Faculdade de Engenharia da Universidade do Porto

ONE SHOT

Projeto Final - LCOM - Turma 4 - Grupo 05

Realizado por:

Paulo Jorge Salgado Marinho Ribeiro, up201806505

Pedro Jorge Fonseca Seixas, up201806227

Índice

Introdução

Para o nosso projeto final, decidimos criar um jogo em 2D, numa perspetiva Top-Down, ou seja, visto de cima para baixo. O jogo tem 2 modos, Singleplayer e Multiplayer.

O principal objetivo do primeiro modo é passar todos os níveis. Dentro de cada nível, o jogador deve conseguir chegar à zona final e, para isso, deverá matar ou escapar-se de todos os inimigos e abrir todas as portas usando as alavancas espalhadas pelos mapas.

No modo Multiplayer, existe uma arena, em que os jogadores se têm de tentar matar um ao outro três vezes. O primeiro jogador a ficar sem as suas 3 vidas, perde.

A essência do título (One Shot) vem do facto de o jogador apenas poder ter uma bala de cada vez, ou seja, apenas poderá disparar uma vez, e, após isso, terá que encontrar uma nova bala.

1. Instruções de Utilização

1.1 Menu Inicial

Ao iniciar o jogo, é mostrado inicialmente o menu inicial, onde é possível escolher o modo de jogo desejado (Single-Player ou Multi-Player), ir para a secção Help (para se informar sobre os controlos) ou sair do jogo (clicando em Exit). Para navegar neste menu deve ser utilizado os movimentos do rato para mover o cursor, e o clique no botão esquerdo para selecionar a opção desejada.

1.2 Single Player

1.2.1 Map

Quando se inicia o jogo no modo Single-Player, é carregado o primeiro nível que serve como um nível de introdução ao movimento do jogador e à abertura de portas através das alavancas.

O jogador move-se com o teclado usando as teclas WASD ou com as setas. Para abrir as portas, é necessário o jogador colocar-se junto a uma alavanca, como mostra a seguinte imagem:

Após se encontrar nesta posição, o jogador deve carregar na tecla E para entrar no modo Lever, que será explicado na próxima secção.

Depois de a porta ser desbloqueada, o jogador deve deslocar-se até à zona destacada para seguir para o nível seguinte.

O segundo nível serve também, tal como o primeiro nível, como introdução. Desta vez, introduzem-se as balas e os primeiros inimigos. Para apanhar uma bala, o jogador deve posicionar-se junto a ela.

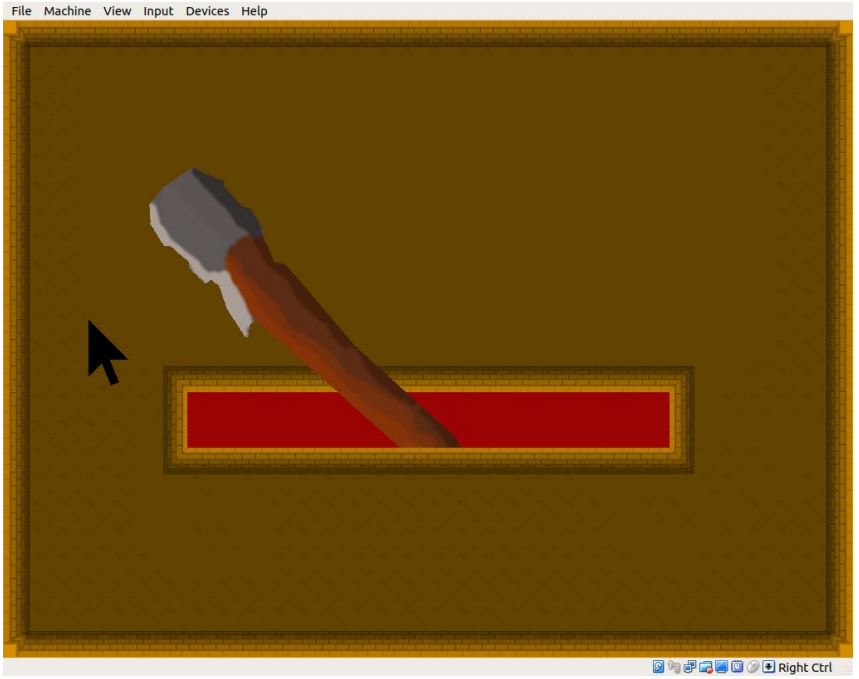
Automaticamente o jogador apanha a bala caso não tenha nenhuma. Caso contrário, se o jogador já tiver uma bala, não irá acontecer nada ao posicionar-se junto a esta. Apenas é permitido ter uma bala de cada vez. Para disparar, o jogador deve premir a Barra de Espaço.

Existem dois tipos de inimigos, os Guardas, que estão em constante movimento, e os Atiradores, que são estáticos e disparam balas em intervalos de tempo aleatórios.

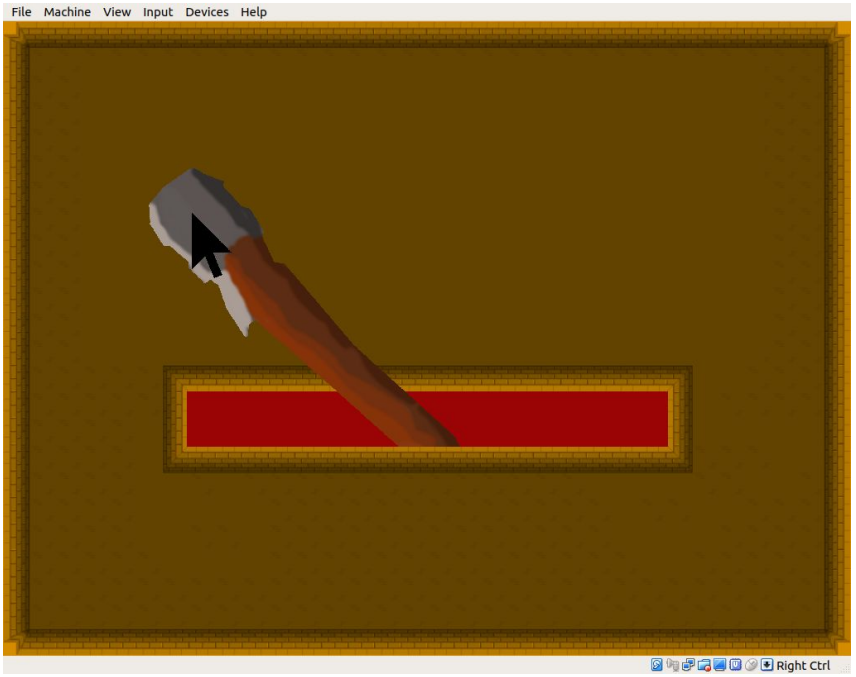
1.2.2 Lever

O objetivo deste modo é apenas utilizar o rato para abrir a alavanca, e, deste modo, abrir a porta.

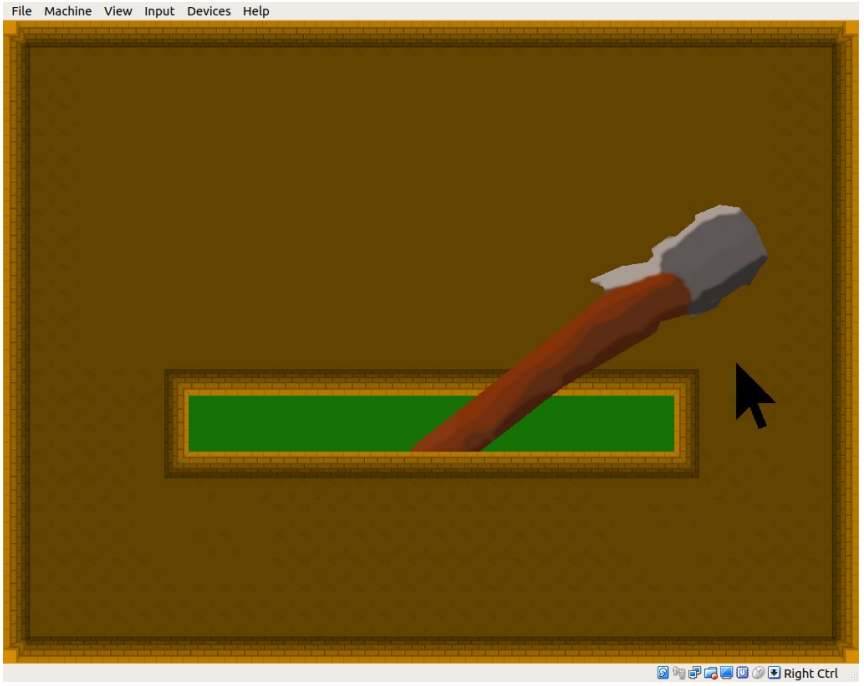
Após carregar na tecla E estando posicionado em cima de uma alavanca, é carregado um close-up da alavanca:



Para desbloquear a alavanca, o utilizador deve posicionar o cursor na pega da alavanca:



Depois de o cursor estar bem posicionado, para iniciar o movimento deve ser premido o botão esquerdo do rato, e deve ser arrastada a alavanca para a direita, até dar luz verde.



Depois de receber luz verde, o utilizador deve largar o botão esquerdo, e poderá regressar ao mapa carregando novamente na tecla E. A porta estará então desbloqueada:

1.2.3 Pause Menu

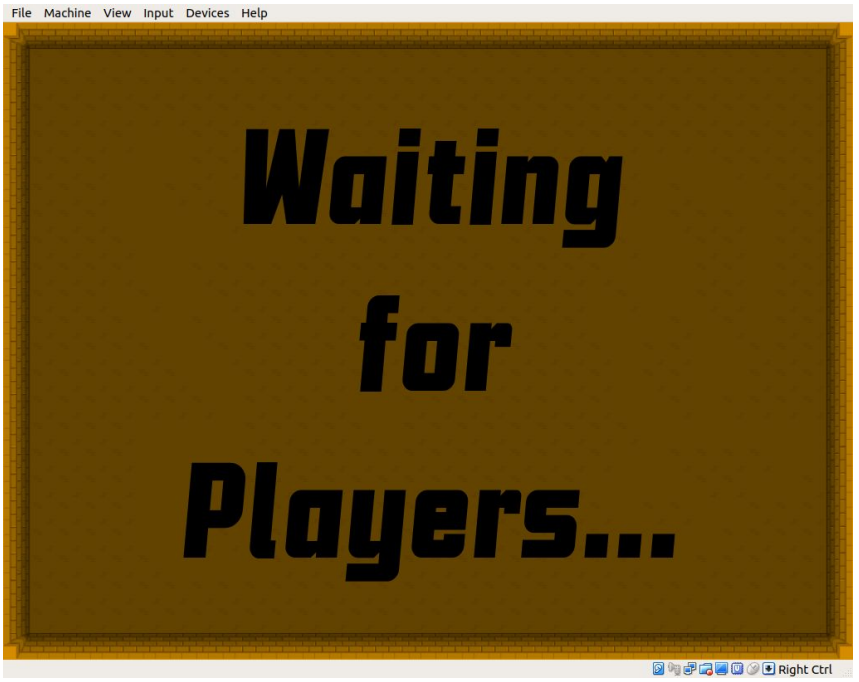
Em qualquer momento do jogo, o utilizador poderá aceder ao Pause Menu carregando na tecla ESC.

Após estar no menu de pausa, o jogador poderá carregar em Continue ou premir a tecla ESC, para voltar ao jogo, ou carregar em Exit, para retornar para o Main Menu.

1.3 Multi Player

1.3.1 Waiting

Quando é carregado no botão Multi-Player, o utilizador entra numa sala de espera, onde se espera que haja outro utilizador que se junte para, nesse momento, ser iniciado o jogo.



O utilizador pode a qualquer momento retornar ao menu inicial carregando na tecla ESC.

1.3.2 Game

Quando estão os dois jogadores conectados, o jogo é carregado, sendo o jogador 1 (o primeiro a aceder o modo Multi-Player) o jogador azul, e o jogador 2 (segundo jogador a aceder o modo) o jogador vermelho.

O objetivo é apanhar balas e tentar matar o adversário 3 vezes. Quando o jogador mata o adversário 3 vezes, ganha o jogo indo para o menu vencedor (secção 1.5). O jogador que perder vai para o menu perdedor (secção 1.5). Se a qualquer momento o utilizador quiser voltar ao menu inicial apenas precisa de premir a tecla ESC.

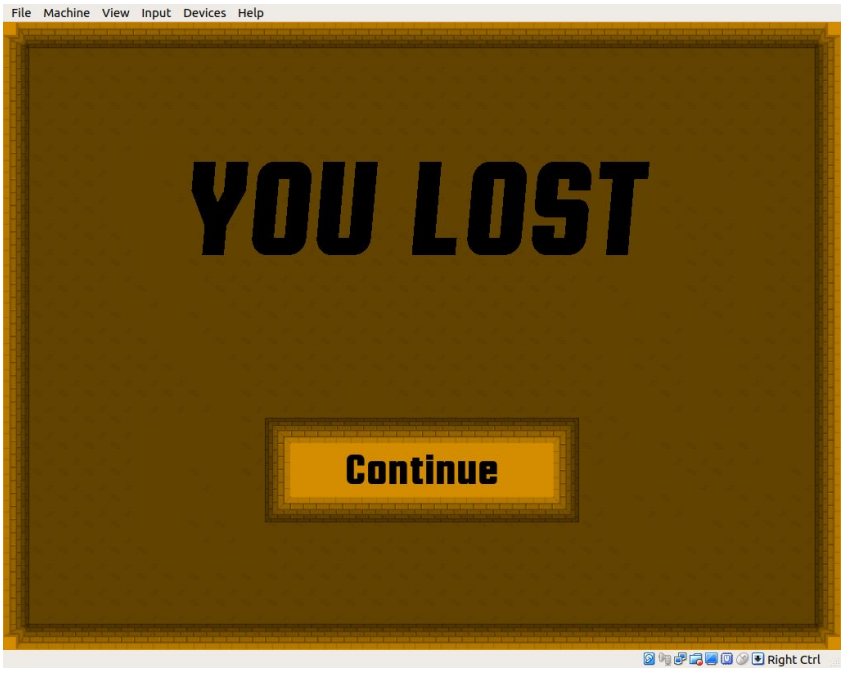
1.4 Help

Esta janela tem o propósito de mostrar um resumo do jogo, bem como as instruções para jogar e outras informações úteis. Para voltar ao menu inicial deve ser premida a tecla ESC.

1.5 Won / Lost

Quando o utilizador passa todos os níveis ou ganha no modo Multi-Player é carregado o menu do Vencedor e quando o jogador perde em qualquer um dos modos é carregado o menu do Perdedor.

O utilizador deve então carregar no botão continue com o cursor para retornar ao menu inicial.



2. Estado do Projeto

DISPOSITIVO	FUNCIONALIDADE	INTERRUPÇÕES
Timer	Controlo do frame-rate. Gerir eventos no jogo. Animações.	Sim
Keyboard	Movimentos do jogador. Close-up às alavancas. Navegar nos menus.	Sim
Mouse	Navegar nos menus. Movimento da alavanca.	Sim
Graphics Card	Apresentar a interface do jogo.	Não
Real Time Clock	Apresentar o dia e a hora nos menus.	Sim
Serial Port	Ainda em produção.	Sim

2.1 Timer

O timer é usado para controlar o frame-rate fazendo atualizações no ecrã 60 vezes por segundo.

Nos menus, o timer também é utilizado para verificar as colisões entre o cursor e os botões, alterando a sua imagem quando colide.

No modo singleplayer (função `SinglePlayerInterruptHandler`), o timer é usado para gerir todos os eventos. Permite fazer animações de idle, reload e shoot, mudando a imagem tanto do jogador como dos inimigos a cada conjunto de um certo número de frames. Neste modo é também usado para verificar colisões (com paredes ou com balas) e atualizar o movimento do jogador (função `check_movement` e `check_collision_enemy`), inimigos ou das balas e verificar se o jogador se encontra na posição final para passar para o nível seguinte.

No modo multiplayer (função `MultiPlayerInterrupthandler`), é usado para as animações e colisões dos dois jogadores com as paredes e para verificar colisões de balas.

As funções relacionadas com o timer encontram-se no **timer.c**.

2.2 Keyboard

O keyboard é usado para movimentar o jogador usando as teclas WASD ou as setas. Foram criadas variáveis bool para cada direção estáticas, para controlar o movimento do jogador, que são atualizadas na função `update_movement_direction` que recebe o make-code ou o break-code e o jogador em questão (este aspeto foi útil no modo multiplayer). Também serve para disparar, carregando na Barra de Espaço ou para entrar na janela de abertura de uma alavanca, carregando no E.

No Pause menu e na secção Help, a tecla ESC permite voltar mais facilmente ao jogo e ao menu inicial, respetivamente.

No multiplayer, a tecla ESC permite sair do jogo e retornar ao menu inicial.

As funções relacionadas com o keyboard encontram-se no **kbd.c**.

2.3 Mouse

O rato é usado para navegar nos menus usando um cursor. Através da função `mouse_events` é obtido o evento do rato, sendo possível saber se foi premido algum botão e desta forma facilitar a navegação.

Na janela da alavanca, o rato serve para realizar o movimento, onde mais uma vez é usado um cursor. É usada a função `lever_move_handler` para gerir os eventos do mouse e verificar se o movimento pretendido já foi realizado.

As funções relacionadas com o mouse encontram-se no **mouse.c** que é também o ficheiro onde guardamos informação sobre o nosso cursor e os seus movimentos.

2.4 Graphics Card

Para o projeto foi utilizado o modo de vídeo 0x118, 1024x768 com 16.8M de cores (8:8:8).

Foi usada a técnica de double buffering, em que a cada interrupção desenhamos para uma memória auxiliar e nas interrupções do timer (de modo a controlar o frame-rate) copiamos o conteúdo do nosso buffer para a memória principal usando a função `double_buffer`.

Foram usados XPMs para todas as imagens presentes no jogo. O nosso player e as suas animações (idle, reload e shoot) foram obtidas da internet, sendo estas de uso grátis. Para fazer os XPMs dos inimigos apenas foram alteradas as cores do jogador, bem como

para fazer os jogadores de cores diferentes no modo multiplayer, tendo estes também o mesmo conjunto de animações). Para fazer os mapas foi utilizado a aplicação Tiled e foi usado um tileset também encontrado na internet. As alavancas foram feitas por nós. Usamos também XPMs dos números para demonstrar o dia e a hora obtidos através do RTC.

Para testar as colisões com as paredes foi criado um mapa apenas a preto e branco onde as paredes estão a preto e a área onde o jogador se pode movimentar está a branco, sendo depois detectadas as colisões pixel a pixel. Assim como este tipo de colisões, todas as outras (desde colisões com os botões nos menus até colisões com balas ou portas) são feitas pixel a pixel.

As fontes usadas no nosso texto foram Motion Control e Alternity sendo todas de uso grátis.

Para a transparência dos XPMs é usada a cor “None”, ou, por outras palavras, a cor de transparência do modo XPM_8_8_8_8 obtida pela função `xpm_transparency_color`.

As funções relacionadas com a placa de vídeo encontram-se no ficheiro **graphics_card.c**.

2.5 RTC

O RTC é usado nos menus para obter o dia e a hora atual. São utilizadas as funções `decide_digit_xpm` e `draw_digit` para escolher, e desenhar cada dígito respetivamente. São utilizadas também as funções `add_to_background` e `clean_digit` para adicionar e limpar o dígito ao e do fundo, respetivamente, e, desta forma, evitar que o movimento do cursor por cima do relógio não o apague.

São utilizadas interrupções de update para atualizar o tempo a cada segundo. Foi criada a função `rtc_read` que recebe o registo a ler, e que é usada para ler os segundos e minutos a cada interrupção e, caso os minutos sejam 59 ou 0 (houve mudança na hora) para ler também a hora. Ao iniciar o jogo é lido também, para além destes, o dia, mês e ano. Todas as funções relacionadas com o rtc estão no **rtc.c** e no **DateTime.c** (ficheiro criado para guardar informação do nosso relógio do jogo).

2.6 Serial Port

A serial port é usada no modo multiplayer para transmitir informação entre os dois computadores. Esta informação são os make-codes e os break-codes correspondentes ao movimento do jogador (WASD ou setas) e ao disparo (Barra de Espaço), ou códigos previamente definidos para informar a outra máquina de que o jogador está, ou não, online. Após receber a informação, é chamada a função `handle_data` que atualiza as variáveis necessárias com a informação recebida.

São subscritas as interrupções do Received Data, Transmitter Holding Empty Register e Line Status. São usadas interrupções em conjunto com polling de modo a diminuir a perda de informação e os erros gerados. O bitrate é de 115200 e o UART foi configurado para enviar 8 bits por char, com paridade ímpar.

As funções relacionadas com a leitura e escrita de e para a serial port estão no ficheiro **serial_port.c**. Foram implementadas queues (ficheiro **queue.c**), com o objetivo de diminuir perdas de informação, juntamente com FIFOs do UART.

3. Organização e Estrutura do Código

3.1 Timer Module (5%)

Neste ficheiro estão presentes as funções desenvolvidas no Lab2 relacionado com as interrupções do timer. Ambos contribuíram igualmente para a realização deste módulo.

3.2 Keyboard Module (5%)

Neste ficheiro estão presentes as funções desenvolvidas no Lab3 relacionado com as interrupções do keyboard. Ambos contribuíram igualmente para a realização deste módulo.

3.3 Mouse Module (5%)

Neste ficheiro estão presentes as funções desenvolvidas no Lab4 relacionado com as interrupções do mouse. Para além disto, foi adicionado o código relativo ao Cursor (estrutura de dados criada para guardar informação sobre o cursor nos menus), os seu movimentos, colisões e funções para o desenhar e limpar. Ambos contribuíram igualmente para a realização deste módulo.

3.4 Graphics Card Module (5%)

Este módulo contém as funções desenvolvidas no Lab5 relativo à placa de vídeo. Para além disso tem também as funções relacionadas com o double buffer, implementado no projeto. Ambos contribuíram igualmente para a realização deste módulo.

3.5 RTC Module (5%)

Módulo relacionado com o Real-Time Clock. Neste módulo encontram-se as funções de comunicação com o RTC, quer para subscrever as interrupções, quer para ler os registos e atualizar a hora e dia. Este módulo foi desenvolvido pelo aluno Pedro Seixas.

3.6 Serial Port Module (10%)

Neste ficheiro encontram-se presentes as funções relacionadas com a serial port: subscrever, configurar, enviar informação e ler informação. Este módulo foi desenvolvido pelo aluno Pedro Seixas.

3.7 Bullet Module (3%)

Neste módulo encontra-se a declaração da estrutura de dados Bullet que guarda informação relativa a uma bala. Ambos contribuíram igualmente para a realização deste módulo.

3.8 Button Module (3%)

Módulo relativo à estrutura de dados Button, que guarda a informação do tipo de botão, posição, imagens, e que permite criar botões e desenhá-los. Esta estrutura facilitou significativamente a verificação de colisões. Ambos contribuíram igualmente para a realização deste módulo.

3.9 DateTime Module (5%)

Neste módulo foi criada uma estrutura DateTime que trabalha em conjunto com o RTC para guardar a informação do dia e hora, bem como mostrá-la no ecrã. Este módulo foi desenvolvido pelo aluno Paulo Ribeiro.

3.10 Door Module (3%)

Neste ficheiro encontra-se o código relativo às portas (Door), bem como a sua gestão. Ambos contribuíram igualmente para a realização deste módulo.

3.11 Enemy Module (6%)

Neste módulo estão presentes as funções relativas à estrutura Enemy, que é usada para guardar informação sobre um inimigo no jogo. Permite também a gestão de todos os eventos relacionados com os inimigos, desde disparos, movimentos e colisões. Ambos contribuíram igualmente para a realização deste módulo.

3.12 Game Module (10%)

Neste ficheiro está presente o ciclo de interrupções que permite a execução do programa. Foi implementada uma state machine quer para o estado do programa, quer para o estado do jogo em si, e para o movimento das alavancas. Consoante o estado do jogo, será chamado o Interrupt Handler correspondente, que lida com a interrupção da forma pretendida. Para além disso, é neste módulo que se encontra o código que permite o funcionamento do modo Singleplayer. Ambos contribuíram igualmente para a realização deste módulo.

3.13 Level Module (7%)

O objetivo deste módulo é a gestão dos níveis do jogo. Foi implementada uma estrutura de dados Level que guarda todas as informações relevantes sobre um nível. Desta forma, tornou a criação de novos níveis consideravelmente mais intuitiva. Ambos contribuíram igualmente para a realização deste módulo.

3.14 Lever Module (3%)

Neste ficheiro está declarada a estrutura Lever e permite a gestão dos eventos relacionados com alavancas num nível. Ambos contribuíram igualmente para a realização deste módulo.

3.15 Menu Module (6%)

Aqui está o código relacionado com os menus: os Interrupt Handlers respectivos a cada menu e as funções que permitem dar load aos menus. Ambos contribuíram igualmente para a realização deste módulo.

3.16 Multiplayer Module (7%)

É neste módulo que se encontra o código que permite o funcionamento do modo Multiplayer bem como as funções auxiliares que trabalham em conjunto com o módulo da Serial port. Este módulo foi desenvolvido pelo aluno Pedro Seixas.

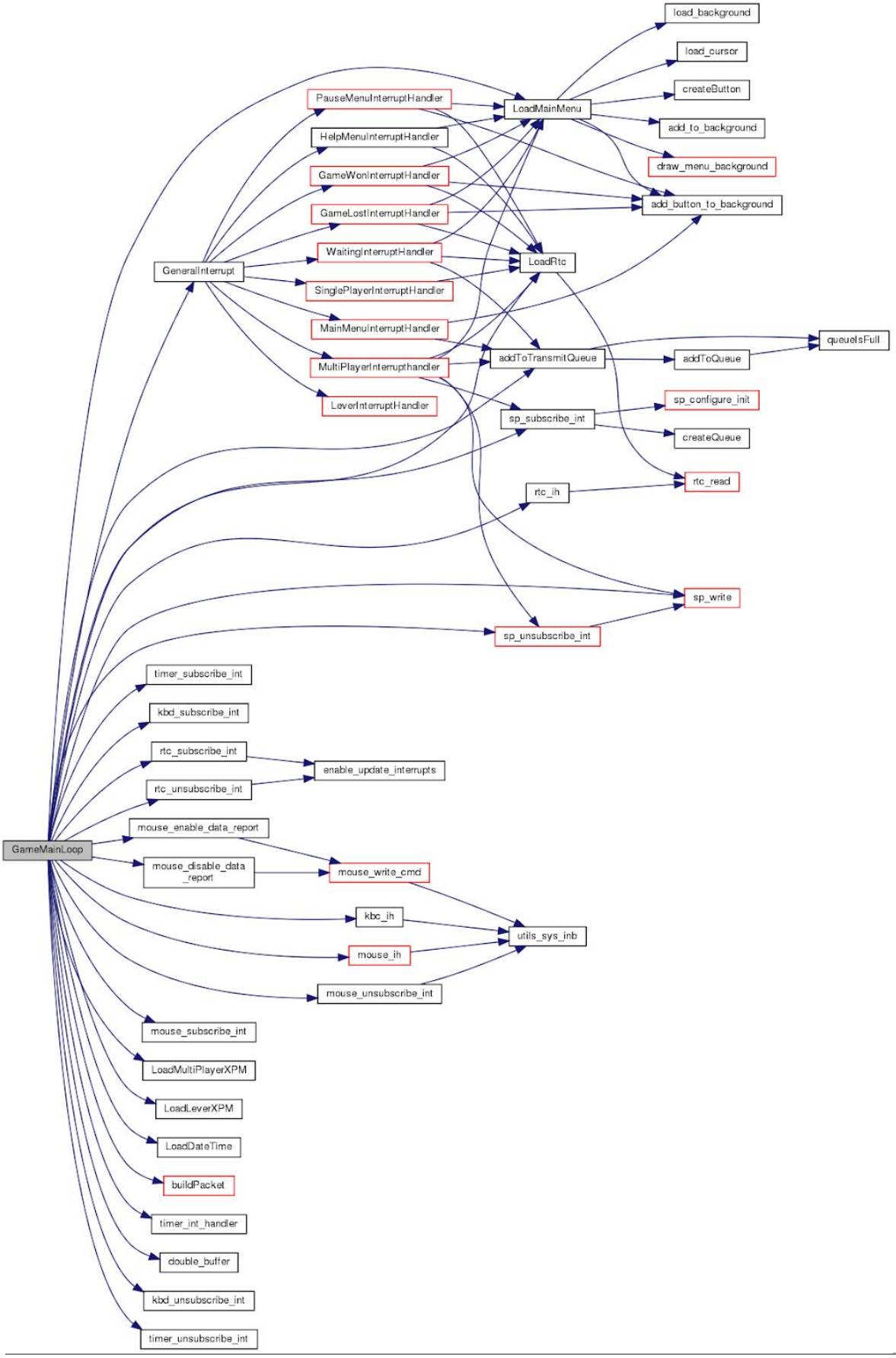
3.17 Player Module (10%)

Neste ficheiro estão todas as funções relacionadas com a estrutura de dados Player, desde o seu movimento, animações, colisões, e toda a gestão em geral de todos os eventos relacionados ao jogador. Ambos contribuíram igualmente para a realização deste módulo.

3.18 Queue Module (2%)

Módulo com a implementação das queues, úteis na leitura e transmissão de informação de e para a serial port. Este código foi retirado do site https://www.geeksforgeeks.org/queue-set-1introduction-and-array-implementation/ tendo sido adaptado para o pretendido no projeto (trabalhar com chars). Este módulo foi desenvolvido pelo aluno Pedro Seixas.

Function Call Graph



4. Detalhes de Implementação

Foi implementada uma **state machine** (GameState) para saber em que estado o jogo está. A state machine permitiu chamar diferentes Interrupt Handlers dependendo do estado o que facilitou o uso de apenas alguns dispositivos em certos momentos, e outros em diferentes ocasiões. Por outro lado, foi criada também outra state machine (SinglePlayerGameState) que permite saber o nível em que o jogador está.

Na implementação dos menus, o nosso método consistiu em ter um **fundo vazio** (sem elementos), e, a este, adicionar os botões pretendidos. Este método diminuiu o tempo demorado a dar load aos XPMs visto que apenas necessita de carregar um XPM do tamanho da janela (o fundo) e vários do tamanho dos botões, ao invés de carregar diversos fundos para diversos menus. Este método facilitou também a animação dos botões quando o cursor colide com estes, mudando a sua imagem e adicionando-a ao fundo.

Foi necessário este passo de **adicionar ao fundo** (função `add_button_to_background`) visto que, ao atualizar a posição do cursor, o cursor é primeiramente limpo (com a cor do fundo), atualizada a sua posição, e só depois desenhado outra vez. Sendo limpo com a cor do fundo, isto implicava que o cursor, ao colidir com os botões, os apagasse parcialmente, daí termos optado por adicioná-los diretamente ao map do fundo. A função que faz este procedimento (ou outras do mesmo género para outros elementos) é usada ao longo de todo o jogo para adicionar os mais diversos elementos ao jogo, para que, quer no movimento do rato, no caso dos menus, quer o movimento do jogador e dos inimigos, no caso do jogo em si, nenhum elemento seja apagado.

As **colisões** são detectadas pixel a pixel, ou seja, é verificada a cor do pixel para testar se há colisão ou não.

Nas colisões do player com as paredes (função `check_collision_walls`), é usado um XPM do nível apenas a preto e branco, onde a preto estão representadas as paredes. Esta abordagem facilitou imenso a detecção das paredes. As colisões com os carregadores (função `check_collision_ammo`) e com os inimigos (função `check_collision_enemy`), a abordagem é ligeiramente diferente. A cada frame é verificada a posição do player, e, caso algum pixel esteja na mesma posição que um inimigo

o jogador perde, caso esteja em algum carregador, o jogador recarrega (caso não tenha já uma bala). O mesmo serve tanto para as colisões com alavancas (função `check_lever_position`) como para a posição final do nível (função `check_final_position`).

Nas colisões no movimento das balas são usadas duas funções, `check_collisions_bullet` e `check_collision_with_player`, para as balas disparadas pelo player e pelos inimigos, respetivamente. Nestas funções é verificada a posição das balas e a posição de cada um dos inimigos ainda no mapa ou do jogador. As funções verificam também as colisões com paredes.

Nas colisões do cursor nos menus é usada a posição relativa do cursor, e é verificado se esta posição está dentro ou fora de algum botão. Caso esteja dentro de algum botão, a função `check_collision_main_menu` para o main menu e outras semelhantes para os outros menus retorna o ID do botão onde há colisão, e, com isso, é alterada a sua imagem.

No modo multiplayer é usada a **serial port** com interrupções e polling juntamente devido à grande quantidade de erros que eram causados quando apenas eram usadas interrupções. Após a implementação do método de polling a qualidade do jogo melhorou significativamente, e, por isso, decidimos manter os dois métodos visto que estava a funcionar corretamente. Assim como estes dois métodos, também são usadas FIFOs do UART e Queues criadas por nós, visto que, inicialmente, dando erros só com FIFOs foram implementadas Queues na tentativa de melhorar. Percebemos nesse momento que estas, trabalhando em conjunto com FIFOs, ajudavam a tornar a experiência de jogo mais agradável devido à menor quantidade de erros ou falha de mensagens.

O **RTC** foi implementado em conjunto com a nossa estrutura de dados **DateTime**, que guarda um array com os XPMs dos números e a informação sobre o dia e hora. Nas interrupções do RTC, são lidos os registos necessários (horas, minutos e segundos) e é atualizado seguidamente o valor na variável do DateTime. Decidimos optar por esta implementação por uma questão de modularidade, com o objetivo de termos um módulo apenas respectivo à comunicação com o RTC e outro que fizesse a ligação entre este e o projeto em si.

Contrariamente à **Project Specification**, houve certos aspetos que não foram conseguidos, ou simplesmente, decidimos não implementar. Após uma longa reflexão, chegamos à conclusão que, para este tipo de jogo, não faria muito sentido guardar os highscores, sendo o jogo guiado por níveis. No entanto, decidimos desenvolver a janela de ajuda para o utilizador perceber mais facilmente a essência do jogo. Outro aspeto não implementado foram as balas especiais, que infelizmente não foi possível devido à grande quantidade de erros que surgiram na implementação da serial port e ao tempo que demoramos para os resolver. Felizmente conseguimos chegar a um projeto final igualmente agradável, onde o modo multiplayer tem uma jogabilidade significativamente melhor do que o esperado quando surgiram os primeiros bugs. Por último, decidimos também, no modo Singleplayer, alterar a forma como as alavancas eram abertas. A ideia do enigma que tínhamos previamente tornou-se demasiado complexa e por isso optamos por desenvolver uma alternativa mais fácil e natural para o utilizador e que utiliza igualmente os recursos do mouse, que é o movimento de arrastar a alavanca.

5. Conclusões

Após o desenvolvimento de todo este projeto podemos afirmar que a cadeira é sem dúvida das mais interessantes que tivemos e que mais nos desafia.

Por um lado, este tipo de programação, apesar de ser complicado perceber a essência inicialmente, acaba por se tornar muito cativante e ensina-nos muito, tanto na forma de programar como na forma de pensar, tendo que trabalhar tendo em atenção à memória que utilizamos, a organização do código e o tempo que demora a correr certos fragmentos de código… algo com que nunca nos tínhamos preocupado antes. Por outro lado, existem ocasiões que é demasiado desafiante, o que momentaneamente nos deixou desmotivados.

Podemos concluir que este é o principal ponto negativo da cadeira em si. Por vezes, o facto de não haver informação suficiente sobre o que é suposto os alunos desenvolverem torna-se demasiado desafiante, e, consequentemente, exige uma carga horária maior. O que acontece por vezes é, devido às outras cadeiras, não ter tempo para acompanhar a matéria devido à grande quantidade de tempo extra-curricular que é necessário investir para realmente perceber a matéria, e isso colocar, não só os Labs, mas também o projeto final, numa dificuldade extrema, desnecessária, e que seria facilmente resolvida com uma melhor informação nos handouts por parte dos professores.

Vendo de outro lado, este aspecto faz com que tenhamos de ser nós a aprender como se deve desenvolver o código, o que, de certa forma, nos prepara para o mundo do trabalho, onde muito provavelmente, nos irão colocar numa posição semelhante.

Outro aspeto menos positivo é o facto de não existirem Labs para o RTC e para a Serial port. Isso dificultou imenso a implementação destes dispositivos no projeto final por não termos nem informação nem tempo suficiente e pensamos que sejam tópicos muito interessantes, estando a grande maioria dos alunos a perder a oportunidade de trabalhar com eles.