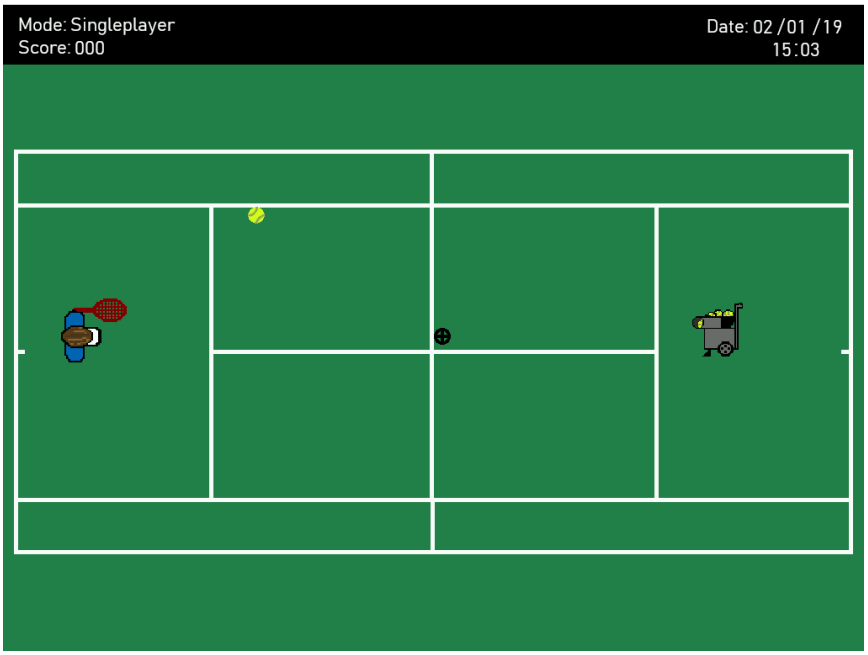
Faculdade de Engenharia da Universidade do Porto

"2D Tennis - Minix Edition"



Laboratório de Computadores (LCOM) – 2018/2019

Turma 2 – Grupo 5

Autores: Liliana Natacha Nogueira Almeida – up201706908

José Miguel Martins Gomes – up201707054

7 de janeiro de 2019

Índice

1. Introdução

Este projeto, com o título “2D Tennis – Minix Edition”, foi realizado no âmbito da unidade curricular Laboratório de Computadores com o intuito de aplicar de forma mais elaborada e complexa os vários módulos lecionados ao longo do semestre.

O jogo consiste em jogar ténis numa plataforma 2D utilizando o teclado e o rato. O desenvolvimento deste projeto foi feito tendo em conta os objetivos da unidade curricular que incluem:

- Utilizar as interfaces de hardware mais comuns em periféricos de computador
- Desenvolver software de baixo nível
- Programar com a linguagem C
- Utilizar variados software de desenvolvimento

2. Instruções de utilização

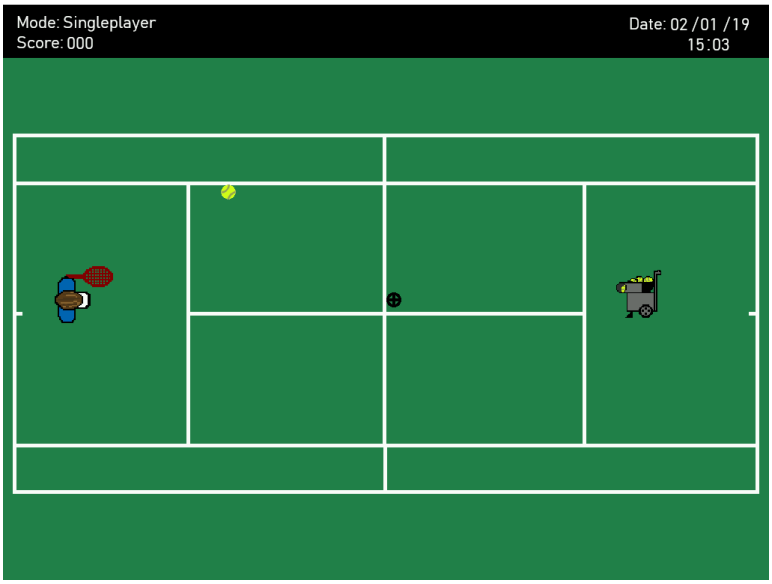
2.1. Menu Inicial

Quando o programa é iniciado é apresentado o menu inicial:

Neste ecrã é possível selecionar as várias opções com as teclas W, S ou as setas de direção vertical e selecionar com a tecla ENTER.

2.2. Singleplayer

Se a opção selecionada for a de "Singleplayer" será iniciado o jogo.



O modo *singleplayer* consiste em bater o máximo de bolas possível. As bolas são atiradas pela Máquina Atira Bolas em intervalos fixos e a velocidade com que são atiradas aumenta gradualmente. O jogador é movido com o teclado através das teclas W, A, S, D ou das setas de direção, sendo que se a bola se encontrar à direita do jogador este tem a raquete na mão direita, se não tem-na na mão esquerda. Com a tecla Esc é possível

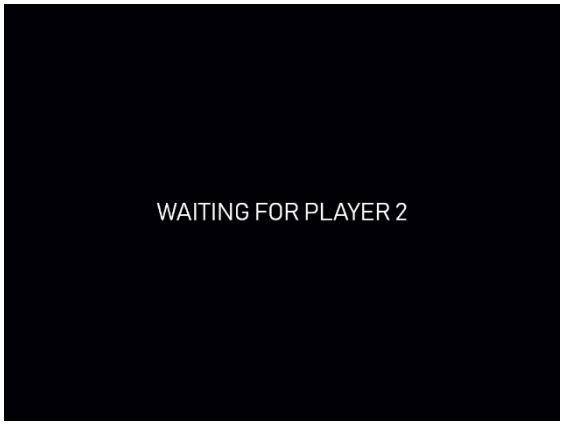
retornar ao menu principal, perdendo todo o progresso realizado até ao momento. A direção para onde a bola é atirada depende da posição do rato indicado pelo cursor preto. Para uma "batida" ser válida e conseguir ganhar um ponto, o jogador tem de clicar no lado esquerdo do rato quando a posição da bola coincidir com a posição da raquete vermelha e a posição do rato estiver contida nas linhas interiores do campo do adversário. Se isto não acontecer, o jogador perde o jogo imediatamente e retorna ao menu principal. A pontuação obtida é guardada num ficheiro que se encontra em "/proj/src/" e as melhores pontuações podem ser revistas no ecrã de *Highscores*.

2.3. Menu Multiplayer

Se a opção selecionada for a de "Multiplayer" será apresentado o Menu *Multiplayer* seguinte:

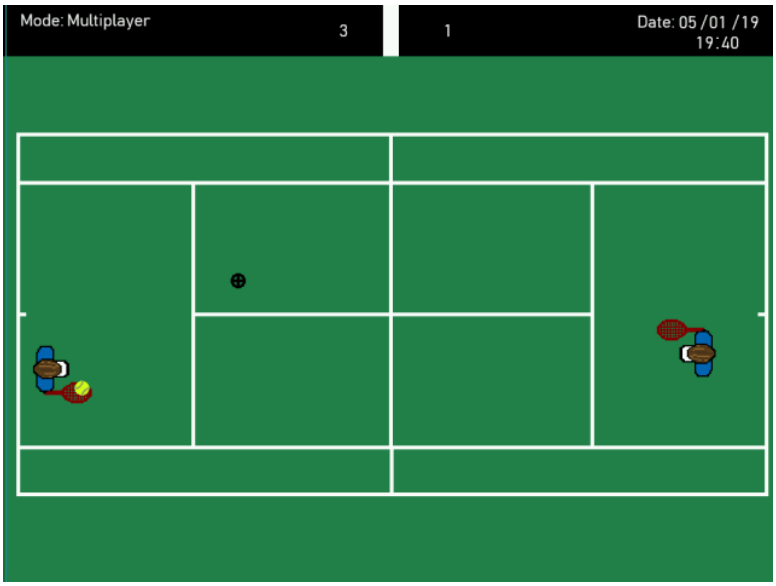
Neste menu é possível selecionar se se pretende ser o *Player* 1 ou o *Player* 2 ou se pretende regressar ao Menu inicial com a opção Exit.

Consoante a opção escolhida (*Player* 1 ou 2) é apresentado um ecrã de espera que garante a sincronização dos dois jogadores através da *serial port*.



2.4. Multiplayer

Após os dois jogadores estarem conectados o jogo entra em modo *multiplayer*. Neste modo, dois jogadores competem um contra o outro, ganhando um ponto da mesma forma que no modo *singleplayer*, exceto que o cursor do rato não tem de estar contido nas linhas interiores do campo do adversário, mas sim numa posição que leve a bola a sair da janela na metade do campo adversário, caso contrário perde a jogada e o outro jogador ganha um ponto. O primeiro a marcar 5 pontos ganha o jogo.



2.5. Highscores

No ecrã de *Highscores* é possível consultar as 4 maiores pontuações bem como a data de quando foram obtidas, lidos a partir do ficheiro "Highscores.txt". Para sair deste ecrã e regressar ao menu inicial basta usar a tecla Esc.

3. Periféricos

Os periféricos implementados foram os seguintes:

Dispositivo	Função no projeto	Interrupções
Timer	Controlo do *frame* rate. Gerir eventos do jogo.	Sim
Teclado	Movimento do *player* e interação com menus.	Sim
Rato	Controlar trajetória da bola e dar tacada na bola.	Sim
Placa Gráfica	Visualização do ecrã de jogo e dos menus.	Não
Real Time Clock	Atualização da data e hora durante os diferentes modos de jogo.	Sim
Serial Port	Comunicação e sincronização entre os dois jogadores, no modo *multiplayer*.	Apenas na receção de dados

3.1. Timer

As interrupções do Timer são utilizadas para redesenhar os objetos nas suas novas posições, 60 vezes por segundo. A cada interrupção, são verificadas também todas as condições que possam levar à realização de eventos, como as colisões ou se o jogo terminou. Tudo isto é tratado nas funções "singlePlayerGame", "gamePlayer1" e "gamePlayer2" do ficheiro **game.c**.

No ficheiro **serial_port.c** são também utilizadas as interrupções do Timer, mais especificamente nas funções "waitingPlayer1" e "waitingPlayer2", para enviar a informação de sincronização a cada interrupção, enquanto os dois *players* esperam que comece o jogo, no modo *multiplayer*.

As funções do Timer encontram-se implementadas nos ficheiros **timer.c**, estando as *macros* utilizadas no ficheiro **i8254.h**.

3.2. Teclado

O teclado é utilizado para controlar o movimento do *player* e para interagir com os menus.

Nos modos de jogo, a cada interrupção é lido o *makecode* ou o *breakcode* de uma tecla. Se for o *makecode* das teclas W, A, S, D ou das setas de direção é iniciado o movimento do *player* na direção indicada e este movimento só para quando atingir um limite da área de jogo ou quando receber o *breakcode* da direção em questão. Esta implementação permitiu um movimento mais suave e controlado do *player*, sendo realizada na função "set_move" do ficheiro **spriteMovement.c**. As interrupções são detetadas pelas funções do ficheiro **game.c**. Nestas funções é ainda possível, com a tecla "ESC", sair do jogo e retornar ao menu principal. Esta última funcionalidade também é

possível encontrar na função “highscoreScreen”, do ficheiro **pointSystem.c**, e nas funções “waitingPlayer1” e “waitingPlayer2”, do ficheiro **serial_port.c**, retornando, a primeira, ao menu principal e as últimas ao menu *multiplayer*.

No ficheiro **menus.c**, as funções “startMenu” e “multiPlayerSelect” permitem, que a cada interrupção, as teclas lidas possibilitem a navegação pelo menu e a seleção de uma das opções disponíveis.

As funções do teclado estão implementadas nos ficheiros **keyboard.c**, exceto a função em *assembly* “kbc_asm_ih” no ficheiro **kbd_Assembly.S**, estando as *macros* utilizadas no ficheiro **i8042.h**.

3.3. Rato

O rato é apenas utilizado nos modos de jogo para mover um cursor que dita a trajetória da bola e para dar a “batida” na bola quando há um clique no botão esquerdo. As funções que tratam esta informação a cada interrupção são a “singlePlayerGame”, a “gamePlayer1” e a “gamePlayer2” do ficheiro **game.c**

Os ficheiros **menus.c** e **serial_port.c** contêm também funções para detetar interrupções do rato, mas apenas para limpar o output buffer e não interferir com as interrupções do teclado.

As funções do rato estão implementadas no ficheiro **mouse.c**, estando as *macros* utilizadas no ficheiro **i8042.h**.

3.4. Placa Gráfica

A placa gráfica é utilizada para apresentar toda a informação gráfica do jogo. O modo gráfico utilizado foi o 0x117 com resolução 1024x768, 16bits por pixel e com modo de cores RGB(5:6:5) que possui 2^{16} cores (sem informação de transparência).

Para carregar e apresentar ficheiros do tipo bitmap é utilizada a biblioteca bmp de Henrique Ferrolho da qual fazem parte os ficheiros **bitmap.c** e **bitmap.h**. Todas as imagens são da nossa autoria. Por outro lado, utilizámos a fonte “Bahnschrift Light”, para imprimir o *score* e a data com dois tamanhos diferentes, um nos modos de jogo, realizado pelas funções “printPoints” e “print_date” e outro para mostrar os *highscores*, implementado pela função “readHighscores”, sendo que todas estas funções pertencem ao ficheiro **pointSystem.c**.

Está implementado um sistema de *double buffering*, no ficheiro **video_gr.c**, que permite uma troca de *frames* mais suave. Para auxiliar esta funcionalidade, implementamos a função “doubleBuffCall” no ficheiro **video_gr.c**, que copia para o “video_mem” tudo o que foi desenhado no outro buffer criado.

Implementámos a função “drawSprite”, em **bitmap.c**, que possibilita a utilização de determinada cor como transparente (0xF81F no nosso caso) e verifica não só a colisão dos *sprites* com os limites do ecrã, mas também a colisão da bola com a raquete “pixel a pixel” através da cor desta. Esta função permite facilmente a colisão com objetos de forma irregular. A única cor utilizada para a colisão “pixel a pixel” é o encarnado da raquete 0x8000.

As funções relacionadas com a placa gráfica estão implementadas nos ficheiros **bitmap.c** e **video_gr.c**, estando as *macros* utilizadas no ficheiro **vbe_macros.h**.

3.5. Real Time Clock

O *Real Time Clock* (RTC) é utilizado para aceder e visualizar a data e a hora utilizada durante o jogo em si e a data guardada em conjunto com os highscores. Nas três funções do ficheiro **game.c**, são usadas interrupções do tipo *Update-ended* que permitem atualizar a data e hora, no final de cada ciclo de atualização realizado pelo RTC, com o auxílio das funções em *assembly* definidas no ficheiro **rtc_Assembly.S**. A função “rtc_ih_asm” lê o registo C para limpar interrupções pendentes e as restantes funções retornam os vários elementos da data e hora necessários. No caso da data e da hora estar em decimal, é ainda feita a conversão para binário, na função “update_date” do ficheiro **pointSystem.c**, para facilitar a impressão destas no visor.

As funções relacionadas com o RTC estão implementadas nos ficheiros **rtc.c**, estando as *macros* utilizadas no ficheiro **rtc_macros.h**.

3.6. Serial Port

A *Serial Port* é utilizada para enviar informação das teclas premidas pelo adversário ou a informação das componentes da velocidade da bola a cada batida. A função responsável por enviar as teclas premidas e também de processar as recebidas é a “set_move” do ficheiro **spriteMovement.c**.

No que diz respeito ao envio da velocidade da bola, a informação é “empacotada” em 16 bits utilizando operações bit a bit. A informação é enviada e recebida nas funções “gamePlayer1” e “gamePlayer2” do ficheiro **game.c** sendo a informação recebida tratada na função “remoteMoveBall” do ficheiro **spriteMovement.c**.

4. Organização do Código

4.1. bitmap.c

Este ficheiro contém as funções que permitem carregar, desenhar e apagar bitmaps. O código é da autoria de Henrique Ferrolho [1] exceto as funções "createSprite" e "drawSprite" que foram adaptadas das funções existentes para melhor satisfazer as nossas necessidades nomeadamente a transparência e a colisão. Para além das estruturas de dados implementadas pelo autor original, adicionamos a *struct* "Sprite" que facilita o movimento e a animação das imagens e a *struct* "Movement" que permite um movimento mais suave.

A fonte do código é: http://difusal.blogspot.com/2014/09/minixtutorial-8-loading-bmp-images.html

Peso: 6%

Aluno responsável: José Gomes e Liliana Almeida (50%/50%)

4.2. game.c

Este ficheiro contém apenas a lógica do jogo Singleplayer e Multiplayer e chama as funções necessárias para a implementação dessa lógica consoante as interrupções que recebe.

Peso: 15%

Aluno responsável: José Gomes e Liliana Almeida (60%/40%)

4.3. initGame.c

Neste ficheiro estão as funções que realizam a subscrição e fazem o *unsubscribe* global de todos os periféricos utilizados no projeto e todos os *getters* para os IRQ's dos respetivos periféricos.

Peso: 2%

Aluno responsável: Liliana Almeida

4.4. keyboard.c e kbd_Assembly.S

Este ficheiro contém as funções desenvolvidas no trabalho laboratorial sobre o teclado, tendo no ficheiro de *assembly* o *handler* também desenvolvido neste *lab*.

Peso: 10%

Aluno responsável: José Gomes e Liliana Almeida (40%/60%)

[1] Ferrolho, H. {2014} http://difusal.blogspot.com/2014/09/minixtutorial-8-loading-bmp-images.html, consultado em 10/12/18.

4.5. menus.c

Neste ficheiro estão implementados o Menu Inicial e o Menu Multiplayer. Ambos os menus estão baseados em s*tate machines (menu_options e multiplayer_options)* que indicam qual das opções está selecionada e o que fazer se essa opção for selecionada.

Peso: 5%

Aluno responsável: José Gomes e Liliana Almeida (40%/60%)

4.6. mouse.c

Este ficheiro contém funções desenvolvidas no trabalho laboratorial sobre o mouse, tendo sido adicionadas funções para ajudar no movimento do cursor do rato e a saber se o botão esquerdo está premido, no modo de jogo.

Peso: 6%

Aluno responsável: José Gomes e Liliana Almeida (50%/50%)

4.7. spriteMovement.c

Aqui estão implementadas todas as funções que dizem respeito ao movimento do *player* e da bola. O sistema de movimento do *player* é baseado na *struct* "Movement" do módulo bitmap.c. Este sistema permite que haja movimento em duas direções simultaneamente e assim é possível andar na diagonal. Para além disso, o movimento deixa de estar dependente da velocidade do PC a enviar *makecodes* sucessivos.

O movimento da bola também é calculado aqui, tendo em consideração a posição atual da bola e a posição do cursor.

Peso: 15%

Aluno responsável: José Gomes e Liliana Almeida (70%/30%)

4.8. pointSystem.c

Neste ficheiro estão todas as funções relacionadas com a validação de uma "batida" da bola, registo de pontuação e visualização da pontuação e da data/hora, utilizando-se para a última funcionalidade funções que permitem ler e escrever os *highscores* com a data correspondente num ficheiro.

Peso: 6%

Aluno responsável: José Gomes e Liliana Almeida (60%/40%)

4.9. proj.c

Este ficheiro apenas chama o módulo “initGame.c” e o “menus.c” que tratam de todo o projeto.

Peso: 1%

Aluno responsável: José Gomes e Liliana Almeida (50%/50%)

4.10. rtc.c e rtc_Assembly.h

No ficheiro em C, encontram-se as funções que permitem subscrever e desativar as interrupções do *Real Time Clock*, enquanto que o ficheiro de *assembly* contém a função “rtc_ih_asm” que lê o registo C para limpar possíveis interrupções pendentes e permitir novas interrupções para atualizar a data e a hora. As restantes funções neste ficheiro, possibilitam o acesso aos registos do RTC necessários com a data e a hora.

Peso: 5%

Aluno responsável: Liliana Almeida

4.11. serial_port.c

Este módulo contém todas as funções que permitem a comunicação entre os dois jogadores.

Peso: 9%

Aluno responsável: José Gomes

4.12. timer.c

Este ficheiro contém as funções desenvolvidas no trabalho laboratorial sobre o timer.

Peso: 10%

Aluno responsável: José Gomes e Liliana Almeida (50%/50%)

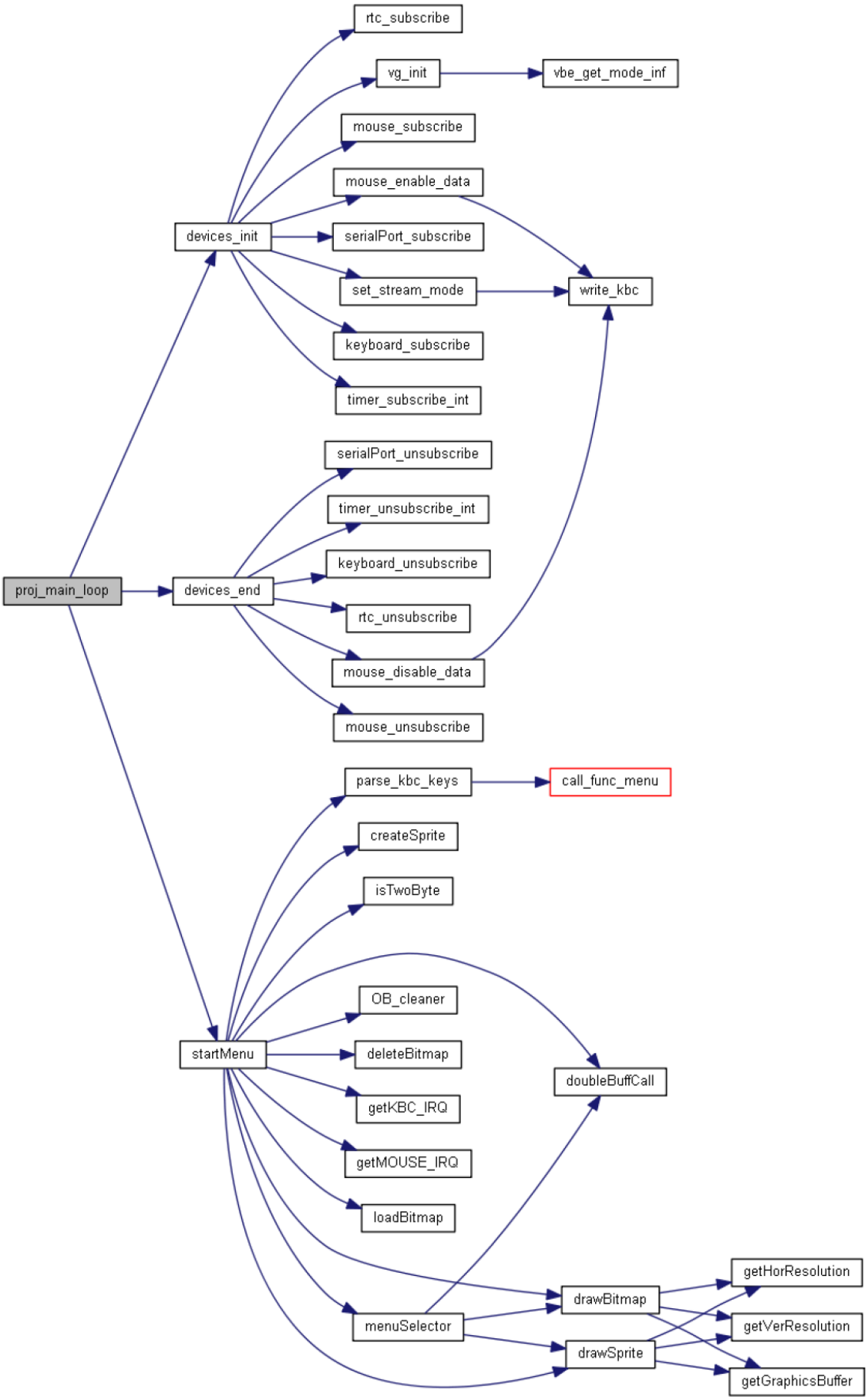
4.13. video_gr.c

Neste ficheiro estão todas as funções que permitem a utilização da placa gráfica e o *Double Buffering*. A maioria das funções foram desenvolvidas durante o trabalho laboratorial sobre a placa gráfica, todavia foram alteradas para satisfazer as necessidades do projeto.

Peso: 10%

Aluno responsável: José Gomes e Liliana Almeida (50%/50%)

5. Gráfico da chamada de funções



6. Detalhes de implementação

6.1. Serial Port

Ao desenvolver o projeto deparámo-nos com bastantes problemas no que diz respeito à implementação do *Serial Port*. O nosso plano inicial de ter dois jogadores, a competir um contra o outro, não foi conseguido como esperávamos devido à desconhecida baixa velocidade do *Serial Port*.

Estamos cientes da existência de FIFO's e da sua possível implementação, mas uma vez que não abordámos este tema nas aulas práticas a sua implementação tornou-se bastante complexa. Apesar de várias tentativas, a aproximação da data de entrega invalidou a implementação de FIFO's e continuámos o projeto sem esta funcionalidade.

Contudo, conseguimos implementar uma versão "jogável" do modo *multiplayer*, ainda que com um *delay* elevado e, por vezes, com uma discrepância considerável entre os dois jogadores.

6.2. RTC

Este tema foi abordado no nosso trabalho para que fosse possível ao jogador ter acesso à data e às horas enquanto usufrui do jogo. Como já referido em pontos anteriores, os vários elementos utilizados (dia, mês, ano, hora e minutos) são atualizados, cada vez que o RTC atualiza os seus registos com estes elementos, usando por isso interrupções *Update-ended*. Através deste mecanismo, garante-se que quando se lê a data e as horas dos registos, estes não estão a ser atualizados e, por isso, não levam a erros de leitura. Para que este método seja completamente implementado é ainda necessário cada vez que se pretende ler os registos do RTC, ler também o registo C do mesmo periférico, pois só assim é que o bit correspondente ao IRQF é colocado a 0, assim como todas as *flags* ativas deste registo, tornando possível novas interrupções por parte do RTC. O bit IRQF do registo C indica se a linha IRQ está ativa, ou seja, se existe uma interrupção pendente.

Neste periférico, recorremos ao uso de *assembly* para aceder aos registos com a data e a hora e também para ler o registo C a cada interrupção. Procedemos desta forma, pois o *assembly* torna-se mais eficiente no caso de operações que ocorram muitas vezes, uma vez que não está dependente do compilador e permite controlar tudo o que é feito em código e lidar diretamente com os endereços de memória, levando a um melhor desempenho(também foi abordado este procedimento no handler do teclado pelas mesmas razões, uma vez que é utilizado numerosas vezes nos menus e nos modos de jogo)

6.3. *State Machines*

No projeto foram utilizadas *state machines*, implementadas para o menu principal (*menu_options*) e para o menu de *multiplayer* (*multiplayer_options*), em estruturas do

tipo enumeração. Estas permitem que, ao receber uma das teclas W, S ou uma das setas de direção vertical, seja possível navegar pelos menus. Isto porque a cada interrupção do teclado, se o *scanbyte* recebido corresponder a uma das teclas referidas acima, a enumeração é percorrida conforme a direção fornecida pela tecla, sendo que se se encontrar num dos limites, isto é, na primeira opção ou na última opção, as *state machines* executam a passagem para o outro limite. Quando a tecla recebida é o *enter*, tendo em conta em que estado se encontra a enumeração correspondente ao menu atual, é chamada a função respetiva que irá dar continuação ao funcionamento do programa.

6.4. Deteção de colisão

Este tema não abordado nas aulas foi um pouco mais complexo do que esperávamos. A razão desta constatação é o facto de que quando ocorria a primeira colisão entre a bola e a raquete do jogador, apesar de a bola andar um pouco noutra direção, a colisão entre os dois *sprites* voltava a acontecer. Para contornar esse problema, adicionamos à *struct* Sprite, que representa as imagens com transparência, dois membros *bool*: "colided" e "canColide". Apesar de característicos a todos os sprites que criámos, só nos interessa e só são verificados os que pertencem à bola.

O primeiro membro "colided", inicializado a *false*, declara se ocorre colisão entre um *sprite* e a cor da raquete a cada *frame* que o *sprite* é desenhado, passando a *true* se isso ocorrer. Esta colisão é verificada aquando do processo de desenho da s*prite* no segundo buffer (que não o "video_mem"), comparando a cor da imagem do *sprite* (ignorando a cor da transparência) a desenhar em cada pixel com a cor que já lá se encontra e que, portanto, pertence à *frame* anterior. A cada *frame* desse *sprite*, a variável "colided" é reinicializada a *false*.

O membro "canColide" é inicializado a *true* e é alterado para *false* quando ocorre uma colisão como descrita acima e esta é válida (quando o botão esquerdo do rato é premido num momento em que "colided" é verdadeiro), para assim não permitir novamente outra colisão até instrução em contrário. Só quando a bola sai da janela do ecrã é que este membro é reinicializado a *true*. No caso do modo *multiplayer*, esta alteração também acontece quando recebe a informação de que o jogador adversário deu uma "batida" na bola.

7. Conclusão

Com a realização deste trabalho sentimos que efetivamente os objetivos da unidade curricular foram cumpridos e aprendemos de facto a utilizar interfaces hardware, desenvolver software de baixo-nível e programar com a linguagem C.

Consideramos que obtivemos conhecimentos e adquirimos competências importantes para o desenvolvimento do curso, contudo, não achamos que o método de ensino seja o mais adequado/adaptado, uma vez que a carga horária necessária à realização de todos os *labs* e do projeto em si é demasiada, quando temos em conta as unidades de crédito (ECTS) atribuídos à unidade curricular.

Outro ponto é a ordem por qual são abordados os periféricos nas aulas práticas. Sendo os três periféricos obrigatórios para o projeto o Timer, o *Keyboard* e a Placa Gráfica achamos que faria mais sentido abordar a Placa Gráfica antes do mouse. Isso facilitaria a compreensão das funcionalidades da placa gráfica aquando da Project *Proposal*.

Também gostávamos de mencionar o excesso de "*self-learning*" que a unidade curricular exige mesmo durante as aulas práticas. Há muita falta de informação sobre como realizar os *labs* e o professor da aula prática não consegue muitas vezes "socorrer" todos os estudantes que precisam de ajuda, aumentando ainda mais a carga horária necessária.

Referimos a dificuldade do Teste de Programação realizado, cuja nota mínima para aprovação foi de 8 valores e o facto de que, para um estudante que reprove, a nota máxima no exame recurso seja de 8 valores.

Percebemos a necessidade da realização deste teste para rastrear os estudantes, mas consideramos que o grau de dificuldade do mesmo foi elevado na medida em que o teste vale apenas 10% da nota final.

Em jeito de conclusão, consideramos que a Unidade Curricular de Laboratório de Computadores é bastante apelativa na sua teoria e consegue ser também interessante na prática, mas o nível de dificuldade e a elevada carga horária desencorajam os estudantes.

Apêndice

Instruções de instalação

Admitindo que o utilizador se encontra na *root* do projeto, é necessário executar o seguinte comando:

```
lcom_conf add conf/proj
```

Esta instrução permite que o programa possa aceder a funções privilegiadas, no caso de o utilizador não ter essas permissões.

De seguida é necessário realizar o seguinte comando, onde se encontram os ficheiros de código e após o qual é possível realizar a instrução `make`:

```
cd src/
```

Instruções para correr programa

Depois de compilar o programa, no diretório resultante do último comando referido, o passo seguinte é correr o programa que é possível da seguinte forma:

```
lcom_run proj "<diretório da pasta com as imagens>"
```

O argumento é utilizado de forma a não usar *hard coded paths* para aceder às imagens *bitmap* que podem não se encontrar num local fixo. Desta forma, se, por exemplo, as imagens se encontrarem numa pasta *res* na *root* do projeto, o comando a executar é:

```
lcom_run proj "/home/lcom/labs/proj/res"
```