SERVIÇO PÚBLICO FEDERAL · MINISTÉRIO DA EDUCAÇÃO
UNIVERSIDADE FEDERAL DE VIÇOSA · UFV
CAMPUS FLORESTAL

Trabalho 1 - AEDS 1

Simulador do jogo Paciência utilizando tipos abstratos de dados e listas encadeadas

EDMARCOS MENDES PINTO FILHO [05387]
DAVI DE SOUZA FERREIRA DO CARMO [5368]
LUCAS CURTY RODRIGUES MATHEUS [5793]

Florestal - MG
2023

Sumário

1. Introdução

O desenvolvimento de um sistema computacional, muito além da lógica e codificação, necessita de uma boa arquitetura e uma boa forma de representar, tratar e manipular os dados. Dessa forma, o presente projeto objetiva a aplicação dos conhecimentos adquiridos em sala de aula sobre Tipos Abstratos de Dados (TADs), para modelar e desenvolver uma versão de terminal do jogo Paciência (ou Solitário), contando também com o uso de listas lineares, implementadas sobre a estrutura Lista Encadeada, para construir algumas das entidades necessárias no fluxo da aplicação.

2. Organização

A estrutura do projeto possui um diretório principal, em que se encontram os arquivos git (como Readme e .gitignore), a pasta src, onde se encontra de fato o código desenvolvido, a pasta docs, em que se encontra a documentação, e a pasta tests, onde há um exemplo de entrada para o modo arquivo. Na Figura 1 é possível observar a estrutura mencionada.

Quanto a organização da pasta src, devido a dificuldades técnicas de configuração do caminho dos arquivos que se deram com o uso das ferramentas escolhidas, foi necessário manter todos os arquivos .c e .h em um mesmo diretório para que o projeto fosse compilado. Embora saiba-se que o mais recomendado é que o programa estivesse separado em módulos, cada um com seu respectivo .h e .c.

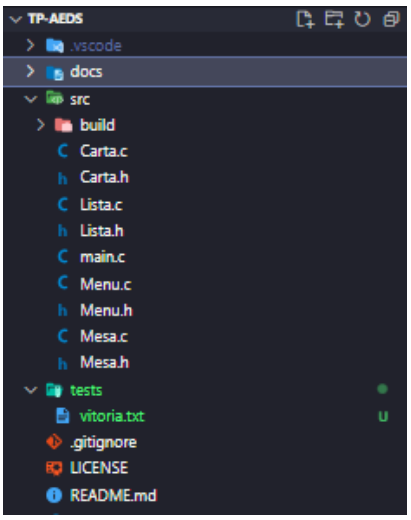

Figura 1 - Repositório do projeto.

3. Desenvolvimento

A construção do projeto envolveu tanto a compreensão de como funciona o jogo Paciência e suas regras de negócio existentes, quanto a modelagem dos TADs necessários para seu funcionamento.

Para isso foi necessário criar a definição e a implementação de um TAD Carta, com todas as operações para criar e modificar uma carta, um TAD Lista, que foi criado utilizando-se do conceito de lista encadeada, para representar as diferentes colunas de cartas presentes no jogo, contendo todas as operações envolvendo a obtenção e movimentação dos itens, e um TAD Mesa, que representaria a mesa onde ocorrer à partida e possui as principais operações de controle e movimentação.

Além disso, um subsistema de menu também precisou ser criado para gerenciar tanto o processo interativo do usuário para com o jogo,quanto um modo automático de inserção por arquivo.

3.1 Transferência de cartas

Uma funcionalidade importante para o desenvolvimento do projeto foi a de transferir cartas, pois ela é necessária em todas as operações que o usuário realiza sobre a mesa do jogo.

A função possuiu o objetivo de transferir as n últimas cartas de uma lista encadeada para outra, assim removendo as últimas da lista de origem e

adicionando-as na lista de destino. A função percorre uma lista até chegar a primeira célula que será transferida para a próxima, além de armazenar a célula anterior, que será a última da primeira lista, assim célula por célula é adicionada, mantendo a ordem primária dos elementos.

```c
void transferirCartas(Lista* listaOrigem, Lista* listaDestino, int quantidade) {
    int tamanhoOrigem = (getTamanho(listaOrigem) + 1);

    if ((quantidade > (tamanhoOrigem - 1)) || quantidade <= 0) {
        printf("Quantidade inválida de cartas para mover.\n");
        return;
    }
    int posicaoPrimeiraCarta = tamanhoOrigem - quantidade;
    Celula* anterior = NULL;
    Celula* atual = listaOrigem->primeira;
    for (int i = 0; i < posicaoPrimeiraCarta; i++) {
        anterior = atual;
        atual = atual->proxima;
    }

    for (int i = 0; i < quantidade; i++) {
        if(atual == NULL){
            break;
        }
        Celula* proxima = atual->proxima;
        Carta carta = atual->carta;
        addCartaAoTopo(listaDestino, &carta);
        free(atual);
        atual = proxima;
    }

    if (anterior) {
        anterior->proxima = atual;
    } else {
        listaOrigem->primeira = atual;
    }

    listaOrigem->ultima = (anterior != NULL) ? anterior : listaOrigem->primeira->proxima;

    if(verificarListaVazia(listaOrigem)) {
        listaOrigem->primeira->proxima = NULL;
        listaOrigem->ultima->proxima = NULL;
    }
}
```

Figura 2 - Lista.c - TransferirCartas().

3.2 Exibir lista

Outra parte importante para o desenvolvimento do trabalho, foi a função de exibir lista, visto que é a principal para a exibição de todo o processo do jogo.

A função possuiu como objetivo a exibição das cartas presentes nas listas encadeadas, podendo ter três formas diferentes de serem apresentadas. A primeira é de toda as cartas presentes na lista serem exibidas para cima, ou seja, é exibido o naipe e valor da carta; outra demonstração da lista é de ser exibida apenas a última carta, ou seja toda as cartas da lista, excluindo a última é apresentada virada para

baixo; por fim, a última representação da lista de carta é de todas as cartas serem exibidas para baixo

```c
void exibirLista(Lista* lista, Visualizar visualizar){
    if(verificarListaVazia(lista)){
        printf("Lista Vazia");
    } else{
        if(visualizar == TODOS){
           Celula* aux = lista->primeira->proxima;
            while (aux != NULL){
                    aux->carta.posicao = CIMA;
                exibirCarta(aux);
                aux = aux->proxima;
            }
            free(aux);
        }
        else if (visualizar == SOMENTE_TOPO){
            Celula* aux = lista->primeira->proxima;
            while (aux != NULL){
                if(aux->proxima == NULL){
                    aux->carta.posicao = CIMA;
                }
                exibirCarta(aux);
                aux = aux->proxima;
            }
            free(aux);
        } else {
            Celula* aux = lista->primeira->proxima;
            while (aux != NULL){
                aux->carta.posicao = BAIXO;
                exibirCarta(aux);
                aux = aux->proxima;
            }
            free(aux);
        }
    }
}
```

Figura 3 - Lista.c - exibirLista().

```
Baralho: [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ] [ ]
```

Figura 4 - Exibição da lista com todas as cartas viradas para baixo.

```
Descarte: [♦ 5] [♥ 10] [♠ 4] [♦ 1] [♦ 4] [♣ 5] [♣ 11] [♥ 8] [♦ 8]
```

Figura 5 - Exibição da lista com todas as cartas viradas para cima.

Figura 6 - Exibição da lista com apenas a última carta virada para cima.

4. Compilação e Execução

Para o desenvolvimento do trabalho prático, foram utilizados o Sistema Operacional Windows 10/11, instalação do GCC para o Windows MSYS2 MSYS, com o seguinte comando no terminal do mesmo "pacman -S --needed base-devel mingw-w64-ucrt-x86_64-toolchain", o editor de texto Visual Studio Code, junto com as seguintes extensões que o compõem com o intuito de proporcionar a compilação e execução no mesmo:

- C/C++ Runner, franneck94, utilizado para a compilação e execução de múltiplos arquivos .c;

- C/C++, Microsoft, utilizado para auxiliar na identificação e correção de erros no desenvolvimento do projeto.

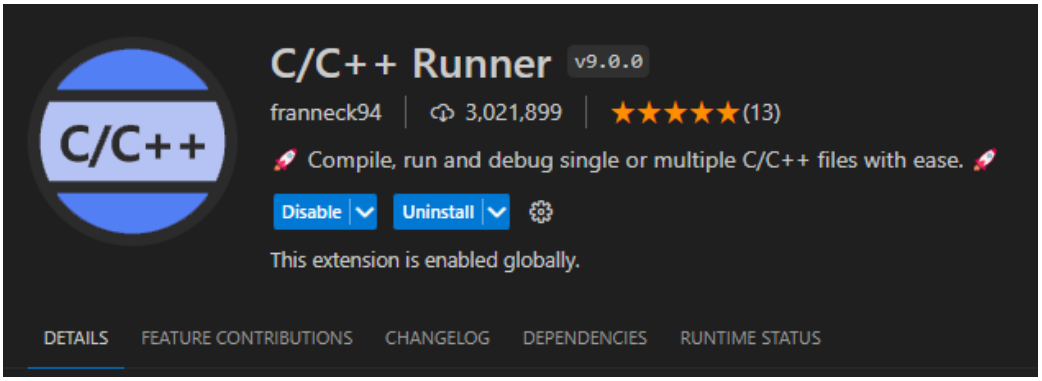

Figura 7 - Extensão VS Code: C/C++ Runner.

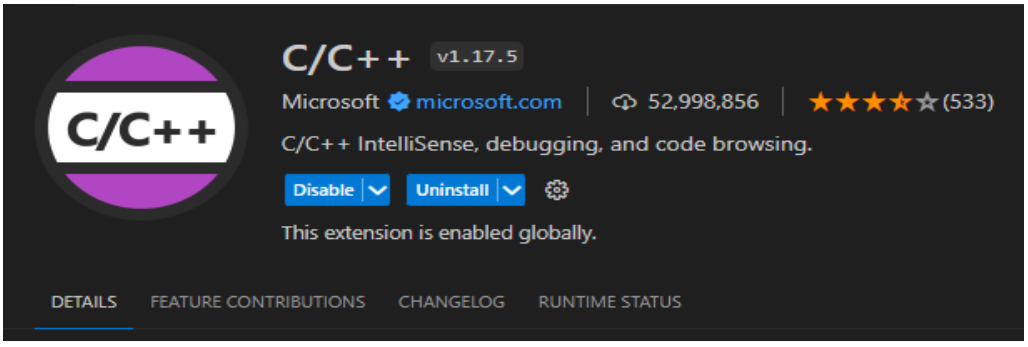

Figura 8 - Extensão VS Code: C/C++.

Para o desenvolvimento de projetos na linguagem C no sistema operacional do Windows é necessário a instalação de compilador, visto que o sistema operacional em si não vem definido de fábrica para o desenvolvimento, desta forma foram seguidos os passos do tutorial C++ e MinGW disponibilizados pelo próprio Visual Studio Code [5]. Aliado com isso, foi necessária a instalação e configuração da extensão “C/C++ Runner”, com o intuito de executar múltiplos arquivos da linguagem C. Para a execução de arquivos em lote, é necessário seguir os seguintes passos com a extensão supra exposta.

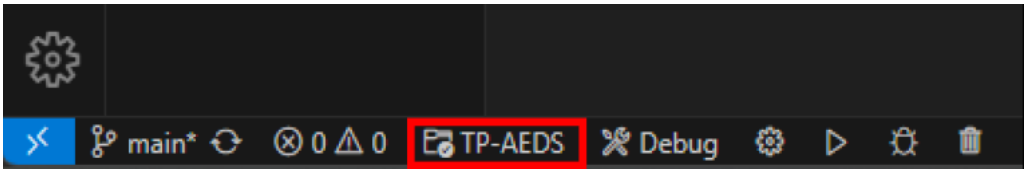

Figura 9 - Selecionar repositório que está nos arquivos da linguagem C.

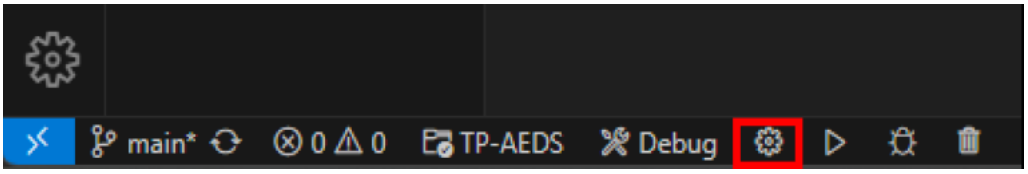

Figura 10 - Compilar o projeto.

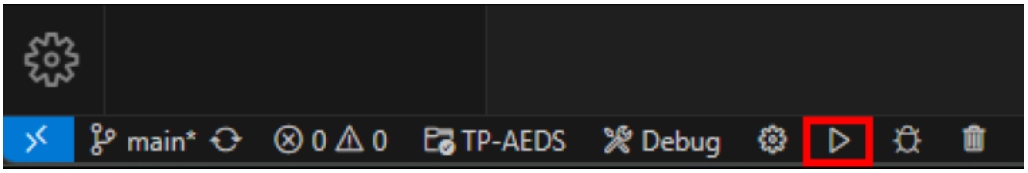

Figura 11 - Executar o projeto.

5. Resultados

Após a realização da atividade proposta, foi possível construir a aplicação solicitada com todas as suas funcionalidades e seus dois modos de operação, interativo e por leitura de arquivo. O modo é definido pelo usuário ao iniciar o jogo.

No modo interativo, é apresentado um menu de opções, no qual o jogador pode definir qual dos seis movimentos ele realizará a seguir, sempre recebendo feedback do novo estado da mesa após a jogada anterior.

```
Bem vindo ao Paciência!
 Escolha uma opção para começar:

 1 - modo interativo
2 - leitura de arquivo
1

Modo interativo! Escolha uma das opções abaixo!

Pontuação atual: 0
Baralho: [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
Descarte: Lista Vazia
Base 0: Lista Vazia
Base 1: Lista Vazia
Base 2: Lista Vazia
Base 3: Lista Vazia
Coluna 0: [♦ 5]
Coluna 1: [  ] [♣ 1]
Coluna 2: [  ] [  ] [♥ 13]
Coluna 3: [  ] [  ] [  ] [♦ 3]
Coluna 4: [  ] [  ] [  ] [  ] [♥ 8]
Coluna 5: [  ] [  ] [  ] [  ] [  ] [♠ 3]
Coluna 6: [  ] [  ] [  ] [  ] [  ] [  ] [♥ 2]
1 - Mover carta entre colunas do tableau.
2 - Mover do tableau para uma base.
3 - Mover da base para o tableau.
4 - Comprar uma carta
5 - Mover do descarte para a base
6 - Mover do descarte para o tableau
7 - Sair
```

Figura 12 - Tela inicial e modo interativo

```
De qual coluna deseja mandar uma carta para a base?
1
Pontuação atual: 15
Baralho: [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ] [  ]
Descarte: Lista Vazia
Base 0: Lista Vazia
Base 1: [♣ 1]
Base 2: Lista Vazia
Base 3: Lista Vazia
Coluna 0: [♦ 5]
Coluna 1: [♠ 7]
Coluna 2: [  ] [  ] [♥ 13]
Coluna 3: [  ] [  ] [  ] [♦ 3]
Coluna 4: [  ] [  ] [  ] [  ] [♥ 8]
Coluna 5: [  ] [  ] [  ] [  ] [  ] [♠ 3]
Coluna 6: [  ] [  ] [  ] [  ] [  ] [  ] [♥ 2]
1 - Mover carta entre colunas do tableau.
2 - Mover do tableau para uma base.
3 - Mover da base para o tableau.
4 - Comprar uma carta
5 - Mover do descarte para a base
6 - Mover do descarte para o tableau
7 - Sair
```

Figura 13 - Estado da mesa após escolher a opção 2.

Já no modo de leitura de arquivo, é requisitado ao usuário apenas o caminho para o arquivo, e apresenta-se o primeiro e o último estado da mesa, após a execução da lista de operações.

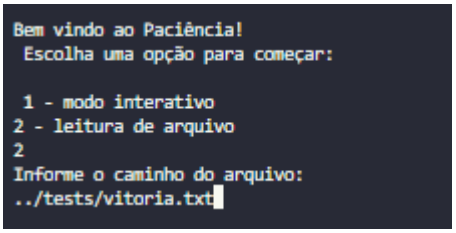

Figura 14 - Início do modo de arquivo

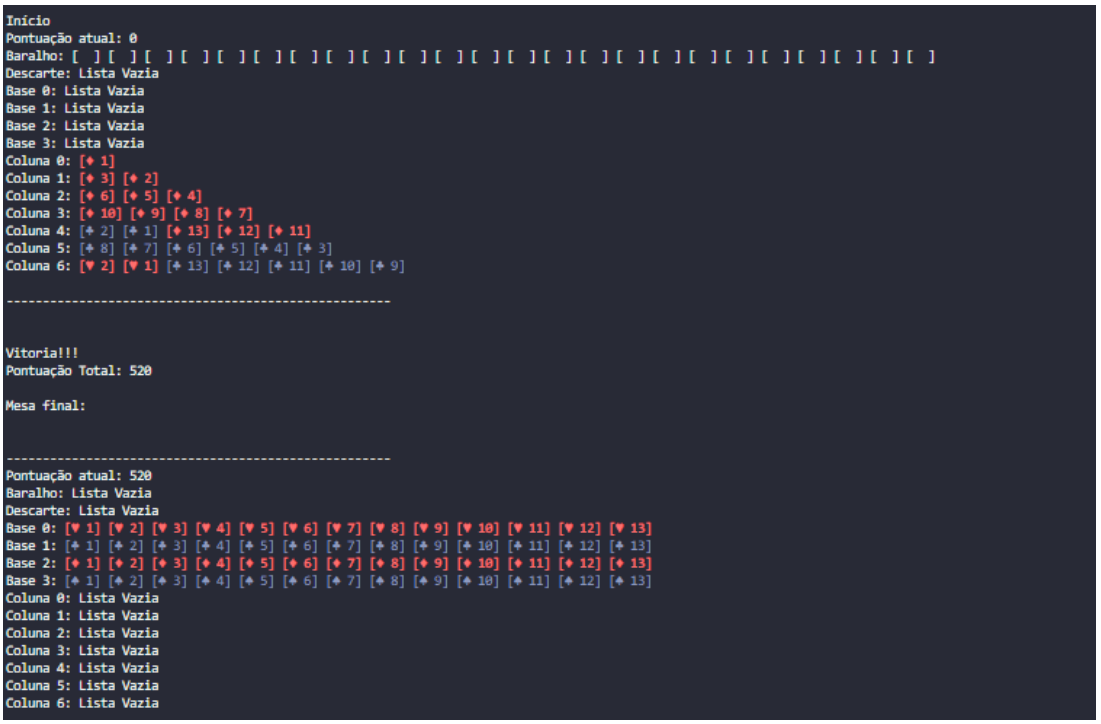

Figura 15 - Exibição da mesa antes e após à realização das operações

Vale também pontuar que a verificação da condição de vitória ocorre a cada jogada tanto no modo interativo, quanto de arquivo, e é atingida quando as quatro bases estão completas.

A aplicação foi elaborada de forma a garantir a melhor otimização para o usuário, assim, foram criadas diversas verificações para que não ocorressem erros e exceções devido a alguma entrada equivocada, mantendo assim, a total liberdade do usuário sem comprometer a experiência do jogo, além de apresentar o sistema de forma familiar e intuitiva para o usuário, desenvolvendo a melhor usabilidade possível.

6. Conclusão

Diante do que foi apresentado, percebe-se que o desenvolvimento do trabalho contribuiu para um melhor aproveitamento e absorção dos conteúdos abordados em sala de aula, visto que a modularização dos tipos abstratos de dados, bem como a implementação de listas encadeadas, foram fundamentais para o cumprimento das exigências e a finalização do jogo com todas as suas funcionalidades.

7. Referências

[1] Github. Disponível em: <https://github.com/> Último acesso em: 30 de setembro de 2023.

[2] Visual Studio Code. Disponível em: <https://visualstudio.microsof t.com/pt-br/> Último acesso em: 25 de setembro de 2023.

[3] Linguagem C Descomplicada. Disponível em: <https://programacaodescomplicada.wordpress.com/> Último acesso em: 30 de setembro de 2023.

[4] Stack overflow. Disponível em: <https://stackoverflow.com/> Último acesso em: 28 de setembro de 2023.

[5] Get Started with C++ and MinGW-w64 in VIsual Studio Code. Disponível em: <https://code.visualstudio.com/docs/cpp/config-mingw> Último acesso em: 30 de setembro de 2023.