Laboratório de Computadores

Relatório

Turma 4 - Grupo 06

Ana Margarida Ruivo Loureiro
João Miguel Ribeiro de Castro Silva Martins

1.Índice

2.Instruções do jogo

O programa inicia num menu onde é possível escolher diferentes opções com o uso do rato - Play, Credits e Quit - sendo que quando se passa o rato por uma das opções, esta fica rodeada por um retângulo.

2.1 Play

Na opção Play, o jogo é iniciado.

O jogador deve utilizar as teclas A e D para deslocar o tetraminó para a esquerda e direita, respectivamente, e a tecla Space para efetuar rotações de 90 graus às peças. Além disso, a tecla S permite a peça descer com mais velocidade. Para sair a meio do jogo e retornar ao menu principal, prima a tecla “ESC”.

O jogo apresenta a hora atual no canto superior esquerdo do ecrã e a pontuação e nível atual no canto superior direito.

É de referir que o jogo possui cinco níveis, sendo que a mudança de cada nível corresponde a um incremento na velocidade das peças. A mudança de níveis efetua-se aos 100 ,200, 500 e 1000 pontos.

A pontuação é atribuída da seguinte forma:

- Uma linha eliminada: 10 pontos;
- Duas linhas eliminadas simultaneamente: 30 pontos
- Três linhas eliminadas simultaneamente: 40 pontos
- Quatro linhas eliminadas simultaneamente: 60 pontos

Além disso, quando o jogador perde o jogo é redirecionado para uma página de *Game Over* onde após alguns segundos volta ao menu principal.

2.2 Credits

Nesta opção, o jogador é redirecionado para o ecrã de créditos, onde é possível obter informação acerca dos criadores do jogo.

2.3 Exit

Permite ao jogador sair do jogo.

3.Solução implementada

Timer	Controlo na atualização da parte gráfica Controlo da velocidade das peças; Controlo do tempo com a imagem *Game Over*;	Interrupções
Keyboard	Movimento dos tetraminós; Sair do jogo e de créditos	Interrupções
Mouse	Utilizado no menu principal	Interrupções
Video Card	Visualização no ecrã do jogo, menu e créditos	Não aplicável
RTC	Visualização da hora atual no PC	Interrupções

3.1 Timer

É de destacar a função **timer_int_handler()** que ao incrementar o contador, permite estabelecer o ritmo da movimentação das peças no jogo e para permitir estabelecer o intervalo de visualização de uma imagem no ecrã (utilizado para mostrar o jogo no ecrã e imagem de “Game Over”). Além disso, as funções de subscrição de interrupções **timer_subscribe_int()** e cancelamento de subscrição **timer_unsubscribe_int().**

3.2 Gráfico

O modo gráfico implementado possui 16 bits por pixel - total de 2^{16} cores - com RGB **5:6:5** e resolução de 800x600 pixels. Este é um modo direto e é identificado pelo número 0x114.

No início do jogo, esta componente é inicializada na função **vg_init()**, e no final a função **vg_exit()** liberta a memória alocada para esta componente gráfica.

Para desenhar na parte gráfica é utilizado ***double buffering***. Os pormenores deste mecanismo serão abordados na seção “Detalhes de Implementação”, sendo

implementado nas funções **drawBitmap()**, **paint_game()**, **write_lvl()**, **write_score()** e **draw_squares()**.

Os componentes apresentados no ecrã são Bitmaps, carregados através das funções implementadas no módulo elaborado pelo antigo aluno Henrique Ferrolho, que estará descrito mais pormenorizadamente na secção “Estrutura Implementada”, no módulo Bitmap. Também existem componentes do ecrã que são pintado em componentes de 20x20 pixels de cada vez.

Nos bitmaps foram utilizadas imagens de letras, cuja fonte é *Sans-serif*.

As funções usadas da VBE foram **change_vbe_mode()** e **vbe_get_info_mode()**.

3.3 Keyboard

Periférico utilizado, com recurso a interrupções, com o objetivo de controlar os movimentos no jogo e permitir ao jogador voltar ao menu principal, em caso de querer sair do jogo, dos créditos ou perdendo o jogo.

É assim de referir a utilização das funções necessárias ao seu funcionamento - **kbc_ih()**, **keyb_subscribe_int()** e **keyboard_unsubscribe_int()**.

3.4 Mouse

O *mouse* é utilizado no menu inicial, com recurso a interrupções, manipuladas através da função **mouse_ih()**.

Este funciona com base na posição onde se encontra presente, sendo que o seu cursor provoca a representação de um retângulo colorido a volta das opções no menu ou pelo clique no botão esquerdo, permitindo redirecionar o jogador para os outros estados do jogo.

Também é de referir que o cursor do rato é desenhado no ecrã.

3.5 RTC

O RTC é utilizado ao longo do decorrer do jogo para obter a informação da data e hora atual do computador, com o recurso a interrupções. Para obter a informação atualizada foram utilizadas interrupções do tipo **updated-ended**. Assim sendo são necessárias ao funcionamento as funções **rtc_subscribe_int()**, **rtc_unsubscribe_int()**, **rtc_read_register()**, **rtc_write_register()**, **rtc_enable_update_ints(), rtc_disable_update_ints()** e **rtc_ih()**.

4. Estrutura implementada

Os módulos criados para este programa podem ser divididos em duas secções: módulos relativos ao funcionamento dos periféricos e módulos mais particulares à estrutura do jogo em si. Encontram-se apresentados também alguns grafos de chamada de funções considerados mais relevantes.

4.1 Módulos relativos ao jogo

4.1.1 Módulo Proj

A partir deste módulo é inicializado o modo gráfico. Possui a máquina de estados interna do jogo. Deste modo, a função implementada irá realizar a mudança de estado, conforme o valor retornado em **detect_options_mouse()**, função do menu principal, visto que em todos os casos, à exceção da saída do programa, o *loop* volta sempre a essa função.Trata-se do módulo principal do programa, visto que todos os restantes vão ser ligado a este.

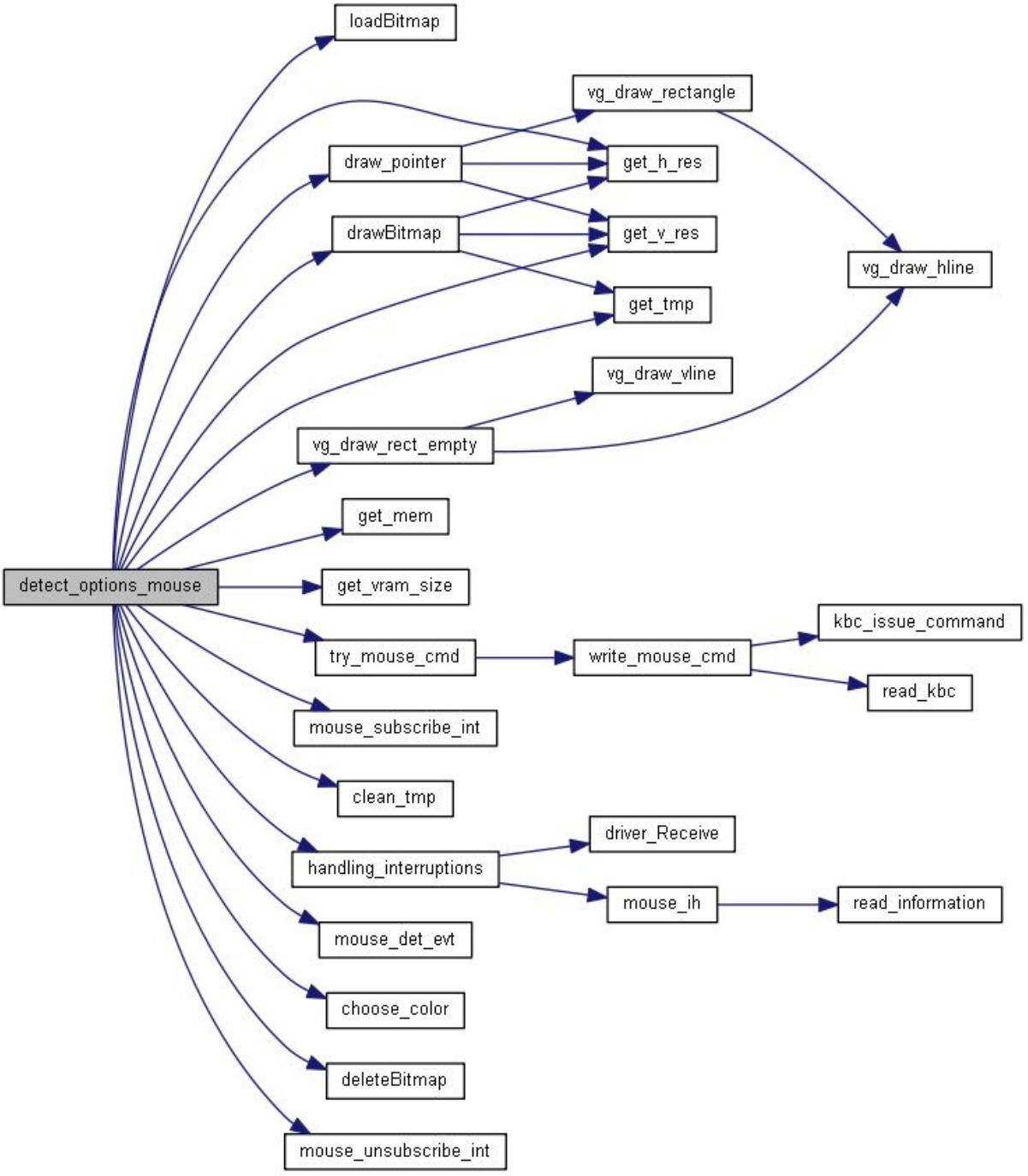
Este módulo foi realizado por ambos os membros.

Peso no projeto: 7%

4.1.2 Módulo main_menu

Este módulo possui um ciclo de interrupções para lidar com a utilização do rato pelo utilizador. A função **detect_options_mouse()** retorna o próximo estado do programa. Este módulo foi realizado por ambos os membros, apesar da contribuição do João Martins ter sido maior.

Peso no projeto: 9%



1.3 Módulo Game

Este é o módulo do jogo em si, pelo que é um módulo principal do programa. Está implementado um ciclo de interrupções onde é feita a manipulação do teclado, *timer* e RTC.

Possui uma função principal **game_main()**, sendo depois auxiliada por **game_movement_options()** que detecta qual a opção de movimento para a tecla

foi selecionada. Adicionalmente, possui funções que permitem escrever no gráfico o tempo real, **draw_date_str()** , atualizar a pontuação ,**write_score()**, o nível, **write_lvL().**

Nestas últimas funções foi utilizada a função **concat()**, de modo a otimizar a estrutura do código, que auxilia na concatenação de duas strings, presente em https://stackoverflow.com/questions/8465006/how-do-i-concatenate-two-strings-in-c?fbclid=IwAR3KqN2_B4Awu0BNk-EcumX-n_a7YSKYu9HoqzR4NgivMn-zFNnOkKRupeg. Esta função aloca a memória necessária para a string final, com base no espaço das duas strings a juntar. Assim, copia a primeira string para a string final com **strcpy()** e por fim junta ambas com **strcat()**, retornado a string final.

Por fim, função **paint_game()** coloca os elementos auxiliares no ecrã e a função **delete_info()** que liberta a memória alocada pelas estruturas que estão presentes quando o jogador termina o jogo.

Para todas estas funções que permitem desenhar na parte gráfica, possuem associados *structs* de apontadores para as respetivas imagens bmp a colocar no ecrã. O grafo de funções deste módulo encontra-se mais à frente, devido à sua extensão.

Este módulo foi realizado por ambos os membros.

Peso no projeto: 10%

4.1.4 Módulo Tetramino

A estrutura interna principal do jogo está armazenada neste módulo, sendo que faz parte desta o **Board** e os **Tetramino**, que são associados a **Squares**. Os detalhes da estrutura interna estão devidamente explicados na seção “Detalhes de Implementação”

Neste módulo estão presentes as funções de inicialização das estruturas internas ao jogo, pelas funções **new_tetramino()** e **new_board()**, sendo que a função **choose_color()** auxilia ao atribuir ao novo tetraminó uma cor de forma aleatória.

Além disso, está presente a função para pintar o *board* no ecrã, **draw_squares()** e várias verificações para o jogo baseadas no tetraminó: se as linhas estão cheias e caso estejam eliminá-las - **check_eliminate_line()**, **eliminate_line()**, **update_board_after_line()** - e os destrutores das estruturas existentes, libertando a memória alocada - **delete_tetramino()** e **delete_board()**.

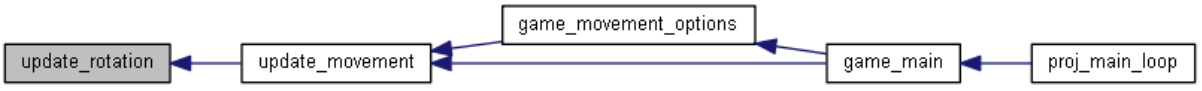
Este módulo foi realizado por ambos os membros.

Peso no projeto: 9%

4.1.5 Módulo update_mov

Este módulo permitem a atualização , ou não, caso não seja possível,dos movimentos dos tetraminó - **update_movement()**, sendo que para isso são feitas verificações de colisões dos tetraminó no jogo, nomeadamente para a atualização do movimento para baixo, para os lados e no caso de rotações, sendo que este último é feito pela função **update_rotation()**. A necessidade da sua implementação decorre de as diferentes formas necessitarem de verificações diferentes. Este módulo foi realizado por ambos os membros, apesar da contribuição da Ana Loureiro ter sido maior.

Peso no projeto: 6%



4.1.6 Módulo Credits

Módulo para a apresentação dos créditos no ecrã, através de uma imagem bmp. Utilizado um ciclo de interrupções para lidar com o teclado, uma vez que para voltar ao menu inicial, o utilizador prime a tecla “ESC”. Isto decorre na função **show_image()**.

Este módulo foi realizado pela Ana Loureiro.

Peso no projeto: 5%

4.1.7 Módulo Bitmap

Este módulo foi desenvolvido por um antigo aluno do MIEIC, Henrique Ferrolho, tendo sido permitida a sua utilização para este projeto. Esta biblioteca pode ser encontrada originalmente em http://difusal.blogspot.pt/2014/09/minixtutorial-8-loading-bmp-images.html.

Foram realizadas pequenas alterações para a sua utilização específica ao nosso programa.

Esta biblioteca é constituída por três funções: **loadBitmap()**, **drawBitmap()** e **deleteBitmap()**.

A função **loadBitmap()** aloca espaço necessário para a imagem bmp e coloca a sua informação no espaço alocado . A informação do espaço necessário é lido de **BitmapInfoHeader** e certifica-se que o tipo de ficheiro é o correto através da leitura

de **BitmapFileHeader**, sendo que estes componentes de informação estão presentes nos ficheiros .bmp.

A função **drawBitmap()** vai desenhar a imagem para o *buffer* temporário - **tmp** - do programa , sendo que a informação é pintada linha a linha para o mesmo. Para isso é usado outro *buffer* temporário (apenas utilizado neste módulo) para colocar a informação da linha antes de ser copiada para **tmp** pela função **memcpy(**).

A última função, **deleteBitmap()**, liberta a memória alocada para a imagem.

Este módulo foi adaptado pela Ana Loureiro.

4.2 Módulos relativos a periféricos

4.2.1 Módulo i8254

Este módulo foi nos fornecido para o desenvolvimento do lab2, sendo necessário para auxiliar manipulações do *timer*.

Peso no projeto: 4%

4.2.2 Módulo i8042

Este módulo foi desenvolvido no decorrer do lab 2 e 3, com o objetivo de auxiliar manipulações do teclado e rato, tendo sido desenvolvido por ambos os membros do grupo.

Peso no projeto: 4%

4.2.3 Módulo KBC

Este módulo foi desenvolvido para os lab 2 e 3 e é utilizado para o controlo do rato e do teclado. Permite ler e enviar informação para realizar ações relativas a estes periféricos. A função **read_kbc()** lê do kbc o seu estado. A função **kbc_issue_command()** permite enviar um determinado comando para o kbc enquanto o número de tentativas for inferior ao máximo estipulado. A função **read_information()** vai buscar os dados do periférico, seja o **teclado** ou o **rato**, de acordo com o **irq** passado. Este módulo foi realizado por ambos os membros.

Peso no projeto: 5%

4.2.4 Módulo Timer

Este módulo foi desenvolvido para o lab2 e possui funções relativas ao funcionamento do *timer*. É de destacar a função **timer_int_handler()** que ao incrementar o contador, permite estabelecer o ritmo da movimentação das peças no jogo e para permitir estabelecer o intervalo de visualização de uma imagem no ecrã (utilizado para mostrar o jogo no ecrã e imagem de "*Game Over*"). Além disso, as funções de subscrição de interrupções **timer_subscribe_int()** e cancelamento de subscrição **timer_unsubscribe_int()**. Este módulo foi realizado por ambos os membros.

Peso no projeto: 7%

4.2.5 Módulo Keyboard

Este módulo foi desenvolvido para o lab3 e possui funções relativas ao funcionamento do teclado, nomeadamente funções para a subscrição, **keyb_subscribe_int()**, e cancelamento da subscrição, **keyboard_unsubscribe_int()**, assim como leitura da sua informação no *handler*, **kbc_ih()**. Este módulo foi realizado por ambos os membros.

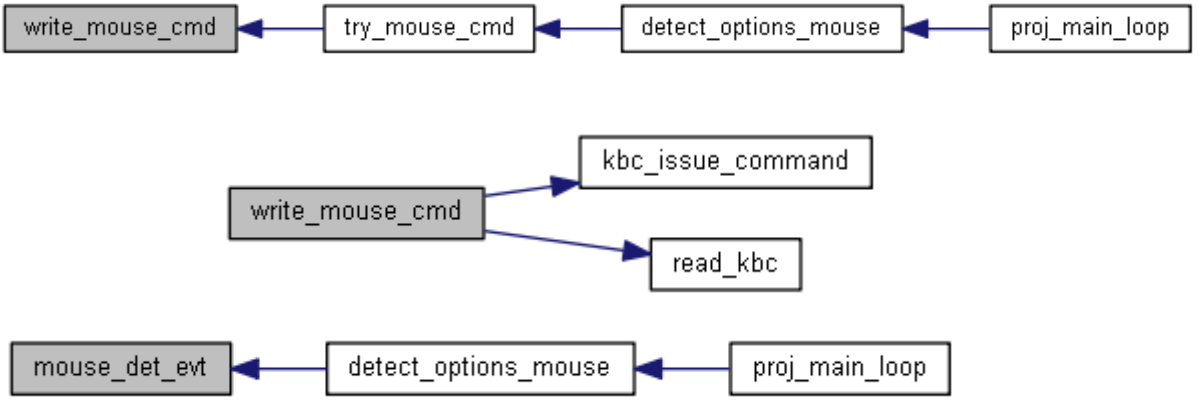
Peso no projeto: 7%

4.2.6 Módulo Mouse

Este módulo foi desenvolvido para o lab4 e possui funções relativas ao funcionamento do rato, nomeadamente a subscrição e cancelamento da subscrição, **mouse_subscribe_int()** e **mouse_unsubscribe_int()**, assim como funções para detectar movimentos do rato, detectar posições no ecrã e premir no botão esquerdo, utilizadas no menu principal. Para modificar o comportamento do rato foram implementadas as funções **try_mouse_cmd()** e **write_mouse_cmd(),** que tratam de enviar os comandos e lidar com o **acknowledge byte**. O *interrupt handler* , **mouse_ih()**, passa por ler a informação do rato, feito através da função **read_information()** do módulo **kbc**. A deteção de eventos do rato é feita através da função **mouse_det_evt()** que retorna uma estrutura com a informação do evento gerado.

Este módulo foi realizado por ambos os membros, apesar da contribuição do João Martins ter sido maior.

Peso no projeto: 7%

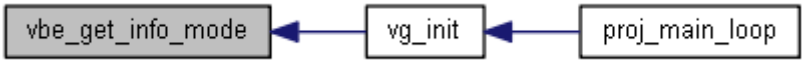


4.2.7 Módulo VBE

Desenvolvido para o lab5 permite a manipulação e controlo da parte gráfica do programa, destacando o seu papel para a inicialização do modo gráfico - **change_vbe_mode()** e **vbe_get_info_mode()**.

Este módulo foi realizado por ambos os membros.

Peso no projeto: 6%



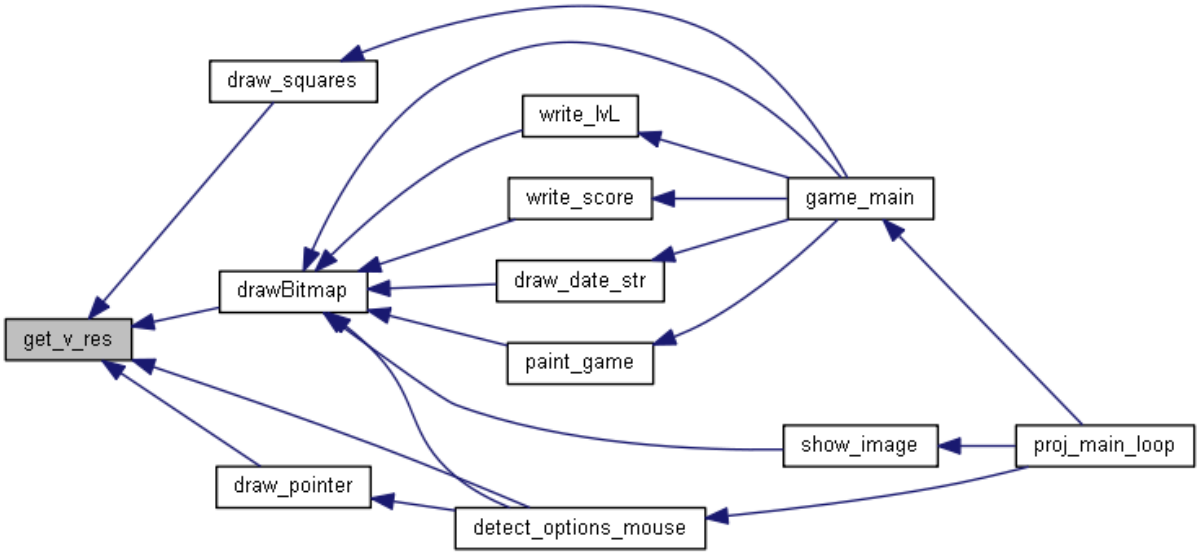
4.2.8 Módulo VideoG

Desenvolvido para o lab5 e tendo sido adicionadas novas funções - **clean_tmp()**

clean_screen(), que permitem limpar a informação do ecrã e do *back- buffer.* As restantes funções permitem: o desenho no ecrã, começando por desenhar por pixel, **vg_draw_pixel(),** cada um destes numa linha,**vg_draw_vline()**, formando um retângulo, **vg_draw_rectangle()**; a escolha do modo gráfico, **color_mode()**; e funções de retorno ("*getter*") de informação relativa a este módulo de forma a ser usado noutros locais, que não possuam acesso direto.

Foram realizadas alterações às funções já existentes,nomeadamente desenhar para um *buffer* temporário em vez de copiar diretamente para a memória, necessário para ***double buffering***.Este módulo foi realizado por ambos os membros, sendo que as novas funções foram criadas pelo João Martins.

Peso no projeto: 8%



4.2.9 Módulo RTC

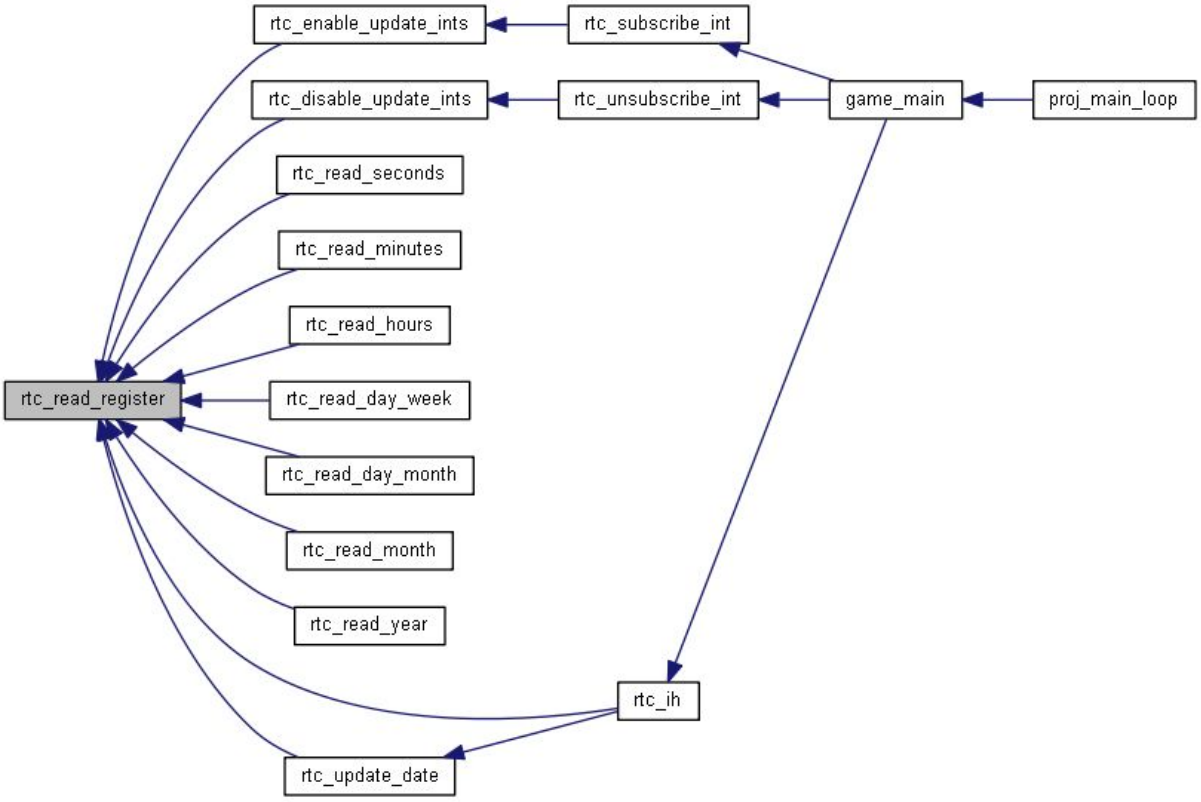
Módulo necessário para obter a informação da hora atual no computador.

Como foi referido anteriormente, recorreu-se a interrupções para obter a informação atualizada, às quais se faz subscrição e cancelamento através da funções **rtc_subscribe_int()** e **rtc_unsubscribe_int()**. Para aceder e alterar a configuração do RTC utilizam-se as funções **rtc_read_register()** e **rtc_write_register()** que permitem a leitura e escrita da informação na porta adequada. O método escolhido para a recolha de informação de forma segura corresponde ao uso de interrupções **update-ended**. Já a ativação e desativação das mesmas é realizada através das funções **rtc_enable_update_ints()** e **rtc_disable_update_ints()**. A interrupção é gerida através da função **rtc_ih()**. Para ler as informações da data e hora foram criadas as funções **rtc_read_*** sendo que cada componente (horas, minutos, etc) tinha a sua própria função.

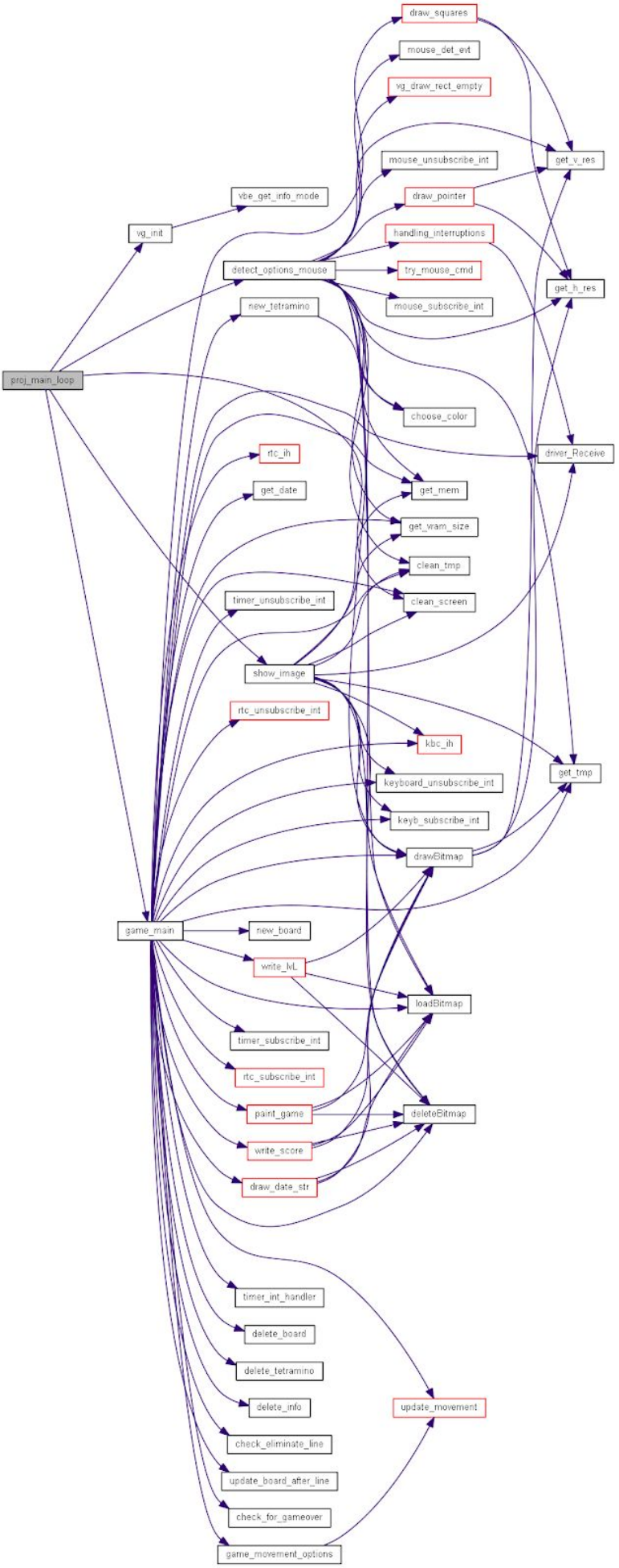
Para manter a informação da data de forma mais organizada foi criada a "classe" **Date**, que na verdade é uma struct que mantém a informação da data no formato BCD e tem associada funções, **getDate()** e **rtc_update_date()**, sendo que a primeira é responsável por enviar a estrutura da data e o seu conteúdo e a segunda atualiza a informação da data de acordo com uma nova interrupção do RTC.

Este módulo foi realizado pelo João Martins.

Peso no projeto: 6%


rtc_enable_update_ints
rtc_subscribe_int
rtc_disable_update_ints
rtc_unsubscribe_int
game_main
proj_main_loop
rtc_read_seconds
rtc_read_minutes
rtc_read_hours
rtc_read_register
rtc_read_day_week
rtc_read_day_month
rtc_read_month
rtc_read_year
rtc_ih
rtc_update_date

4.3 Diagramas de chamada de funções (proj e game)



5. Detalhes de implementação

Neste trabalho utilizamos uma programação orientada a objetos, sendo de destacar o uso de *structs* e *pointers*, sendo os primeiro essenciais para armazenar a estrutura do jogo e imagens bmp a implementar no ecrã. Já o uso de *pointers*, levou à necessidade de organização da memória em execução, através da criação de construtores e destrutores para assegurar que não ocorriam falhas de memória.

5.1 Estrutura interna

A estrutura interna do programa é necessária para guardar a informação do jogo.

Para isso recorremos a definição de três estruturas fundamentais: **Board** (quadro), **Tetramino** e **Square** (quadrado) . O **Board** é constituído por vários quadrados, e ao tetraminó estão associados os índices dos quadrados respetivos no quadro. Assim, apenas se encontra um tetraminó ativo no jogo de cada vez, uma vez que, quando pousam, toda a informação necessária ao correto funcionamento do jogo está presente na informação dos quadrados do quadro.

Deste modo, as três estruturas fundamentais (*structs*) possuem as seguintes características:

- cada quadrado (**Square**) contém informação da sua cor e posição inicial no quadro do jogo;

- cada tetramino é associado aos quadrado do board através de um array de índices dos respectivos quadrados no tabuleiro. Além disso, possui informação acerca da cor, posição de rotação (1 a 4) e tipo de forma (1 a 7). É de salientar o uso de um *pointer* para a struct do Board, o que facilita a recolha de dados dos quadrados referentes ao tetraminó, sem necessidade de passar um *pointer* do *board* na mesma função em que os tetraminós estão a ser utilizados.

- quadro de jogo (**Board**) é representado por um array de 300 quadrados (**Square**) e pelas respetivas dimensões e posição inicial na tela, em pixels.

Consideramos que a correta implementação da estrutura do programa foi essencial para o sucesso deste projeto.

```
typedef struct {
  uint16_t x;
  uint16_t y;
  uint16_t color;
} Square;
```

```
typedef struct {
  uint16_t sqidx[4];
  uint16_t color;
  int rot; //1 to 4
  Board *bd;
  int shape_n;
} Tetramino;
```

```
typedef struct {
  Square positions[300];
  uint16_t width;
  uint16_t height;
  uint16_t x_gini;
  uint16_t y_gini;
} Board;
```

5.2 Máquina de estados

Para realizar este trabalho considerámos essencial a estruturação de uma máquina de estados interna. Esta máquina de estados está definida no módulo proj. O estado inicial é o menu inicial, de onde saem três transições para outros estados, sendo que dois deles, jogo e créditos, retomam ao estado inicial. O terceiro é a saída do jogo e representa o estado final.

5.3 Apresentação

Neste jogo tentamos ser coerentes ao utilizar um conceito gráfico que se reflete em todo o programa. É importante referir que a imagem do menu inicial foi adaptada da imagem presente em http://projects.pixelatedawesome.com/tetris/images/screenshots/mainmenu.jpg .

É de destacar como efeito visual, o apontador do rato no menu que representa um quadrado dos tetraminós, sendo que com o movimento do rato, este muda de cor. Tal foi possível, atualizando o ecrã a cada interrupção do rato. Também é de realçar o retângulo que aparece a volta de cada opção do menu, quando o rato passa por essa região. Isto é feito através da função **vg_draw_rect_empty()** que desenha um retângulo sem preenchimento. A ilusão do desaparecimento do retângulo é feita tirando partido do facto da tela de fundo ser preta, pelo que desenhando um retângulo vazio produz esse efeito visual.

5.4 Jogo

É de salientar a existência de níveis de jogo,baseados na pontuação do jogador, sendo que, o seu aumento, tem como consequência, o aumento da rapidez do movimento das peças no jogo.

5.5 Gerar frames

A utilização de ***double buffering*** foi essencial para gerar corretamente as imagens no nosso jogo. Sem este mecanismo, desenhar no ecrã a cada interrupção do timer iria originar "*tearing*", uma vez que o desenho dos pixels novos seria demasiado lento em relação a necessidade de atualização do ecrã.

Para isto utilizamos um *back-buffer* e a memória principal (*front buffer*). Assim, a informação é enviada para o *back-buffer* e só depois é que é copiada para a memória principal através do uso de **memcpy()**.

6. Conclusão

Consideramos que esta unidade curricular transmite conceitos essenciais do funcionamento interno do computador, nomeadamente na utilização da interface de *hardware* de periféricos comuns.

Além disso, permite adquirir mais conhecimentos a nível de programação de baixo nível que apesar de terem menor atenção atualmente, continuam a ter extrema importância para qualquer profissional na nossa área de estudo.

Inicialmente os conceitos apresentaram-se bastante confusos, uma vez que os periféricos começaram a ser abordados sem compreensão dos conceitos base das chamadas ao *kernel* do sistema operativo. Contudo, ultrapassado este problema, apenas é necessária a compreensão do funcionamento de cada periférico.

Outra dificuldade que encontramos ao longo da cadeira é a falta de documentação online dos conteúdos abordados o que leva a que pequenos problemas demorem muito tempo a serem resolvidos.

Contudo, é de salientar a rápida resposta dos docentes quando dúvidas são colocadas no Moodle. Também acreditamos que as aulas teóricas foram fundamentais para compreender corretamente o funcionamento dos periféricos e que as aulas práticas se revelaram essenciais para corrigir problemas que surgiram ao longo dos vários laboratórios.

Por fim, consideramos que, apesar da cadeira transmitir conhecimentos essenciais na nossa área de estudo, o número de horas dedicadas e a complexidade da cadeira não se coadunam com o peso que esta tem.