Universidade Federal de São Carlos
Departamento de Computação

Disciplina: Estruturas de Dados
Professor Roberto Ferrari

JOGO 3- EDinius

Integrantes:
Bruna Zamith
João Victor Pacheco
Marcos Faglioni
Rodrigo Salmen

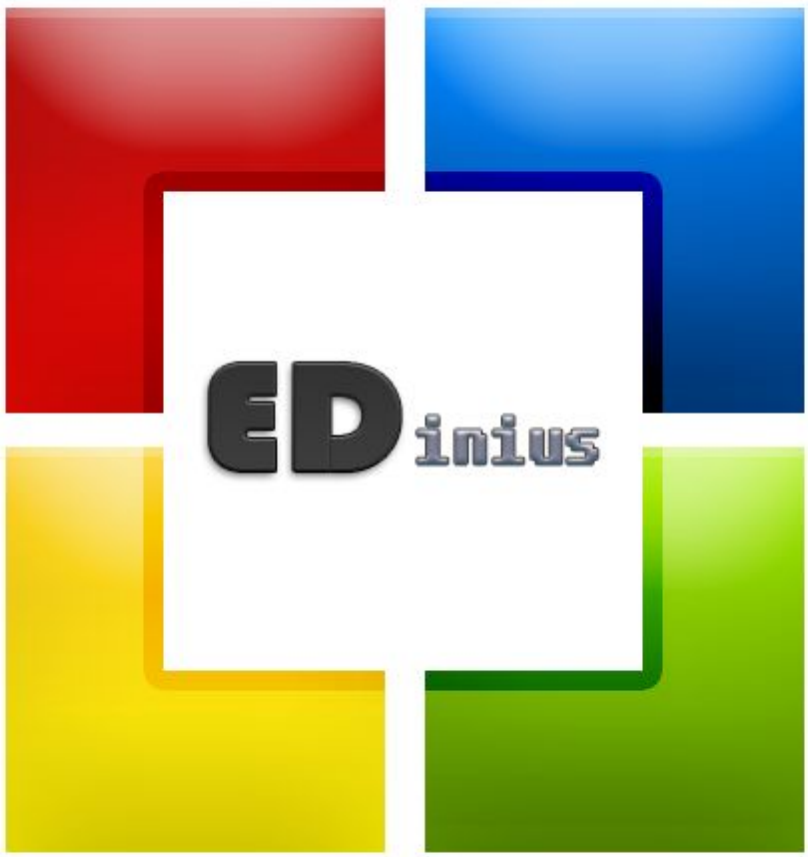
Desenvolvedores:

Bruna Zamith	bruna.zamith@hotmail.com
João Victor Pacheco	jvp1805@gmail.com
Marcos Faglioni	marcosfagli@hotmail.com
Rodrigo Salmen	rodrigosalmen2012@hotmail.com

OBJETIVOS

O jogo desenvolvido (EDinius) foi baseado no famoso jogo Genius, cujo objetivo é memorizar e reproduzir uma sequência de cores aleatórias. Inicialmente, o jogo apresenta uma sequência de cores. Em seguida, solicita que o usuário insira as cores na ordem mostrada. Caso o usuário o faça corretamente, o jogo acrescentará à sequência mais uma cor, tornando o jogo gradativamente mais difícil. O jogador perde quando ele erra a sequência.

A imagem abaixo mostra a interface do jogo:



METODOLOGIA

A estrutura utilizada para o jogo foi a pilha estática, já que o jogo requer poucos elementos para serem armazenados.
Cada vez que o jogo introduz uma nova cor à sequência, essa mesma é empilhada. Como a primeira cor a ser empilhada deve ser a primeira a ser inserida pelo usuário, se faz necessária a utilização de uma pilha auxiliar.
Antes de a sequência ser impressa, a pilha que contém a ordem das cores é esvaziada com a função desempilha, e os elementos são empilhados em uma pilha auxiliar. Daí, o primeiro a ser inserido na pilha principal será o primeiro a ser desempilhado da pilha auxiliar, podendo ser utilizado para imprimir a imagem correta ou para decidir se o input do usuário está correto.

DESENVOLVIMENTO

O jogo foi desenvolvido com o auxílio do SDL2, daí a necessidade de introduzir as bibliotecas "SDL.h" e "SDL_image.h". Essa última é adicionada para possibilitar o carregamento de imagens em formatos diferentes de BMP(bitmap).

A primeira função, loadSurface, carrega uma imagem na forma de uma superfície (SDL_Surface).

```
SDL_Surface* loadSurface( std::string path )
{
    SDL_Surface* loadedSurface = IMG_Load( path.c_str() );
    if( loadedSurface == NULL )
    {
        cout << "Impossivel carregar imagem! SDL Error: " << path.c_str() << SDL_GetError() ;
    }

    return loadedSurface;
}
```

O único argumento dessa função é o diretório que contém a imagem a ser carregada. Caso não seja possível carregar a imagem, a função imprime uma mensagem de erro.

A outra função, loadMedia(), utiliza a função anterior para carregar as imagens utilizadas no jogo, retornando verdadeiro caso isso seja feito com sucesso. As imagens são cinco: default , na qual nenhuma das cores está pressionada; vermelho, amarelo, verde e azul. Essas últimas quatro são colocadas em um vetor de superfícies.

```
bool loadMedia()
{
    bool success = true;

    Default = loadSurface( "Default.png" );
    if( Default == NULL )
    {
        cout << "Impossivel carregar imagem";
        success = false;
    }


    Imagens[ Azul ] = loadSurface( "Azul.png" );
    if( Imagens[ Azul ] == NULL )
    {
        cout << "Impossivel carregar imagem";
        success = false;
    }

    Imagens[ Vermelho ] = loadSurface( "Vermelho.png" );
    if( Imagens[ Vermelho ] == NULL )
    {
```

Repare que foi utilizado um enum, para tornar o código mais intuitivo: a imagem do botão vermelho foi guardada em Imagens[Vermelho].

A última das funções é a ImprimeSuperfície, que imprime uma superfície na tela e espera alguns instantes. O código da função é mostrado abaixo.

```
void ImprimeSuperficie(SDL_Surface* Imagem ,SDL_Surface *tela, SDL_Window* window){
    SDL_BlitSurface( Imagem, NULL, tela, NULL );
    SDL_UpdateWindowSurface( window );
    SDL_Delay(400);
}
```

Essa função recebe três parâmetros: a superfície a ser impressa, a superfície da janela do jogo e a janela do jogo, respectivamente.

A função main contém a inicialização das bibliotecas e o loop principal do jogo, esse último será pormenorizado adiante.

Uma variável do tipo booleana 'rodando' mantém o loop em funcionamento quando verdadeira. Caso essa torne-se falsa a aplicação é encerrada.

O loop consiste em:

- Sortear uma cor com a função rand();
- Empilhar essa cor na pilha que contém a sequência de cores;
- Imprimir a sequência contida na pilha com a função ImprimeSuperficie();
- Receber uma sequência de cores como input do usuário;
- Caso essa sequência esteja errada, a variável rodando recebe falso e a aplicação se encerra;
- Caso contrário, o loop se repete.

CONCLUSÕES

Neste trabalho aprimoramos nosso conhecimento sobre a implementação de Tipos Abstratos de Dados, especificamente pilha. Além disso, aprendemos sobre interface gráfica através do SDL e conseguimos a partir disso implementar um jogo no estilo Genius.