A Nyan Game

Um jogo de combate de dois jogadores feito a partir de más ideias

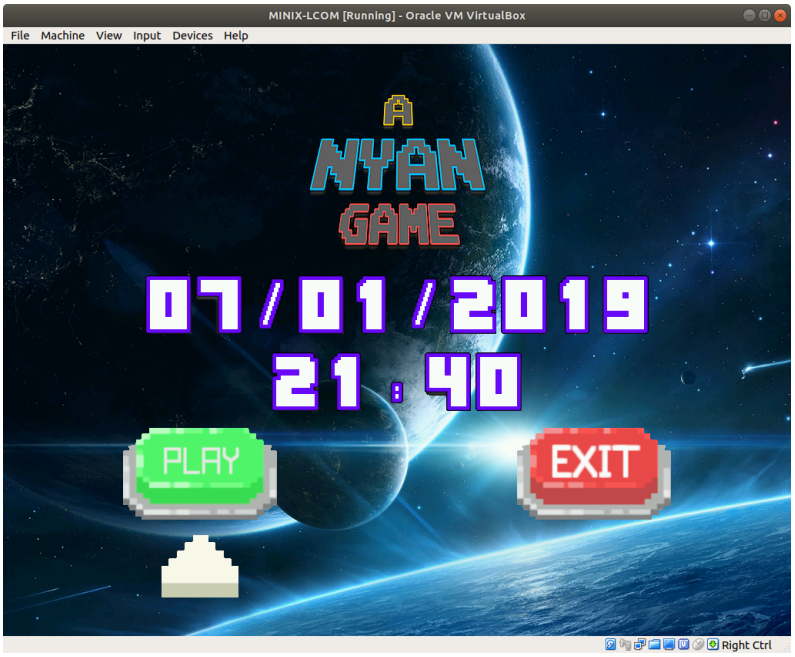
Projeto feito para a disciplina de LCOM por:
Sofia de Araújo Lajes, up201704066
Vitor Emanuel Moreira Ventuzelos, up201706403

Índice

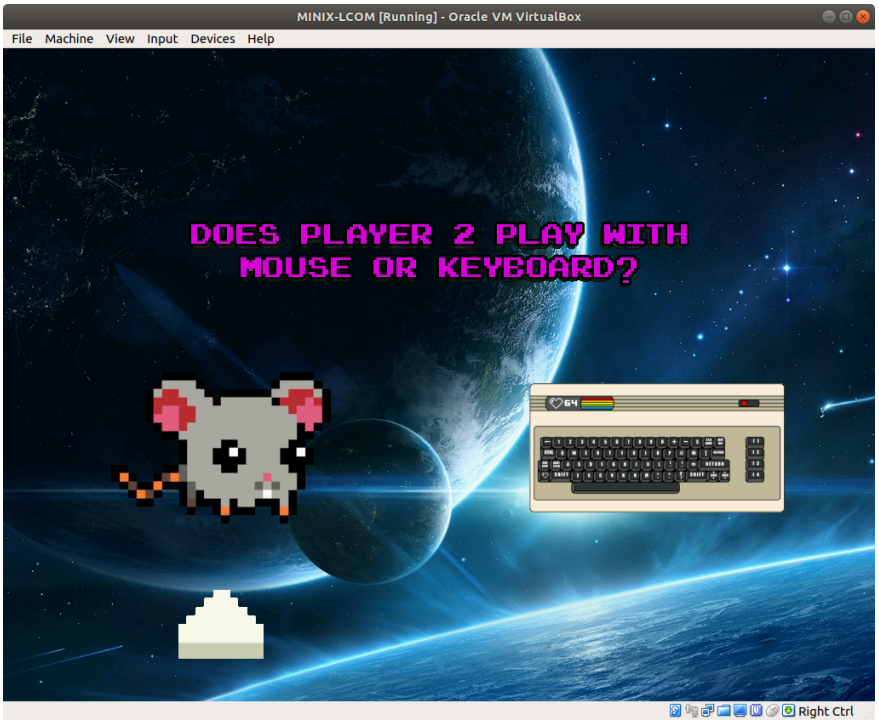
Instruções / Como jogar

Menus:

No menu inicial pode-se alternar entre dois botões com as setas esquerda e direita e selecionar um deles com a tecla "Enter". O botão "EXIT" permite sair do programa, enquanto o botão "START" permite iniciar o jogo, apresentando de seguida o menu de escolha do esquema de controlo do jogador 2.

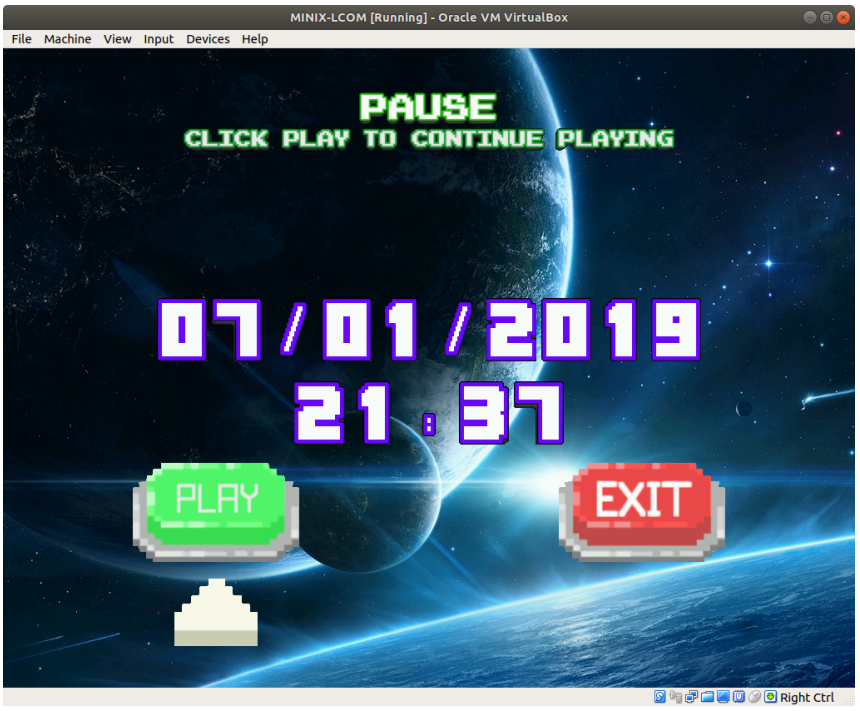


No menu de seleção do esquema de controlo do jogador 2, controlado da mesma forma que o primeiro, há, novamente, dois botões, sendo um deles um rato, que, ao ser selecionado permite controlar o jogador 2 com o rato, e sendo o outro um teclado, que permite controlar o jogador dois com o teclado. O jogo inicia-se ao selecionar uma destas opções.

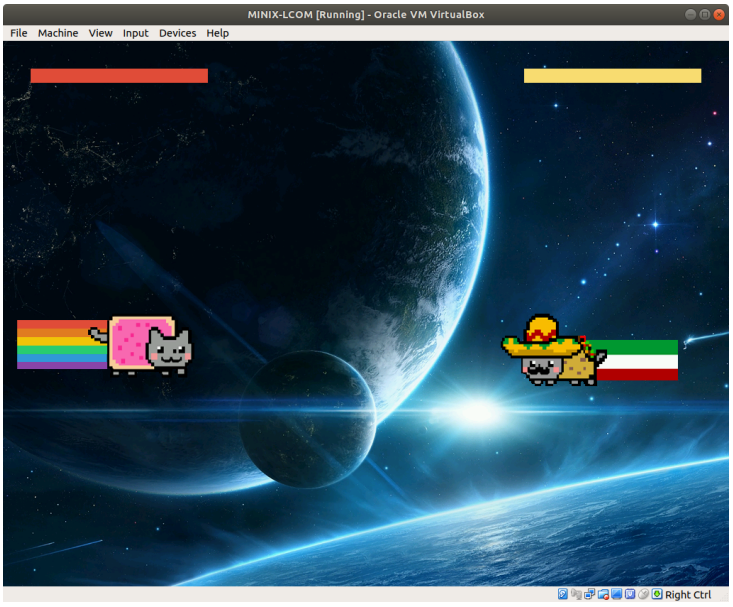


Durante o jogo:

Para pausar, pode premir a tecla “Esc”. O menu de pausa é controlado da mesma forma que o menu inicial e permite retornar ao menu inicial, apagando o jogo atual, caso seja selecionado o botão “EXIT” ou retornar ao jogo, caso “PLAY”.



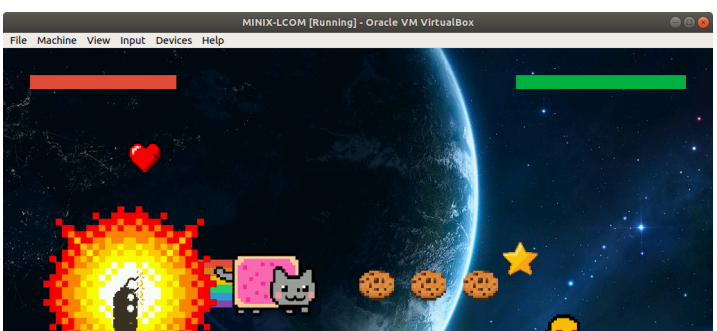
O jogador 1, que possui o sprite do Nyan Cat, é sempre controlado pelo teclado. Para se mover, utiliza as teclas ‘w’ para se mover para cima, ‘a’ para se mover para a esquerda, ‘s’ para se mover para baixo e ‘d’ para se mover para a direita. Este também possui duas teclas para disparar projéteis, sendo a tecla ‘g’ para disparar um projétil básico (“bullet”) na direção horizontal para a qual o jogador está orientado e na direção vertical para a qual o jogador se estiver a mover. Além desta, tem também a tecla ‘h’, que posiciona uma bomba atrás de si (cujo tempo entre a largada e a explosão é de apenas um segundo).



O jogador 2, que possui o sprite do Taco Cat, pode ser controlado de duas formas diferentes, sendo esta decisão feita no menu inicial como mencionado acima:

-> O primeiro esquema de controlo é semelhante ao do jogador um, através do teclado. Neste esquema de controlo o jogador 2 utiliza as setas para se mover na respetiva direção, dispara uma “bullet” premindo a tecla ‘.’ e larga uma bomba premindo a tecla ‘-’.

-> O segundo esquema de controlo utiliza exclusivamente o rato. O movimento é controlado através dos movimentos do rato, sendo limitado à mesma velocidade dos movimentos



controlados a partir do teclado, para manter o jogo equilibrado. Para disparar um projétil básico é utilizado o botão esquerdo do rato e para largar uma bomba é utilizado o botão direito do rato.
Nota: Use as bombas taticamente, uma vez que as "bullets" não tiram pontos de vida a quem as lançou mas estas atingem qualquer jogador!

Periodicamente, aparecem no campo de jogo *power-ups* que podem ser apanhados por ambos os jogadores. A estrela dá ao jogador que a apanhar 5 segundos de invulnerabilidade. O coração restaura até 100 pontos de vida ao jogador que o apanhar.

Estado do Projeto

Para uma maior facilidade de tratamento de "momentos" durante todo o jogo (*in-game* e *off-game*), e também para conseguir um código mais limpo e legível, decidimos criar dois ciclos de interrupções: um no módulo **dispatcher** (função **play()**) e outro no módulo do **menu** (função **Menu()**), lidando assim com a função **driver_receive()**, chamadas de interrupções e encaminhando para os seus respetivos *handlers*. (A forma como se entrelaçam e complementam é descrita na secção "Detalhes de Implementação).

Periférico	Utilização	Utiliza Interrupções?
Timer	- Controlar o *frame rate* - Normalizar a velocidade de movimento dos sprites	Sim
Keyboard	- Controlar os jogadores - Interação com os menus	Sim
Mouse	- Controlar o jogador 2	Sim
Video Card	- Apresentação gráfica do conteúdo jogável	Não
Real-Time Clock	- Recolher informação da data e hora - Programação de alarmes aplicada ao controlo de objetos	Sim

Timer

O *frame rate* do jogo é controlado através de um contador incrementado pelas interrupções do *timer* (irq 0), controlando assim também a velocidade de movimento dos sprites, sendo que as suas posições são atualizadas a cada frame.

A implementação utilizada para interagir com o *timer* é baseada na implementação desenvolvida para o Lab 2, sendo a maioria das funções criadas nessa implementação reutilizadas nesta, nomeadamente **timer_subscribe_int()**, **timer_unsubscribe_int()** e **timer_int_handler()**.

O controlo do *frame rate* é feito diretamente dentro da função **play()**.

Keyboard

O *keyboard* é utilizado para mover os jogadores (ou apenas o jogador um, caso esteja a ser usado o rato) para disparar projéteis pelos mesmos. Para estes efeitos são utilizadas interrupções, sendo as funções de subscrição (**kbd_subscribe_interrupt()** e **kbd_unsubscribe_interrupt()**) e processamento de interrupções (**kbc_ih()**) desenvolvidas no Lab 3 reaproveitadas.

Para determinar o que ocorre quando uma tecla específica é premida ou largada foram criadas duas funções importantes, **handling_scanbyte()**, que, dependendo do código recebido do *keyboard*, chama as funções apropriadas e altera os valores apropriados, aplicada *in-game*, e **menu_and_pause_kbc_handler()**, mais pequena uma vez que lida com menos teclas, específica para os menus.

Mouse

O Mouse é utilizado para controlar o movimento do jogador 2, caso essa opção seja selecionada. Para tal efeito, utiliza os packets recebidos através de interrupts para determinar as direções de movimento e para determinar se algum projétil é disparado premindo os botões do rato. Para processar os packets foi criada uma nova função, **handling_mouse_packet()**, que deteta mudança de estados nos botões e o sinal e o valor do deslocamento em ambos os eixos.

Video Card

Sendo o projeto um jogo, este apoia-se largamente na componente gráfica, pelo que grande parte do desenvolvimento se dedicou a este periférico, direta e indiretamente.

O modo de apresentação escolhido (**modo 0x117**) tem **resolução 1024x768**, com 16 bits por píxel, modo de cor direta **5-6-5**, permitindo assim um espectro de **65536 cores**.

Foram reaproveitadas algumas funções desenvolvidas no Lab 5, nomeadamente a **vg_init()**, que processa e inicia o modo gráfico escolhido, e a **vg_exit()**, retornando às definições normais do minix.

Utiliza ***double buffering***, gerido no módulo ***Draw***, tal que os objetos (estáticos ou em movimento) são "desenhados" no *buffer* secundário, declarado no mesmo módulo, que, de seguida, após a escrita de todos os objetos, é copiado para o *buffer* da memória de vídeo através da função **write_to_vram()**, chamada pela função que solicitou a impressão.

Este periférico é utilizado tanto para apresentar os menus como para apresentar o jogo em si, utilizando a função **draw_sprite()** para escrever no *buffer* secundário os sprites requeridos pelo contexto da função que a chama, usando também a função **draw_life_bars()** para desenhar as barras de vida dos jogadores, conferindo tamanho proporcional à vida de cada.

Por último, foi desenvolvido o módulo ***Colision***, com o fim de deteção de objetos, recorrendo a quatro funções: **verify_collisions()**, **isFrameCollision()**, **isObjectCollision()**, e **DrawCollisionBuffer()**; recorrendo a um *buffer* de tamanho mutável (adapta-se ao tamanho da janela de colisão).

Real-Time Clock

É utilizado o Real-Time Clock para dois fins: em primeiro lugar, a leitura da data e hora atuais através das funções **get_date()** e **get_time()**, situadas no módulo do Menu, apresentando-as tanto no menu inicial como no de pausa; para tal é feito uso das interrupções *update*. Em segundo lugar, para definir alarmes (**set_alarm()**) no módulo ***Dispatcher***, uma vez que as interrupções de alarme são utilizadas apenas durante o *in-game* (com o objetivo de fazer aparecer o objeto "Star" ou retirar o poder conferido por esta ao jogador após o disparo do alarme, e ainda, mostrar, durante 3 segundos, o ecrã de *game over*).

De modo a subscrever às interrupções deste periférico, bem como a desubscrever, foram designadas a estes fins as funções **rtc_subscribe_interrupt()** e **rtc_unsubscribe_interrupt()**, respetivamente. Foi ainda desenvolvido um handler, **rtc_ih()**, que devolve o tipo de interrupção (*UPDATE* ou *ALARM*) ou *UNIMPORTANT* caso não seja nenhuma destas, apenas por simbolismo e precaução.

Organização / Estrutura do Código

O código desenvolvido para este projeto está separado em módulos, para tornar a metodologia de trabalho mais eficaz, e também para o tornar minimamente reutilizável e legível.

Módulos de interação com periféricos

Os módulos apresentados nesta secção interagem diretamente com periféricos, e maioria é baseada nas implementações dos labs desenvolvidos ao longo do semestre.

- Módulos do Timer

Módulo "timer"

Neste módulo encontram-se as funções de subscrição, remoção da mesma e *handling* de interrupções relativas ao Timer 0. Este módulo foi desenvolvido maioritariamente durante o Lab 2, tendo ambos os membros do grupo contribuído igualmente na sua criação.

Módulo "i8254"

Este módulo contém constantes utilizadas para manipular o Timer 0, e foi-nos fornecido para execução do Lab 2, sendo que parte dele não é da nossa autoria. A este foram adicionadas outras constantes relevantes pela Sofia Lajes.

- Módulos do Keyboard

Módulo "keyboard"

Neste módulo encontram-se as funções de subscrição, remoção da mesma e *handling* de interrupções relativas ao keyboard, contendo também uma função que permite ler corretamente interrupções de teclas que enviam dois códigos. Este módulo foi desenvolvido maioritariamente durante o Lab 3, tendo ambos os membros do grupo contribuído igualmente na sua criação.

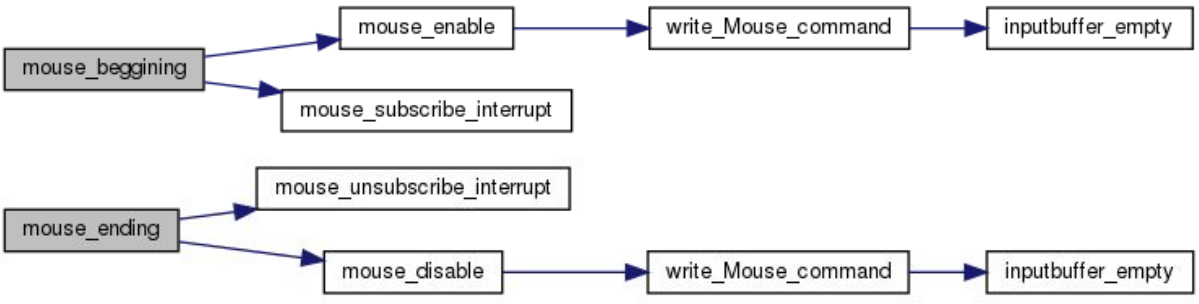
Módulo "i8042"

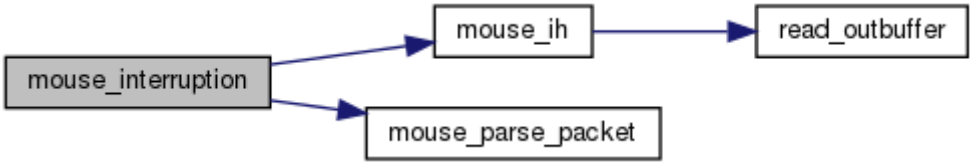
Este módulo é comum ao keyboard e ao mouse (descrito na próxima secção), pois ambos controlados pelo KBC (KeyBoard Controller). Aqui estão contidas constantes necessárias para manipular ambos os periféricos. Foi desenvolvido ao longo dos Labs 3 e 4 por ambos os membros do grupo.

- Módulos do Mouse

Módulo "mouse"

Este módulo contém as funções de configuração do mouse, subscrição, remoção da mesma e *handling* de interrupções relativas ao mouse e *parsing* dos *packets* recebidos através destas interrupções. Foi desenvolvido ao longo do Lab 4, com contribuição semelhante de ambos os membros do grupo.

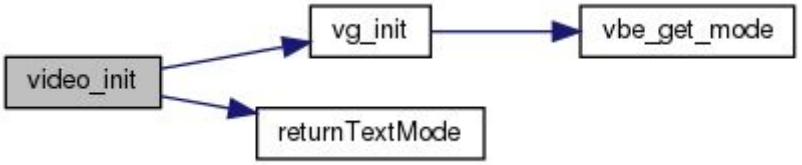




- Módulos da Placa de Vídeo

Módulo "graphics"

Este módulo contém as funções de configuração da VBE e da placa gráfica, e contém também algumas funções utilizadas para debugging. Foi desenvolvido durante o Lab 5, maioritariamente pela Sofia Lajes.



Módulo "draw"

Neste módulo estão as funções que desenham os sprites no buffer temporário e que copiam o conteúdo deste buffer para o buffer da memória de vídeo, sendo assim o módulo responsável pelo *double buffering* utilizado no projeto. Foi desenvolvido durante o Lab 5, maioritariamente pelo Vitor Ventuzelos.

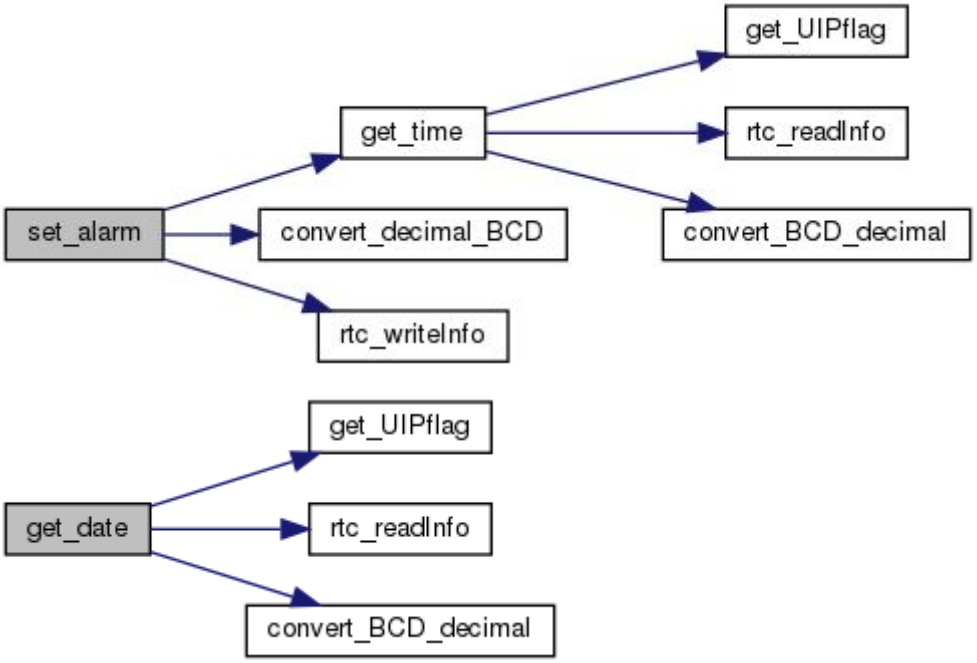
Módulo "vbe_macros"

Contém constantes utilizadas para manipular o VBE. Foi desenvolvido pela Sofia Lajes.

- Módulos do Real-Time Clock

Módulo "rtc"

Neste módulo encontram-se funções destinadas a configurar o Real-Time Clock, subscrever, processar e remover subscrições de interrupções e gerir alarmes. Foi desenvolvido pela Sofia Lajes.



Módulo "rtc_macros"

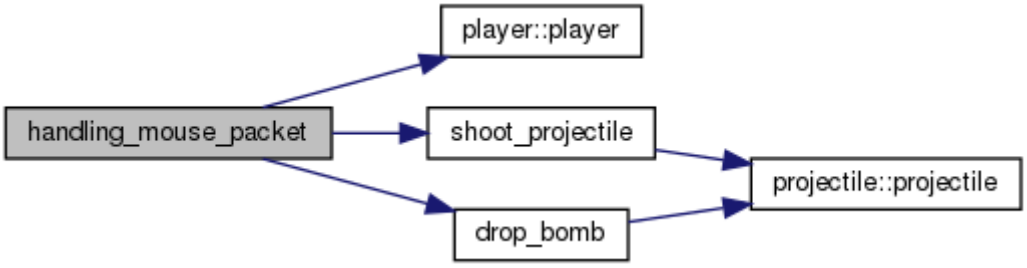
Contém constantes úteis para configurar e manipular o Real-Time Clock. Foi desenvolvido pela Sofia Lajes.

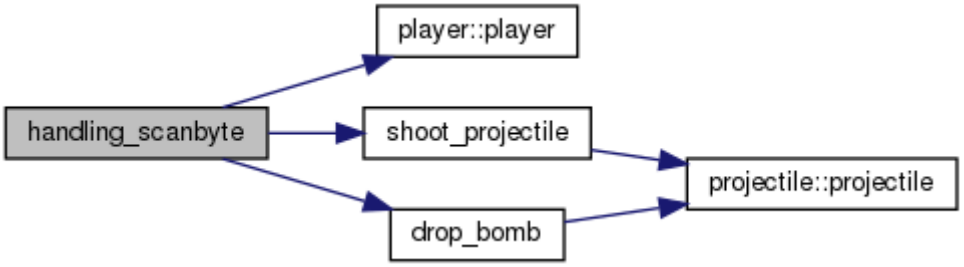
Módulos de processamento do jogo

Os módulos descritos nesta secção contém as funções que detetam eventos, processam os mesmos, e dão progresso ao jogo em concordância com estes.

Módulo "kbc_handler"

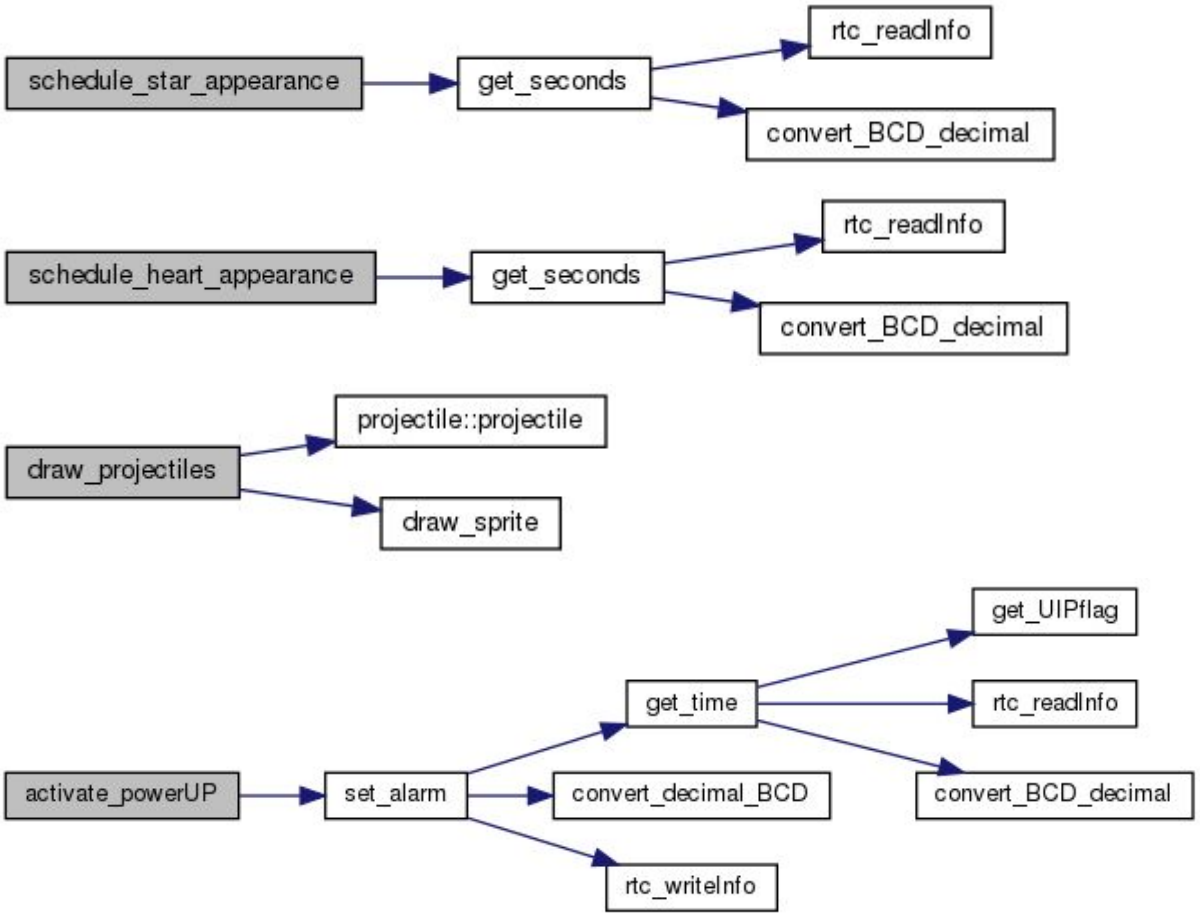
Este módulo contém um par de funções que funcionam de forma semelhante a uma máquina de estados, sendo chamadas quando há uma interrupção do keyboard ou do mouse (uma função para cada evento), e alteram o estado dos jogadores consoante o tipo de evento. Este módulo foi desenvolvido maioritariamente pelo Vitor Ventuzelos.





Módulo "projectile_handler"

Este módulo contém funções relativas à criação, movimentação e desenho de projéteis. Contém também objetos que funcionam como templates de projéteis para tornar a criação destes durante o jogo mais rápida. Este módulo foi implementado pelo Vitor Ventuzelos.

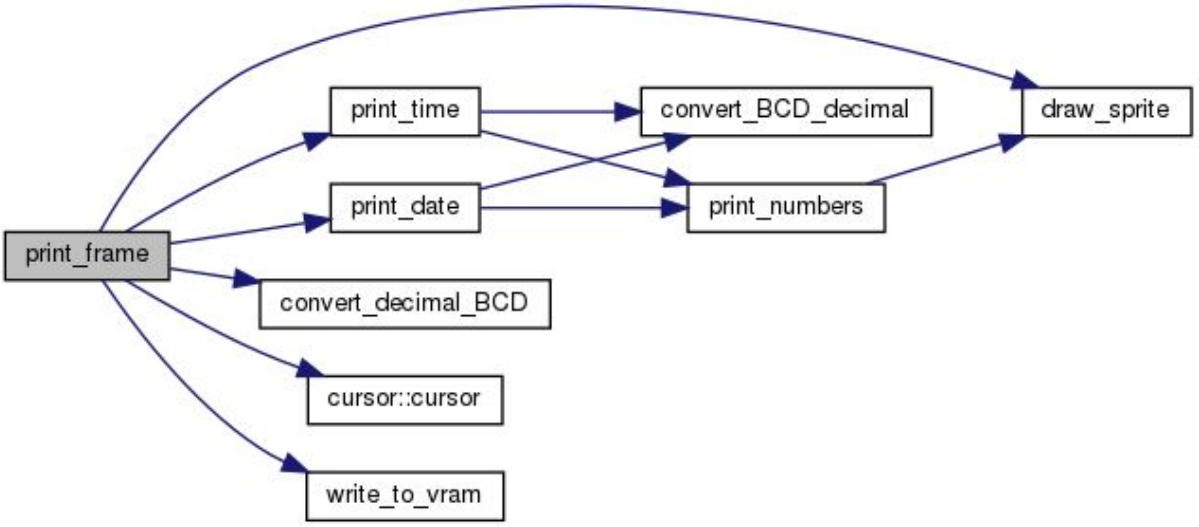


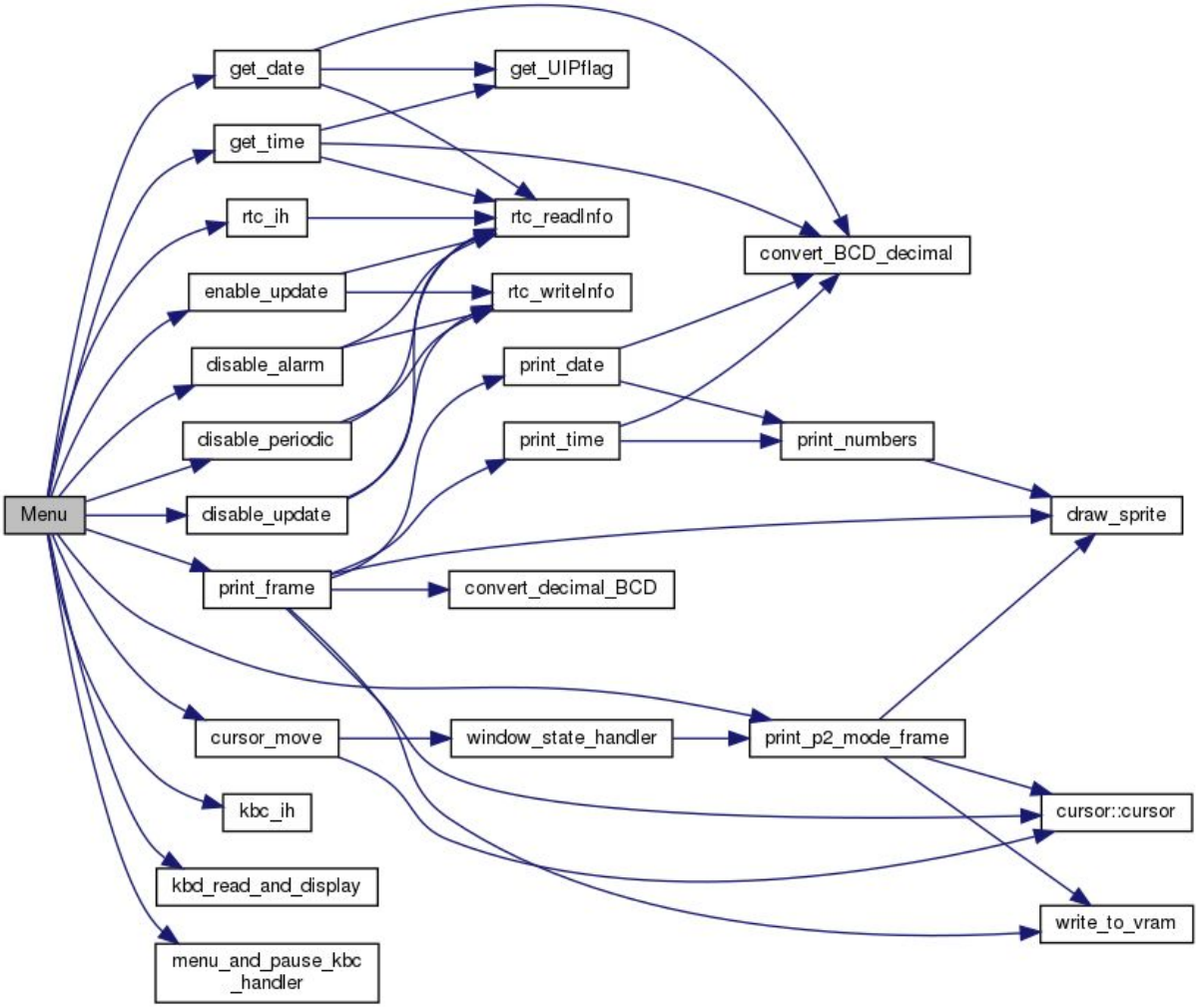
Módulo "player_handler"

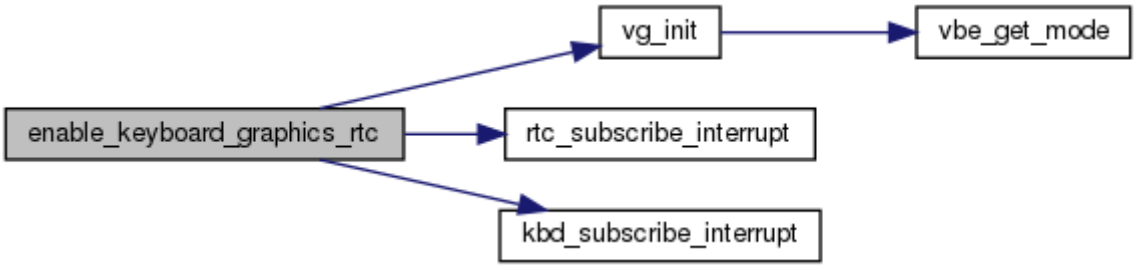
Neste módulo encontram-se funções respetivas ao movimento dos jogadores e a função responsável por inverter a orientação dos mesmos ao se cruzarem. Este módulo foi desenvolvido por ambos os membros do grupo.

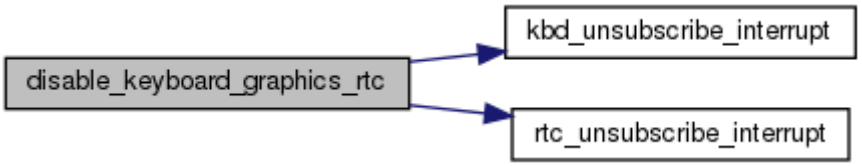
Módulo "main_pause_menu"

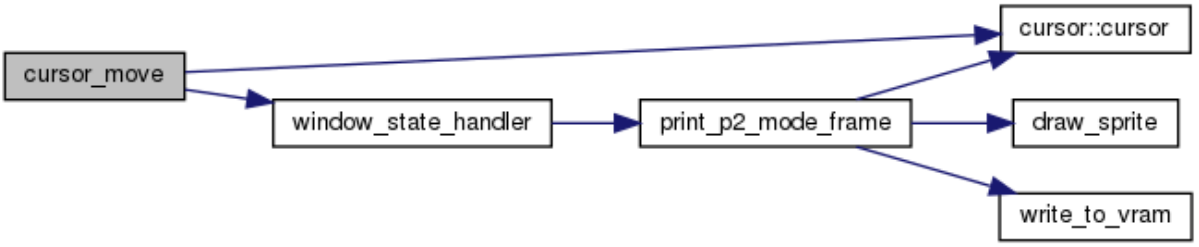
Este módulo contém código referente ao controlo e operação de todos os menus do jogo, contendo também funções de reinicialização do jogo. Contém também a função **Menu()** que é uma das funções que chama **driver_receive()**. Foi desenvolvido pela Sofia Lajes.




get_date
get_UIPflag
get_time
rtc_ih
rtc_readInfo
convert_BCD_decimal
enable_update
rtc_writeInfo
disable_alarm
print_date
disable_periodic
print_time
print_numbers
Menu
disable_update
draw_sprite
print_frame
convert_decimal_BCD
cursor_move
window_state_handler
print_p2_mode_frame
kbc_ih
cursor::cursor
kbd_read_and_display
write_to_vram
menu_and_pause_kbc
_handler

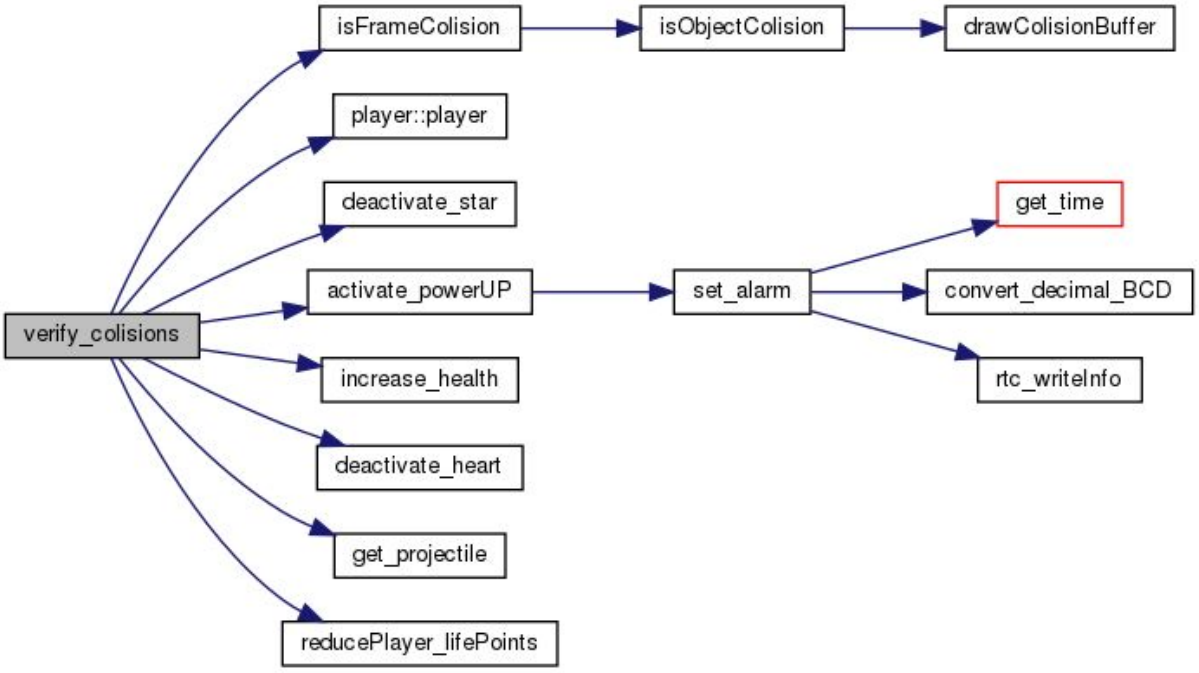

vg_init
vbe_get_mode
enable_keyboard_graphics_rtc
rtc_subscribe_interrupt
kbd_subscribe_interrupt


kbd_unsubscribe_interrupt
disable_keyboard_graphics_rtc
rtc_unsubscribe_interrupt


cursor::cursor
cursor_move
window_state_handler
print_p2_mode_frame
draw_sprite
write_to_vram

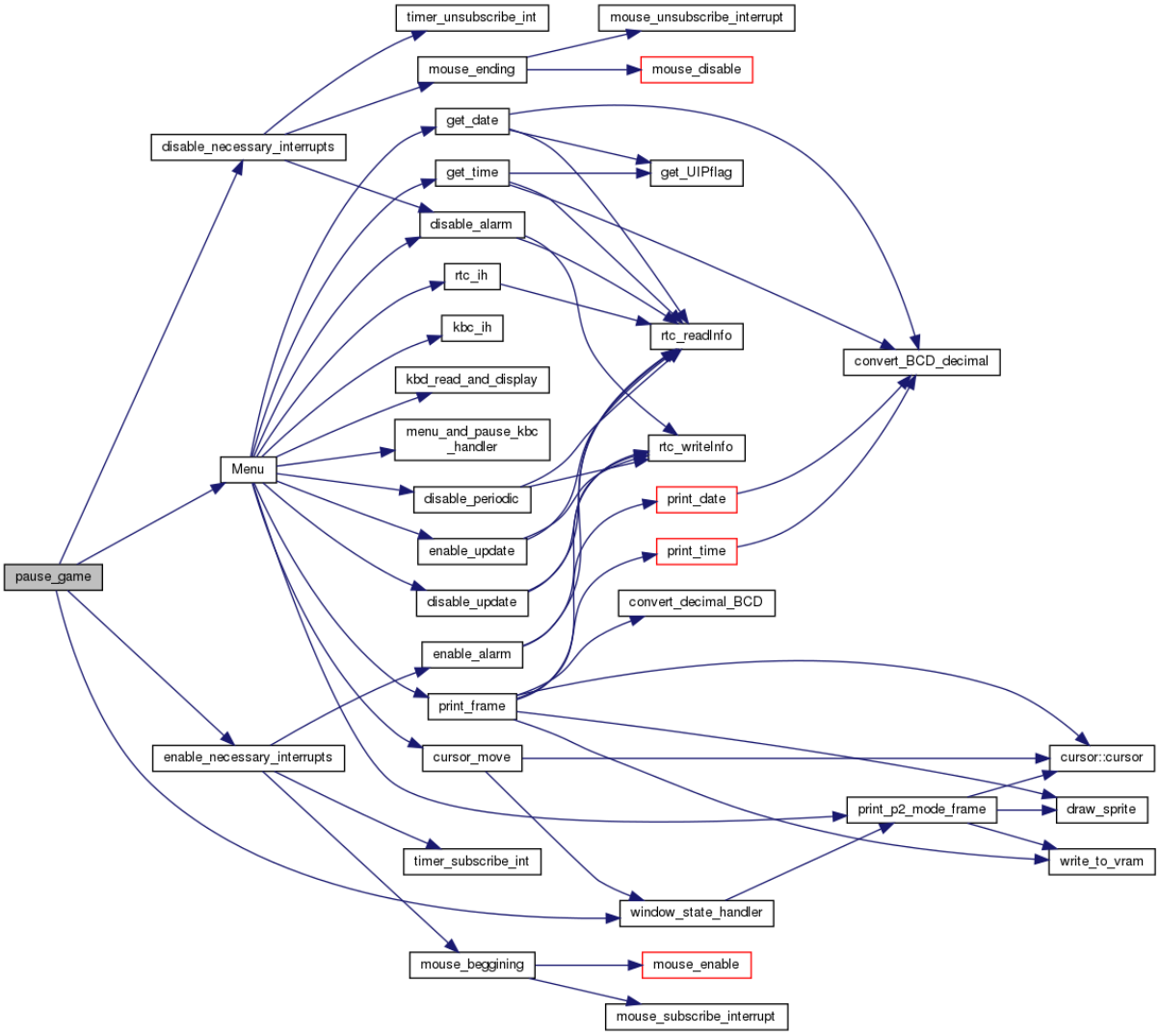
Módulo "collisions"

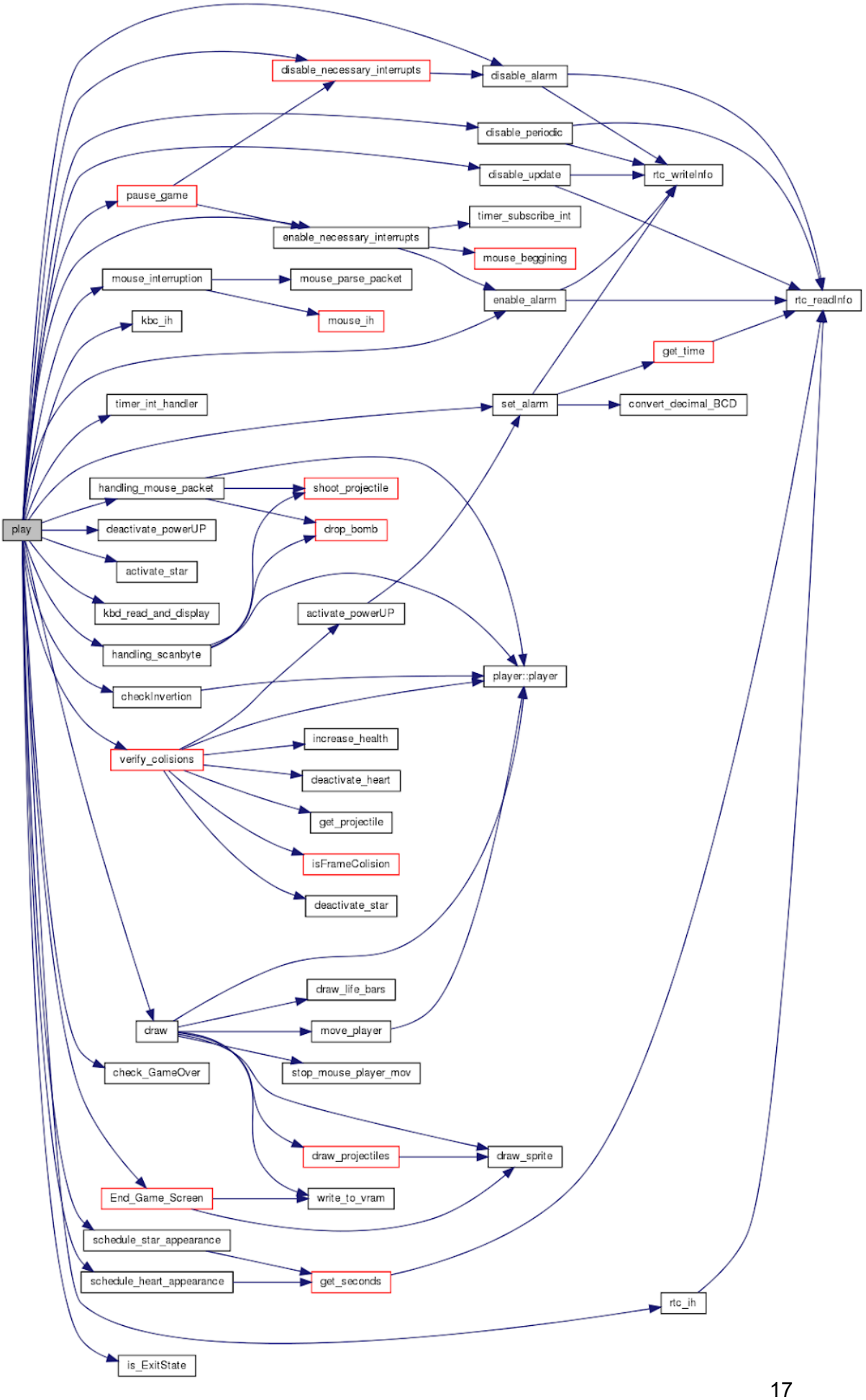
A colisão entre dois objetos é feita em dois passos: *Object Frame Testing* (**isFrameCollision()** que verifica se duas frames de *sprites* se *intersetam*) e *Pixel Collision Testing* (**isObjectCollision()** - que prepara a janela de colisão - e **DrawCollisionBuffer()** - que escreve, num *buffer* criado à medida desta, os píxeis de cada objeto (contidos na janela), verificando se há colisão de píxeis. Como mencionado anteriormente, **verify_collisions()** todos os objetos que possam colidir, dois a dois, chamando as respetivas funções ou aplicando diretamente as suas consequências. Este módulo foi maioritariamente desenvolvido por Sofia Lajes.



Módulo "dispatcher"

Este módulo contém a função principal do jogo, **play()**, dentro da qual se encontra o ciclo de **driver_receive()**, e a partir da qual se invoca as restantes funções, e também a função que invoca a função **menu()** no modo de menu de pausa. Este módulo foi desenvolvido por ambos os membros do grupo.


timer_unsubscribe_int
mouse_unsubscribe_interrupt
mouse_ending
mouse_disable
get_date
disable_necessary_interrupts
get_time
get_UIPflag
disable_alarm
rtc_ih
kbc_ih
rtc_readInfo
convert_BCD_decimal
kbd_read_and_display
menu_and_pause_kbc
_handler
rtc_writeInfo
Menu
disable_periodic
print_date
enable_update
print_time
pause_game
disable_update
convert_decimal_BCD
enable_alarm
print_frame
enable_necessary_interrupts
cursor_move
cursor::cursor
print_p2_mode_frame
draw_sprite
timer_subscribe_int
write_to_vram
window_state_handler
mouse_beggining
mouse_enable
mouse_subscribe_interrupt


play
disable_necessary_interrupts
disable_alarm
disable_periodic
disable_update
rtc_writeInfo
pause_game
timer_subscribe_int
enable_necessary_interrupts
mouse_beggining
mouse_interruption
mouse_parse_packet
enable_alarm
rtc_readInfo
kbc_ih
mouse_ih
get_time
timer_int_handler
set_alarm
convert_decimal_BCD
handling_mouse_packet
shoot_projectile
deactivate_powerUP
drop_bomb
activate_star
kbd_read_and_display
activate_powerUP
handling_scanbyte
player::player
checkInvertion
increase_health
verify_colisions
deactivate_heart
get_projectile
isFrameColision
deactivate_star
draw_life_bars
draw
move_player
check_GameOver
stop_mouse_player_mov
draw_projectiles
draw_sprite
End_Game_Screen
write_to_vram
schedule_star_appearance
schedule_heart_appearance
get_seconds
rtc_ih
is_ExitState

Módulo "object"

Neste módulo encontram-se declaradas structs utilizadas para representar os objetos do jogo, sendo algumas delas extensões da estrutura "mãe" **objeto**. Este módulo foi idealizado e desenvolvido maioritariamente pela Sofia Lajes.

Gráficos do jogo

Este parâmetro, quer relativo ao arranjo ou desenho de sprites e as suas conversões para **X PixMap** (extensão .xpm), quer a sua aplicação no jogo foi maioritariamente desenvolvido por Sofia Lajes.

Estas foram guardados como *header files* na pasta do projeto, isto porque foi a forma escolhemos para guardar e aceder às *sprites*, adicionando estes *headers* ao módulo *Objects*. Há, contudo, uma complicação, uma vez que não conseguimos agrupar todos os xpm num único ficheiro header (o que gerava um erro do EDI (Visual Studio Code) quando o número de linhas de código e tamanho da informação era muito grande), o que criou uma impossibilidade de organização dos ficheiros, havendo cerca de 27 ficheiros header relativos apenas a conteúdos de sprites. Lamentamos esta situação, podemos, contudo, garantir a funcionalidade do programa e a utilidade e impacto de mais de 30 sprites.

A maioria dos sprites utilizados foram obtidos através de imagens adquiridas e editadas no GIMP (GNU Image Manipulation Program). Achamos importante referir que nem todas foram feitas por nós, nomeadamente a que representa o Player 1, o "Nyan Cat" um desenho pixel-art popularizado em 2011.

É de referir ainda a utilização da função **xpm_load()** que nos foi fornecida e facilitou imenso a obtenção e utilização de *pixmaps*.

Detalhes da Implementação

O nosso objetivo era criar, acima de tudo, um programa robusto, o qual foi possível devido à idealização prévia de toda a estrutura e de cada uma suas partes.

Tendo em consideração o trabalho realizado nos laboratórios, e aprendendo com os erros cometidos, em especial, a desorganização e falta de generalidade em alguns materiais, tentámos não repetir o sucedido, e fazer uma **construção por camadas**: aperfeiçoar a interação com cada periférico, garantindo assim a sua funcionalidade com sucesso e sem sobressaltos; ler e trabalhar a informação num nível não completamente disperso do periférico, mas tentando encontrar um intermédio; escalar a abstração até que a manipulação da informação seja percetível e lógica.

Foi aplicado o conceito de **máquina de estados**, umas de extrema simplicidade (como o registo da *window* (estado atual do jogo), declarado e utilizado como tipo **enum** no módulo Menu, com o objetivo de identificar quais os passos seguintes: se vai para o Menu Inicial ou sai, se volta ao jogo ou ao Menu de Pausa) como a trabalhado no **window_state_handler()**, e outras de dificuldade mais acentuada e específica ao seu contexto: como se pode ver nos handlers como o **handling_mouse_packet()**.

Tendo em consideração o que foi falado no parágrafo anterior, é necessário referir a forma como operam individual e conjuntamente os dois ciclos de interrupções, localizados no módulo Menu e Dispatcher.

O programa inicia-se no Menu Inicial, a window atual refere-se a este portanto, neste momento, há um ciclo de interrupções que lida apenas com o keyboard e o rtc, das quais este último gera apenas interrupções de update, atualizando a data e hora se diferente, e a tela é atualizada, apenas se houver interrupção de um destes, por via a poupar acessos desnecessários à VRAM. Se o jogador escolher sair, esta passa a ter um valor "Exit", que, ao sair do menu e ser verificado, termina o ciclo (Menu -> Jogo -> Menu) e sai do programa, voltando às definições originais do minix. Caso a escolha do utilizador seja jogar, o valor da window é diferente de Exit, é IN-GAME, e assim se inicia a jogada. Neste momento, sai-se do primeiro ciclo e entra-se no do Dispatcher, onde se ativa o mouse, o timer e o alarme.

Onde estes se interlaçam é no momento em que o jogador escolhe pausar o jogo clicando "ESC", assim, o ciclo atual é parado, as interrupções do alarme são desligadas (bem como as do timer e do mouse para não haver interferência) e se entra no ciclo de interrupções do menu, até que o jogador volte ao jogo ou saia.

Como mencionado anteriormente as colisões são tratadas através de dois passos: *Object Frame Testing* e *Pixel Collision Testing.* O raciocínio é:

-> Primeiro, testamos se os retângulos limitadores das sprites cuja colisão queremos testar, formados pelos respetivos comprimento e altura, têm coordenadas em

comum. Em caso negativo, não há qualquer possibilidade de haver colisão. Em caso positivo, passamos para o segundo teste;

-> Segundo, desenhamos num buffer (array de tamanho mutável consoante a janela de colisão) primeiro, o primeiro objeto "o1", onde escrevemos livremente, incluíndo as invisibilidades, (tendo, obviamente, em atenção, quais os píxeis deste objeto que estão dentro da janela de colisão), e, de seguida, o segundo objeto, verificando, a cada escrita no buffer, se há sobreposição de píxeis não-transparentes.

Este projeto é maioritariamente orientado a objetos, como pode ser evidenciado pelo uso extensivo das *structs* definidas no ficheiro *objects.h*, utilizadas para representar todos os objetos envolvidos no processamento do jogo, sendo esta a forma mais simples e eficaz que guardar e utilizar dados referentes a um jogador ou um projétil.

Conclusões

Como a primeira cadeira em que tivémos que fazer algum cujo tema dependia quase totalmente da nossa criatividade, consideramo-la desafiante e energizante. Sendo também o nosso primeiro contacto com periféricos, apresentou alguma dificuldade no desenvolvimento do código, dificuldade que foi parcialmente colmatada devido à decisão por parte dos professores de disponibilizar duas semanas para cada Lab, o que nos deu tempo para perceber o funcionamento dos periféricos com que trabalhámos.
No entanto, temos também algumas falhas a apontar:

- Apesar de o professor Pedro Souto explicar a maioria da matéria nas teóricas, os apontamentos disponibilizados na página da disciplina contêm uma quantidade considerável de informação irrelevante, que apenas confunde ao ler.

- Consideramos a cotação atribuída à implementação de periféricos apenas referidos teoricamente um pouco excessiva.

- As respostas às dúvidas apresentadas nos fóruns são, por vezes, pouco esclarecedoras.

Temos também que valorizar alguns pontos fortes da cadeira, nomeadamente:

- A disponibilidade para esclarecer dúvidas dos professores monitores é louvável, e a sua ajuda nas aulas práticas foi indispensável.

- A adição da LCF ao Minix facilitou consideravelmente a cadeira em geral, sendo uma grande ajuda para testar as funções criadas nos Labs.