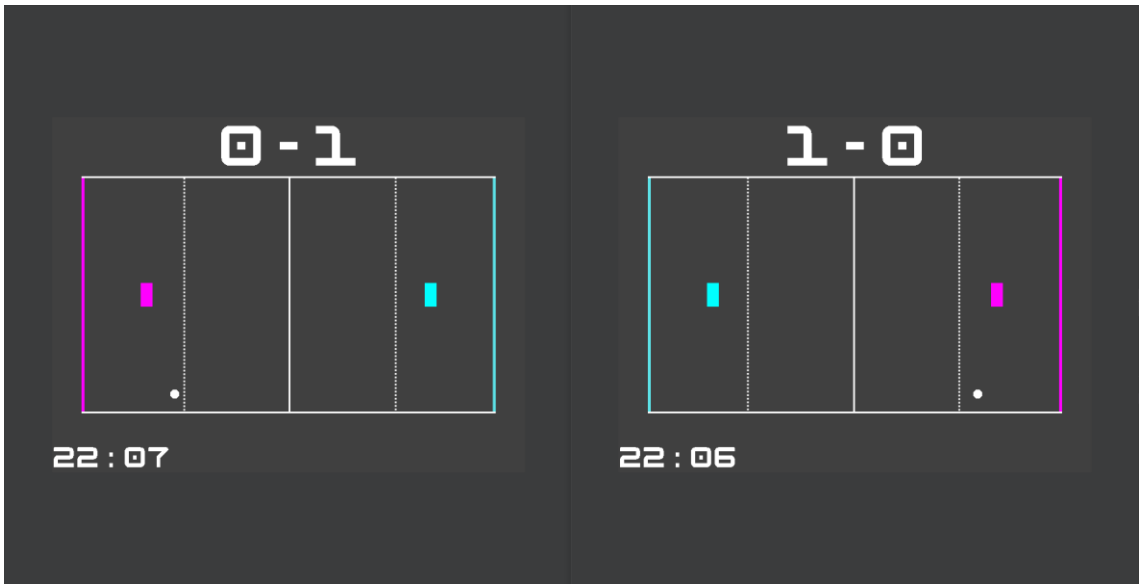
Modo de jogo Single-player.



Sequência das sprites após a ativação dos powerups.

No modo de jogo "Multi-player", existe a possibilidade de dois jogadores reais participarem na partida, sendo a lógica do jogo a mesma. Um dos utilizadores é o jogador cor-de-rosa e o outro o azul. Para os movimentar

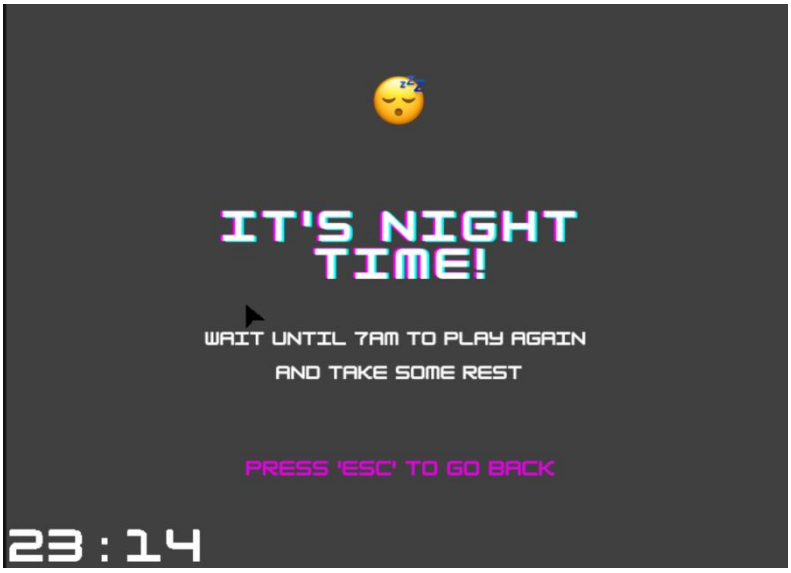
utilizamos, na mesma, as teclas ‘↑’ e ‘↓’, para subir e para descer, respetivamente. O objetivo do jogo também se mantém - proteger a nossa "parede” e marcar golo na do adversário. Neste modo, no entanto, não temos acesso a powerups, sendo que o jogo se aproxima do “clássico” Pong.



Os dois ecrãs do multi-player lado a lado.

Ecrã do utilizador que deve esperar pelo adversário.

Ambos os modos ficam indisponíveis desde as 23h às 7h do dia seguinte. Esta feature foi implementada como um incentivo a não permitir o utilizador jogar durante as suas horas de sono.



Ecrã de bloqueio horário.

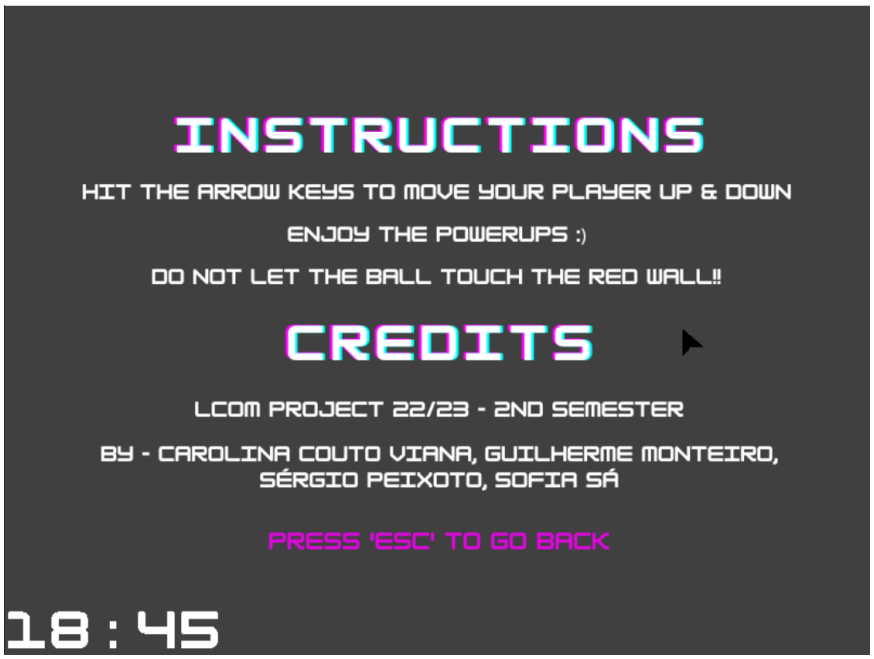
➔ **Fim de jogo**

Ao perder ou ao ganhar o jogo, é apresentado um menu de derrota ou vitória, respetivamente, e a opção de voltar atrás.

Menu de derrota e de vitória, respetivamente.

➔ **Instruções e Créditos**

Neste menu, aparece um conjunto de instruções básicas do jogo. De seguida na secção dos créditos, são apresentados os nomes dos criadores do jogo e, por fim, a opção de voltar atrás.



Menu de créditos e instruções.

ESTADO DO PROJETO

Dispositivo	Para que foi usado	Interrupções
Timer	Controlar o frame rate do jogo Atualizar o ecrã do jogo "Cooldown" dos powerups	Sim
Keyboard	Navegar pelo menu e para mover os jogadores em ambos os modos de jogo. Voltar para o menu principal	Sim
Mouse	Navegar pelo menu e para ativar os powerups no modo de jogo Single Player	Sim
Video card	Display dos menus e de todas as frames de jogo	Não
Real time clock	Display da hora atual em todos os menus e modos de jogo Feature de controlo de horário	Sim
Serial port	Possibilitar o modo de jogo Multi-player com os 2 jogadores controlados por utilizadores reais	Sim

➔ **Timer**

O timer é responsável por atualizar o nosso jogo. Definimos uma frame-rate de 30fps, isto é, o jogo atualiza 30 vezes por segundo.

As interrupts no timer é o que permite atualizar o ecrã - a cada duas interrupções o jogo verifica se houve colisões e atualiza a posição da bola. Para além disso, a cada duas interrupções também verificamos e, caso necessário, atualizamos a pontuação e, aquando o 7º golo de um jogador, termina a partida.

No modo de jogo Single-Player, é o timer que define quando disponibilizar e bloquear os powerups e o estado do contador de tempo de espera dos mesmos, à direita da pontuação. As 6 fases deste são escolhidas de acordo com o tempo restante de "cooldown" (80s, 60s, 45s, 30s, 15s, 0s)

As funções são:

- int(timer_subscribe_int)(uint8_t *bit_no);
- int(timer_unsubscribe_int)();
- void(timer_int_handler)();

E as suas implementações encontram-se no ficheiro timer.c

➔ Keyboard

No menu inicial do projeto, o keyboard é utilizado para navegar entre as 3 opções disponibilizada.

Nos dois modos de jogo, utilizamos as setas para mover o jogador para impedir que a bola toque na "nossa" parede. Em qualquer momento, movemos verticalmente com as teclas "UP" e "DOWN" e, após ativar o powerup que permite um movimento horizontal do jogador, com as teclas "LEFT" e "RIGHT".

Em qualquer momento, ao pressionar a tecla "ESC" voltamos ao menu inicial. No menu inicial, pressionando novamente "ESC", fechamos o jogo.

As funções são:

- int (kbd_subscribe_int)(uint8_t *bit);
- void (kbc_ih)();
- int (kbd_unsubscribe_int)();
- int (KBC_read_output_keyboard)(uint8_t* output);
- int (KBC_write_cmd)(uint8_t port, uint8_t cmdByte);

E as suas implementações encontram-se no ficheiro kbd.c

➔ Mouse

No projeto, utilizamos o mouse para diferentes objetivos.

Como já mencionado, no Menu, o rato é (à semelhança do teclado) utilizado para navegar e selecionar uma das opções disponíveis.

No modo Single Player, o mouse é utilizado para ativar ambos os powerups. Clicando em qualquer lado do ecrã, o botão esquerdo ativa o powerup do

movimento horizontal e o botão direito aumenta o tamanho do jogador. Para isto, aquando uma interrupção do rato, o controlador verifica que botão foi pressionado (direito ou esquerdo) e se o powerup está disponível (na função singlePlayerController()). Se estas duas condições se verificarem, então é ativado o powerup escolhido.

As funções são:

- void (mouse_ih)();
- int (mouse_subscribe_int)(uint8_t *bit);
- int (mouse_unsubscribe_int)();
- void (mouse_bytes_to_packet)(struct packet *mouse_packet);
- int (mouse_write)(uint8_t cmd);
- int (KBC_read_output_mouse)(uint8_t port, uint8_t* output);
- int (KBC_write_cmd)(uint8_t port, uint8_t cmdByte);

E as suas implementações encontram-se no ficheiro mouse.c

➔ Video card

A placa gráfica é fundamental no jogo para expor visualmente todos os gráficos inerentes ao jogo, servindo de interface e permitidno o contacto utilizador/jogo.

No desenvolvimento deste projeto, utilizamos o modo de vídeo 0x115 com uma resolução de 800x600 e modo direto de cores, cada cor sendo representada com 24 bits, 8 para cada canal R, G e B.

Conforme o planeado na nossa *proposal*, utilizamos a técnica de double buffering. Para isto, desenvolvemos duas funções - swap_buffer() e clear_buffer() - que trocam o buffer "principal" com o "secundário" e apagam o conteúdo do buffer. Com isto, conseguimos garantir uma experiência de renderização muito mais "smooth", evitando "flickering" e "tearing", visto que garantimos que apenas mostramos imagens totalmente renderizadas.

Assim, a cada dois interrupts do timer, os conteúdos do jogo são desenhados num buffer auxiliar. De seguida, esses conteúdos são copiados para o buffer principal.

Todos os elementos a serem expostos no jogo, à exceção dos jogadores, são desenhados através da função draw_sprite(). Para isto, elaboramos ficheiros .xpm correspondentes a todas as sprites que queríamos utilizar (Menu, Arena, bola, pontuação, etc). Essas sprites são desenhadas com a função supramencionada, que, por sua vez, chama a função draw_pixel() e que tem em conta as transparências através do "modo de cor" TRANSPARENCY_COLOR_8_8_8_8. Ainda, a função permite espelharmos a sprite (argumento booleano flipped) - utilizado no modo Multi Player, para espelhar a arena numa das VMs.

Os jogadores, sendo apenas retângulos, foram desenhados utilizando a função draw_rectangle().

As funções são:

- int(set_graphics_mode)(uint16_t mode);
- int(build_frame_buffer)(uint16_t mode);
- void (swap_buffer)();
- void (clear_buffer)();
- int (draw_pixel)(uint16_t x, uint16_t y, uint32_t color);
- int (draw_rect)(uint16_t x, uint16_t y, uint16_t width, uint16_t height, uint32_t color);
- int(draw_xpm)(xpm_image_t xpm, uint16_t x, uint16_t y, int flipped);

E as suas implementações encontram-se no ficheiro gpu.c

➔ Real Time Clock

O RTC desempenha dois papéis no Pong 2.0.

Primeiramente, é responsável por exibir a hora atual no canto inferior esquerdo de todos os menus e modos de jogo.

Além disso, desenvolvemos uma feature de controlo de horário. O seu objetivo é impedir que o utilizador jogue entre as 23h e as 7h, garantindo o seu bem-estar e descanso. Para isso, o RTC atualiza a estrutura "real_time_info" uma vez por minuto, com a função rtc_update_real_time(). Em seguida, na função rtc_check_permission() bloqueia/desbloqueia o jogo conforme a hora.

As funções são:

- int(rtc_subscribe_int)(uint8_t *bit_no);
- int(rtc_unsubscribe_int)();
- uint8_t(rtc_convert_to_binary)(uint8_t n);
- int(rtc_update_real_time)();
- void(rtc_ih)();
- int(rtc_check_permission)();

E as suas implementações encontram-se no ficheiro rtc.c

➔ Serial Port

A Serial Port é utilizada para o modo de jogo Multi-player, possibilitando uma partida com jogadores controlados por dois utilizadores reais. Desta forma, é transmitida informação de um jogo para o outro, nomeadamente a posição dos jogadores.

Inicialmente, de modo a permitir a ligação das duas máquinas, ao selecionar a opção de Multiplayer, a Serial Port é configurada de maneira a gerar interrupções quando há dados para ler (ativando o bit 0 do Interrupt Enable Register) e é enviada uma mensagem (SERIALP_STARTGAME) para avisar a outra máquina que estamos prontos a iniciar o modo de jogo Multiplayer - função serialp_init(). De seguida, se a outra máquina receber essa mesma mensagem, sinaliza a primeira que, de facto, podemos começar o jogo, ao enviar uma outra mensagem (SERIALP_PLAY) – função send_data_serialp(). Assim, a primeira máquina, ao receber esta nova mensagem, parte para o modo Multiplayer, visto que já sabe que as duas estão conectadas.

Já ao longo do jogo, devido à nossa implementação não ser baseada em FIFOs, estamos limitados na quantidade de informações que podemos enviar a outra máquina. No nosso caso, apenas conseguimos enviar 1 byte de cada vez. Desta forma, tiveram de ser feitos alguns compromissos: apenas enviamos informação da posição Y do jogador. No entanto, dado que a posição Y do jogador pode assumir valores entre 100 e 500 (altura da arena), estamos impossibilitados de mandar todas as posições possíveis, dado que o limite para 8 bits é 256 valores. De modo a contornar este problema, decidimos dividir a posição atual do jogador pelo valor da sua velocidade em Y (20 pixéis por movimento) de modo que o novo intervalo seja de 5 a 25. Assim, ao receber os dados na outra máquina, simplesmente multiplicamos novamente pela velocidade, obtendo assim a posição correta do jogador.

Necessitamos também de enviar informações mais especificas, como quem marcou golo, quem ganhou ou quando sair do jogo. Para isto é necessário garantir que as mensagens enviadas não sejam possivelmente confundidas com posições do jogador. Repare-se que, como o intervalo de valores para a posição (já comprimidos) é de 5 a 25, deixa-nos disponíveis vários valores para utilizarmos para estas informações, assim, definimos que os bits 7 e 6 correspondem a se marcamos ou sofremos golos, respetivamente, os bits 0 e 1 correspondem a se perdemos ou ganhamos o jogo, respetivamente. Por último, mandamos o break code da tecla ESC, de modo a possibilitar acabar o jogo nas duas máquinas.

Para enviar dados para a Serial Port utilizamos a função send_data_serialp() e, a cada interrupt gerado pela Serial Port (quando há dados para ler, como explicado), chamamos a função que lê esses dados e os escreve na variável “data” (read_data_serialp()).

As funções são:

- int (init_serialp)();
- void(serialp_ih)();
- int (read_data_serialp)();
- int (read_status_serialp)(uint8_t* status);

- int (send_data_serialp)(uint8_t data);
- int (serialp_connection)(int *ready);
- int (serialp_subscribe_int)(uint8_t *bit_no);
- int (serialp_unsubscribe_int)();
- int (serialp_get_status)(uint8_t* status);
- int (serialp_check_error)(uint8_t *status);
- int (write_address_with_offset)(int offset, uint8_t data);

E as suas implementações encontram-se no ficheiro serialport.c

ORGANIZAÇÃO DO CÓDIGO

➔ **Módulo do timer (timer.c)**

Peso no projeto: 9%

Descrição**:** Este módulo possui uma estrutura semelhante ao do lab2, com funções para subscribe e unsubscribe de interrupções e dar handle às mesmas. Neste módulo, apenas incluímos as funções do lab2 que utilizamos no projeto.

Realizado por**:** Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **Modulo do video card (gpu.c)**

Peso no projeto: 6%

Descrição: Este módulo também reutiliza as funções desenvolvidas no lab. Incluímos funções para iniciar o modo gráfico e mapear a memória virtual, bem como as funções necessárias para o *double buffering* e desenho de pixéis, retângulos e *sprites*.

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **Módulo do keyboard (kbd.c)**

Peso no projeto: 9%

Descrição: Em relação a este módulo, a sua estrutura e conteúdo não diferem significativamente das funções desenvolvidas para o lab3. Desenvolvemos as funções típicas de subscribe e unsubscribe de interrupções e interrupt handlers, bem como escrita e leitura de diferentes endereços.

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **Módulo do mouse (mouse.c)**

Peso no projeto: 5%

Descrição: Em relação a este módulo, a sua estrutura e conteúdo não diferem significativamente das funções desenvolvidas para o lab3. Para o mouse, desenvolvemos as funções típicas de subscribe e unsubscribe de interrupções e interrupt handlers, escrita e leitura de diferentes endereços, lemos e interpretamos packets de dados para utilizá-los noutros módulos do projeto.

Realizado por: Guilherme Monteiro

➔ **Módulo do real time clock (rtc.c)**

Peso no projeto: 5%

Descrição: Neste módulo estão as funções de subscrição de interrupções do RTC e da leitura e atualização dos vários parâmetros relacionados com a hora corrente.

Estrutura de dados relevante: struct real_time_info – permite guardar os dados referente à hora real lida pelo RTC (no nosso caso, minutos e horas)

Realizado por: Sofia Sá

➔ **Módulo da serial port (serialport.c)**

Peso no projeto: 10%

Descrição: No módulo da Serial Port, damos subscribe e unsubscribe a interrupções, escrevemos e lemos dados para/da Serial Port, para além de consultar o estado e todas as verificações de erros.

Estrutura de dados relevante: uint8_t data – este é o byte para qual é escrito e lido toda a informação que passa na Serial Port. Optamos por não utilizar FIFO, pois esta seria desnecessária - apenas comunicamos com mensagens de 8 bits.

Realizado por: Sérgio Peixoto e Carolina Viana.

➔ **MVC: controller.c**

Peso no projeto: 22%

Descrição: Este módulo contém a definição de várias funções relacionadas com o controlo do jogo, bem como a definição de enumerações para os dispositivos utilizados e os estados do jogo. Adicionalmente, define uma struct "state" que contém um estado do jogo e um apontador para o controlador respetivo.

A enumeração "devices" enumera os dispositivos utilizados no jogo, incluindo o timer, o keyboard, o mouse, o RTC (real time clock) e a serial port.

A enumeração "game_states" enumera os estados do jogo:

- Menu Principal
- Jogo para 1 e para 2 jogadores
- Vitória e Derrota
- Créditos
- Estado de espera da conexão e estado bloqueado
- Saída

O array de structs "game_states_list" contem uma listagem das estruturas que associam todos os estados do jogo às respetivas funções de controlo.

As outras funções são os controladores específicos para cada estado do jogo. Estas lidam com eventos específicos de cada estado, como inputs do keyboard, inputs e movimentos do mouse e updates do timer. Realizam ações relacionadas a cada evento e atualizam o estado do jogo conforme necessário.

No caso dos controladores dos modos 2 jogadores e espera de conexão também é tida em conta a serial port e os dados transferidos por esta.

Estrutura de dados relevante: enum devices; enum game_states; struct state; array de structs do tipo state game_state_list

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **MVC: model.c**

Peso no projeto: 19%

Descrição: Neste módulo estão definidas as estruturas para o cursor do mouse, os jogadores, a bola, e o campo. Estão também definidas duas enumerações: "CollisionType" e "Winner", que representam os vários tipos de colisões e quem ganhou o jogo, respetivamente.

Para além disto, temos também implementadas as funções responsáveis pela navegação no menu principal, colisões e movimento da bola e dos jogadores, gestão dos powerups, verificação de golos e vitória, reset às posições e ao jogo e movimento do rato. Para uso exclusivo no modo de 2 jogadores, temos a implementação das funções que espelham a bola e o jogador e que comprimem os dados para envio pela serial port.

Estrutura de dados relevante: struct Cursor; struct Player; struct Ball; struct Arena; enum CollisionType; enum Winner

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **MVC: viewer.c**

Peso no projeto: 8%

Descrição: Este módulo define e implementa as várias funções relacionadas com o render dos gráficos no ecrã. São responsáveis pelo desenho dos elementos visuais, como menus, imagens de fundo, cursor do rato, campo, jogadores, bola e relógio.

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ **MVC: xpms.c**

Peso no projeto: 2%

Descrição: Este módulo contém o array de imagens que serão incluídas no projeto e funções que lhes permite dar load.

Estrutura de dados relevante: enum images, array images (xpm_image_t)

Realizado por: Carolina Couto Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

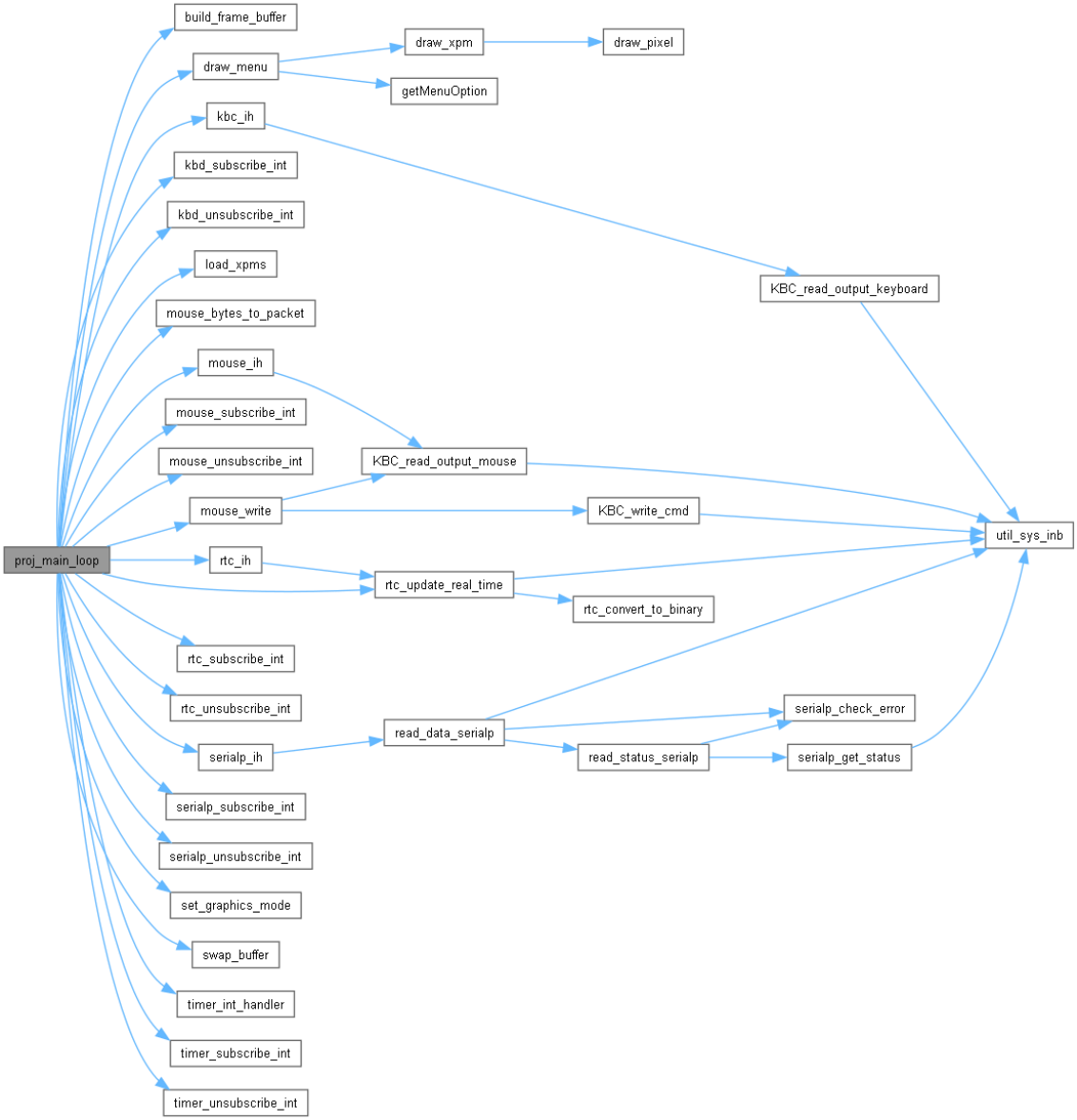
➔ **main.c**

Peso no projeto: 5%

Descrição: Este módulo subscreve todas as interrupções e contém o driver_receive loop para lhes dar handle. Por fim, cancela todas as subscrições. É, portanto, o motor do jogo, interagindo com todos os módulos.

Realizado por: Carolina Viana, Guilherme Monteiro, Sofia Sá, Sérgio Peixoto

➔ Function call graph



DETALHES DA IMPLEMENTAÇÃO

No nosso jogo, decidimos adotar o princípio **Model-View-Controller** (MVC). O Model é responsável pela lógica e armazenamento de todos os dados do jogo. A View renderiza e apresenta as interfaces ao utilizador, permitindo a interação com o jogo. O Controller controla todo o fluxo do jogo, lidando com as diferentes interrupções dos dispositivos de entrada e saída, e invocando as funções que manipulam os dados conforme necessário. É, também, a ponte entre o model e a view.

Implementar o **RTC (Real-Time Clock)** foi surpreendentemente simples, optando por fazê-lo com métodos de interrupções, sendo o grande objetivo apenas ler a informação disponível nos seus registos relativamente à hora atual. A hora e os minutos são apresentados em todos os menus e modos de jogo. Como *feature* extra, decidimos "bloquear" o jogo das 23h às 7h da manhã do dia seguinte. Esta iniciativa pretende incentivar o utilizador a descansar, impedindo-o de jogar o Pong 2.0 durante as suas horas de sono.

A **Serial Port**, por outro lado, foi mais complexa de implementar, tal como era expectável. A sincronização entre as duas máquinas foi algo mais difícil: passar as mensagens, de modo que toda a informação necessária fosse possível de enviar em 8 bits, sem que houvesse confusão entre os diferentes tipos de mensagens foi algo que demorou algum tempo. Garantir também que ambos os jogos continuassem sincronizados foi um desafio com a quantidade limitada de dados que tínhamos.

Para controlar os diferentes estados do jogo, implementámos uma **Máquina de Estados,** que representa os diferentes estados. O utilizador, ao selecionar diferentes opções, vai alternando entre os vários estados da máquina. Em cada estado, as funções chamadas e features do jogo vão sendo diferentes e, para obtermos isso, fizemos com que cada estado lidasse com as interrupções dos periféricos de formas diferentes.

Quando utilizamos **duas** máquinas (Serial Port - modo Multi Player), a máquina de estados é replicada nas duas Virtual Machines. Isto é, a mesma lógica do jogo vai correndo nas duas máquinas separadamente. O jogo decorre "normalmente" e as VMs vão enviando a posição do seu player uma à outra, conforme explicitado anteriormente. A cada interrupção da Serial Port, a VM que recebe a informação atribui a posição recebida ao seu "player2" e repetindo para que o jogo decorra sem problemas.

A implementação de **function pointers** nos estados do jogo, permitiu tornar o nosso código muito mais modular, pois facilita a criação e aplicação de novos modos de jogo e respetivos controladores. Também nos permitiu proteger o código que lida com as mudanças dos estados de jogo. Por último, melhorou a performance do nosso programa de uma forma bastante significativa, algo que nos surpreendeu, pois após a nossa pesquisa verificamos não é regra geral em todos os casos que esta técnica é aplicada. Assim, contribuiu para a fluidez enquanto jogávamos.

Para detetar as **colisões** dos jogadores com a bola, utilizamos a função "checkBallCollision()", que recebe os jogadores por referência. Verifica se a próxima posição da bola coincide com uma posição ocupada pelo jogador e retorna o tipo de colisão (cima/baixo, lateral ou sem colisão), dependendo se esta acontece e em que lado do jogador tal se verifica. Esta distinção dos lados é necessária para indicar à bola em que direção se deve mover após colidir, pois uma colisão lateral implica inverter a velocidade no eixo horizontal, mas uma no topo/parte de baixo do jogador altera o sentido no eixo vertical. As colisões da bola com os limites do campo são tratadas pela função "moveBall", que lida com as mudanças de trajetória.

Para navegar entre as várias opções do menu e selecioná-las, criámos as funções: "moveCursor()", para o movimento do rato sob as várias opções do menu, "upMenuOption()" e "downMenuOption()" para mudar a opção selecionada de acordo com o input da tecla pressionada, "setMenuOption()"

para mudar a opção e “getMenuOption()” para obter a opção selecionada do menu.

Devido à maneira como optamos por implementar a Serial Port (lógica do jogo a correr nas duas VMs, apenas transmitem a posição do jogador), apercebemo-nos que o jogo a correr numa das VMs está ligeiramente atrasado perante o outro. Isto acontece porque a informação enviada (posição do jogador) só é recebida e, consequentemente, tratada, no loop seguinte do jogo, o que, naturalmente, causa este delay. Tentamos várias abordagens para resolver e/ou colmatar este “bug”, no entanto, não fomos bem-sucedidos. De momento, como este delay é mínimo, não impede o bom funcionamento do jogo e pode ser ainda mais reduzido utilizando um computador com um processador poderoso, optamos por “assumir” o problema e continuar a disponibilizar a opção de Multi Player, visto que estamos orgulhosos da nossa implementação da Serial Port.

Outro “bug” que não conseguimos resolver, foi um problema de colisões entre o jogador e a bola. De facto, se a bola bater exatamente num dos cantos do jogador, há dificuldade em distinguir qual foi o tipo de colisão e, por conseguinte, qual direção deverá a bola tomar. Assim, a bola acaba por ficar “presa” dentro do jogador, pois está constantemente a sofrer colisões. Tentamos resolver, mas apesar de não termos conseguido não é nada que penalize muito a experiência do utilizador, uma vez que basta mover o jogador para a bola prosseguir o seu rumo no campo.

CONCLUSÕES

Fazer este projeto foi bastante enriquecedor: aprendemos muito com os nossos sucessos, mas também com os nossos insucessos. Todo o tempo e trabalho investido semanalmente na implementação dos Labs foi importantíssimo para uma boa compreensão dos tópicos lecionados na Unidade Curricular de Laboratório de Computadores e para o desenvolvimento do projeto de forma produtiva e eficaz. Alguns aspetos, não falados durante as aulas, foram mais complicados de implementar, nomeadamente a Serial Port.

De facto, como explicado anteriormente, a nossa maior dificuldade ao implementar a Serial Port, deparamo-nos com uma descoordenação entre os dois ecrãs. Como já foi explicado algumas vezes, não conseguindo implementar a FIFO, apenas estabelecemos uma ligação que envia dados de, no máximo, 8 bits. Desta maneira, não conseguimos enviar toda a informação que desejávamos (nomeadamente a posição da bola) pelo que sempre houve um delay entre as duas virtual machines, notório principalmente no movimento da bola. Para combater o problema, tentámos implementar a FIFO, no entando não tivemos tempo. Tentámos forçar um delay "hard-coded", mas também foi mal-sucedido. Seria um bug que gostávamos de ter corrigido se tivéssemos mais tempo e que provavelmente tornaria o jogo mais estável e mais "future proof" a possíveis adições de features futuras.

Por fim, nos últimos dias, fizemos um refactor ao nosso código para o tornar mais eficiente. Aceitamos o desafio de utilizar function pointers nos controladores principais, o que nos tirou algum tempo mas que melhorou a fluidez do programa.

Ficámos bastante orgulhosos com o resultado final, tendo em conta que conseguimos implementar todos os dispositivos e ter algumas ideias originais que destacaram o nosso jogo dos restantes. Após bastantes bugs e aprendizagens, o nosso jogo acabou por ficar desafiante e cativante para o

user-side. Foi recompensador ver o nosso jogo correr ao final de um semestre trabalhoso e exigente.