



**FEUP** FACULDADE DE ENGENHARIA  
UNIVERSIDADE DO PORTO

# PO(N)G

Miguel Norberto Costa Freitas - up201906159  
Joana Teixeira Mesquita - up201907878

Mestrado Integrado em Engenharia Informática e de Computação  
Laboratório de computadores  
Turma 2 - Grupo 1

4 de janeiro de 2020

## Índice

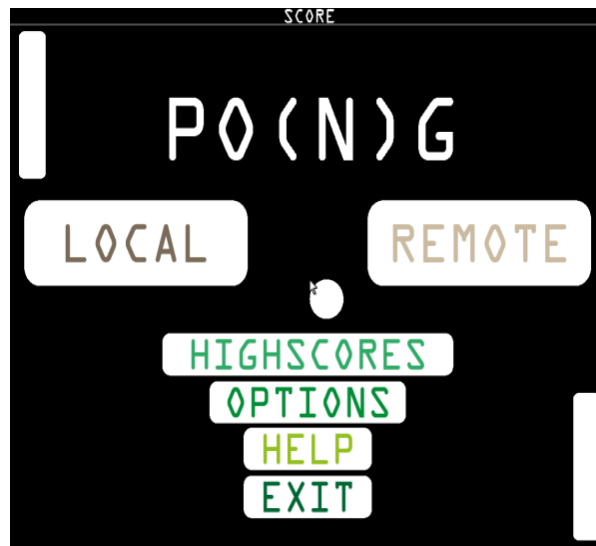
User Instructions .....	4
1.1 Ecrã principal .....	4
1.2 Escolher o tipo de jogo .....	5
1.2.1 Escolher um limite de tempo .....	5
1.2.2 Escolher um limite de pontos .....	6
1.3 Jogo .....	7
1.4 Menu de pausa .....	8
1.5 Menus de final de jogo .....	8
1.6 Highscores .....	9
1.7 Resoluções .....	10
1.8 Outros .....	11
Project Status .....	12
1. Timer .....	12
Funções utilizadas: .....	12
Funções que utilizam dados do timer: .....	13
2. Keyboard .....	13
Funções utilizadas: .....	13
Funções que utilizam dados do keyboard: .....	13
3. Mouse .....	13
Funções utilizadas: .....	13
Funções que utilizam dados do mouse: .....	13
4. Video Card .....	13
Funções utilizadas: .....	14
Funções que utilizam dados da video card: .....	14
5. Real Time Clock .....	14
Funções utilizadas: .....	14
Funções que utilizam dados do rtc: .....	14
Code organization/ structure .....	15
Módulos .....	15
Keyboard .....	15
Mouse .....	15
RTC .....	15
Timer .....	15
Video_gr .....	15

Logic.c .....	16
UI Elements .....	16
Animations/Sprites .....	16
Power ups.....	17
Implementation details .....	17
RTC .....	17
Colisões .....	17
Call Graph .....	18
Sprites .....	19

# User Instructions

## 1.1 Ecrã principal

Começando o programa com o comando `lcom_run proj` traz-nos para o ecrã inicial, em que o utilizador pode seleccionar, usando o cursor, qualquer um dos seguintes botões:



- 1. Local** - Leva o utilizador ao menu de escolha de jogo para iniciar um jogo local entre dois jogadores.
- 2. Remote** - Leva o utilizador ao menu de escolha de jogo para iniciar um jogo de dois jogadores em dois computadores diferentes.
- 3. Highscores** - Abre o menu que contém o registo do highscore atingido por um jogador, com a data em que este foi atingido.
- 4. Options** - Leva o utilizador ao menu em que pode escolher mudar de resolução.
- 5. Help** - Leva o utilizador para um ecrã que comunicaria como controlar cada jogador, para além de conter informação sobre o que cada power up faz.
- 6. Exit** - Termina o programa.

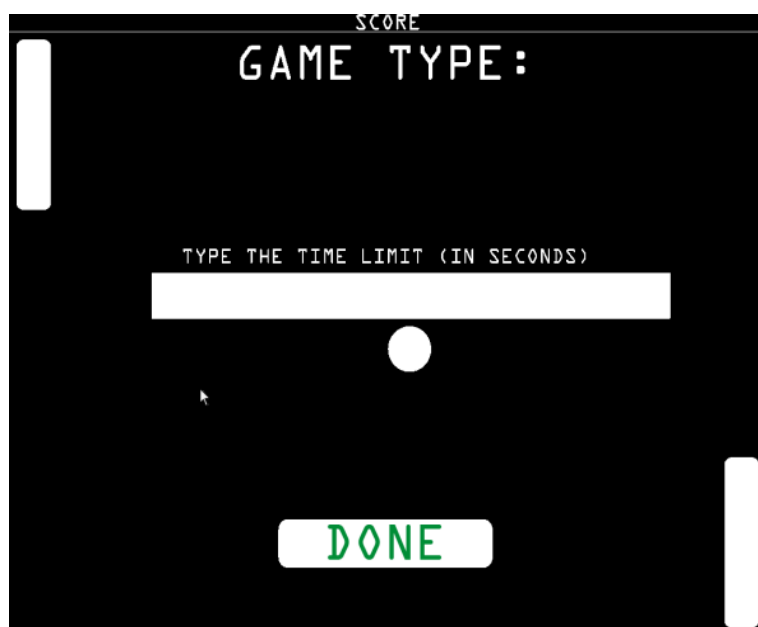
## 1.2 Escolher o tipo de jogo

Após escolher se o jogo é local ou em computadores diferentes, o seguinte menu aparece para o utilizador seleccionar o tipo de jogo que quer fazer.



1. **Endless** - Inicia o jogo, sem qualquer limite de pontos (útil para testar o programa) sendo que a única forma de parar o jogo é entrar no menu de pausa.
2. **Time** - Leva o utilizador para o ecrã que permite escolher o limite de tempo para a partida.
3. **Points** - Leva o utilizador para o ecrã que permite escolher o limite de pontos para a partida.

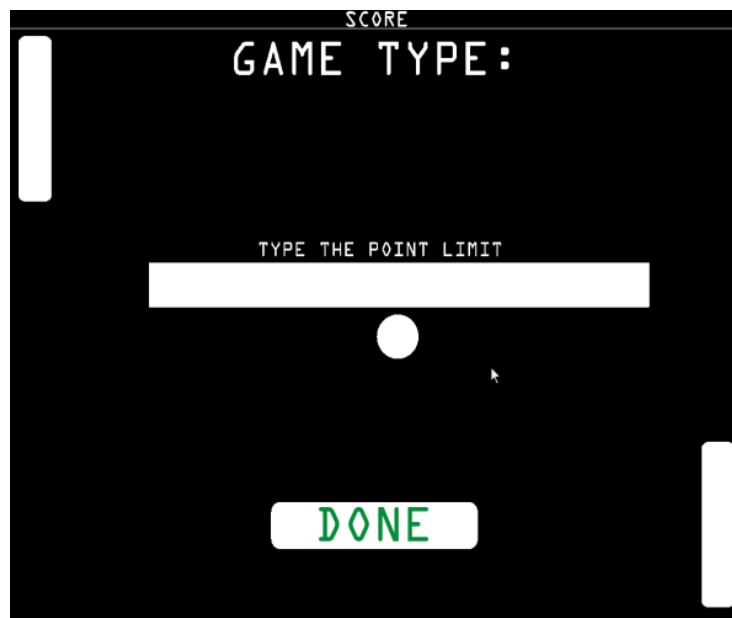
### 1.2.1 Escolher um limite de tempo



**1. Escrever o limite de tempo** - Para escrever o limite de tempo do jogo é necessário clicar com o botão esquerdo na caixa de texto e depois usar o teclado para o escrever. A tecla "backspace" pode ser usada para apagar caracteres já escritos.

**2. Done** - Regista o valor escrito na caixa de texto como o tempo limite do jogo e envia-nos para o ecrã do jogo.

### 1.2.2 Escolher um limite de pontos

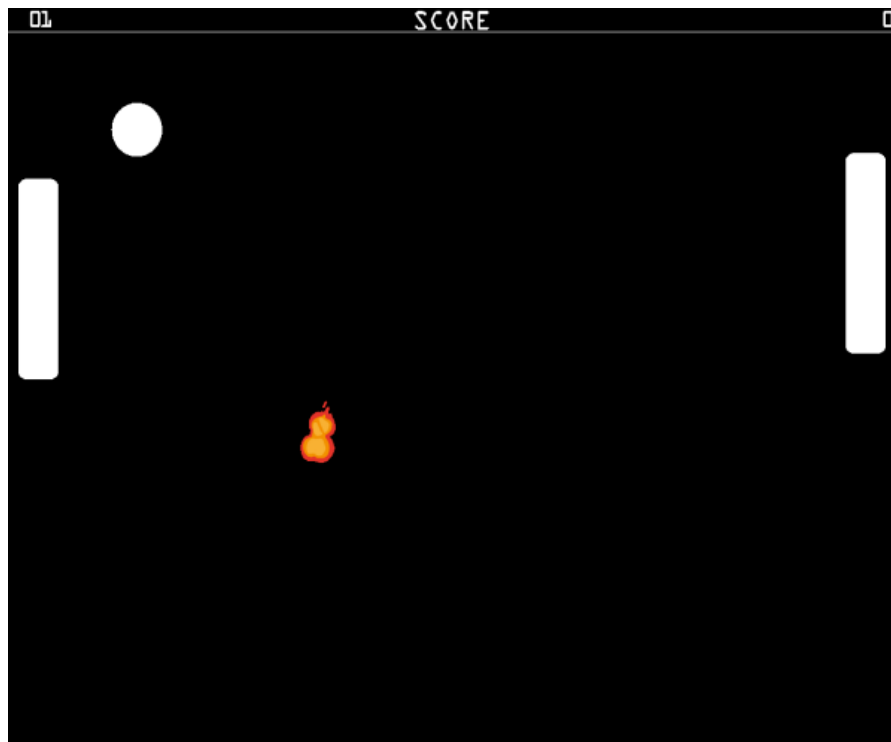


**1. Escrever o limite de pontos** - Para escrever o limite de pontos do jogo é necessário clicar com o botão esquerdo na caixa de texto e depois usar o teclado para o escrever. A tecla "backspace" pode ser usada para apagar caracteres já escritos.

**2. Done** - Regista o valor escrito na caixa de texto como o número de pontos limite do jogo e envia-nos para o ecrã do jogo.




## 1.3 Jogo

Após escolher o tipo de jogo e o limite de pontos ou de tempo, se necessário, o jogo em si é iniciado.



Durante o jogo o jogador 1 (do lado esquerdo) poderá mover-se com o teclado em que a tecla 'w' o faz mover para cima e a tecla 's' o faz mover para baixo e o jogador 2 (do lado direito) poderá mover-se usando o rato sendo que este se moverá para cima e para baixo com o rato.

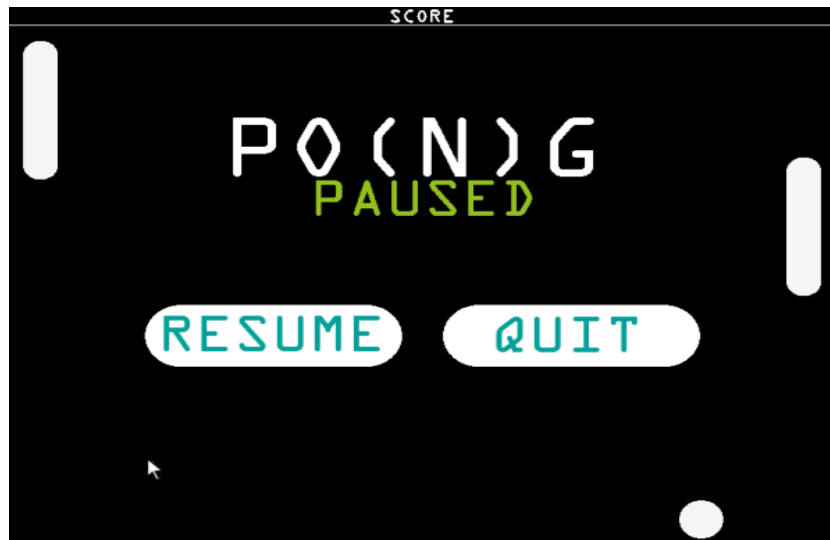
Os jogadores podem ainda tentar apanhar power ups que irão aparecer em posições aleatórias no campo de jogo sendo que estes têm diferentes efeitos.

	Quando atingido pela bola aumenta a sua velocidade.
	Quando atingido, diminui o tamanho do jogador que não o atingiu, pondo-o em desvantagem em relação ao jogador que atingiu o power up.
	Quando atingido aumenta o tamanho do jogador que o atingiu.

Por fim, pode ainda ser pressionada a tecla 'q' que levará o utilizador para o ecrã de pausa.

## 1.4 Menu de pausa

Este menu pode ser aberto ao clicar na tecla 'q' durante um jogo (independentemente do modo de jogo escolhido)



1. **Resume** - Retorna ao jogo.

2. **Quit** - Volta para o menu principal.

## 1.5 Menus de final de jogo

Se for escolhido um jogo com limite de pontos ou limite de tempo, quando esse jogo terminar aparecerá um menu de fim de jogo.

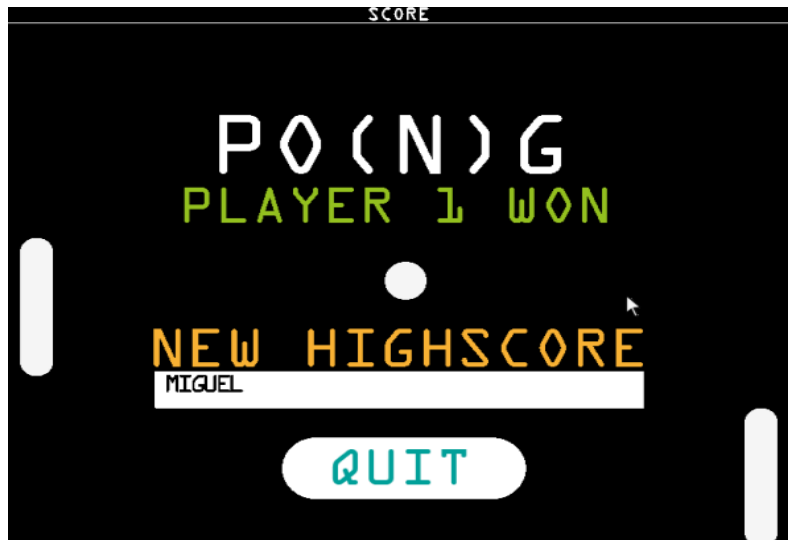
Este menu indicará o jogador vencedor (ou em caso de empate que os jogadores empataram).



1. **Quit** - Volta para o menu principal.



Caso algum dos jogadores tenha atingido um novo highscore vai ser pedido que este introduza um nome para aparecer no menu de highscores.

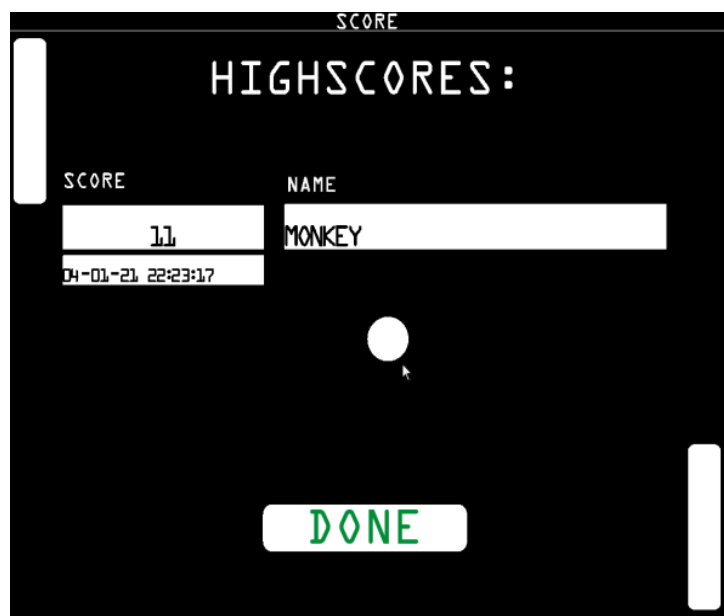


**1. Escrever o nome** - Para o jogador escrever o seu nome é necessário clicar com o botão esquerdo na caixa de texto e depois usar o teclado para o escrever. A tecla "backspace" pode ser usada para apagar caracteres já escritos.

**2. Quit** - Volta para o menu principal.

## 1.6 Highscores

A partir do menu inicial pode aceder-se ao menu dos "highscores". Este menu permite-nos ver a pontuação máxima atingida no jogo, por um jogador, bem como o seu nome, a data e hora a que essa pontuação foi atingida.



**1. Done** - Volta ao menu principal.

## 1.7 Resoluções

Este menu permite ao utilizador escolher a resolução na qual quer jogar o jogo.



1. **640 x 480** - Muda o jogo para o modo de vídeo 0x110.
2. **800 x 600** - Muda o jogo para o modo de vídeo 0x115.
3. **1280 x 1024** - Muda o jogo para o modo de vídeo 0x11A.
4. **1152 x 864** - Muda o jogo para o modo de vídeo 0x14C.
5. **Done** - Volta ao menu principal

## 1.8 Outros

Em qualquer um dos menus anteriores, mudará também a cor de qualquer botão quando o cursor se encontrar por cima dele.



# Project Status

Device	What for	Int.
Timer	Controlar o frame rate. Controlar o tempo para o jogo com limite de tempo. Controlar o tempo em que os power ups estão ativos.	Yes
Keyboard	Controlar o jogador 1. Escrever o limite de tempo do jogo num jogo por limite de tempo. Escrever o limite de pontos do jogo num jogo por limite de pontos. Escrever o nome quando o jogador realiza um highscore.	Yes
Mouse	Controlar o jogador 2. Cursor para seleccionar botões no menu e navegá-los.	Yes
Video Card	Iniciar o modo de vídeo no início do programa. Alterar o modo de vídeo quando o utilizador clica no botão correto.	N/A
Real Time Clock	Guardar a data e hora de um novo highscore.	Yes

Para além destes devices, inicialmente, queríamos implementar também a porta de série para realizar jogos em duas máquinas virtuais diferentes, no entanto acabámos por não o conseguir fazer pelo que o botão remote apresentado na secção anterior não tem de momento qualquer uso.

Em adição, o botão "Help" também não está funcional apenas porque não conseguimos fazer as sprites para esse menu, a tempo.

## 1. Timer

No timer foram utilizados os seus interrupts para desenhar todos os sprites a um rate constante, para controlar o frame rate das animações (os power ups que incluímos têm um frame rate de 6 frames por segundo) e para controlar o tempo quando necessário.

Para todos os sprites serem desenhados simultaneamente, todas as funções que desenhavam sprites foram colocadas diretamente dentro do "if" que recebe os interrupts do timer (if (msg.m\_notify.interrupts & irq\_set\_timer)) fazendo assim com que todos estes sejam desenhados constantemente a cada interrupção do timer.

Para controlar o frame rate das animações foi usado o valor de um counter\_timer que é incrementado a cada interrupção do timer e, conforme o frame rate da animação e a frequência do timer, as imagens foram desenhadas nos intervalos de tempo certos. Para isto a struct Animation guarda o valor do counter\_timer no momento em que é chamada a animação para saber exatamente quando deverá mudar de frame.

### Funções utilizadas:

- timer\_int\_handler()

Funções que utilizam dados do timer:

- `handle_animation(struct Animation* anim, int ticks_to_wait, bool reversed);`
- `set_animation_initial_frame(struct Animation* anim, int counter);`
- `handleRemovePowerUp();`

## 2. Keyboard

O teclado foi utilizado tanto para o controlo de um jogador, como para pedir input do utilizador em contextos diferentes, tendo sido utilizados scancodes para todas as letras (exceto o ç) e números assim como o scancode da tecla “backspace”.

Funções utilizadas:

- `kbd_ih();`
- `keyboard_print_scancode();`

Funções que utilizam dados do keyboard:

- `control_player1();`
- `realTimeWritting();`

## 3. Mouse

O rato foi utilizado em dois contextos distintos. Primeiramente foi usado para controlar o jogador dois através do seu valor de movimento relativo em y. Depois, foi ainda utilizado nos menus onde controla o movimento de um cursor através do seu movimento relativo em x e em y, detetando ainda clicks do botão esquerdo, permitindo assim seleccionar botões do menu (usado diretamente no `int(proj_main_loop)(int argc, char *argv[])` através de um “if (packet1.lb)”).

Funções utilizadas:

- `kbc_ih();`
- `assemble_mouse_packet();`

Funções que utilizam dados do mouse:

- `control_player2();`
- `void mouse_movement();`

## 4. Video Card

Foram usados quatro dos cinco modos de vídeo da video card, 0x110 (resolução 640x480 com 15 bitd ) , 0x115 (resolução 800x600 com 24 bits), 0x14c (resolução 1280x1024 com 32 bits) e 0x11a (resolução 1152x864 com 16 bits), todos modos de cor direta, sendo que este pode ser escolhido pelo utilizador.

Foi utilizado um método de double buffering (para desenhar os sprites no ecrã, estes consistem de figuras geométricas simples, sprites complexos, letras, números e frames de animações (guardadas numa struct apropriada).

As colisões são detetadas de maneira simples quando pelo menos um pixel de dois objetos coincide.

#### Funções utilizadas:

- `vg_init();`
- `vg_exit();`
- `double_buffer();`
- `vg_draw_pixel();`
- Outras funções no ficheiro `video_gr.c` que desenhavam estruturas diferentes no ecrã;

#### Funções que utilizam dados da video card:

- `handle_animation(struct Animation* anim, int ticks_to_wait, bool reversed);`
- `set_animation_initial_frame(struct Animation* anim, int counter);`
- outras funções que desenhavam alguma coisa no ecrã;

## 5. Real Time Clock

O real time clock foi utilizado em modo de update para registar a data e hora de quando ocorre um highscore.

#### Funções utilizadas:

- `rtc_get_date();`

#### Funções que utilizam dados do rtc:

- `write_highscore_to_file();`
- `write_date_to_screen(struct Date date, int x, int y, int color);`

# Code organization/ structure

## Módulos

### Keyboard

Não difere muito da versão implementada no lab3, sendo capaz de ler scancodes do KBC, capaz de escrever comandos para o KBC e possui uma função para limpar o seu buffer no final da execução para o Minix não ficar preso.

Desenvolvido por: Miguel Freitas / Joana Mesquita

### Mouse

Mais uma vez, as suas funcionalidades são as que já tinham sido implementadas no lab4. É capaz de enviar makecodes e breakcodes, fazer “assemble” de um mouse packet para a struct Packet, escrever argumentos para o buffer do mouse.

Desenvolvido por: Miguel Freitas / Joana Mesquita

### RTC

Quanto ao RTC apenas nos interessou a possibilidade de ler a data atual para a armazenar num ficheiro .txt com os highscores, para depois ser mostrada no programa, daí que essa foi a única função implementada (`rtc_get_date()`). Tal função lê a data dos vários registos do RTC e coloca-a numa struct Date para depois ser escrita de uma maneira legível por humanos. Ainda foi implementado o interrupt handler do RTC, que apenas lê o register C para

Desenvolvido por: Miguel Freitas/ Joana Mesquita

### Timer

Não possui mais funcionalidade que aquelas já implementadas no lab2, sendo, portanto, capaz de alterar a frequência do Minix e ler a configuração atual do timer, entre outras.

Desenvolvido por: Miguel Freitas / Joana Mesquita

### Video\_gr

Possui as funcionalidades implementadas no lab5, se bem que algumas das funções desenvolvidas foram adaptadas para desenhar sprites específicas do nosso projeto. Implementamos uma função que escreve uma string para o ecrã (dá assemble dos xpm's correspondentes a cada letra da string numa posição variável do ecrã), sendo que possuímos uma função semelhante para escrever números para o ecrã. O nosso módulo do video\_gr permite ainda chamar `vg_init()` no meio da execução do programa, alterando a resolução do ecrã e fazendo load de novas sprites para a nova resolução.

Foi ainda desenvolvida a nossa própria versão da função `vbe_get_mode_info()`, chamada `vbe_get_mode_info_custom()`.

Desenvolvido por: Miguel Freitas/Joana Mesquita

## Logic.c

Trata tudo que tem a ver com o jogo de Pong em si, com structs para o jogo, o game state, etc. Trata de verificar colisões da bola com o jogador, atualizando a direção da bola de acordo com a parte da paddle do jogador com a qual colidiu. De notar que possui uma função que trata do edge case em que a bola está por cima ou por baixo de um dos players e fica presa devido ao jogador a pressionar contra as paredes do Minix. Essa função reverte a posição do player e desprende a bola para não criar uma situação em que a bola fica eternamente presa num canto do jogo. Este módulo é ainda responsável por tratar dos menus do jogo, na função `handle_menu_events()` que muda de menu/ começa o jogo dependendo do input do mouse do utilizador. (Mudar de menu no nosso projeto implica copiar os botões do próximo menu para o array `CurrentMenu` que é depois escrito para o ecrã).

Structs:

Player - struct que representa um player contendo as suas possíveis sprites, informação sobre que sprite deve escrever agora no ecrã e o seu score.

Ball - struct que representa a bola do jogo, armazenando a sua velocidade atual, velocidade no eixo dos x e dos y e a sua struct `Sprite`.

Desenvolvido por: Miguel Freitas/Joana Mesquita

## UI Elements

Este módulo é apenas responsável por realizar os `xpm_load()` dos vários sprites que têm uma variação para cada modo usado. Decidiu-se criar um módulo diferente para o código ficar mais organizado.

Desenvolvido por: Miguel Freitas/Joana Mesquita

## Animations/Sprites

Lida com a implementação de animações e sprites no nosso projeto, com a função `handle_animation()` a escrever o frame correspondente a um intervalo variável na correspondente posição.

Structs:

Sprite - struct que representa uma Sprite, contendo um endereço base e os dados obtidos do `xpm_image_t` object que se passa para a função de `xpm_load()` nomeadamente a sua width e height, bem como o x e y do seu canto superior esquerdo e ainda o scancode ao qual corresponde (este campo só é utilizado no caso de a sprite representar uma letra).

Animation - struct que faz handle de animações no nosso programa, contendo um array de Sprites que são os diferentes frames, o número de frames e informação sobre o frame atual.

Desenvolvido por: Miguel Freitas/Joana Mesquita



## Power ups

Este módulo é usado para criar e tratar dos efeitos de power ups incluídos no jogo. A criação destes power ups é feita pela função `handlePowerUpSpawn()` que tem uma probabilidade de 33% de gerar um power up e o seu efeito é removido pela função `handleRemovePowerUp()` após este ter estado ativo por 7 segundos. As funções que tratam dos efeitos destes dependem do power up específico e estão incluídas na struct `PowerUp`.

Structs:

`PowerUp` - struct que representa um power up, contendo uma struct `Animation` (com os sprites da animação e todos os restantes elementos respetivos a esta), uma flag a indicar se o power up está ativo, outra a indicar se está a ser renderizado, o valor do counter do timer quando o sprite foi desenhado e a função que trata dos efeitos desse power up específico.

Desenvolvido por: Miguel Freitas/Joana Mesquita

Module	Relative Weight
Keyboard	6%
Mouse	12%
RTC	5%
Timer	5%
Video_gr	16%
Logic	50%
UI elements	N/A
Power ups	6%

## Implementation details

### RTC

A implementação do rtc foi bastante simples sendo a nossa única dificuldade no processo, o facto de, inicialmente, não termos percebido que ele enviava os números (dia, mês, hora, ...) em hexadecimal e não inteiro. Acabamos por transformar o hexadecimal em inteiro escrevendo-o para um ficheiro com o formato "%02X" cada elemento da data e horas (highscore.txt, ficheiro este, que tem como objetivo guardar este tempo assim como o nome do jogador e o highscore que ele atingiu) onde ele aparece já como inteiro e depois lemo-lo como inteiro a partir do ficheiro.

### Colisões

Em termos de colisões, no nosso jogo temos uma bola a colidir ou com um dos jogadores ou com um power up. No entanto, apercebemo-nos inicialmente, que era muito fácil que a bola e os jogadores acabassem não só por colidir como também por ficar presos, quando, devido à velocidade do player ou da bola, eles acabavam por ter vários pixeis a coincidir acidentalmente.

Para resolver este problema começámos por verificar se este tipo de colisão indesejada iria acontecer da próxima vez que a bola ou o player se mexessem e tivemos que considerar duas situações distintas.

Temos a situação em que, simplesmente, a bola vai coincidir com o jogador apenas devido à sua velocidade, que pode ser resolvida facilmente mudando a posição da bola de modo a que isto não aconteça.

Por outro lado, temos ainda a situação em que eles coincidem, não devido à velocidade da bola, mas devido à velocidade do jogador. Esta situação particular, ocorre quando a bola se encontra em baixo do jogador ou acima deste e este se mexe na direção da bola, antes de ela poder sair dessa posição, para resolver este problema foi necessário verificar se este tipo de colisão aconteceu e colocar a bola no topo ou no fundo do jogador dependendo da sua posição inicial dando a ideia de que foi “arrastada” por este.

A deteção da colisão em si foi feita com um pixel de distância entre o jogador e a bola para evitar apagar parte do jogador quando estes tocam. O método utilizado foi o de percorrer todos os pontos do jogador e, ver se pertenciam à bola que possui uma equação de um círculo  $((x-x_1)^2 + (y-y_1)^2 = r^2)$ .

## Call Graph

