WILLIAN AYRES WEB DEVELOPER
RELATÓRIO DO PROJETO
JOGO RESTA UM

Willian Joris Ayres

RELATÓRIO FINAL DO PROJETO

JOGO DE RESTA UM EM ALLEGRO

Distração e entretenimento

Curitiba

Fevereiro / 2020

Willian Joris Ayres

RELATÓRIO FINAL DO PROJETO

JOGO DE RESTA UM EM ALLEGRO

Distração e entretenimento

Relatório final de projeto, apresentado como requisito de confirmação de conhecimentos no desenvolvimento de softwares em linguagem de programação C com a utilização de bibliotecas externas.

Curitiba

Fevereiro / 2020

Sumário

1. Lista de ilustrações 4
2. Lista de Apêndices 5
3. Introdução 7
4. Explicação do Jogo 8
5. Desenvolvendo o jogo 15
    5.1. Parte Gráfica do Jogo 16
    5.2. Fontes Personalizadas 20
    5.3. Codificação 21
6. Considerações pessoais 46
7. Referências 47

Lista de ilustrações

1. Explicação do Jogo

1.1. Menu principal 8
1.2. Cursor e seletor 8
1.3. Tela e regras 8
1.4. Tela de ajustes 9
1.5. Tabuleiros 9
1.6. Primeiras impressões 10
1.7. Movimentos possíveis 10
1.8. Jogada Realizada 11
1.9. Botão retorna jogada 11
1.10. Antes de retornar jogada 11
1.11. Depois de retornar jogada 11
1.12. Botão sair 11
1.13. Jogo quase perdido 12
1.14. Jogo perdido 12
1.15. Jogo quase ganho 12
1.16. Jogo ganho 12
1.17. Histórico 13
1.18. Primeira página de records 13
1.19. Segunda página de records 13
1.20. Botão para sair do jogo 14
1.21. Fecha programa 14

2. Parte Gráfica do Jogo

2.1. GraphicsGale 16
2.2. Criando nova imagem 16
2.3. Definindo tamanho da imagem 16
2.4. Tamanho da imagem 17
2.5. Iniciando desenho 17
2.6. Salvando imagem 18
2.7. Peça 50x50 18
2.8. Espaço 50x50 18
2.9. Logo 329x92 18
2.10. Título 281x51 18
2.11. Botão_1 161x41 19
2.12. Botão_2 101x21 19
2.13. Botão_3 75x55 19
2.14. Campo Inglês 350x350 19
2.15. Campo Francês 350x350 19
2.16. Fundo 800x600 19

3. Fontes Personalizadas

3.1. TTF_PCX_Font_Converter 20
3.2. Fontes usadas no jogo 20

Lista de Apêndices

1. Iniciando

1.1. Iniciação do Allegro 22
1.2. Variáveis Globais 22
1.3. Função para fechar o programa 22
1.4. Função para atualizar o timer 23
1.5. Travando funções e variáveis 23
1.6. Telas 23
1.7. Struct para volume 23
1.8. Struct para caminhos 24
1.9. Variáveis 24
1.10. Inicializando as structs 24
1.11. Função para leitura dos ajustes 25
1.12. Sounds 25
1.13. Game loop 25
1.14. Check de escape 26
1.15. Finalização 26
1.16. Defines 26

2. Menu

2.1. Carregando um bitmap 27
2.2. Criando um buffer 27
2.3. Carregando fontes e sons 27
2.4. Variáveis auxiliares 27
2.5. Estado de tecla 28
2.6. Leitura do seletor 28
2.7. Check de botão selecionado 28
2.8. Atualização do seletor 29
2.9. Saindo da tela 29
2.10. Desenhando o campo no menu 29
2.11. Desenhando os botões 29
2.12. Efeito de piscar 30
2.13. Destruindo ponteiros 30

3. Regras

3.1. Leitura das regras 31
3.2. Botão de voltar pressionado 31
3.3. Desenhando botão de voltar 31

4. Histórico

4.1. Leitura do histórico 32
4.2. Troca de buffer 32
4.3. Limpa histórico 33
4.4. Controle de buffer 33

5. Records

5.1. Leitura dos records 34

6. Ajustes
6.1. Configurando volume 35
6.2. Reset de configurações 35
6.3. Alterando configurações 36
6.4. Desenhando botões de som 36
6.5. Desenhando botões de configuração 36
7. Tabuleiros
7.1. Struct campo 37
7.2. Declarando tabuleiro 37
7.3. Iniciando tabuleiro 38
7.4. Desenhando os botões do tabuleiro 38
8. Game
8.1. Início do timer 39
8.2. Retorna jogada 39
8.3. Retorna ao menu 39
8.4. Selecionando peça 40
8.5. Game over 40
8.6. Check game 40
8.7. Desenhando tabuleiro 41
9. Jogada
9.1. Definindo a direção 42
9.2. Salva jogada 42
9.3. Atualiza tabuleiro 42
10. Player
10.1. Limpando buffer do teclado 43
10.2. Lendo nome 43
10.3. Check de record 43
10.4. Apagando record 44
10.5. Salvando record 44
10.6. Concatenando record 44
10.7. Desenhando tela de nome completa 45

Introdução

Este documento apresenta a versão final do simulador do jogo “Resta Um” feito por Willian Ayres. Tendo em vista como aperfeiçoamento de conhecimentos prévios de lógica de programação, algoritmos, desenvolvimento de softwares na linguagem C, além da melhoria na abstração de resolução de problemas pelo meio procedural, e também a utilização de bibliotecas externas, que no caso deste relatório é a biblioteca gráfica Allegro 4.4.2.

O contexto de codificação deu-se pela IDE “Code Blocks”, sendo de fácil acesso e gratuito. Porém, para melhor utilização das funcionalidades da biblioteca Allegro 4.4.2, utilizou-se a versão 13.12 da IDE. Existem vários motivos para utilização dessa IDE, por exemplo, ela possui várias ferramentas adicionais, como os “debuggers” e os “compilers”. Além da ótima compatibilidade com a biblioteca gráfica dessa versão.

O código, em sua integridade, é escrito completamente em C, sem muitas enrolações. É compreensível e conciso, porém é necessário um nível intermediário de lógica, conhecimentos da linguagem C e abstração. Algumas partes no desenvolver do programa podem se tornar um pouco complexas caso o desenvolvedor não esteja habituado. O programa foi otimizado até o ponto que o conhecimento atual permitiu.

Para melhor entendimento da codificação, é recomendado ter conhecimentos na utilização da biblioteca Allegro, por conta da utilização de MACROS e funcionalidades na qual possui. Caso haja falta de habitualidade no uso da biblioteca, problemas de entendimento podem se mostrar presentes.

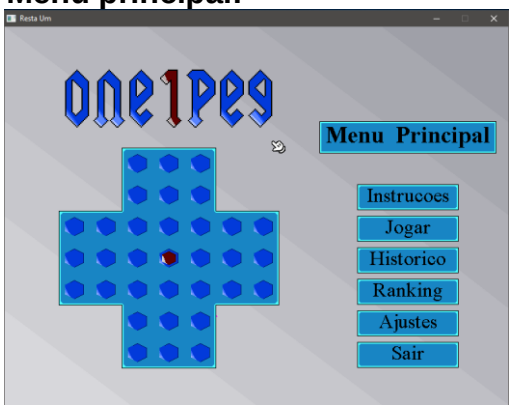
Para utilização de recursos adicionais da biblioteca, foram usados programas como o TTFPCX_Font_Converter, para criação de fontes personalizadas, e o GraphicsGale para o desenho dos sprites.

Se você é atraído pelas belezas do mundo da programação, seguir esse relatório para criar o seu próprio simulador de Resta Um não será um grande problema. Basta seguir corretamente e ajustar conforme suas métricas e seus próprios gostos.

Explicação do Jogo

Assim que iniciado o jogo, abre-se a janela inicial (nomeada de Resta Um) na qual mostra uma logo elaborada em um plano de fundo, um tabuleiro com uma peça ilustrativa e ao lado as opções do menu.

Menu principal:

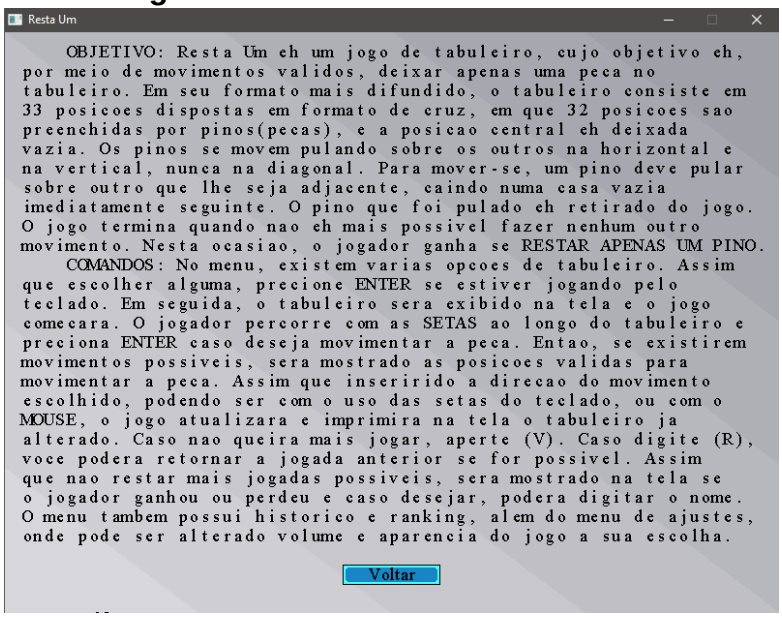


Nota-se que o cursor do mouse é alterado quando dentro da tela do jogo. Como todos os outros botões do jogo, caso passe o mouse sobre ou esteja selecionado ele com o teclado, aparecerá uma marcação sobre o mesmo, para indicar a posição de seleção.

Cursor e seletor:

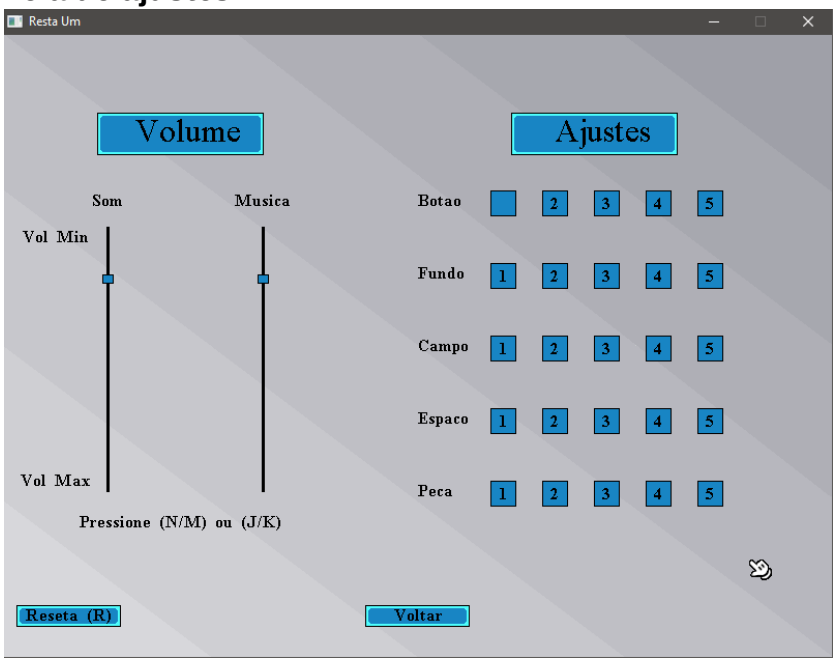
Caso escolha-se a primeira opção, troca-se de tela e exibe-se as instruções do jogo. Além do texto, existe um botão na parte de baixo da tela que caso selecionado, retorna ao menu.

Tela de regras:



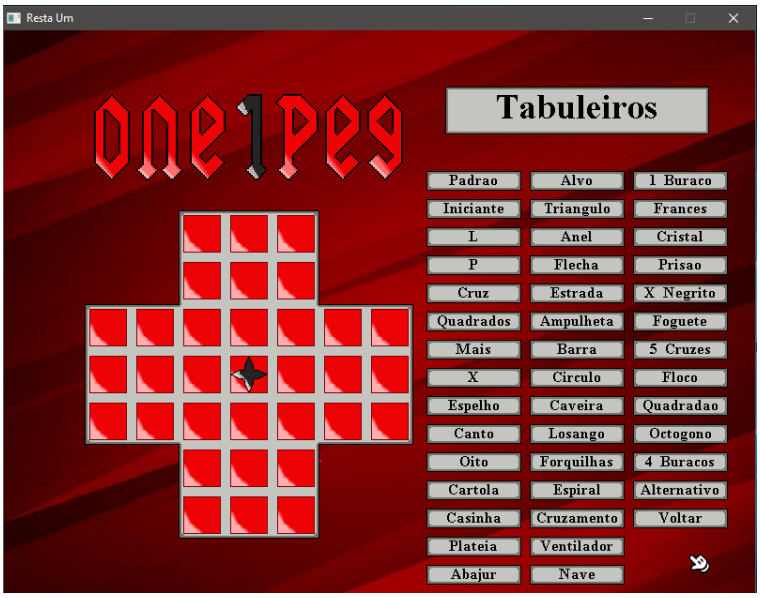
Retornando ao menu, na aba ajustes é onde pode-se alterar várias das configurações básicas do jogo. Dentre elas temos os volumes dos efeitos sonoros e as opções de aparência, podendo ser ajustado conforme o jogador desejar. No canto esquerdo existem um botão para resetar os ajustes para o padrão de fábrica. OBS: Caso o jogador feche o jogo, as suas configurações serão salvas!

Tela de ajustes:



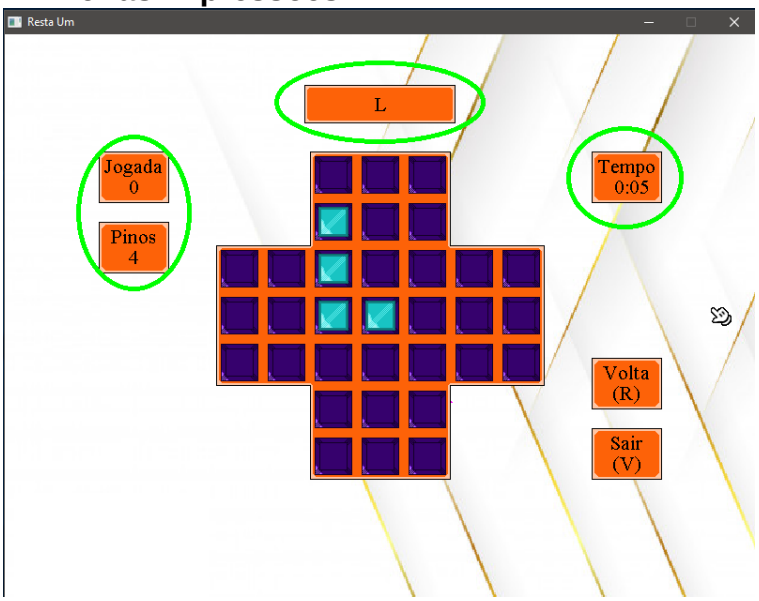
Após usufruir das diferentes configurações, caso retorne ao menu e selecione a opção jogar, será aberto o menu de tabuleiros, contendo 31 botões para tabuleiros padrão inglês, 11 botões para tabuleiros padrão francês e 1 botão para retornar ao menu.

Tabuleiros:



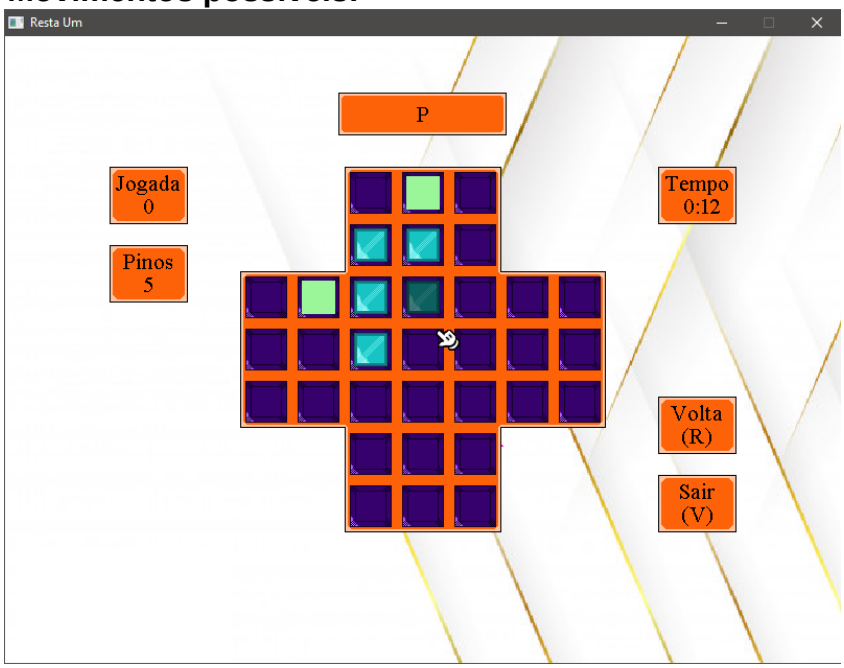
Então, após escolher alguns dos tabuleiros, inicia-se a parte do jogo em si. Aparentemente, as primeiras informações vistas são: o nome do tabuleiro escolhido, a quantidade de jogadas realizadas pelo jogador, o número de pinos restantes em jogo e o tempo decorrido em jogo.

Primeiras impressões:



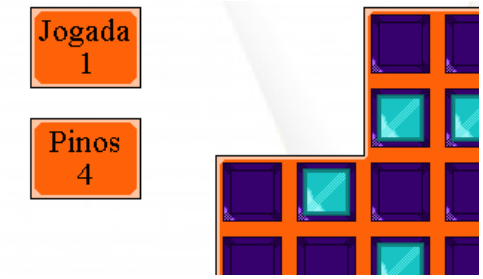
Para realizar alguma jogada, basta analisar o tabuleiro e verificar qual peça pode mover-se efetivamente, e então seleciona-la. Caso a peça não realize jogada nenhuma, é necessário deselecionar ela. Senão, irá aparecer sombras auxiliares que mostram as direções que podem ser realizados os movimentos.

Movimentos possíveis:



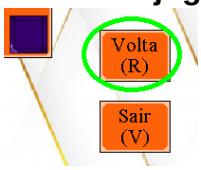
Então, caso deseja-se realizar a jogada, basta apertar alguma das setas do teclado, ou então selecionar para onde deseja movimentar a peça. Após isso, o tabuleiro é atualizado, assim como o número de pinos e jogadas.

Jogada realizada:

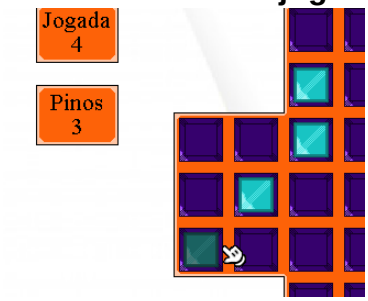


Conforme o desenrolar do jogo, o jogador pode equivocar-se ao fazer uma jogada. Para isso, existe a opção de retornar uma jogada. Para isso, basta apertar o botão de retornar jogada ou pressionar R.

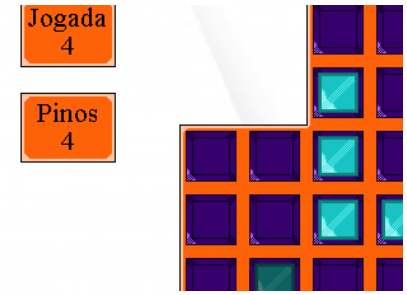
Botão retorna jogada:



Antes de retornar jogada:



Depois de retornar jogada:

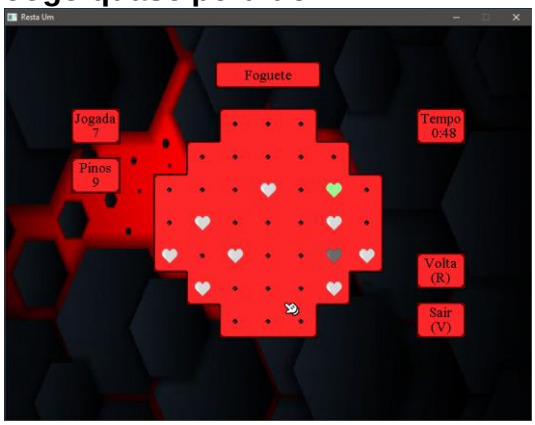


Se você selecionar o botão Sair ou pressionar V, o jogo será retornado ao menu principal.

Botão sair:

Continuando o jogo normalmente, conforme for executando-se as jogadas, válidas ou inválidas, se o jogador não desistir do jogo, fechando-o ou retornando ao menu, uma hora ele precisa chegar ao fim. Caso reste mais de um pino em campo, será exibido uma mensagem de derrota do jogador e então pede-se pelo nome ou apelido dele. Note que na imagem, o jogador não terá mais movimentos possíveis para serem feitos, e caso faça o ultimo movimento possível, ele perderá. Após digitado o nome, pressiona-se a tecla ENTER ou seleciona o botão de mesmo nome.

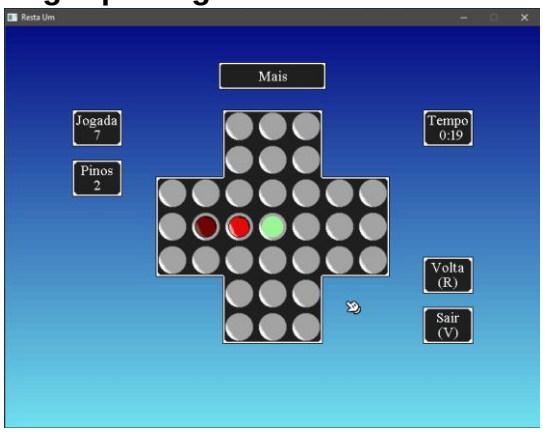
Jogo quase perdido:



Jogo perdido:

Caso reste apenas 1 pino do final do jogo, será exibido uma mensagem de vitória do jogar e novamente pedido pelo nome do mesmo. Note que na imagem, o único movimento possível irá resultar na vitória do jogador.

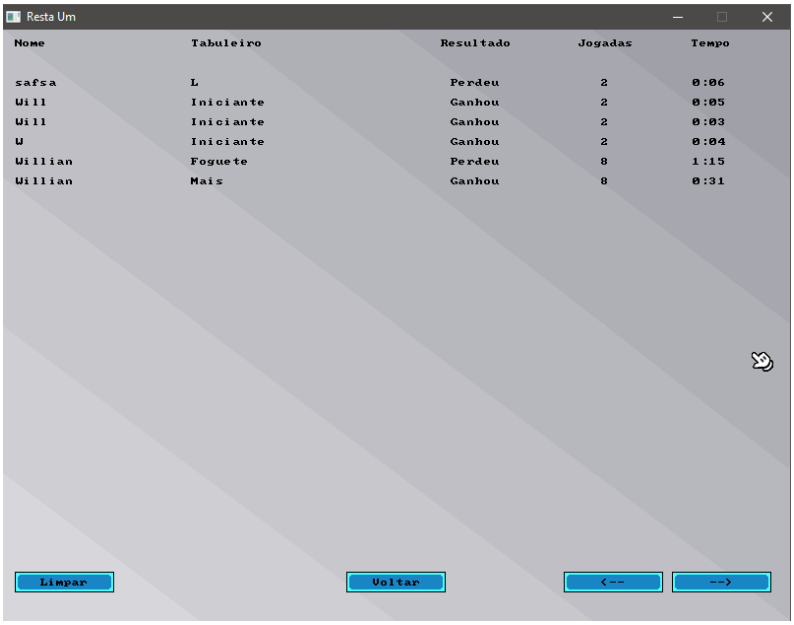
Jogo quase ganho:



Jogo ganho:

Após terminar o jogo, são computadas as informações e salvas no histórico do jogo. Para acessa-lo, basta seleciona a aba histórico no menu. Assim que aberto, são exibidas as informações dos últimos jogos realizados. Caso existam muitas informações, será criada outra página com mais informações e para alterar entre elas, basta selecionar os botões no canto inferior direito. Caso queira limpar essas informações, basta selecionar o botão de limpar ao lado inferior esquerdo. Para retornar ao menu, basta selecionar o botão voltar.

Histórico:



Retorna ao menu, ainda existe a aba de records. Nela, estão gravadas as informações de melhores resultados para cada um dos 42 tabuleiros. Caso em alguma das vezes que o jogador terminar um jogo, ele ganhar e fizer em menos jogadas ou em menor tempo, seu nome será gravado e irá substituir o nome que já estava nos records. Como existem muitos tabuleiros, você precisa alternar entre as páginas de records, assim como na aba de histórico.

Primeira página de records:

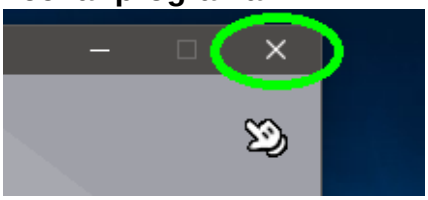
Segunda página de records:

Caso o jogador esteja no menu principal e selecione a opção sair, o jogo será encerrado.

Botão de sair do jogo:

Caso deseje, a qualquer momento do jogo, o jogador pode selecionar o botão de fechar no topo superior direito da janela, para executar a mesma função de sair.

Fechar programa:



Caso o jogador esteja usando teclado, basta pressionar ESC para realizar todas essas funções.

Basicamente, isso cobre todas as funções do jogo. Basta desfrutar da junção de todas elas para ter uma experiência maravilhosa e se divertir muito!

Desenvolvendo o Jogo

Como a lógica e o ambiente de programação foram todos em C, para a criação de uma nova janela com recursos gráficos, necessita-se de uma biblioteca gráfica. A biblioteca escolhida foi a Allegro.

Existem várias versões disponíveis, mas a escolhida foi a 4.4.2. Mesmo sendo uma versão antiga, acaba tendo maior utilidade por ser voltada a programação em C.

A instalação da biblioteca é relativamente fácil, sendo que só é necessário baixar os arquivos da página oficial da Allegro e extrai-los na pasta de origem da sua IDE.

No caso de uso do Code Blocks, é necessário o linker para efetuar a compilação. Basta adicionar o arquivo “allegro-4.4.2-monolith-md-debug.dll”.

Após feita a instalação, antes de começar a codificação do programa, precisa-se ter em mãos os desenhos gráficos e as fontes do jogo. E se desejar ter algum efeito sonoro no jogo, precisa-se converter arquivos de efeitos sonoros, por exemplo alguma batida ou efeito de clique, para o formato wave (.wav), e as músicas do jogo precisam ser no formata midi (.midi).

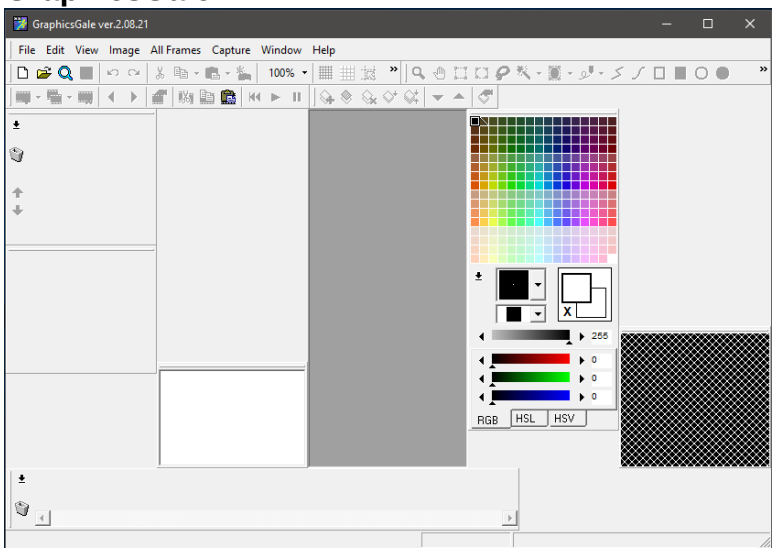
Após termino da codificação, se deseja executar o programa fora da IDE, é necessário colocar na mesma pasta do executável os arquivos: “allegro-4.4.2-monolith-md-debug.dll”, “allegro-4.4.2-monolith-md.dll”, “libgcc_s_dw2-1.dll”. Todos esses arquivos estão presentes nos arquivos baixados da Allegro.

Parte Gráfica do Jogo

Deve-se ter um conhecimento prévio que ao usar a biblioteca Allegro, você precisa inserir imagens com extensão do tipo bmp. Essas imagens são conjuntos de pixels que carregam informações de cor, sendo que a união desses pixels é chamada de sprite. Um sprite é carregado como bitmap.

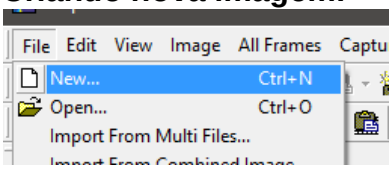
Para a criação dessas imagens, a aplicação usada foi o GraphicsGale. É uma aplicação de fácil uso, com muitas ferramentas para desenhos pixelados.

GraphicsGale:



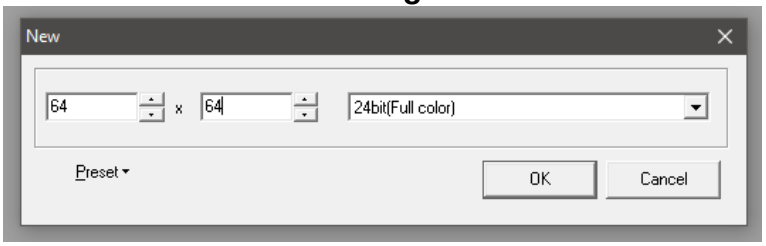
Para iniciar o desenho de alguma imagem, basta clicar na aba “File” e então em “New”.

Criando nova imagem:



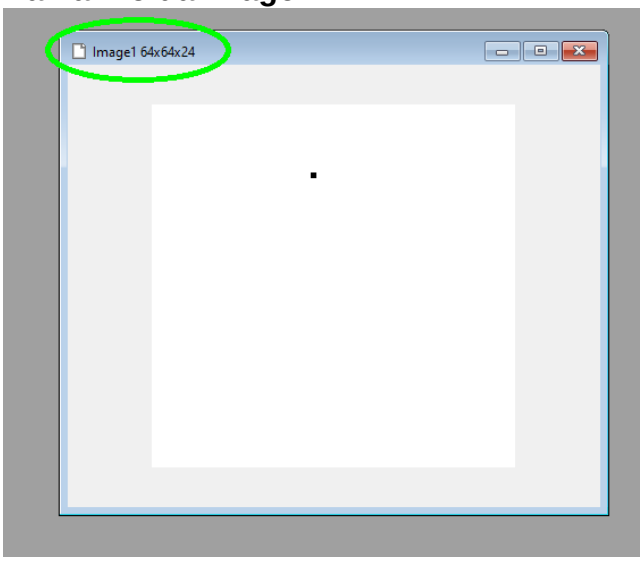
Então, deve-se especificar o tamanho da imagem e o tipo de leitura de cores para mesma. OBS: todas as imagens desse jogo foram feitas usando o padrão 24bit (Full color).

Definindo tamanho da imagem:



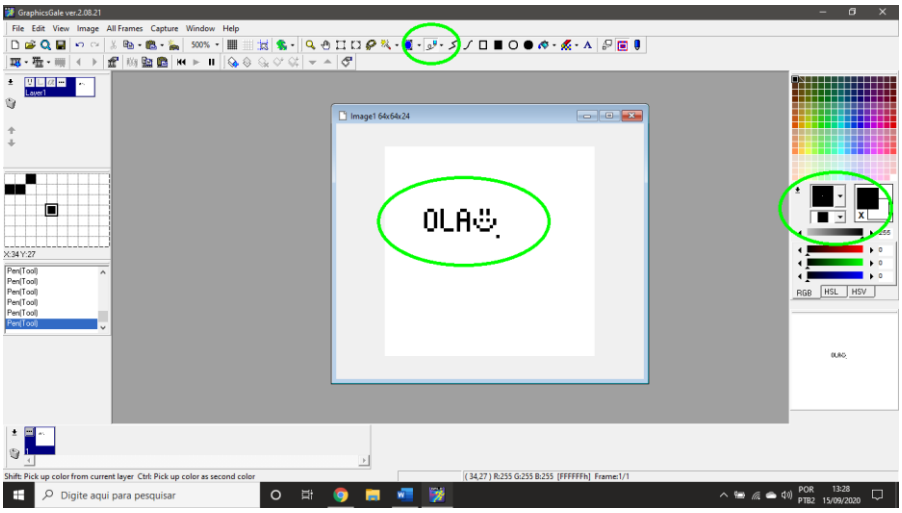
A imagem criada terá o tamanho (em pixels) especificado anteriormente, e poderá ser desenhada dentro desse espaço definido.

Tamanho da imagem:



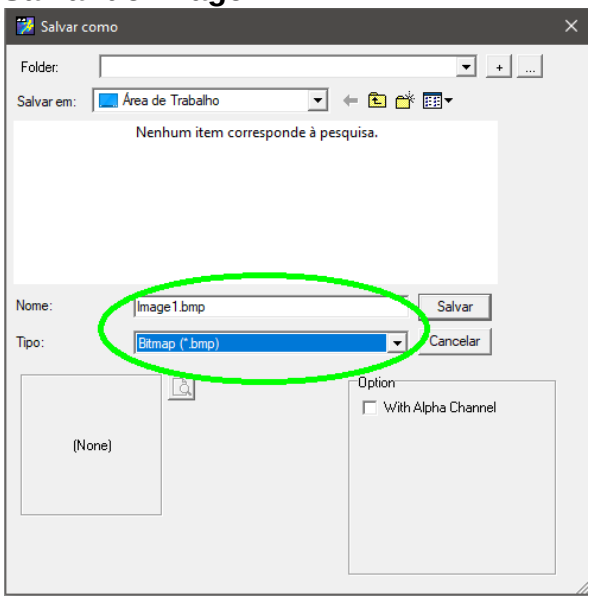
Para desenhar basta selecionar a caneta de desenho e pressionar o botão esquerdo do mouse no pixel que deseja alterar, assim ele assumirá a cor na qual está selecionada.

Iniciando desenho:



Caso tenho finalizado seu desenho, basta salva-lo na pasta do seu programa para ter maior controle e evitar erros de caminho do arquivo. Não se esqueça de salvar a imagem como bitmap.

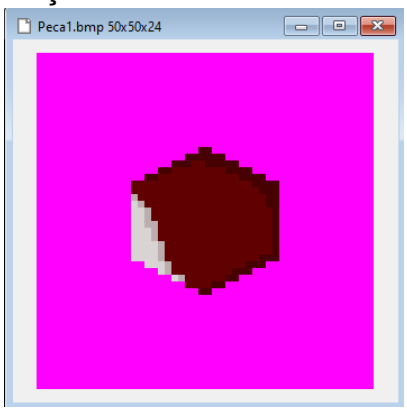
Salvando imagem:



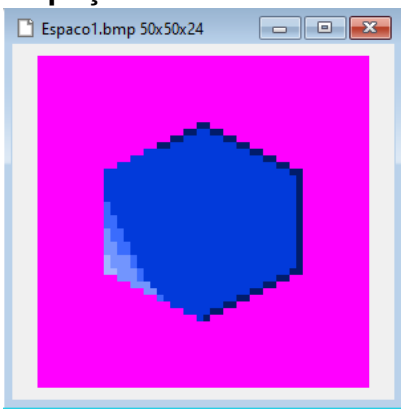
Após salva, sua imagem está pronta para ser usada no jogo.

O tamanho das imagens deve ser de escolha do desenvolvedor, porém para esse jogo foram usados alguns tamanhos específicos. A seguir, será mostrado alguns tamanhos e desenhos de bitmaps para o jogo.

Peça 50x50:

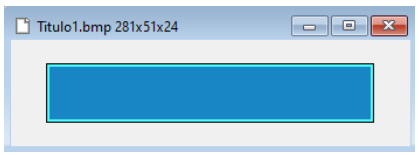


Espaço 50x50

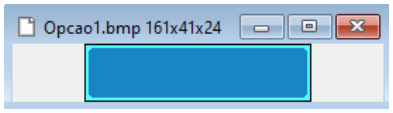


Logo 329x92:

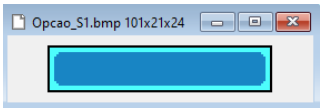
Titulo 281x51:



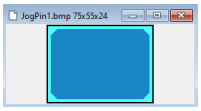
Botão_1 161x41:



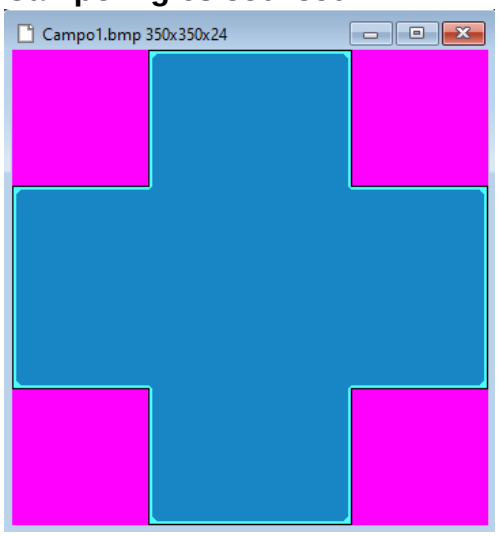
Botão_2 101x21:



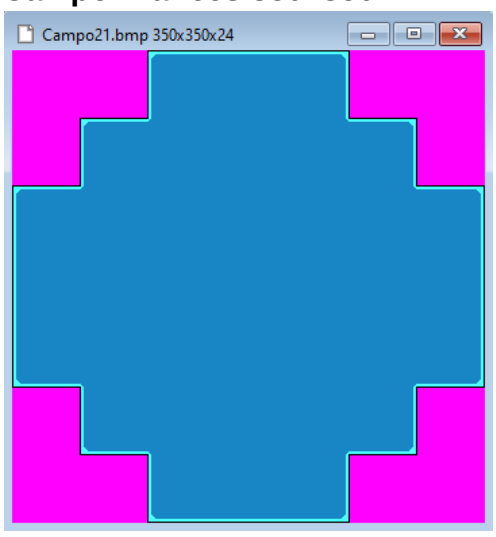
Botão_3 75x55:



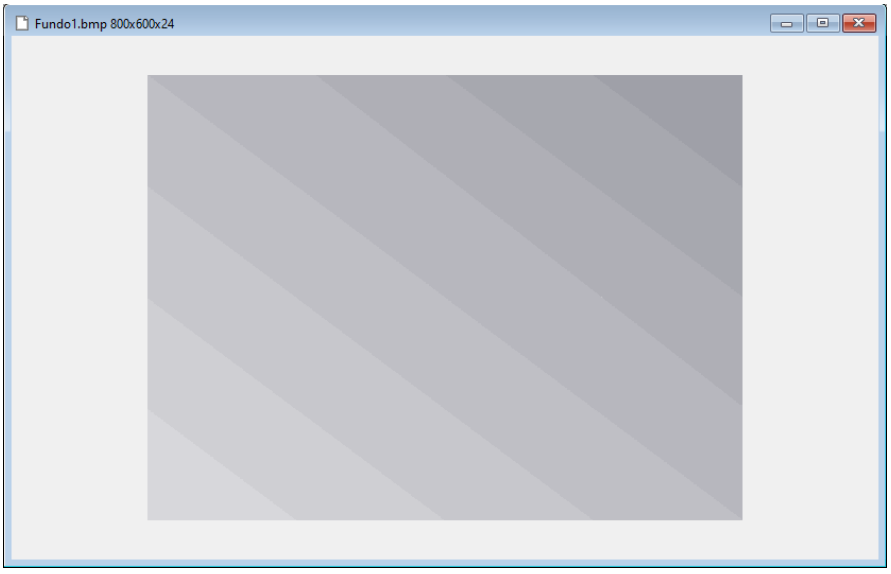
Campo Inglês 350x350:



Campo Francês 350x350:



Fundo 800x600:



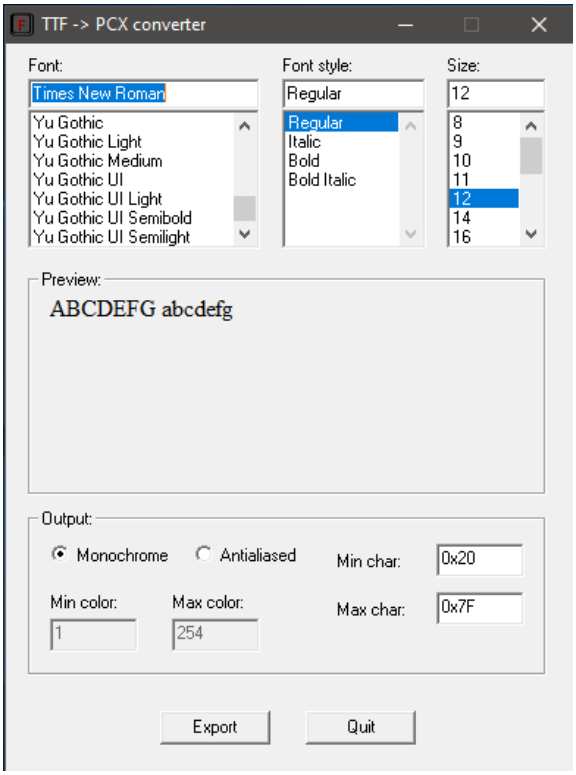
Obs.: A coloração rosa ao fundo das imagens serve para transparência na hora da leitura dos bitmaps. O código rgb (código de geração de cor baseado nas três cores: vermelho, verde, azul) dessa cor é 255, 0, 255 e o código hexadecimal dela é #ff00ff.

Fontes Personalizadas

Para utilização de fontes personalizadas com a biblioteca gráfica Allegro, é necessário criar um arquivo especial com extensão pcx. Só assim, o programa poderá ler corretamente essa fonte.

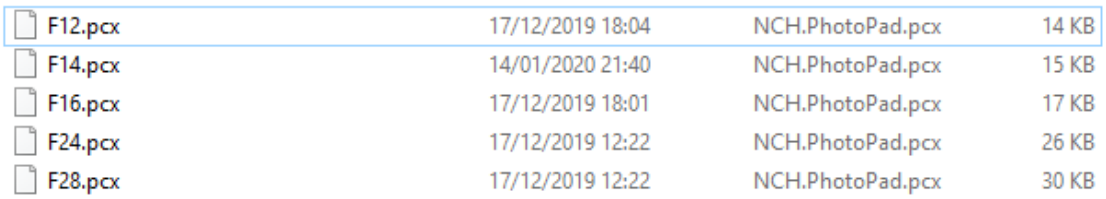
Pode-se baixar alguma fonte pronta na internet, mas para esse jogo, foi utilizado o programa TTF_PCX_Font_converter. Ele irá receber uma fonte padrão (com extensão ttf) e irá converter para uma fonte com extensão pcx.

TTF_PCX_Font_converter:



Basta selecionar a fonte padrão que deseja usar, o seu estilo e tamanho, e então exporta-la. As fontes usadas no jogo têm os tamanhos a seguir:

Fontes usadas no jogo:



Para ter um controle maior, coloque em uma pasta junto aos arquivos de programa do jogo.

Codificação

Tendo todas as configurações básicas em mãos, hora de iniciar a parte do código do jogo. A separação em arquivos diferentes é feita conforme métrica do desenvolvedor, não alterando o produto final, apenas facilitando o entendimento do código.

Um código em allegro, normalmente é divido em partes para ter maior controle do código. Primeiro, declara-se as variáveis que serão usadas na função, deixando todas previamente declaradas e se necessário, inicializadas. Então, inicia-se os BITMAPS que serão usados (as imagens gráficas usados na sua aplicação, vulgo sprites). Após, inicia-se os sons e as fontes personalizadas. Parte-se então para o início do loop da função, tendo como base alguma tecla de escape para o encerramento da função ou até mesmo do programa.

Dentro do loop, primeiramente codifica-se a parte de INPUTS, a leitura e entrada de dados dos usuários (como teclas pressionadas, opções a serem selecionadas). Após isso, a parte de UPDATE, tanto de variáveis de funções, de estados ou até mesmo para troca de telas. E então, executa-se a parte de DRAWN, onde desenha-se todos os bitmaps importados em buffer, para então no final desenhar esse buffer na tela padrão.

E no final, a parte de FINALIZAÇÃO, destruindo os bitmaps, fontes e sons usados, desalocando espaços alocados e então retornando valores se necessário.

Se seguir esse padrão para toda função que utilize as funções do allegro, não encontrará erro, além do código tornar-se muito mais conciso.

Iniciando

A primeira coisa a se fazer no código, é instalar as drivers do allegro. Primeiro, declara-se a biblioteca da “allegro.h” e então chama-se as funções inicias de instalação. Faz-se a verificação se o allegro inicia corretamente, então instala-se o timer, o mouse, o teclado, o som (com parâmetros para autodetecção), verificação de cores padra padrão 32bits, tamanho da janela (para este jogo é de 800x600), o padrão de letras e por último o nome da janela.

Iniciação do Allegro:

```
#include <allegro.h>

if (allegro_init ())
    exit (1);
install_timer ();
install_mouse ();
install_keyboard ();
install_sound (DIGI_AUTODETECT, MIDI_AUTODETECT, NULL);
set_color_depth (32);
set_gfx_mode (GFX_AUTODETECT_WINDOWED, 800, 600, 0, 0);
set_uformat (U_ASCII);
set_window_title ("Resta Um");
```

Variáveis globais:

```
volatile int exit_program;
volatile int timer;
```

Após a iniciação correta do allegro no programa, precisamos definir duas funções auxiliares para o decorrer do programa. Cada uma dessas funções necessita de uma variável global própria (O nome das variáveis é pessoal).

Função para fechar o programa:

```
void fecha_programa () {
   exit_program = TRUE;
}
END_OF_FUNCTION (fecha_programa);
```

A primeira variável está relacionada ao fechamento da janela do programa. Note que a variável é tratada como booleana, e será atribuída como verdadeiro na função. Ao final dela, precisa-se adicionar a macro de término de função. A implementação da função seria algo do gênero.

Função para atualizar timer:

```
void incrementa_timer () {

   timer++;

}

END_OF_FUNCTION (incrementa_timer)
```

A segunda função refere-se à utilização de um timer no jogo. A implementação dela é apenas a atualização da variável timer. No final da dela, é necessário colocar a macro de fim de função.

Travando funções e variáveis:

```
exit_program = FALSE;

LOCK_FUNCTION (fecha_programa);

LOCK_VARIABLE (exit_program);

set_close_button_callback (fecha_programa);

LOCK_FUNCTION (incrementa_timer);

LOCK_VARIABLE (timer);
```

Voltando à função main, inicializa-se a variável de fechamento do programa como falso. Então, é preciso travar essas funções e variáveis de acesso global usando as macros nativa do allegro, e por último, ativa-se o botão de fechamento de janela, atribuindo a função de fechar o programa a ela.

Após isso, cria-se uma variável, podendo ser local ou global (se for local, precisará ser passada como parâmetro para todas as outras funções) para definir em qual tela o jogador estará. Para facilitar, cria-se também uma enumeração de nomes com todos os estados de tela possíveis no jogo.

Telas:

```
enum {MENUSCREEN, RULE, HISTORIC, RECORD, SETTINGS, BOARDS, GAMEI, GAMEF, MOVE,
NAME};

int screen_state;
```

Para ter maior controle do que irá acontecer no código, utiliza-se duas structs. A primeira é para o controle do volume do jogo, contendo uma variável para o controle do som dos efeitos sonoros, e outra para o controle do som da música.

Struct para volume:

```
typedef struct {int noise; int music;} Volume;
```

Struct para caminhos:

```
typedef struct {
    char bg [30];
    char boardF [30];
    char boardI [30];
    char buttonB [30];
    char buttonM [30];
    char buttonS [30];
    char logo [30];
    char piece [30];
    char pieceBg [30];
    char pieceV [30];
    char title [30];
} Paths;
```

A segunda é para as configurações do programa, possuindo strings que contém os caminhos de onde estão salvos os sprites que serão impressos na tela. Conforme decorrer do código, fará mais sentido. Um caminho para o sprite de background, outros dois para os tabuleiros, três para botões, um para a logo, um para a peça, outro para o espaço da peça, outro para um movimento de peça válido e o outro para o fundo do título. Os nomes das variáveis são ilustrativos.

Variáveis:

```
/// Variáveis ///
screen_state = MENUSCREEN;
int help = 0;
```

Para facilitar a leitura de um código em allegro, o código é separado por funções. Na parte de variáveis, declara-se uma variável do tipo inteiro, para ter controle de qual tabuleiro será jogado, inicializando com 0, e inicializa-se a estado de tela no MENU.

Inicializando as structs:

```
// Global scope //
Paths *paths;
Volume *vol;

// Main scope //
vol = (Volume *) malloc (sizeof (Volume));
path = (Paths *) malloc (sizeof (Paths));
setSettings (vol, paths);
```

Para fácil acesso, ponteiros para structs do tipo “Volume” e “Paths” são criadas globalmente, e então alocados dinamicamente. Após isso, é preciso atribuir valores aos campos dessas structs. Para tornar o jogo mais dinâmico e flexível, deixa-se as informações salvas em algum arquivo, para então recuperar essas informações do arquivo. Cria-se uma função para fazer a leitura e atribuição desses valores.

Função para leitura dos ajustes:

```
void setSettings (Volume* vol, Paths* paths) {
    char c;
    FILE *file = fopen ("files\\settings.txt", "r");
    while (c != EOF) {
        fscanf (file, "%s", paths->boardI);
        fscanf (file, "%s", paths->boardF);
        fscanf (file, "%s", paths->pieceBg);
        fscanf (file, "%s", paths->bg);
        fscanf (file, "%s", paths->buttonB);
        fscanf (file, "%s", paths->logo);
        fscanf (file, "%s", paths->buttonM);
        fscanf (file, "%s", paths->buttonS);
        fscanf (file, "%s", paths->title);
        fscanf (file, "%s", paths->piece);
        fscanf (file, "%s", paths->pieceV);
        fscanf (file, "%d %d", &vol->noise, &vol->music);
        c = fgetc (file);
    }
    fclose (file);
}
```

Nessa função, recebe-se as structs como argumentos. Então, abre o arquivo onde estão salvas as informações, e começa a leitura do arquivo onde as informações estão salvas. Cada um dos caminhos salvos está separado por um espaço, ou seja, utiliza-se o fscanf por irá ler até o espaço. Após ler os caminhos, lê-se as duas últimas informações, que são as de volume. Após leitura, fecha-se o arquivo.

Para exemplificar, uma estrutura possível seria ter uma pasta de “bitmaps” e dentro dela possuir subpastas para melhor controle, e então o arquivo estar lá dentro.

fscanf (file ”%s”, paths->bg) irá ler “bitmaps\bg\bg1.bmp”.

Após carregar as informações do arquivo de texto, retorna-se à função main.

Sounds:

```
MIDI* musica = load_midi ("sounds\\music.mid");
play_midi (musica, TRUE);
set_volume (vol->noise, vol->music);
```

Para a parte de Sounds, carrega-se a música e começa a tocar, com o volume sendo definido pelas informações carregadas do arquivo.

Obs: Sempre que entrar em uma nova tela, é preciso definir o volume novamente usando a função set_volume. Basta chamar ela assim que entrar no loop da nova tela.

Para o loop do game, utiliza-se o um while para ser quebrado ate a variável de escape do jogo se torne verdadeira.

Game loop:

```
while (! exit_program) {}
```

Check de escape:

```
if (key [KEY_ESC])

        fecha_programa ();
```

Então, para todas as telas novamente, precisa-se ter a leitura do teclado na parte de Inputs, e uma checagem para ver se a tecla de escape não foi apertada.

Após isso, entra-se numa estrutura de decisão para ver qual é o estado de tela e qual função precisa ser chamada. Como o estado começa em MENU, é chamada a uma função menu (). Caso a tecla ESC tivesse sido pressionada, o loop se encerraria, precisando para a música do jogo, e desalocando o seu ponteiro.

Finalização:

```
/// FINALIZATION ///

stop_midi ();

destroy_midi (musica);

return 0;
```

Após encerrar a função main do programa, então é preciso colocar a macro do Allegro: “END_OF_MAIN ()” para que o programa possa entender que foi finalizado.

Para as telas é preciso definir algumas constantes. Utiliza-se a função makecol (red, green, blue), que aceita três parâmetros de geração de cores no padrão rgb.

Defines:

```
#define  BRANCO      makecol (255, 255, 255)
#define  CINZA       makecol (196, 196, 192)
#define  PRETO       makecol (0, 0, 0)
#define  PRETIN      makecol (31, 31, 31)
#define  VERDE       makecol (253, 98, 8)
#define  VERDIN      makecol (24, 133, 196)
#define  VERMELHO makecol (253, 38, 41)
```

Menu

Se na função main, o estado de tela for MENU, então é chamada uma função genérica menu (). Assim como para qualquer tela, é executada em partes.

Nessa tela, usa-se os caminhos: paths->bg, paths->board, paths->buttonB, paths->buttonM, paths->logo, paths->title, paths->piece, paths->pieceBg. Como exemplo para um:

Carregando um bitmap:

```
BITMAP* bg = load_bitmap (paths->bg, NULL);
```

Faça isso para todos os outros caminhos da struct paths. Crie também um bitmap auxiliar caso queira implementar um mouse diferente dentro do jogo. É necessário criar um bitmap auxiliar para executar a prática de double buffering. Esse buffer precisa ter o mesmo tamanho da janela do jogo. Para isso, utiliza-se as macros que recuperam os valores de largura e altura.

Criando um buffer:

```
BITMAP* buffer = create_bimap (SCREEN_W, SCREEN_H);
```

Carrega-se as fontes personalizadas e os sons, caso existirem.

Carregando fontes e sons:

```
FONT *fonte = load_font ("CAMINHO\\fonte.pcx", NULL, NULL);

SAMPLE *efeito = load_sample ("CAMINHO\\efeito.wav");
```

Precisa-se de uma variável auxiliar de escape de tela, que funciona da mesma maneira que a variável de fechamento de jogo. Além disso, precisa-se de variáveis para loops, uma para efeito de seleção, e uma variável para visualização do campo de maneira dinâmica.

Variáveis auxiliares:

```
char aux [9] [9] = {"*********", "***OOO***", "***OOO***", "*OOOOOOO*", "*OOOIOOO*",
"*OOOOOOO*", "***OOO***", "***OOO***", "*********"};

    int exit_screen = FALSE;

    int t = 0, sel_i = 0, i, j;
```

Após isso inicia-se o game loop, verifica-se se ESC não é pressionado, ajusta o volume, e então chama-se uma função para conferir os estados das teclas, para que não aconteça bugs: keyboardInput ().

Estado de tecla:

```
void keyboardInput () {
   int i;
   for (i = 0; i < KEY_MAX; i++)
     beforeKey [i] = key [i];
   poll_keyboard ();
}


int pressed (int TECLA) {
    return (beforeKey [TECLA] == FALSE && key [TECLA] == TRUE);
}
```

A primeira confere o estado de tecla, percorrendo todas as entradas de tecla recebida do teclado.

A segunda é para conferir se esse estado é aquele em que a tecla anterior for diferente a posterior, sabendo se o jogador só apertou a tecla.

Então confere-se se apertou as setas para e cima e para baixo, e atualiza o seletor, tocando também o efeito sonoro. Faça isso para ambas os sentidos.

Leitura do seletor:

```
if (pressed (KEY_UP)) {
    sel_i--;
    play_sample (efeito, vol->noise, 0, 1000, FALSE);
}
```

Então, é preciso checar se algum desse botões é selecionado, pois atribui-se o valor do seletor para cada um dos botões. Por exemplo, se o seletor for 0 e for selecionado, ele deverá entrar na aba de regras. Além da checagem do teclado, é preciso chegar se o mouse está dentro de tal região. Para isso, precisa-se saber previamente o tamanho dos sprites dos botões.

Check de botão selecionado:

```
int mouseIn (int xi, int xf, int yi, int yf) {
    int v = 0; if (mouse_x > xi && mouse_x < xf && mouse_y > yi && mouse_y < yf) v = 1; return v;}
if ((sel_i == 0 && pressed (KEY_ENTER)) || (mouseIn (560, 721, 250, 291))
    screen_state = RULE;
```

Faça-se isso para todos os botões, mudando apenas o valor do seletor, os argumentos da função mouseIn e para qual estado de tela irá ser trocado.

Atualização do seletor:

```
if (sel_i > 5) sel_i = 0;
if (sel_i < 0) sel_i = 5;
for (i = 0; i < 6; i++) {
    if (mouseIn (560, 721, 250 + i * 50, 291 + i * 50))
        sel_i = i;
}
```

Passando para a parte de update, é preciso atualizar o seletor quando extrapola os seus limites e quando o mouse entra em contato com algum dos botões.

Saindo da tela:

```
if (screen_state != MENUSCREEN)
    exit_screen = TRUE;
```

Sempre que alterar o estado na tela, precisa-se fechar essa tela, para que assim não fique nenhuma tela aberta ocupando memória.

Entrando então na parte de desenho da função, é precisa desenhar no buffer na ordem contrária na qual irá aparecer na tela, ou seja, se for aparecer por cima, precisa ser a última a ser desenha no buffer. Desenha-se o fundo, a logo e o fundo do campo no buffer. Para desenhar o resto do campo, utiliza-se:

Desenhando o campo no menu:

```
for (i = 0; i < 9; i++) {
    for (j = 0; j < 9; j++) {
        if (aux [i] [j] == 'O' || aux [i] [j] == 'I')
            draw_sprite (buffer, pieceBg, 86 + (j - 1) * 50, 192 + (i - 1) * 50);
    }
}
draw_sprite (buffer, piece, 236, 342);
```

Após isso, desenha-se os botões, o fundo do título, e as escritas na tela. Novamente, precisa-se saber o tamanho dos botões e exatamente onde deseja desenha-los. Para escrever os textos, ousa-se a textout_centre_ex, para escrever com base no centro da posição.

Desenhando os botões:

```
for (i = 0; i < 6; i++)
        draw_sprite (buffer, button, 560, 250 + 50 * i);
textout_centre_ex (buffer, fonte, "Menu Principal", 640, 150, PRETO, -1);
```

E para adicionar o efeito do seletor piscando, basta ajustar o timer auxiliar, e desenha na metade desse tempo, algo que cubra o botão, da mesma cor. Então, desenha-se o cursor no buffer e por último, desenha-se o buffer na tela, incrementado a variável t.

Efeito de piscar:

```
if (t > 100) t = 0;
if (t < 50)
    rectfill (buffer, 565, 253 + sel_i * 50, 715, 287 + sel_i * 50, cor_state);
if (mouseIn (560, 721, 250 + 50 * sel_i, 291 + 50 * sel_i))
    rectfill (buffer, 565, 253 + 50 * sel_i), 715, 287 + 50 * sel_i), PRETO);
draw_sprite (buffer, cursor, mouse_x, mouse_y);
draw_sprite (screen, buffer, 0, 0);
t++;
```

Assim chega-se ao final do loop, e ele ficará repetindo-se até ser selecionado alguma das opções ou o jogo ser fechado. Caso isso aconteça, então o loop termina, destrói-se todos os bitmaps, sons e fontes importados, e chega-se ao fim da função.

Destruindo ponteiros:

```
destroy_bitmap (buffer);
destroy_font (fonte);
destroy_sample (efeito);
```

A maioria dessas funções e funcionalidades irá repetir-se nas outras funções, basta aplica-las conforme for a necessidade.

Regras

Caso no menu seja escolhido o botão de regras, muda-se o estado de tela e chama-se uma função genérica para leitura das regras. Dentro da função apenas precisa dos bitmaps um buffer, outro buffer auxiliar, cursor, botão simples e de fundo. Além disso importa-se também as fontes e músicas personalizadas. Após inicializar tudo corretamente, basta fazer a leitura do arquivo onde as regras estão salvas, e imprimir no buffer auxiliar. Nesse trecho, o arquivo será lido letra a letra e será impresso no buffer com um espaçamento especifico. Se chegar ao final da linha, basta reiniciar o espaçamento. O arquivo em que as regras se encontram precisa estar formatado corretamente.

Leitura das regras:

```
int aux = 0, no = 0; char c;
FILE* file = fopen ("CAMINHO\\regras.txt", "r");
draw_sprite (bufferaux, bg, 0, 0);
while (c != EOF) {
    if (c == '\n') {no++; aux=0;}
    aux++;
    if (c != '\n') textprintf_centre_ex (bufferaux, font, 11.5 * aux, no * 20 + 10, PRETO, -1, "%c", c);
    c = fgetc (file);
}
fclose (file);
```

Botão de voltar pressionado:

```
if (pressed (KEY_ENTER) || (mouseIn (349, 450, 550, 571)) {
screen_state = MENUSCREEN; exit_screen = TRUE;
}
```

Após isso entra-se no game loop, verifica-se se a tecla ESC é pressionada, ou se o botão de voltar é pressionado.

Por último, desenha-se o buffer auxiliar no buffer, desenha-se o botão e implementa-se os efeitos de botão selecionado. E por final, os bitmaps, fontes e sons são destruídos.

Desenhando o botão de voltar:

```
draw_sprite (buffer, bufferaux, 0, 0);
draw_sprite (buffer, button, 349, 550);
```

Histórico

Assim como as outras telas, precisa-se de bitmaps para o fundo, cursor se necessário, botão, um buffer e outros dois buffers auxiliares. Importa-se também os sons e a fontes personalizadas.

Primeiramente, desenha-se o fundo nos dois buffers auxiliares, e inicia-se a leitura do arquivo texto. Durante a leitura, se o tamanho da tela for 800x600, com esta função, apenas será possível ler e imprimir 20 linhas do arquivo de histórico. Por isso, cria-se dois buffers (se desejar, pode-se criar mais conforme necessidade) para separar a leitura desses dados. Em cada um, imprime-se as letras uma a uma, com um certo espaçamento, tanto entre linhas, como entre elas mesmas.

Leitura do histórico:

```
int aux = 0, no = 0; char c;
FILE* file = fopen ("CAMINHO\\historico.txt", "r");
while (c != EOF) {
    if (c == '\n') {no++; aux = 0;}
    aux++;
    if (c != '\n' && no < 20)
        textprintf_centre_ex (bufferaux1, fonte, aux * 8.5, no * 20 + 50, PRETO, -1, "%c", c);
    if (c != '\n' && (no > 19 && no < 40))
        textprintf_centre_ex (bufferaux2, fonte, aux * 8.5, (no - 20) * 20 + 50, PRETO, -1, "%c", c);
    if (no >= 40)
        textout_centre_ex (bufferaux2, fonte, "CHEIO! Limpe-o.", 400, 500, PRETO, -1);
    c = fgetc (file);
} fclose (file);
```

Troca de buffer:

```
int sel_j = 1, controle = 0;
if ((pressed (KEY_ENTER) && sel_j == 2) || mouseIn (570, 671, 550, 571)) {
    controle--; play_sample (ef2, vol->noise, 0, 1000, FALSE);}
if ((pressed (KEY_ENTER) && sel_j == 3) || mouseIn (680, 781, 550, 571)) {
    controle++; play_sample (ef2, vol->noise, 0, 1000, FALSE);}
```

Então, entra-se no game loop. Checa se a tecla ESC foi pressionada, ou se algum outro botão foi selecionado. O botão de voltar é o mesmo botão aplicado na tela de REGRAS. Existirá, além do botão de retornar ao menu, 3 outros botões. Dois deles são para troca de buffer que será exibido, mostrando informações diferentes. Para isso, precisa-se de uma variável de controle para saber qual é o buffer a ser desenhado na tela naquele instante.

Limpa histórico:

```
if (pressed (KEY_ENTER)) && sel_j == 0) {
    FILE* file = fopen ("CAMINHO\\historico.txt", "w+");
    exit_screen = TRUE;
}
```

Também se atualiza o sel_j usualmente forme apertado as setas do teclado. E para o botão de limpar o arquivo de histórico, basta abrir o histórico e apagar tudo que tem dentro e então atualiza-se a tela.

Na parte de update, precisa-se atualizar o sel_j conforme o extrapola o limite, e o controle a mesma coisa (exemplo: if (controle > 1) controle = 0).

Então, na parte de desenho, desenha-se o buffer auxiliar, referente a variável de controle, no buffer principal. Desenha-se todas as opções e botões e textos na tela. Aplica-se o efeito de seleção, desenha-se o cursor na tela e por último, desenha-se o buffer na tela.

Controle de buffer:

```
if (controle == 0) draw_sprite (buffer, bufferaux1, 0, 0);
if (controle == 1) draw_sprite (buffer, bufferaux2, 0, 0);
```

Por último, finaliza-se a função destruindo todos os bitmaps, fontes e sons importados.

Records

Se no menu escolhe-se o botão de records, é trocado para a tela de records, chamando-se uma função genérica record ().

Basicamente, essa função segue o mesmo raciocínio da função de histórico. Difere apenas em não ter o botão de limpar, e na parte de leitura do arquivo. No arquivo de records, utiliza-se o $ para separar os dados, então basta ignorar ele na hora da leitura.

Leitura dos records:

```
int aux = 0, no = 0; char c;
FILE* file = fopen ("CAMINHO\\record.txt", "r");
while (c != EOF) {
    if (c == '\n') {no++; aux = 0;}
    aux++;
    if (c != '\n' && no < 22 && c != '$')
        textprintf_centre_ex (bufferaux1, fonte, aux * 8.5, no * 20 + 50, PRETO, -1, "%c", c);
    if (c != '\n' && (no > 21 && no < 44) && c != '$')
        textprintf_centre_ex (bufferaux2, fonte, aux * 8.5, (no - 22) * 20 + 50, PRETO, -1, "%c", c);
    c = fgetc (file);
} fclose (file);
```

O resto do código executa-se da mesma maneira que a função de histórico. Só não é necessário adicionar o botão para limpar o arquivo.

Ajustes

Caso no menu seja escolhido o botão de ajustes, chama-se uma função genérica de settings (). Para essa tela, precisa de um bitmap para os botões, para o fundo do título, para o fundo, cursor e o buffer auxiliar. Além disso, importa-se as fontes e sons personalizados. Como nas funções de histórico e record, o botão de voltar funciona da mesma maneira. Declara-se as variáveis para seletores e auxiliares dos loops.

No game loop, checa-se se ESC foi pressionado. Para os botões de volume, precisa-se de uma função para checar se uma tecla está sendo pressionada continuamente, atualizar o volume (apenas assume valores entre 0 e 255). Além de configurar pelo teclado, também chega se o botão está sendo pressionado. Precisa-se ajustar o valor conforme posição do botão na tela.

Configurando volume:

```
int hold (int TECLA) {return (beforeKey [TECLA] == TRUE && key [TECLA] == TRUE);}

if ((pressed (KEY_N) || hold (KEY_N)) && vol->noise > 0) vol->noise--;

if ((pressed (KEY_M) || hold (KEY_M)) && vol->noise < 255) vol->noise++;

if ((pressed (KEY_J) || hold (KEY_J)) && vol->music > 0) vol->music--;

if ((pressed (KEY_K) || hold (KEY_K)) && vol->music < 255) vol->music++;

if (mouseIn (95, 105, 185, 440) && mouse_b == 1) vol->noise = mouse_y - 185;

if (mouseIn (246, 256, 185, 440) && mouse_b == 1) vol->music = mouse_y - 185;
```

Resete de configurações:

```
if ((pressed (KEY_R)) || mouseIn (12, 103, 550, 571)) {
    vol->noise = 50;
    vol->music = 50;
    strcpy (paths->bg, "CAMINHO\\background1.bmp");
    (...)
}
```

Atualiza-se os seletores conforme teclas pressionadas. Se o botão de reset for pressionado, então atribui-se os valores padrão para cada uma das configurações da struct de caminhos e da struct de volume. Obs: faça isso para todos os outros caminhos também.

A quantidade de botões para alterar as configurações é arbitrária, assim como as configurações que serão atribuídas para esses botões. Para cada botão, basta apenas alterar o valor dos seletores e a posição do mouse que ele se encontra.

Alterando configuração:

```
if (((sel_i == 0 && sel_j == 1) && (pressed (KEY_ENTER)) || mouseIn (521, 543, 151, 173))) {
    strcpy (paths->buttonS, "CAMINHO\\small_button2.bmp");
    exit_screen = TRUE;
}
```

Desenhando botões de som:

```
rectfill (buffer, 99, 185, 101, 440, PRETO);
rectfill (buffer, 249, 185, 251, 440, PRETO);
rect (buffer, 95, 181 + vol->noise, 105, 189 + vol->noise, PRETO);
rect (buffer, 245, 181 + vol->music, 255, 189 + vol->music, PRETO);
rectfill (buffer, 96, 182 + vol->noise, 104, 188 + vol->noise, PRETO);
rectfill (buffer, 246, 182 + vol->music, 254, 188 + vol->music, PRETO);
```

Na parte de UPDATE, basta atualizar os seletores conforme extrapolem seus valores. Na parte de DRAW, basta desenhar o fundo no buffer, e logo após os botões e os textos.

Desenhando botões de configurações:

```
for (i = 0; i < 5; i++) {
    for (j = 0; j < 5; j++) {
        rect (buffer, 470 + 50 * i, 150 + 70 * j, 494 + 50 * i, 174 + 70 * j, PRETO);
        rectfill (buffer, 471 + 50 * i, 151 + 70 * j, 493 + 50 * i, 173 + 70 * j, cor_state);
        textprintf_centre_ex (buffer, fonte, 482 + 50 * j, 153 + 70 * i, PRETO, -1, "%d", j + 1);
    }
}
```

Após desenhar todos os botões, basta aplicar o efeito de selecionar.

No final, basta destruir todos os bitmaps, sons e fontes que foram utilizados.

Tabuleiros

Caso no menu principal seja escolhido a opção jogar, chama-se uma função genérica para abrir a tela de tabuleiros disponíveis. Como no menu, precisa-se imprimir um campo ilustrativo ao lado, ou seja, precisa-se dos bitmaps de fundo, campo, espaço da peça, peça, para o botão, título e um buffer auxiliar. Após importar todos esses bitmaps, fontes e sons personalizados. Basicamente, a função para o menu e o menu de tabuleiros é a mesma. Diferencia-se apenas no tipo de botão a ser desenhado e sua quantidade. Basta fazer tudo como na função menu, adicionando apenas um seletor horizontal.

Struct campo:

```
typedef struct {
    int points;
    int piece;
    char result [7];
    char player [21];
    char name [15];
    char board [9] [9];
} Board;
```

A partir daqui, será preciso um struct para salvar as informações do campo. Nela, precisa-se de um número para contagem de movimentos feitos, o número de peças, uma string para o resultado do jogo, string para o nome do jogador, string para o nome do tabuleiro, e uma matriz para o tabuleiro. Para iniciar, declara-se um ponteiro para struct do tipo Board.

Os 42 primeiros botões servem para iniciar o jogo em um tabuleiro. Caso seja selecionado o botão de número 43, ele retornará ao menu. Caso seja selecionado um tabuleiro, usa-se uma variável auxiliar para verificar o número do botão selecionado, utilizando os seletores para obter tal informação, então aloca espaço para o tabuleiro e muda o estado de tela.

Declarando tabuleiro:

```
h = sel_i + 15 * sel_j;
if (pressed (KEY_ENTER) || mouseIn (450 + 110 * sel_j, 551 + 110 * sel_j, 150 + 30 * sel_i, 171 + 30 * sel_i)) {
bd = (Board *) malloc (sizeof (Board));
    if (h >= 0 && h < 31) screen_state = GAMEI;
    else if (h > 30 && h < 42) screen_state = GAMEF;
    else if (h == 42) {free (bd); screen_state = MENUSCREEN; exit_screen = TRUE;}
}
```

Iniciando tabuleiro:

```
void setBoard (    * h) {
    bd->points = 0; bd->piece = 0;
    char ch, b [82]; int i, aux = 0;
    FILE *file = fopen ("board.txt", "r");
    while (ch != EOF) {
        if (aux == *h && ch != '\n') fscanf (file,"%s", b);
        if (aux == *h + 42 && ch != '\n') { fgets (bd->name, 30, file); break;}
        if (ch == '\n') aux++;
        ch = getc (file);
    }
    for (i = 0; i < strlen (bd->name); i++) {
        if (bd->name [i] == '\n') bd->name [i] = '\0';
    }
    for (i = 0; i < strlen (b); i++) {if (b [i] =='I') bd->piece++;} b [81] = '\0';
    memcpy (bd->board, b, sizeof (bd->board));
    fclose (file);
}
```

Na parte de UPDATE, basta atualizar os valores dos seletores conforme as setas são pressionadas ou o mouse é movimentado sobre os botões.

Na parte de DRAW, basta desenhar primeiro o fundo no buffer, então desenhar o campo como na função menu, e então os botões.

Desenhando os botões de tabuleiro:

```
for (i = 0; i < 15; i++) {
    for (j = 0; j < 2; j++)
        draw_sprite (buffer, button, 450 + 110 * j, 150 + 30 * i);
}
for (i = 0; i < 13; i++)
    draw_sprite (buffer, button, 670, 150 + 30 * i);
```

Após desenhar os botões, basta adicionar o efeito de seleção, desenhar o cursor na tela, e por último, desenhar o buffer na tela principal.

Finalizando a função, destrói-se os bitmaps, fontes e sons importado

Game

Caso escolha-se um tabuleiro, então é iniciada uma função genérica game, que recebe como parâmetro 0 ou 1 para saber se é um tabuleiro do tipo inglês ou francês. Entrando na função, inicia-se todos os bitmaps para o campo, efeitos sonoros e fontes personalizadas. Antes de entrar no loop, inicia-se o timer para gravar o tempo:

Início do timer:

```
timer = 0; install_int_ex (incrementa_timer, SECS_TO_TIMER (1));
```

Entrando no loop, verifica-se fechamento de jogo. Para percorrer as peças do jogo, utiliza-se dois seletores, que são lidos conforme pressionar das setas do teclado ou quando o mouse passa encima de uma peça.

Caso o jogador pressione R ou selecione o botão de retornar jogada, será chamada uma função para ler de um arquivo, o tabuleiro da jogada anterior. Para poder retornar jogada, deve ter acontecido pelo menos uma jogada.

Retorna jogada:

```
if ((bd->points > 0) && (pressed (KEY_R) || mouseIn (625, 701, 342, 401)) returnMove (bd);

void returnMove (Board *bd) {
    char ch, b [82]; int i;
    bd->piece = 0;
    FILE *file = fopen ("returnboard.txt", "r");
    while (ch != EOF) {fscanf (file, "%s", b); ch = getc (file);}
    for (i = 0; i < 82; i++) {if (b [i] == 'I') bd->piece++;}
    memcpy (bd->board, b, sizeof (bd->board));
    fclose (file);
}
```

Caso o jogador pressione V ou selecione o botão de sair, libera-se o espaço alocado pela struct tabuleiro, e retorna para a tela de menu.

Retorna ao menu:

```
if (pressed (KEY_V) || mouseIn (625, 701, 420, 476)) {
    free (bd); screen_state = MENUSCREEN; exit_screen = TRUE;
}
```

Caso deseja-se movimentar uma peça, é preciso verificar se onde o seletor está é uma peça válida para movimento. Caso for, troca-se a tela para a tela de execução de movimento (Que será explicado mais a frente).

Selecionando peça:

```
if (bd->board [sel_i + 1] [sel_j + 1] == 'I') {

if (pressed (KEY_ENTER) || mouseIn (225 + sel_j * 50, 276 + sel_j * 50, 125 + sel_i * 50, 176 + sel_i * 50)

    screen_state = MOVE;}
```

Na parte de update, precisa-se atualizar os seletores conforme percorre o tabuleiro. Após retornar da função que executa a jogada, precisa-se atualizar o seletor em 2 unidades, para ajustar conforme movimento executado (Exemplo: if (pressed (KEY_UP)) sel_i = sel_i – 2).

Chama-se uma função para verificar se o jogo tem continuidade ou não. Caso retorne 1, o jogo não tem mais continuidade.

Game over:

```
int GameOver () {

    int i, j, v = 1;

    for (i = 1; i < 9; i++) {

        for (j = 1; j < 9; j++) {

            if ((bd->board [i] [j] != '*') && (bd->board [i] [j] == 'I') &&

                ((bd->board [i + 1] [j] == 'I' && bd->board [i + 2] [j] != '*' && bd->board [i + 2] [j] != 'I') ||

                 (bd->board [i] [j + 1] == 'I' && bd->board [i] [j + 2] != '*' && bd->board [i] [j + 2] != 'I') ||

                 (bd->board [i - 1] [j] == 'I' && bd->board [i - 2] [j]  != '*' && bd->board [i - 2] [j]  != 'I') ||

                 (bd->board [i] [j - 1] == 'I' && bd->board [i] [j - 2] != '*' && bd->board [i] [j - 2] != 'I')))

                    v = 0;

        }

    }

    return v;

}
```

E então verifica-se se o jogador venceu ou perdeu o jogo. Ela irá retornar 1 caso tenha vencido, e zero caso tenha perdido.

Check game:

```
int checkGame () {int i, j, v = 0;

    for (i = 1; i < 9; i++) {for (j = 1; j < 9; j++) {if (bd->board [i] [j] == 'I') v++;}} if (v > 1) v = 0; return v;}
```

Para desenhar o tabuleiro, basta fazer como na função menu.

Desenhando tabuleiro:

```
draw_sprite (buffer, buttonBig, 625, 345);

draw_sprite (buffer, buttonBig, 625, 420);

    int i, j;

    draw_sprite (buffer, board, 225, 125);

    for (i = 0; i < 9; i++) {

        for (j = 0; j < 9; j++) {

        if (bd->board [i] [j] == 'O' || bd->board [i] [j] == 'I')

            draw_sprite (buffer, pieceBg, 225 + (j - 1) * 50, 125 + (i - 1) * 50);

        if (bd->board [i] [j] == 'I')

            draw_sprite_ex (buffer, piece, 225 + (j - 1) * 50, 125 + (i - 1) * 50,
DRAW_SPRITE_NORMAL, DRAW_SPRITE_NO_FLIP);

        }

    }
```

Após desenhar o tabuleiro, então, aplica-se o efeito de seletor e destrói-se os bitmaps, sons e fontes personalizados.

Jogada

Caso na função game seja selecionada uma peça válida, então muda-se a tela para a tela de JOGADA. Nela, repete-se tudo da parte de desenho da tela de GAME. Em adição, precisa-se de um bitmap para desenhar as opções de jogada possíveis na tela. Os botões de retornar a jogada e retornar ao menu continuam ativos. Primeiramente, verifica-se se a peça selecionada tem algum movimento válido na direção que foi sugerida para movimentar-se.

Definindo a direção:

```
if ((pressed (KEY_RIGHT) ||
(mouseIn (225 + (sel_j + 2) * 50, 276 + (sel_j + 2) * 50, 125 + sel_i * 50, 176 + sel_i * 50) &&
 mouse_b == 1)) && bd->board [sel_i + 1] [sel_j + 1] == 'I' && bd->board [sel_i + 1] [sel_j + 2] == 'I' &&
 bd->board [sel_i + 1] [sel_j + 3] == 'O')
        d = 'R';
```

Faça isso para cada um das direções, atribuindo uma letra especifica a variável de controle de direção d, conforme as seguintes condições: a matriz do campo nas posições da onde o seletor está precisa ser um pino; a posição mais próxima dentro do campo também precisa ser um pino; a posição onde deseja-se chegar precisa ser um espaço vazio.

Após fazer isso para todas as direções, salva-se a jogada anterior, atualiza-se o número de jogadas, o número de pinos e o tabuleiro conforme a jogada lida do teclado.

Salva jogada:

```
FILE *file = fopen ("returnboard.txt", "w+"); fprintf (file, " "); int i, j;
for (i = 0; i < 9; i++) {
   for (j = 0; j < 9; j++) {
      fprintf (file, "%c", bd->board [i] [j]);}
fprintf (file, "\n"); fclose (file);
```

Atualiza tabuleiro:

```
bd->points++; bd->piece--;
switch (d) {
case 'D':
   bd->board [sel_i + 1] [sel_j + 1] = 'O'; bd->board [sel_i + 2] [sel_j + 1] = 'O';
   bd->board [sel_i + 1] [sel_j + 3] = 'I'; break;
(…) }
```

Faça o mesmo para todas as outras direções.

Player

Após terminar o jogo, entra-se na tela NAME. Nela, é onde o jogador irá digitar o seu nome. Para esta tela, precisa-se de bitmaps para o fundo, para logo ilustrativa, para um botão, o cursor e o buffer auxiliar. Além disso importa-se os sons e fontes personalizados. Implementa-se um botão de retornar ao menu, igual ao das telas de REGRAS, HISTÓRICO E RECODS.

Limpando buffer do teclado:

```
int j, ch, index = 0;
char *read = (char *) calloc (21, sizeof (char));
clear_keybuf ();
```

Para esta função, são necessárias algumas variáveis auxiliares na parte da leitura do nome. E para poder ler corretamente as teclas do teclado, limpa-se o buffer de entrada de dados do teclado antes de se entrar no loop.

Lendo nome:

```
while (keypressed ()) {
   ch = readkey ();
   if ((ch >> 8) == KEY_BACKSPACE) {
      if (index > 0) {
         read [index - 1] = '\0';
         index--;
      }
   } else {
      if (index < 20) {
         if ((ch & 0xff) >= 32 && (ch & 0xff) <= 126)
            read [index++] = (ch & 0xff);}}}
```

No game loop, após verificar escape de tela com a tecla ESC, caso seja apertada alguma tecla que não seja ESC, entra-se em outro loop para a leitura do nome do teclado. Nela, o caractere irá receber a tecla pressionada, e irá checar se não foi apertado a tecla para apagar algum caractere. Caso seja e o número de letras lidas seja maior que zero, será apagado digito já lido. Ao contrário, enquanto não chegar ao limite, aceitará os caracteres lidos. Utiliza-se os valores hexadecimais de cada tecla para poder ler caracteres especiais.

Check de Record:

```
int aux = 0, n, n2, n3, naux; char c, re [7];
FILE *file = fopen ("ranking.txt", "r");
while (ch != EOF) {
   if (aux == help && c == '$') fscanf (file, "%s %d %d:%d", re, &n, &n2, &n3);
   if (ch == '\n') aux++;
   ch = getc (file);} fclose (file); naux = 60 * n2 + n3;
if (! strcmp (re, bd->result) && n >= bd->points && naux >= timer)
```

Então, caso seja pressionado selecionado o botão de retornar ao menu, verifica-se se o jogador bateu o record.

Apagando o record:

```
char c; int cont = 0;
FILE *file1 = fopen ("ranking.txt", "r")
FILE *file2 = fopen ("rankinga1.txt", "w");
FILE *file3 = fopen ("rankinga2.txt", "w");
while (c != EOF) {
    if (c == '\n') cont++;
    if (cont <= help && cont != help) putc (c, file2);
    if (cont > help) putc (c, file3);
    c = getc (file1);
}
fclose (file1); fclose (file2); fclose (file3);
```

Caso o jogador tenha batido o record, então, separa-se o arquivo texto onde estão salvos os records em dois, para poder apagar o record que já estava salvo. Para saber qual record apagar, utiliza-se o parâmetro de escolha do tabuleiro, para apagar a linha do arquivo correspondente a ele. Para apagar, basta colocar o que tem antes de linha em um arquivo, e o que tem depois em outro arquivo.

Depois, basta escrever as informações do jogo no primeiro arquivo.

Salvando record:

```
FILE *file = fopen (“rankinga1.txt", "a+");
fprintf (file, "\n%-20s %-30s $%-18s%-11d", bd->player, bd->name, bd->result, bd->points);
fprintf (file, "%d:%02d", (timer / 60) % 60, timer % 60);
fclose (file);
```

Depois, concatena-se os dois arquivos texto auxiliar. Abre-se o segundo arquivo auxiliar para leitura e o primeiro para adição.

Concatenando record:

```
FILE *file1 = fopen (“rankinga2.txt”, "r"), *file2 = fopen (“rankinga1.txt”, “a”);
int n = 0;
while (c != EOF) {n++; c = fgetc(file1);}
rewind (file1);
char *read = (char *) calloc (n, sizeof (char));
while ((fgets (read, n, file1)) != NULL)
    fputs (read, file2);
free (read); fclose (file1); fclose (file2);
```

Depois, basta executar o mesmo trecho de código, porém escrevendo tudo no arquivo de record principal.

Retornando a tela de NOME, na parte de DRAW basta fazer algo parecido com isso, apenas alterando as posições conforme gosto.

Desenhando tela de nome completa:

```
draw_sprite (buffer, bg, 0, 0);
draw_sprite (buffer, logo, 235, 50);
textprintf_centre_ex (buffer, fonte, 400, 200, PRETO, -1, "Voce %s!!!", bd->result);
textout_centre_ex (buffer, fonte, "Digite seu nome:", 400, 300, PRETO, -1);
draw_sprite (buffer, button, 349, 550);
textout_centre_ex (buffer, fonte, "ENTER", 400, 550, PRETO, -1);
for (j = 0; j < 20; j++)
    rectfill (buffer, 200 + 20 * j, 385, 210 + 20 * j, 387, PRETO);
for (j = 0; read [j] != '\0'; j++)
    textprintf_centre_ex (buffer, fonte, 202 + 20 * j, 350, PRETO, -1, "%c", read [j]);
if (t > 100) t = 0;
if (t < 50) {
    rectfill (buffer, 354, 553, 443, 565, cor_state);
    rectfill (buffer, 195 + 20 * index, 360, 196 + 20 * index, 380, cor_state);
}
if (mouseIn (349, 450, 550, 571))
    rectfill (buffer, 354, 553, 443, 565, cor_state);
textprintf_centre_ex (buffer, fonte, 400, 450, PRETO, -1, "Caracteres restantes: %d", 20 - index);
draw_sprite (buffer, cursor, mouse_x, mouse_y);
draw_sprite (screen, buffer, 0, 0);
t++;
```

E por último, ao fim do game loop, destrói-se todas os bitmaps, sons e fontes personalizados.

Considerações Pessoais

Desde o início não foi um trabalho fácil, muitas das vezes se tornando complexo, e de difícil entendimento. Vários erros foram acumulados e para administração e exclusão deles foi necessário muito tempo investido. Aos poucos, erros de leitura de caractere, erros de retorno em funções do tipo void, ou até erros de passagem por parâmetros. O domínio de funções foi elevado com sucesso. Funções com e sem retorno, funções com e sem parâmetros, recursividade. Tudo nesse quesito foi muito melhorado no desenvolvimento do simulador. Todos os erros foram completamente removidos, resultando em programa que compila sem nenhum warning da IDE.

Aprender a concatenar arrays de caractere, vulgo strings, comparar e usufruir de várias funções da biblioteca string.h, foi um dos resultados ao desenvolver esse simulador. Em linguagem C, a utilização de strings é um pouco limitada, tornando o trabalho com elas um pouco árduo e dificultoso. Mas com a boa prática a abstração e lógica de strings foram muito bem aperfeiçoadas.

Outra melhoria no aprendizado foi sobre arquivos texto. Abertura para leitura, confirmação de existência do arquivo, recuperação e gravação de dados, retorno de ponteiros para arquivos. Foram vários os conhecimentos adquiridos e bem trabalhos sobre o assunto. Aprender a abrir e fechar o arquivo no momento correto, manuseá-lo da maneira correta, foram habilidades adquiridas na hora de comparação dos records. Separar um arquivo em dois, gravar novas informações, concatenar e então substituir no principal, não foi fácil.

Aprender também a importar bibliotecas externas para o programa, mostrou-se um trabalho um pouco complicado. Utilização de funções não pertencentes a bibliotecas padrão C, para quem ainda não é acostumado, pode-se tornar difícil. Como a Allegro possuí uma vasta documentação, explicando todas as funções, variáveis a macros disponíveis para uso, realizar a leitura e executar passo a passo torna a tarefa muito mais fácil. Importar arquivos que sejam diferentes de “.txt” para dentro do código mostrou-se possível.

Além do projeto ambientar e familiarizar mais com a linguagem de programação junto de bibliotecas gráficas, a melhoria em ter que correr atrás por conta própria das resoluções dos erros, implementação de funções por maneira correta, ou até o uso de funções que até então eram desconhecidas. Tudo isso ajudou para o crescimento profissional, no sentido de correr atrás das respostas sozinho. Afinal, um engenheiro precisa detectar um problema e apontar uma solução para ele, tendo ajuda ou não.

Referências

CodeBlocks – Compilador. Disponível em: http://www.codeblocks.org Acesso em: 10 de Julho de 2019.

Download CodeBlocks(13.12). Disponivel em: https://sourceforge.net/projects/codeblocks/files/Binaries/13.12/Windows/ Acesso em: 10 de Fevereiro de 2020.

Download GraphicsGale. Disponível em: https://graphicsgale.com/us/ Acesso em: 11 de Fevereiro de 2020.

Download TTF_PCX_Font_Converter. Disponível em: https://www.softpedia.com/get/Others/Font-Utils/TTF-to-PCX-Converter.shtml Acesso em: 15 de Fevereiro de 2020.

AKSTEIN e HEINE, Liliana e Evelyn. Divertudo – Resta Um. Disponível em: https://www.divertudo.com.br/restaum/restaum.html Acesso em: 10 de Novembro de 2019.

BACKES, André. Linguagem C Descomplicada. Disponível em: https://www.youtube.com/watch?v=GiCt0Cwcp-U&ab_channel=LinguagemCPrograma%C3%A7%C3%A3oDescomplicada Acesso em: 5 de Julho de 2019.

BACKES, André. Linguagem C – Completa e Descomplicada. 2.ed, GEN LTC, 2018.

SCHILDT, Herbert. C Completo e Total. 3.ed, Makron Books, 1996.

Biblioteca Allegro. Disponível em: https://www.allegro.cc/ Acesso em: 12 de Fevereiro de 2020.

Documentação Allegro. Disponível em: https://liballeg.org/a5docs/trunk/ Acesso em: 12 de Fevereiro de 2020.