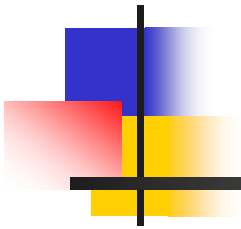


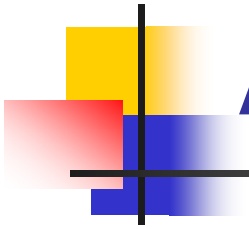
Programação Gráfica – Parte 1

Versão em C – 2006 – PUCPR – Tutoria de Jogos – 1º Ano

Paulo V. W. Radtke

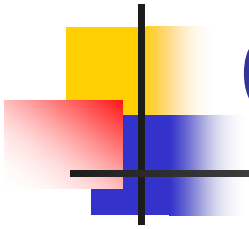
pvwradtke@gmail.com

http://www.ppgia.pucpr.br/~radtke/jogos/



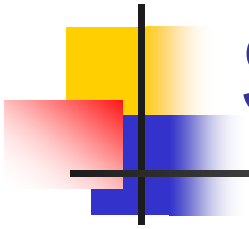
AVISO IMPORTANTE!!

- Esta versão é dedicada exclusivamente para os cursos de **Ciência da Computação** e **Engenharia da Computação**.
- Para a versão de **Sistemas de Informação**, utilizando ***Java***, pegue o arquivo correspondente e participe da aula no horário adequado.



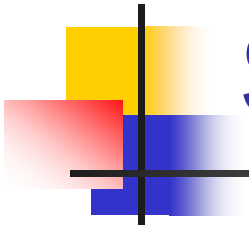
Conteúdo

- Instalando as bibliotecas SDL no DevC++.
- Introdução à biblioteca **Chien2dLite**.
- Ciclo de vida básico de um programa.
- Primeiro exemplo: carregando uma tela usando SDL.
- Carregando *sprites* e mostrando-os na tela.



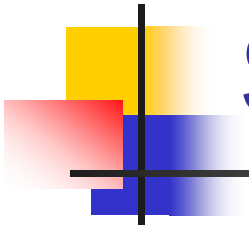
SDL

- Serão utilizadas 3 bibliotecas da SDL:
  - SDL: biblioteca núcleo. Fornece funcionalidades básicas.
  - SDL_Image: biblioteca para a manipulação de imagens 2D.
  - SDL_Mixer: biblioteca que extende o suporte de áudio da SDL. Suporta vários formatos de músicas, como MIDI, MP3, Ogg, tracked e diversos de áudio.



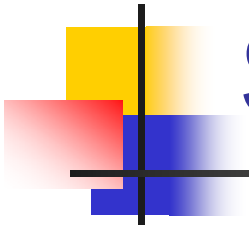
SDL

- Para facilitar a instalação, foi criado um pacote, disponibilizado na página da tutoria: **pacote-dev-cpp.zip**.
- Este arquivo deve ser descompactado no diretório aonde se encontra instalado o Dev-C++.
- Ex: Se ele se encontra instalado em **C:\Dev-Cpp**, o mesmo deve ser descompactado em **C:**, a raiz do drive.



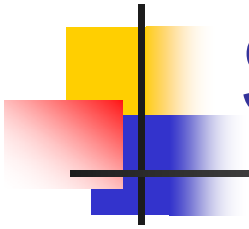
SDL

- Além disto, é possível fazer o *download* dos pacotes diretamente do site da SDL: http://www.libsdl.org
- Como toda biblioteca compartilhada, a DLL possui um grupo de arquivos que devem ser colocados no diretório **System** do Windows ou distribuídos junto com a aplicação.



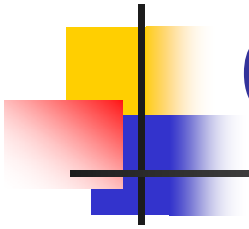
SDL

- Como não temos permissão de escrita no diretório System, utilizaremos a segunda opção.
- Apesar de aparentemente mais desvantajosa, esta abordagem é mais prática do ponto de vista de distribuição.



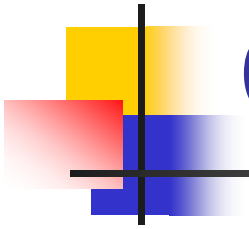
SDL

- Arquivos DLL relevantes:
  - SDL.dll
  - SDL_image.dll
  - SDL_mixer.dll
  - zlib1.dll
  - jpeg.dll
  - libpng13.dll



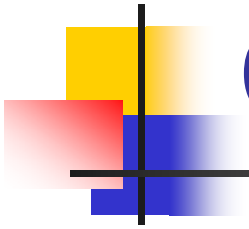
Chien2DLite

- Biblioteca básica 2D para *sprites* e fontes *bitmap*.
- Se preocupa com o gerenciamento das primitivas escolhidas
- Baseada em uma **classe** (C++).
- A classe é um *singleton*, e a instância do vídeo pode ser acessada a partir de qualquer ponto do programa.



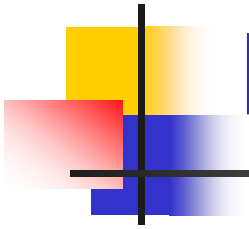
Chien2DLite

- Limitações:
  - Adaptada a partir de uma versão OpenGL, a biblioteca somente carrega imagens quadradas, com tamanhos potência de 2. Ex: 8x8, 16x16, 32x32, 64x64, 128x128, 256x256, 512x512, 1024x1024, etc.
  - A biblioteca somente desenha blocos (*azulejos*, do inglês *tiles*), não dando suporte a imagens.



Chien2DLite

- A primeira versão básica não possui primitivas 2D tradicionais.
- Será implementado para as próximas aulas o suporte a desenho de pontos, retas e retângulos.

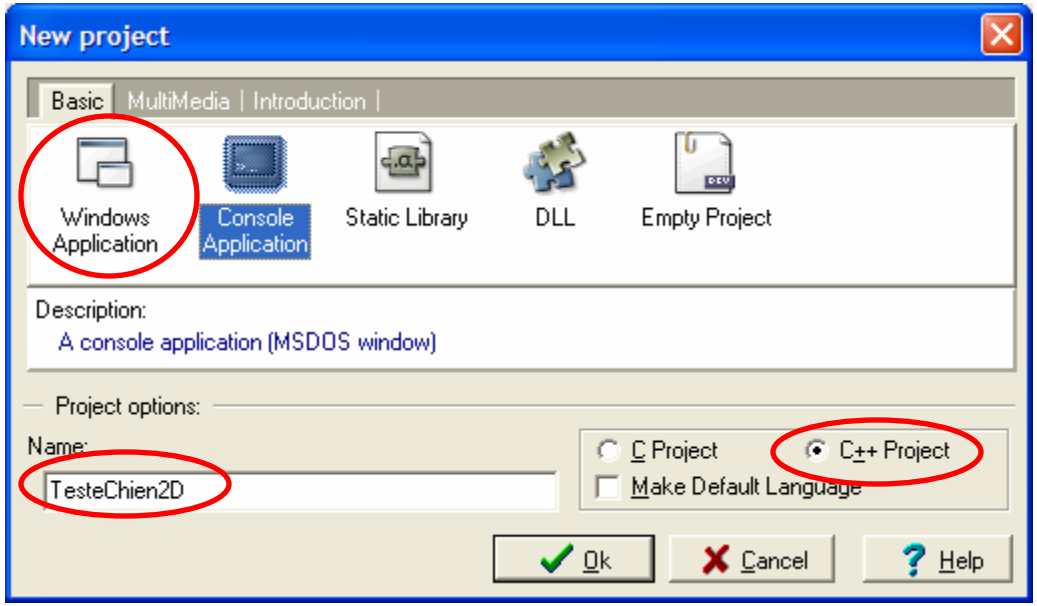


Projeto no Dev-C++

- Para fazermos qualquer coisa com a biblioteca, devemos criar um projeto no Dev-C++.
- O primeiro passo é criar um diretório e extrair nele os arquivos da **Chien2DLite**.
- O arquivo **exemplo01-c-imagem.zip** será utilizado neste exemplo (o arquivo contém a biblioteca *Chien2DLite*).

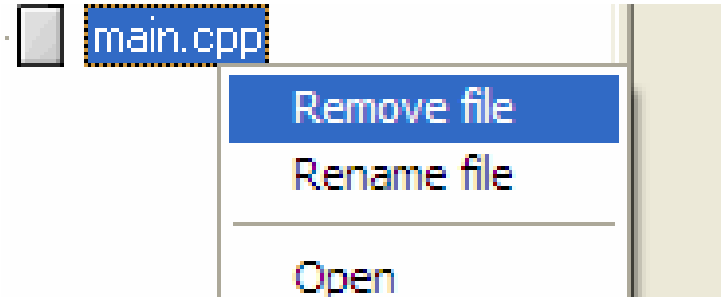
Projeto no Dev-C++

- Crie um projeto novo no Dev-C++ e salve-o no diretório escolhido:



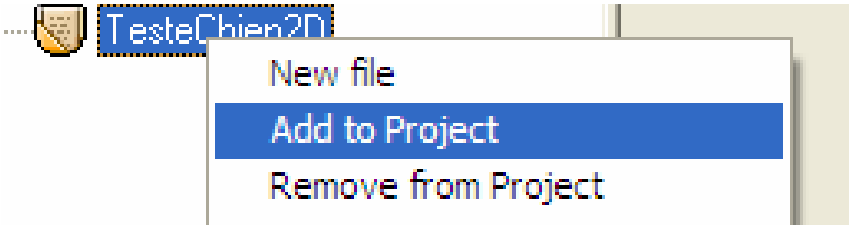
Projeto no Dev-C++

- O projeto por default vem com um arquivo **main.cpp**, remova-o.
- Para isto, pressione o botão direito do mouse sobre o nome do arquivo e selecione a opção “Remove File”.



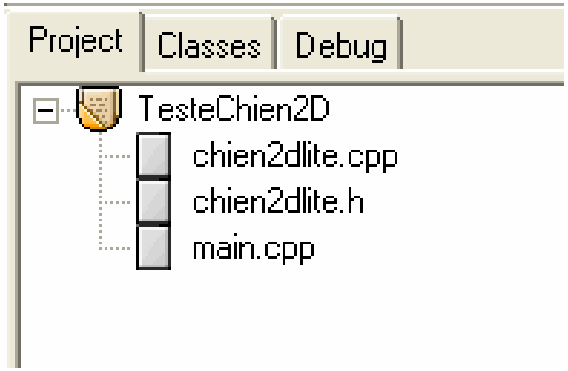
Projeto no Dev-C++

- Em seguida, adicione os arquivos do exemplo clicando com o botão direito do mouse no nome do projeto:



Projeto no Dev-C++

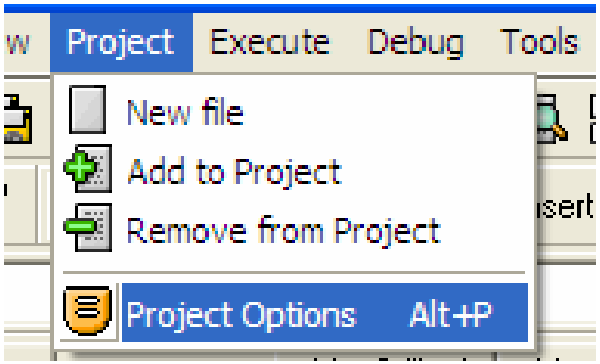
- Uma vez inclusos os arquivos, temos a seguinte listagem:



- Obs: os arquivos da Chien2DMappy, responsável pelos mapas estão inclusos mas não serão utilizados neste exemplo. A sua inclusão no projeto não faz diferença.

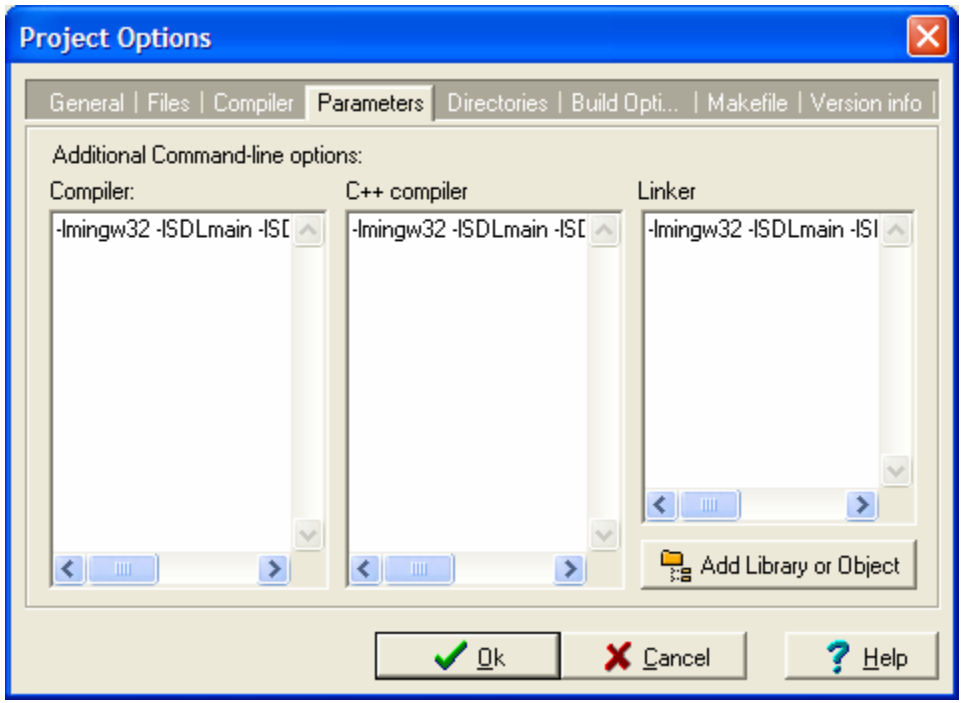
Projeto no Dev-C++

- Se compilarmos o projeto como está, o mesmo não funcionará.
- É preciso indicar ao compilador quais bibliotecas ligar ao executável com o *linker* em *Project -> Project Options*.



Projeto no Dev-C++

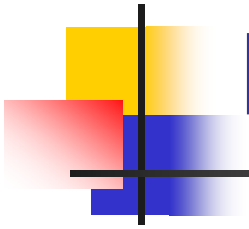
- Selecione a aba *Parameters*:



Projeto no Dev-C++

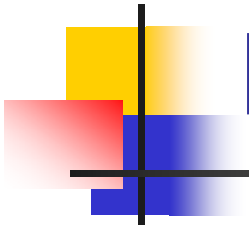
- Coloque os seguintes parâmetros no compilador C, C++ e no Linker:

-lmingw32 -lSDLmain -lSDL -lSDL_image



Projeto no Dev-C++

- Uma vez colocados os parâmetros, o programa compilará e o executável será gerado.
- Ao executar o mesmo funciona.
- Só que por enquanto, o programa só exibe uma tela preta.
- Para encerrar o programa, pressione ESC ou feche a janela.



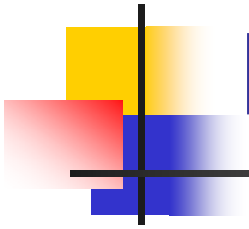
Explicação do Código Fonte

- Vamos nos concentrar na função **main**, ignorando (por enquanto), a função *entrada*, responsável pelo polling do teclado.
- A funcionalidade básica da biblioteca é simples, apenas 6 funções resolvem a parte de inicialização, encerramento e redesenho de tela.

Explicação do Código Fonte

```
Chien2DLite *video =
   Chien2DLite::pegaInstancia();
```

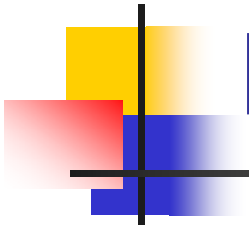
- Aqui criamos uma referência ao objeto da Chien2DLite (um ponteiro).
- O objeto deve ser recuperado SEMPRE através deste método estático.
- Este método é acessível de qualquer ponto do programa.



Explicação do Código Fonte

```
video->inicia(640, 480, Chien2DLite::janela,
   true, "Exemplo 01 - Imagem");
```

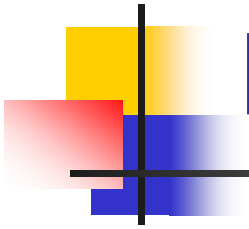
- Este método cria a janela em si.
- Os 5 parâmetros são descritos a seguir, na assinatura do método.



Explicação do Código Fonte

video->inicia(**largura**, **altura**, modo, retraço, titulo);

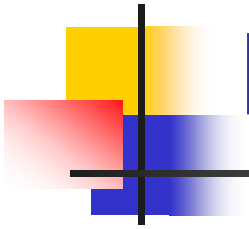
- Largura e altura indicam a resolução horizontal e vertical da tela.
- Em modo janela, podem ser qualquer valor. Em tela cheia, deve ser uma resolução suportada pelo monitor.
- Garantido em Windows: 640x480 e 800x600.



Explicação do Código Fonte

video->inicia(largura, altura, **modo**, retraço, titulo);

- Modo indica se a tela vai ser em modo janela ou tela cheia.
- Valores possíveis (valores constantes em Chien2DLite.h):
  - Chien2DLite::janela
  - Chien2DLite::telaCheia



Explicação do Código Fonte

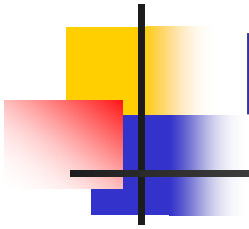
video->inicia(largura, altura, modo, **retraço**, titulo);

- Valor booleano indicando se a aplicação deve ou não esperar o retraço.
- Em modo janela ele é ignorado, já em tela cheia ele é implementado em conjunto com *double buffer*.
- Valores possíveis: **true** e **false**.

Explicação do Código Fonte

video->inicia(largura, altura, modo, retraço, **titulo**);

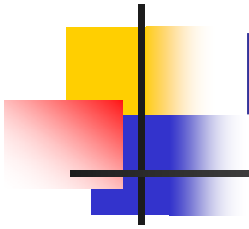
- String indicando o nome da aplicação.
- Qualquer valor string válido (entre aspas) é aceito.



Explicação do Código Fonte

video->limpaTela();

- O método **limpaTela**, como o nome sugere, limpa a tela com a cor preta.
- Útil entre redesenho de *frames* nos jogos.



Explicação do Código Fonte

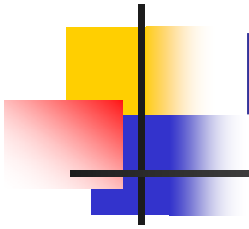
- video->sincroniza();

- O método **sincroniza** faz a troca de *buffers* e efetiva o redesenho da tela.

Explicação do Código Fonte

video->encerra();

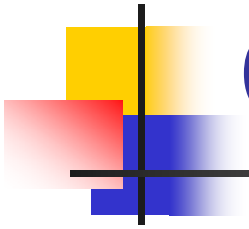
- A função encerra fecha a janela e remove os recursos alocados **PELA Chien2DLite**.
- Recursos alocados com funções SDL devem ser removidos manualmente.



Explicação do Código Fonte

Chien2D::removeInstancia();

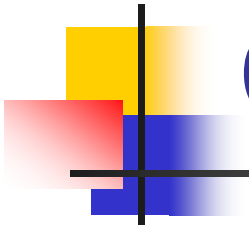
- Finalmente, a função removeInstancia remove a instancia da classe da memória.
- Como boa prática de pogramação, é interessante chamar essa função no final dos seus programas.



Carregando uma Imagem

- Imagens na SDL são do tipo SDL_Surface.
- Este tipo é uma estrutura que é alocada quando a imagem é criada/alocada.
- Para usarmos uma imagem do disco, devemos criar um ponteiro para uma surface:

SDL_Surface *imagem;



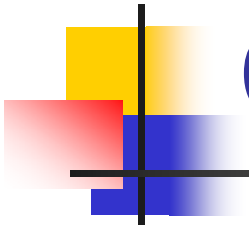
Carregando uma Imagem

- Para carregar imagens de diversos formatos do disco, devemos utilizar a SDL image.
- O include desta biblioteca é o *SDL_image.h*.
- Ele já é importado através do *chien2dlite.h*. Normalmente, aplicações SDL devem incluir este arquivo.

Carregando uma Imagem

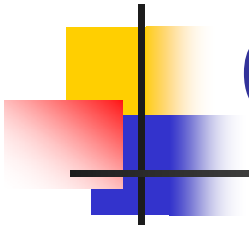
- A função que carrega uma imagem é a IMG_Load.
- Ela carrega automaticamente diversos tipos de arquivos (BMP, PNG, etc).
- A função retorna uma SDL_Surface da imagem indicada por uma string:

```
SDL_Surface *imagem =
  IMG_Load("imagem640480.png");
```



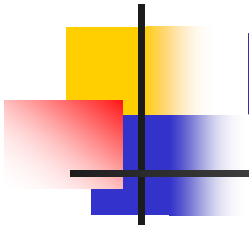
Carregando uma Imagem

- Ao carregar a imagem pode ocorrer um erro.
- Caso a imagem não exista ou não haja mais espaço em memória, a surface retornada é nula (NULL, 0).
- Em geral, é interessante testar esta condição para não usar um ponteiro nulo.



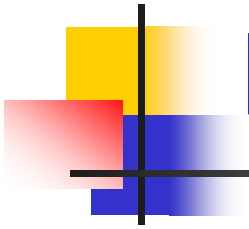
Carregando uma Imagem

- Pesquisa adicional na documentação da SDL:
  - Normalmente, surfaces criadas de diferentes fontes não tem o mesmo formato que o display de vídeo.
  - Para acelerar o processo de redesenho, devemos converter as imagens para o formato do display.
  - Pesquise na documentação da SDL como fazer isto.



Desenhando a Imagem

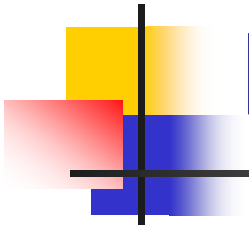
- Para desenharmos diretamente na tela enquanto usamos a Chien2DLite, precisamos pegar a surface do display do sistema.
- Para isto, usamos a função:
- **SDL_GetVideoSurface**();
- A função retorna o ponteiro da surface associada ao display do sistema.



Desenhando a Imagem

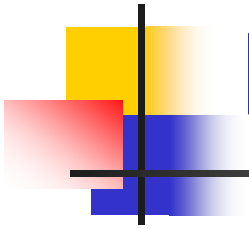
- Uso típico:

```
SDL_Surface *display =
  SDL_GetVideoSurface();
```



Desenhando a Imagem

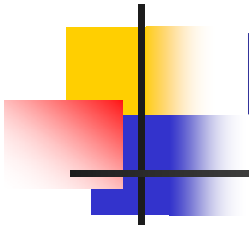
- Uma vez com a imagem e o display de vídeo, utilizamos a função SDL_BlitSurface.
- A função copia parte de uma surface para outra.
- É uma cópia rápida de pontos.



Desenhando a Imagem

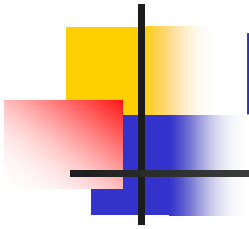
- Uso típico:

- **SDL_BlitSurface**(*imagem*, 0, *display*, 0);
- Copia a surface imagem (inteira) para a surface display (inteira), a partir das coordenadas (0,0).



Desenhando a Imagem

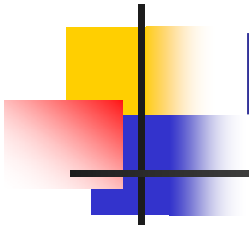
- A função SDL_BlitSurface serve para outras situações.
- Podemos inclusive copiar um pedaço da surface de origem em um pedaço da surface de destino.
- Caso você tenha maior interesse nesta função, consulte a documentação da SDL (online ou faça o download).



Eliminando a Imagem

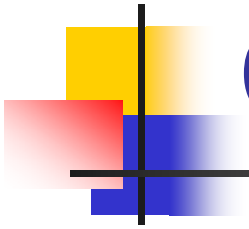
- Uma surface criada na SDL ocupa memória alocada dinamicamente.
- Esta memória DEVE ser liberada quando ela não for mais ser utilizada.
- Assim, antes de encerrar o programa, devemos eliminar a imagem:

SDL_FreeSurface(imagem);



Exercício 1

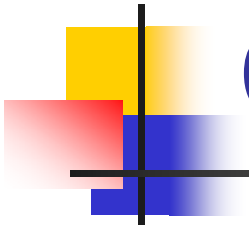
- Modifique o código fonte básico para mostrar uma imagem no redesenho.
  1. Carregar uma imagem do disco (imagem640480.png).
  2. Recuperar o display do vídeo.
  3. Durante o redesenho, faça o *blit* da tela.
  4. Antes de encerrar o programa, libere a memória associada a imagem.
- Salve os arquivos em disco para a próxima aula (pode apagar o executável e os .o).



Carregando Sprites (Azulejos)

- Agora vamos discutir como carregar sprites e blocos de mapa, chamados de Azulejos na Chien2DLite.
- Para isto, utilizamos o seguinte método:

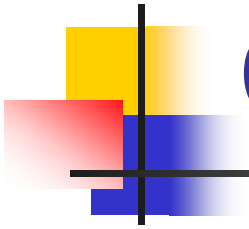
```
unsigned int carregaAzulejo(string
  arquivo, string apelido, int tipoAzulejo);
```



Carregando Sprites (Azulejos)

unsigned int carregaAzulejo(**string arquivo**, string apelido, int tipoAzulejo);

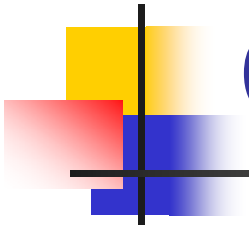
- Arquivo indica o nome do arquivo a ler do disco.



Carregando Sprites (Azulejos)

unsigned int carregaAzulejo(string arquivo, **string apelido**, int tipoAzulejo);

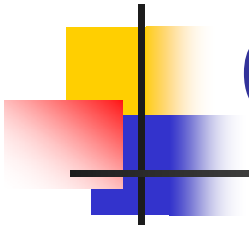
- Apelido indica o nome pelo qual o azulejo será chamado no sistema.



Carregando Sprites (Azulejos)

unsigned int carregaAzulejo(string arquivo, string apelido, **int tipoAzulejo**);

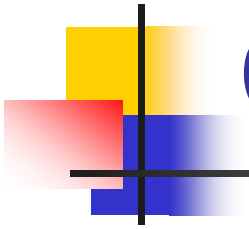
- tipoAzulejo indica o tamanho do bloco:
  - Chien2DLite::azulejo8
  - Chien2DLite::azulejo16
  - Chien2DLite::azulejo32
  - Chien2DLite::azulejo64



Carregando Sprites (Azulejos)

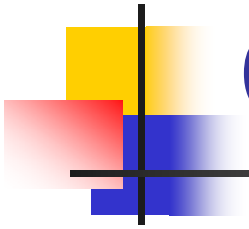
- **unsigned int** carregaAzulejo(string arquivo, string apelido, int tipoAzulejo);

- O inteiro retornado é o identicador único do azulejo no sistema, diferente de zero.
- Se retornar zero, falhou o carregamento.



Carregando Sprites (Azulejos)

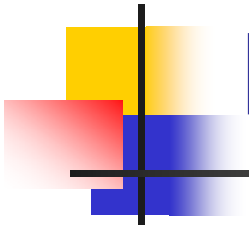
- Normalmente, esta operação falha se:
  1. A imagem não existe no caminho indicado.
  2. Não há mais memória no sistema para a imagem.
  3. A imagem não é quadrada com tamanho potência de 2. Este será o erro mais provável inicialmente.



Carregando Sprites

- Uso típico:

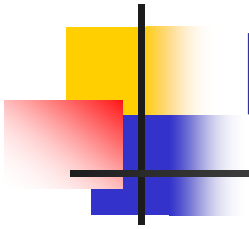
```
unsigned int sprite = video->
   carregaAzulejo(“imagem.png", “apelido",
   Chien2DLite::azulejo32);
```



Mostrando o Sprite

- O azulejo é um tile set, ou seja, ele pode conter vários blocos dentro dele.
- Para desenharmos um sprite precisamos do identificado do sprite, o índice dentro do tile set e da posição x e y:

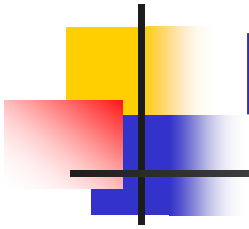
bool desenhaAzulejo(unsigned int identificador, unsigned int indice, int x, int y);



Mostrando o Sprite

- Uso típico:

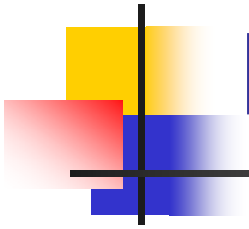
```
video->desenhaAzulejo(sprite, 0,
  30,224);
```



Eliminando o Sprite

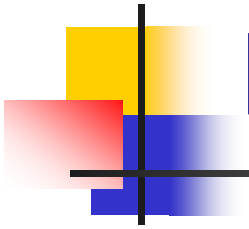
- Em um jogo com muitas fases, po carregamento de recursos a cada fase é inevitável.
- Nestes casos, é preciso eliminar azulejos que não serão mais utilizados:

void removeAzulejo(unsigned int id);



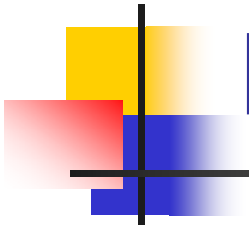
Exercício 2

- Modifique o exercício anterior para:
  1. Carregue o sprite no arquivo “sprite.png”.
  2. Mostre-o na tela SOBRE a imagem carregada.
  3. Elimine o sprite antes de encerrar o modo de vídeo.
- Obs1: a seqüência no redesenho importa. O que é desenhado por último é desenhado em cima.
- Obs2: guarde os arquivos para a próxima aula.



Desafio

- Modifique o programa para desenhar mais de um *sprite* na tela.
- Depois, experimente fazer um sprite atravessar a tela.
- Para ficar mais interessante, faça o sprite ir e vir de um canto a outro da tela.



Próxima Aula

- Desenhando um Tilemap
- Desenhando Fontes
- Tratando a Entrada