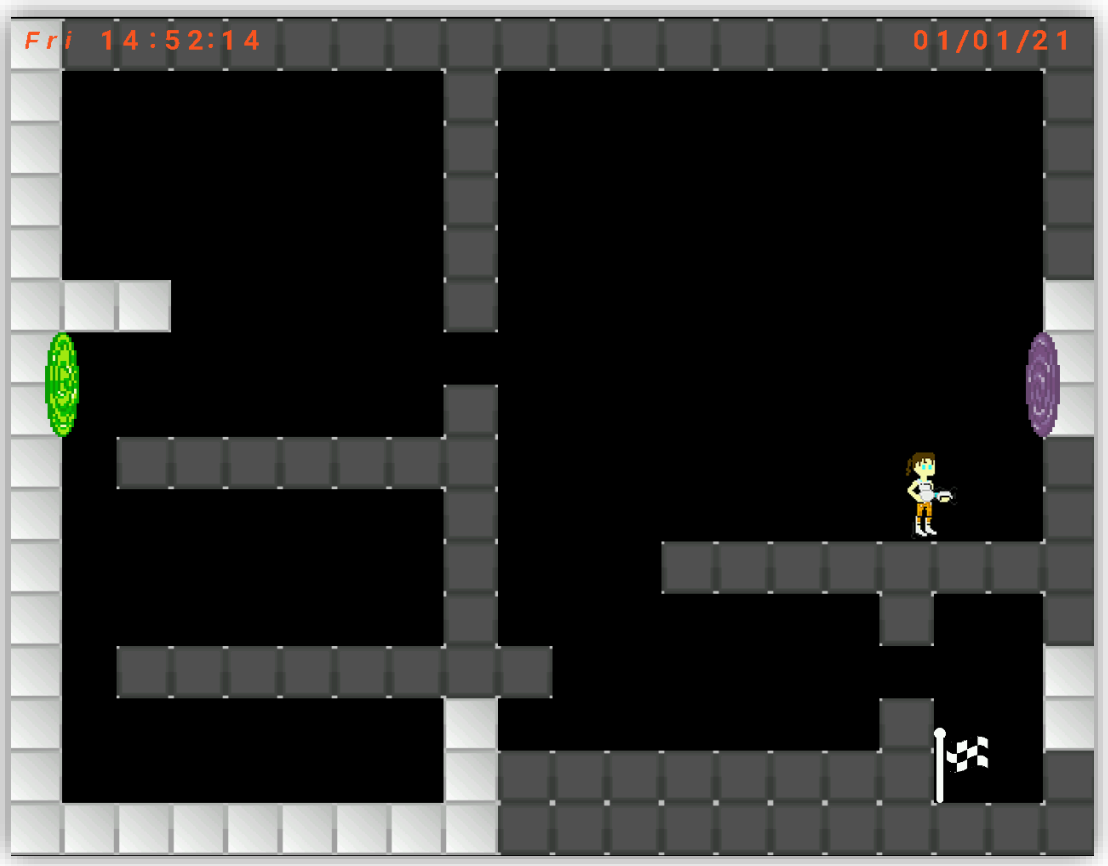
U.PORTO

FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

Portal 2D

Jogo 2D inspirado na saga “Portal”



Laboratório de Computadores 2020/21

Turma 7 – Grupo 7

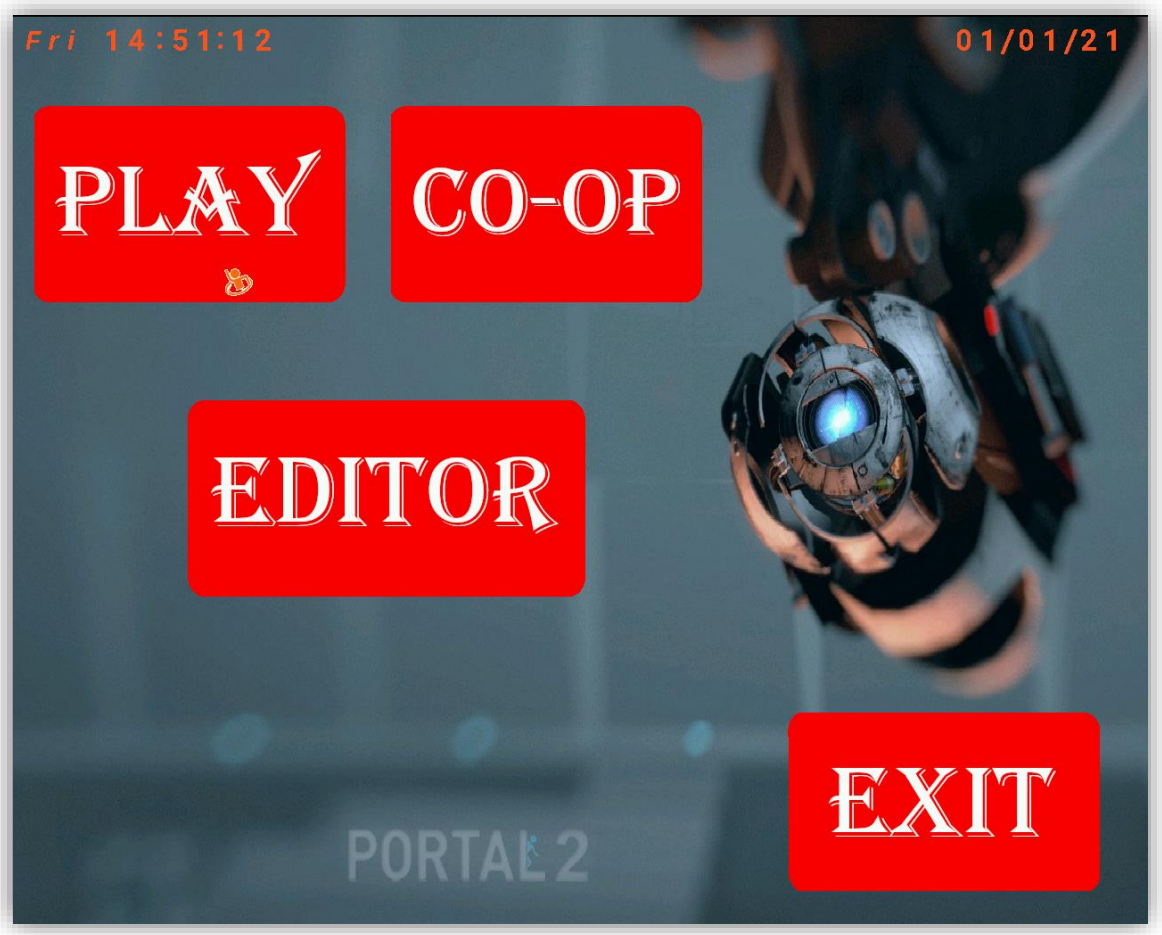
Bruno André Moreira Rosendo up201906334@fe.up.pt

Índice:

1- Instruções de Utilização:

O programa inicia no menu principal, onde são apresentados quatro botões:

- Play - Inicia o jogo em modo *singleplayer* (ver secção 1.1)
- Co-op - Inicia o jogo em modo cooperativo e multijogador remoto (ver secção 1.2)
- Editor – Inicia o editor de níveis (ver secção 1.3)
- Exit – Termina o programa.

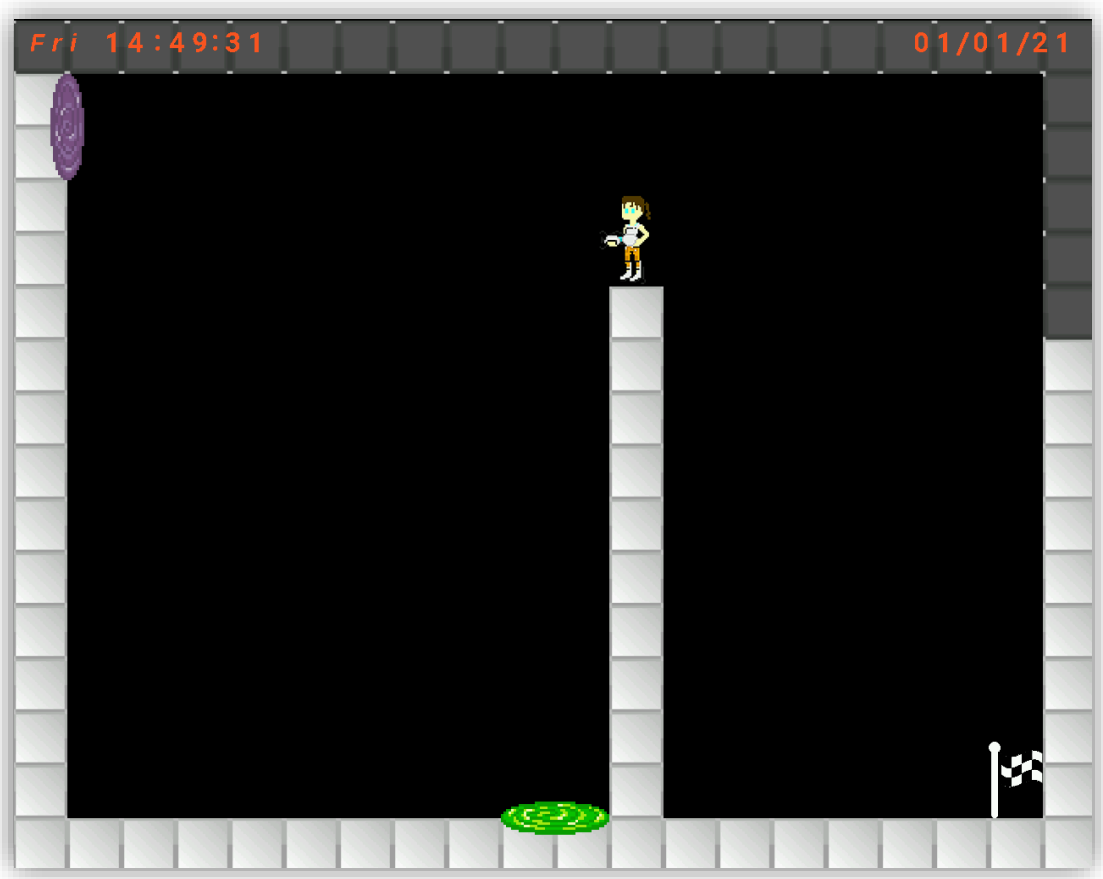


Para escolher uma das opções, basta clicar com o botão esquerdo do rato na opção pretendida.

1.1-Modo *Singleplayer*:

Quando é selecionada a opção “Play”, o jogo principal é iniciado, sendo carregado o primeiro nível. O jogo tem um número indefinido de níveis, visto que o jogador pode criar a quantidade de níveis que pretender (ver mais na secção 1.3). O jogo-base já conta com 4 níveis prontos a jogar.

O objetivo do jogador, em cada nível, é de apenas chegar à bandeira. No entanto, isto pode não ser tão simples quanto parece. Cada nível tem a sua dificuldade (dependendo também dos que foram criados) e, para o completar, o jogador terá que ser criativo, usando (e reutilizando) os seus portais para chegar a locais que de outro modo seriam impossíveis de alcançar e tirando proveito das físicas do jogo, tal como a gravidade e o impulso dado pelo *input* do teclado.



Para controlar a sua personagem, o jogador utiliza as teclas “A” e “D” para andar para a esquerda e para a direita, respetivamente. Para saltar, pode tanto usar o “W” como a tecla do espaço.

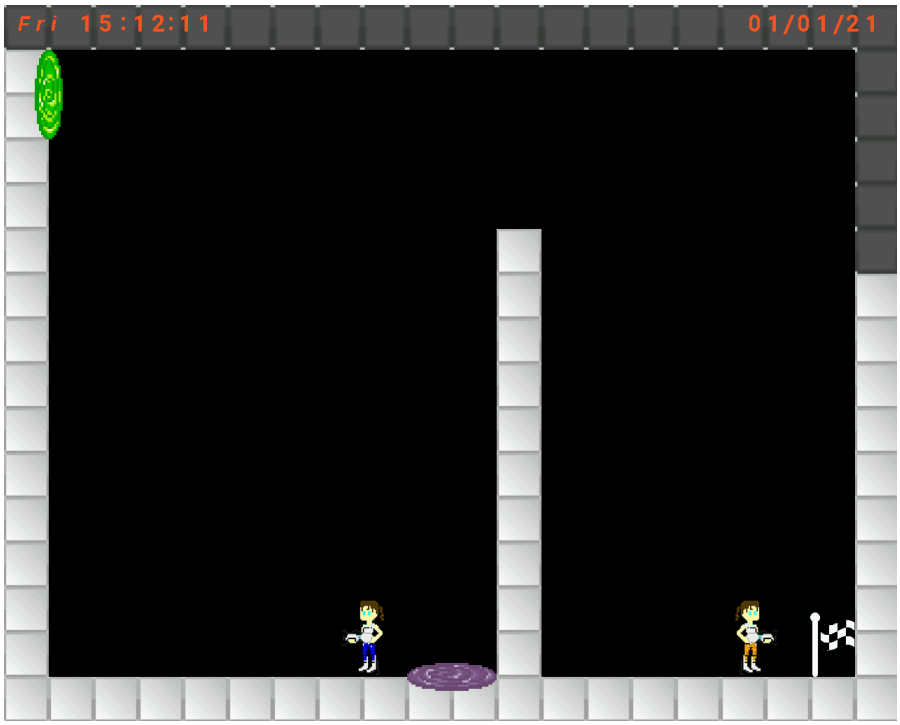
Relativamente aos portais, estes podem ser colocados pressionando o botão esquerdo do rato para o portal roxo e o botão direito para o portal verde, sendo que estes serão colocados no bloco branco mais próximo no sentido da personagem até ao cursor do rato. Se colidir com um bloco preto (ou nenhum), o portal não será colocado. No entanto, o jogador não pode utilizar esta ferramenta de forma

ilimitada, pois apenas podem existir um portal roxo e um verde em simultâneo, estando eles ligados entre si. Se o jogador tentar colocar um portal já existente, o antigo será destruído!

Se o jogador desejar sair do jogo antes de completar todos os níveis, poderá carregar na tecla “ESC” e regressar ao menu principal.

1.2-Modo Cooperativo:

Quando é selecionado a opção “Co-op”, o jogador é levado para um modo muito semelhante ao modo *singleplayer*. A diferença é que desta vez ele estará a jogar com um jogador remoto. Se o segundo jogador ainda não tiver entrado neste modo, o jogo ficará congelado até ele se conectar.



Os controlos funcionam de forma semelhante à descrita na secção 1.1, com as seguintes diferenças:

- Cada jogador pode apenas colocar um tipo de portal. Para isto, o jogador local pode clicar no botão esquerdo do rato para criar portais roxos, enquanto que o jogador remoto pode criar portais verdes.
- Para terminarem o nível, ambos os jogadores terão de chegar à bandeira.

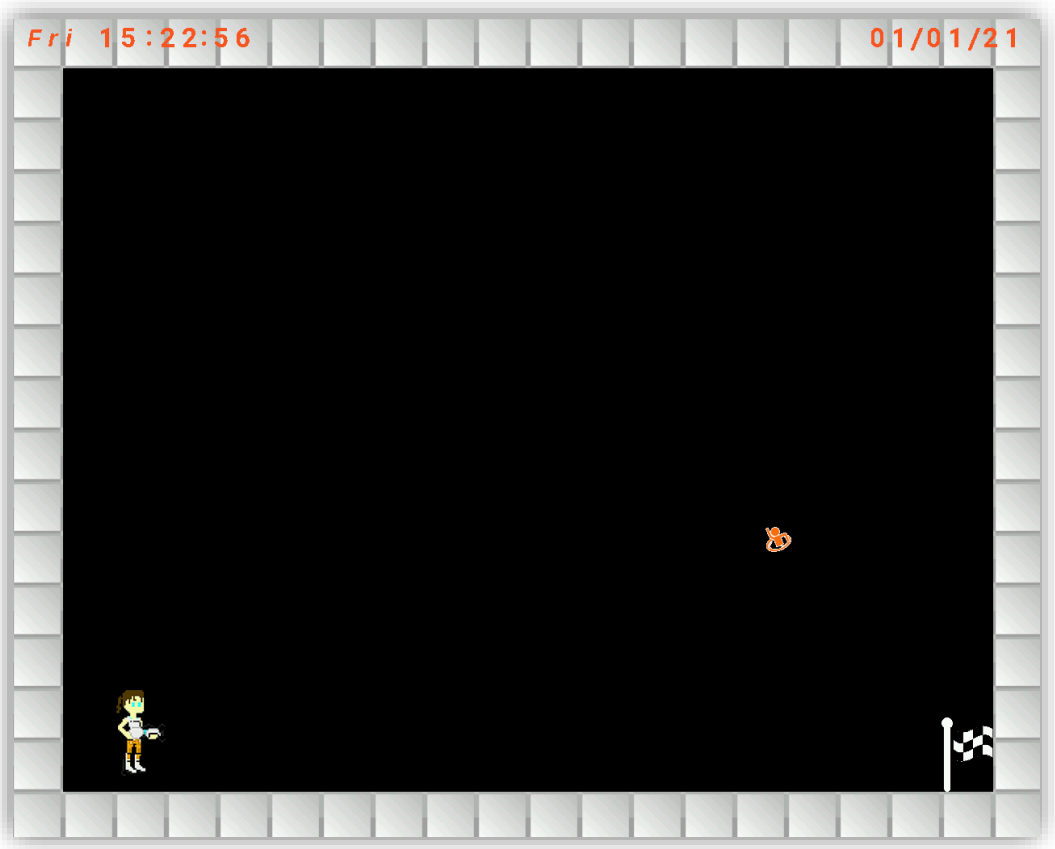
Portanto, neste modo de jogo, a comunicação é muito importante!

Se um dos jogadores desejar sair do jogo, pode, de igual forma ao modo *singleplayer*, carregar na tecla “ESC”. Ambos os jogadores serão retornados ao menu principal.

1.3-Editor de Níveis

Uma das principais funcionalidades do "Portal 2D" é a possibilidade de criar novos níveis dentro do jogo, sem a necessidade de alterar ficheiros ou código, para que o jogador nunca fique sem conteúdo para jogar.

Assim sendo, primeiramente, é carregado uma base simples para o nível, apenas com uma moldura de blocos, um *spawn* e uma bandeira:



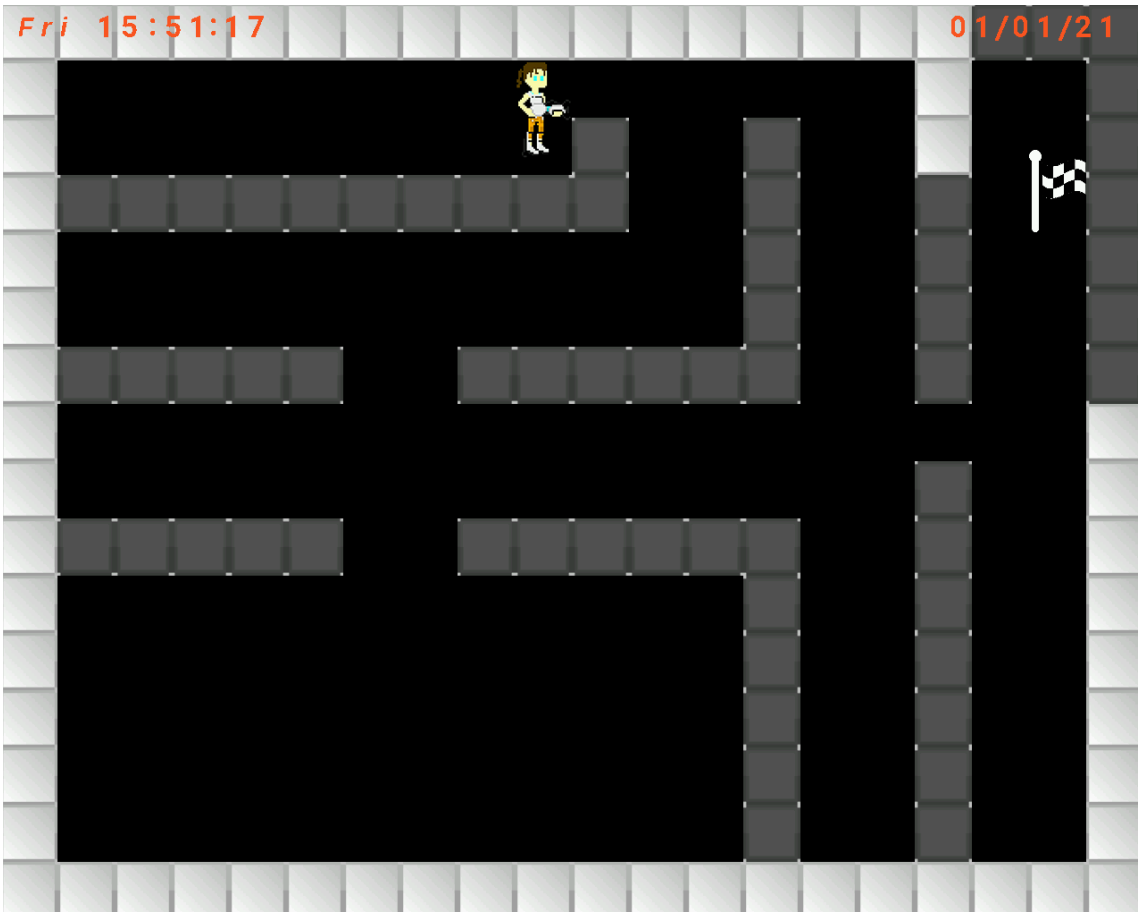
Para a criação do nível, elementos podem ser criados, apagados e alterados, sendo usado uma estrutura de matriz de 20x16 (cada bloco ocupa uma posição), da seguinte forma:

- Clicando no botão esquerdo do rato, será criado um bloco branco e, clicando no botão direito, um bloco preto (se não houver nenhum objeto naquela célula).
- Ambos os tipos de bloco podem ser eliminados clicando na tecla "D", ficando a célula livre.
- Clicando na tecla "S", a posição de *spawn* será alterada, se a nova posição for válida. O *spawn* é o sítio onde o jogador começa o jogo.
- Clicando na tecla "E", a posição da bandeira será alterada, se a nova posição for válida.

A posição da célula a ser alterada é dada pela posição do cursor do rato.

Quando o jogador estiver satisfeito com o seu nível, pode carregar na tecla “ESC”, guardando o nível num ficheiro de texto e voltando para o menu principal. Este poderá agora ser jogado nos modos *singleplayer* e cooperativo, sendo anexado ao agora penúltimo nível.

O utilizador será responsável por criar um nível possível de ser resolvido e que não afete negativamente a jogabilidade do jogo (como por exemplo, prender o jogador num local impossível de sair).



(Exemplo de nível feito no editor)

2- Estado do Projeto:

A tabela seguinte contém a informação resumida sobre os periféricos usados, para que foram usados e se foram usadas interrupções ou não. O ciclo *driver_receive()* está presente na função *main_loop()*, sendo que a configuração e "fecho" destes dispositivos se encontram nas funções *start_loop()* e *end_loop()*, respetivamente.

Periférico	**Utilização**	**Interrupções**
Timer	Controlo de *frames* do jogo, atualizando o ecrã. Mantém a lógica do jogo a uma *framerate* constante	Sim
Keyboard	Interação com o jogo (controlar a personagem e desenhar níveis)	Sim
Mouse	Controlo do cursor e interação com o jogo (opções no menu, criação de portais, desenho de níveis, etc.)	Sim
Video Card	Mostrar no ecrã os elementos do jogo/editor, opções no menu e data/hora	N/A
RTC	Consulta da data e hora atuais, mostrando-as no ecrã e atualizando-as de acordo com as interrupções	Sim
Serial port	Comunicação entre dois jogadores no modo cooperativo	Sim

2.1- Timer:

O timer é responsável por controlar os *frames*/*ticks* do jogo e por manter a lógica do jogo a uma *framerate* constante, através das suas interrupções (usando a sua frequência padrão, 60 interrupções por segundo), de 60 fps. Este é também responsável pela animação de sprites animadas, pela gravidade do jogo e pelo tratamento consistente do *page flipping* (mais informações na secção 2.4).

Cada interrupção é tratada pela função draw_new_frame(), que tem em conta o estado do jogo (variável global estática do tipo enumerável Game_state)

para decidir o que deve fazer. Esta função chama outras funções importantes para a lógica do jogo, como para verificar colisões (*check_block_colisions(), checkportals() e check_level_end()*), para animar e apagar sprites (*animate_sprite(), animate_asprite(), erase_sprite() e erase_everything()*) e a leitura de novos níveis (*read_level_from_file()*).

2.2- Keyboard:

O teclado é usado para controlo da personagem, nos modos *singleplayer* e cooperativo, para desenhar níveis no modo editor e para voltar ao menu principal.

Cada interrupção é tratada pela função *kbc_int_handler(),* que lê a informação seguidamente processada na função *kbc_event_handler().* Nos modos de jogo principal (*play* e *co-op*), esta função chama ainda a *key_char_event()*, que atualiza os parâmetros de uma personagem de acordo com o evento ocorrido. No modo editor, a função invoca a *editor_draw_element()*, que adiciona à matriz o elemento adequado de acordo com a tecla pressionada.

2.3- Mouse:

O rato é utilizado em todos os modos de jogo (incluindo o menu), usando tanto os botões como o seu movimento, seja para escolher opções no menu, disparar portais ou desenhar elementos do nível.

Cada interrupção é tratada pela função *mouse_int_handler().* Se for recebido um *packet* completo, é então chamada a função *mouse_event_handler()* (que tem como um dos argumentos uma variável do tipo enumerável mouse_event). Esta função atualiza a posição do cursor, no caso de evento de movimento, e pode ainda chamar as funções *shoot_portal(), editor_draw_element()* e *menu_change(),* dependendo do estado do jogo e do tipo de evento.

2.4- Video Card:

A placa gráfica é utilizada com o modo direto **0x11A**, com resolução **1280x1024** e **16 bits** por pixel, utilizando **RGB 5:6:5**. No projeto, usou-se *double buffering* com *page flipping,* objetos animados, fontes (para desenho das horas e da data) e três funções da VBE: 0x01 (obter as informações do modo VBE), 0x02 (escolher o modo VBE) e 0x07 (ativar *page flipping* com *vertical tracing).*

A iniciação do modo gráfico e a alocação de memória para três *buffers* são feitas na função *vg_init()* (dois *buffers* principais que vão alternando com *page flipping, vídeo_mem e aux_buff,* e um *buffer* para conter o fundo do jogo, *background,* que dispensa a necessidade de desenhá-lo em todos os *frames*, causando uma melhoria significativo na performance do jogo). Esta função

também chama *vbe_get_mode_info2(),* desenvolvida no projeto, para obter as informações do modo de vídeo.

Para a troca de *buffers* (e consequente troca de *frame* a aparecer no ecrã), é usada a função *buffer_screen()*. A função *flip_page()* também é importante, pois muda o *buffer* em que programa está a escrever. No fim do programa, é invocada a função *vg_exit_program(),* que liberta os *buffers* e retorna o Minix ao modo de texto padrão.

Para criar um aspeto apelativo ao programa, foram usadas *sprites* e fontes encontradas na *web* (com devidas licenças de utilização), ligeiramente modificadas para este projeto, que podem ser encontradas nas pastas *Chars* e *Sprites* (dentro da pasta *Resources*), usadas durante todo o programa em forma de *Sprites* ou *Animated Sprites*. Assim, conseguiu-se desenhar todos os elementos do jogo, os botões, a data, as horas e os portais animados. No entanto, não foi encontrada uma versão animada (e gratuita) da *sprite* da personagem principal, pelo que esta é estática.

Para apagar as *sprites* e as fontes durante a execução do jogo, é usada a função *vg_draw_from_background(),* que pinta na posição da imagem as cores que estão no *background buffer,* previamente carregado pela função *load_background().*

2.5- RTC:

Do Real-Time Clock, são usadas as funcionalidades de leitura da data e das horas, tal como os *Update Interrupts*, que permitem mostrar e atualizar estes dados ao longo de todo o programa. Para isso, é usada uma fonte dedicada a esta causa, usando um *array* com todos os dígitos, um *array* bidimensional com os dias de semana representados por 3 letras e ainda duas *sprites* para representar os caracteres ‘:’ e ‘/’ (contidos no módulo *font*).

As interrupções do RTC são tratadas pela função *rtc_int_handler(),* que invoca a função *read_time(),* sendo que esta vai ler os registos do relógio e atualizar as variáveis estáticas do módulo. Para os módulos exteriores lerem a data e a hora, usa-se as várias funções *get* do módulo (p.ex. *get_day()*)

2.6- Serial Port:

O serial port é usado no modo cooperativo, para efetuar a comunicação entre dois computadores (jogadores local e remoto). Utilizou-se interrupções para a receção de mensagens e *polling* para o envio destas, sendo que foram usados “acknowledgment bytes” para a deteção de erros durante o envio de mensagens (esta técnica consiste em enviar um dado *byte* cada vez que o computador recebe uma mensagem, para que o outro PC saiba que a mensagem foi recebida com sucesso/insucesso). Para ler esse byte, é usada a função *uart_read_ack().*

A configuração usada na porta série é **8 bits** por caractere, **1 stop bit**, paridade **ímpar** e uma bit-rate de **9600 bits** por segundo. Esta configuração é feita na função *uart_init()*. Escolheu-se esta configuração para tentar que o envio de informação fosse o mais rápido possível, evitando erros de sincronização.

As interrupções são tratadas pela função *uart_int_handler()* (se o segundo PC não estiver no modo cooperativo, a mensagem é descartada), que por sua vez invoca a função *uart_read_data()*, que lê o *byte* recebido, verifica a ocorrência de erros, envia um "Ack/Nack byte" e verifica se a mensagem chegou ao fim, comparando o *byte* recebido com o *byte* de fim de mensagem (todos os *bytes* especiais usados estão no ficheiro protocol.h).

Para enviar uma mensagem, a função *uart_write_message()* é invocada, sendo que o primeiro *byte* da mensagem é constituído por um caractere especial que identifica o tipo de mensagem. Uma mensagem é enviada, durante o jogo, em duas ocasiões: quando o jogador local efetua *input* pelo teclado, movimentando a sua personagem (na função *key_event_handler()*), e quando clica no botão direito do rato, criando um portal (na função *mouse_event_handler()*). Estas mensagens são então lidas pelo jogador remoto, recriando o evento no seu PC (ambos os computadores correm a lógica do jogo). O processamento de uma mensagem é feito na função *uart_event_handler()*.

3-Organização/Estrutura do Código:

Ao lado do seu nome, pode ser encontrado o peso relativo de cada módulo.

3.1- Módulo Timer (5%):

Neste módulo encontram-se as funções para a manipulação do timer, como as funções de *subscribe/unsubscribe* de interrupções e a leitura e escrita de registos.

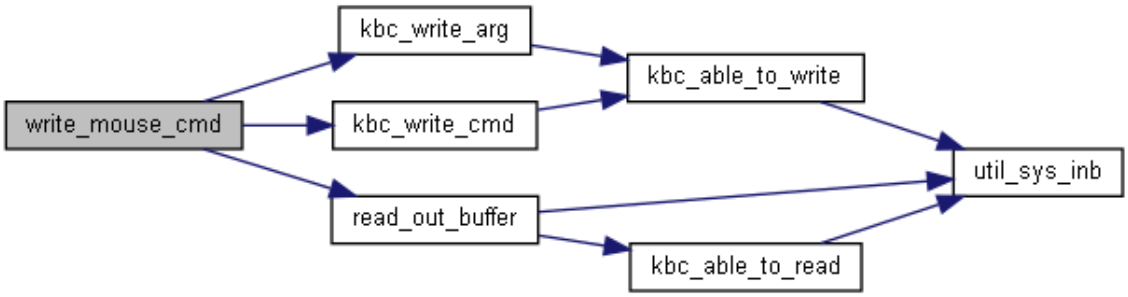
Este módulo também contém o submódulo i8254, que foi fornecido para a realização do Lab 2, que contém as constantes simbólicas para o uso do Timer.

3.2- Módulo Kbc (5%):

Neste módulo encontram-se as constantes simbólicas para o uso do KBC e as funções de manipulação direta deste controlador, como as funções de *subscribe/unsubscribe* de interrupções, o *interrupt handler*, escrita e leitura de registos, tal como funções que permitem aumentar o nível de abstração no uso deste dispositivo (que são também utilizadas no uso do rato), como o *kbc_write_cmd()*, *kbc_write_arg()*, etc.

3.3- Módulo Mouse (7%):

Este módulo contém todas as constantes simbólicas para o uso do PS2 e as funções de manipulação do rato, como as funções de *subscribe/unsubscribe* de interrupções, o *interrupt handler*, escrita de comandos relacionados com o rato (*write_mouse_cmd()*) e a função de construção de uma *struct packet* a partir de um *packet* recebido do rato. Também contém uma *struct mouse_event* que é usada para converter os *packets* numa notação mais legível e fácil de manipular.



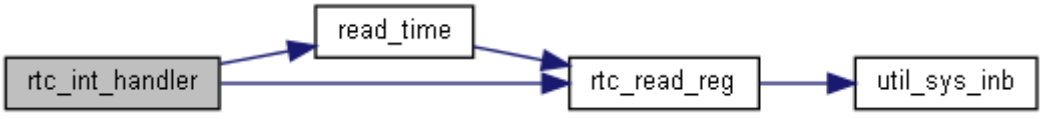
3.4- Módulo Graphics (12%):

Este módulo contém todas as constantes simbólicas para o uso do VBE (incluindo chamadas à BIOS), tal como todas as funções para a manipulação direta da placa gráfica. Entre elas, estão as funções de configuração e "fecho" (neste caso, "fecha-se" a placa gráfica libertando a memória dos *buffers* e retornando ao modo padrão do Minix) da placa, uma função alternativa à *vbe_get_mode_info()* anteriormente fornecida, funções de manipulação de pixels (tanto para desenhar como para apagar) e *buffers*, e funções para alterar o buffer a ser lido/escrito, tirando partido do *double buffering* com *page flipping*.

3.5- Módulo RTC (5%):

Neste módulo encontram-se todas as constantes simbólicas para o uso do Real-Time Clock e as funções responsáveis por o manipular diretamente, tal como funções de *subscribe/unsubscribe* de interrupções, o *interrupt handler* e funções de leitura e escrita de registos.

Este módulo também contém variáveis globais estáticas que guardam a data e hora atuais, que são atualizadas dinamicamente e podem ser acedidas pelas várias funções *get* presentes no módulo.



3.6- Módulo Uart (7%):

Este módulo contém todas as constantes simbólicas para o uso da porta série, tal como as funções para manipulação direta deste dispositivo. Entre elas, encontram-se as funções de *subscribe/unsubscribe* de interrupções, o *interrupt handler,* funções de leitura e escrita de registos, funções de configuração da Uart e funções para envio e recessão de mensagens remotas.

Também é usado neste módulo o submódulo Protocol, que define as constantes simbólicas usadas no protocolo de comunicação do projeto.

3.7- Módulos para Desenho de Imagens e Texto (10%)

3.7.1- Módulo xpm_files (1%):

Neste módulo, são incluídos no projeto todos os ficheiros XPM usados para representar *sprites* e fontes.

3.7.2- Módulo Sprites (4%):

Este módulo define duas estruturas de dados, *Sprite* e *Sprite_type*. A primeira é usada para representar uma imagem, definida por um mapa de pixels carregado a partir do ficheiro XPM respetivo, das suas dimensões, da sua posição e da sua velocidade. Por sua vez, a segunda estrutura de dados é usada para facilmente explicitar o tipo de *sprite* que o jogador deseja desenhar no editor de níveis (ou noutro modo qualquer que possa ser futuramente criado).

O módulo também contém todas as funções de manipulação de *sprites*, ou seja, funções para desenhar, apagar e destruir *sprites*, para animá-las e virá-las (*flip_sprite()*). Estas funções são bastante usadas para mostrar interação no jogo.

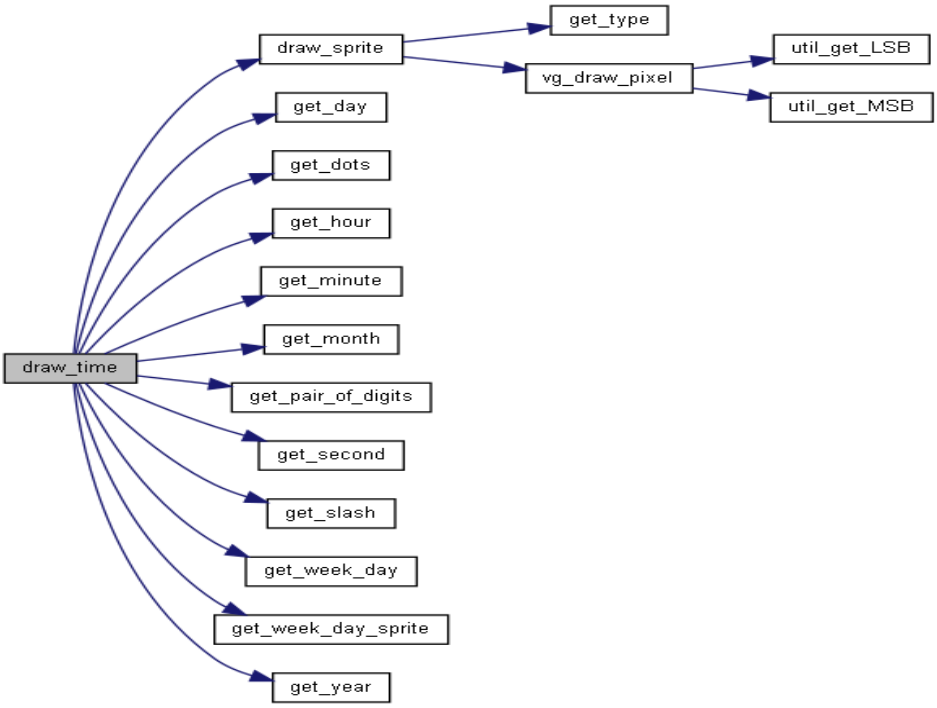
3.7.3- Módulo ASprite (3%):

Este módulo define uma *sprite* animada, ou seja, uma *sprite* que muda automaticamente de imagem a uma dada frequência, de forma a dar a ilusão de animação. Estes objetos são definidos por uma *struct* que contém, além de um apontador para Sprite, um *array* de mapas de pixels, correspondentes às várias imagens que se vão alternando. Tal como no módulo Sprite, este também contém todas as funções para criar, destruir e animar este tipo de *sprite*.

O módulo também contém outra estrutura de dados, *portal_type*, que é usada para explicitar com que tipo de *ASprite* (neste caso, que tipo de portal) estamos a tratar a dado ponto do programa, o que é útil para manipular os atributos da personagem que o jogador controla.

3.7.4- Módulo Font (2%):

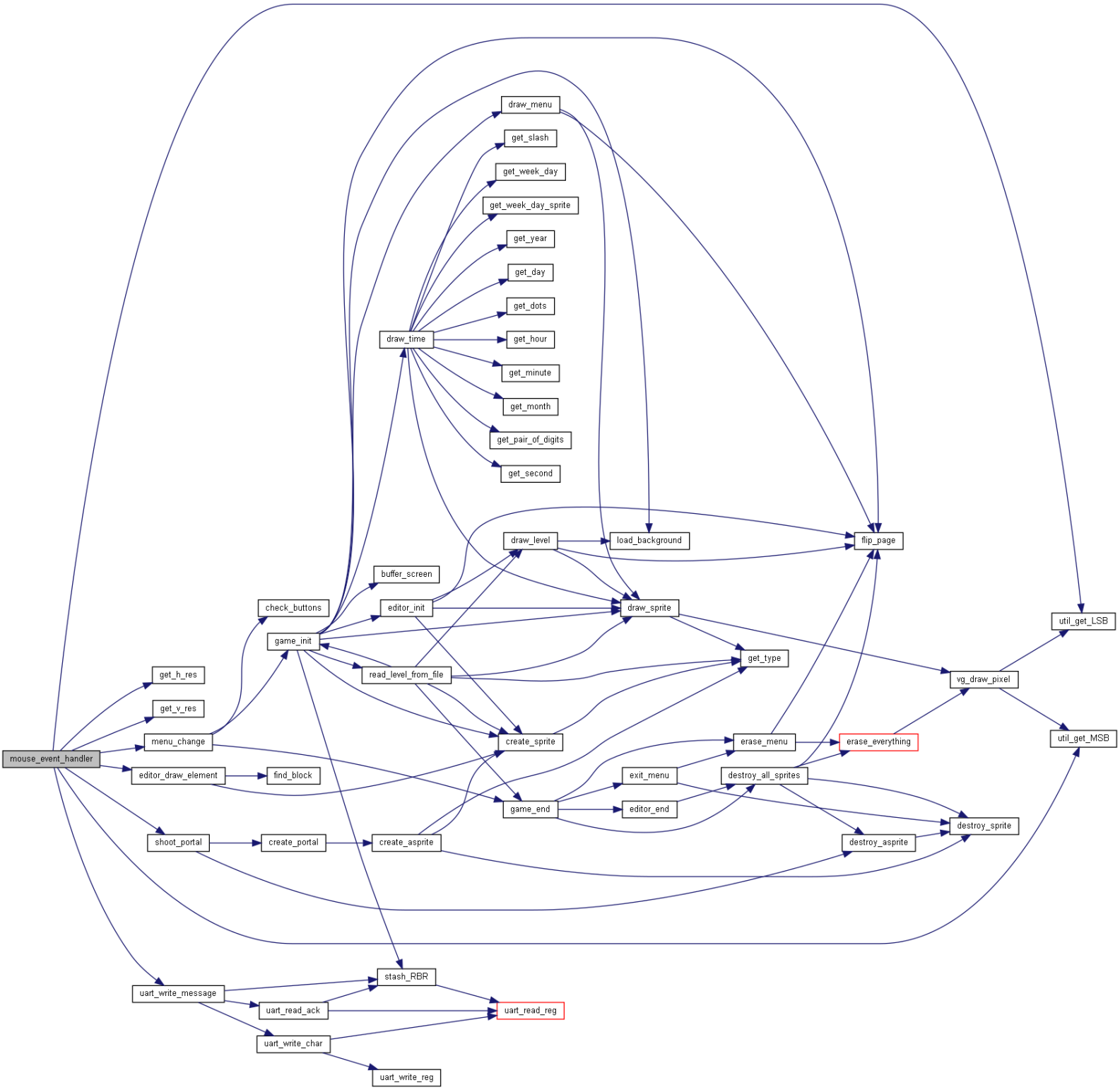
Para representar a data e a hora, foi criado um módulo que define uma fonte especial para este tipo de informação. Assim, este módulo conta *sprites* que representam os dígitos e as letras dos dias da semana (organizadas num *array* bidimensional *week_days*), entre outros caracteres úteis. O módulo conta com funções para inicializar e libertar (a memória) estas variáveis, funções *get* para obter um determinado número ou dia da semana (*get_pair_of_digits()* e *get_week_day_sprite()*) e uma função que desenha o tempo atual no ecrã (*draw_time()*).

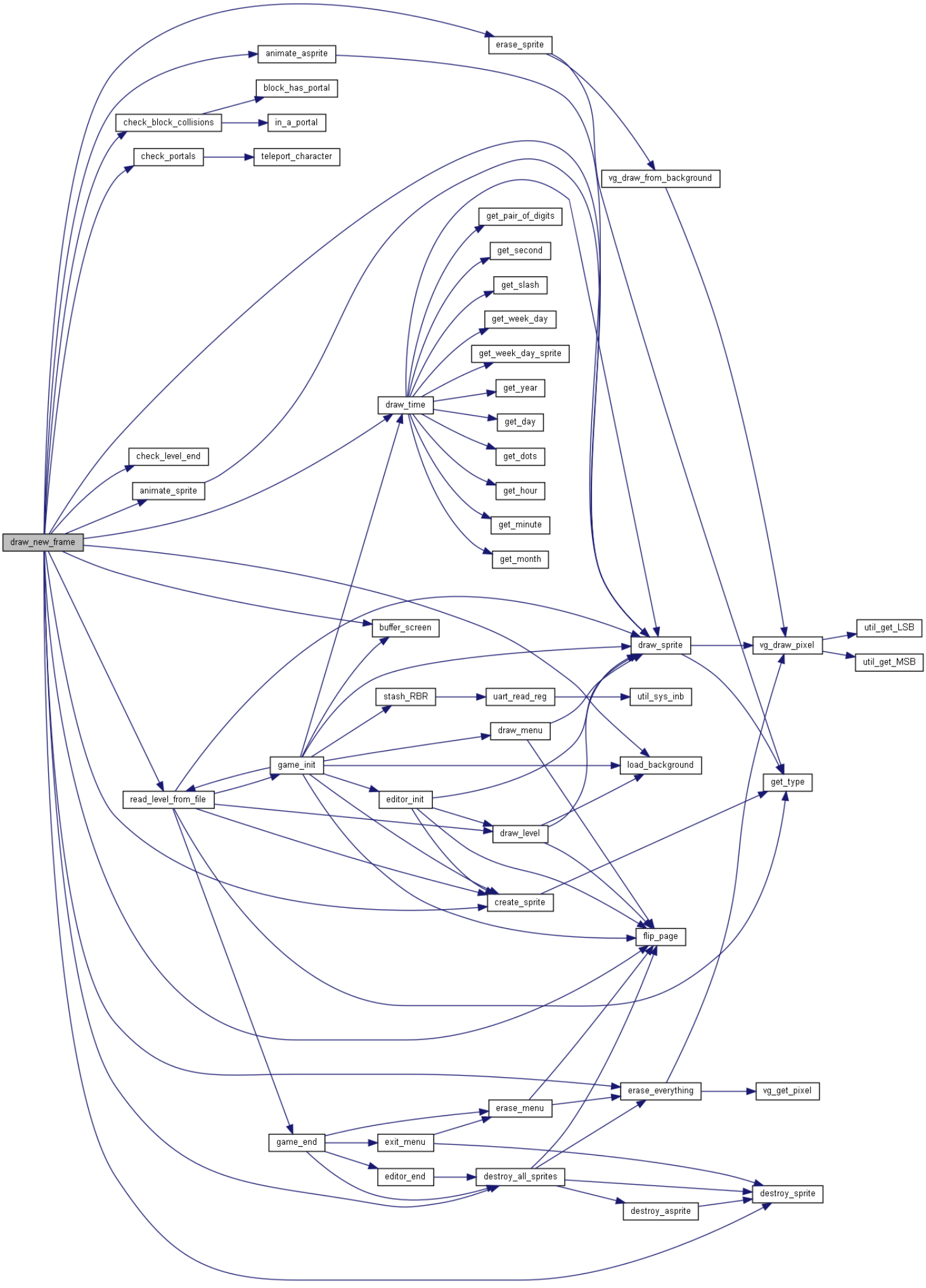


3.8- Módulo Game (40%):

Este é o módulo mais importante do projeto. Trata de toda a lógica do jogo, contém todos os objetos dele e recebe eventos provenientes dos periféricos, convertendo-os em ações e eventos do próprio jogo (p.ex. Recebe um *byte* do teclado e faz a personagem saltar). Este módulo contém uma máquina de estados, explicitada na estrutura enumerável *Game_state*, que possibilita tratar os *inputs* de forma diferente de acordo com o estado do jogo.

De seguida, apresentam-se dois dos diagramas de chamada de funções mais importantes deste módulo:


draw_menu
get_slash
get_week_day
get_week_day_sprite
get_year
get_day
get_dots
get_hour
get_minute
get_month
get_pair_of_digits
get_second
draw_time
draw_level
load_background
flip_page
buffer_screen
editor_init
draw_sprite
check_buttons
game_init
util_get_LSB
get_type
get_h_res
read_level_from_file
vg_draw_pixel
get_v_res
menu_change
create_sprite
erase_menu
erase_everything
util_get_MSB
mouse_event_handler
editor_draw_element
find_block
exit_menu
destroy_all_sprites
game_end
editor_end
destroy_sprite
shoot_portal
create_portal
create_asprite
destroy_asprite
stash_RBR
uart_write_message
uart_read_ack
uart_read_reg
uart_write_char
uart_write_reg


draw_new_frame
erase_sprite
animate_asprite
block_has_portal
check_block_collisions
in_a_portal
check_portals
teleport_character
vg_draw_from_background
get_pair_of_digits
get_second
get_slash
get_week_day
get_week_day_sprite
get_year
draw_time
get_day
get_dots
get_hour
get_minute
get_month
check_level_end
animate_sprite
buffer_screen
draw_sprite
vg_draw_pixel
util_get_LSB
util_get_MSB
stash_RBR
uart_read_reg
util_sys_inb
draw_menu
game_init
load_background
get_type
read_level_from_file
editor_init
draw_level
create_sprite
flip_page
erase_everything
vg_get_pixel
erase_menu
game_end
exit_menu
editor_end
destroy_all_sprites
destroy_sprite
destroy_asprite

3.9- Módulo Menu (3%):

Este módulo é responsável por apresentar ao utilizador as escolhas de modos de jogo disponíveis e por retornar informação sobre o botão que este escolheu. Assim sendo, o módulo contém funções para inicializar e destruir as *sprites* do menu, para as desenhar e para verificar se o utilizador tem o cursor em algum botão, explicitando qual (*check_button()*).

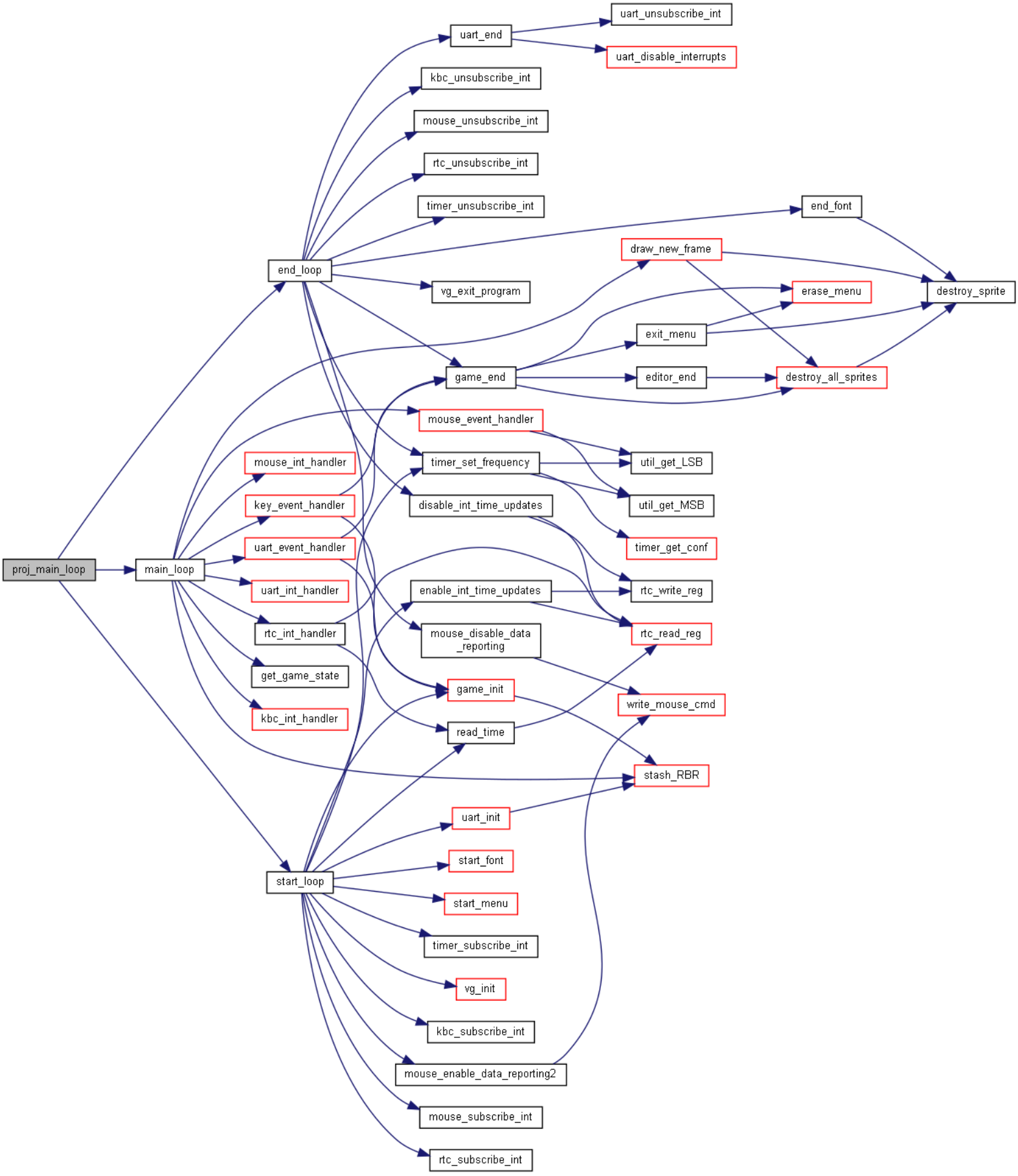
3.10- Módulo Utils (1%):

Este módulo foi desenvolvido no Lab2 e conta com alguma funções úteis ao longo do projeto, nomeadamente, *util_sys_inb(), util_getLSB() e util_get_MSB()*.

3.11- Módulo Loop (5%):

Este módulo é responsável pela estrutura geral do programa. Assim sendo, contém unicamente três funções (chamadas pelo *proj_main_loop()*):

- *start_loop()*- Subscreve as interrupções necessárias de todos os periféricos, configura-os com as definições apropriadas, inicializa todos os objetos necessários ao longo do programa (*start_font(), start_menu()* e *game_init()*) e coloca o Minix em modo gráfico.
- *main_loop()*- Função que contém o ***driver_receive()***, pelo que é responsável por chamar os *interrupt handlers* apropriados cada vez que é recebida uma interrupção. Em alguns casos, também chama os *event handlers*.
- *end_loop()*- Anula as subscrições às interrupções de todos os dispositivos, voltando a pô-los no modo padrão, liberta toda a memória usada durante o jogo e volta a colocar o Minix em modo de texto.


proj_main_loop
main_loop
end_loop
uart_end
uart_unsubscribe_int
uart_disable_interrupts
kbc_unsubscribe_int
mouse_unsubscribe_int
rtc_unsubscribe_int
timer_unsubscribe_int
vg_exit_program
end_font
draw_new_frame
destroy_sprite
erase_menu
exit_menu
game_end
editor_end
destroy_all_sprites
mouse_event_handler
timer_set_frequency
util_get_LSB
util_get_MSB
disable_int_time_updates
timer_get_conf
mouse_int_handler
key_event_handler
uart_event_handler
uart_int_handler
rtc_int_handler
get_game_state
kbc_int_handler
enable_int_time_updates
rtc_write_reg
rtc_read_reg
mouse_disable_data_reporting
game_init
read_time
write_mouse_cmd
stash_RBR
start_loop
uart_init
start_font
start_menu
timer_subscribe_int
vg_init
kbc_subscribe_int
mouse_enable_data_reporting2
mouse_subscribe_int
rtc_subscribe_int

4-Detalhes de Implementação:

No desenvolvimento do projeto, tal como foi recomendado durante os Labs, foi tida em conta uma organização por camadas, de modo a que código ficasse mais organizado e para facilitar a construção do projeto. Assim sendo, existe um conjunto de funções destinadas a interagir diretamente com os dispositivos (configurar dispositivos, ler e escrever em registos, etc.), outro conjunto com o objetivo de usar as funções anteriores para criar abstrações, tais como eventos (mouse, uart, etc.), verificações do estado dos dispositivos ou envio de comandos a um nível mais alto (p.ex. Os comandos do rato exigem várias leituras e escritas em registos mas foi criada uma função que apenas necessita de um comando como argumento). Há também uma terceira camada de funções que usa as anteriores como abstrações, de modo a desenvolver o jogo num nível superior e mais simplificado (há várias funções mas um bom exemplo seria o *draw_sprite()*, que nos permite desenhar um objeto de uma forma simples).

Foi também usada uma máquina de estados (existem várias mas esta é a principal que trata do estado do programa), sendo o código orientado a eventos. Isto quer dizer que quando ocorre um evento (interrupção do teclado, do rato, etc.), este é analisado e, além de atualizar os parâmetros dos objetos, também pode mudar o estado do programa. Estes eventos são, na sua maioria, criados pelos *interrupt handlers*, mas estes últimos são independentes do programa, pois a análise dos eventos é feito à parte dos *handlers*, pela lógica do jogo.

O projeto também é influenciado pelo paradigma da programação orientada a objetos. O jogo é constituído por vários objetos que podem ser criados e destruídos em vários instantes da execução do programa (sendo a maioria representados por Sprites e ASprites). Além disso, a maioria dos módulos restantes representam objetos que "nascem e morrem" com o módulo Loop (estão ativos até ao fim da execução do programa).

Relativamente à criação de *frames*, foi usada uma estratégia que pode ser resumida em: os eventos são gerados e tratados aquando uma interrupção, atualizando os parâmetros dos objetos necessários, mas sem escrever em nenhum dos *buffers.* Quando ocorre uma interrupção do Timer (sendo que ocorre 60 vezes por segundo), são chamadas as funções que tratam da lógica do jogo periodicamente, podendo alterar os objetos (funções que vêm colisões, gravidade, animação de *sprites*, carregamento de níveis, etc.) e os objetos do jogo (com movimento) são desenhados num buffer auxiliar e de seguida usa-se a técnica de *page flipping* para esse *buffer* ser mostrado no ecrã, tornando-se o principal. Depois disso, é necessário apagar os objetos desatualizados do *buffer* agora auxiliar, sendo que o *background* nunca é apagado nem redesenhado, de forma a estar pronto para o próximo *frame* num tempo curto.

Para estabelecer comunicação entre dois PCs, foi usada a porta série UART. Para isto ser possível, foi desenvolvido um protocolo de comunicação, constituído por um byte de cabeçalho que identifica o tipo de mensagem recebido, pelos dados da mensagem (sendo que o tamanho varia com o tipo de mensagem) e por um byte de rodapé, que fecha uma mensagem. Este protocolo permite gerar eventos a partir de mensagens recebidas remotamente. Para tornar o protocolo mais robusto, foi também desenvolvido um sistema de *acknowledgment bytes*, semelhante ao usado pelo PS2. Assim, quando uma máquina recebe uma mensagem, envia à outra uma mensagem ACK (*acknowledged byte*) para comunicar que a mensagem foi recebida com sucesso. Se, pelo contrário, houver uma falha na recessão da mensagem, é enviado um NACK (*not acknowledged byte*) para solicitar o reenvio da mensagem completa. Se a mensagem não conseguir ser enviada após 4 tentativas (com delay de 10ms) é escrita uma mensagem de erro e a mensagem é descartada.

No modo cooperativo, ambos os programas correm a lógica do jogo, usando as mensagens recebidas da porta série para gerar eventos e enviando os dados necessários para que a máquina remota possa gerar esses mesmo eventos.

Tal como foi explicitado na secção 3, foram implementadas algumas sugestões feitas pelo professor, tais como *sprites* estáticas e animadas e uma fonte, tendo esta sido feito com a mostragem da data e das horas em mente.

Também foram desenvolvidos alguns conceitos que não foram abordados nas aulas de LCOM:

- Em várias ocasiões, é necessário analisar e detetar colisões entre os objetos, entre elas colisões: entre as personagens (controladas pelo jogador) e os blocos do nível, entre as mesmas personagens e bandeira que marca o fim de um nível e entre elas e os portais (de modo a conseguir invocar a função de teletransporte e fazer com que a personagem ignore as colisões com os blocos, visto que estará prestes a entrar numa parede), entre os portais e os blocos e entre o cursor do rato e os limites da janela e com os botões do menu. Também foi preciso detetar possíveis colisões no editor de níveis, para impedir o utilizador de sobrepor objetos.
- Para ser possível disparar portais (usando a *portal gun* que a personagem segura nas mãos), foi necessário implementar um algoritmo de lançamento de projéteis. Neste caso, os projéteis (portais) têm uma trajetória linear desde a posição da personagem em direção ao cursor do rato, até atingir um bloco ou sair do mapa. Um portal será criado apenas se atingir um bloco branco não isolado, tendo cuidado para que o portal fique numa posição que faça sentido (visto que o portal ocupa o dobro da largura de um bloco, é necessário haver dois blocos para o colocar, sendo que também não pode ser colocado no canto de um "L", pois não seria acessível).

Para isto ser possível, usou-se trigonometria com coordenadas polares, começando com um raio de 0 e usando o ângulo calculado a partir das duas posições consideradas. De seguida, entra-se no ciclo, aumentando o raio em uma unidade em cada iteração, até que se colida com um bloco ou se saia dos limites da janela.

- Tendo em conta a natureza do jogo, também foi necessário usar uma técnica para o teletransporte de objetos. Como estes objetos estarão em movimento, nem sempre é óbvio o que deverá acontecer ao passarem de um portal para outro. Assim sendo, quando atravessa um portal, a posição da personagem muda para o outro portal e a velocidade perpendicular à orientação do portal torna-se na soma das velocidades horizontal e vertical antes de ela entrar no portal, enquanto que a velocidade tangencial ao portal fica nula.
- Para possibilitar a criação de novos níveis, estes são guardados em ficheiros de texto. Assim sendo, quando se carrega um novo nível é necessário processar a informação contida no seu ficheiro, que consta duma matriz 20x16 em que cada caractere mapeia o tipo de objeto na célula, exceto a primeira linha que contém o número de blocos no nível. Os níveis são nomeados por um número, começando no 1 e incrementando esse número cada vez que é criado um nível (se existirem 2 níveis, são chamados “1.txt” e “2.txt”). O número do próximo nível pode ser lido no ficheiro “next.txt”. Quando o utilizador cria um nível, este é guardado com o respetivo nome e o número em “next.txt” é incrementado.

Conclusões:

Concluindo, LCOM foi, na minha opinião, a cadeira mais difícil e trabalhosa que tive até à data. No entanto, penso que transmite alguns conceitos essenciais para a compreensão dos computadores e de programação em baixo nível.

Penso que nas primeiras aulas práticas (principalmente no Lab2), os alunos tiveram um "choque" em relação ao trabalho que lhes foi proposto, sendo a razão principal não entenderem ainda como trabalhar com o ambiente da LCF e do Minix, sendo necessário procurar informação bastante espalhada (transparências, guides, doxygen, informação na *web*, etc.), mesmo entendendo os conceitos teóricos da aula anterior, sendo este problema muitas vezes resolvido perguntando a colegas que já entenderam ou já fizeram a UC. Assim sendo, penso que seria muito útil parte da aula teórica anterior ao Lab2 ser usada para explicar os conceitos básicos da manipulação dos dispositivos em C, tal como saber usar as funções *sys_inb()* e *sys_outb()*, fundamentais no decorrer dos trabalhos. Este ano, apenas tivemos uma aula a mostrar isto depois da primeira aula prática do Lab2, o que achei que foi tarde para muitos alunos.

Acho também que os Labs, visto serem bastante trabalhosos, deviam contar para a avaliação (como em anos anteriores), nem que seja uma pequena percentagem por termos entregado as funções feitas. Da forma que aconteceu este ano, senti que os alunos que gastavam muito tempo com os Labs (para aprenderem os conceitos) não eram suficientemente recompensados. Ainda sobre as avaliações, visto que aconteceram alguns erros durante os testes práticos, achava uma ótima ideia dois (pelo menos) docentes resolverem o teste antes dos alunos. Não sei se isto foi feito ou não, mas achei bem dar a sugestão à mesma.

Não sei até que ponto é possível, mas penso que a informação que nos foi sendo dada está dispersa e organizada duma forma um pouco "rudimentar", pelo que acho que merecia uma reestruturação (a informação em si não é o problema, mas a sua organização). Penso que uma forma de *notebooks* (como os usados pelo docente João Lopes em FPRO) com informação concisa sobre os periféricos e os Labs seria muito boa e até uma folha completa com os comandos do Minix/LCF específicos de LCOM, de forma semelhante à que é dada na UC de MPCP sobre *Assembly*.

Também achei que o uso de um sistema operativo tão específico dificulta a procura de informação na *web*. No entanto, entendo a razão da sua utilização, pelo que não tenho uma sugestão para melhorar.

O último ponto é que achei que a exigência e carga de trabalho da UC não corresponde ao número de créditos que lhe é dada (sendo que deveria valer mais créditos).

No entanto, também acho que LCOM teve vários pontos positivos: ajudou-me a compreender melhor o funcionamento do computador e dos seus periféricos num baixo de nível de abstração; sendo desafiante, também dá um sentimento de satisfação quando conseguimos realizar uma tarefa, e o projeto ajudou a melhor criar abstrações e organizar o código de modo a conseguir continuar a desenvolver novas funcionalidades.

Apêndice- Instruções de Instalação:

Para instalar o jogo, é apenas necessário executar o comando ‘make’ dentro da pasta “Src”. Para o correr, é usado o comando ‘lcom_run proj “path” ‘, onde path é o caminho completo para a pasta “Resources” que se encontra dentro da pasta “Src”, por exemplo, “/home/lcom/labs/proj/Src/Resources”. Se for desejado usar o caminho no exemplo anterior, é também possível executar o comando ‘lcom_run proj “default” ’.

Se ocorrer algum erro, verifique se o caminho da pasta “Resources” está correto e se ela está realmente dentro da pasta “Src”. É recomendável reiniciar o Minix na ocorrência de um erro, visto que o programa foi fechado abruptamente e os dispositivos não puderem ser desconfigurados propriamente.