# PG LIB

## Biblioteca de suporte a Programação I


### Abstract

Biblioteca de suporte à construção de aplicações gráficas
com integração de mecanismos de comunicação



ISEL

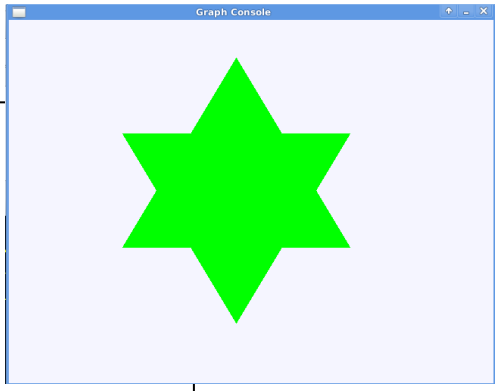
dezembro 2020

# Contents

# Biblioteca de suporte a Programação I

Durante o semestre será usada uma biblioteca que exporta, entre outras que veremos mais tarde, funções de desenho de formas gráficas, nomeadamente pontos, linhas, triângulos, retângulos e círculos. O conjunto completo das funções de desenho é apresentado no final deste documento.

A seguir apresenta-se um programa que exemplifica a utilização da biblioteca o resultado da sua execução (o programa termina quando se fechar a consola gráfica criada):




```
// ficheiro de include necessário
#include "pg/pglib.h"

int main() {
        // chamada necessária antes da iniciação da biblioteca
        graph_init2("Graph Console", 640, 480);

        // desenho de um triângulo dados os pontos dos vértices.
        // O penúltimo argumento indica a côr e o último que o
        // triângulo é desenhado "a cheio"
        graph_triangle(150, 300, 450, 300, 300,  50,  c_green, true);

        // desenho de um segundo triângulo dados os pontos dos vértices,
        // a cor e com desenho "a cheio"
        graph_triangle(150, 150, 450, 150, 300,  400, c_green, true);

        // chamada necessária para o correto refrescamento da consola gráfica
        graph_start();
        return 0;
}
```

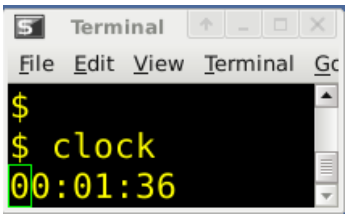
Em geral, para compilar um programa que usa a biblioteca deverá usar-se o seguinte comando:

```
gcc –o <prog> -Wall <prog>.c –lpg
```

***Eventos***

Esta biblioteca permite registar funções (ditas de callback) que serão chamadas na ocorrência de um evento de teclado, de um evento de rato ou de um evento de temporização (por cada período de tempo previamente definido).

Como exemplo de um programa que regista uma função de tratamento de evento de temporização, o programa seguinte mostra na consola de texto um relógio digital (HH:MM:SS):



```
#include <stdio.h>
#include “pg/pglib.h”

#define WINDOW_WIDTH 300
#define WINDOW_HEIGHT 200
int hours, minuts, secs;

void myTimerHandler() {
       secs++;
       if (secs == 60) {
              secs = 0;
              minuts++;
              if (minuts == 60 ) {
                   minuts == 0;
                   hours = (hours + 1) % 24;
               }
       }
       printf(“%02d:%02d:%02d\r”,  hours, minuts, secs);
       fflush(stdout); // esta função força os caracteres a serem escritos na consola
}

int main()  {
       graph_init2(“Digital Clock”, WINDOW_WIDTH, WINDOW_HWIGHT);
       // regista a função “myTimerHandler”
       // para ser chamada em cada segundo
       graph_regist_timer_handler(myTimerHandler, 1000);

       graph_start();
```

Servidor de comunicação em grupo (arranque)

## Arranque e terminação do servidor

Na instalação da biblioteca de PG1 é instalado um servidor que permite a comunicação entre grupos de utilizadores. Este servidor é necessário para suportar, entre outros, o jogo da batalha naval distribuído.

É necessário que os jogadores se registem previamente no servidor, bem como os tipos de jogos necessários. Para os jogos de batalha naval é criado o tipo de jogo `battleship`.

Na versão actual o servidor apenas funciona se estiver a correr na mesma rede local dos jogadores.

Para arrancar com o servidor execute o comando numa janela de consola:

```
launch_server <user_1> <user_2>
```

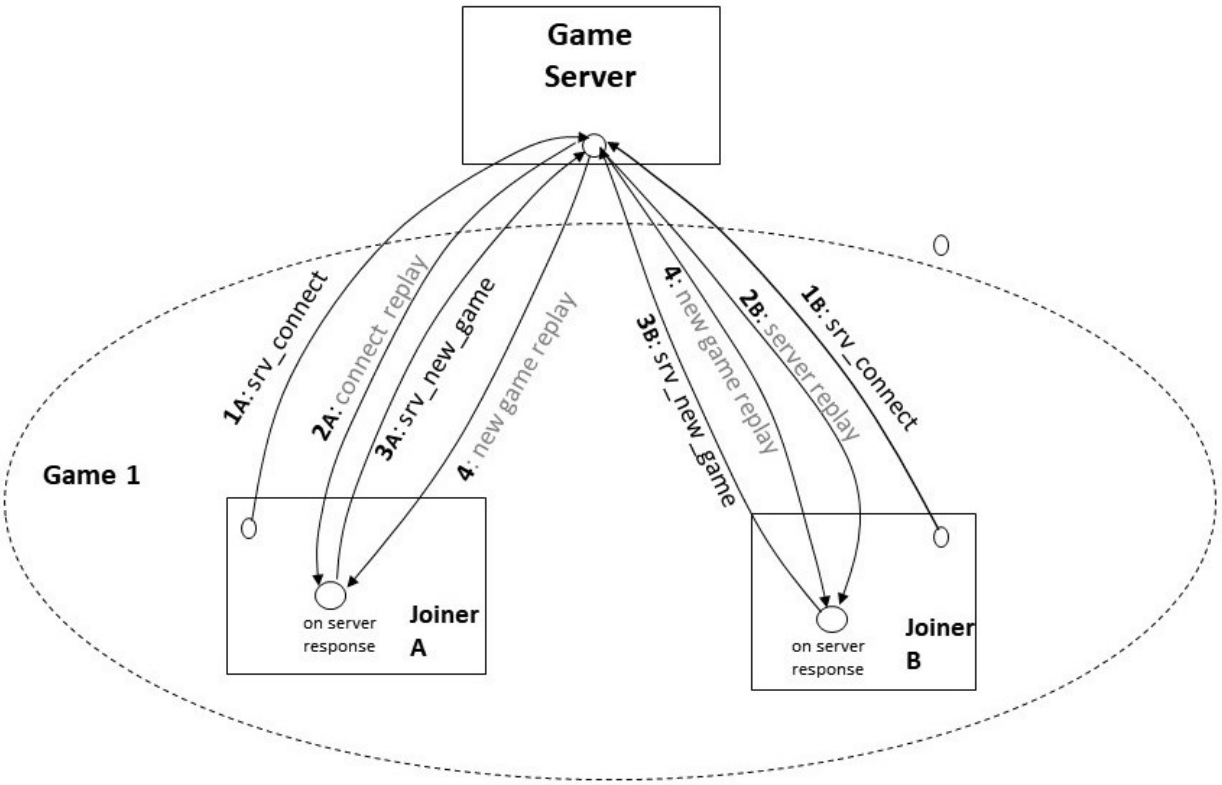
onde `user_1` e `user_2` são os nomes dos utilizadores que serão imediatamente registados. São também criados de imediato os tipos de jogos `battleship` e `tictactoe` (jogo do galo). O servidor fica imediatamente disponível para suportar os jogos. Esta consola fica dedicada em exclusivo à execução do servidor.

Para terminar o servidor execute noutra consola (da máquina onde o servidor está a correr) o comando:

```
reg_stop
```

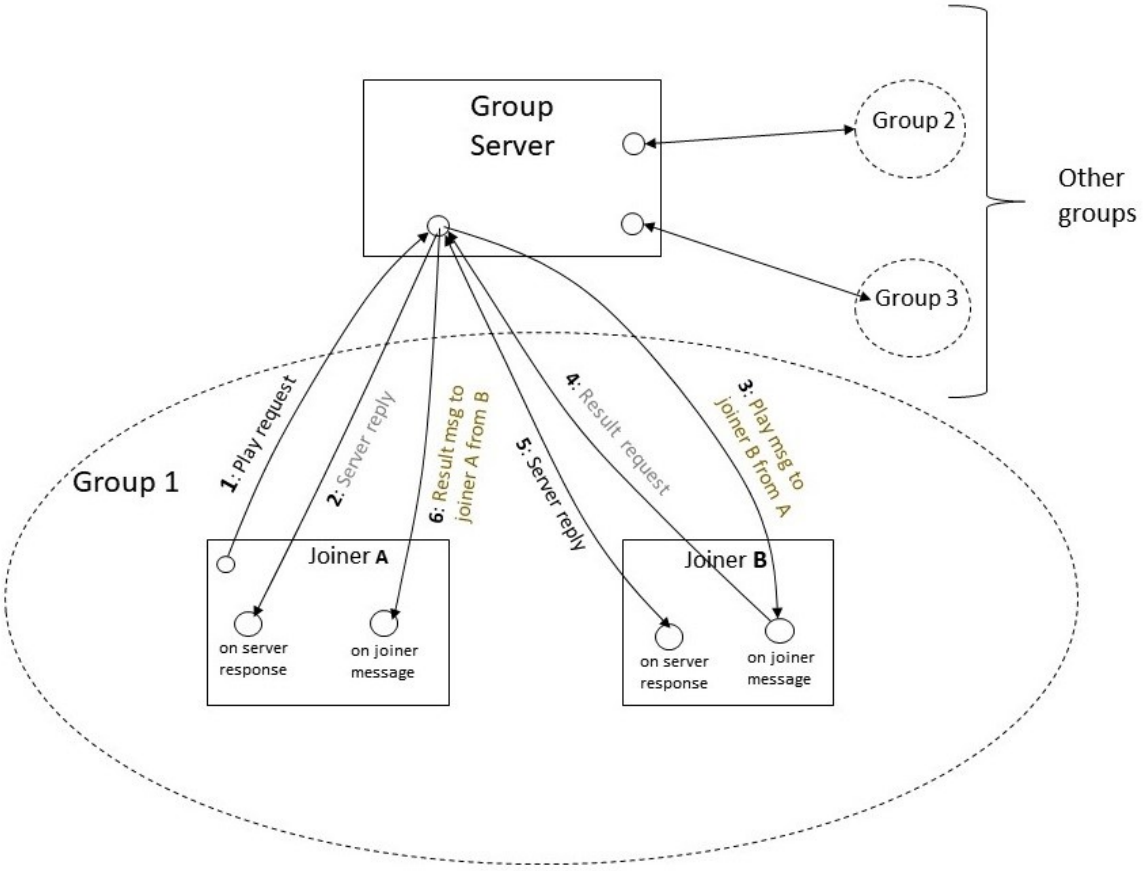
Servidor de comunicação em grupo (criação de jogo)

**Fase de criação de um jogo**



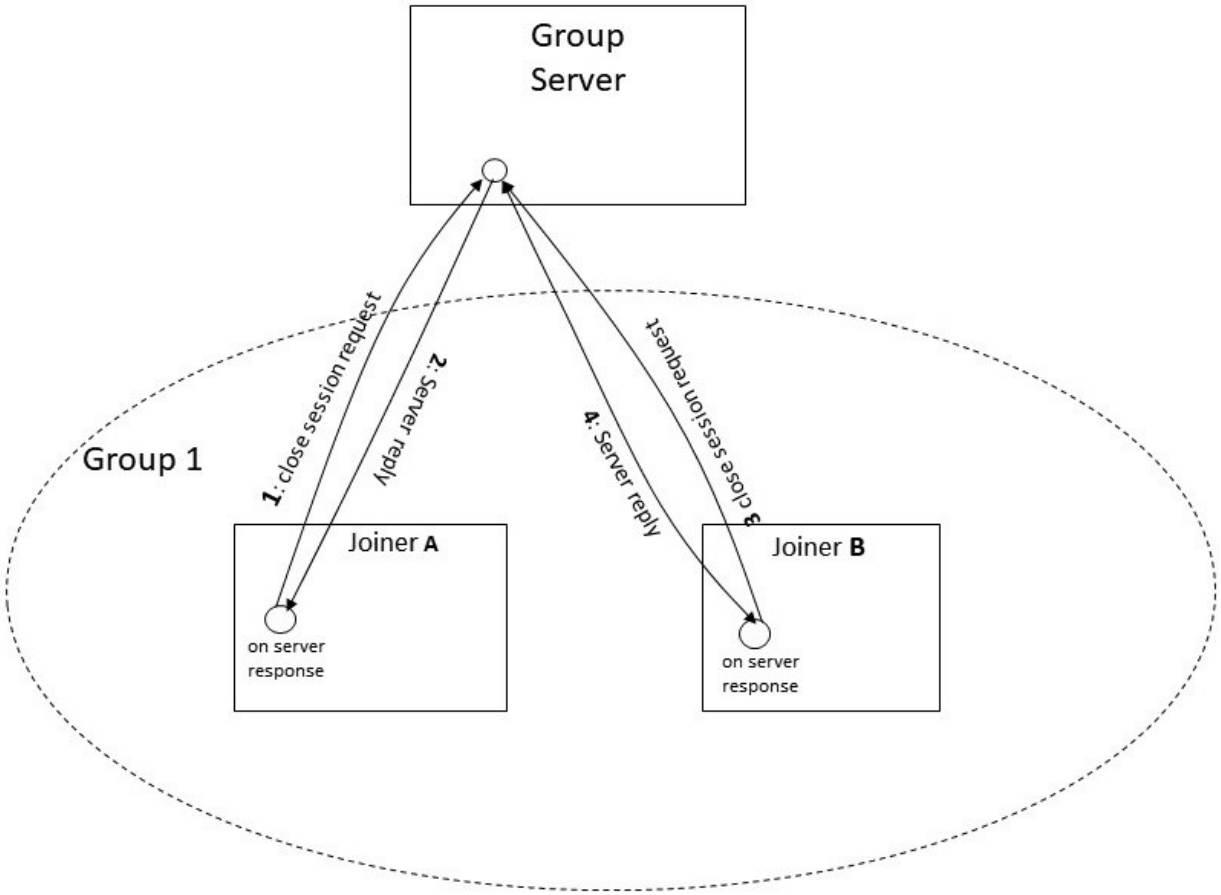

| | |
|---|---|
| **1A -** Ligação ao servidor por parte do jogador **A**<br>**session = server_connect**(user_name, ip_addr, on_server_response, on_joiner_msg) | **1B -** Ligação ao servidor por parte do jogador **B** |
| **2A -** Resposta do servidor ao pedido de ligação do jogador **A**<br>(**on_server_response**) | **2B -** Resposta do servidor ao pedido de ligação do jogador **B** |
| **3A -** Pedido de criação de novo jogo ao servidor, por parte do jogador **A**, no caso de ligação bem sucedida. A resposta ao pedido só ocorre quando o outro jogador executar o mesmo comando.<br>**server_new_game**(session, game_type, game_name) | **3B -** Pedido de criação de novo jogo ao servidor, por parte do jogador **B**, no caso de ligação bem sucedida. A resposta ao pedido só ocorre quando o outro jogador executar o mesmo comando. |
| **4 -** Resposta a ambos os jogadores (em **on_server_response**) ao pedido de criação de jogo dos jogadores **A** e **B**, em caso de sucesso (STATUS_OK), com a informação necessária para o jogo se iniciar (se o jogador é o primeiro a começar, e qual o nome do adversário). Nesta versão o primeiro a começar é aquele que primeiro enviou o pedido de criação de jogo ao servidor. | |

**Desenrolar de uma jogada**




1. Pedido ao servidor para executar jogada (**srv_play**) por parte do jogador **A**
2. Resposta do servidor ao pedido de jogada (**on_server_response**)
3. O servidor envia mensagem ao jogador **B** com a jogada do jogador **A** (**on_joiner_msg**)
4. O jogador envia ao servidor o resultado da jogada (**srv_send_result**) do jogador **A** (se se aplicar).
5. Resposta do servidor ao pedido de envio de resultado (**on_server_response**)
6. O servidor envia mensagem ao jogador **A** com o resultado indicado pelo jogador **B** (**on_joiner_msg**)

**Terminar Sessão**




1. Pedido ao servidor de terminação da sessão (e jogo) (**srv_session_close**) por parte do jogador **A**. O primeiro jogador a executar o pedido provoca a destruição do jogo.

2. Resposta do servidor ao pedido (**on_server_response**)

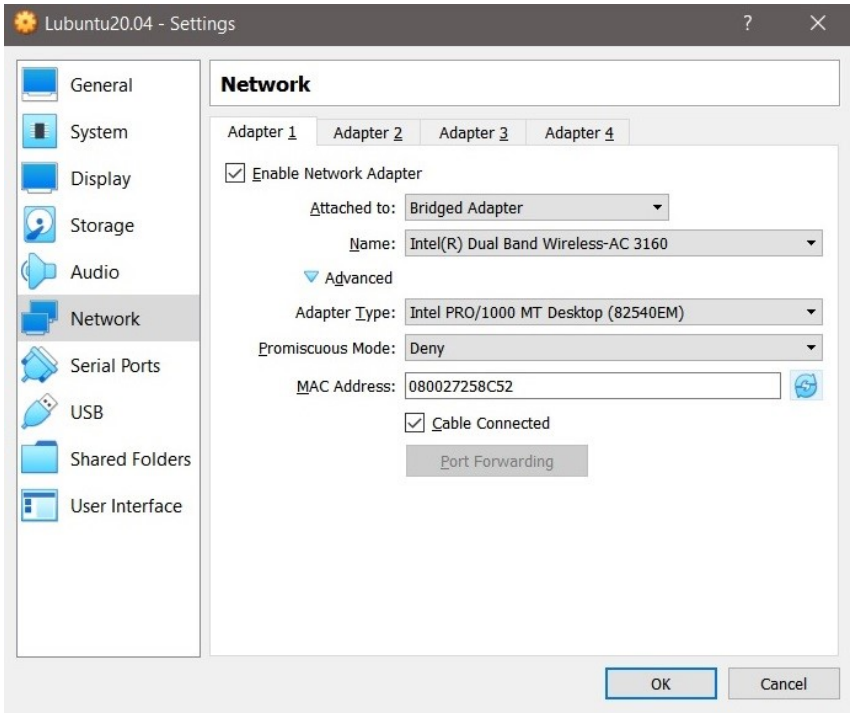
3. Pedido ao servidor de terminação da sessão (e jogo) (**srv_session_close**) por parte do jogador **B**. Se o jogo já estiver destruído apenas fecha a comunicação com o servidor.

4. Resposta do servidor ao pedido (**on_server_response**)

## Configuração de rede das máquinas virtuais

De modo a permitir a comunicação entre as máquinas virtuais dentro da mesma rede física local, execute os procedimentos pela ordem indicada:

1. Instalar programa para obter informações sobre a rede, em particular o endereço IP da máquina. Numa janela de consola execute o comando: sudo apt install net-tools
2. Alterar o endereço ethernet e o modo de ligação à rede da máquina virtual. Com a máquina virtual (Lubuntu) desligada, abra os settings de rede da máquina virtual na VirtualBox e expanda o modo "Advanced". A figura abaixo mostra a janela apresentada. Sobre a janela execute as seguintes ações:
    a. Na opção "Attached to:", que deve estar como "NAT", modifique para "Bridged Adapter" como mostra a figura.
    b. Na opção "MAC Address click", prima o icon azul à direita para criar um novo endereço ethernet (MAC) para a máquina virtual. Como foram todas copiadas da máquina virtual distribuída no início do semestre, estão com o mesmo endereço, o que causaria problemas na comunicação.
    c. Modifique o nome do "Network Adapter" para o valor apresentado ("Intel® Dual Band Wireless-AC 3160")




3. Arranque com a máquina virtual e execute o comando ifconfig numa janela de consola. verá o endereço IP como valor do campo inet (192.168.1.95) e o endereço ethernet do "Network Adapter" no campo ether (08:00:27:25:8c:52), como mostra o texto abaixo:

```
~$ ifconfig
enp0s3: flags=4163<UP,BROADCAST,RUNNING,MULTICAST>  mtu 1500
        inet 192.168.1.95  netmask 255.255.255.0  broadcast 192.168.1.255
        inet6 fe80::d58:ecc1:7b25:d9bb  prefixlen 64  scopeid 0x20<link>
        ether 08:00:27:25:8c:52  txqueuelen 1000  (Ethernet)
```

Índice de funções e tipos da biblioteca

## Funções de controle  e cores

```
/**
 *  initializes the graphics. Must be called first
 */
int graph_init();

/**
 * initializes the graphics, specifying dimensions and a title for the graphic window
 * Must be called first.
 */
int graph_init2(const char title[], int width, int height);

/**
 *  start the event loop
 */
void graph_start();

/**
 *  stop the event loop
 */
void  graph_exit();

/**
 *  refresh graphics windows.
 *  Must be called to immediately show operations result
 */
void graph_refresh();

/**
 *  get the graph window width
 */
int graph_get_width();

/**
 *  get the graph_window height
 */
int graph_get_height();

/**
 * obter as componentes "red", "green" e "blue" de uma côr RGB
 */
int rgb_red(RGB rgb);

int rgb_green(RGB rgb);

int rgb_blue(RGB rgb);

/**
 *  returns an rgb color from components
 */
RGB graph_rgb(int r, int g, int b);

#define      c_white        graph_rgb(255,255,255)
#define      c_lightblue    graph_rgb(245,245,255)
#define      c_dgray        graph_rgb(64,64,64)
#define      c_cyan         graph_rgb(32,255,255)
#define      c_black        graph_rgb(0,0,0)
#define      c_gray         graph_rgb(210,210,210)
#define      c_red          graph_rgb(255, 0, 0)
#define      c_green        graph_rgb(0, 255, 0)
#define      c_blue         graph_rgb(0, 0, 255)
#define      c_orange       graph_rgb(255, 146, 36)
#define      c_yellow       graph_rgb(255, 255, 0)
```

## Tipos e funções de desenho

```
// Ponto 2d, pode representar por exemplo uma posição no ecrâ
typedef struct {
      int x;
      int y;
} Point;

// Representa uma dimensão 2D
typedef struct {
      int width, height;
} Size;

// Representa uma área retangular através do ponto origem e dimensões
typedef struct {
      Point p;
      Size sz;
} Rect;

/*

/**
 *  draw a pixel at position with specified color
 */
void graph_pixel(short x, short y, RGB color);

/**
 *  draw a line at position with specified color
 */
void graph_line(short x1, short y1, short x2, short y2, RGB color);

/**
 *  draw a circle at position with specified color, filled or not
 */
void  graph_circle(short x0, short y0, short radius, RGB color, bool toFill);

/**
 *  draw a rect at position with specified color, filled or not
 */
void graph_rect(short x0, short y0, short w, short h, RGB color, bool toFill);

/**
 *  draw a ellipse at position with specified color, filled or not
 */
void  graph_ellipse(short x0, short y0, short xr, short yr, RGB color, bool toFill);

/**
 *  draw a round rectangle at position with specified color, filled or not
 */
void graph_round_rect(short x0, short y0, short w, short h, RGB color, bool toFill);

/**
 *  draw a triangle at position (x0,y0) with specified color, filled or not
 */
int graph_triangle2(int x0, int y0, int a, int b, int c, RGB color, bool toFill);

/**
 *  draw a triangle given his points (x0,y0), (x1,y1), (x2,y2)
 * with specified color, filled or not
 */
void graph_triangle(int x0, int y0, int x1, int y1, int x2, int y2, RGB color, bool toFill );
```

Funções para apresentação de texto

```
//font types for text write
#define SMALL_FONT    1
#define MEDIUM_FONT   2
#define LARGE_FONT    3

/**
 * Writes a string starting at point x,y with "color" as text color
 * and witth font size specified in fontsize (SMALL, MEDIUM, LARGE)
 */
void graph_text( short x, short y, RGB color, const char text[], int fontsize );

/**
 * Apresenta o texto passado em "text". Os parâmetros são:
 *      x,y é o ponto de inicio (canto inferior esquerdo da área do texto)
 *      "fore_color" é a cor da letra e back_color é a cor de fundo.
 *      "text" é a string que se deseja escrever;
 *      "font_type" indica a fonte a usar: SMALL, MEDIUM, LARGE
 */
void graph_text2( short x, short y, RGB fore_color, RGB back_color, const char text[], int font_type );

/**
 * Retorna uma estrutura Size com as dimensões da fonte
 * de texto especificada.
 * Parâmetros:
 *      "font_type": tipo da fonte (SMALL_FONT, MEDIUM_FONT, LARGE_FONT)
 * Retorna:
 *      estrutura Size com as dimensões da fonte
 */
Size graph_font_size(int font_type);


/**
 * Retorna uma estrutura Size com as dimensões ocupado pelo número
 * de caracteres na  dimensão de fonte indicados
 * Parâmetros:
 *             "nchars": número de caracteres
 *             "font": dimensão da fonte (SMALL_FONT, MEDIUM_FONT, LARGE_FONT)
 * Retorna:
 *             estrutura Size com as dimensões do texto
 */
Size graph_chars_size(int nchars, int font);


/**
 * Retorna uma estrutura Size com as dimensões ocupado pelo texto "text"
 * na  dimensão de fonte "font"
 * Parâmetros:
 *             "text": string com o texto a avaliar
 *             "font": dimensão da fonte (SMALL_FONT, MEDIUM_FONT, LARGE_FONT)
 * Retorna:
 *             estrutura Size com as dimensões do texto
 */
Size graph_text_size(const char text[], int font);
```

```
// Possibilidades de alinhamento do texto na mensagem
#define ALIGN_CENTER 1
#define ALIGN_RIGHT  2
#define ALIGN_LEFT   3



/*
 * Criação de elemento gráfico para afixar mensagens
 */
void mv_create(MsgView  *mv, int x, int y, int tchars, int font, RGB tc, RGB bc);


/*
 * Apresentação de um dado numero através do MsgView especificado
 */
void mv_show_number(MsgView* nv, int num);

/*
 * Apresentação de uma dada string através do MsgView
 * e alinhamento especificados
 */
void mv_show_text(MsgView* mv, const char  msg[], int align);

/*
 * Apresentação de uma dada string através do MsgView
 * côr e alinhamento especificados
 */
void mv_show_clrtext(MsgView* mv, const char  msg[], RGB tcolor, int align);

// redefinir a côr do texto
void mv_set_clr(MsgView *mv, RGB clr);

// redefinir as margens da caixa de texto
void mv_set_margins(MsgView *mv, int mw, int mh);



/*
 * Funções para criação e utilização de relógio digital com minutos e segundos
 */


#define SMALL_CLOCK SMALL_FONT
#define MEDIUM_CLOCK  MEDIUM_FONT
#define LARGE_CLOCK LARGE_FONT


/*
 *  Criação de relógio
 *  Parâmetros:
 *            x, y : coordenadas do canto superior esquerdo
 *            tcolor: côr dos dígitos
 *            bcolor: côr do fundo
 */
void clk_create(Clock  *c, int x, int y, int font, RGB tcolor, RGB bcolor);
```

```
/*
 *  Criação de relógio count down
 *  Parâmetros:
 *             x, y : coordenadas do canto superior esquerdo
 *             tcolor: côr dos dígitos
 *             bcolor: côr do fundo
 */
void clk_create_cron(Clock *c, int x, int y, int sm, int ss, int font, RGB tcolor, RGB bcolor);


// mostrar tempo
void clk_show(Clock *c);


// incremento de um segundo
void clk_tick(Clock *c);

// decremento de um segundo
Clock clk_down_tick(Clock c);

// Reset, por a zeros
void clk_reset(Clock *c);



/*
 *Funções para lidar com contadores
 */

// criação de um contador na posição dada
void ct_create(Counter  *ct, int x, int y, int initial, char title[], int font_size);

// apresentação do contador
void ct_show(Counter *c);

// retorna um contador que resulta do incremento do contador dado
void ct_inc(Counter *c);

// retorna um contador que resulta do decremento do contador dado
void ct_dec(Counter *c);


/*
 * Funções para lidar com botões
 */


// criação de botão com texto e posição dados
void bt_create(Button  *b, int x, int y, char text[], RGB text_color);

// desenho de botão
void bt_draw(Button *bt, int state);

// verifica se o ponto está contido no botão no caso do botão estar enabled
// caso o botão esteja disabled retorna sempre false;
bool bt_selected(Button *bt, int x, int y);

// modifica a cor do texto do botão
void bt_set_text_color(Button *bt, RGB text_color);

// coloca o botão enabled ou disabled
void bt_set_enable(Button *bt, bool enabled);

// verifica se o botão está enabled
bool bt_is_enabled(Button *bt);
```

```
/**
 *  apresenta o conteúdo de um ficheiro imagem (ex: jpg) na janela gráfica
 *     Parâmetros:
 *     "path" : nome do ficheiro imagem
 *     "x", "y": coordenadas do canto superior esquerdo do "viewport"
 *     "width", "height": dimensões do "viewport"
 */
bool graph_image(const char *path, int x, int y, int width, int height);

/**
 * captura o conteúdo da janela gráfica para um ficheiro
 */
bool graph_save_screen(const char *path);

/*
 * necessário chamar antes de usar graph_get_pixel
 */
void graph_start_capture();

/*
 * necessário chamar depois de usar um ou mais graph_get_pixel
 */
void graph_end_capture();

/**
 *  Obtem o pixel na posição "x", "y" da janela gráfica
 *  Para chamar esta função (uma ou mais vezes) é necessária
 *  invocar as funções graph_start_capture e graph_end_capture imediatamente
 *  antes e depois da utilização, respectivamente.
 */
RGB  graph_get_pixel(int x, int y);

// obter pixeis dentro de uma área do ecrâ especificada
ScrArea graph_start_capture2(Rect r);

void graph_end_capture2(ScrArea area);

/**
 *  Obtém o pixel na posição "x", "y" dentro da área da janela gráfica
 *  representada pela ScrArea "area".
 *  Para chamar esta função (uma ou mais vezes) é necessário
 *  invocar as funções graph_start_capture2 e graph_end_capture2 imediatamente
 *  antes e depois  da utilização, respectivamente.
 */
RGB graph_get_pixel2(ScrArea area, int x, int y);

/**
 * Toca o ficheiro audio (apenas formato raw) de nome "path"
 */
bool  sound_play(const char *path);

/**
 * Pausa a reprodução do som
 */
void sound_pause();

/**
 * Recomeça a reprodução do som
 */
void sound_resume();

/**
 * Parar a reprodução do som
 */
void sound_stop();
```

Tipos e funções relacionados com eventos de teclado, rato e temporização

```
/*
 * constantes de eventos de rato
 */

// campo type de MouseEvent
#define MOUSE_MOTION_EVENT 1
#define MOUSE_BUTTON_EVENT 2

// campo button de MouseEvent
#define BUTTON_LEFT         1
#define BUTTON_RIGHT        2


// campo state de MouseEvent
#define BUTTON_PRESSED      1
#define BUTTON_RELEASED     2
#define BUTTON_CLICK        4
typedef struct MouseEvent  {
      int type;             /* mouse motion or mouse button event type */
      byte state;           /* button pressed, released or click */
      int button;           /* which button is pressed */
      ushort x, y;          /* mouse coordinates */
      short dx, dy;         /* delta move when mouse motion event */
} MouseEvent;

/*  KEYBOARD Events constants */
// campo state de KeyEvent
#define KEY_PRESSED 0
#define KEY_RELEASED1

typedef struct KeyEvent {
      int state;            /* key pressed or released */
      ushort keysym;        /* key code */
} KeyEvent;

// assinatura das funções que tratam eventos
void KeyEventHandler(KeyEvent ke);
void MouseEventHandler(MouseEvent me);
void TimerEventHandler();

/* funções para registar as funções que processam eventos */

// Regist the keyboard event handler
void graph_regist_key_handler(KeyEventHandler ke);

// Regist the mouse event handler
void graph_regist_mouse_handler(MouseEventHandler me);

// Regist the timer event handler
void graph_regist_timer_handler(TimerEventHandler te, uint period);
```

Funções e constantes relacionadas com comunicação entre jogadores

```
// limits

#define MAX_NAME_SIZE              31            // tamanho máximo do nome de utilizador
#define MAX_IP_ADDR                15            // dimensão máximo de endereço IP



// status codes

#define STATUS_OK                  201

#define CONNECTION_ERR             -111
#define COMM_ERROR                 -1

#define UNKNOWN_USER               403
#define ERR_TOPIC_DUPLICATE        432
#define UNKNOWN_GAME_TYPE          461

#define SERVER_ERROR               500


// callbacks de comunicação
typedef  void (*RespostaEventHandler)(int status, const char Resposta[]);
typedef  void (*MsgEventHandler)(const char sender[], const char msg[]);



/**
 * Para ligação ao servidor de jogos
 * Parâmetros:
 *   ip_addr: Endereço ip do servidor.
 *            Caso tenha arrancado o servidor na máquina local utilize o endereço
 *            de loopback (127.0.0.1)
 *   user:    nome do utilizador (já previamente registado no servidor)
 *   on_server_resp:
 *            função que vai receber e processar as respostas do servidor aos comandos enviados
 *            Nalguns casos as respostas apenas contêm um status de erro ou sucesso, noutros
 *            incluem informação adicional
 *  on_joiner_msg:
 *            função que vai receber e processar as mensagens originadas pelos adversários do jogo
 *
 *  Retorno:
 *     A função retorna um valor do tipo session_t que em caso de ligação
 *     bem sucedida será enviada nos comandos posteriores
 *   O status da reposta ao pedido de ligação pode ser:
 *                STATUS_OK – ligação bem sucedida
 *                CONNECTION_ERR – O servidor não está disponível
 *                UNKNOWN_USER – O utilizador indicado não está registado no servidor
 */
session_t srv_connect(const char ip_addr[],  const char user[],
                      RespostaEventHandler on_server_resp,
                      MsgEventHandler on_joiner_msg);
```

```
/**
 * criar (ou associar a) um novo jogo
 * Parâmetros:
 *    session: representante da sessão com o servidor obtido na chamada a srv_connect
 *    game_type: para a batalha naval será battleship
 *    game:  nome do jogo a criar (ou associar)
 *    O status da reposta ao pedido de criação de jogo pode ser:
 *      STATUS_OK – Criação bem sucedida. Neste caso a resposta indica se o jogador é o primeiro ou
 *                  o segundo a começar (rank) e qual o nome do adversário, com o formato:
 *                  RANK OPPONENT
 *      UNKNOWN_GAME_TYPE – Tipo de jogo inválido
 *      COMM_ERR    – Erro de comunicação
 */
void srv_new_game(session_t session, const char game_type[], const char game[]);



/**
 * enviar a jogada
 * Parâmetros:
 *    session: representante da sessão com o servidor obtido na chamada a srv_connect
 *    msg: mensagem com a jogada (depende do jogo)
 *    O status da reposta ao envio da jogada pode ser:
 *    STATUS_OK – Jogada entregue ao servidor com sucesso
 *    COMM_ERR  – Erro de comunicação
 *
 */
void srv_play(session_t session, const char msg[]);


/**
 * enviar o resultado da jogada do adversário
 * Parâmetros:
 *    session: representante da sessão com o servidor obtido na chamada a srv_connect
 *    msg: mensagem com o resultado (depende do jogo)
 *    O status da reposta ao envio do resultado pode ser:
 *    STATUS_OK – Jogada entregue ao servidor com sucesso
 *    COMM_ERR  – Erro de comunicação
 *
 */
void srv_send_result(session_t game_session, const char msg[]);



/**
 * terminar a sessão com o servidor
 *    session: representante da sessão com o servidor obtido na chamada a srv_connect
 */
void srv_close_session(session_t session);
```