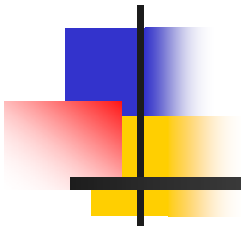


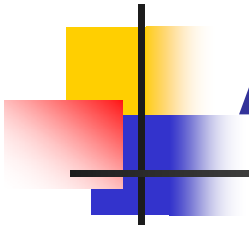
Programação Gráfica – Parte 2

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

Paulo V. W. Radtke

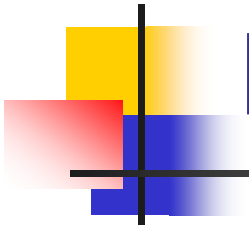
pvwradtke@gmail.com

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



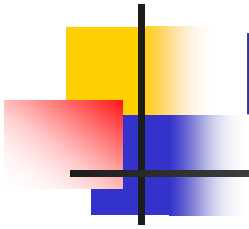
AVISO IMPORTANTE!!

- 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.



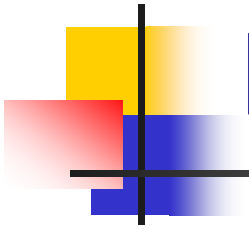
Entrega da 2ª Parcial

- Datas importantes:
  - 12 a 17 de Junho.
  - Relatório impresso contendo:
    - Código fonte do protótipo da interface.
    - Impressão no relatório dos recursos gráficos da fase/jogo (*tilemaps*, sprites, cenários de fundos, protótipo, etc).
    - Discussão do uso dos recursos com a lógica do jogo no terceiro bimestre.
  - Defesa em laboratório do protótipo e entrega do relatório com a equipe completa.



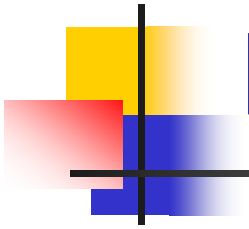
Entrega da 2ª Parcial

- Por recursos gráficos,entende-se que neste bimestre já teremos:
  1. TODOS os sprites necessários para o demo.
  2. TODAS as telas de fundo/tilemaps.
  3. TODAS as fontes.
- Logo, espera-se que o relatório inclua TODOS estes elementos.



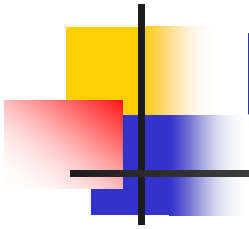
Desafios e Tarefas

- Alguém descobriu como converter uma surface de um formato qualquer para o formato do display?
  - **SDL_DisplayFormat**
- Qual a estrutura usada para indicar os recortes da SDL_BlitSurface?
  - **SDL_Rect**
  - A explicação desta estrutura está nos docs da SDL.



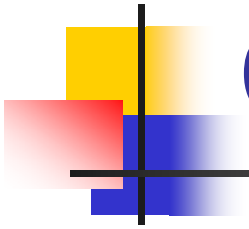
Desafios e Tarefas

- É importante mantermos as SDL_surfaces desenhadas na tela no mesmo formato que o display.
- O maior impacto é o desempenho do blit.
- Experimente converter a imagem carregada no último exercício e veja o impacto na velocidade do sprite se movimentando ANTES e DEPOIS.



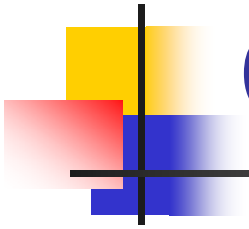
Desafios e Tarefas

- Para mover um sprite, precisamos de duas variáveis auxiliares em cada eixo:
  - Posição do sprite.
  - Velocidade (deslocamento em pixels por quadro).
- A cada quadro, somamos a posição a velocidade.
- Se a posição sai da tela, invertemos o sentido da velocidade.



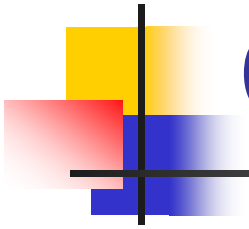
Conteúdo

- Desenhando Fontes
- Tratando a Entrada



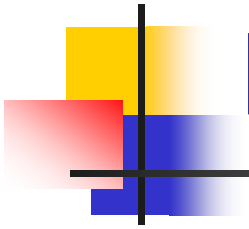
Conteúdo

- Para a aula de hoje utilizaremos como base o exemplo da aula passada.
- Além do exemplo da aula passada, vamos utilizar um arquivo de fontegerado pelo **Bitmap Font Builder**.
- O arquivo se encontra no site, com o nome ***comics32.png*** (32x32).



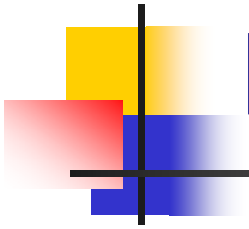
Conteúdo

- Nota: o Bitmap Font Builder grava os arquivos no formato TGA.
- Para economizar espaço no disco, podemos utilizar um formato como PNG.
- Como fazer isso?
- Utilizando o Paint.NET, abrimos o arquivo e salvamos no formato desejado.



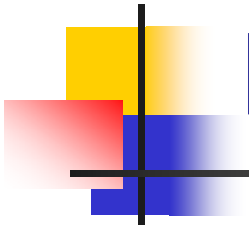
Desenhando Fontes

- As fontes geradas pelo Bitmap Font Builder são suportadas diretamente pela **Chien2DLite**.
- A biblioteca procura automaticamente por pontos transparentes (magenta) para determinar as larguras de fontes e desenhá-las com um pixel de espaçamento.



Desenhando Fontes

- Similar a sprites (azulejos), quatro métodos são utilizados para gerenciar e desenhar fontes:
  - unsigned int carregaFonte(string arquivo, string apelido, int tipoFonte);
  - void removeFonte(unsigned int id);
  - bool desenhaTexto(string texto, unsigned int idFonte, int x, int y, int largura, int alinhamento);
  - bool dimensoesTexto(string texto, unsigned int idFonte, int *largura, int *altura);



Desenhando Fontes

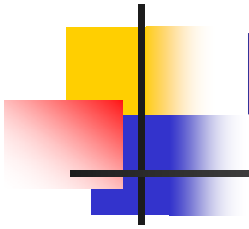
- *unsigned int carregaFonte(string arquivo, string apelido, int tipoFonte);*
- O método retorna o identificador da fonte.
- **arquivo** é o nome em disco da imagem com a fonte.
- **apelido** é o nome lógico da fonte.
- **tipoFonte** é o tamanho da fonte:
  - Chien2DLite::fonte8
  - Chien2DLite::fonte16
  - Chien2DLite::fonte32
  - Chien2DLite::fonte64

Desenhando Fontes

- Uso típico:

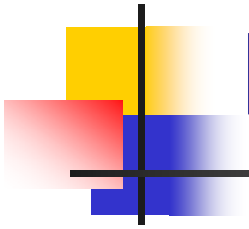
```
unsigned int fonte32 = video->
  carregaFonte("comics32.png", "Comics
  Sans 32", Chien2DLite::fonte32);
```

*Obs: **video** é o ponteiro para um objeto do tipo Chien2DLitep.*



Desenhando Fontes

- *void removeFonte(unsigned int id);*
- Este método remove uma fonte do sistema.
- Para tal, ele recebe **id**, o identificador único da fonte (retornado no *carregaFonte*).



Desenhando Fontes

- Uso típico:

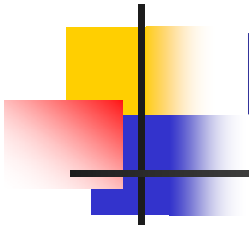
```
video->removeFonte(comics32);
```

Desenhando Fontes

- *bool dimensoesTexto(string texto, unsigned int idFonte, int *largura, int *altura);*
- Este método retorna, em *pixels*, a altura e largura necessários para desenhar um texto com uma fonte.
- Este método retorna valores via ponteiros. O *booleano* retornado indica se os valores retornados são válidos ou não.

Desenhando Fontes

- *bool dimensoesTexto(string texto, unsigned int idFonte, int *largura, int *altura);*
- **texto** é a string a ser deenhada na tela.
- **idFonte** é o identificador único da fonte.
- **largura**: ponteiro para a largura total do texto (uma linha).
- **altura**: ponteiro para a altura total do texto, retorna a altura da fonte selecionada.

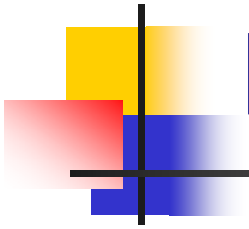


Desenhando Fontes

- Uso típico:

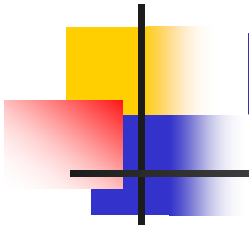
```
int largura, altura;
video->dimensoesTexto("Largura?",
   comics32, &largura, &altura);
```

Obs: caso não tenha certeza da fonte, verifique o retorno do método.



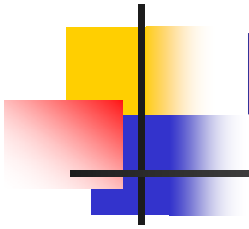
Desenhando Fontes

- *bool desenhaTexto(string texto, unsigned int idFonte, int x, int y, int largura, int alinhamento);*
- Este método desenha um texto na tela, usando a fonte especificada.
- O texto é desenhada em uma certa coordenada referência.
- A largura é usada para efeitos de alinhamento.



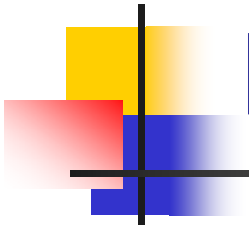
Desenhando Fontes

- *bool desenhaTexto(string texto, unsigned int idFonte, int x, int y, int largura, int alinhamento);*
- **texto:** string desenhada na tela.
- **idFonte**: identificador único da fonte.
- **x,y**: coordenada referência do desenho do texto (canto esquerdo superior).



Desenhando Fontes

- *bool desenhaTexto(string texto, unsigned int idFonte, int x, int y, int largura, int alinhamento);*
- **largura**: largura máxima na qual a linha de texto vai ser desenhada.
- **alinhamento**: indica como o texto será alinhado em relação à largura:
  - Chien2DLite::textoCentralizado
  - Chien2DLite::textoEsquerda
  - Chien2DLite::textoDireita
  - Chien2DLite::textoJustificado

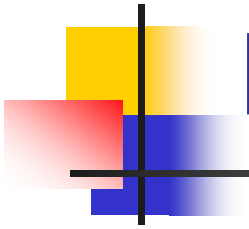


Desenhando Fontes

- Uso típico:

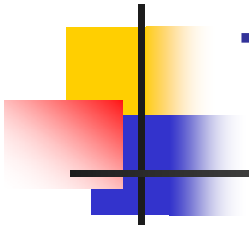
```
video->desenhaTexto("Texto desenhado
   na tela", comics32, 0, 370, 640,
   Chien2DLite::textoCentralizado);
```

Obs: desenha o texto centralizado na tela em relação à resolução horizontal.



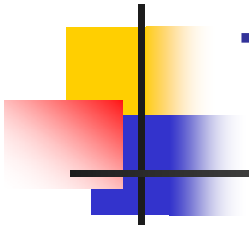
Exercício 01

- Modifique o exercício da aula anterior para escrever um texto na tela.
  1. Carregue a fonte no arquivo **comics32.png** no site da tutoria.
  2. Desenhe o texto centralizado na tela na linha 370 (vertical).
  3. Antes de encerrar o sistema de vídeo, remova a fonte da memória.



Tratando a Entrada

- Toda entrada na SDL é tratada como um evento.
- Pressionar uma tecla é um evento.
- Soltar uma tecla é outro evento.
- Apertar o botão do mouse é um evento.



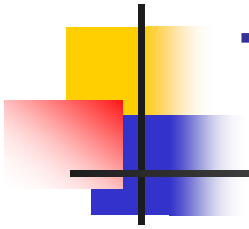
Tratando a Entrada

- Analisando o código fonte main.cpp, temos a função **entrada**.
- Esta função modifica o valor da variável global **sai**.
- Para isto, ela utiliza os eventos da SDL, quando a tecla **ESC** é pressionada o quando o botão de fechar a janela é clicado pelo mouse.

Tratando a Entrada

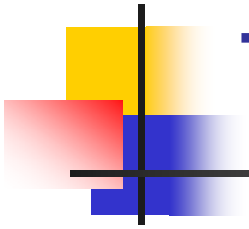
- A função entrada tem basicamente esta estrutura:

```
void entrada()
{
    // A estrutura que recebe os eventos
     SDL_Event event;
     // Enquanto houverem eventos a processar ...
     while(SDL_PollEvent( &event ))
     {
         // Trata aqui os eventos recebidos
      }
}
```



Tratando a Entrada

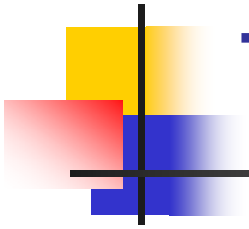
- Dois elementos chave participam desta operação:
  - A união SDL_Event.
  - A função SDL_PollEvent:
- A função SDL_PollEvent procura o próximo evento na fila do sistema.
- Se não existe evento, ela retorna false.
- Se há um evento, seus dados são preenchidos na estrutura passada por ponteiro.



Tratando a Entrada

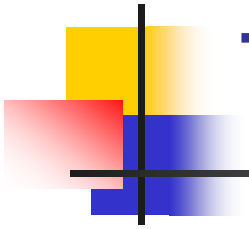
- A união SDL_Event tem esta aparência:

```
typedef union{
    Uint8 type;
    SDL_ActiveEvent active;
    SDL_KeyboardEvent key;
    SDL_MouseMotionEvent motion;
    SDL_MouseButtonEvent button;
    SDL_JoyAxisEvent jaxis;
    SDL_JoyBallEvent jball;
    SDL_JoyHatEvent jhat;
    SDL_JoyButtonEvent jbutton;
    SDL_ResizeEvent resize;
    SDL_ExposeEvent expose;
    SDL_QuitEvent quit;
    SDL_UserEvent user;
    SDL_SywWMEvent syswm;
} SDL_Event;
```



Tratando a Entrada

- Nos interessarão seis tipos de evento (entre parênteses, estrutura e nome do parâmetro):
  - SDL_KEYUP (campo SDL_KeyboardEvent, *key*)
  - SDL_KEYDOWN (campo SDL_KeyboardEvent, *key*)
  - SDL_QUIT
  - SDL_MOUSEBUTTONUP (SDL_MouseButtonEvent, *button*)
  - SDL_MOUSEBUTTONDOWN (SDL_MouseButtonEvent, *button*)
  - SDL_MOUSEMOTION (SDL_MouseMotionEvent, *motion*)



Tratando a Entrada

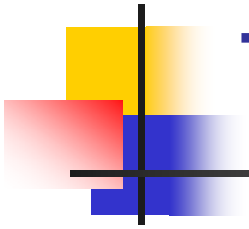
- Exemplo: tratando pressionamento de uma tecla.

```
case SDL_KEYDOWN:
                switch(event.key.keysym.sym)
                {
                        // Pressionou ESC?
                        case SDLK_ESCAPE:
                            sai=true;
                            break;

                        default:
                            break;
                }
                break;
```

Tratando a Entrada

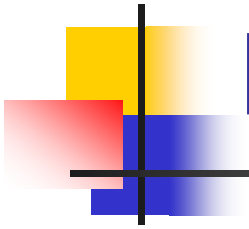
- Exemplo: tratando uma tecla deixando de ser pressionada

```
case SDL_KEYUP:
                switch(event.key.keysym.sym)
                {
                        // Pressionou ESC?
                        case SDLK_ESCAPE:
                            sai=false;
                            break;

                        default:
                            break;
                }
                break;
```



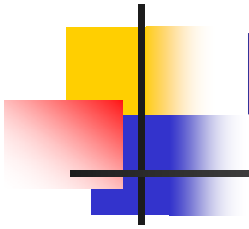
Tratando a Entrada

- Teclas importantes:
  - SDLK_ESCAPE
  - SDLK_UP, SDLK_DOWN, SDLK_LEFT, SDLK_RIGHT
  - SDLK_SPACE
  - SDLK_RALT, SDLK_LALT
  - SDLK_RSHIFT, SDLK_LSHIFT



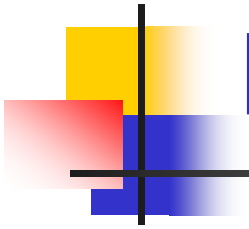
Exercício 02

- Modifique o exercício anterior para cumprir as seguintes etapas:
  1. Desenhe um novo sprite, controlado pelo usuário, usandoas variáveis **px** e **py** para controlar a sua posição na tela.
  2. Crie as variáveis globais **cima**, **baixo**, **esquerda** e **direita**, inicializadas com false.
  3. Modifique a função entrada para que alterar os valores de **cima**, **baixo**, **esquerda** e **direita** de acordo com o pressionamento das teclas do cursor.
  4. De acordo com o estado das teclas do cursor, mude a posição do sprite controlado pelo usuário (mude os valores de **px** e **py**).



Desafio

- O exercício anterior possui um *sprite* controlado pelo teclado.
- Você consegue fazer um *sprite* ser controlado simultaneamente pelo mouse?
- Quando necessário, faça referência à documentação da SDL.



Próxima Aula

- Desenhando Tilemaps.
- Testando colisão de sprites.