CENTRO UNIVERSITÁRIO DE JOÃO PESSOA – UNIPÊ
CURSO DE CIÊNCIAS DA COMPUTAÇÃO

Marcos Otávio Novais Antunes - 25873946
Matheus José Ramos da Silva 26581337
Joycêvania Lima Salvino - 26861976
Josemilson Elias Lopes - 26708281

Projeto Jogo da Memória

JOÃO PESSOA
2021

SUMÁRIO

1 INTROUÇÃO

1.1 Objetivos

O objetivo do projeto foi para que em equipe desenvolvêssemos um jogo, dentre as opções oferecidas, e que cumprisse todos os requisitos. Esse projeto foi lançado em uma participação conjunta dos professores Walace Bonfim e Douglas Andrade e conta também com, além deste relatório, uma apresentação e um vídeo. O jogo escolhido pela equipe foi o Jogo da memória (4X4) e todos os requisitos necessários foram atendidos,

1.2 Tecnologia envolvida

Durante o processo de desenvolvimento da aplicação foi utilizado a linguagem de programação C, sendo essa um requisito indispensável para a aprovação do projeto. Além disso, foi necessário o uso de IDE (Integrated Development Environment ou Ambiente de Desenvolvimento Integrado), como o Dev C/C++ e Code Blocks, cuja escolha foi critério de cada desenvolvedor. A interface gráfica do sistema foi construída com base na tecnologia CLI (Command Line Interface, ou Interface de Linha de Comando), tendo como justificativa a agilidade no processo desenvolvimento do software, uma vez que o processo de construção se baseia em comando simples que podem exibir mensagens na tela e receber dados do usuário.

2 O JOGO

2.1 Como funciona: Regras

Como equipe tentamos fazer um jogo que ficasse o mais próximo da realidade possível, sem deixar de lado ferramentas que ajudem ao jogador na sua interação com a interface. Como o jogo funciona: O jogador da vez tem a chance de ver a “face” das “cartas” apenas uma vez por rodada durante 5 segundos, após esse tempo as cartas são viradas ao avesso e o jogador precisa indicar as posições em que há cartas iguais. Enquanto o jogador da vez acertar cartas com o mesmo número ele mantém a vez; quando este errar a vez é passada para o outro jogador, que poderá ver a “face das cartas” restantes durante 5 segundos e indicar onde estão os pares de cartas. Da mesma forma, enquanto esse jogador acertar a vez dele continua.

A cada carta escolhida corretamente soma-se um ponto para o jogador que acertou; ao fim, quando todas as “cartas” estiverem “viradas”, ganha o jogador com mais pontos.

2.2 Telas

Além das telas do jogo temos também outras seções como o menu principal, ranking de jogadas prévias e os créditos. Aqui temos algumas imagens que ilustram isso:

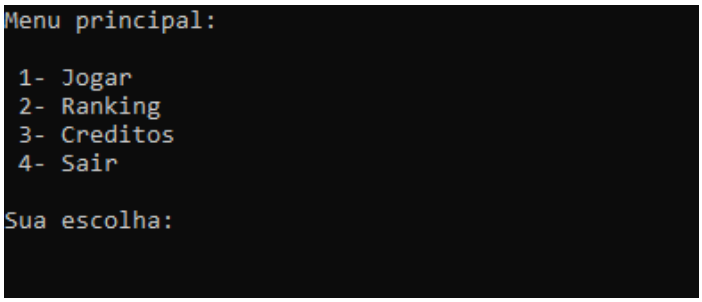

Foto: Tela de menu principal.

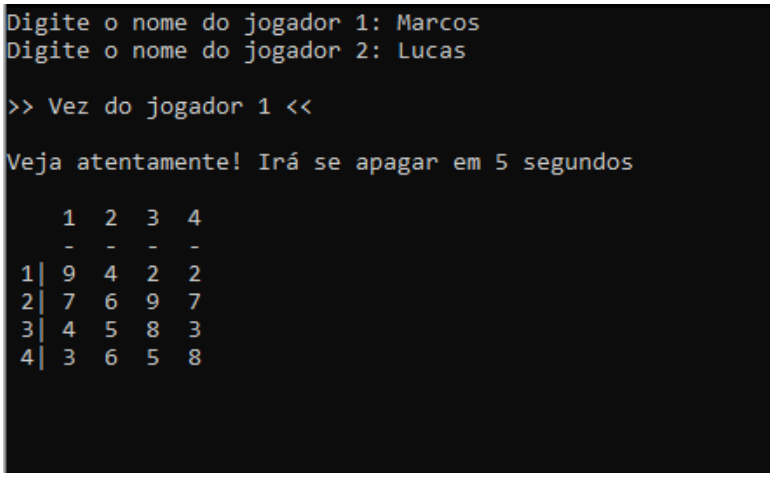

Foto: Após os jogadores escreverem seus nomes o jogo começa pelo primeiro jogador. Perceba que as linhas e colunas são enumeradas para melhor visualização da posição de cada carta.

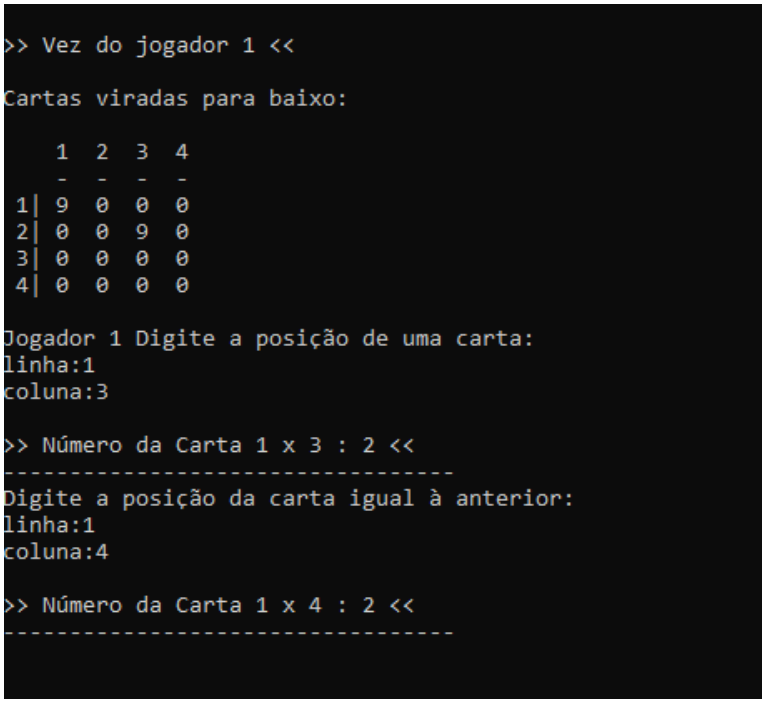

Foto: Aqui podemos observar as cartas viradas para baixo, sendo este o momento em que é permito que o jogador indique as posições das duas primeiras cartas.

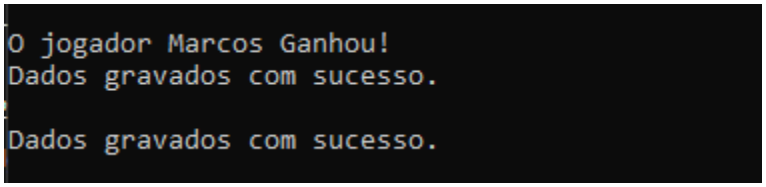

Foto: Após todas as cartas terem sido descobertas o jogo termina e os dados são gravados em arquivo.

2.3 Código e Dificuldades Encontradas

O código possui seu escopo principal, onde estão algumas funções mais importantes e a função main. Além disso, o projeto conta com um conjunto de bibliotecas, de onde importamos funções adjacentes necessárias para o funcionamento do jogo.

Começando pelo Menu principal, não houve segredos, sendo utilizado uma estrutura switch case para receber a escolha do usuário, sendo a maior dificuldade aqui a implementação da lógica.

Já uma questão importante que tivemos bastante atenção foi no tratamento dos dados recebidos do usuário. No Menu, não é possível para o usuário digitar um número fora das opções disponíveis e nem letras e símbolos.

Há também o tratamento de erros do usuário no escopo principal do jogo; aqui o usuário não pode digitar uma letra e nem um número de uma posição fora da matriz, caso o faça, será exibido uma mensagem de opção inválida. Além disso também não é possível que o usuário escolha a mesma posição da carta digitada anteriormente, ou de uma carta que já foi virada. A maior dificuldade encontrada aqui foi integrar todas as funções responsáveis no código, de forma que todas funcionassem independentemente e interrompessem o fluxo do jogo, ou seja, o jogador precisaria digitar de novo e a jogada anterior não seria computada.

Outra questão que nos tomou bastante tempo foi o requisito de salvamento do ranking em um arquivo de texto. Após algum tempo de pesquisa, descobriu-se que seria necessário a implementação de uma biblioteca para persistir esses dados em um arquivo. Para isso, foi desenvolvida um arquivo de extensão “.h” com o nome “saveRankingGame.h”, em que consistia na implementação de uma função para escrever as informações recebidas em um arquivo de formato “.txt”. Conforme a imagem a figura:

```
36  void escreverDadosNoArquivoDeRanking(struct Jogador jogador,struct tm *dataEHora){
37
38      int result;
39      pont_arq = fopen("rankingGame.txt", "a");
40
41      verificarAberturaDeArquivos(pont_arq);
42
43     //Gravando strings no arquivo
44      result = fprintf(pont_arq,"Jogador:%s\t%ipts\t salvo em: %i/%i/%i\t as:%i:%i:%i\n",jogador.nomeDoJogador,
45                                                          jogador.pontuacaoDoJogador,
46                                                          dataEHora->tm_mday,
47                                                          dataEHora->tm_mon + addValueMonth,
48                                                          dataEHora->tm_year + addValueYear,
49                                                          dataEHora->tm_hour,
50                                                          dataEHora->tm_min,
51                                                          dataEHora->tm_sec);
52
53      checarGravacaoDeDados(result);
54
55      fclose(pont_arq);
56  }
```

Foto: Função escreverDadosNoArquivoDeRanking()

A função apresentada acima consiste em duas entradas do tipo “struct”, uma para “Jogador” e outra para “dataEHora”; dentro do escopo há a declaração de uma variável do tipo “int”, chamada “result”, e um ponteiro do tipo “FILE”, de nome “pont_arq”, que recebe o retorno da função “fopen()”. A função “fopen()” pertence às dependências do compilador da linguagem C e nesse cenário serve para criar um

arquivo chamado “rankingGame.txt”, com o comando de escrever um novo dado (representado por “a”). Seguindo temos a variável “result”, que recebe o retorno da função "fprintf()", essa por sua vez tem como parâmetros o ponteiro do arquivo, o formato como será escrito os dados e as informações a serem escritas. Depois, temos a chamada da função “checarGravacaoDeDados()”, que serve para verificar se é possível abrir um arquivo existente ou criar um novo; por fim há também a função “fclose()” para fechar o arquivo em que foi gravado os dados. O formato da saída dessa função é representado pela figura 1.1:

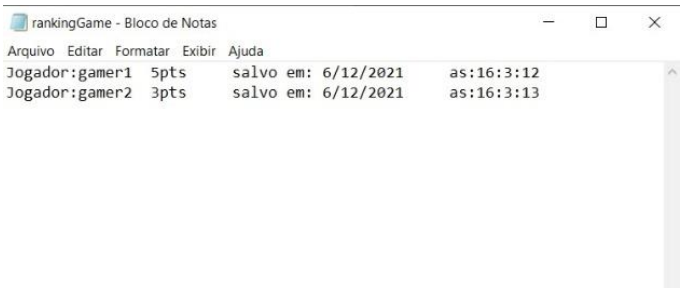

Foto: arquivo rankingGame.txt

Além disso, sendo uma consequência do processo de persistência de dados em um arquivo, foi criado uma outra função para ler os dados do arquivo “rankingGame.txt” e exibir na tela do console, esta com o nome de “lerDadosDoArquivosDeRanking()”. Dentro do escopo dessa função foi declarado um array do tipo “char” de nome “textoASerLidoDoArquivo” com tamanho de 255, já na linha 65 a chamada da função “imprimirDadosDoArquivo()”, que tem como parâmetros o array “textoASerLidoDoArquivo” e o ponteiro “pont_arq”. A figura mostra a implementação da função:

```
58  void lerDadosDoArquivosDeRanking(){
59
60      char textoASerLidoDoArquivo[255];
61
62      //abrindo o arquivo frase em modo "somente leitura"
63      pont_arq = fopen("rankingGame.txt", "r");
64
65      imprimirDadosDoArquivo(textoASerLidoDoArquivo,pont_arq);
66
67      fclose(pont_arq);
68
69      getch();
70
71  }
```

Foto: Função lerDadosDoArquivosDeRanking()

Falando um pouco sobre a parte central do código que é o jogo percebemos que era essencial a criação de uma matriz de alocação dinâmica, pois em c a única forma de você receber ou devolver um vetor ou uma matriz é através de um ponteiro. O primeiro obstáculo surge da necessidade de criarmos uma matriz aleatória com números pré-definidos (2,3,4,5,6,7,8,9) e de forma que haja exatamente dois números iguais em posições distintas. Para isso utilizamos a função “srand” no alcance de 0 a 3 para que essa função nos gerasse dois números aleatórios, que seriam a linha e a coluna da matriz a ser preenchida. Se essa posição já estivesse ocupada, outros números aleatórios seriam gerados, até que uma posição vazia fosse encontrada, caso contrário o próximo número da lista seria atribuído ao local. Após todos os espaços da matriz terem sido preenchidos a função retorna um ponteiro da matriz preenchida aletoriamente. Imagem:

```c
//Função para gerar uma matriz aletória
int *matriz_aleatoria(){
    //Cria 4 vetores com tamanho int
    int **matriz = malloc(sizeof(int*)*4);

    int sorteados[8]={2, 3, 4, 5, 6, 7, 8, 9};
    int linha, coluna, random_coluna, random_linha, index, x, apoio =0;


    //aloca para cada vetor 4 espacos do tamanho int
    for(index=0;index<4; index++){
        matriz[index] = malloc(sizeof(int)*4);
    }

    //preenche a matriz toda com 0s
    for(linha=0; linha<4; linha++){

    srand(time(NULL));
    for(index=0; index<8; index++){
        x=1;
        while(x==1){
            apoio++;
            random_coluna = rand() % 4;
            random_linha = rand()% 4;

            if (matriz[random_linha][random_coluna] == 0){

                matriz[random_linha][random_coluna] = sorteados[index];
                x=2;
            }
        }
        x=1;
        while(x==1){
            apoio++;
            random_coluna = rand() % 4;
            random_linha = rand()% 4;
            if (matriz[random_linha][random_coluna] == 0){

                matriz[random_linha][random_coluna] = sorteados[index];
                x=2;
            }
        }
    }

    return matriz;
}
```

Foto: Função para geração de uma matriz dinâmica e aleatória.

Tendo criado a função para a matriz partimos para o código do jogo. Na função principal temos a estrutura de repetição While sendo responsável pela partida, em que sua condição de parada é se a soma da pontuação dos jogadores chegar a 8. Em seguida, a estrutura de repetição For cumpre o papel de rodada, balizando qual a vez de qual jogador a partir do seu index. Para que pudéssemos tratar os dados recebidos do usuário de forma que a jogada só fosse contabilizada a partir de um valor correto outra estrutura while foi usada, em que o loop só era quebrado caso o jogador digitasse um valor possível. Por fim utilizamos estruturas condicionais simples para averiguar se o jogador acertou e para atribuir o valor da pontuação a cada jogador. Segue imagens do código:

```
//Esse while representa toda a partida
while(jogador1.pontuacao + jogador2.pontuacao < 8){

    //o for representa cada rodada
    for(index=1; index<3; index++){

        if(jogador1.pontuacao + jogador2.pontuacao == 8){
            break;

        }

        printf("\n>> Vez do jogador %i <<\n", index);
```

Foto: Estruturas de repetição principais.

```
while(1){
    printf("\nJogador %i Digite a posição de uma carta: \n", index);
    printf("linha:");
    scanf(" %s", &primeira_linha_digitada);
    printf("coluna:");
    scanf(" %s", &primeira_coluna_digitada);

    int linhaSeForDoTipoNumerico = verificarSeAEntradaNumerica(primeira_linha_digitada);
    int colunaSeForDoTipoNumerico = verificarSeAEntradaNumerica(primeira_coluna_digitada);

    primeiro_linha = converterEntradaDoJogadorParaTipoNumerico(linhaSeForDoTipoNumerico,primeira_linha_digitada);
    primeiro_coluna = converterEntradaDoJogadorParaTipoNumerico(colunaSeForDoTipoNumerico,primeira_coluna_digitada);
    //Se retornar 1 o usuário digitou uma posição impossível
    //Se retornar 2 o número já foi escolhido
    //se retornar 0 está ok, e ele quebra o laço
    apoio = conferir_se_poscao_possivel_e_valor_naoZero(primeiro_linha-1, primeiro_coluna-1, matriz_original);
    if(apoio == 1){
        printf("\nEssa posição não existe! \nDigite Novamente\n");
        sleep(2);
    }else if(apoio == 2){
        printf("\nEssa carta já foi escolhida! \nDigite Novamente\n");
        sleep(2);
    }else if(apoio==0){
        break;
    }
}
```

Foto: Caminho encontrado para tratar os dados recebidos do usuário.

```
// Conferir se os números digitados são iguais e adicionar a pontuação a cada jogador
system("cls");
if(matriz_original[segundo_linha-1][segundo_coluna-1] == matriz_original[primeiro_linha-1][primeiro_coluna-1]){
    printf("..................................... \n");
    printf("Você Acertou!! \n");
    //Caso Jogador 1
    if(index == 1){
        jogador1.pontuacao++;
        printf("-------------- \n");
        printf("Pontuação de jogador %s é: %i\n", nomeDoJogador1, jogador1.pontuacao);
        printf("-------------- \n");
        printf("Pontuação de jogador %s é: %i\n", nomeDoJogador2, jogador2.pontuacao);
        printf("..................................... \n");
        sleep(4);
        system("cls");

        //atribuindo os valores da outra matriz para a nova (virando as cartas nessa matriz)
        cartas_para_baixo[primeiro_linha-1][primeiro_coluna-1] = matriz_original[primeiro_linha-1][primeiro_coluna-1];
        cartas_para_baixo[segundo_linha-1][segundo_coluna-1] = matriz_original[segundo_linha-1][segundo_coluna-1];

        //Virando as cartas nessa matriz
        matriz_original[primeiro_linha-1][primeiro_coluna-1] = matriz_original[segundo_linha-1][segundo_coluna-1] = 0;

        //Repete a vez do jogador(index--) e evita que ele veja novamente as cartas(repetir=0)
        repetir = 0;
        index --;

    //Caso Jogador 2
    }else{
        jogador2.pontuacao++;
        printf("----------------------\n");
        printf("Pontuação de jogador %s é: %i\n", nomeDoJogador1, jogador1.pontuacao);
        printf("Pontuação de jogador %s é: %i\n", nomeDoJogador2, jogador2.pontuacao);
        printf("---------------------- \n ");
        sleep(4);
        system("cls");
```

Foto: Estrutura condicional para exibição e armazenamento da pontuação dos jogadores.

3 CONSIDERAÇÕES FINAIS

Após vários testes e correções de bugs a equipe ficou satisfeita com os resultados e com a solução dos problemas encontrados, além de reconhecer a importância do projeto para o processo de aprendizagem. Esperamos que nosso Jogo da Memória seja aprovado e possa servir de diversão para outras pessoas, assim como esse projeto foi para nós.