BIBLIOTECA ALLEGRO

Alexandre Gomes da Costa
alexandre0851@hotmail.com

MATERIAIS E MANUAL

Materiais e manual

- O manual (em inglês) com a descrição e modo de utilização de todas as funções (métodos), encontra-se em: www.allegro.cc/manual
- Material de apoio e exemplos se encontram na página pessoal do Prof. Dr. Simão: www.pessoal.utfpr.edu.br/jeansimao/Fundamentos1/Fundamentos1.htm

O QUE SERIA ALLEGRO?

O que seria Allegro?

- Originalmente foi desenvolvida para ser empregada na plataforma Atari.
- Atualmente a biblioteca apresenta um conjunto amplo de funcionalidades, o que facilita a implementação de jogos, por exemplo.
- Permite outras aplicações, desde a criação de softwares mais complexos (editor de imagens) até funcionar como um "tocador" de MP3.

POR QUE A UTILIZAR?

Por que a utilizar?

- Por possuir funções de fácil aplicação e entendimento.
- Permite uma relação usuário/processo (entenda-se o programa rodando) mais simples.
- Disponibiliza uma série de recursos de um modo quase imediato.

ANÁLISE DOS MÉTODOS

Análise dos métodos

- Antes de ser feita uma análise completa de um programa, deve-se levar em conta que essa biblioteca, quando utilizada, possui uma programação em janelas.
- Por tal motivo, alguns detalhes e cuidados devem ser tomados para que não ocorram problemas durante a execução do programa.

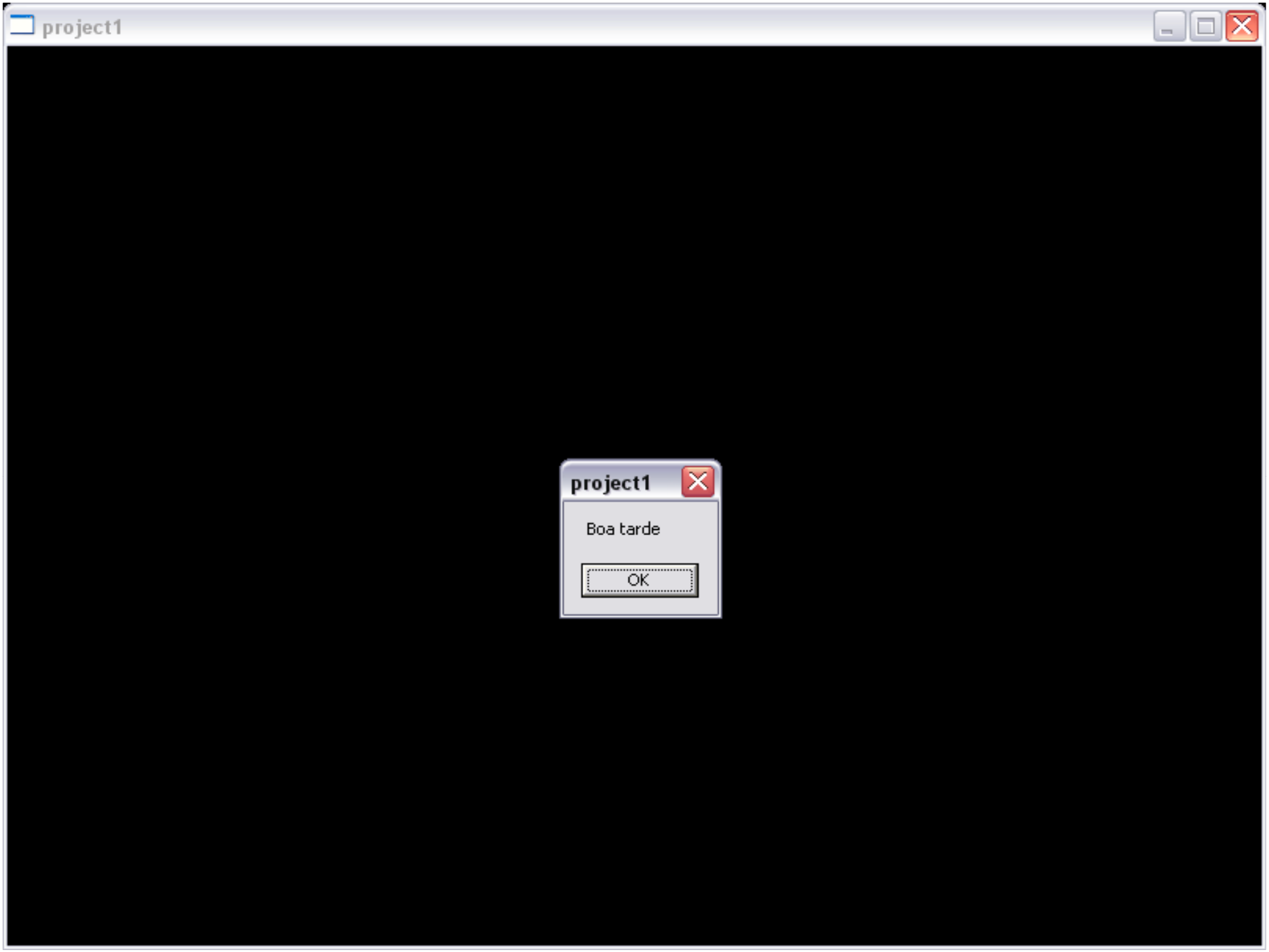
Análise dos métodos

- É interessante que o programador tenha uma organização do projeto, pois isso facilita na detecção e correção de erros de programação (normalmente é o que acontece).
- Montar uma estrutura lógica (esboço) do que será feito ANTES de fazer, isso minimiza as chances de erros.
- Trabalho em equipe, se este funcionar bem, aumenta a produtividade. Comunicação e entendimentos de ambas as partes é essencial.
- Uma boa prática de programação (caso seja necessário utilizar um recurso, quando terminar de usar, libere ele, pois é outra fonte de erros).

Análise dos métodos

- A seguir um exemplo de código simples que irá abrir uma janela e uma caixa de diálogo:

```
#include <allegro.h>
#include <stdlib.h>

int main()
{
    allegro_init();

    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);

    install_keyboard();
    allegro_message("Boa tarde");

    remove_keyboard();
    allegro_exit();

    return 0;
}
END_OF_MAIN();
```

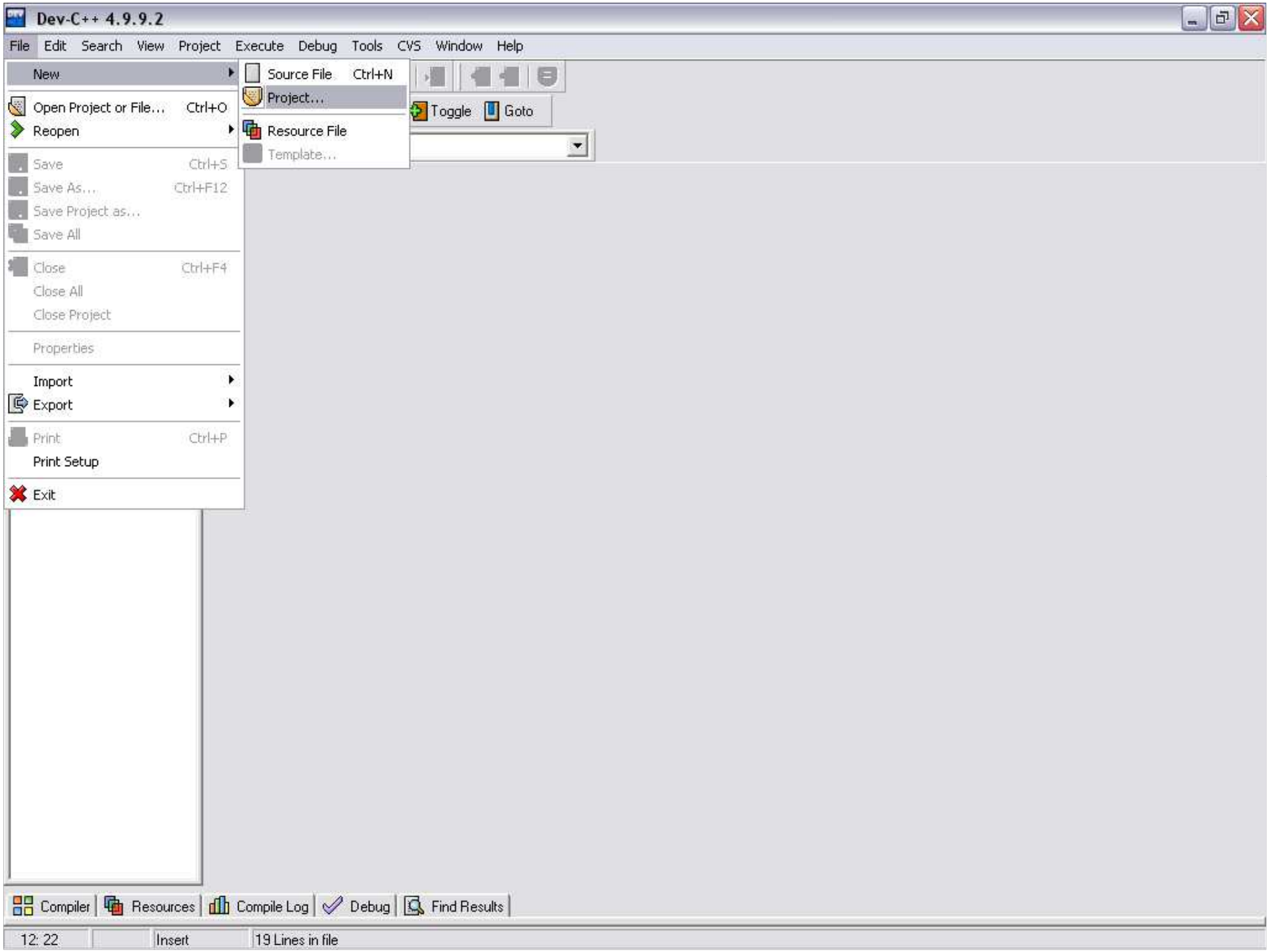

project1
project1
Boa tarde
OK

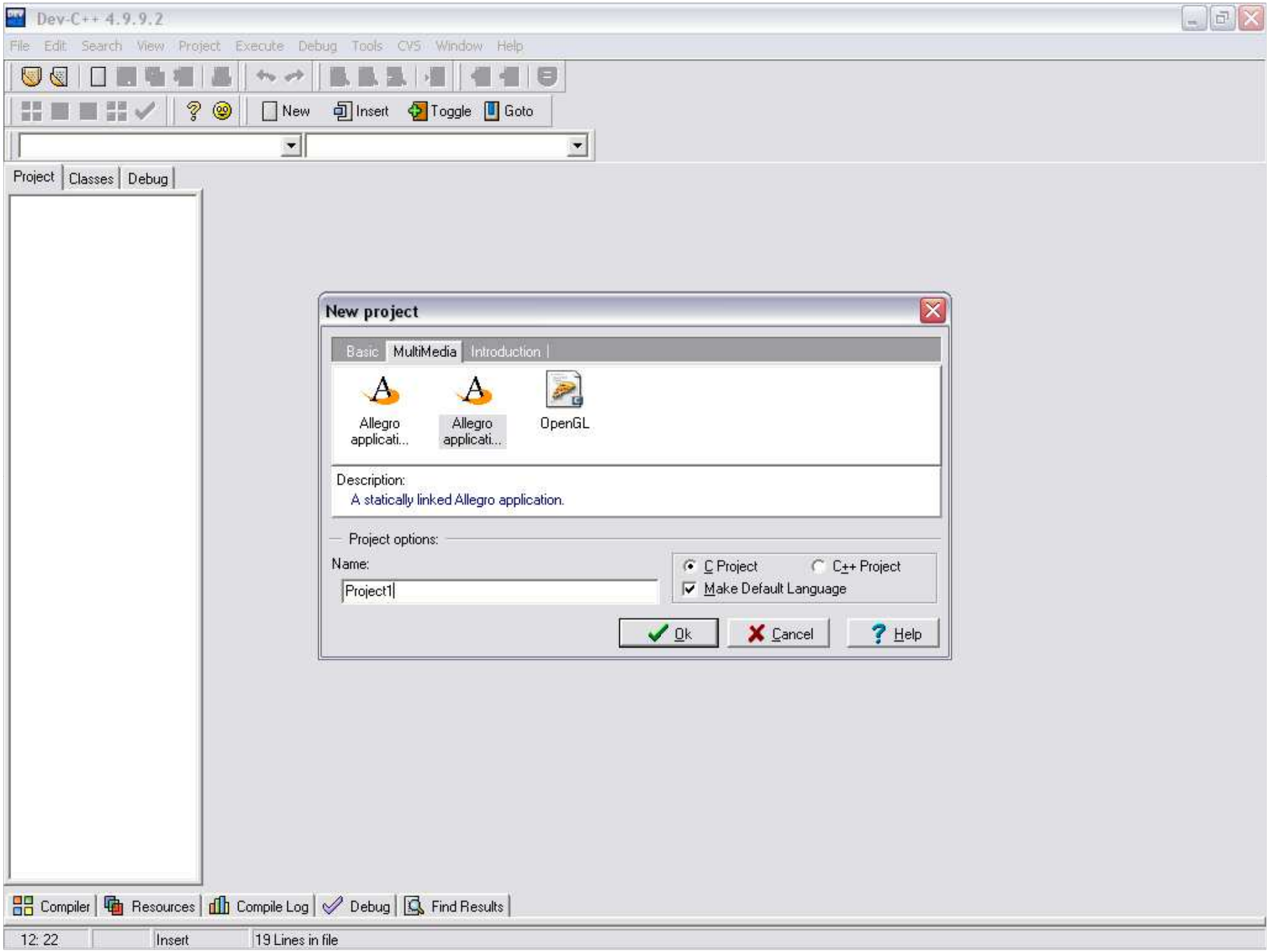
Análise dos métodos

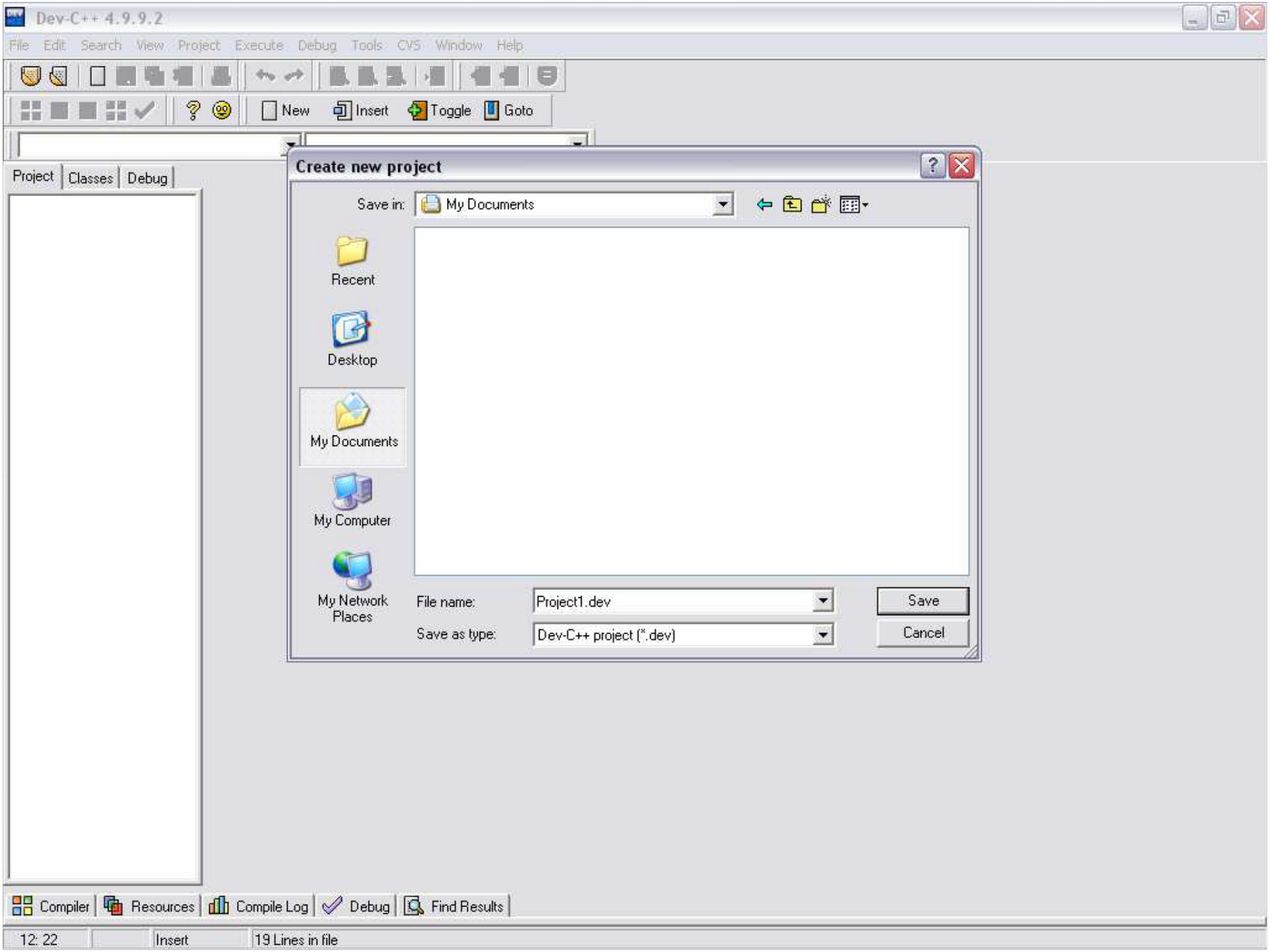
- Para obter tal resultado, foi utilizado o compilador Bloodshed Dev-C++ com a biblioteca Allegro instalada. Caso seja utilizada o Microsoft Visual Studio Express Edition existe uma outra apresentação que trata disso.
- Outro ponto que deve ser ressaltado é que se faz necessária a criação de um novo projeto ANTES de compilar o código, do contrário, não serão incluídos ao projeto os linkers necessários para a sua utilização.

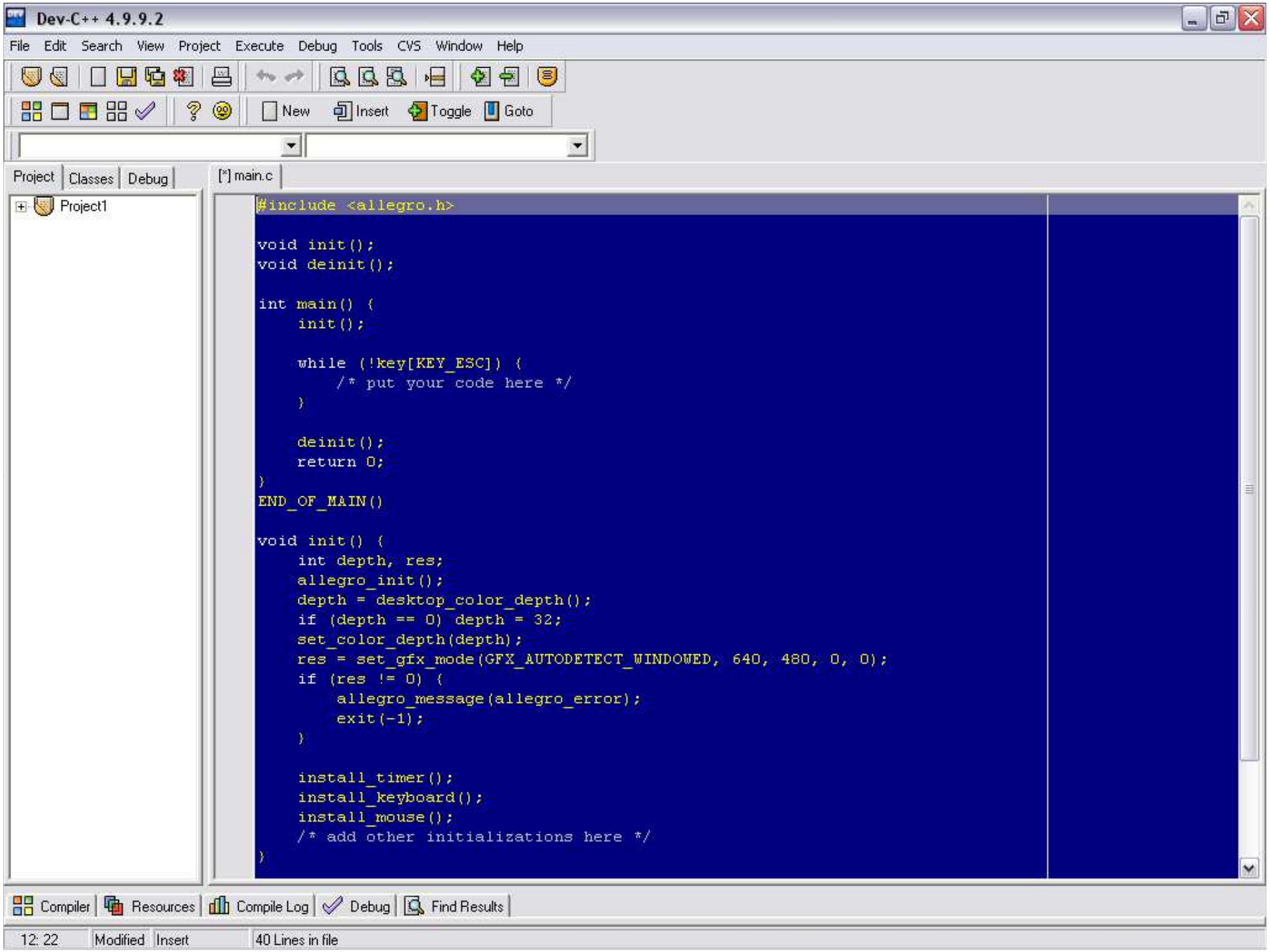
Análise dos métodos

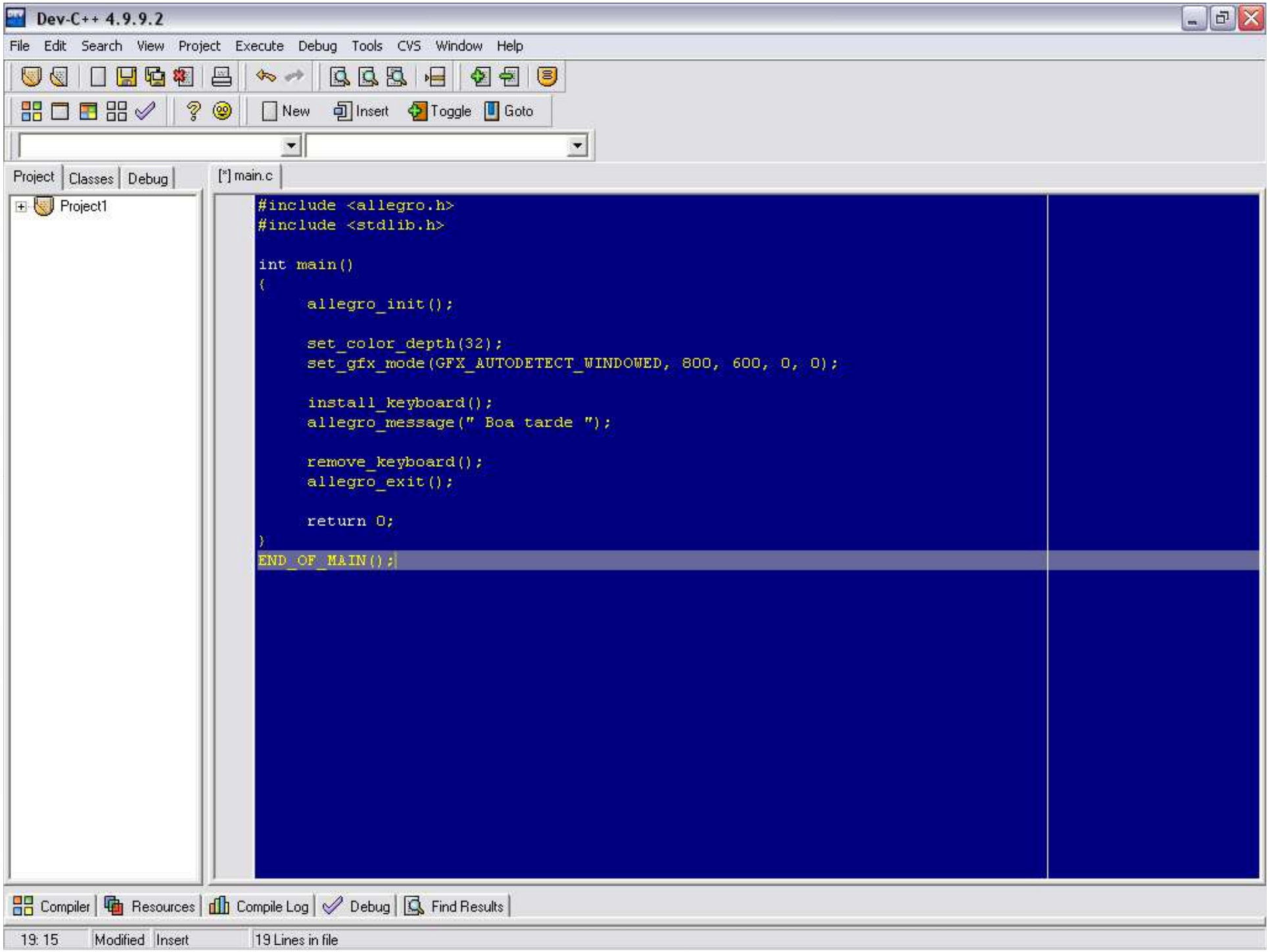
- O próximo conjunto de slides mostram como se deve realizar a criação de um novo projeto, utilizando a biblioteca Allegro.

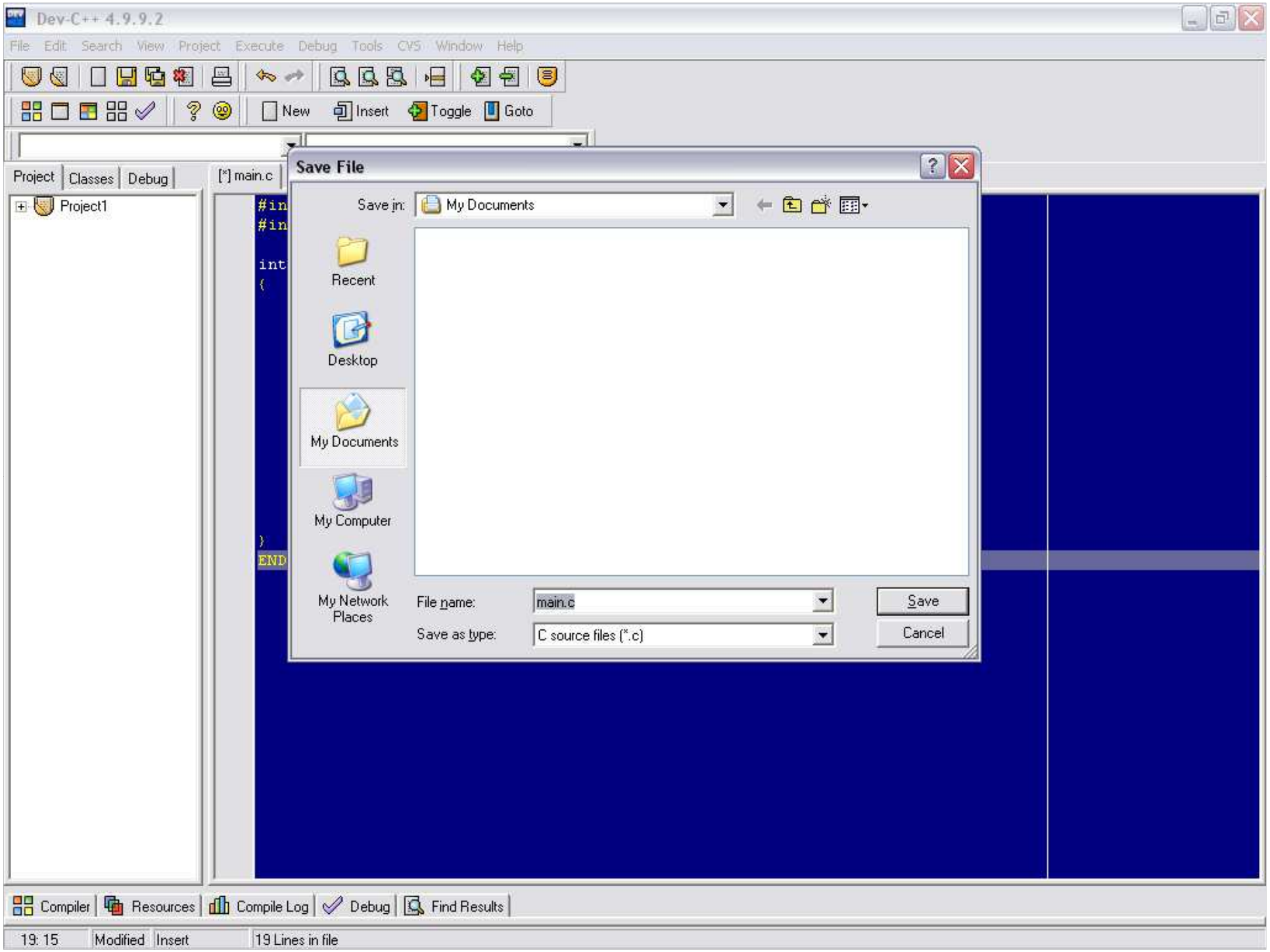

Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New
Open Project or File... Ctrl+O
Reopen
Save Ctrl+S
Save As... Ctrl+F12
Save Project as...
Save All
Close Ctrl+F4
Close All
Close Project
Properties
Import
Export
Print Ctrl+P
Print Setup
Exit
Source File Ctrl+N
Project...
Resource File
Template...
Toggle
Goto
Compiler
Resources
Compile Log
Debug
Find Results
12: 22
Insert
19 Lines in file

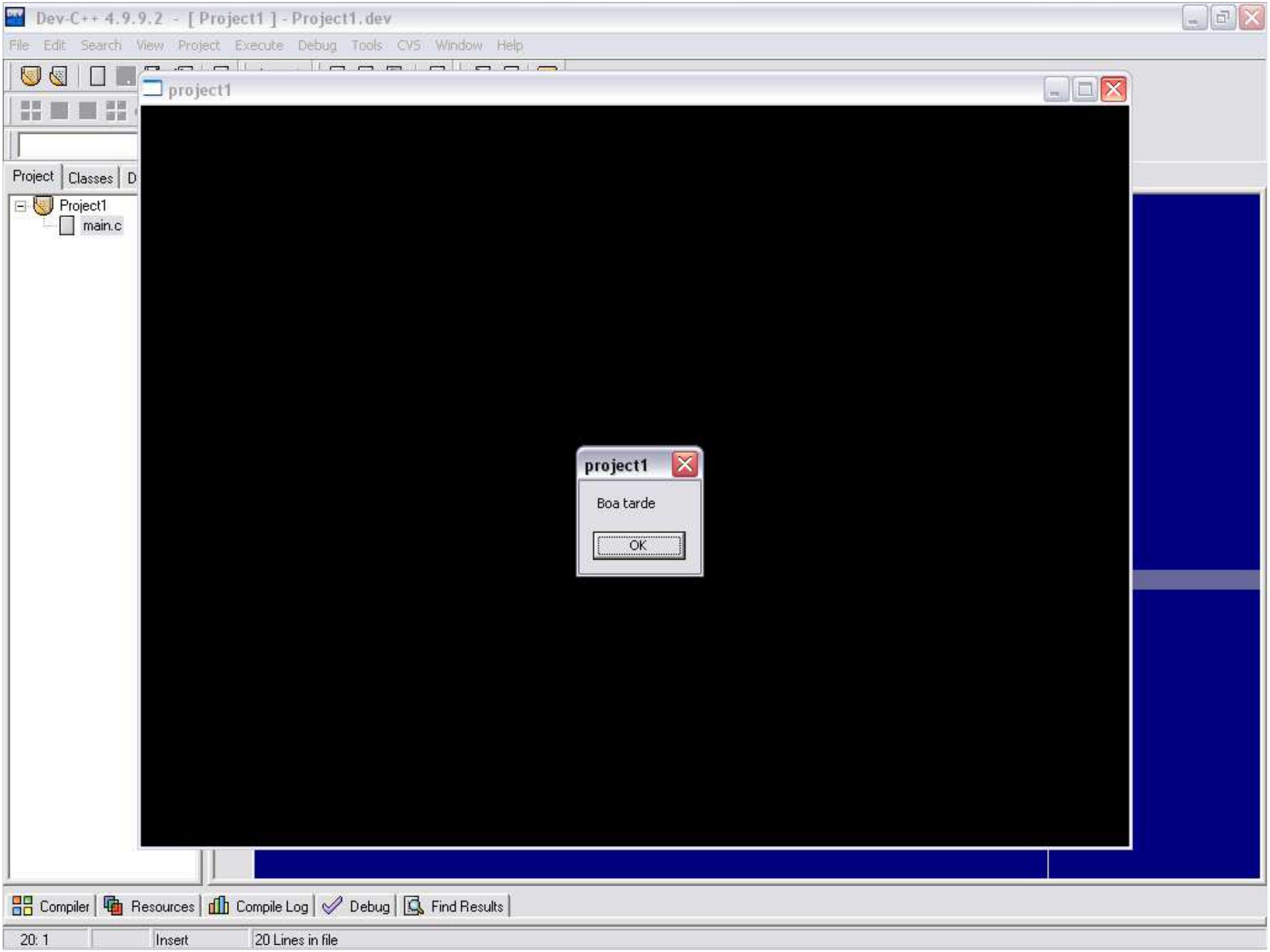

Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
New project
Basic MultiMedia Introduction
Allegro applicati...
Allegro applicati...
OpenGL
Description:
A statically linked Allegro application.
Project options:
Name:
Project1
C Project
C++ Project
Make Default Language
Ok
Cancel
Help
Compiler Resources Compile Log Debug Find Results
12: 22
Insert
19 Lines in file


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Create new project
Save in: My Documents
Recent
Desktop
My Documents
My Computer
My Network Places
File name: Project1.dev
Save as type: Dev-C++ project (*.dev)
Save
Cancel
Compiler Resources Compile Log Debug Find Results
12: 22 Insert 19 Lines in file


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project1
[*] main.c
#include <allegro.h>
void init();
void deinit();
int main() {
init();
while (!key[KEY_ESC]) {
/* put your code here */
}
deinit();
return 0;
}
END_OF_MAIN()
void init() {
int depth, res;
allegro_init();
depth = desktop_color_depth();
if (depth == 0) depth = 32;
set_color_depth(depth);
res = set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
if (res != 0) {
allegro_message(allegro_error);
exit(-1);
}
install_timer();
install_keyboard();
install_mouse();
/* add other initializations here */
}
Compiler Resources Compile Log Debug Find Results
12: 22 Modified Insert 40 Lines in file


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project1
[*] main.c
#include <allegro.h>
#include <stdlib.h>
int main()
{
allegro_init();
set_color_depth(32);
set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
install_keyboard();
allegro_message(" Boa tarde ");
remove_keyboard();
allegro_exit();
return 0;
}
END_OF_MAIN();
Compiler Resources Compile Log Debug Find Results
19: 15 Modified Insert 19 Lines in file


Dev-C++ 4.9.9.2
File Edit Search View Project Execute Debug Tools CVS Window Help
New Insert Toggle Goto
Project Classes Debug
Project1
[*] main.c
#in
#in
int
{
}
END
Save File
Save in: My Documents
Recent
Desktop
My Documents
My Computer
My Network Places
File name: main.c
Save as type: C source files (*.c)
Save
Cancel
Compiler Resources Compile Log Debug Find Results
19: 15 Modified Insert 19 Lines in file


Dev-C++ 4.9.9.2 - [Project1] - Project1.dev
File Edit Search View Project Execute Debug Tools CVS Window Help
project1
Project Classes D
Project1
main.c
project1
Boa tarde
OK
Compiler Resources Compile Log Debug Find Results
20: 1 Insert 20 Lines in file

Análise dos métodos

- Nesse ponto será analisado de fato o código, pois o programador deve ter conseguido rodar com sucesso o exemplo anterior.
- Do contrário, reinstale o compilador, a biblioteca ou reveja o código.

Análise dos métodos

- #include <allegro.h> - A grosso modo, “inclue” de fato a biblioteca no programa.
- allegro_init(); - Antes de qualquer outro método relacionado a biblioteca, utilize sempre PRIMEIRO esse método, pois apenas depois disso será possível utilizar todos os métodos da biblioteca.
- set_color_depth(32); - Antes de definir o driver de vídeo, é necessário definir a resolução das cores, seus valores possíveis são: 8, 15, 16, 24 e 32 bits (no exemplo, foi utilizado 32 bits);
- set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0); - Serve para definir a janela de trabalho, o primeiro parâmetro é um enum que em tempo de execução o programa identifique qual é o tipo da placa de vídeo, carrega as informações necessárias para sua utilização e em um modo de janela. Caso o programador queira que seja em modo tela cheia, utilize o seguinte parâmetro: GFX_AUTODETECT_FULLSCREEN. O segundo e o terceiro parâmetro são os que definem as dimensões da tela em pixels (no caso, 800x600). Os dois últimos definem a tela virtual. Para uma primeira aproximação, o recomendado é ambos serem iguais a 0.

Análise dos métodos

- install_keyboard(); - Permite que o usuário utilize o teclado para executar alguma ação.
- allegro_message("Boa tarde"); - Caixa de diálogo. Pode ser invocada a qualquer instante posterior a devida inicialização. Uma possível utilização é como um auxílio ao debug.
- remove_keyboard(); - Desaloca o recurso utilizado faz parte da boa prática da programação (pode evitar algum erro crítico, um deadlock por exemplo).
- allegro_exit(); - Permite que o programador não tenha que desalocar todos os recursos utilizados, contudo caso tenha alocado dinamicamente memória por algum motivo, esse método NÃO libera a memória utilizada, deve-se utilizar métodos próprios conforme o caso para lidar com essa situação (não desalocar memória é uma das maiores fontes de erro, logo uma atenção extra é necessária).
- END_OF_MAIN(); - Método que deve estar sempre presente no final da int main(), basicamente informa a biblioteca que terminou o seu fluxo de execução e que deve terminar o processo.

DESCRIÇÃO DOS MÉTODOS

Descrição dos métodos

- Inicialização:

1. int allegro_init();
2. void set_color_depth(int depth);
3. int set_gfx_mode(int card, int w, int h, int v_w, int v_h);
4. void request_refresh_rate(int rate);
5. int install_mouse();
6. int install_keyboard();
7. int install_joystick(int type);
8. int install_sound(int digi, int midi, const char *cfg_path);
9. void set_window_title(const char *name);

Descrição dos métodos

1. Trata da inicialização da biblioteca.
2. Define a resolução de cores.
3. Define o driver de vídeo, bem como a configuração da janela a ser utilizada.
4. Taxa de atualização do vídeo (varia de acordo com a configuração do desktop, pois taxas não toleradas não serão executadas).
5. Inicia o módulo do mouse.
6. Inicia o módulo do teclado.
7. Inicia o módulo do joystick.
8. Inicia o módulo de áudio. Utilização comum: install_sound(DIGI_AUTODETECT , MIDI_AUTODETECT , NULL);
9. Define o título da janela.

Descrição dos métodos

- Utilização:

  Mouse:

1. void set_mouse_sprite(BITMAP *sprite);
2. extern volatile int mouse_x;
3. extern volatile int mouse_y;
4. void show_mouse(BITMAP *bmp);
5. void scare_mouse();
6. extern volatile int freeze_mouse_flag;
7. void set_mouse_range(int x1, int y1, int x2, int y2);
8. void set_mouse_speed(int xspeed, int yspeed);

Descrição dos métodos

1. Define o BITMAP (desenho, ícone) do cursor do mouse a ser utilizado na aplicação.
2. É usado como o valor da coordenada X do cursor em relação a tela.
3. É usado como o valor da coordenada Y do cursor em relação a tela.
4. Define onde a imagem do cursor do mouse será exibido (padrão: show_mouse(screen);).
5. "Esconde" o cursor.
6. Evita que o cursor seja atualizado com frequência (evita que fique "piscando").
7. Define o retângulo de ação do mouse.
8. Define a velocidade de deslocamento no eixo X e no eixo Y (padrão: set_mouse_speed(2, 2);).

Descrição dos métodos

- Utilização:

  Teclado:

1. extern volatile char key[KEY_MAX];
2. int keypressed();
3. void clear_keybuf();

Descrição dos métodos

1. Caso alguma tecla específica tenha sido apertada, é atribuído *true* ao evento, do contrário, é *false*. Serve para recuperar informações do usuário ou utilizar o teclado para movimentar um personagem dentro de um jogo, por exemplo.
2. Segue o mesmo princípio do key[KEY_MAX];, contudo o valor retornado é o seu código scancode (usado pelo Allegro), podendo ser utilizado também para ser um limitador de operações (executa uma parte do código se e somente se uma tecla qualquer for pressionada).
3. Limpa o buffer de memória relacionado ao teclado (é útil quando se trabalha com loops recuperando informação).

Descrição dos métodos

- Utilização:

  BITMAP:

1. BITMAP *create_bitmap(int width, int height);
2. void destroy_bitmap(BITMAP *bitmap);
3. extern BITMAP *screen;
4. #define SCREEN_W;
5. #define SCREEN_H;
6. void clear_bitmap(BITMAP *bitmap);
7. void _putpixel(BITMAP *bmp, int x, int y, int color);

Descrição dos métodos

8. int _getpixel(BITMAP *bmp, int x, int y);
9. void vline(BITMAP *bmp, int x, int y1, int y2, int color);
10. void hline(BITMAP *bmp, int x1, int y, int x2, int color);
11. void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
12. void fastline(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
13. void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
14. void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int color);
15. void circle(BITMAP *bmp, int x, int y, int radius, int color);
16. void circlefill(BITMAP *bmp, int x, int y, int radius, int color);

Descrição dos métodos

17. void blit(BITMAP *source, BITMAP *dest, int source_x, int source_y, int dest_x, int dest_y, int width, int height);
18. void stretch_blit(BITMAP *source, BITMAP *dest, int source_x, source_y, source_width, source_height, int dest_x, dest_y, dest_width, dest_height);
19. void draw_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y);
20. void stretch_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int w, int h);
21. void rotate_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, fixed angle);
22. void pivot_sprite(BITMAP *bmp, BITMAP *sprite, int x, int y, int cx, int cy, fixed angle);
23. BITMAP *load_bitmap(const char *filename, RGB *pal);
24. int makecol32(int r, int g, int b);

Descrição dos métodos

1. Um objeto da classe BITMAP. Utilizando-se desse método, é possível alocar dinamicamente memória para desenhar, escrever ou colocar imagens dentro das dimensões limitadas por *width* e *height*. Como foi alocada memória, esta DEVE ser liberada após o uso. Utilização:

   BITMAP *Imagem = create_bitmap(800, 600);

1. É usado para liberar a memória alocada para um BITMAP específico. Utilização: destroy_bitmap(Imagem);
2. É um BITMAP especial que trata da tela que o usuário vê. Qualquer imagem que se deseja mostrar, tem que ser "impressa" nesse BITMAP.
3. Variável global que contém a informação da largura da tela que o programador definiu.
4. Variável global que contém a informação da altura da tela que o programador definiu.
5. Limpa o BITMAP, apagando o que tem nele. Pode ser invocada a qualquer instante.
6. Permite colocar um pixel num local de destino (screen), numa posição dada (x, y) e com uma cor (color). O programador deve ter conhecimento que existem métodos similares com outros formatos que são usados caso a resolução de cores tenha sido informada. Caso tenha sido executado set_color_depth(16);, é necessário então usar _putpixel16(...);

Descrição dos métodos

8. Retorna a cor de um dado pixel em um dado BITMAP em uma posição definida. Também deve ser utilizado de acordo com a resolução de cores (_getpixel16(...); utilizado no exemplo anterior).
9. Desenha um linha vertical em um dado BITMAP, com a posição do ponto de origem (x, y1) e a componente Y do final (y2), usando uma cor definida (color).
10. Desenha um linha vertical em um dado BITMAP, com a posição do ponto de origem (x1, y) e a componente X do final (x2), usando uma cor definida (color).
11. Desenha um linha qualquer em um dado BITMAP, com a posição do ponto de origem (x1, y1) e a componente final (x2, y2), usando uma cor definida (color).
12. Similar ao line(...), contudo executado de um modo mais rápido.
13. Desenha as bordas de um retângulo em um dado BITMAP, com o canto superior esquerdo em x1 e y1, o canto inferior direito em x2, y2 e com a cor color.
14. Similar ao rect(...), contudo desenha um retângulo preenchido com a cor definida.
15. Desenha as bordas de um círculo em um dado BITMAP, com o centro em x e y, com o raio de radius e com a cor color.
16. Similar ao circ(...), contudo desenha um círculo preenchido com a cor definida.

Descrição dos métodos

17. Cópia de um BITMAP (source) em outro BITMAP (dest), definido pelo ponto da fonte (source_x, source_y) no ponto do destino (dest_x, dest_y) com o tamanho width e height em relação a esses pontos.
18. Similar ao blit, contudo caso a área especificada do destino fosse maior que o da fonte, existiriam espaços vazios. Se a intenção era "esticar" a imagem até os limites do destino, este método deve ser utilizado.
19. Desenha um BITMAP de uma origem (sprite) em um BITMAP destino (bmp) a partir da coordenada x e y (NÃO SE DEVE CONFUNDIR draw_sprite(...) COM blit(...), APESAR DE EXECUTAREM QUASE AS MESMAS FUNCIONALIDADES).
20. Similar ao strecht_blit(...), contudo levando em conta que é desenhado, não copiado.
21. Rotaciona um BITMAP, desenhando em x e y, com a rotação determinada por angle em relação ao centro da imagem.
22. Similar ao rotate_sprite(...), diferencia pois o programador define qual é o ponto de rotação (cx, cy).
23. Similar ao create_bitmap(...), contudo é mais interessante pois é possivel carregar uma imagem no formato bmp. Utilização: BITMAP Imagem = load_bitmap("imagem.bmp", NULL). Alguns detalhes que devem ser levados em conta. Caso o programador execute duas vezes esse método SEM liberar o espaço alocado, a imagem anterior ainda existirá na memória, contudo não será mais acessível. Se a utilização do programa for longa, pode estorar o limite da memória, forçando o programa a ser fechado, isso se não ocorrerem outros problemas. O parâmetro RGB *pal é uma palheta de cores, podendo ser omitido. Contudo se o programador deseja utilizar (criando um sistema de cores), deve criar um objeto do tipo PALETTE e executar ações referentes a isso. Inicialmente é recomendado deixar esse parâmetro como NULL.
24. Nos métodos anteriores que pediam o parâmetro int color, deve-se utilizar esse método, pois ele converte a escala RGB informada pelo programador em um valor utilizado pelo Allegro. Outro ponto que deve ser mencionado é que esse método também depende da resolução de cores utilizada. Utilização: rect(Imagem, 100, 100, 200, 200, makecol32(255, 255, 255));. Caso a resolução seja de 16 bits, é usado makecol16(....). Os parametros do makecol (r, g, b) são os valores da cor vermelha, verde e azul respectivamente, que variam entre 0 e 255.

Descrição dos métodos

- Utilização:

  SAMPLE/MIDI

1. void set_volume(int digi_volume, int midi_volume);
2. SAMPLE *load_sample(const char *filename);
3. void destroy_sample(SAMPLE *spl);
4. int play_sample(const SAMPLE *spl, int vol, int pan, int freq, int loop);
5. void stop_sample(const SAMPLE *spl);
6. MIDI *load_midi(const char *filename);
7. void destroy_midi(MIDI *midi);
8. int play_midi(MIDI *midi, int loop);
9. void stop_midi();

Descrição dos métodos

1. Atribui o valor especificado para as intensidades do som. O primeiro parâmetro se referencia ao som SAMPLE e o segundo ao som MIDI. SAMPLE é uma classe que trata de sons no formato .wav e .voc. Já a MIDI, trata apenas de .mid. A intensidade varia entre 0 e 255.
2. Alocação de memória para um SAMPLE. Utilização: SAMPLE *Som = load_sample("som.wav");.
3. Libera o espaço de memória utilizado por um determinado SAMPLE. Como ressaltado algumas vezes, é interessante sempre chamar esse método quando não é mais útil um SAMPLE. Utilização: destroy_sample(Som);
4. "Toca" de fato o SAMPLE para o usuário ouvir em um determinado instante. Alguns parâmetros são necessários: SAMPLE *spl = SAMPLE a ser "tocado"; int vol = intensidade inicial (0 a 255); int pan = distribuição do som nas caixas (0 a 255, onde 0 existe apenas na caixa esquerda e 255 na caixa direita); int freq = indica a velocidade como será "tocada" (1000 para a mesma velocidade, 2000 para o dobro e 500 para a metade da velocidade); int loop = quantidade de vezes que será "tocada".
5. Pára o fluxo de execução de um determinado SAMPLE. Utilização: stop_sample(Som);.
6. Similar ao load_sample(...);.
7. Similar ao destroy_sample(...);.
8. Similar ao play_sample(...);, contudo apenas o MIDI *midi e int loop devem ser informados.
9. Similar ao stop_sample(...);.

Descrição dos métodos

- Utilização:

  FONT

1. FONT *load_font(const char *filename, RGB *pal, void *param);
2. void destroy_font(FONT *f);
3. FONT *load_bitmap_font(const char *filename, RGB *pal, void *param);

Descrição dos métodos

1. Um objeto da classe FONT é criado (possue memória alocada). É um objeto útil, pois com ele é possível utilizar uma fonte que o programador escolheu para ser exibida em textos dentro do aplicativo. Outras aplicações são possíveis mais fogem do intuito deste documento. Utilização: FONT *Fonte = load_fonte("Fonte.pcx", NULL, NULL);. Algumas observações merecem ser feitas: O arquivo Fonte.pcx é gerado por um programa chamado TTF2PCX (disponível em: http://www.talula.demon.co.uk/ttf2pcx/ttf2p16.zip). O segundo parâmetro, como discutido anteriormente, é da palheta de cores utilizada que, a princípio, não é utilizada, logo esse parâmetro pode ser NULL. O último também não é interessante ser utilizado, pois é de uma rotina feita pelo programador para carregar essa fonte. Por tal motivo, ele também é NULL.
2. Novamente esse método é invocado quando não se faz mais necessária a utilização de um determinado FONT. Utilização: destroy_font(Fonte);.
3. Método versátil, sendo muito interessante em algumas aplicações dentro de um programa. Com este método é possível carregar uma fonte no formato .bmp. Contudo algumas modificações devem ser feitas: Ao se abrir o arquivo de uma fonte utilizada em .pcx, será notada que ela é monocromática (ou próximo disso). O fundo é (255, 255, 255) (em RGB), o contorno da letra é (0, 0, 0) e a letra é (254, 254, 254). Para utilizar ela como sendo uma entidade .bmp, o fundo deve ser mudado para (255, 0, 255) (magenta); o contorno para (255, 255, 0) (índigo) e a letra, o programador escolhe qual cor esta será. Após estas modificações, esse arquivo deve ser salvo em .bmp (de preferência em 24 bits para não ter informações perdidas) e assim pode ser utilizado normalmente. Vantagens desse método em relação ao primeiro: Quando for colocar um texto na tela, o primeiro método deixa uma "caixa preta" no redor do texto, podendo ser um efeito inesperado. Já esse método, anula isso. Contudo o ponto contra é que apenas a cor definida no .bmp da letra será utilizada, independentemente da cor que o programador informar na hora de digitar um texto.

Descrição dos métodos

- Utilização:

Outros

1. fixed itofix(int x);
2. int text_length(const FONT *f, const char *str);
3. int text_height(const FONT *f)
4. void textout_ex(BITMAP *bmp, const FONT *f, const char *s, int x, int y, int color, int bg);
5. void textout_centre_ex(BITMAP *bmp, const FONT *f, const char *s, int x, y, int color, int bg);
6. void textout_right_ex(BITMAP *bmp, const FONT *f, const char *s, int x, int y, int color, int bg);
7. void textprintf_ex(BITMAP *bmp, const FONT *f, int x, int y, int color, int bg, const char *fmt, ...);
8. void textprintf_centre_ex(BITMAP *bmp, const FONT *f, int x, int y, int color, int bg, const char *fmt, ...);
9. void textprintf_right_ex(BITMAP *bmp, const FONT *f, int x, y, color, bg, const char *fmt, ...);

Descrição dos métodos

1. Retorna um valor corrigido do ângulo em graus para o ângulo que o Allegro utiliza. Observações: 256 = 360°, se valor negativo ou positivo interfere na rotação da figura. Utilização: rotate_sprite(screen, Imagem, 10, 10, itofix(45));
2. Para um texto a ser escrito na tela, é possível calcular qual será sua largura e altura em pixels. Esse método retorna qual será o tamanho utilizado para escrever esse texto com uma dada fonte.

   Utilização: int larg = text_lenght(Fonte,"ALGUMA COISA"); Observação: Caso o programador optou por não usar uma fonte definida por ele, pode-se usar a do Allegro utilizando no lugar de Fonte o parâmetro font.
3. Retorna a altura de um texto. Requer apenas o FONT como parâmetro.
4. Serve para escrever um texto na tela sem justificação.

   Utilização: textout_ex(screen, Fonte, 10, 20, "TEXTO", makecol(255, 255, 255), -1); O parâmetro BITMAP *bmp é onde o texto será escrito (para colocar na tela, utiliza-se screen); FONT *f é a fonte utilizada; const char *s é o texto a ser escrito; int x e int y são as coordenadas do texto ser colocado na tela; int color é a cor a ser utilizada no texto, no caso, será branco; int bg escreve o texto com transparência.
5. Similar ao textout_ex(...), contudo é justificado em relação ao centro.
6. Similar ao textout_ex(...), contudo é justificado em relação à direita.
7. Similar ao textout_ex(...), contudo neste é possível colocar variáveis no texto (no anterior, as variáveis a serem utilizadas já deveriam estar presentes no texto). Utilização: textprintf_ex(screen, Fonte, 10, 20, makecol(255, 255, 255), -1, "Olá %s", Texto);. Onde o Texto seria um vetor de caracteres que contém o nome do usuário, por exemplo.
8. Similar ao textprintf_ex(...), contudo é justificado em relação ao centro.
9. Similar ao textprintf_ex(...), contudo é justificado em relação à direita.

Descrição dos métodos

- Destrutoras:

1. void remove_keyboard();
2. void remove_mouse();
3. void remove_sound();
4. void remove_joystick();
5. void allegro_exit();
6. void rest(unsigned int time);

Descrição dos métodos

1. Desliga o módulo do teclado.
2. Desliga o módulo do mouse.
3. Desliga o módulo do som.
4. Desliga o módulo do joystick.
5. Desliga os módulos utilizados pelo Allegro (normalmente os métodos supracitados podem ser substituidos apenas por esse ao final do programa).
6. Tempo de espera dado em milissegundos (fazer o programa "dormir"). Pode ser empregado para aumentar o tempo de execução de um laço de repetição (controlador de tempo). Utilização: rest(1000); (tempo de espera de um segundo).

Descrição dos métodos

- Ao se utilizar as estruturas que tratam de carregar os arquivos, alguns cuidados devem ser tomados. É possível automatizar o processo, fazendo com que as imagem a serem recuperadas sejam carregadas a partir de um arquivo texto. Esse arquivo contém o caminho exato para as imagens. O cuidado que se deve ter é que caso o caminho fornecido não for válido (não existir a imagem nesse local) e o programador não tomou o cuidado para analisar antes de usar caso a imagem tenha sido propriamente carregada, ao se tentar usar pela primeira vez a imagem, acontecerá um *crash* no programa. Caso um local exato seja informado, deve seguir esse padrão. Exemplo: o programador quer acessar uma imagem que se encontra na pasta C:\Imagens\ cujo nome é Imagem.bmp. Para carregar esse bitmap, deve proceder da seguinte maneira: BITMAP *Imagem = load_bitmap("c://imagens//imagem.bmp", NULL);. Note as barras, caso não seja feito desta maneira, também acontecerá um erro. Agora caso queira carregar uma imagem que esteja na mesma pasta do programa, o modo alternativo para se carregar uma imagem é o seguinte: BITMAP *Imagem = load_bitmap("imagem.bmp", NULL);.

PONTOS IMPORTANTES

Pontos importantes

- A essa altura é possível constatar que a biblioteca possue muitas funcionalidades, outras foram omitidas pois dependem do que o programador queira fazer (é aconselhado a ler o manual e ver quais foram omitidas).
- Nesse momento serão abordado algumas formas de se programar utilizando os recursos que são ofertados.
- Noções básicas de colisão de bitmaps, controle da utilização dos recursos e otimização do algoritmo são o foco da atual discussão.

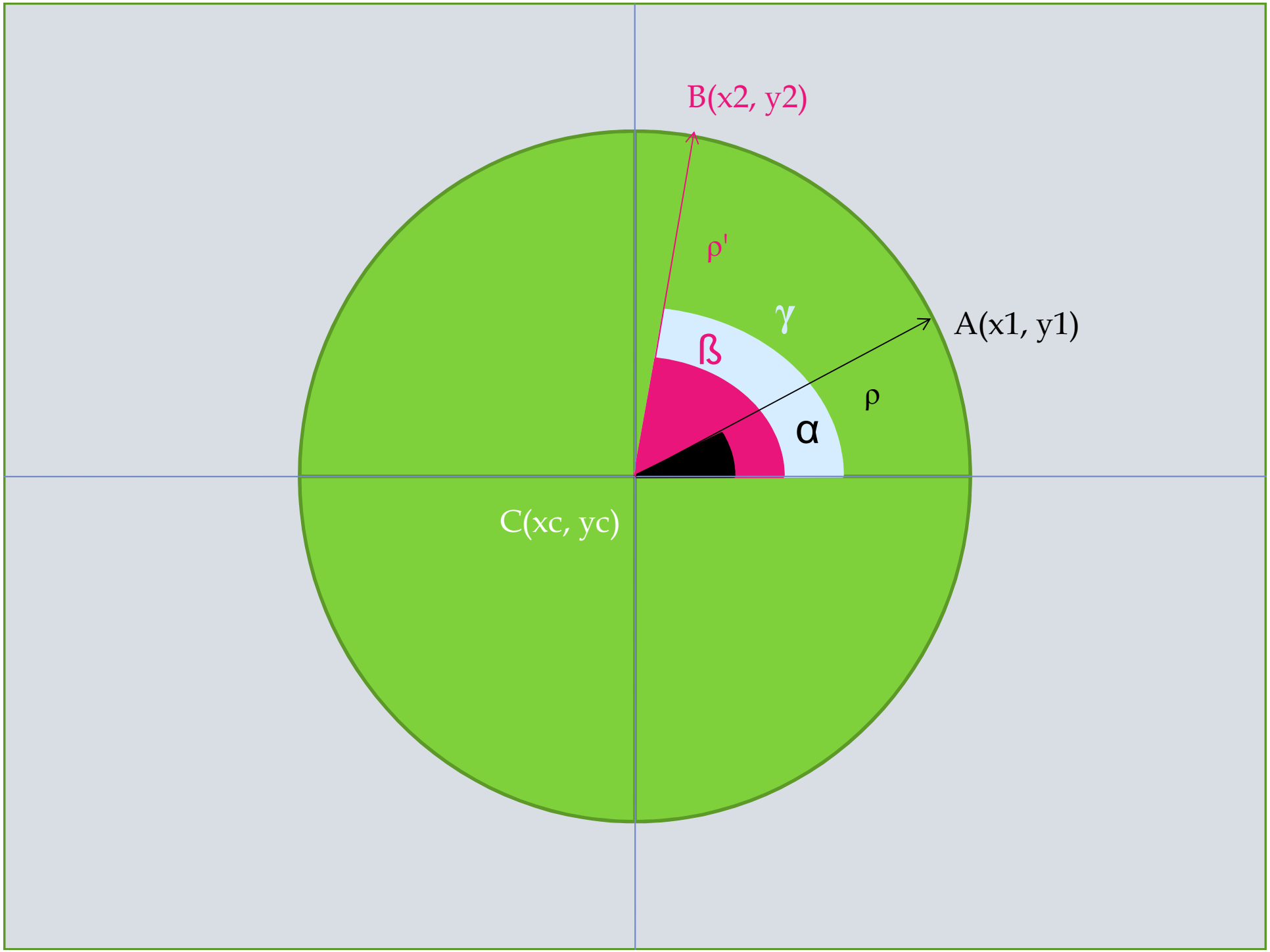
COLISÃO DE BITMAPS

Colisão de bitmaps

- Um dos pontos mais relevantes e mais importantes dentro de um jogo pode ser considerado a correta formulação de um sistema de colisão. Pois dependendo do jogo, tal evento é crucial para sua jogabilidade, ou é o foco principal do jogo (exemplo: sinuca).
- O grau de complexidade desse sistema depende e muito de quão perfeito que o programador quer que este seja ou então da geometria em questão. Hipóteses simplificadoras podem ser muito úteis na hora de sua implementação.

Colisão de bitmaps

- Antes de ser analisado o código bem como sua implementação, alguns conceitos básicos devem ser utilizados.
- Um deles é que para se trabalhar com o sistema de colisão, o programador deve ter em mente que apenas saber programar não é o suficiente para contornar algumas situações. Utilizar a sua experiência é um pré-requisito.
- Relembrando alguns conceitos de Matemática I é possível encontrar uma boa solução.


B(x2, y2)
ρ'
γ
ß
A(x1, y1)
ρ
α
C(xc, yc)

Colisão de bitmaps

- Pela figura anterior, é possível verificar que os pontos A e B possuem coordenadas x1, y1 e x2, y2 respectivamente e estão distantes de um centro de um círculo de ρ e ρ'. Como possuem um ponto em comum (a origem em xc e yc), é possível determinar que ρ e ρ' são função de x1, y1, xc, yc e x2, y2, xc, yc.

Colisão de bitmaps

$(x2-xc) = \rho'*\cos(ß)$ $\qquad$ $(y2-yc) = \rho'*\mathrm{sen}(ß)$

$ß = \alpha + \gamma \rightarrow \cos(ß) = \cos(\alpha + \gamma)$ | $\mathrm{sen}(ß) = \mathrm{sen}(\alpha + \gamma)$

$(x2-xc) = \rho'* \cos(\alpha + \gamma)$ $\qquad$ $(y2-yc) = \rho'* \mathrm{sen}(\alpha + \gamma)$

$$(x2-xc) = \rho'*[\cos(\alpha)*\cos(\gamma) - \mathrm{sen}(\alpha)*\mathrm{sen}(\gamma)]$$
$$(y2 -yc) = \rho'*[\mathrm{sen}(\alpha)*\cos(\gamma) + \mathrm{sen}(\gamma)*\cos(\alpha)]$$

$$x2 = \rho'*\cos(\alpha)*\cos(\gamma) - \rho'*\mathrm{sen}(\alpha)*\mathrm{sen}(\gamma) + xc$$
$$y2 = \rho'*\mathrm{sen}(\alpha)*\cos(\gamma) + \rho'*\mathrm{sen}(\gamma)*\cos(\alpha) + yc$$

Contudo sabemos ainda que: $\rho' = \kappa*\rho$, onde κ é um coeficiente de dilatação ou contração (depende do caso).

Colisão de bitmaps

$$x2 = \rho*\kappa*\cos(\alpha)*\cos(\gamma) - \rho*\kappa*\text{sen}(\alpha)*\text{sen}(\gamma) + xc$$
$$y2 = \rho*\kappa*\text{sen}(\alpha)*\cos(\gamma) + \rho*\kappa*\text{sen}(\gamma)*\cos(\alpha) + yc$$

Mas pela figura, obtemos que:

$$(x1-xc) = \rho*\cos(\alpha)$$
$$(y1-yc) = \rho*\text{sen}(\alpha)$$

$$x2 = \kappa*(x1-xc)*\cos(\gamma) - \kappa*(y1-yc)*\text{sen}(\gamma) + xc$$
$$y2 = \kappa*(y1-yc)*\cos(\gamma) + \kappa*\text{sen}(\gamma)*(x1-xc) + yc$$

Se $xc = yc = 0$ e $\kappa = 1$, concluimos que:

$$x2 = x1*\cos(\gamma) - y1*\text{sen}(\gamma)$$
$$y2 = y1*\cos(\gamma) + x1*\text{sen}(\gamma)$$

Colisão de bitmaps

- Resumindo: Dado dois vetores girantes (fatores) que possuem um ângulo inicial de α e ß, existe a diferença de fase de γ entre eles. Conhecendo as coordenadas do fasor inicial e o ângulo de defasagem, é possível determinar a posição do outro fasor em função do primeiro. Simplificando a equação retirando o efeito de dilatação e supondo que o centro (xc, yc) esteja na origem dos eixos, a relação se torna simples, nada mais do que a matriz de rotação usada em álgebra linear.

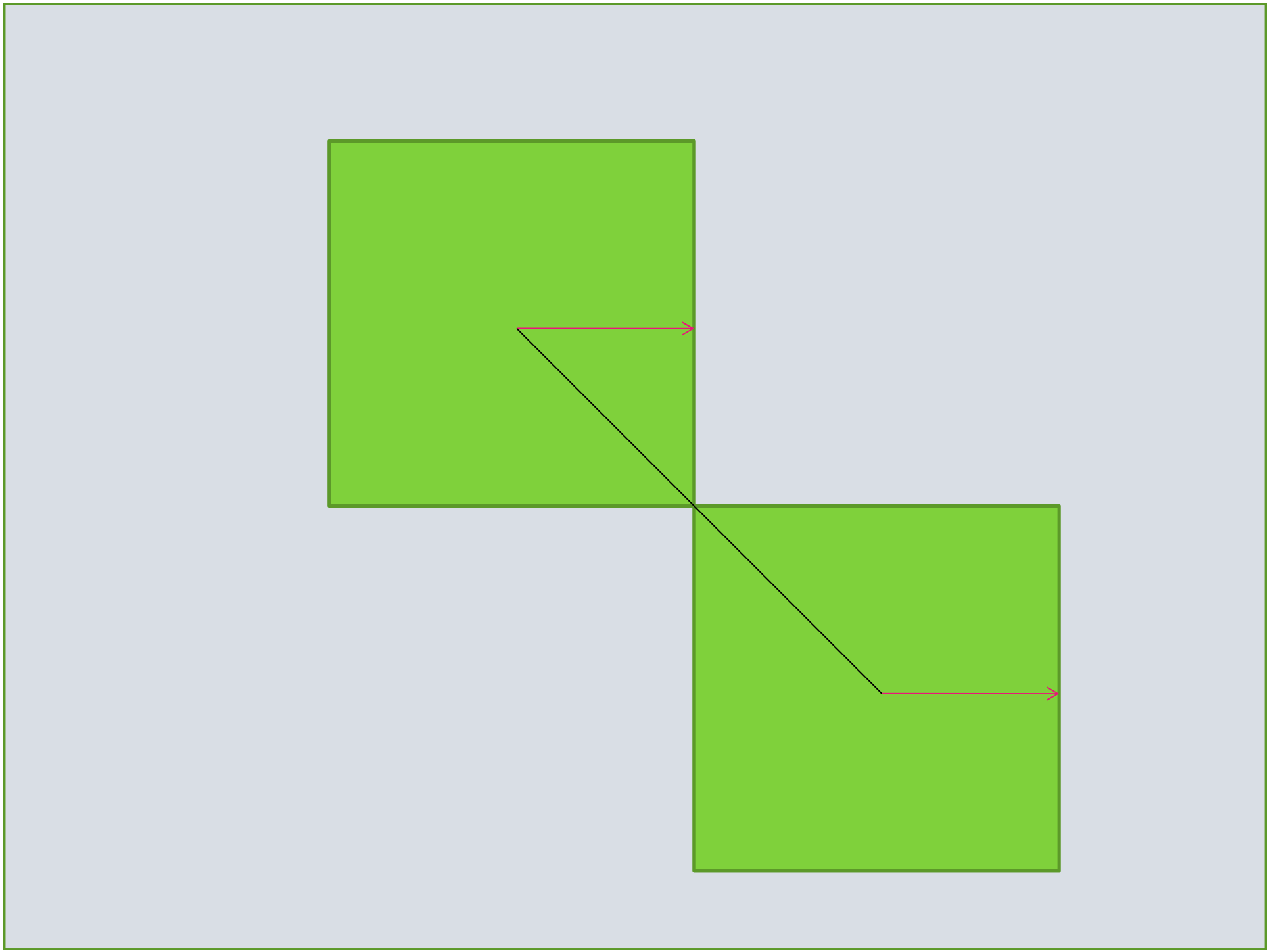
Colisão de bitmaps

- Agora para um caso hipotético. Supondo duas esferas maciças, qual é a condição suficiente para que estas estejam tangenciando uma a outra?
- Uma simplificação que pode ser feita é julgar que ambas estejam em um plano, reduzindo o problema a uma simples análise em 2D, onde as esferas se comportam como círculos de um raio ρ qualquer.
- Uma das soluções seria determinar a equação de ambos círculos e verificar se existe algum ponto pertencente a um que satisfaz a equação do outro. Nesse caso é possível afirmar que houve uma colisão.
- Parece razoável pensar desta maneira, pois normalmente o programador conhece o centro do círculo bem como o seu raio (lembre-se do comando circlefill(…);). Contudo caso o sistema sofra alguma modificação, ou a cada instante que isto aconteça, ambas as equações devem ser atualizadas e novamente verificadas as condições.
- Computacionalmente seria gasto muito tempo do processador para se efetuar essas contas e nem seria certeza que essas contas revelariam a colisão, logo ela não é muito efetiva.
- Uma hipótese mais simples seria pensar como o programador já conhece o raio de ambas e seu centro, somente seria necessário calcular a distância entre os centros e comparar com a soma dos raios. Caso essa distância seja menor ou igual, houve a colisão. Essa hipótese já é bem mais praticável e é a indicada para resolver essa questão, uma vez que cálculos complexos não são exigidos.
- Ver o algoritmo em Aula-Allegro/Aulas/Colisao Bitmaps - Ex 1.

Colisão de bitmaps

- Outro evento que ocorre com freqüência é a colisão entre quadrados ou retângulos. Dar-se-á em um jogo de carros, por exemplo. Tal motivo acontece por ser possível aproximar sem muito erro um carro visto de topo por um retângulo com uma certa rotação em relação a um sistema de referência qualquer. Como seria possível quantificar ou analisar quais são as condições suficientes para que uma colisão ocorra?

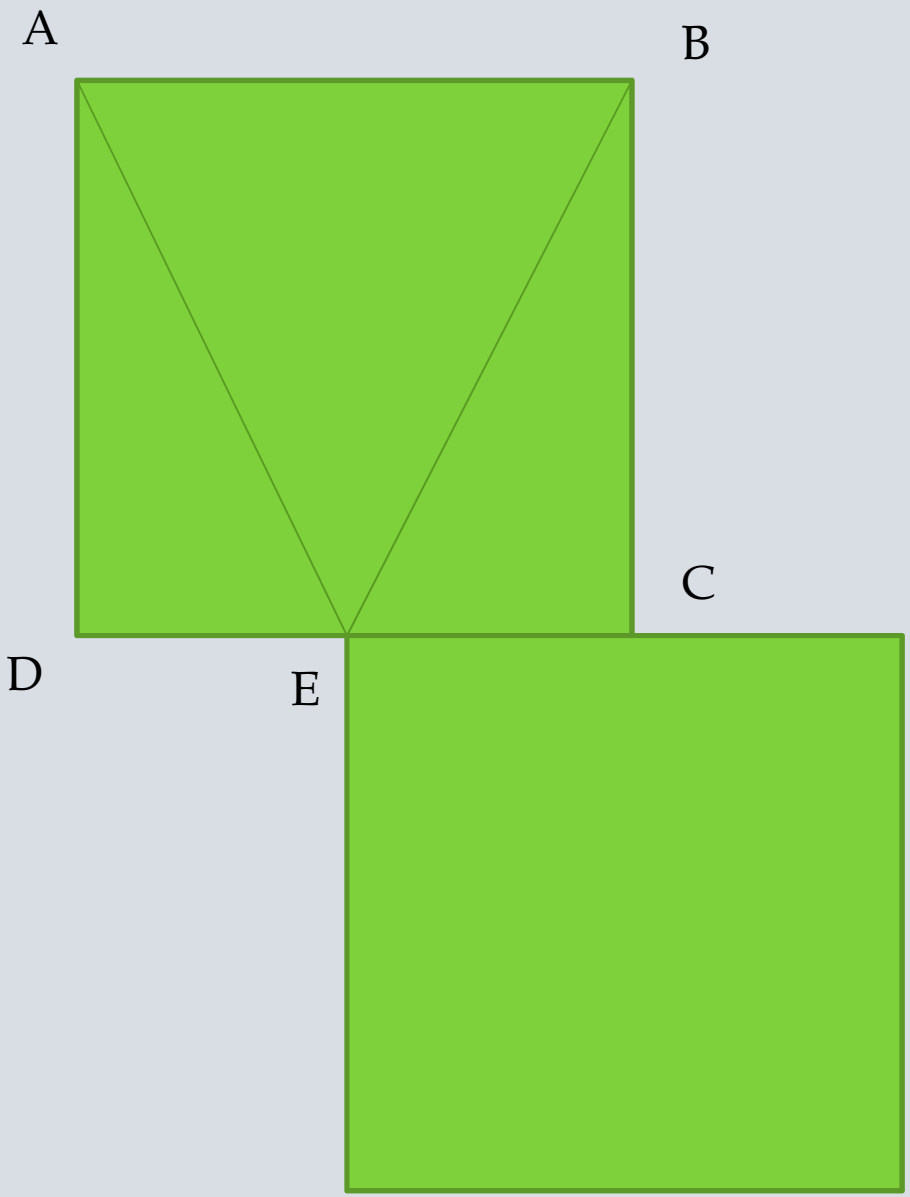
- Inicialmente supondo que ambos retângulos não possuam rotação um em relação ao outro (faces paralelas duas a duas) e serem quadrados, é possível então pensar na mesma solução encontrada para os círculos, raciocinando sobre o apótema do quadrado. Seria uma boa solução, contudo a soma das apótemas gera um valor qualquer, dependendo da posição dos quadrados, é possível que essa soma seja tal que ocorra uma superposição (colisão) de ambos quadrados.



Colisão de bitmaps

- Como é possível constatar, ocorreu uma colisão, contudo seria possível ainda movimentar um retângulo sobre o outro, pois não cumpriu a condicional. Logo esse modelo deve ser abandonado.

- Outro muito útil e que satisfaz essa condição, seria trabalhar com triângulos definidos entre os vértices de um retângulo com os vértices do segundo retângulo. Quando a soma da área dos triangulos formados por um vértice der aproximadamente igual (o programador define qual é a precisão a ser adotada), acontece a colisão.

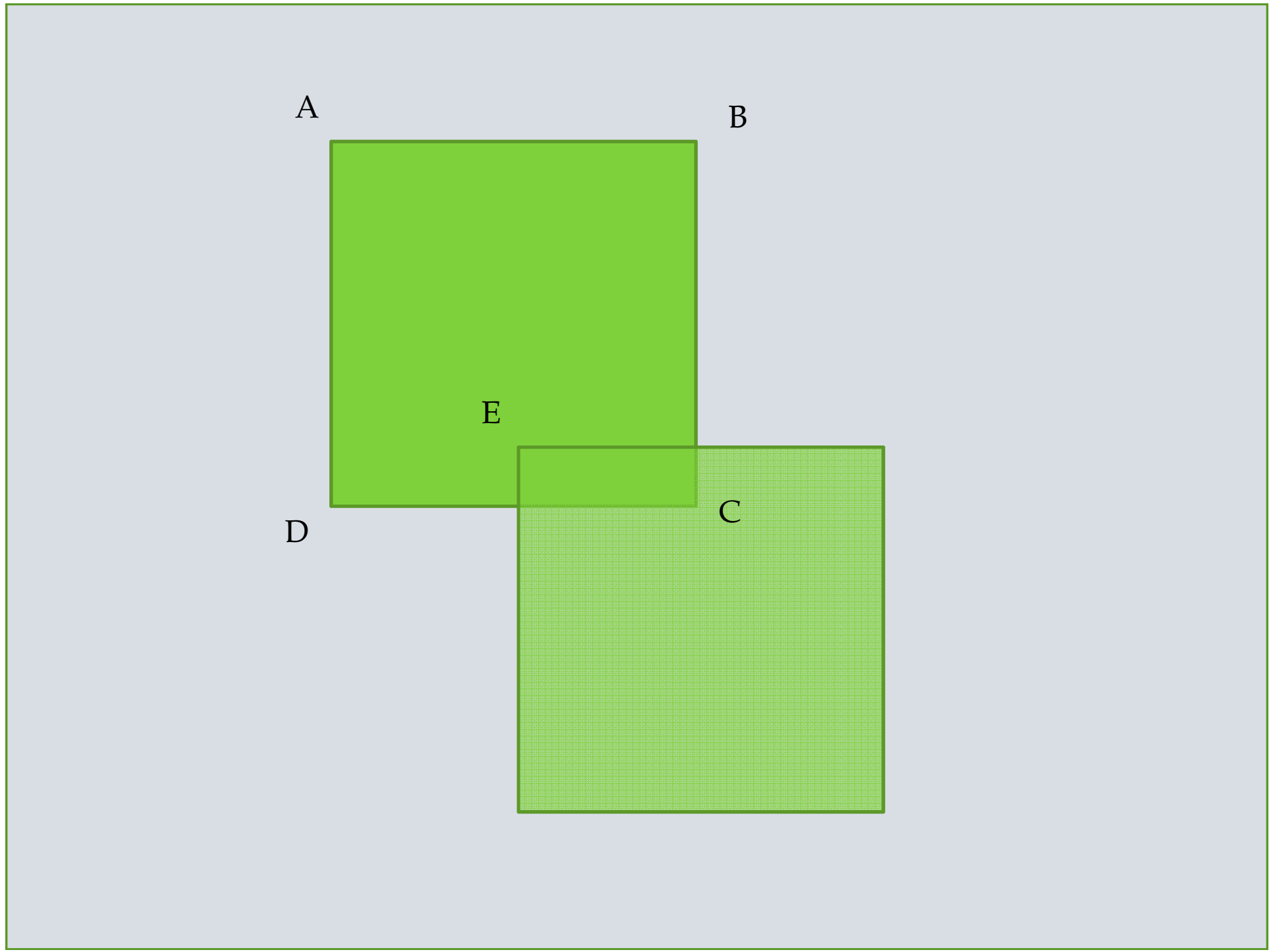

A
B
C
D
E

Colisão de bitmaps

- Caso a área formada pelos triângulos ADE, ABE e BCE tenham como valor próximo da área total do retângulo ABCD aconteceu a colisão no ponto E.

- A primeira vista, esse método soluciona a questão, pois para calcular essas dos triangulos seriam necessários três determinantes. Contudo, novamente demanda muito esforço de cálculo, sendo um método não muito praticável.

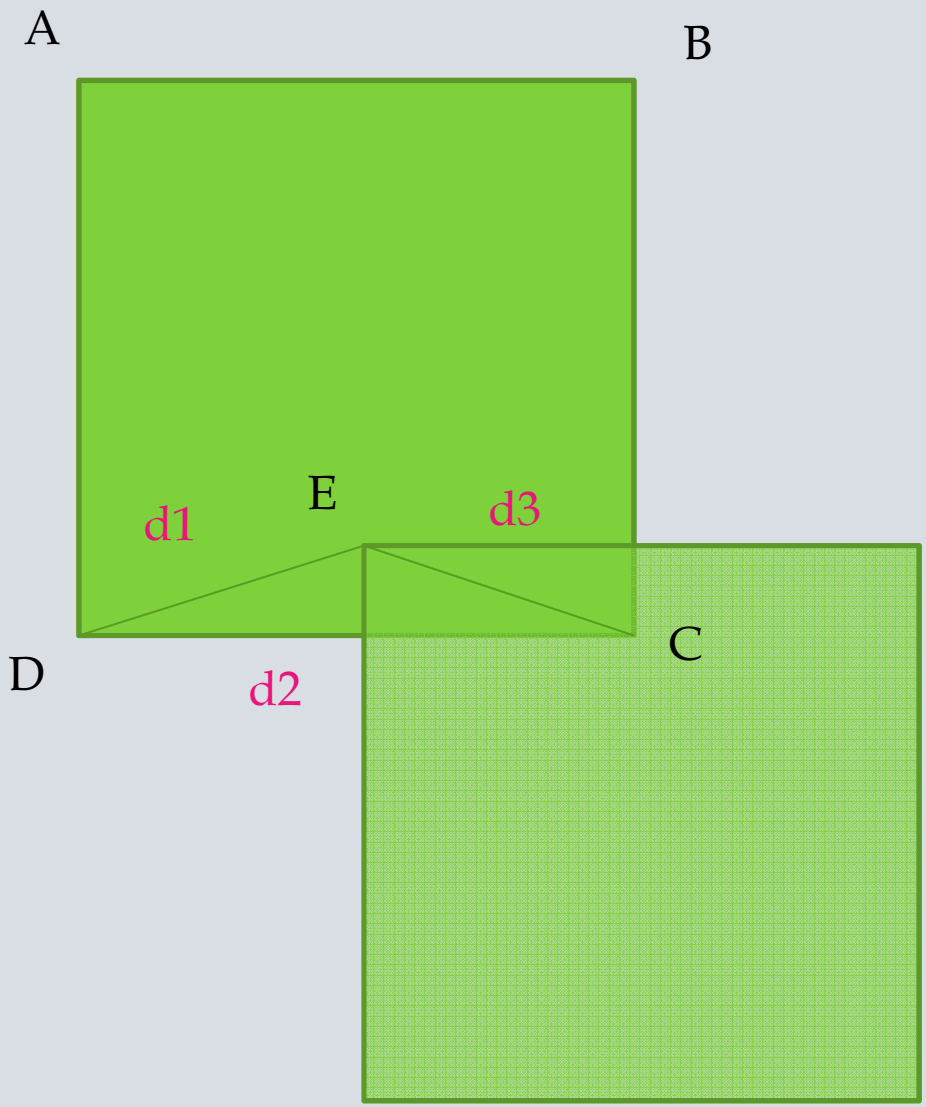
- Utilizando-se os pontos, é possível determinar equações de segmentos de retas que limitam um retângulo em questão. Para satisfazer a colisão, algum dos quatro pontos do outro retângulo deve satisfazer a condição de pertencer a reta.


A
B
E
C
D

Colisão de bitmaps

- Na imagem anterior é possível verificar que não aconteceu a condição de parada e houve a superposição. Por tal motivo esse método não satisfaz.

- Utilizando ainda a idéia de trabalhar com os pontos, se a soma dos segmentos formatos pelos pontos (perímetro de um triângulo) fechar em um dado valor, aconteceu uma colisão. Para proceder desta maneira é escolhido um ponto a ser analisado.


A
B
E
d1
d3
C
D
d2

Colisão de bitmaps

- Para o ponto D, a distância até E vale d1; a distância entre D e C vale d2; e de E para C, vale d3. Verifica-se qual das três distâncias é a maior, após isso, caso a soma das outras duas seja menor poderá ter ocorrido ou não a colisão. Essa análise é coberta pela soma das áreas dos triângulos. Uma simplificação seria que no caso que houve a colisão (paredes justapostas), a soma das duas menores distâncias possuem uma diferença em relação à maior que reside dentro de um intervalo estipulado pelo programador (obviamente que se for muito grande esse intervalo, poderá ser visível a superposição ou um relativo afastamento entre as peças quando ocorre a colisão).
- Esse procedimento requer menos do CPU, pois as contas são mais simples, apenas cálculo de distâncias e somas. Mas será necessário uma nova análise a cada modificação nova que venha a ser feita em relação ao estado anterior. Caso seja constatada a colisão, o estado anterior é restaurado. Essa aproximação é relativamente boa e pode ser extendida para retângulos.
- Ver o algoritmo em Aula-Allegro2008/Aula-Allegro/Aula/Colisao Bitmaps - Ex 2.

Colisão de bitmaps

- Essa teoria se aplica bem para retângulos que não possuem rotação relativa entre si. Agora quando existe, como será avaliada a colisão?
- Definir uma equação para cada segmento de reta pode ser uma solução, contudo demanda novamente de muito tempo de cálculo (definir as 8 retas, depois verificar as possíveis combinações desses pontos em cada uma das retas para se certificar que houve ou não a colisão).
- É possível mostrar que a solução anterior satisfaz também a condição para retângulos rotacionados entre si, é interessante o programador tentar verificar essa solução e analisar qual deve ser o erro adotado para que tenha um efeito esperado.
- Ampliando o raciocínio, utilizando-se dessa verificação é possível criar um sistema de colisão para qualquer figura geométrica a partir das somas de distâncias (para se aplicar esse conceito, a figura de preferência não deve ser muito complexa, pois demanda muito tempo de cálculo, logo uma nova solução deve ser feita para resolver esses problemas).
- Ver algoritmo em Aula-Allegro2008/Aula-Allegro/Aula /Protótipo de um jogo.

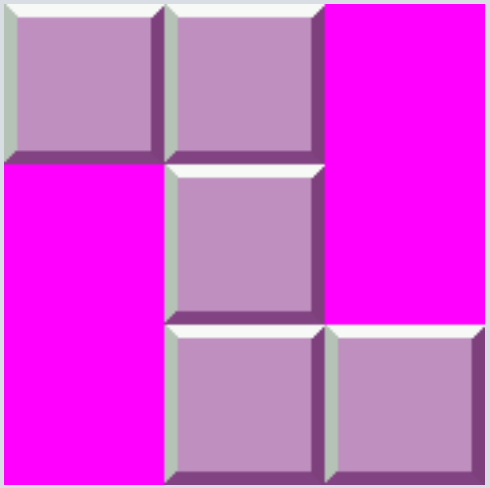
TRANSPARÊNCIA DE BITMAPS

Transparência de bitmaps

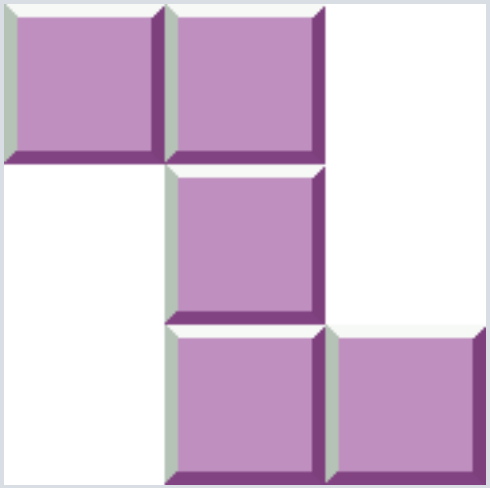
- A transparência em um bitmap serve para isolar a imagem desejada a ser exibida de seu fundo. Um especial cuidado deve ser tomado na hora de realizar tal procedimento, pois qualquer variação pode não gerar o efeito visado (a não correta isolação da imagem).

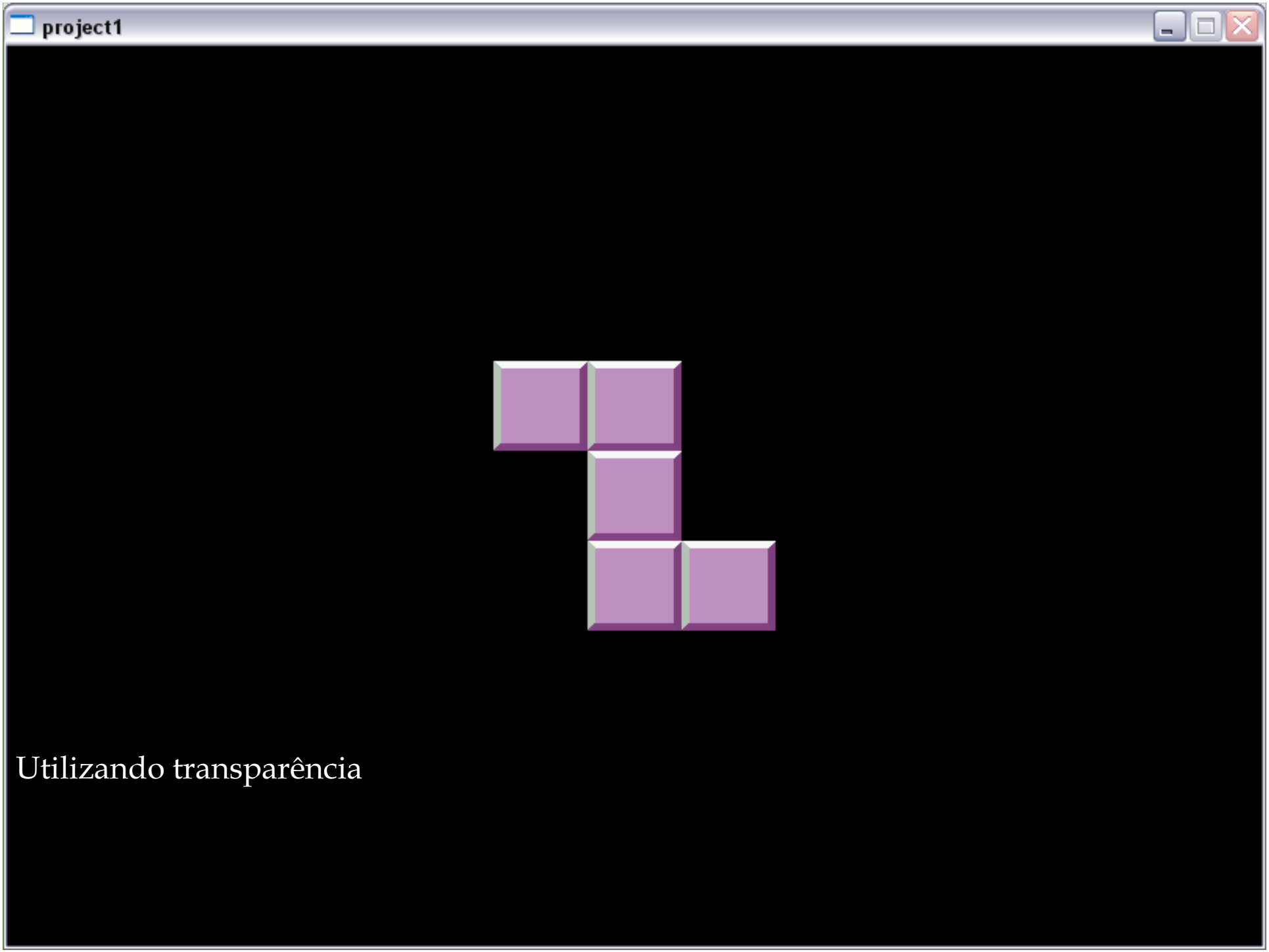
- Para ser feito isso, o fundo da imagem deve ter a cor (em RGB [255, 0, 255]) magenta. Isso não se aplica só ao fundo (seria uma aplicação), pode ser usado em qualquer lugar do bitmap onde se deseja uma transparência.

Peça com transparência no fundo

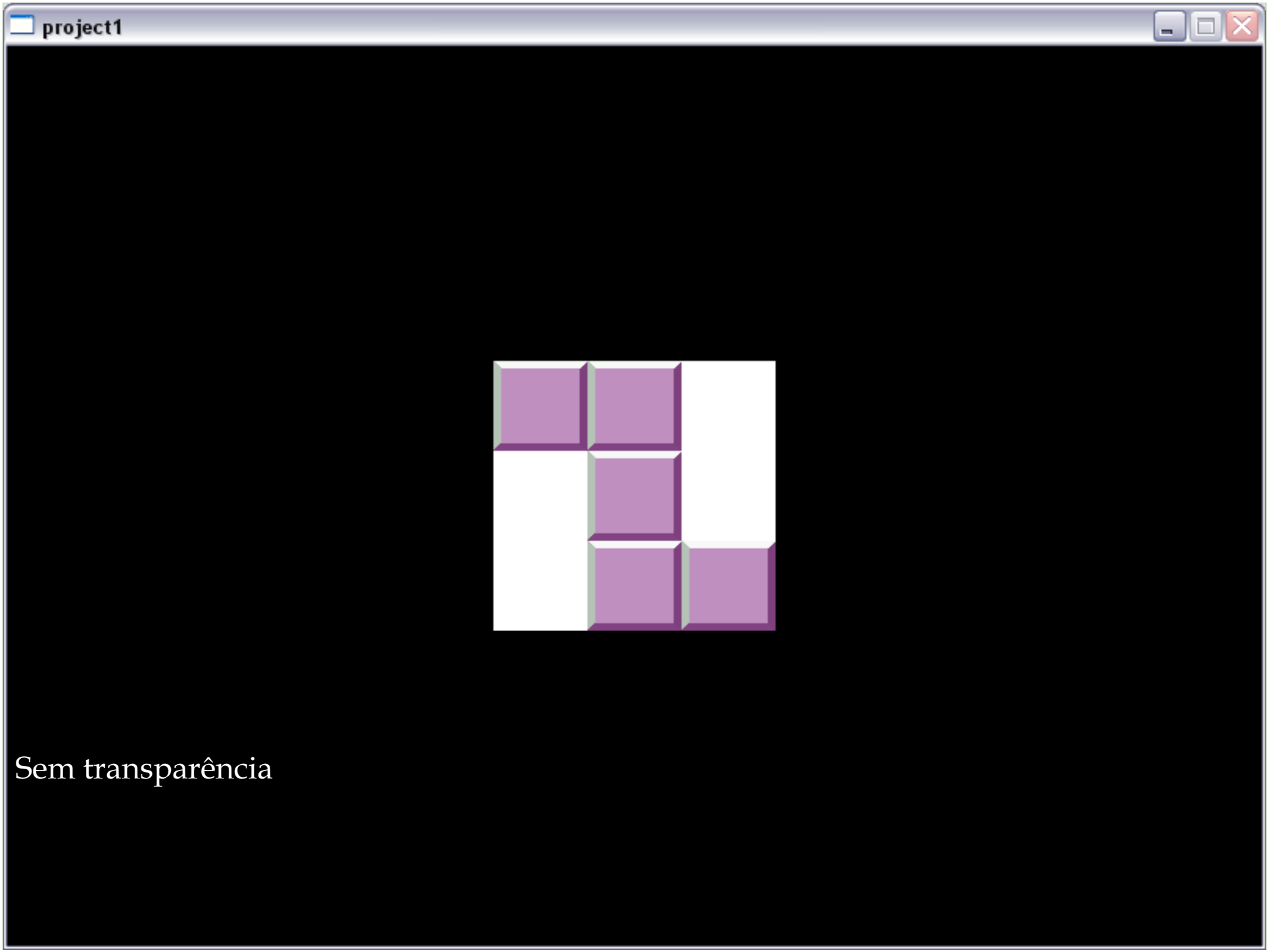


Peça sem transparência no fundo





Utilizando transparência



Sem transparência

OUTRAS TÉCNICAS

Outras técnicas

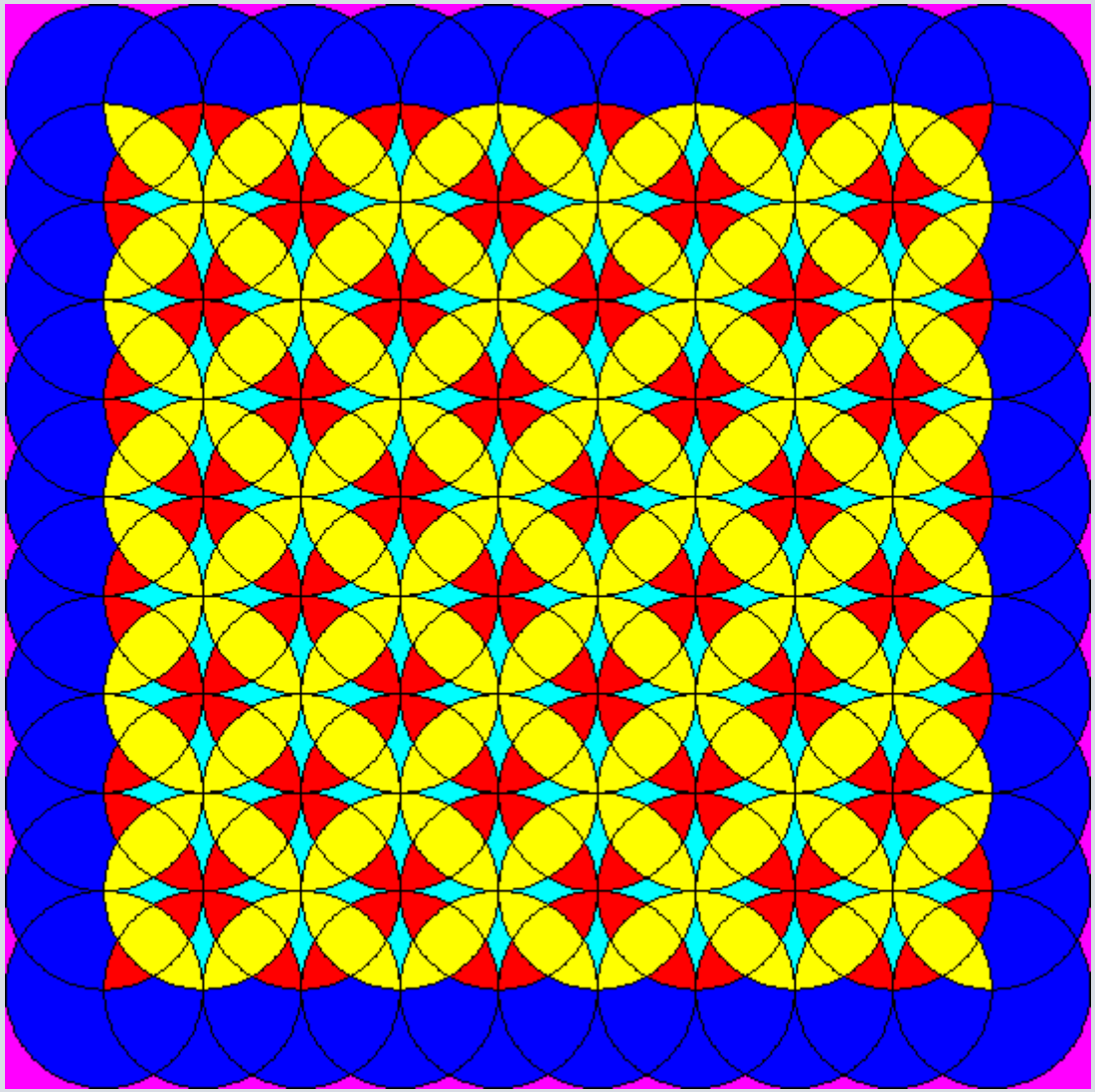
- Quando se trabalha com bitmaps, alguns cuidados devem ser tomados. No geral, boa parte desses bitmaps estão no formato .bmp, o que deixa esses arquivos um tanto quanto "pesados". Em casos particulares é possível reduzir esse tamanho, utilizando-se da simetria da imagem. Tal abordagem será tratada mais adiante.
- A exibição de um bitmap na tela consome tempo, dependendo da quantidade de imagens a serem desenhadas, essa operação é demorada e/ou normalmente é possível que a imagem "pisque" por um intervalo de tempo. Tal acontecimento é visto sem muita dificuldade, diminuindo o grau da qualidade do aplicativo.
- Uma das técnicas possíveis para se corrigir isso é a utilização de um buffer (local de armazenamento temporário para uma dada aplicação). Ao invés de se colocar o parâmetro screen nos métodos de bitmpas, utilizar um bitmap intermediário ajuda nessa operação. Outra possibilidade é regular o tempo do loop antes de atualizar a imagem (controlado por rest(...)). Ou então (também é recomendado), redesenhar a imagem APENAS quando esta se faz necessária, por exemplo quando a imagem é transladada ou rotacionada, do contrário não se atualiza a imagem.
- Ver o algoritmo em Aula-Allegro/Aula/Rotacao e Translacao - Ex 1

```
#include <allegro.h>
#include <stdlib.h>

int main()
{
    allegro_init();

    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);

    BITMAP *Imagem, *buffer;

    buffer = create_bitmap(SCREEN_W, SCREEN_H);
    clear(buffer);

    Imagem  = load_bitmap("Imagem.bmp", NULL);

    draw_sprite(buffer, Imagem, 0, 0);
    allegro_message("Sem exibicao de imagem");

    draw_sprite(screen, buffer, 0, 0);
    allegro_message("Imagem na tela !!");

    destroy_bitmap(Imagem);
    destroy_bitmap(buffer);

    allegro_exit();

    return 0;
}
END_OF_MAIN();
```

Outras técnicas

- A utilização de mosaicos é indicada sempre onde exista simetria na imagem. A seguir será exibida uma imagem que apresenta alguns tipos de simetria, servindo como um exemplo de como proceder diante desse fato. Vale ressaltar que no fundo é utilizado o magenta, caso o programador quisesse colocar tal imagem dentro do aplicativo.

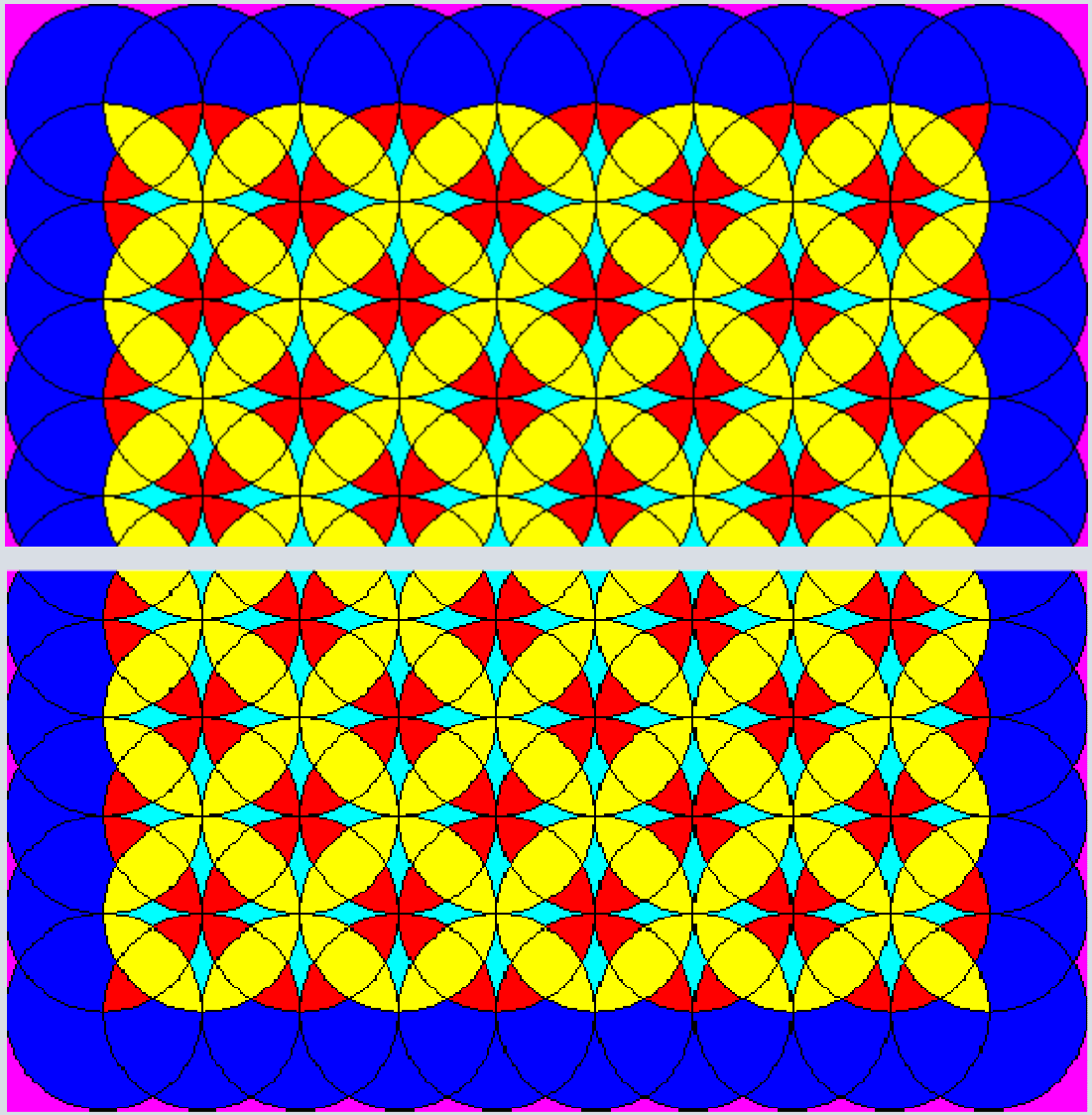
- É interessante determinar quais são os tipos de simetria possíveis a serem utilizadas. A imagem a seguir tem 870 kb.



Outras técnicas

- A mais elementar suposição a ser feita é dividir a figura ao meio. Pois a princípio iria diminuir o tamanho pela metade também.

- A idéia é válida, realmente se aplica a casos como esse. Contudo além de efetuar um corte na imagem, a justaposição das duas metades (da metade selecionada como monômero) deve reconstruir a imagem com perfeição. É possível também que seja necessária efetuar uma rotação para o correto encaixe de ambas.



Outras técnicas

- A imagem do topo foi escolhida como monômero de repetição. A inferior, é uma cópia da primeira rotacionada de 180º. Fica fácil notar que se as duas metades forem superpostas, restaura a imagem original, servindo assim como um mosaico.

- Esse monômero tem o tamanho de 436 kb.

Outras técnicas

- Será que não existiria outras possibilidades para serem exploradas?
- Uma delas seria quebrar a imagem em outras duas.
- A primeira a ser exibida será chamada de Imagem 1, e a segunda, de Imagem 2.

Imagem 1

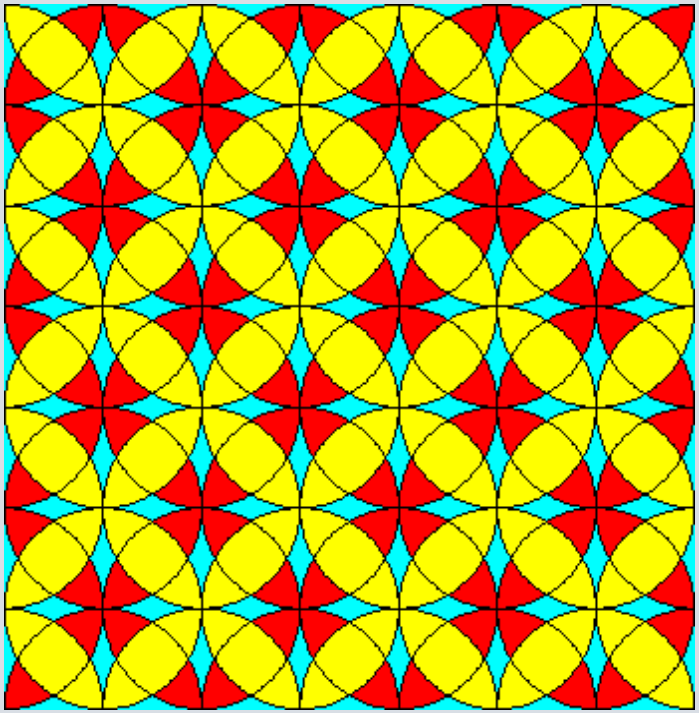
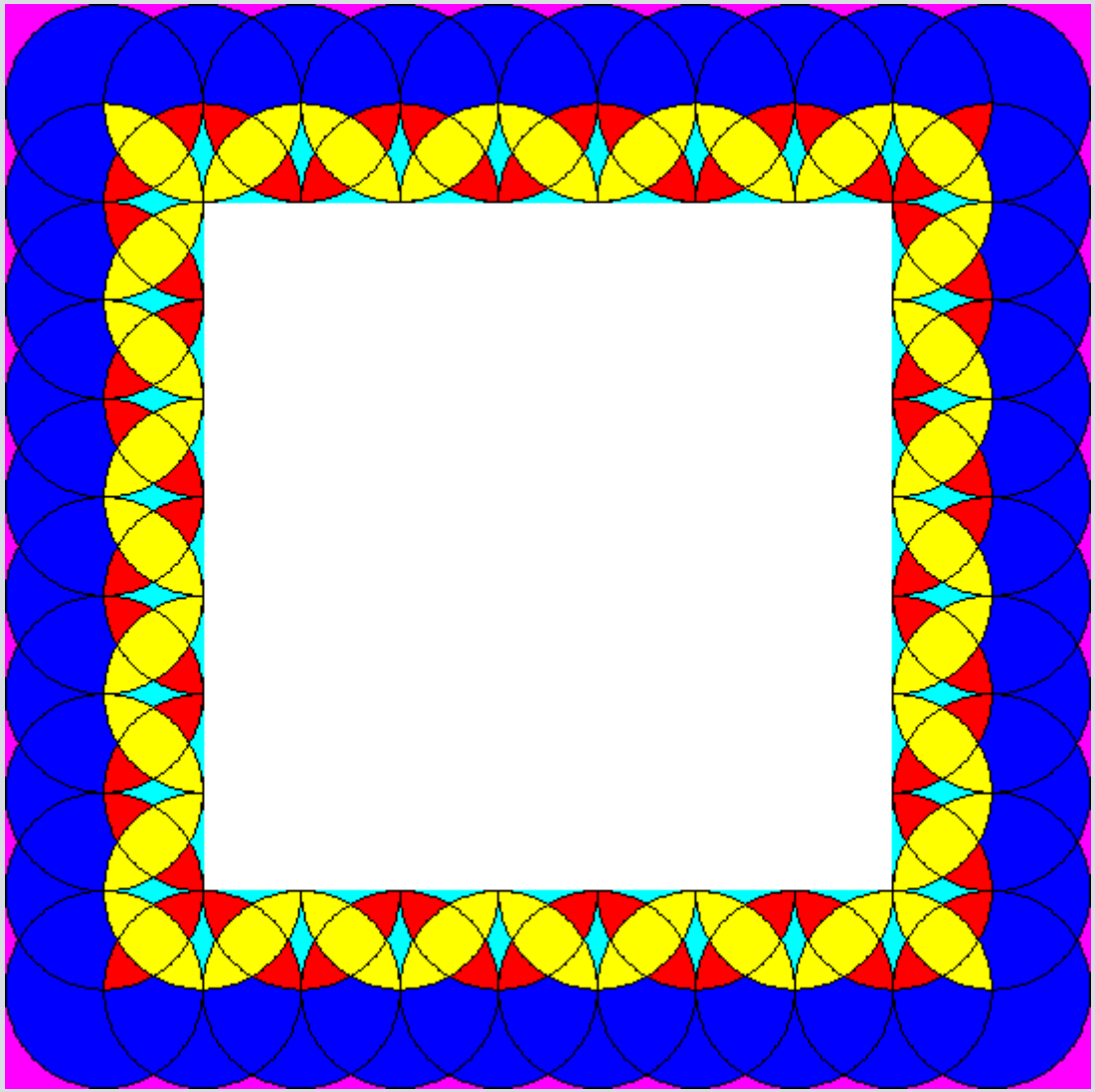


Imagem 2



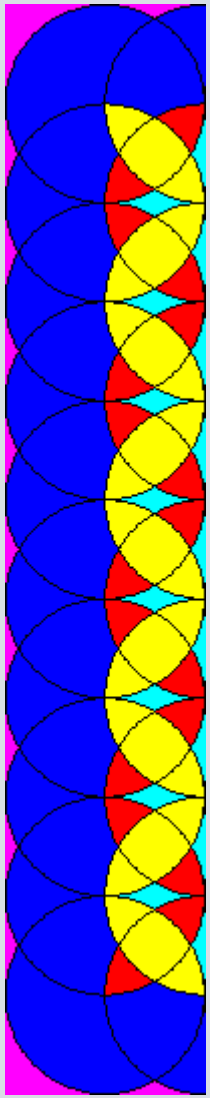
Outras técnicas

- Para a Imagem 1 existem outras possibilidades de serem criados monômeros. É recomendado fazer um levantamento desses, treinando assim a capacidade de reconhecer simetrias em figuras. Além de facilitar o trabalho em alguns casos, economiza tempo de processamento por gerenciamento de memória (minimiza o tempo de acesso a memória e a recuperação da imagem por ter um tamanho menor).
- A melhor possibilidade para um monômero será mostrado a seguir. Com este é possível recriar a Imagem 1 com justaposição desses monômeros. Para isso, alguns devem ser rotacionados.
- Monômero a ser utilizado: 
- Tamanho do monômero: 3,91 kb

Outras técnicas

- Falta tratar do monômero da Imagem 2. As opções de construções de monômeros desta imagem sejam mais restritas. Uma primeira aproximação, seria interessante repartir a Imagem 2 em quatro partes, sendo assim possível reconstituir o todo usando apenas uma dessas quatro partes escolhida apropriadamente.

Monômero da Imagem 2

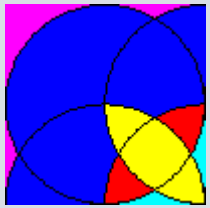


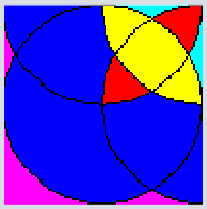
Outras técnicas

- Contudo ainda é possível verificar que existem simetrias dentro desse último. Será necessário quebrar novamente a imagem em frações menores.

- Um cuidado a ser tomado, os extremos superior e inferior da imagem são DIFERENTES, logo deve ser criado um monômero para ambos. Já na região central, é possível determinar uma unidade de repetição sem muitos problemas.

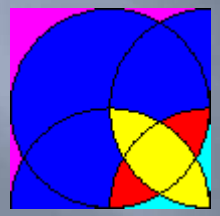
Monômero da Imagem 2



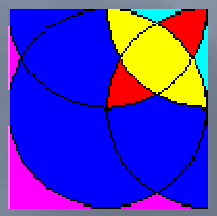


Outras técnicas

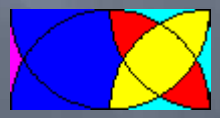
- Novamente utilizando-se de algumas rotações com essas unidades básicas, a Imagem 2 é perfeitamente reconstruída.
- Tamanho dos monômeros:



29,3 kb



29,3 kb



14,7 kb

Outras técnicas

- Resumindo: O tamanho total ocupado por todos os 4 monômeros utilizados foi de aproximadamente 77,21 kb. Ou seja, corresponde a 8,87% do tamanho utilizado originalmente. O ganho em economia de espaço e tempo de processamento é muito alto nesse caso, sendo recomendado assim a utilização do mosaico. Contudo nem sempre é possível ou trivial determinar simetria em imagens. Vem do tato do projetista ou do programador perceber.

- Outro ponto importante que as simetrias não existem apenas em imagens, e sim em problemas. Alguns problemas podem guardar uma relação de simililaridade entre si. Se for possível determinar qual é a relação existente entre os dois, a solução para ambos é mais fácil. Um emprego dessa técnica (tanto de imagem, quanto de problema) esta tratado no jogo Tetris Another Remake. O indicado não é simplesmente copiar pedaços de código utilizados. Além de ser abominável essa prática para com o programador que o fez, quem copia não sabe o raciocínio que tem por trás, muito menos dá o valor pelo trabalho alheio. Se realmente for necessário retirar fragmentos de código para uma dada aplicação, quem o faz deve ter noção como foi feito e para o que serve.

CONSIDERAÇÕES FINAIS

Considerações finais

- Ao final deste é possível ter uma noção dos recursos e como utilizar eles da melhor forma possível. Nem sempre é recomendado utilizar todas as técnicas, pois se o problema a ser resolvido é simples não é necessário criar um algoritmo muito elaborado.

- Outros tópicos e emprego de conceitos estão em Aula-Allegro/Testes (tratam de protótipos de métodos específicos do jogo Panzerhaubitze I e II, criados para a disciplina de Fundamentos de Programação I e II). Um jogo simples utilizando mouse, fontes customizadas e o mínimo de recursos é o Sudoku (Aula-Allegro\Sudoku – DEV), o mesmo é o recomendado a este jogo o que foi dito para o Tetris Another Remake.

- Jogos parecidos com estes, ou que utilizem partes inalteradas são completamente desaconselháveis, pois a intenção de ser cobrado um jogo nas disciplinas é fazer com que o programador aprenda a pensar e a programar. De nada adianta saber programar e não ser capaz de desenvolver, por tal motivo é interessante que o grupo trabalhe em algo novo, no máximo BASEADO em algo pronto.