

LTRON

Um versão de Tron para Minix

Group 9 - Class 2

Carlos Jorge Direito Albuquerque (up201706735)

Paulo Daniel da Silva Araújo Marques (up201705615)

Índice:

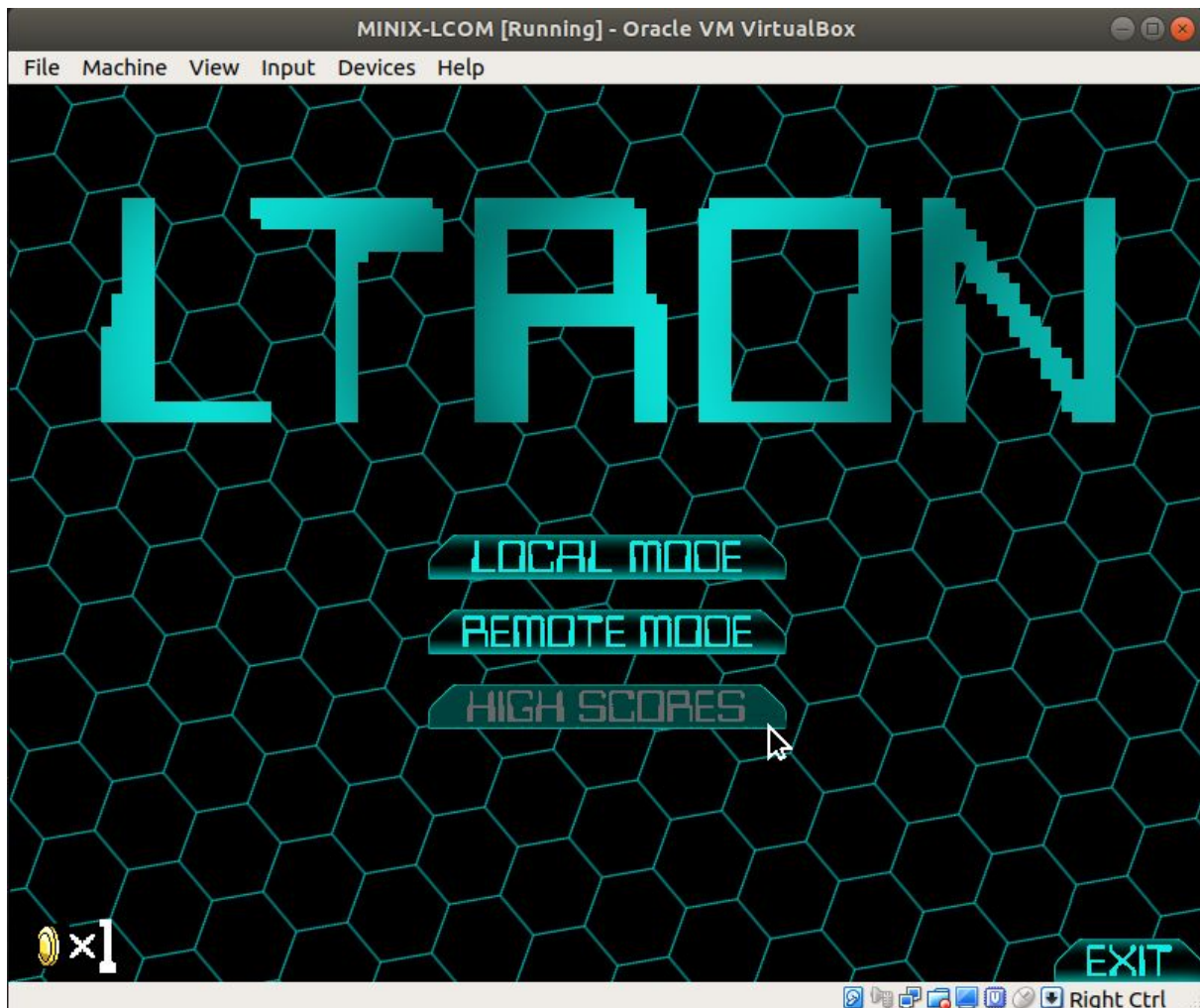
Índice:	2
1- Instruções de utilização:	4
1.1- Local Mode	5
1.2- Remote Mode	6
1.3 Highscores	7
1.4 Exit	8
2-Estado do Projeto	9
2.1-Timer	9
2.2 Keyboard	10
2.3 Mouse	10
2.4 Video Card	11
2.5 Real Time Clock (RTC)	12
2.6 Serial Port	12
3-Organização/Estrutura do Código	13
3.1 Módulos Timer	13
3.1.1 Módulo Timer	13
3.1.2 Módulo i8254	13
3.2 Módulos Keyboard	14
3.2.1 Módulo i8042	14
3.2.2 Módulo Keyboard	14
3.3 Módulo Mouse	14
3.4 Módulos Video Card	15
3.4.1 Módulo vbe	15
3.4.2 Módulo v_gr	15
3.4.3 Módulo vMacros	15
3.4.4 Módulo lmlib	15
3.5 Módulo RTC	16
3.6 Módulos do Serial Port	16
3.6.1 Módulo serialPort	16
3.6.2 Módulo uart_macros	17
3.6.3 Módulo Protocol	17
3.7 Módulo Game	17
3.8 Módulo Player	17
3.9 Módulo Lógica	18
3.10 Módulo Menu	20

3.11 Módulo Highscore	21
3.12 Módulo Loop	21
3.13 Módulo Sprites	23
3.14 Módulo XPM	23
3.15 Módulo AnimSprite	23
4-Detalhes de Implementação	23
5-Conclusões	26

1- Instruções de utilização:

O programa inicia no menu principal. Nele são apresentados quatro botões:

1. Local Mode - Inicia o jogo em modo multijogador local (ver secção 1.1);
2. Remote Mode - Inicia o jogo em multijogador remoto (ver secção 1.2);
3. High Scores - Entra no menu de High Scores (ver secção 1.3);
4. Exit - Termina o programa, mostrando o janela de Créditos (ver secção 1.4).



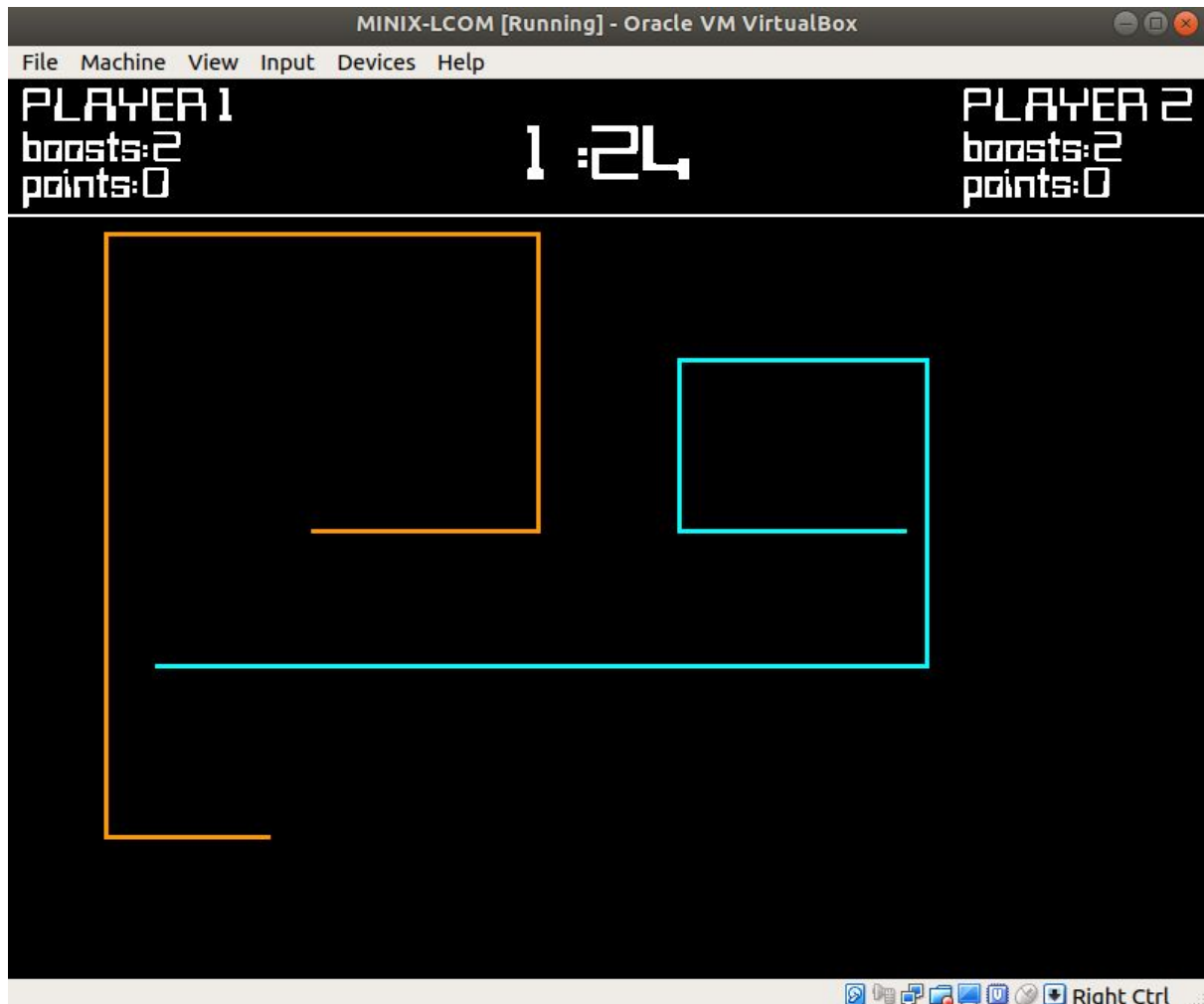
Para escolher uma das opções acima enumeradas basta clicar com o botão esquerdo do rato sobre o botão pretendido.

É também apresentado o contador de moedas (canto inferior esquerdo) que regista o número de créditos restantes. Para poder jogar, quer em modo local quer em modo remoto, deve ter pelo menos um crédito inserido. Para inserir créditos o jogador deve clicar com o botão direito do rato, não sendo relevante o sítio onde efetua o clique. O número máximo de créditos permitidos é 10.

1.1- Local Mode

Quando selecionada a opção Local Mode, o jogo propriamente dito é carregado.

Com efeito, o ecrã contém o tempo de jogo e os pontos correspondentes ao player 1 e player 2, assim como os boost de cada um.



Após pressionar o botão de ESC o jogo começa, aparecendo no ecrã as motas de ambos os jogadores a moverem-se, cada uma deixando um rasto de cor diferente. O jogador 1 é a mota da esquerda, que deixa um rasto azul ciano. O jogador 2 é a mota da direita, que deixa um rasto cor de laranja.

O jogo começa com ambos os jogadores com 0 pontos e termina quando um deles atingir 3 vitórias, voltando para o menu principal. Cada ronda começa com os jogadores nas suas respetivas posições e ambos possuem 3 boosts que podem usar nessa ronda. Tal como o nome sugere, o boost faz com que a mota do jogador se desloque mais rápido durante 1 segundo. A ronda termina quando um dos jogadores colidir contra um rasto deixado quer pelo inimigo, quer por si mesmo, ou contra os limites do espaço de jogo. Caso nenhum dos jogadores tenha perdido até o tempo acabar, o jogo passa a um estado de Morte Súbita - ambos os jogadores

ficam sem boosts e passam a andar à velocidade do boost, forçando assim um jogo mais intenso, rápido e divertido.

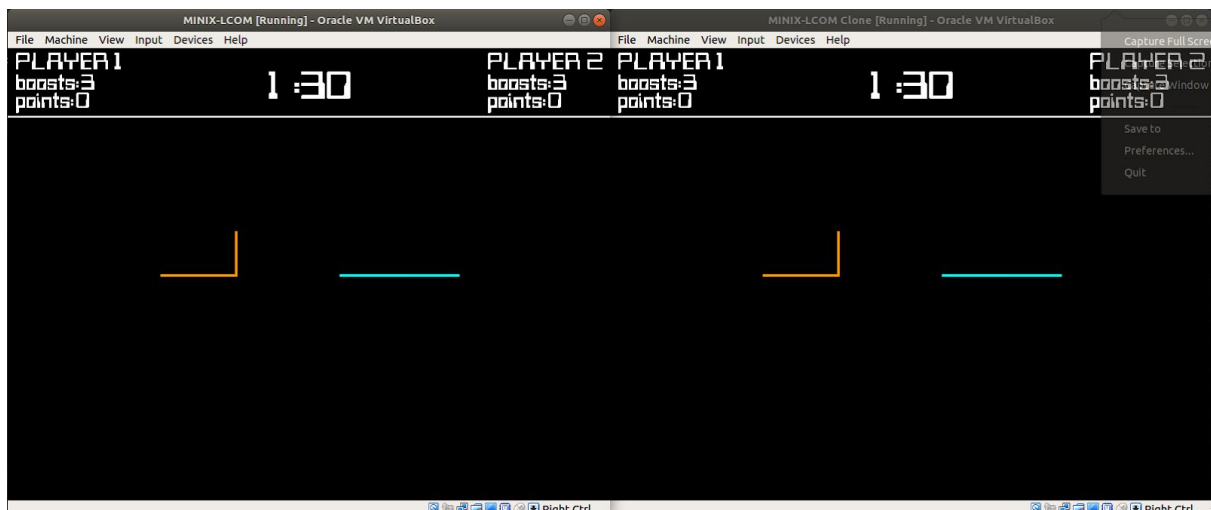
As teclas para controlar a mota do player 1 são 'W', 'S', 'A', 'D' para o movimento e o 'SPACE' para ativar o boost. Já para a mota do player 2, as setas direcionais são usadas para movimento e o 'ENTER' para ativar o boost. Só é possível mudar a direção da mota em 90°, isto é, não é possível virar para a direção de onde se vinha, caso contrário haveria uma colisão imediata e o jogador perderia instantaneamente.

Caso se pressione a tecla ESC durante o decorrer do jogo, este entra num estado de pausa. Para voltar ao jogo basta pressionar ESC mais uma vez.

1.2- Remote Mode

Quando seleccionada a opção Remote Mode o jogo é carregado e vai ser jogado por duas pessoas em computadores distintos. O jogador que seleccionar primeiro a opção Remote Mode será considerado o Host da sessão, ou seja, é ele quem pode começar, pausar e retomar o andamento do jogo e é o player 1. Contudo, o Host não pode começar o jogo enquanto o Guest não se tiver conectado. Para isso basta que o outro jogador escolha a opção Remote Mode no seu programa e, como foi o segundo a entrar na partida, passará a ser o Guest, pelo que não pode começar, pausar ou retomar o jogo e será o player 2.

A partir do momento em que ambos os jogadores tenham seleccionado a opção Remote Mode o jogo está pronto para decorrer normalmente, esperando que o Host/player 1 pressione ESC para começar.



Desta vez, como os jogadores estão em computadores diferentes, ambos utilizam as teclas 'W', 'S', 'A' e 'D' para movimentar a sua mota e o botão esquerdo do rato para ativar o boost.

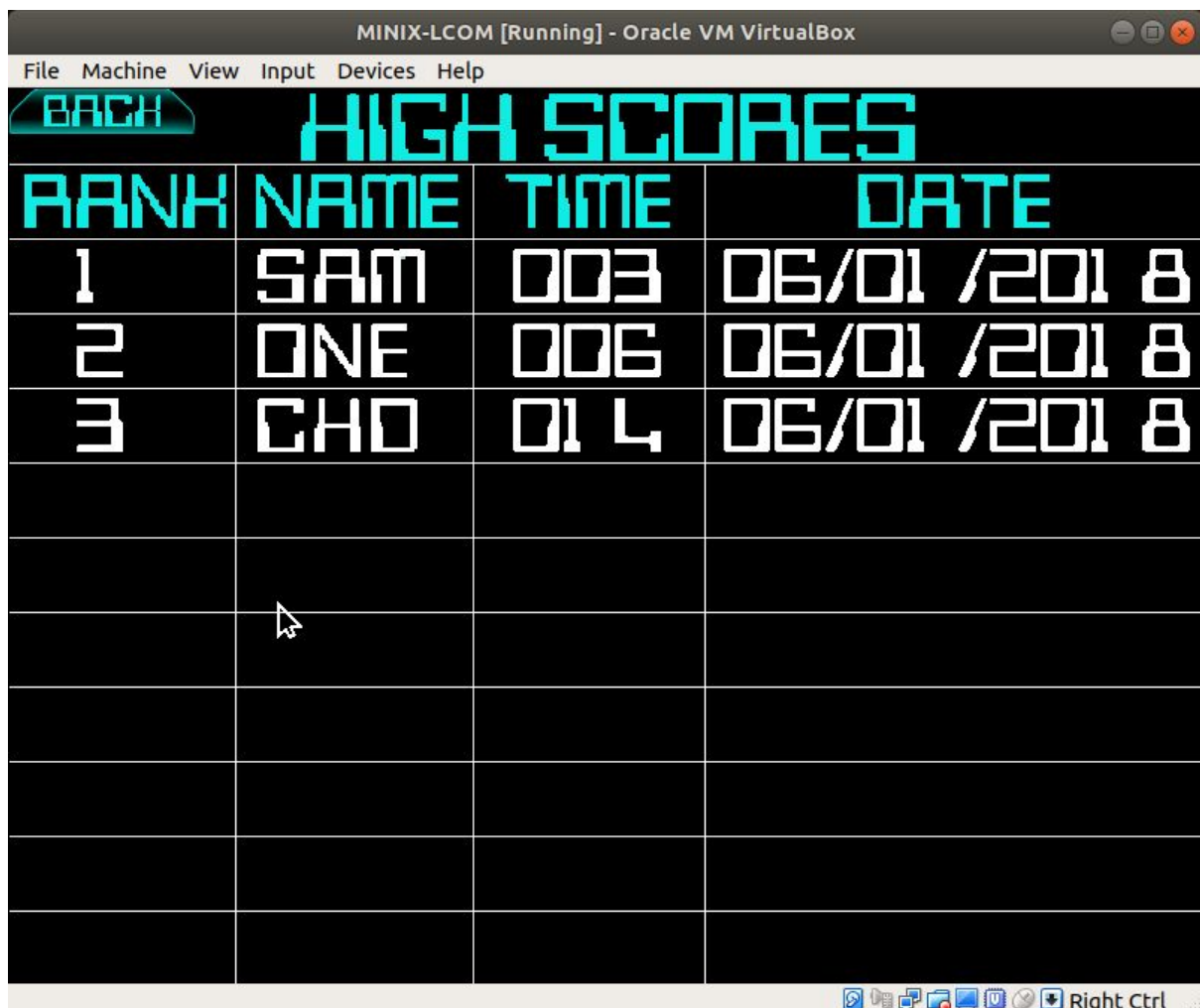
1.3 Highscores

Existem duas maneiras de chegar a este menu: a partir do botão “High Scores” do menu principal ou após o final do jogo se a classificação obtida for um highscore.

Ao carregar no botão “High Scores” abre-se um novo menu onde estão registadas as melhores classificações. Desta forma não é possível alterar nenhuma das pontuações guardadas. Uma classificação é caracterizada por 4 elementos distintos:

1. Rank - lugar da classificação;
2. Name - 3 caracteres que representam o nome do jogador;
3. Time - tempo decorrido de jogo;
4. Date - data em que foi registada a classificação.

Melhores tempos (vitórias mais rápidas) implicam melhor classificação. Em caso de empate a classificação que foi registada primeiro é a que aparece com maior rank.



RANK	NAME	TIME	DATE
1	SAM	003	06/01 /201 8
2	ONE	006	06/01 /201 8
3	CHO	01 4	06/01 /201 8

Ao obter uma boa classificação no final do jogo, o jogador que ganhou é reencaminhado para este menu. Em caso de empate, a classificação é descartada pois não houve um vencedor. Ao entrar neste menu o jogador vai-se deparar com uma linha cujo nome é “AAA”. Esta linha representa o novo highscore cujo nome ainda não foi definido. Caso o jogador não pretenda registar a classificação, basta premir a tecla ESC para a eliminar da tabela. Para alterar o nome a guardar, o jogador deve clicar com o rato nos caracteres do nome - um clique com o botão esquerdo passa à letra seguinte do alfabeto e um clique com o botão direito à letra anterior do alfabeto. Após ter acabado de editar o nome, o jogador tanto pode premir a tecla ENTER para confirmar o nome, deixando assim de poder alterá-lo e a classificação passando a ser definitiva, como pode simplesmente com o rato carregar sobre o botão BACK, que não só confirma o nome como também retorna o jogador para o menu inicial.

É de notar que só são registadas as 3 melhores classificações e que estas se perdem ao fim de cada sessão (ao sair do jogo).

1.4 Exit

Ao seleccionar a opção Exit o programa encerra após mostrar a janela de Créditos durante 3 segundos.



2-Estado do Projeto

A seguinte tabela apresenta as funcionalidades e os periféricos que utilizamos no nosso programa. A inicialização e configuração destes periféricos é feita em *game_start()* e o seu uso em *game_loop()* (onde está presente o ciclo *driver_receive()*), ambas funções fazem parte do Módulo Loop.

Periférico	Utilização	Interrupções
Timer	Controlo de frames e dos ticks do jogo	Sim
Keyboard	Controlo das personagens (motas) do jogo	Sim
Mouse	Selecionar os botões no menu principal Ativar o boost em modo remote Editar o nome no menu High Scores	Sim
Video Card	Mostrar no ecrã os vários elementos do jogo e os vários menus existentes	Não se aplica
RTC	Obter data e hora para os Highscores Interrupções periódicas para registar o tempo decorrido em jogo	Sim
Serial Port	Comunicação entre dois computadores diferentes	Sim

2.1-Timer

O timer é responsável por controlar os frames do jogo e a lógica através de interrupções. Cada interrupção é tratada pela função *timer_ih()* (Módulo Timer) e após o tratamento da interrupção é chamada a função *timer_logic()* (Módulo Lógica), que tem em conta a máquina de estados que coordena o que fazer dependendo de qual o estado do programa quando ocorreu a interrupção. Os vários estados do programa são enumerados em *game_state_t* (Módulo Lógica), permitindo assim

uma gestão fácil e intuitiva de qual o estado atual (contido na variável global estática *main_state* presente no Módulo Lógica).

O timer é importante para todos os estados do jogo pois é o responsável por invocar *pageFlip()* do Módulo Video Card, função que faz o page flipping essencial no sistema de double buffering que implementamos no programa. Contudo, é mais utilizado no menu principal e em ambos os modos de jogo.

No menu principal, o timer é responsável pela animação da moeda no canto inferior esquerdo. Nos modos de jogo (quer seja remoto ou local) a cada interrupção do timer é efetuada uma atualização da posição dos jogadores e uma atualização do que é mostrado no ecrã (função *game_tick()* do Módulo Game), havendo 60 interrupções por segundo (frequência normal do timer) atingimos assim 60 fps. A cada 60 interrupções é deduzido 1 segundo ao tempo restante de jogo.

2.2 Keyboard

O keyboard é responsável pelos controlo do movimento dos jogadores (tanto em modo remote como em local) e ativação dos boosts de cada um (quando em modo local), bem como pela confirmação ou anulamento de um highscore. É apenas relevante para os modos de jogo e para o menu de highscores. Cada interrupção é tratada pela função *kbd_event()* do Módulo Keyboard que, não só chama o *kbd_ih()* do Módulo Keyboard, como também interpreta os dados lidos para gerar um evento (de entre os listados em *event_keyboard* do Módulo Lógica), passando assim de um baixo nível de tratamento da informação para um alto nível de abstração. Após o tratamento da interrupção, caso tenha havido um evento (o que acontece sempre exceto nos casos em que a tecla carregada gera dois bytes) este é passado para a função *kbd_logic()* do Módulo Lógica.

Caso o estado do programa seja um modo de jogo, são chamadas funções (todas do Módulo Game) como *set_direction()* - altera a direção da mota de um jogador, *boost_triger()* - se possível ativa o boost de um jogador, e *game_runner()* - retoma ou coloca em pausa o jogo. Caso esteja no menu de highscores, aquando a inserção de um novo highscore, são chamadas as funções do Módulo Highscore *cancel_highscore()* (cancela a inserção) e *finish_score()* (confirma/finaliza a edição do novo highscore).

2.3 Mouse

O mouse é utilizado em todos os estados do jogo exceto no modo local (*playing_local*). São utilizadas tanto a posição como os botões do mouse e, por vezes, os dois em simultâneo (cliques em diferentes botões). Para tal, analogamente ao que acontece no keyboard, após o tratamento de uma interrupção pela função *mouse_ih()* do Módulo Moude, a função *mouse_get_event()* do mesmo Módulo analisa o packet recebido pelo rato e gera um evento (um dos valores

presentes em *mouse_event* presente no Módulo Mouse) que depois será interpretado por *mouse_logic()* do Módulo Lógica.

No menu principal o mouse tem um cursor (Módulo Sprite) associado e colocar o rato sobre um botão torna-o realçado. Carregar no botão esquerdo seleciona a opção descrita no botão. Para tal, são chamadas as seguintes funções do Módulo Menu:

- *move_cursor()* - altera as coordenadas do cursor sempre que houver movimento no mouse;
- *mouse_hover()* - verifica se o cursor está sobre algum dos botões e, caso esteja, altera as propriedades desse botão (struct *button_t* do Módulo Menu) para o realçar;
- *menu_assert_click()* - verifica se o clique efetuado foi sobre algum dos quatro botões e, caso tenha sido, efetua a operação associada ao botão clicado.

No menu de highscores o mouse tem também um papel muito importante, visto que é com ele que se pode editar o nome de um novo highscore. Para tal, o menu de highscores possui também um cursor associado ao rato e sempre que se carrega com o botão esquerdo ou com o botão direito numa das 3 letras do nome essa letra é alterada para a letra anterior ou posterior no alfabeto, respetivamente. Para tal, as funções do Módulo Highscore *digit_hover()* (verifica se o cursor está sobre alguma das letras editáveis), *hs_assert_click()* (verifica se o clique efetuado foi numa das letras editáveis) e *edit_name()* (edita efetivamente a letra onde ocorreu o clique) são essenciais e permitem ao Módulo Mouse, através do Módulo Loop e do Módulo Lógica, interagir com o Módulo Highscore.

No modo remoto, a posição do rato torna-se irrelevante e a única função necessária é *boost_trigger()* do Módulo Game que é invocada quando é pressionado o botão esquerdo do mouse.

2.4 Video Card

A video card é essencial para tudo o que acontece, pois sem ela não era possível apresentar nada ao utilizador. Em todo e qualquer momento de execução a video card está a ser utilizada. A função *vg_init()* do Módulo Video Card aloca a memória em espaço de endereçamento virtual necessária para 3 buffers de vídeo: um deles é o mapeamento da memória vídeo propriamente dita (*video_mem* do Módulo Video Card); o segundo é um buffer auxiliar (*buffer_mem* do mesmo Módulo) onde tudo é escrito antes de ser enviado para a memória vídeo, o que, a par com a função *pageFlip()* (do mesmo Módulo) que copia a memória do buffer auxiliar para a memória de vídeo, permitem utilizar Double Buffering para tornar as animações e movimentos mais suaves; o terceiro e último é um buffer para os fundos (*background_buffer* do mesmo Módulo) dos menus, principal e high scores, que permite desenhar todos os elementos dinâmicos sobre um fundo estático sem ter que estar a redesenhar tudo novamente, causando assim um enorme impacto

positivo no desempenho do programa nos menus). Para além disto a função *vg_init()* inicializa a video card em modo VBE 0x115, com endereçamento direto de cor, em que cada componente possui 8 bits, fazendo um total de 24 bits por pixel e um número total de 16.777.216 cores disponíveis para utilizarmos.

Para definir qual o modo a utilizar recorreremos à função 0x02 - Set VBE Mode, e para guardar informações sobre o modo escolhido (resolução horizontal e vertical, número de bits por componente e sua posição, entre outras) utilizamos a função 0x01 Return VBE Mode Information.

Fazendo uso da biblioteca fornecida pelos professores que permite a fácil manipulação de pixelmaps (incluída em *lcom/lcf.h*) e de uma *font* que descarregamos da internet (com devida licença de utilização, incluída no módulo Files do Redmine) criámos uma biblioteca de imagens (conteúdo do Módulo Resources) usada em toda o programa através de sprites e animSprites (outras duas bibliotecas fornecidas pelos professores nos slides apresentados na aula que foram ajustadas às necessidades do nosso programa). Assim, conseguimos ter uma moeda animada no canto inferior esquerdo do menu principal e todo um HUD durante os modos de jogo para os jogadores saberem não só quantos pontos têm, mas também o tempo de jogo restante e quantos boosts ainda têm disponíveis.

Durante os modos de jogo a Video Card tem um papel especialmente importante porque existe detecção de colisão, isto é, as funções *draw_trail()* e *check_collision()* do Módulo Player verificam se na memória de vídeo já foi desenhado algum pixel com cor. Caso isso aconteça então o jogador colidiu e perdeu a ronda. Sem esta capacidade a essência do jogo não seria possível.

2.5 Real Time Clock (RTC)

O RTC, embora usado em poucas situações, é essencial para o Módulo Highscore. Através de interrupções periódicas a cada 0.5ms que são ativadas no início do jogo é possível, quando o jogo acaba, saber qual a duração do mesmo de forma eficiente e rápida. Para além disso, é lida logo no início do programa a data usando as funções *read_year()*, *read_month()* e *read_day()* do Módulo RTC para depois ser registada a data em que foi alcançada determinada classificação.

Para a configuração das interrupções periódicas do RTC é invocada a função *rtc_set_periodic_rate()* do Módulo RTC que define a frequência das interrupções e as funções *rtc_enable_periodic()* e *rtc_disable_periodic()* que ativam e desativam, respectivamente, as interrupções periódicas do RTC.

2.6 Serial Port

O serial port é usado para fazer a comunicação entre duas máquina para quando o jogo é utilizado em modo remote. Usamos o serial port com interrupções para a receção de informação e com polling para o envio desta. Também são

usadas fifos para a receção. Para estas gerarem interrupções basta estas terem um byte para que a dessincronização entre os jogadores seja mínima. Para o processo de receção e de envio de informação são usadas respectivamente as funções *uart_read_data()* e *uart_write_message()* .

A configuração usada da porta séria é 8 bits por caractere, com 1 bit de stop e paridade ímpar. O bitrate utilizado é 9600 bits por segundo. Toda esta configuração é realizada por *uart_init()*. A configuração escolhida justifica-se com a necessidade dos eventos de serial port serem transmitidos o mais depressa possível, tentando evitar ao máximo a dessincronização.

A informação trocada entre os serial ports consiste primeiramente num byte trocado entre os dois, para decidir quem é o jogador 1 e o jogador 2, ou seja, o computador que clicar primeiro no modo remote envia um byte de conexão ao outro, para “avisar” que o emissor desse byte é o player 1(host), e consequentemente quando o outro computador clica em remote assume-se automaticamente em player 2 (guest).

A partir do momento que ambos os jogadores estão em modo remote, o host pode dar início ao jogo clicando no botão de ESC sendo enviado um código para o guest começar o jogo simultaneamente ao host. A partir daí são trocados eventos, bytes que codificam eventos no jogo, como mudanças de direção dos jogadores e assim. Sempre que um jogador gera algum evento o serial port transmite ao outro e este actualiza a sua versão do jogo. Todos os bytes transmitidos aqui estão guardados no módulo do protocolo e a sua frequência depende da frequência dos inputs do jogadores.

Qualquer interrupção gerada pelo serial port é atendida pelo *uart_event()*.

3-Organização/Estrutura do Código

Ao lado do nome encontra-se o peso do módulo no projeto

3.1 Módulos Timer(5%)

3.1.1 Módulo Timer

Neste módulo encontra-se funções para a manipulação direta do timer como: o interrupt handler, funções de subscrição, manipulação de registos(...). Este módulo foi desenvolvido no decorrer do Lab2, tendo sido realizado igualmente por ambos os membros.

3.1.2 Módulo i8254

Neste módulo encontra-se todas as constantes simbólicas usadas na manipulação do timer. Este módulo foi-nos fornecido para a realização do Lab2, não sendo, portanto da nossa autoria.

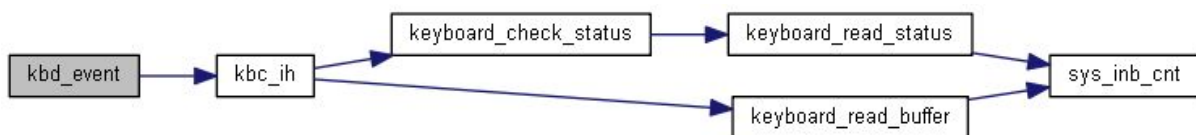
3.2 Módulos Keyboard(5%)

3.2.1 Módulo i8042

Neste módulo encontra-se todas as constantes simbólicas usadas na manipulação do KBC, usado tanto para o teclado como para o rato. Este módulo foi realizado no decorrer do Lab3 e aumento no decorrer do Lab4, tendo sido realizado igualmente por ambos os membros.

3.2.2 Módulo Keyboard

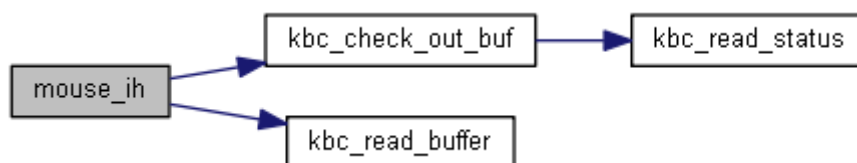
Este módulo contém todas as funções para manipulação do kbc e consequentemente do teclado, como o interrupt handler, funções de subscrição de interrupção, escrita e leitura de registos do kbc,...). Este módulo foi realizado no decorrer do Lab3, tendo sido realizado igualmente por ambos os membros. Contém também uma função que torna as interrupções em eventos para serem interpretados pela logic (*kbd_event()*), feita por Paulo Marques.



3.3 Módulo Mouse(5%)

Este módulo contém todas as funções para manipulação do kbc e consequentemente do rato, como o interrupt handler, funções de subscrição de interrupção, escrita e leitura de registos do kbc,...). Contém também a função `mouse_get_event` transforma a informação vinda da interrupção em eventos.

Este módulo foi realizado no decorrer do Lab4, tendo sido realizado igualmente por ambos os membros.



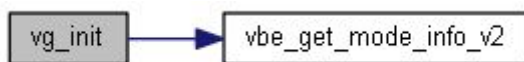
3.4 Módulos Video Card(5%)

3.4.1 Módulo vbe

Este módulo contém as funções e estruturas de dados usadas na manipulação da VBE. Este modo foi-nos fornecido na realização do Lab5. Foi também desenvolvido no lab5 a função `vbe_get_mode_info_v2()` (presente em `v_gr`), pelo o Paulo Marques, como alternativa a `vbe_get_mode_info()` presente no VBE.

3.4.2 Módulo v_gr

Neste módulo encontram-se funções para manipulação direta dos buffers de vídeo e obter informação dos mesmo e a função de pageflip. Esta módulo foi realizado no contexto do Lab5 maioritariamente por Paulo Marques, tendo sido modificado posteriormente para incorporar o double buffer pelo mesmo.



3.4.3 Módulo vMacros

Neste módulo encontram-se todas as constantes simbólicas para manipulação de serviços da BIOS e, mais especificamente, da VBE. Este módulo foi desenvolvido no decorrer do Lab5, tendo sido realizado igualmente por ambos os membros.

3.4.4 Módulo lmlib

Este módulo contém funções para a manipulação da memória no primeiro MegaByte da memória física (necessário para a BIOS). Este módulo foi fornecido para a realização do Laboratório 5, não sendo, portanto, da nossa autoria. (Módulo incluído pela LCF)

3.5 Módulo RTC(5%)

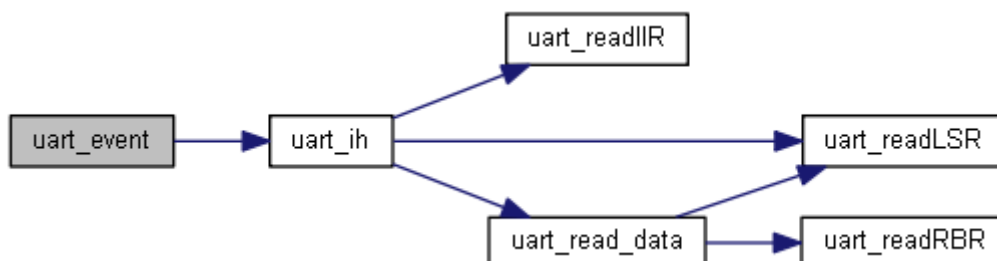
Neste módulo encontram-se as funções responsáveis por manipular diretamente o RTC, ou seja, contém o interrupt handler, funções de subscrição de interrupções, funções de escrita e leitura dos registos do RTC, funções para programar as interrupções periódicas... Contém, também todas as constantes simbólicas usadas para a manipulação do RTC. Este módulo foi desenvolvido maioritariamente por Paulo Marques, mas também com a contribuição de Carlos Albuquerque na função responsável pela alteração da frequência das interrupções periódicas.



3.6 Módulos do Serial Port(10%)

3.6.1 Módulo serialPort

Este módulo contém as funções responsáveis por manipular diretamente o UART, ou seja, contém o interrupt handler, funções de subscrição de interrupções, funções de escrita e leitura dos registos do UART, configuração do UART,...). Contém também uma função que converte as interrupções do UART em eventos para serem interpretados pelo módulo de lógica(uart_event()). Este módulo foi desenvolvido por Paulo Marques.



3.6.2 Módulo uart_macros

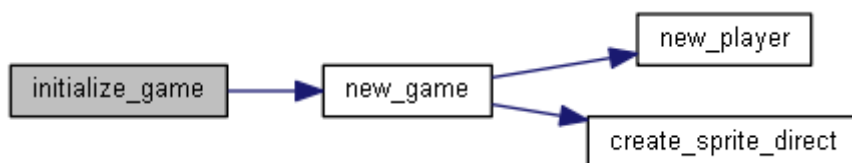
Este módulo contém as constantes simbólicas para a manipulação do UART e dos seus registos. Este módulo foi desenvolvido por Paulo Marques.

3.6.3 Módulo Protocol

Este módulo contém as constantes simbólicas e dados para o funcionamento do protocolo desenhado para a comunicação usando a porta série. Este módulo foi desenvolvido por Paulo Marques.

3.7 Módulo Game(5%)

Este módulo contém a implementação da classe Game, que é uma representação do jogo propriamente dito. Esta classe guarda toda a informação do jogo tal como os jogadores(que são uma classe em si, descrita no módulo seguinte), o tempo de jogo, os pontos, sprites usadas no hud do jogo,(...). Contém também as funções usadas para manipular a informação do jogo. Este módulo foi maioritariamente implementado por Paulo Marques, tendo Carlos Albuquerque implementado as funções *end_round* e *game_tick*.

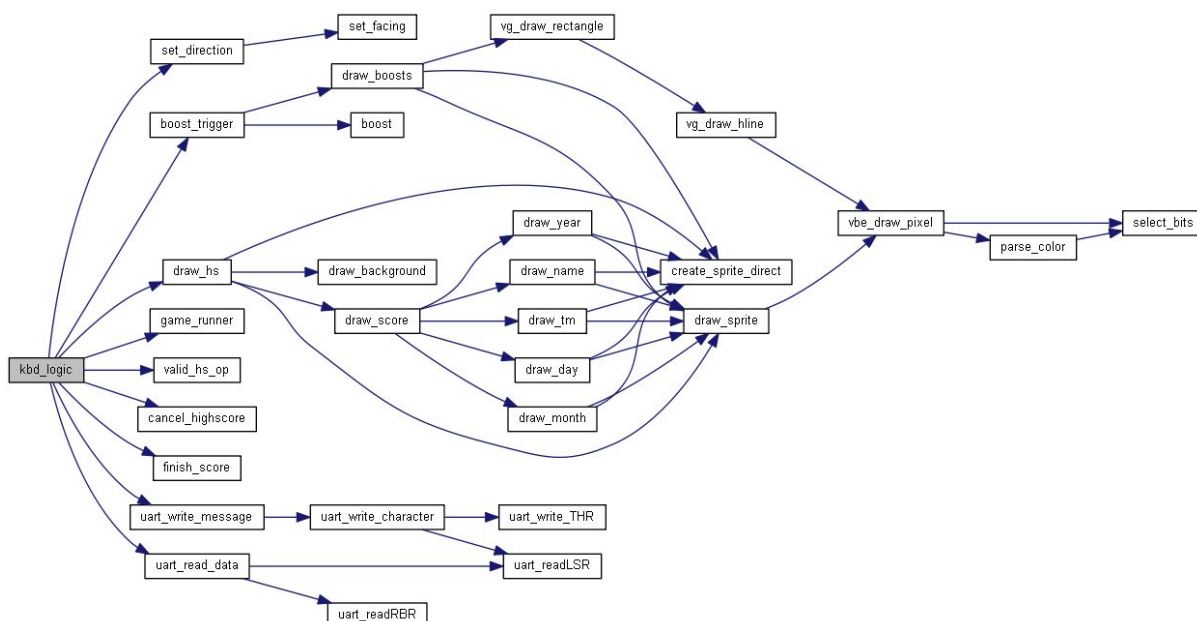


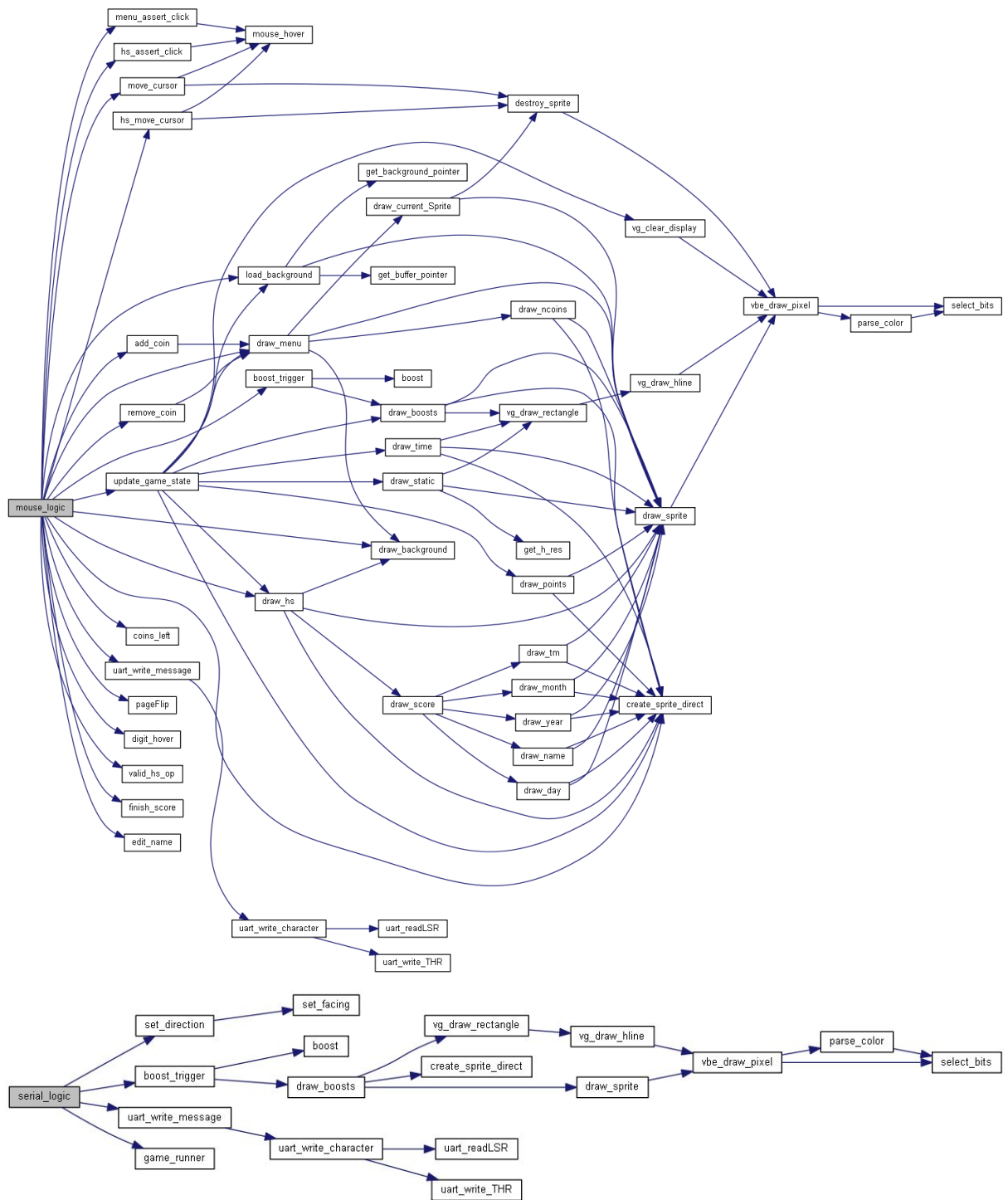
3.8 Módulo Player(5%)

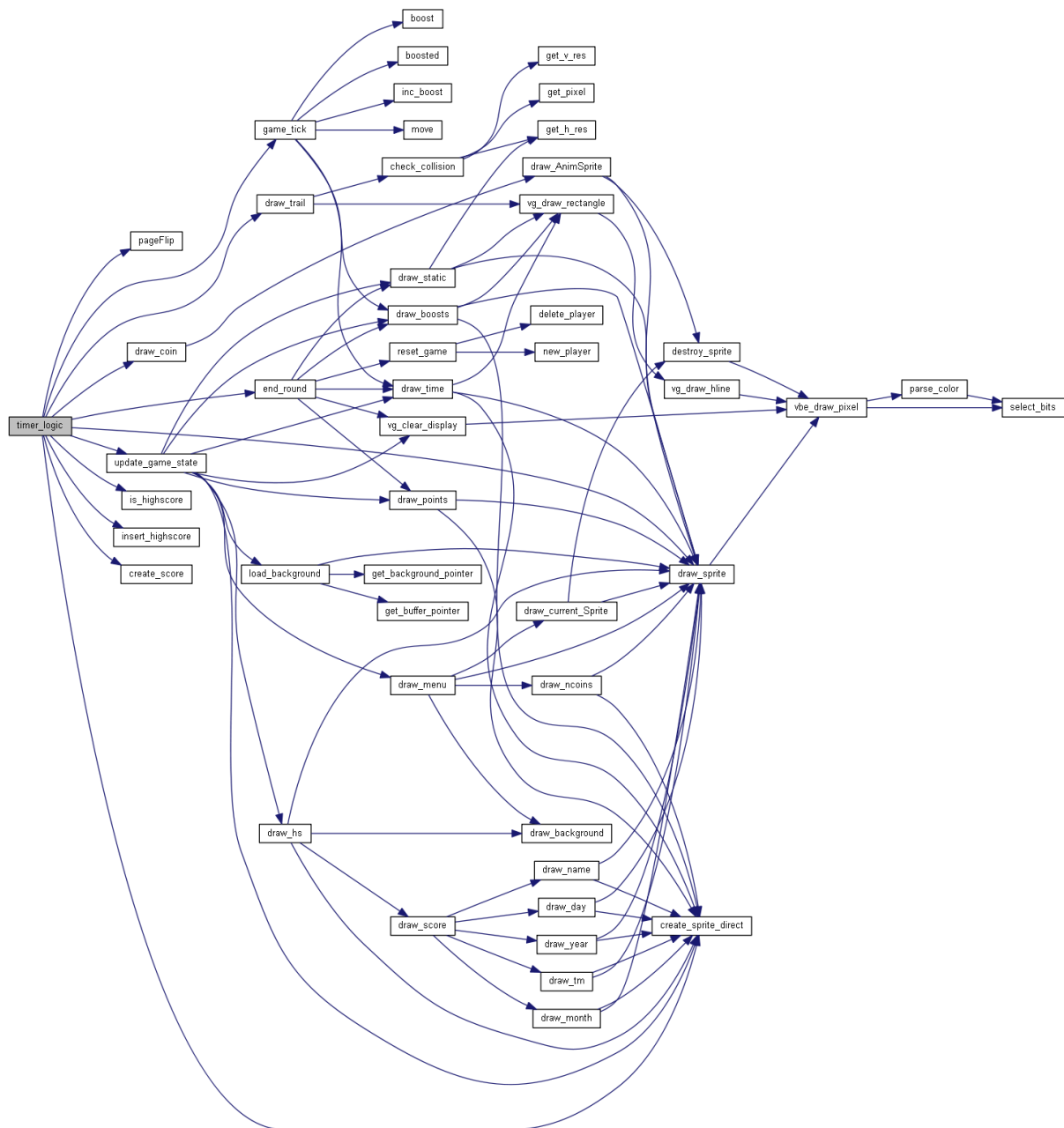
Este módulo contém a implementação da classe player, que é uma abstração de toda a informação necessária para cada jogador, tal como a velocidade, o número de boosts, as coordenadas da mota,(..). Contém também todas as funções necessárias para interagir com a classe. Este módulo foi desenhado por Carlos Albuquerque, tendo sido implementado pelo mesmo com algumas alterações menores por Paulo Marques.

3.9 Módulo Lógica(20%)

Este módulo é um dos mais importantes do projeto. Com efeito, o módulo lógica funciona como um tradutor entres os módulos do periférico e os módulos de maior nível de abstração do jogo, ou seja, recebe eventos provenientes dos periféricos e processa-os, transformando-os em ações no jogo, como por exemplo o premir de uma tecla faz a mota mudar de direção. Contém também, a máquina de estados usada para decidir o que os eventos significam num determinado contexto do programa. Este módulo foi desenhado por Paulo Marques, tendo sido implementado de forma igual por ambos os membros, ficando Paulo Marques responsável pela lógica do serial port, do keyboard, pela iniciação dos objetos usados e pela máquina de estados. Carlos Albuquerque ficou responsável pela lógica do timer e do rato, e por toda a lógica usada no estado de highscores. Este módulo corresponde a 20% do trabalho.

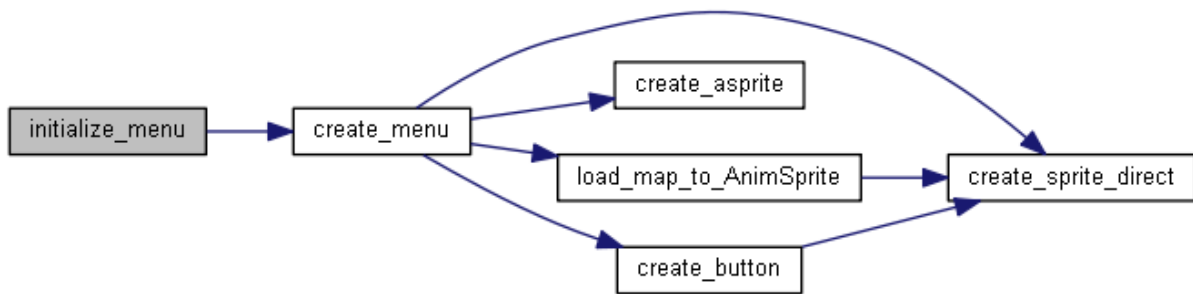






3.10 Módulo Menu(5%)

Este módulo é responsável pela classe botão e pela classe menu. Estes são abstrações usadas para interagir com a informação do menu inicial. Também é responsável por alguma interação do utilizador com os botões através do rato e mudar o estado do jogo de acordo. Paulo Marques implementou a base da class “menu” tendo Carlos Albuquerque implementado tudo o resto.

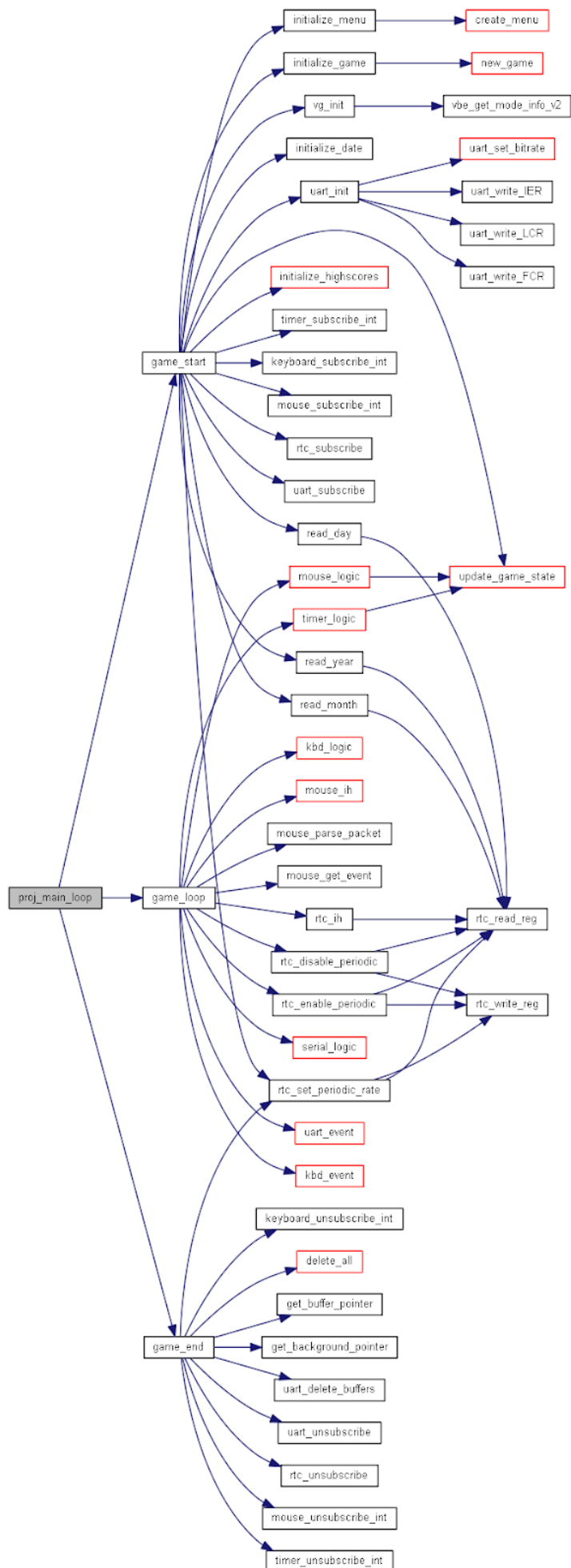


3.11 Módulo Highscore(9%)

Este módulo é responsável por criar, gerir, modificar e eliminar classificações e o menu dos highscores. Para tal são utilizadas duas estruturas (*highscores_t* e *score_info_t*) para uma aproximação à programação orientada a objetos de modo a facilitar a gestão e instanciação de objetos de ambas as estruturas/classes. Guardando no atributo *scores* da classe *highscores_t* todas as classificações até ao momento, conseguimos obter uma gestão fácil e intuitiva, recorrendo a funções como *is_highscore()*, *insert_highscore()*, *cancel_highscore()* e *edit_name()*. Em relação à parte gráfica, as funções *draw_hs()* e *draw_score()* tratam de organizar onde e qual a informação que deve ser escrita no ecrã. Todo este módulo foi implementado por Carlos Albuquerque.

3.12 Módulo Loop(10%)

Este módulo é responsável pela a organização do programa. Com efeito, contém as unicas 3 funções chamadas pelo *proj_main_loop()*: *game_start()* responsável por subscrever os periféricos todos, configurá-los com as definições apropriadas e inicializar os objetos usados na lógica e por o minix em modo gráfico; *game_loop()* que contém o ***driver_receive()***, sendo portanto responsável por chamar os interrupt handlers dos periféricos e chamar as funções de lógica apropriadas para processar os eventos gerados; *game_end()* que liberta toda a memória usada para os objetos no *game_start()*, anula as subscrições e volta a pôr o minix em modo texto. Ambos os membros foram alterando as três funções sempre que necessário, podendo assim admitir-se uma participação igual em todas as funções deste módulo.



3.13 Módulo Sprites(5%)

Este módulo é responsável pelas operações relacionadas com sprites. Desde criar e desenhar a destruir e eliminar, tudo é tratado por funções deste módulo. *draw_sprite()* e *destroy_sprite()* são largamente usadas para movimentar o cursor no ecrã e desenhar. A função *draw_sprite()* é também muito importante pois é ela que permite desenhar todos os elementos gráficos no ecrã. Esta biblioteca não é da nossa autoria, foi implementada pelos professores da UC. Contudo, como não contemplava o uso de xpm's em modo de endereçamento direto, Paulo Marques desenvolveu a função *create_sprite_direct()* para carregar uma xpm em modo direto para a sprite e Carlos Albuquerque adaptou a função *draw_sprite()* para que as cores fossem corretamente lidas da xpm.

3.14 Módulo XPM(1%)

Este módulo é responsável por incluir no projeto toda a biblioteca de imagens, bem como dar acesso a arrays globais e estáticos (*numbers*, *letters*, *counters* e *wins*) que permitem, entre outros, escolher dinamicamente o sprite que deve ser escrito no temporizador do jogo a partir do tempo restante e circular por todas as letras do alfabeto ao escolher o nome da classificação no menu highscores. Toda a biblioteca de imagens foi criada por Carlos Albuquerque, com base numa *font* e em imagens retiradas da internet e criadas por ele. As imagens *background.xpm* e toda a sequência *coin.xpm* são usadas tal e qual como encontradas na internet.

3.15 Módulo AnimSprite(5%)

Este módulo é responsável pela animação de sprites através de uma sequência de imagens. É apenas usado para animar a moeda no canto inferior esquerdo do menu principal. Este módulo foi maioritariamente desenvolvido por Paulo Marques. Carlos Albuquerque fez apenas alterações às funções *draw_Anim_Sprite()* e criou a função *draw_current_Sprite()*.

4-Detalhes de Implementação

O nosso projeto é baseado numa filosofia de design que pode ser **separada em 3 camadas**, de modo ao código ficar mais bem estruturado, organizado e também facilitar a construção do projeto. Uma camada é constituída por funções de mais baixo nível, de interação com os periféricos. Outra são funções de alto nível

que lidam com eventos no jogo, e uma outra que traduz as interrupções de baixo nível em informação útil às de alto nível(módulo de lógica e o módulo loop).

Podemos assim afirmar que o todo o código do projeto é **orientado a eventos**. Com efeito, qualquer interrupção é traduzida para um evento através duma função específica do periférico, e é enviada para a lógica. Aí é decido o impacto da interrupção no programa. Na lógica do jogo utilizamos uma **máquina de estados** para saber o contexto em que as interrupções são geradas e como tal as suas consequência no estado do programa. Com este tipo de implementação, separamos o tipo de código de acordo com o nível de abstração.

O nosso jogo é também influenciado pela programação **orientada a objetos**, sendo a maioria dos elementos do jogo uma classe. Todos os objetos têm construtor, destrutor, os métodos get e set necessários e elementos para o desenho dos mesmos. Algumas classes usadas são por exemplo: Menu, Button, HighScores, Player...

Relativamente à **geração de frames**, todos os eventos são gerados na ocorrência duma interrupção sendo tratados após a sua geração. Quando os eventos implicam desenhar algo no ecrã estes são primeiramente desenhados num buffer auxiliar. A cada tick do timer ocorre um **page flip** do buffer auxiliar para a memória de vídeo principal, usando assim **double buffering**. Também a cada tick do timer é actualizado os elementos do jogo, como a nova posição dos jogadores.

No nosso jogo também há de ter atenção há verificação de **ocorrência de colisões**, entre o cursor do rato e os botões no menu, que entre as mota e os seus rastos. No primeiro caso é avaliado se a sprite do rato fica sobreposta à de um botão. Caso isto se verifique o botão fica num estado de “hovered” mudando assim o seu aspeto. Já o segundo caso é essencial para as condições de vitória no jogo. Com efeito se uma mota colide com a borda da área de jogo ou com um rasto de outro jogador, o jogador correspondente da outra mota ganha um ponto. No caso de estes colidirem de frente ou ao mesmo tempo, é considerado um empate.

Temos também uma **sprite animada** no menu inicial, para o utilizador ser obrigado a introduzir moeda, “simulando” os jogos de arcade antigos no qual o jogo é inspirado.

O menu de highscores é ligeiramente complexo por algumas razões: a edição do nome é feita mesmo no menu, assim que é adicionado a nova classificação a este, e a rotação entre as retas não tem limite, ou seja é possível passar do Z para o A e do A para o Z sem qualquer problema, tendo que carregar dinamicamente cada uma das letras de acordo com as alterações feitas pelo utilizador. Para além disso, o utilizador pode escolher não registar a classificação carregando na tecla ESC.

Note-se que a classificação é automaticamente introduzida e é retirada só após o utilizador carregar ESC. No caso de a tabela estar cheia, o último highscore anteriormente presente será eliminado da tabela. Se o utilizador decidir confirmar a inserção do novo highscore então o eliminado desaparece de vez. Contudo, se o utilizador cancelar a inserção do novo highscore a tabela volta ao estado inicial e o elemento previamente eliminado volta a aparecer na tabela. Para além disso, graças à implementação da variável global estática *last_index*, a função *is_highscore()* percorre uma vez o array de scores e para onde tiver que ser inserida a nova classificação, guardando esse índice em *last_index* para que a função *insert_highscore()* não tenha que voltar a percorrer todo o array e se foque apenas em mover todas as classificações para o seu devido lugar, melhorando assim a eficiência do programa que não tem de repetir o acesso ao array duas vezes. Sempre que o highscore é confirmado ou cancelado, a variável *last_index* passa a -1, valor verificado pela função *valid_hs_op()* que é invocada sempre que se tenta fazer uma alteração ao menu de highscores, não permitindo que nada aconteça e poupando tempo e memória do programa que não tem de estar a alocar dinamicamente letras caso não seja possível editar o nome das classificações.

Para os highscores implementamos também um **font** para mostrar os resultados.

Para estabelecer a comunicação entre dois computadores, utilizamos o UART. Quando um jogador carrega num botão de remote, o computador desse jogador envia ao outro um byte (PROT_CONNECT) que faz com que a lógica do outro computador ative uma flag significando que o jogador caso carregue em remote passa a ser o jogador 2. Com efeito, o jogador que enviou o byte é o jogador 1.

Como ambos os jogos são independentes do outro o nosso protocolo de comunicação necessitava sobretudo de ser rápido evitando a dessincronização. Com efeito, tentamos enviar o mínimo de informação possível, para que esta seja processada rapidamente e demore o mínimo de tempo possível a transmitir. No entanto, isto não permitiu ao protocolo o *acknowledgement* da informação recebida estando sujeito a alguns erros. Depois de alguns testes, como os erros não se manifestaram decidimos manter o protocolo assim.

Também optamos pela utilização de **FIFOs** para a receção no UART. Desta forma evitamos que um byte recebido por um jogador seja substituído por outro evitando assim a perda de informação. Para ser enviada uma interrupção basta um byte estar nas fifos para que estes sejam lidos o mais rapidamente possível. Usamos polling, pois este método gerou melhores resultados ao evitar a dessincronização.

A nível gráfico tentamos manter um design relativamente simples, inspirado no jogo arcade em que o nosso jogo se basia.

5-Conclusões

Em suma, a cadeira de Icom foi a cadeiras mais difícil e trabalhosa que tivemos no nosso percurso académico. No entanto, a realização do projeto foi uma experiência divertida e satisfatória.

Por um lado, há uns aspectos que gostaríamos de chamar a atenção para:

- O serial port é sem dúvida o periférico mais complicado, apesar de não ser objeto de avaliação obrigatória, gostaríamos que para o próximo ano pudesse ser disponibilizado uns testes para verificar a sua implementação, tal como tivemos nos labs ao longo do semestre.
- Os avisos no fórum do moodle deveriam ser evitados. Embora seja nossa obrigação verificar os nossos e-mails, recebemos tantos que os realmente importantes ficam perdidos no meio do spam. Desta forma sugerimos que todos os avisos sejam enviados por email geral, para ser mais fácil distinguir das dúvidas lançadas no moodle.
- Também gostaríamos de ter mais feedback nos labs. Com efeito, o facto de não termos as notas dos labs antes do projeto não nos diz se o que fizemos até à altura é o mais correto, podendo um erro dado no início do semestre propagar-se até ao projeto.
- A falta de informação como desenhar o projeto. Uma das maiores dificuldade sentidas foi onde começar desta forma gostaríamos que fosse lecionado como fazer um game engine, ou os básicos de um (visto que a maioria dos projetos são jogos).
-

Por outro lado:

- Os labs preparam-nos bem para o projeto final. Com efeito, as dúvidas sobre periféricos no projeto foram muito reduzidas já que os labs os exploram muito.
- As aulas sobre máquinas de estado e outros tópicos não relacionados diretamente com periféricos foram muito úteis para o desenvolvimento do projeto

Gostaríamos também de louvar e agradecer a ajuda dada pelo nosso professor das práticas assim como a do monitor lá presente.