

MENTES BRILHANTES

SUPER TRUNFO



Manual de regras e detalhes do jogo

SUMÁRIO

1 - Regras do Jogo	3
2 - Tema	5
3 - Modos de Jogo	5
4 - Gerenciador de Cartas	12
5 - Detalhes da Programação.....	14
6 - Detalhes de compilação e execução	31
7 - Observações	33
8 - Créditos	34

REGRAS DO JOGO

O estilo de jogo Super Trunfo é um dos jogos de cartas mais clássicos e fáceis de jogar de todos, pois apenas necessita de um baralho de cartas de um determinado tema e de dois ou mais jogadores. Ele envolve estratégia, conhecimento e sorte. O jogo consiste em um baralho de cartas temáticas (como carros, aviões, animais, ou super-heróis), em que cada carta possui atributos numéricos específicos, como velocidade, força, peso, ou potência.

O objetivo do jogo é conquistar todas as cartas dos adversários, comparando os atributos e vencendo as rodadas com base no valor superior de um atributo escolhido. As regras detalhadas são as seguintes:

1. Preparação do jogo

O baralho é embaralhado e dividido igualmente entre os jogadores. Cada jogador deverá manter sua pilha de cartas viradas para baixo sem olhar as cartas nela.

2. Início do jogo

Um jogador deverá ser escolhido para jogar, ele olha a primeira carta de seu monte (sem mostrar ainda aos seus adversários) e escolhe um dos atributos para competir.

3. Comparação dos atributos

Todos os jogadores devem revelar a carta do topo de seus montes, então deverão comparar os valores do atributo escolhido. A carta com o maior valor vencerá a jogada e o respectivo jogador coleta todas as cartas da rodada colocando-as no final de seu monte.

4. Caso de empate

Se houver um empate no atributo escolhido, as cartas da rodada devem ser colocadas num monte separado. O próximo jogador a vencer uma rodada ganha todas as cartas acumuladas no centro, adicionalmente as cartas da própria rodada.

5. O Super Trunfo

No baralho geralmente há uma carta especial, chamada de Super Trunfo, essa carta tem uma regra específica para vencer. Essas condições variam do baralho, mas para fins gerais, essa carta ganhará de qualquer carta do baralho, excluindo-se as cartas 'A', essas ganharão automaticamente do Super Trunfo.

6. Passagem da Vez

Acabando uma rodada, o jogador vencedor da rodada escolherá o atributo da próxima rodada após olhar a próxima carta de cima do monte.

7. Eliminação e Vitória

Um jogador é eliminado do jogo após perder todas as suas cartas, e o jogo continuará até que reste apenas um jogador com todas as cartas do baralho, sendo assim o ganhador.

Essas são as regras básicas do popular jogo, porém, de acordo com o baralho, os tipos de atributo, a quantidade de Super Trunfos no baralho e outras características, essas regras podem ser diferentes das apresentadas, porém elas sempre seguem essa mesma base de construção e simplicidade, em nossa versão de jogo digital, as regras que foram apresentadas são as que valem para ambos os modos de jogo.

TEMA

Os clássicos Super Trunfos de cartas são dos mais variados temas possíveis, indo de carros clássicos até espécies de animais. O que limita a criação de um novo tema é a imaginação, sendo assim as possibilidades de novos temas são quase infinitas, o que dá a gama para pensar em coisas muito abstratas ou complexas.

Para destoar da maioria dos temas já existentes, escolhemos para nosso jogo, o tema de **MENTES BRILHANTES**, englobando nesse caso todas as pessoas que colaboraram significativamente para o avanço da sociedade, principalmente cientistas e filósofos. Embora não seja um tema habitual pois abre possibilidade para subtemas dentro do principal, dá a chance de infinitas possibilidades de cartas, e acaba sendo usual jogar com ele, afinal as características são bem definidas e os personagens conhecidos.

MODOS DE JOGO

Em geral, o Super Trunfo físico é jogado por grupos de 2 ou mais jogadores, mas pensando na praticidade tanto de quem está jogando, quanto nas questões da própria construção do jogo, decidimos limitar o jogo para no máximo jogos de duas pessoas. Mas para dar uma maior possibilidade para quem joga, além de jogar com outra pessoa, há também possibilidade de jogar contra um bot, assim se tornando um jogo singleplayer.

SINGLEPLAYER:

Nesse modo de jogo o oponente do jogador será um robô (bot), o qual possui uma **pseudo IA** para as escolhas realizadas por ele baseado na dificuldade que o jogador escolher quando o jogo começa. O jogador poderá escolher entre três níveis de dificuldade para o bot.

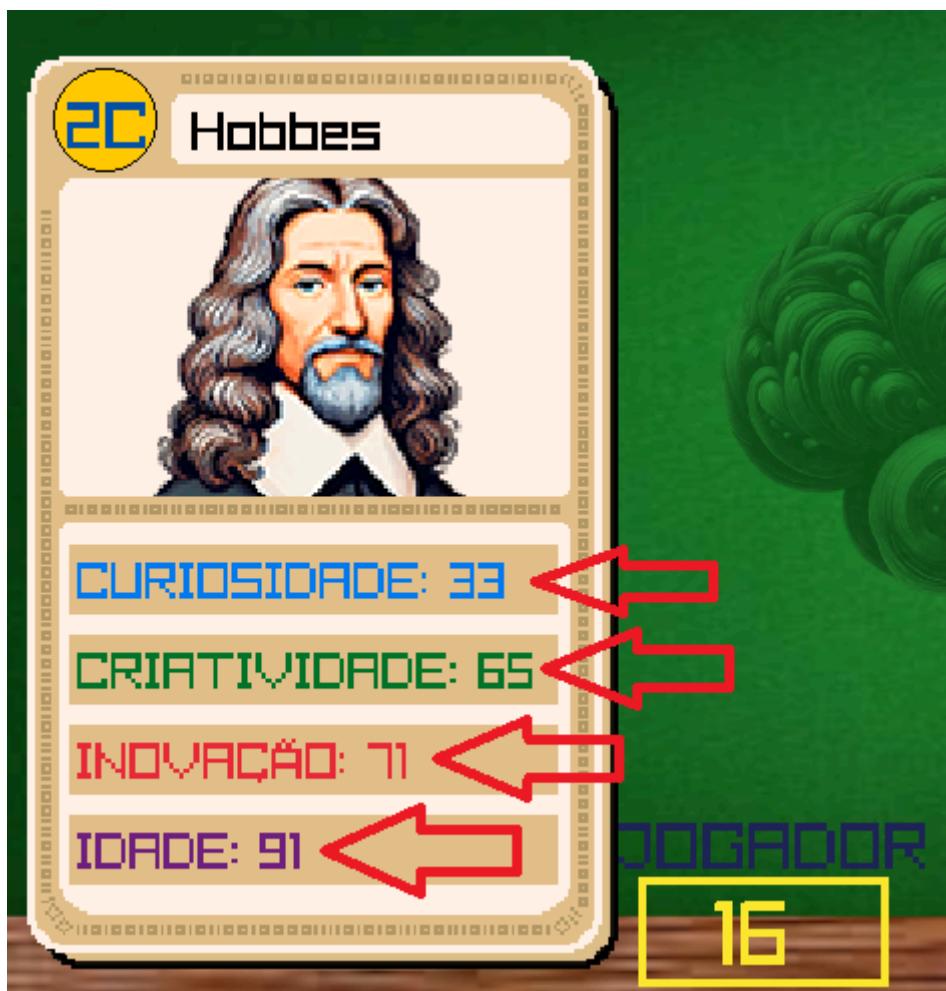


Escolhas possíveis de dificuldade do bot

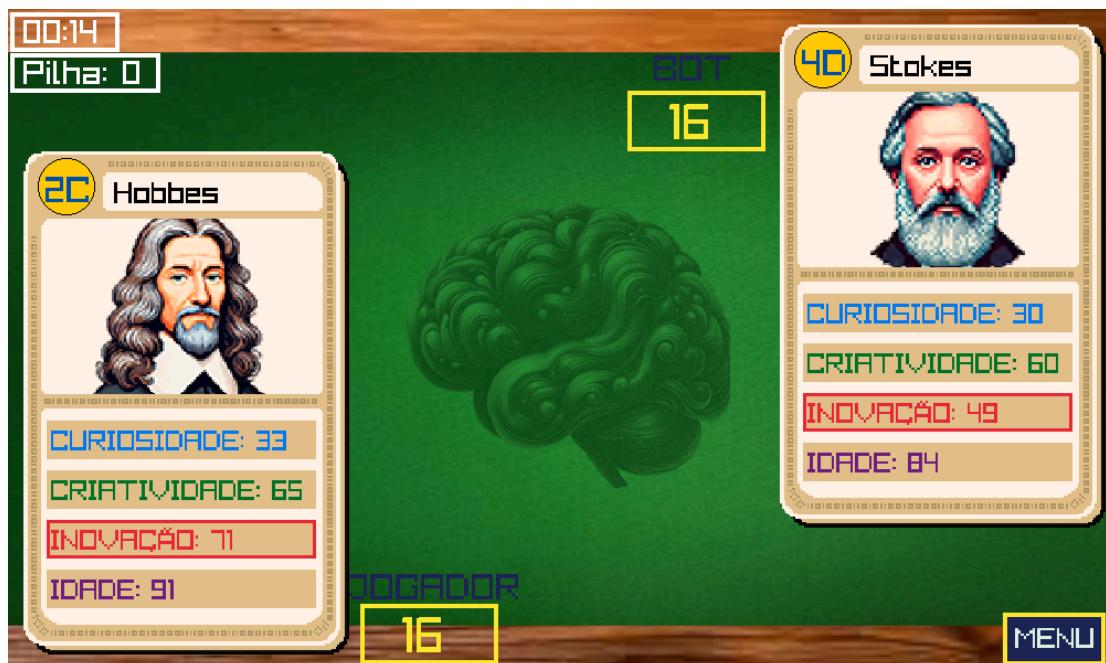
Após isso o jogo irá começar, seguindo as regras já apresentadas, neste menu, o primeiro a jogar será sempre o jogador, ele poderá escolher um dos quatro atributos possíveis de sua carta, os atributos são:

- Curiosidade;
- Criatividade;
- Inovação;
- Idade;

Após escolher um dos atributos o jogo irá exibir a carta de cima do monte do bot por um tempo, e após isso o jogo colocará ambas as cartas no fim do monte do vencedor, se ocorrer um empate as cartas serão colocadas na pilha de empate, que não é exibida na tela, mas que junto ao cronômetro na parte superior da tela é exibida a quantidade de cartas acumuladas nesse monte de empate (Pilha).



Exemplo de possibilidade de escolhas dos atributos



Tela completa desse modo, após a escolha de um atributo pelo jogador

Seguindo as regras descritas, o jogo vai ocorrendo até que ou o jogador ou o bot tenha todas as cartas do monte, sendo assim, o jogo irá encerrar e irá exibir uma mensagem mostrando quem ganhou o jogo. Os detalhes da programação e das regras adicionais podem ser vistas no tópico *DETALHES DA PROGRAMAÇÃO*.

MULTIPLAYER:

Nesse modo de jogo, teremos um jogo entre dois jogadores em máquinas diferentes, ou seja, eles poderão estar em locais diferentes e ainda sim jogar uma partida, o sistema inicial desse menu, se consiste na criação de uma sala para jogar / conectar em uma já existente, porém, antes disso, para o jogo multiplayer funcionar, ele precisa se conectar aos servidores da CORE-NEXUS, onde está hospedado esses serviços de gerenciamento das salas. A primeira coisa que o jogo fará é a conexão ao servidor.



Exemplo de tentativa de conexão ao servidor



Mensagem de erro caso o usuário esteja sem internet, ou o servidor esteja fora de serviço

Porém se tudo estiver ok com o servidor e com a internet / conexão do usuário, então o menu de conexões ficará disponível para iniciar.



Menu de conexões do multiplayer

Com a mensagem “conectado ao servidor”, os serviços do multiplayer estarão disponíveis, e portanto o usuário terá duas opções, a primeira é de criar uma nova sala, e a segunda é de entrar à alguma já existente, para ambas as opções o usuário será solicitado de digitar um nome para jogar. No caso de entrar em alguma sala já existente, ele será solicitado a digitar o **código de 8 dígitos da sala**. Esse código é composto apenas por números, e o jogo irá antes de conectar à sala, verificar se a sala foi devidamente criada e se não há outro jogador já conectado. Qualquer caso contrário será tratado com um alerta de erro.

Se ele decidir criar uma nova sala, o jogo irá gerar um código de sala totalmente verificado pelo servidor, para evitar conflitos de salas de mesmo código, e ele será exibido no canto superior direito da tela, para que o jogador 1, assim nomeado, passe para o jogador 2, para então que ele conecte.



Código de conexão da sala criada

No caso do jogador 2, colocar um código válido, ele entrará na sala e o jogo se iniciará, similar à tela do jogo com bot, porém, no canto superior o jogo exibirá de qual jogador é a vez de jogar, no resto o jogo segue o mesmo princípio e regras já apresentadas.



Tela de jogo do multiplayer

Assim como no singleplayer, o jogo vai seguindo usando as regras já apresentadas e assim que um dos jogadores ganhar a partida, ou seja, tiver em mãos as 32 cartas, o jogo automaticamente encerra a sala e exibirá uma mensagem do jogador vencedor;

OBSERVAÇÕES DESSE MODO DE JOGO:

- O jogador não deve sair do jogo de maneira a fechar o programa pelo “esc” ou pelo X da janela, apenas pelo botão de menu, pois assim evita conflitos ao servidor, que não irá reconhecer essa saída “forçada”;
- Ainda sim se os jogadores saírem da maneira não recomendada, o servidor fecha a sala automaticamente após 40 minutos da última movimentação da mesma;
- Pode ocorrer, devido à conexão do usuário, um delay mínimo entre a jogada de um player e a exibição na tela do outro, porém, como o jogo é muito leve, esse delay será quase imperceptível em muitos dos casos;
- Após abandonar a partida, será impossível retornar, pois o servidor ao identificar partidas abandonadas irá automaticamente encerrá-las.

GERENCIADOR DE CARTAS

As cartas usadas no modo de jogo singleplayer são cartas que podem serem alteradas pelo jogador, já as cartas do multiplayer são as 32 cartas originais do jogo, para manter a unicidade das cartas nesse modo. Para alterar as cartas o menu **gerenciador de cartas** é um menu que apresenta algumas operações possíveis de serem realizadas nas cartas. As operações são:

- Adicionar: Nesse, o usuário poderá criar uma nova carta, que será colocada no fim do baralho existente, a imagem dessas cartas criadas será uma imagem padrão para todas as cartas criadas, para simplificar o menu, além de ter valores que deverão ser respeitados para os campos de entrada, os *filtros*;
- Excluir: Este botão fará a exclusão da carta atualmente selecionada, após a exclusão da carta, o jogo automaticamente irá ajustar o baralho;
- Editar: Quando usado, poderá modificar atributos das cartas, seguindo os *filtros*, os valores podem ser modificados um a um, ou todos de uma vez, se não houver nada a editar, ele manterá a originalidade da carta;
- Pesquisar: Neste submenu o usuário poderá pesquisar entre as cartas, aplicando um filtro por vez, conforme a sua escolha, o Super Trunfo terá prioridade sobre os outros filtros;
- Listar: Por padrão quando o usuário abre o gerenciador, esse menu é o habilitado, ele apenas exibe todas as cartas existentes e o seu total;
- Exportar: Se o usuário quiser exportar um arquivo CSV contendo todas as cartas existentes e ou modificadas, ele poderá usar esse submenu, que colocará esse arquivo de saída na pasta raiz do jogo;
- Resetar Cartas: Nesse caso, se o usuário clicar no botão, o jogo irá resetar o vetor de cartas do usuário para o vetor de cartas originais do arquivo do banco CSV, retirando também a quantidade de cartas para 32.



Tela do Gerenciador de Cartas

Os filtros descritos no submenus de adicionar e editar são bem simples, mas que previnem entradas erradas por parte do usuário. Para os atributos de Curiosidade, Criatividade, Inovação e Idade, os valores aceitos são de 0 - 100. O valor do campo Número e Letra são aceitos apenas nessa ordem, sendo número aceito de 1 até 8 e a letra sendo aceita apenas de A - D. No parâmetro de Nome não foi definido um limite mínimo de nome, porém, sem nada o jogo não salvará, e o limite máximo para o nome não deverá ultrapassar o limite do próprio desenho da carta. Um parâmetro que existe, mas que não fica disponível diretamente para o usuário é o caminho para a imagem do desenho de cada carta, por padrão, as cartas originais já vem com esse caminho ajustado e as novas cartas que são adicionadas possuem caminho para uma imagem padronizada.

O menu do gerenciador como abordado, tem algumas funcionalidades baseadas na carta em que o usuário está visualizando no momento, logo, as setas amarelas servem para se mover dentro dessas cartas disponíveis, permitindo além da visualização o uso das funcionalidades já descritas.

DETALHES DA PROGRAMAÇÃO

Todo o jogo MENTES BRILHANTES foi programado utilizando a clássica linguagem C, tanto para os sistemas de cartas e jogos, quanto para a criação da interface gráfica (GUI), porém por não ser bem o tipo de público da linguagem, o desenvolvimentos de jogos, alguns detalhes dessa programação tiveram de ser adaptados e modificados, detalhes que de fato mudaram o jeito que o jogo se comporta como um todo.

O jogo como um todo foi construído baseado em funções que controlam os menus com o laço principal de atualização, e chamando um outro arquivo de desenho, que irá desenhar aquele menu específico, ainda no jogo existe as funções gerais que foram usadas por todo o código, as funções de desenho gerais, as funções adicionais em jogo que são usadas pelos dois modos de jogo e as de rede que são usadas pelo multiplayer. A interface gráfica projetada foi toda construída baseada no uso da biblioteca RayLib, uma biblioteca de uso simplificado para construções de janelas 2D e 3D para jogos em diversos tipos de linguagens de programação, porém a maioria dos projetos com RayLib é voltado para uso com linguagem C e C++. Essa biblioteca adiciona funções simples que gerenciam inicializações de telas, fechamento, desenho, áudio e tipos de objetos pré definidos para uso durante o jogo. Existe ainda a biblioteca auxiliar da RayLib, que se chama RayGui, essa adiciona objetos e interações com interface de usuário ainda mais simples, mas que para esse projeto não foi utilizado.

Para objetos da interface, como botões, efeitos animados de texto e imagens, foi criado algumas funções usando a RayLib, com uma lógica própria para que esses elementos funcionassem da maneira adequada no jogo. Outros elementos usam apenas linguagem C e bibliotecas nativas.

- Botões com texto, nulos e invisíveis:

Nos botões que não são vazios, que possuam texto, a lógica de desenho do botão é baseado no arredor do texto. Há um struct que foi criado para esse tipo de botão contendo os seguintes campos:

```

typedef struct
{
    Rectangle colisao;           // Caixa de colisão
    Color cor_botao;            // Cor
    char texto[100];             // texto
    int tamanho_fonte_texto;     // Tamanho de fonte
    Color cor_texto;             // Cor
} Botao; // Struct botão com texto

```

Onde há elementos para o design, a caixa de colisão dele e os objetos da fonte, a caixa de colisão sendo o mais importante, pois é por meio desse retângulo que checamos os cliques do mouse, baseado em colisão.

O botão deve ser declarado, no caso de vários no mesmo menu, usemos um vetor desse struct, e depois, antes do laço principal de cada menu, definimos os conteúdos de cada campo, usando uma função que irá preencher cada campo deste botão copiando os conteúdos para ele, exemplo a seguir a definição dos botões do menu principal do jogo:

```

// Definindo os botões do menu:
Botao botoes_menu[5]; // Declara todos os botões da tela de menu

botoes_menu[0].colisao = criarBotao(&botoes_menu[0], 42, 525,
NOSSO_AZUL, "SAIR", 33, NOSSO_CINZA);

botoes_menu[1].colisao = criarBotao(&botoes_menu[1], 42, 450,
NOSSO_AZUL, "Créditos", 33, NOSSO_CINZA);

botoes_menu[2].colisao = criarBotao(&botoes_menu[2], 42, 375,
NOSSO_AZUL, "Gerenciar cartas", 33, NOSSO_CINZA);

botoes_menu[3].colisao = criarBotao(&botoes_menu[3], 42, 300,
NOSSO_AZUL, "Singleplayer", 33, NOSSO_CINZA);

botoes_menu[4].colisao = criarBotao(&botoes_menu[4], 42, 225,
NOSSO_AZUL, "Multiplayer", 33, NOSSO_CINZA);

```

Após isso, o botão está pronto para ser desenhado e clicado, usemos as funções *desenhaBotaoTxt(Botao *botao)*, para desenhar o botão na tela, o princípio dela é desenhar um retângulo com borda amarela, com um tamanho baseado no tamanho do texto respectivo ao botão, e *checarClique(Rectangle *botao.colisao)*, para checar se o usuário está com o cursor em cima do botão e

clicou com o botão esquerdo do mouse, se isso for verdadeiro a função retorna true.

Ainda podemos ressaltar o botão, que é aplicar um efeito de destaque no botão quando o usuário está sob o botão com o mouse, usando uma lógica de uma variável global, que é usada para verificar todos os botões de um menu de uma vez só, se a variável se manter em 0, a função de ressaltar irá colocar o cursor do mouse como normal, se não, se for exemplo 1, irá aplicar o cursor de clicável, e assim para os tipos necessários, a parte que troca a cor do botão é feita na própria função de desenho do botão, que irá verificar se atende a condição e desenhar o fundo do botão com a cor de destaque. Abaixo a função que troca o cursor do mouse, que é aliada ao ressaltar dos botões:

```
void resaltaBotoes()
{
    if (botoes_resaltar == 1)
    {
        SetMouseCursor(MOUSE_CURSOR_POINTING_HAND); // Clicavel
    }
    else if (botoes_resaltar == 0)
    {
        SetMouseCursor(MOUSE_CURSOR_DEFAULT); // Normal
    }
    else if (botoes_resaltar == 2)
    {
        SetMouseCursor(MOUSE_CURSOR_NOT_ALLOWED); // Proibido
    }
    return;
} // Função que troca o tipo de cursor, aliado ao ressaltar dos botões
```

Há ainda o som que o botão faz quando o mouse está sobre ele, e o de quando clica, porém isso iremos abordar na parte de sons. Também existe o botão invisível e o botão nulo, ambos são derivações diretas do botão com texto, seguindo a mesma lógica de desenho e clique, o nulo apenas não tem texto e seu tamanho é definido pelo próprio programador, e o invisível, tem apenas caixa de colisão, com sua caixa acompanhando algo, normalmente uma imagem, assim como é feito por

exemplo no gerenciador de cartas, com as setas amarelas, que são botões invisíveis.

- Desenho de imagens e texturas:

Dos detalhes de programação, esse é um dos mais simples pois utiliza apenas funções da própria RaLlib e lógica de programação. Para exibir uma imagem, o importante é definir qual o tipo será utilizado, no caso usamos apenas imagens PNG, que são as mais usuais e permitem canal alpha, segundo é definir uma pasta apenas para elas dentro do projeto do jogo, e para tal usamos a pasta img. Com isso pronto, iremos carregar a imagem, usá-la e ao término de um menu / programa, iremos liberar o seu uso da memória, é bem similar às funções de alocação de memória da linguagem C. Para carregar as imagens, usamos o tipo que irá armazená-la, que é o tipo Texture2D, e após isso é só carregar a imagem usando a função *LoadTexture(Char *caminho da img)*, que irá carregar a imagem e salvar na variável por atribuição. Outra das funções úteis para usar com imagens é a de redimensionar uma imagem, essa função não vem por padrão nas funções da RayLib porém, é bem simples:

```
Texture2D ResizeTexture(Texture2D texture, int novoComprimento, int novaLargura)
{
    Image image = LoadImageFromTexture(texture); // Obtem a imagem da textura
    ImageResize(&image, novoComprimento, novaLargura); // Redimensiona a imagem
    Texture2D newTexture = LoadTextureFromImage(image); // Cria uma nova textura a partir da imagem redimensionada
    UnloadImage(image); // Libera a imagem temporária
    return newTexture;
} // Função para redimensionar a textura (Imagens)
```

Usar o tipo de Texture2D ao invés do Image é útil para melhor manipulação dela com as funções da própria RayLib. Após carregar uma textura, usando a função *DrawTexture(Texture2D * textura, int x, int y, Color cor)*, que irá desenhar na tela essa textura. Após todos os usos necessários da textura, como boa prática no código, iremos descarregar essa textura da memória, usando a função *UnloadTexture(Texture2D textura)*, que irá descarregar ela da memória. Outro uso de

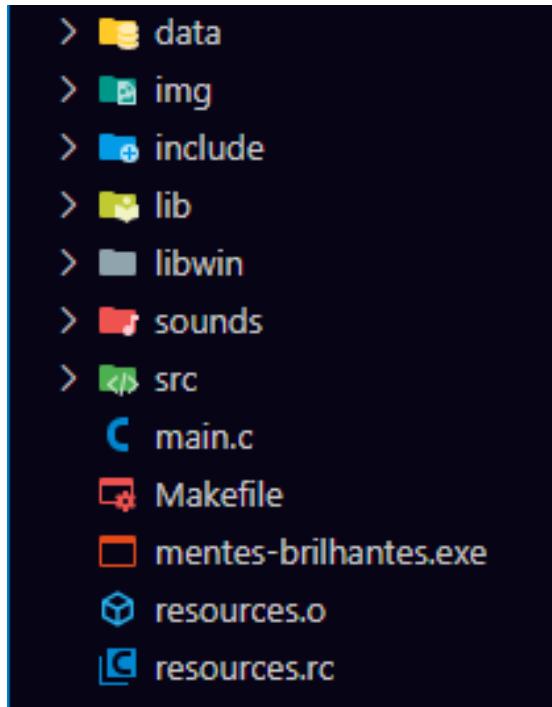
texturas é quando está desenhando uma carta na tela, nesse caso, a imagem do rosto da carta é uma textura, e para tal, precisamos sempre usar esse mesmo processo, porém, como fazer no gerenciador para mudar a textura usando a mesma variável? A resposta é simples, usando uma simples lógica de estados anteriores, como por exemplo, o gerenciador de cartas, que a cada nova carta desenhada, troca a textura do rosto:

```
if (id_anterior != cartas[carta_atual - 1].id || exclusor)
{
    exclusor = false;
    id_anterior = cartas[carta_atual - 1].id;
    UnloadTexture(img_carta);
    img_carta = LoadTexture(cartas[carta_atual - 1].imagem);
} // Usado para carregar a textura de uma carta apenas na
troca de carta / exclusão da atual
```

Esse código irá trocar a imagem da carta, a cada mudança do id da carta atual, ou seja, se a carta que está sendo exibida não é a mesma da passada, ele irá descarregar e recarregar a nova imagem para utilizar, o mesmo caso ocorre quando a carta atual é excluída pois nesse caso a carta irá ser deletada, então a imagem precisa ser trocada.

- Organização geral do código:

Como sabemos a linguagem C tem em geral apenas dois tipos de arquivos usados nos projetos, os arquivos C e H, sendo respectivamente, os de código fonte e de cabeçalho. O código do jogo soma mais de 6 mil linhas entre esses arquivos. Embora pudéssemos fazer o programa todo em um só arquivo, isso é totalmente anti-organizacional, portanto os arquivos de cabeçalho ficarão na pasta include e os arquivos fonte na pasta src, dentro dessa, subpastas para organizar todos os setores das funções do jogo, como funções do gerenciador, funções de rede, funções de controle e de desenho. Outras pastas são a data, que contém os arquivos externos adicionais do jogo, a pasta sounds que contém os sons/músicas do jogo, lib e libwin que contém a biblioteca estática da RayLib, e outras coisas a mais na pasta raiz.



Organização das pastas do jogo.

- Operações sobre as cartas e seu desenho:

Nesse jogo, as cartas que são exibidas no gerenciar cartas, são as cartas globais, que são carregadas do “banco.csv” na primeira abertura do jogo e posteriormente são carregadas no arquivo “cartas.bin”, elas são usadas nesse menu e no jogo do singleplayer também, o vetor se chama apenas cartas e pertence ao tipo Struct Carta. O tipo possui os seguintes campos:

```

typedef struct
{
    int id;                      // Id base da carta, usado nos loads das imagens
    bool super_trunfo;           // Sinalizador para Cartas Super-trunfo
    char hexadecimal[5];         // Numeros e letras que ficam no canto da carta
    char imagem[256];            // Endereço relativo das imagens da carta
    char nome[51];               // Nome da carta
    unsigned int curiosidade;
    unsigned int criatividade;
    unsigned int inovacao;
    unsigned int idade;
} Carta;

```

Os valores do id são atribuídos inicialmente pelo CSV, porém, a função de excluir e adicionar do gerenciador manipulam esses id's. As operações do gerenciador de cartas, sobre esse vetor de cartas são bem simples, sendo exclusão de uma carta, adição de uma nova, edição, pesquisa, exportar o vetor, e reset das cartas.

O **adicionar** irá pegar as entradas das caixas de texto digitadas pelo usuário, verificar se estão no padrão e irá adicionar essa carta no final do baralho, com uma imagem padrão para todas que não são originais do banco. Os filtros padrões em questão são: Para os campos de idade, curiosidade, criatividade, inovação, devem ser números entre 1 - 100, para o campo hexadecimal, deve ser primeiro um número entre 1 - 8 e depois uma letra entre (A, B, C, D), o nome não deve ser vazio, e por fim, a função irá verificar se já há super-trunfo no baralho. Qualquer coisa oposta a esses filtros não irá permitir que a carta seja adicionada e irá ativar uma mensagem de erro.

O **Editar** irá realizar as mesmas checagem de filtros, e retornar as mesmas mensagens de erros, se os parâmetros estiverem conforme, ele irá substituir nos valores da carta que está sendo exibida no gerenciador no momento.

O **excluir** irá excluir do vetor de cartas a carta que estiver sendo exibida no momento no gerenciador, e irá ajustar os id's das cartas que estão atrás da excluída.

O **pesquisar** irá permitir que o usuário busque dentro do vetor de cartas, por intervalos os atributos já mostrados, onde o campo do super-trunfo tem prioridade sobre os outros, ou seja, se estiver ativo, mostrará essa carta.

Em todos esses casos, exceto o pesquisar, após realizar a operação, ele irá retornar ao **listar**, que mostra todas as cartas presentes no vetor, no caso do pesquisar, se não houver parâmetros a pesquisar ele retorna ao listar, se houver mas não tiver cartas correspondentes, mostrará uma carta sem parâmetros, caso houver, mostrará as cartas. Basicamente, essas são as operações sobre as cartas que o usuário pode realizar. O jogo ainda deve em alguns menus desenhar as cartas na tela, para isso, temos uma função que irá fazer isso, ela é a *desenhaCarta(int x, int y, Texture2D *frente_carta, Carta *carta, Texture2D *img_carta)*, recebendo respectivamente os parâmetros de posição x e y do canto superior esquerdo da carta, a textura da frente da carta, ou seja, do frame básico da carta, recebe também os dados pelo atributo carta, e recebe a imagem do rosto pelo img_carta, a carta deve ser desenhada como um sanduíche de texturas para que a imagem possa ser desenhada corretamente,

utilizando o frame para todas, se a carta não for uma do tipo super-trunfo, o seu desenho é bem simples:

```

DrawRectangle(x + 15, y + 45, 260, 175, NOSSO_CREME); //Desenha o fundo da
imagem

DrawTextureEx(*img_carta, (Vector2){x + 61, y + 53}, 0.0f, 1.75f, WHITE);
//Desenha o rosto da carta

DrawTextureEx(*frente_carta, (Vector2){x, y}, 0.0f, 1.75f, WHITE); // //Desenha o frame da carta

for (int t = 0; t < 4; t++)
{
    DrawRectangle(x + 21, y + 244 + t * 45, 244, 35, NOSSO_BEGE);
//Desenhando os fundos dos atributos
}

if (carta->hexadecimal[0] == '9' && carta->hexadecimal[1] == 'Z')
{
    DrawText(TextFormat("SEM CARTAS QUE", carta->inovacao), x + 25, y +
100, 26, RED);
    DrawText(TextFormat("CORRESPONDEM!", carta->inovacao), x + 30, y +
129, 26, RED);
} //Para cartas vazias do gerenciador - pesquisar

//Desenha os parametros e o hexadecimal:

DrawText(TextFormat("CURIOSIDADE: %d", carta->curiosidade), x + 25, y +
251, 25, BLUE);

DrawText(TextFormat("CRIATIVIDADE: %d", carta->criatividade), x + 25, y +
296, 25, DARKGREEN);

DrawText(TextFormat("INOVAÇÃO: %d", carta->inovacao), x + 25, y + 341, 25,
RED);

DrawText(TextFormat("IDADE: %d", carta->idade), x + 25, y + 386, 25,
DARKPURPLE);

DrawText(TextFormat("%s", carta->nome), x + 82, y + 25, 27, BLACK);

DrawCircle(x + 20 + (MeasureText(carta->hexadecimal, 33) / 2), y + 32, 26,
BLACK);

DrawCircle(x + 20 + (MeasureText(carta->hexadecimal, 33) / 2), y + 32, 25,
GOLD);

```

```
DrawText(TextFormat("%s", carta->hexadecimal), x + 20, y + 20, 33,
DARKBLUE);
```

Porém quando a carta é do tipo super-trunfo ela contém uma animação, bem simples, das letras escritas “SUPER TRUNFO”, ficarem mudando de cor uma a uma, para isso também é bem simples, foi preciso criar um vetor do tipo Color do tamanho do texto, e a cada 10 frames do jogo, essas cores são atualizadas randomicamente uma a uma, e o desenho do texto é feito letra por letra aplicando a cor respectiva desse vetor de cores.

- Caixas de texto:

As caixas de texto são desenhadas e operadas de forma bem similares à operações de botões, existe uma struct para ela, que irá gerenciar seu uso e desenho, o tipo é:

```
typedef struct
{
    Rectangle caixa;           // Caixa de colisão
    Color cor_caixa;          // Cor de fundo da caixa
    char texto[128];          // Armazenamento do texto da caixa
    char texto_placeholder[128]; // Placeholder
    Color cor_texto;          // Cor do texto
    bool placeholder;         // Se o placeholder deve ser desenhado
    bool habilitado;          // Se a caixa esta ativa para entradas
    int tamanho;               // Tamanho do texto da caixa
    int subgrupo;              // Tipo possivel de entrada
    int subcaixa;              // Para ativar a leitura de multiplas caixas
                               // de texto
} TextBox;
```

O que mais chama atenção de novidade nessa estrutura são os parâmetros de verificação, mais de fato, esses são bem simples. Uma caixa de texto é basicamente construída a partir de um retângulo de fundo com texto, nesse caso, quando o conteúdo da caixa, ou seja o texto, for vazio, a caixa irá por padrão desenhar na tela a informação do placeholder, que guia o usuário mostrando para ele qual tipo de

informação a caixa permite entrada, ainda sobre esse tipo de entrada, que realiza esse controle é a variável subgrupo, que na função de entrada de texto irá restringir ou não para apenas alguns tipos de dados a entrada da caixa. Já quando você tem várias caixas desenhadas simultaneamente na tela, ambas não podem receber dados juntas, para tal usamos a variável subcaixa, que irá ativar apenas uma caixa para entrada, aliadas a uma lógica de clique sob a caixa de colisão da própria caixa, ativando / desativando a caixa. Por fim, a variável habilitada irá de fato permitir que a caixa por completa seja desenhada na tela, usamos isso quando há várias caixas de texto na mesma posição e queremos escolher qual irá ser desenhada de fato. Antes de usar a caixa, antes deve criá-la e atribuir os valores iniciais a ela, e para tal usamos o seguinte, exemplo da tela de multiplayer:

```
TextBox caixa_texto[2];
    caixa_texto[0] = (TextBox) {{627, 200, 340, 46}, NOSSO_CREME, "", "Código:", NOSSO_AZUL, true, true, 0, 0, 0};
    caixa_texto[1] = (TextBox) {{310, 95, 340, 46}, NOSSO_CREME, "", "Nome Player:", NOSSO_AZUL, true, true, 0, 0, 0};
```

Após criar e atribuir os dados iniciais à ela, iremos desenhar na tela, usando a função *desenhaTextBox(TextBox *caixaTexto)*, que irá checar se ela está ativa para desenhar, irá também verificar se o placeholder está ativo para desenhá-lo, se não irá desenhar o conteúdo da caixa. O uso da caixa de texto normalmente ou está atrelado à um radio button que irá ativar ela e permitir sua entrada de dados, ou atrelada ao clique que irá ativá-la quando houver mais que uma. Fazendo essa lógica, com if, for, e as funções que checam clique, podemos usar a função *leTeclado(char *string, int *conta_tecla, int subgrupo, int limitador)*, que irá realizar a leitura de teclas do teclado e salvar na string, nesse caso o texto da caixa e aumentar operar o conta_tecla, que nesse caso é o tamanho do texto da caixa, além de receber o subgrupo para filtrar e o limitador que limita a quantidade de caracteres que a caixa de texto irá ler.

- IA do bot e funções gerais de partida:

Durante uma partida contra bot, singleplayer, o jogador estará jogando contra a máquina, e portanto, as escolhas desta, devem ser realizadas seguindo o nível de dificuldade que o jogador selecionou no início da partida. As escolhas possíveis são 4,

logo, o bot terá que escolher entre esses, no caso a dificuldade escolhida irá afetar diretamente nessas escolhas possíveis:

1. **Fácil:** A IA do bot irá excluir das possibilidades as 2 maiores valores, depois irá escolher com randomização entre as outras duas que sobraram;
2. **Médio:** A IA do bot irá excluir das possibilidades o maior valor, depois irá escolher com randomização entre as outras três que sobraram;
3. **Difícil:** A IA do bot irá excluir das possibilidades os 2 menores valores, depois irá escolher com randomização entre as outras duas que sobraram.

Durante a partida, a variável int `btn_clicado`, irá ser usada para verificar o clique do botão relativo ao parâmetro da carta durante a vez do jogador, e usado também pela IA do bot, pois a função `clicado(int *btn_clicado, Carta *cartas_bot, Carta *cartas_jogador, Carta *pilha_empate,int *quantidades_cartas, int tipo_jogador)`, usa essa variável para de fato, computar a jogada, ela irá baseado nesses parâmetros processar os dados do ganhador, empate e perdedor. Durante as jogadas uma função que verifica se alguém ganhou, o bot ou jogador, `checaVitoria(int *quantidades_cartas, int *submenu_tela)`, e poderá computar uma vitória para um desses dois se a quantidade de cartas de algum dos dois zerar, para isso ela usará a variável `sub_menu`, que dentro do menu do jogo, simboliza em qual estado o jogo está acontecendo, a seguir mostra a tabela de estados dessa variável para o jogo singleplayer:

Variável <code>sub_menu</code> :	Estado da tela:
0 (padrão)	Aguarda o jogador setar a dificuldade do jogo
1	Contador de 3 segundo para início do jogo
2	Partida contra o bot
5	Jogador Ganhou
6	Bot Ganhou

Ainda sim, em cada uma dessas telas o botão de retornar ao menu principal é renderizado, permitindo a saída do usuário desse modo quando quiser. Tudo de forma bem similar é realizado no menu de partidas multiplayer, porém, ao invés de ter a IA do bot, teremos dois jogadores em clientes distintos conectados ao servidor do jogo, que

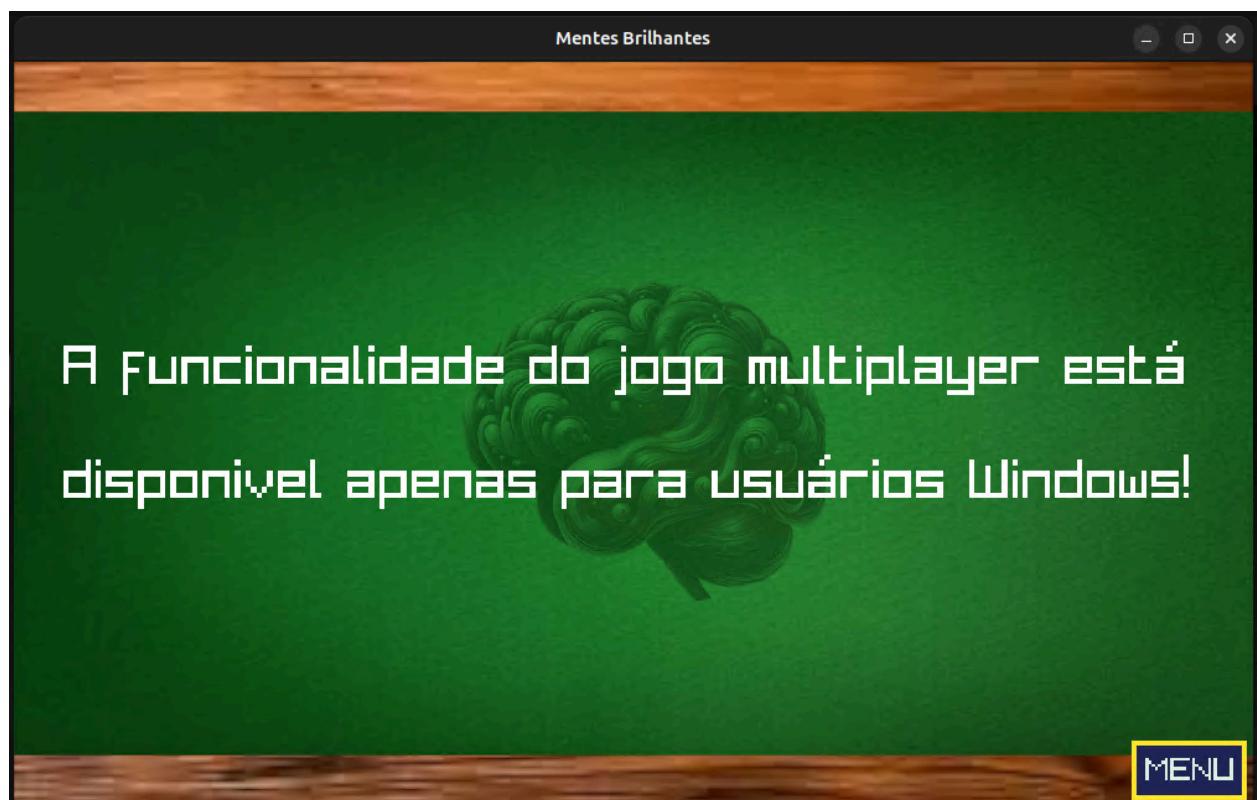
realizará toda essa comunicação intermediária dos processos, isso será melhor mostrado, nas funções de rede, porém a variável submenu_tela continua aqui e sua tabela é essa, com uma adição da variável int conexao_server, que mostra se está conectado ao servidor do jogo, auxiliando o submenu no estado da tela:

Variável sub_menu:	Variável conexao_server	Estado da tela:
0 (padrão)	0 (padrão)	Tentando conexão ao servidor
0 (padrão)	1	Servidor conectado, menu de conexões e salas aberto
0 (padrão)	2	Erro ao conectar ao servidor
1	x	Sala criada, aguardando jogador 2 conectar
2	x	Partida Multiplayer
3	x	Contador de 3 segundos para início da partida
5	x	Jogador 1 Ganhou
6	x	Jogador 2 Ganhou
8	x	Jogador abandonou jogo

Durante o jogo multiplayer também usamos um vetor de inteiro, chamado comandos, que é recebido e enviado ao servidor do jogo, esse vetor é tudo que nosso cliente precisa para fazer o jogo funcionar, explicaremos mais dele nas funções de rede, porém é ele que altera a vez de jogar dos clientes, e para tal, fazendo que o cada cliente se comporte ou como vez de jogar, ou vez de receber a jogada, o importante é saber, que as mesmas funções são aproveitadas do bot para esse menu, e portanto chamamos durante o código de jogador 1 e 2, o 1 sendo aquele que é do seu cliente e o 2 o convidado, isso para ambos os clientes, logo, cada cliente trata como jogador 1 o seu jogador, simplificando o processo para as funções de checagem, para fazer a diferenciação dos jogadores, uma variável auxiliar int tipo_entrada é usada para mostrar se o jogador criou a sala ou se ele conectou a sala, e assim seu tipo será processado diferentemente pelas funções do jogo.

- Funções de rede:

Dentro do jogo, há uma série de funções de rede que foram projetadas para comunicação com um servidor remoto, usando o protocolo http. Esse servidor hospeda os serviços e salas do jogo, de forma que não precisamos usar conexão direta, webSocket e threads, essa diferença já será explicada, o mais importante agora é saber que essas funções de rede foram construídas para o sistema operacional Windows usando as bibliotecas windows.h e winhttp.h, e portanto quando um usuário linux tenta entrar no menu de jogo multiplayer, uma tela indicando isso ocorre:



Usuário linux tentando acessar o multiplayer

Como já apresentado, a comunicação entre o servidor e os clientes é realizado pela variável comandos, mas também usamos numa primeira conexão, a variável que representa o id's de todas as cartas para mostrar ao cliente do jogador 2 quais cartas são dele e quais são do jogador 1, além dos nomes dos jogadores que é um vetor char que é enviado na primeira recepção de dados também. Na primeira posição do comandos é transportado para o servidor e para os clientes o btn_clicado, ou seja, qual botão das cartas o usuário pressionou na sua vez, na posição 2 temos uma série de códigos que regem esse menu:

Variável comandos[1]:	Estado do jogo:
0	Sala desconectada / inexistente / fechada
16	Sala criada, aguardando jogador 2
15	Jogador 2 conectado
1	Vez do jogador 2
-1	Vez do jogador 1

Existe uma coletânea de funções que se comunicam com o servidor, todas elas são bem parecidas entre-si, porém cada uma cumpre o seu propósito. Todas elas usam a mesma base, primeiro elas instanciam o endereço do servidor e preparam o path que é o caminho dos scripts e salas do jogo, após isso a função abre uma conexão http e conecta ao servidor, depois cria e mandar / recebe requests do servidor e por fim finaliza todos esses processos. Para o endereço e o path, usamos o tipo wchar_t, que é similar ao char, para instanciar os processos usamos uma variável do tipo HINTERNET, que é usada para armazenar os dados das conexões e processos, e serem reutilizadas, durante o uso de cada funções internas, elas possuem muitos parametros, porem, muitos deles são inutilizados, logo é algo bem simples esse tráfego de dados.

Para enviar dados, é utilizado um formato JSON de dados para envio, os dados são formatados e enviados todos de uma vez, para ler do servidor, os dados são lidos totalmente através de um buffer, que lê o conteúdo total dos arquivos do servidor. Cada sala do servidor é representada por um arquivo txt, que é gerido no servidor por scripts PHP. Todas as funções operam de maneira similar, usando como parâmetro as variáveis necessárias e executando o ciclo já explicado, as funções de rede são:

Função:	Descrição:
conectar_server()	Na abertura do menu multiplayer, faz uma primeira conexão com o servidor.
enviarDados()	Envia os dados após uma sala ser criada
encerrar_secao()	Finaliza uma sala ao ser chamada, excluindo o

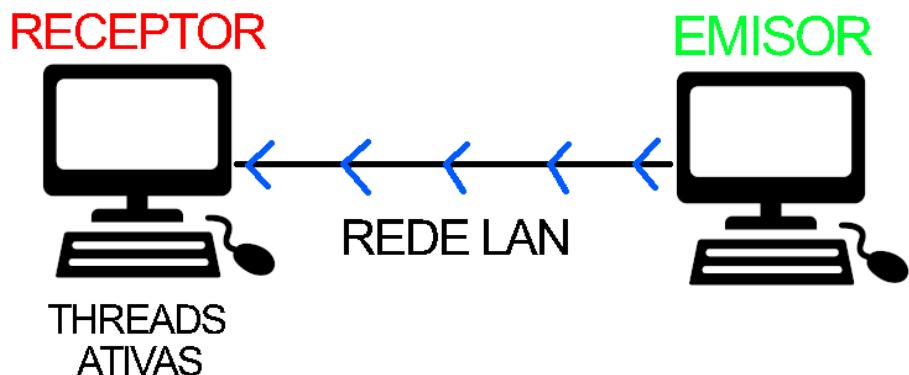
	arquivo txt do servidor
leServidor1()	Le o vetor de comandos do servidor
leServidor2()	Le o vetor de comandos do servidor e processa se a sala for fechada / vazia para alterar o submenu_tela para 8
receberDadosIniciais()	O cliente que conecta na sala recebe os dados de preparação do jogo
verificarSalaVazia()	Verifica se uma sala está vazia no servidor

Como prometido, vamos analisar a vantagem de usar um servidor central, ao invés de conexões diretas com webSocket. Uma conexão usando webSocket é pautada em uma rede lan normalmente, ou seja, os clientes deveriam estar na mesma rede, já sendo uma desvantagem, porém o maior problema de usar essa tecnologia, é que o cliente que recebe dados, deveria ficar escutando dados até eles serem recebidos, e isso travaria as outras funções do jogo, principalmente as de desenho, logo, para contornar esse problema pode se usar threads, assim várias funções do jogo poderiam ser executadas juntas, porém isso é de certa forma inviável, pois durante as jogadas, muda-se muito rápido do modo de enviar e receber dados, outro fator que pesa nesse quesito é que por estarmos usando mais recursos, mais processador do cliente seria usado para manter essas funções em atualização constante. A grande vantagem desse tipo de conexão, é a ausência de qualquer sistema central, ou seja um servidor destinado, já que as máquinas se comunicam de maneira independente.

A grande vantagem de se usar o protocolo http, e um servidor central é que dispensa o uso de threads, pois é baseado em alterações de valores em arquivos no servidor, não numa mensagem de recepção rápida, e portanto, acaba gastando menos recursos das máquinas, a maior desvantagem desse método, é que os computadores dos cliente precisam estar conectados a internet para o tráfego de dados, e o servidor precisará estar ativo e em serviço para tal, uma vantagem é que as máquinas não precisam estar na mesma rede, podem estar em locais completamente distintos. Portanto, como era mais viável, e mais simples de imediato, foi usado nesse jogo o método usando o protocolo http e servidor central, o endereço do servidor http é gg128.ddnsfree.com, que usa um ddns automático para atualizar o endereço do servidor e usando um host apache para que os scripts PHP e arquivos estejam disponíveis na web, além de outros detalhes para manter o servidor em pleno

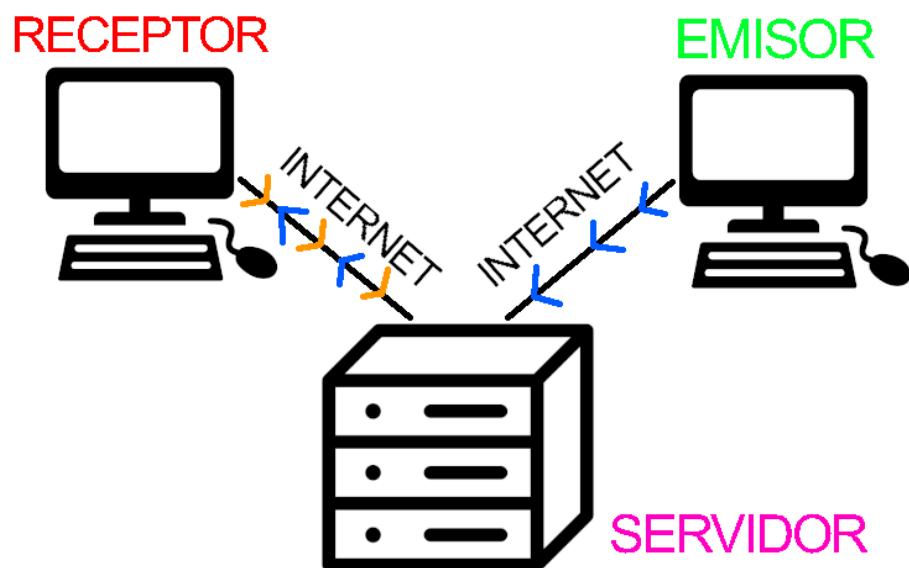
funcionamento. Uma coisa que é válida explicar superficialmente é os scripts PHP, já que não são muito o objetivo desse projeto, mas que como o integra, a explicação é válida. Os scripts são apenas 2, o processar_arquivo.php que cria o arquivo da sala, após receber uma solicitação com o json enviado, e começa a colocar dados nesse arquivo. E o arquivo destruir_jogo.php que ao receber o id da sala, exclui o arquivo txt relativo e também remove todos os arquivos txt que tem mais de 40 minutos sem alterações.

WEBSOCKET



SERVIDOR

- Tráfego de dados
- Requisições



Diferença de infraestrutura de rede apresentadas

DETALHES DA COMPILAÇÃO E EXECUÇÃO

Nesse tópico se encontram detalhes sobre a compilação do arquivo fonte para a geração do arquivo executável, e sua execução detalhada através do arquivo makefile.

COMPILAÇÃO:

Ambiente Windows:

1. Baixe o repositório do jogo Mentes Brilhantes pelo github, <https://github.com/usuario/projeto.git>;
2. Baixe o compilador (se ainda não possuir) MINGW-64, pelo link: <https://github.com/skeeto/w64devkit/releases/download/v2.0.0/w64devkit-x64-2.0.0.exe>, e execute o arquivo;
3. Coloque o patch Bin do compilador nas variáveis de ambiente do seu sistema Windows, se encontrar algumas dificuldades o vídeo <https://www.youtube.com/watch?v=aducc6-ra90>, pode ser útil;
4. Pelo terminal navegue até a pasta do jogo usando "CD" e "DIR", a pasta principal é a que contém o arquivo MAKEFILE, dentro dela siga os próximos passos:
5. Execute o comando “make exec” que irá automaticamente limpar arquivos antigos, se houver, e compilar automaticamente o jogo, e irá executá-lo após a compilação;

Ambiente Linux:

1. Baixe o repositório do jogo Mentes Brilhantes pelo github, <https://github.com/usuario/projeto.git>;
2. Instale as dependências para as bibliotecas de vídeo e áudio, execute o seguinte comando como administrador no terminal: “`sudo apt install libasound2-dev libx11-dev libxrandr-dev libxi-dev libgl1-mesa-dev libglu1-mesa-dev libxcursor-dev libxinerama-dev libwayland-dev libxkbcommon-dev`”;

3. Pelo terminal navegue até a pasta do jogo usando "CD" e "DIR", a pasta principal é a que contém o arquivo **MAKEFILE**, dentro dela siga os próximos passos:
4. Execute o comando “make exec” que irá automaticamente limpar arquivos antigos, se houver, e compilar automaticamente o jogo, e irá executá-lo após a compilação;

REQUISITOS:

Ambiente Windows:

Os requisitos de execução nesse ambiente giram em torno dos próprios requisitos da raylib (Biblioteca gráfica usada):

1. OPENGL: Esse é um dos requisitos que a raylib possui, em geral, placas de vídeo mais novas, ou atualizadas já possuem versão compatível para a raylib!
2. DirectX: Para esse ambiente, o DirectX é necessário já que são uma série de api's multitarefas pensadas para jogos!
3. Permissão de Administração para jogar em modo "Multiplayer", pois esse recurso necessita de acesso a rede e passagem por firewalls. Pode ocorrer de funcionar sem permissão em alguns casos.

Ambiente Linux:

Nesse ambiente os requisitos são parecidos com os do windows, mas há um detalhe adicional:

1. *Bibliotecas e api gráfica atualizadas: Em geral, para evitar incompatibilidade com a raylib, manter esses dois itens sempre atualizados evita esses problemas.*
2. *Bibliotecas de áudio e vídeo: Durante a compilação no passo 2, mostramos bibliotecas adicionais necessárias para compilação do jogo.*

OBSERVAÇÕES

Para completar todas as informações do jogo, há algumas observações que devemos comentar para complementar tudo sobre o código e os detalhes:

- Tempo de programação: Mais de 200 horas;
- Linhas de código fonte: Mais de 8000 linhas;
- Tempo de escrita do documento: 18 horas;
- Nome do projeto: Mentes Brilhantes, super trunfo;
- Arquivos de projeto: C, H e PHP: 49;
- Quantidade máxima teórica de salas disponíveis multiplayer: 89.999.999;
- Bibliotecas importantes usadas: RayLib, Winhttp, stdio e string;
- Sistemas operacionais suportados: Windows e Linux;
- Arquivos do projeto, sem arquivos fonte: 64;
- Ferramenta geradora de executável: MakeFile;
- Compilador: GCC x64;
- Início e FIm do projeto: 14/11/2024 - 14/02/2025;
- Professor orientador: Prof. Dr. Muriel de Souza Godoi (UTFPR/AP);
- Licença do projeto: Livre para modificações, porém, não comercial;
- Adicionais: Manual de regras e detalhes, slide apresentativo, site para hospedagem dos arquivos e servidor para gerenciamento das partidas;

O projeto do jogo MENTES BRILHANTES, super trunfo, é um projeto requisito total para entrega do trabalho final da disciplina Fundamentos de Programação 2, do professor Muriel de Souza Godoi, da UTFPR campus Apucarana, fruto do trabalho “jogo de modelo super-trunfo” requerido pela disciplina, no segundo semestre de 2024.

CRÉDITOS



Gabriel Felipe



Gustavo Ferreira



Thiago André

Isso conclui o manual de regras e detalhes do jogo, MENTES BRILHANTES, Super Trunfo!