U.PORTO
FEUP FACULDADE DE ENGENHARIA
UNIVERSIDADE DO PORTO

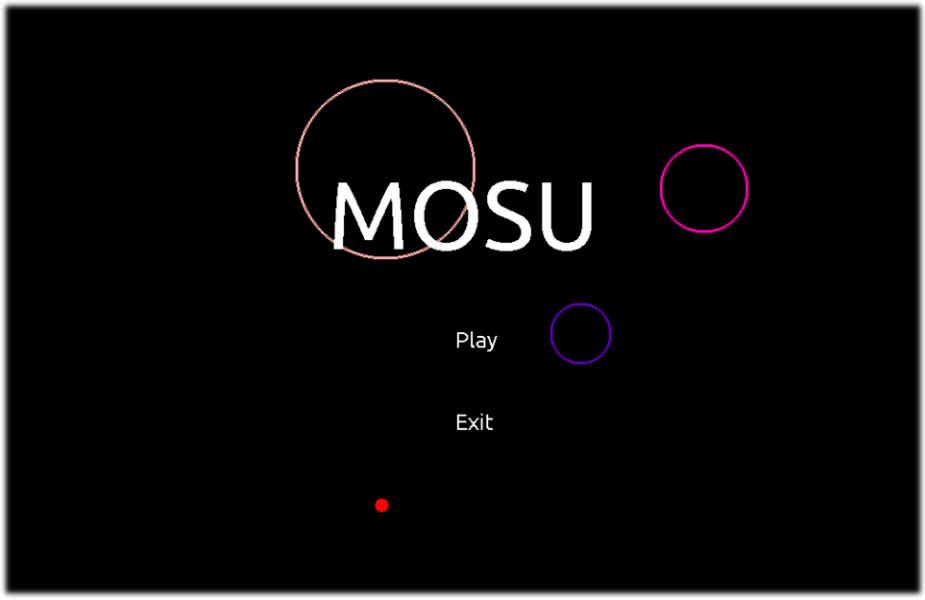
MOSU

Trabalho realizado por:

- Miguel Alexandre Brandão Teixeira – up201605150
- Pedro Xavier T. M. C. de Pinho – up201605166

1. Instruções de utilização do jogo

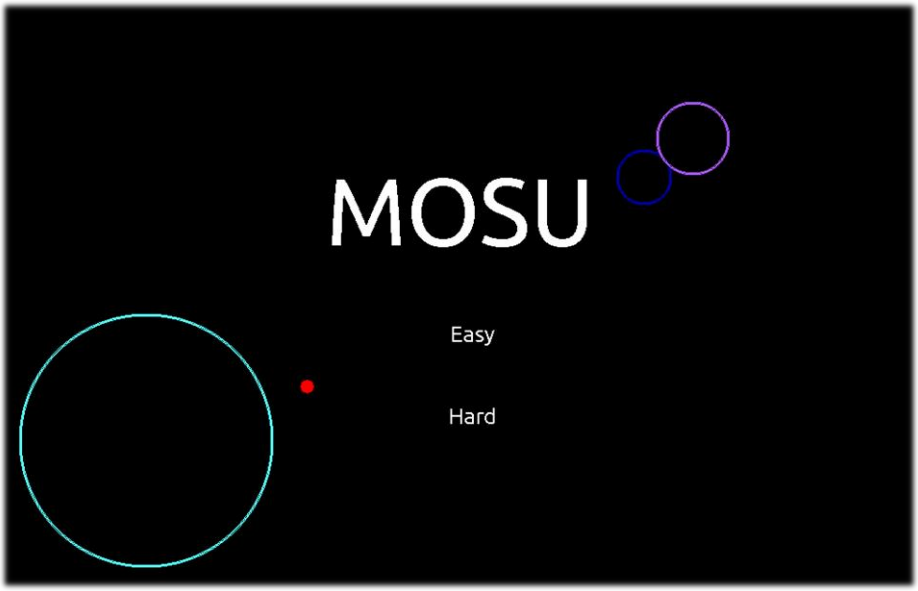
1.1 Menu Inicial



Ao iniciar o jogo, o utilizador é apresentado com um menu inicial. O menu tem um estilo minimalista, com o logótipo do jogo e as opções disponíveis ao jogador, assim como vários circulos a aparecer em posições aleatórias do ecrã. Este menu apresenta duas opções ao utilizador: Play e Exit. Cada opção do menu é highlighted quando o utilizador passa o cursor por cima da mesma.

A primeira opção abre um segundo menu, permitindo ao utlizador escolher a dificuldade do jogo (informações relativas aos modos de jogo mais à frente).

Se pretender andar para trás (para o primeiro menu) estando no segundo menu, pode premir a tecla “ESC” para o fazer (contudo para sair do jogo apenas o poderá fazer clicando na opção “Exit”).



A segunda opção do primeiro menu sai do jogo, retornando à linha de comandos do Minix.

1.2 Jogo

O jogo está dividido em dois modos de jogo: modo easy e o modo hard. Em ambos os modos, o utlizador tem de mover o cursor e clicar, utilizando o teclado, nos círculos, tendo em atenção o *timing* (círculo exterior que diminui ao longo tempo).

A pontuação atribuída por círculo depende do *timing* do utilizador. Para obter a pontuação máxima, o jogador deve clicar no momento exacto em que o círculo externo coincide com o círculo principal, recebendo assim 300 pontos. Dependendo da antecedência com que clica no círculo, o jogador pode ser penalizado na pontuação, recebendo apenas entre 0 a 100 pontos.

Para ajudar o utilizador a ganhar mais pontos existe um *multiplier*. Este é incrementado cada vez que o utilizador acerta num círculo, motivando o jogador a não falhar nenhum círculo. A pontuação final de cada círculo é calculada multiplicando a pontuação do círculo pelo *multiplier*.

O utilizador tem também uma vida limitada. No início do jogo, esta começa a 100 (estando limitada a 100 no máximo, durante o jogo todo) podendo, ao longo do jogo, diminuir ou aumentar.

Sempre que o utilizador recebe:

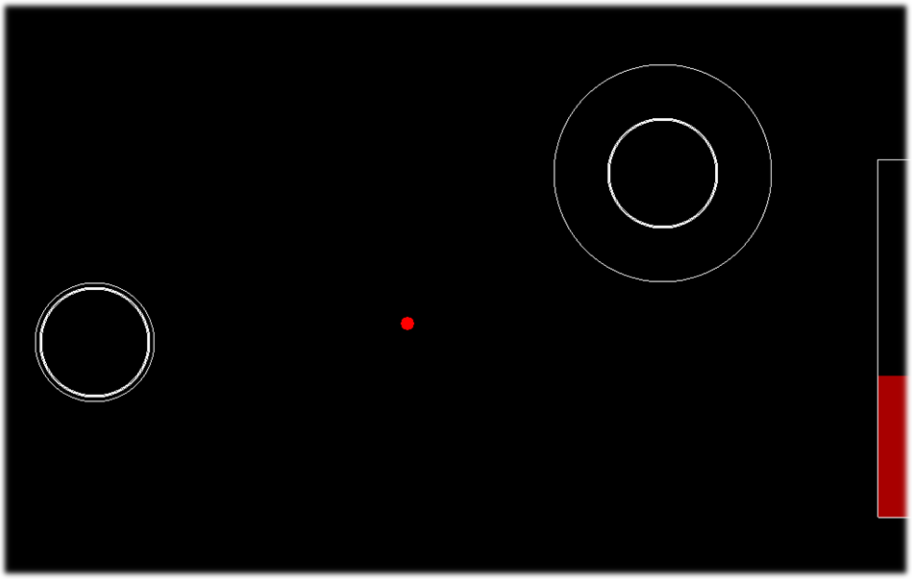
- 300 pontos, este recebe 15 de vida;
- 100 pontos, este recebe 5 de vida;
- 50 pontos, este perde 5 de vida (de modo a desmotivar o utilizador clicar no círculo mal ele apareça)
- 0 pontos, este perde 15 de vida (quando o utilizador falha um círculo).

Para o utilizador ter informação relativa à vida restante, implementamos uma barra (vermelha) no lado direito do ecrã que se vai aumentando / diminuindo à medida que sofre alterações.

Para ajudar o utilizador a saber se acertou ou falhou nos círculos, sempre que um dos círculos é clicado aparece um “certo” ou “errado”, respectivamente.

Não foi implementado nenhum menu de pausa, podendo, no entanto, clicar na tecla “ESC” para voltar ao menu principal.

1.2.1 Modo *easy*



Este modo de jogo é feito para o utilizador se habituar aos controlos, ao estilo do jogo e ao *timing* dos círculos. É um modo simplificado do modo "normal" (modo *hard*), em que o utilizador apenas tem de mover o rato para o círculo e clicar no teclado, sendo qualquer tecla aceite (exceto "ESC").

1.2.1 Modo *hard*

Este modo de jogo é idêntico ao modo *easy*, adicionando a mecânica das letras, não sendo todas as teclas aceites. O utilizador, para além de ter de mover o cursor para o círculo, tem de clicar na tecla correspondente do mesmo, de modo a acertar no círculo.

2. Estado do Projeto

2.1 Funcionalidades implementadas

As funcionalidades implementadas foram mencionadas na secção anterior, todas com sucesso.

2.2 Periféricos usados

Tabela com os periféricos usados no nosso projeto:

Periférico	Propósito	Interrupções
Placa Gráfica	Visualização do menu e dos elementos do jogo	Não
Timer	Geração de círculos e controlo dos frames	Sim
Rato	Controlo do cursor no ecra durante o jogo e no menu	Sim
Teclado	Controlo de teclas premidas durante o jogo	Sim
RTC	Guardar a data de inicio do jogo e alteração do multiplier para guardar num ficheiro os “high scores”	Sim

2.2.1 Placa Gráfica

Nós usamos o modo de vídeo 0x105 (resolução 1024 x 768 com um total de 256 cores por pixel) especificado no *standard VBE 2.0.*

É utilizado double buffering para reduzir a ocorrência de *screen tearing*.
Funções usadas (video_gr.c):

- vg_init
- vg_exit
- vg_nextFrame
- vg_clear

Usamos *sprites* (estáticas), tanto para o menu como para o jogo em si, também usamos um algoritmo para desenhar círculos de acordo com as nossas especificações (retirado da *net*).

Funções usadas (video_gr.c):

- vg_paintPixel
- vg_sprite
- create_sprite (sprite.c)
- destroy_sprite (sprite.c)
- vg_drawSprite (sprite.c)
- vg_eraseSprite (sprite.c)
- vg_circle
- vg_Rectangle

Não usamos fonts diretamente mas sim *sprites* que representam letras ou palavras.
Ficheiros usados:

- pixmap.h (localização dos *xpm’s* usados)

Apenas é usada uma função *VBE*: vbe_get_mode_info.

2.2.2 Teclado

É usado na deteção de teclas premidas (Q,W,E,R,ESC) pelo utilizador durante o jogo.
Principal função usada (para além da *subscribe* e *unsubscribe* já usadas anteriormente nos outros labs):

- keyboard_int_handler (keyboard.c)

2.2.3 Rato

É usado para selecionar opções no menu e durante o jogo (detetando o movimento do cursor e o butão esquerdo para selecionar determinada opção no menu).

Principal função usada (para além da *subscribe* e *unsubscribe* já usadas anteriormente nos outros labs):

- mouse_int_handler (mouse.c)

Também criamos uma *struct* para a variável *Cursor* (cursor.c e cursor.h).

Funções usadas (cursor.c):

- create_cursor
- draw_cursor (usa função vg_sprite)
- erase_cursor (usa função vg_sprite)
- update_cursor (usa função vg_sprite)
- destroy_cursor

2.2.4 RTC (Real Time Clock)

É usado para ler a data atual e se o jogador estiver a jogar à mais que 30 minutos irá receber um *bonus* de *multiplier*. Também irá usar a hora atual obtida para guardar num ficheiro os *scores* com a data da sua ocorrência.

Principal função usada (para além da *subscribe* e *unsubscribe*):

- rtc_int_handler (rtc.c)
- handle_update_int (rtc.c)
- checkClock (rtc.c)

2.2.5 Timer

É usado para controlar a criação de círculos e a *frame-rate* do jogo (60 *fps*) e também os eventos mandados para a *state machine* quando é requerido.
Apenas tiramos partido das interrupções do timer, logo não existe nenhuma função que se destaque (apenas usamos *subscribe* e *unsubscribe* já usadas anteriormente nos labs).

3. Organização e estrutura do código

3.1 circle.c

Este módulo lida com tudo relacionado com círculos. Contém funções que criam, desenham, apagam, fazem *tick* (ver documentação para mais informações sobre funções deste tipo) e destroem círculos.

Estrutura de dados utilizada neste módulo:

- Circle – contém todas a informações do círculo, tal como a posição no ecrã (no eixo do x e do y); o raio do círculo principal e do círculo externo, a cor, a letra e o tempo restando (*frames*) do círculo no ecrã.

Para desenhar círculos, utilizamos o *Bresenham's circle algorithm*:
https://rosettacode.org/wiki/Bitmap/Midpoint_circle_algorithm#C

Peso no projecto: 10%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.2 cursor.c

Este módulo lida com tudo relacionado com cursores. Contém funções que criam, desenham, apagam e destroem cursores. Contém ainda uma função lê a posição atual do cursor e faz *update* às componentes x e y do mesmo, conforme os argumentos que recebe (novamente, ver documentação para mais detalhes).

Estrutura de dados utilizada neste módulo:

- Cursor – contém todas as informações sobre o círculo, tal como a sua posição no ecrã (no eixo do x e do y).

Peso no projecto: 10%
Desenvolvido por: Pedro Pinho

3.3 game.c

Este módulo contém funções que iniciam a *state machine* com os valores iniciais, assim como as funções com toda lógica do menu e dos dois modos de jogo.

Peso no projeto: 20%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.4 i8042.c

Este módulo contém funções comuns ao mouse keyboard, permitindo a ambos interagir com o KBC.

Peso no projecto: 1%
Desenvolvido por: Miguel Teixeira

3.5 game_core_st.c

Este módulo lida com tudo relacionado com a *state machine*. Contém uma função que faz o *handling* dos diversos eventos que a *state machine* pode receber, assim como algumas funções que devolvem informação das variáveis globais definidas neste ficheiro (tal como o jogador e o estado atual da *state machine*).

Eventos existentes (event_t):

- MENU1
- MENU2
- IN_GAME_EASY
- IN_GAME_HARD
- END_GAME

Estados possíveis da *state machine* (state_t):

- ACTIVATE
- LEFT_B
- START_EASY
- START_HARD
- KEY_PRESS
- MISSED
- SAVE

Estruturas de dados utilizada neste módulo:

- event_t – contém todas informações relativas a um evento, tal como o tipo de evento, o código do tecla premida (0x00 caso o evento não precise dele), um pointer para um círculo (NULL caso o evento não precise dele) e um pointer para um cursor (novamente, NULL caso o evento não precise dele).
- player_t - contém todas as informações relativas a um jogador, tal como a sua pontuação atual, o seu *multiplier* atual (assim como o máximo multiplier atingido durante o jogo) e a sua vida atual.

Peso no projecto: 15%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.6 keyboard.c

Este módulo lida com tudo relacionado com o teclado. Contém funções que permitem subscrever interrupções do teclado, assim como fazer o *handling* das mesmas. Contém ainda uma função que devolve o último código lido.

Peso no projecto: 10%
Desenvolvido por: Miguel Teixeira

3.7 mouse.c

Este módulo lida com tudo relacionado com o rato. Contém funções que permitem subscrever interrupções do rato, assim como fazer o *handling* das mesmas. Contém ainda funções que devolvem o último movimento no eixo do x e do y e se o *left click* foi pressionada.

Peso no projecto: 10%
Desenvolvido por: Pedro Pinho

3.8 pixmap.c

Este módulo contém apenas variáveis definidas, que são *sprites* utilizadas ao longo do jogo.

Peso no projecto: 1%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.9 rtc.c

Este módulo lida com tudo relacionado com o RTC. Contém funções que permitem subscrever interrupções do RTC, assim como fazer o *handling* das mesmas.

Estrutura de dados utilizada neste módulo:

- rtc_t – contém todas as informações relativas à data actual, obtida a partir do RTC (data com o dia, mês, ano, hora, minuto e segundo).

Peso no projecto: 1%
Desenvolvido por: Miguel Teixeira

3.10 timer.c

Este módulo lida com tudo relacionado com o *timer*. Contém funções que permitem subscrever interrupções do *timer*, assim como fazer o *handling* das mesmas.

Peso no projecto: 5%
Desenvolvido por: Pedro Pinho

3.11 vbe.c

Este módulo contém uma função que retorna informação sobre o modo de vídeo (ver documentação para mais detalhes).

Estrutura de dados utilizada neste módulo:

- vbe_mode_info_t – contém todas as informações relativas ao modo VBE.

Peso no projecto: 1%
Desenvolvido por: Miguel Teixeira

3.12 video_gr.c

Este módulo lida com tudo relacionado com a memória de vídeo. Contém funções que iniciam e saem do modo gráfico, desenham pixels com uma certa cor e contém ainda uma função que aplica o *double buffering*.

Peso no projecto: 5%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.13 sprite.c

Este módulo lida com tudo relacionado com sprites. Contém funções que criam, desenham, apagam e destroem sprites.

Estrutura de dados utilizada neste módulo:

- Sprite – contém todas as informações relativas a um sprite, tal como a sua posição (no eixo x e y), o tamanho da sprite, entre outros

Peso no projecto: 5%
Desenvolvido por: Miguel Teixeira e Pedro Pinho

3.14 queue.c

Este módulo contém a implementação de uma estrutura de dados em queue, com base em *arrays* dinâmicos. Não foi efectuada nenhuma alteração a esta implementação.

Peso no projecto: 5%

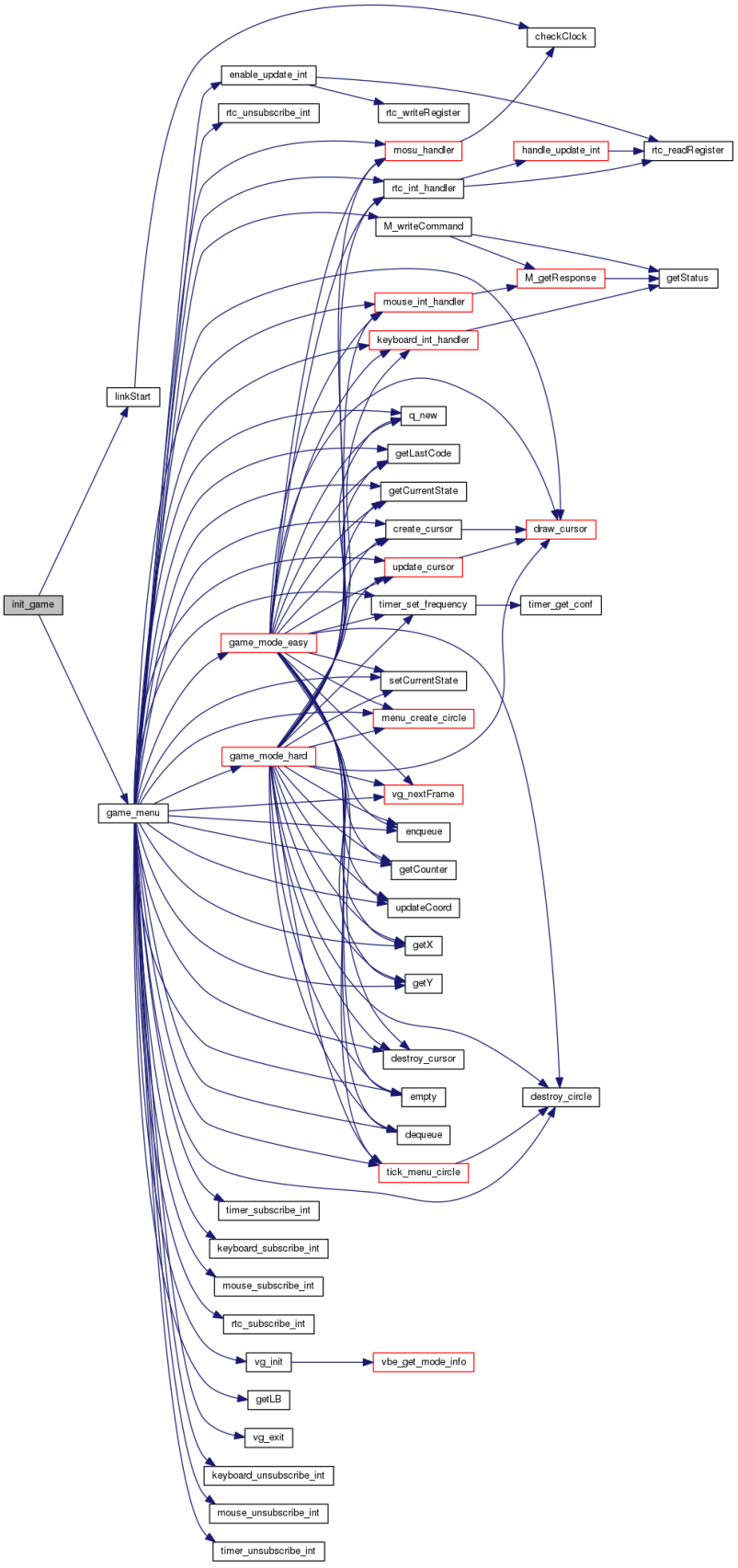
Source: https://rosettacode.org/wiki/Queue/Definition#Dynamic_array

3.15 read_xpm.c

Este módulo contém apenas uma função que permite ler xpm. Este foi nos fornecida por um docente da cadeira.

Peso no projecto: 1%

3.16 Function Call



4. Detalhes da Implementação

4.1 Desenhar Círculos

Foi necessário implementar um algoritmo para visualização de círculos, no entanto não foi difícil arranjar uma opção viável para tal na net. Também tivemos que arranjar uma forma de criar movimento fluído com círculos, que conseguimos atingir desenhando e apagando círculos sucessivamente em intervaloes de tempo extremamente curtos (interrupções do *timer*);

4.2 Implementar uma Queue em C

Tivemos de arranjar uma forma de guardar os círculos criados aleatóriamente para os podermos visualizar, inicialmente utilizamos um *array* para guardar essa informação, pelo que não existia nenhuma estrutura de dados em C que se adapta-se ao que queriamos, contudo esta implementação tinha problemas de memória devido à impossibilidade de alterar o tamanho de um *array* sem ter que alocar memória extra manualmente e os círculos que já não precisavamos mantinham-se no array ocupando memória que poderiamos necessitar mais tarde. Por estas razões decidimos procurar como implementar uma *queue* em C (criando uma estrutura de dados extra), encontramos uma opção bastante boa para o que pretendiamos fazer e assim já podemos controlar os círculos criados bastante facilmente, visto que a implementação da *queue* fazia isso automaticamente.

4.3 Programação Orientada a Objetos

Visto que a linguagem C não tem suporte para classes nem objetos foi preciso criar diversas *structs* para facilitar o código e a transferência de dados entre funções, de modo a ser mais legível e percetível para nós enquanto criavamos o jogo.

4.4 *Double Buffering*

De modo a aumentar a fluidade do jogo, decidimos implementar este método, que em cada interrupção do timer copia a memória da *frame* que queremos fazer display (a *frame* da interrupção anterior) para a memória de vídeo alocada e calcula-se a *frame* seguinte.

4.5 *State Machine*

Achamos que para tratar das interrupções todas seria muito mais eficiente criar uma *state machine* que decidia o que fazer com os evetos (interrupções) que lhe mandamos. Este é o "cérebro" do jogo visto que trata de todo o tipo de informação relativa ao utilizador (vida, *multiplier*, *score*, etc).

5.Conclusões

Concluindo, encontramos várias dificuldades no desenvolvimento do projeto mas acabamos por conseguir atingir o nosso objetivo final e achamos que ficou bastante apelativo, quer do ponto de vista do utilizador (aspeto visual) quer do ponto de vista do programador (código estruturado).

6. Appendix

Também pode ser necessário alterar o “source” file no ficheiro *doxygen*, para criar a documentação.

Para compilar o programa só é necessário fazer o comando “make” dentro da pasta “src”.