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

# Trabalho Prático 0
# Projeto e Análise de Algoritmos

Pâmela Lúcia Lara Diniz [5898]

Florestal - MG
2025

**Sumário**

## 1.Introdução

O projeto descrito neste documento trata da implementação de um quadro gerador de obras de arte a partir da utilização de valores aleatórios que poderão determinar a configuração das figuras geradas pela saída do programa. Saída esta que poderá ser parcialmente influenciada pelo usuário a partir de um menu interativo em que será possível selecionar diferentes tipos de figuras para a montagem da arte, além da quantidade de desenhos que serão mostrados na tela. O principal objetivo da implementação é colocar em prática as habilidades de programação com a linguagem C e também o desenvolvimento de uma aplicação funcional que implemente uma obra criativa e exclusiva atendendo aos critérios descritos na especificação do trabalho.

## 2.Organização

A organização do projeto pode ser observada na figura 1. A pasta *include* é o local em que se encontram os arquivos header dos *TADs* criados, já o diretório *src* é responsável por armazenar arquivos .c que contém a implementação das funcionalidades utilizadas e também o arquivo principal da aplicação, *main.c*. Já a pasta *bin* guarda o executável gerado após a compilação do programa. Por fim, no diretório raiz há um pdf que documenta informações relevantes sobre o projeto e também um arquivo *makefile* para auxiliar o processo de compilação que será detalhado posteriormente neste documento.

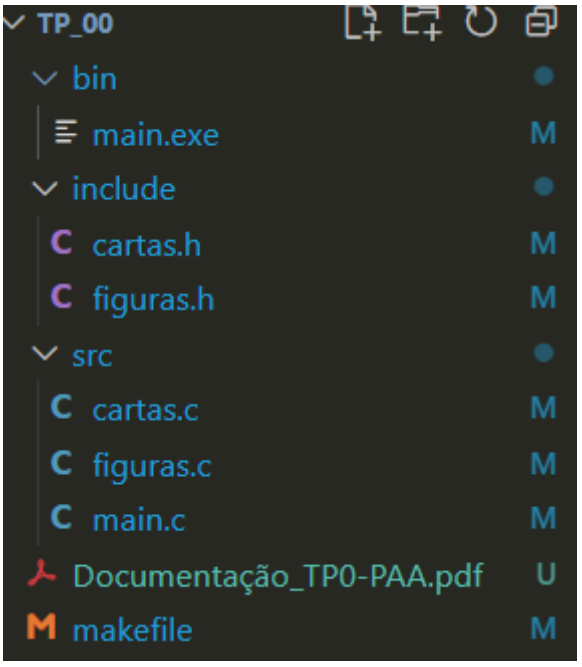


Figura 1 - organização do projeto

## 3.Desenvolvimento

Nesta seção serão destacados os principais pontos referentes ao processo de desenvolvimento do projeto, que se deu de maneira linear. Inicialmente foram implementadas as opções que lidam com figuras genéricas haja vista que se mostraram mais fáceis de lidar, isso também possibilitou que fosse feito um estudo e testes em relação à coloração das figuras e uma vez que tudo isso foi destrinchado, os conhecimentos adquiridos serviram como base para uma implementação mais fluida da operação cinco que se mostrou mais complexa, principalmente para que fosse feita a inserção das imagens (cartas) na matriz, respeitando as verificações de colisão.

### 3.1 Figuras genéricas

Para a execução das opções 1 a 4 do menu, que se referem à geração de obras de arte a partir de asteriscos, cruz e a letra x, além da possibilidade de mesclagem dessas figuras, foi utilizado um tipo abstrato de dados *tipoFigura* e desenvolvidas algumas funções para promover a sua utilização. Tais definições podem ser encontradas na figura 2.

```c
typedef struct tipoFigura{
    char caractere;
    char tipoFig;
    int idFigura;
} tipoFigura;

void criaQuadroVazio (tipoFigura quadro[LINHAS][COLUNAS]);
tipoFigura* inicializaX(int id);
tipoFigura* inicializaPonto(int id);
tipoFigura* inicializaCruz(int id);

void insereAsterisco(int quantidade,tipoFigura quadro[LINHAS][COLUNAS],int* id);
void insereCruz(int quantidade,tipoFigura quadro[LINHAS][COLUNAS],int* id);
void insereX(int quantidade,tipoFigura quadro[LINHAS][COLUNAS],int* id);

int sorteiaQuantFiguras();
void sorteiaFigura (int quantidade,tipoFigura quadro[LINHAS][COLUNAS],int* id);

void sorteiaCores(int quant, int vetorCores[]);
void selecionaCor(int id);

void resetarQuadro(tipoFigura quadro[LINHAS][COLUNAS], int *id);
void imprimeQuadro(int quantFiguras, tipoFigura quadro[LINHAS][COLUNAS]);
```

Figura 2 - implementação do TAD figuras

O primeiro bloco de funções retratado na figura 2 é responsável por inicializar as estruturas necessárias para a aplicação, sendo a função *criaQuadroVazio* responsável por inicializar uma matriz de proporções 20x80 com apenas as bordas preenchidas, que é justamente a estrutura que irá representar o quadro, essa estrutura será do tipo *tipoFigura*, ou seja, cada posição da matriz representa uma espécie de pixel, armazenando uma struct que carrega informações como o caractere daquela posição, o tipo de figura que aquele caractere está representando assim como um id que ajuda a diferenciar figuras de mesmo tipo. Quanto às funções de inicialização das figuras, elas realizam a alocação de memória além da atribuição de valores aos campos a depender das características de cada uma.

Para a inserção dos elementos na matriz, foram utilizadas insereCruz, insereX e insereAsterisco. Cada uma dessas funções atende às características de sua respectiva figura mas todas funcionam de maneira similar. A figura 3 representa a inserção da cruz e observa-se que é feito o sorteio das posições i e j da matriz para representar o local em que o centro da figura será inserido, a partir disso são feitas algumas verificações para garantir que não haverá colisão com alguma outra figura já inserida no quadro ou com as bordas e o elemento é adicionado ao quadro em caso afirmativo. Esse procedimento é repetido *n* vezes, a depender da quantidade daquele tipo de figura que deve ser inserido no quadro. Em relação às outras operações de inserção, o funcionamento se dá de maneira similar e se difere apenas no conjunto de caracteres inseridos.

```
void insereCruz(int quantidade,tipoFigura quadro[LINHAS][COLUNAS],int* id){

    int count=0;
    tipoFigura* pixel;

    while(count<quantidade){
        int i = rand()%LINHAS;
        int j = rand()%COLUNAS; //sorteia a posição em que a figura será inserida

        if (quadro[i][j].caractere==' ' && quadro[i][j-1].caractere==' ' && quadro[i][j+1].caractere==' '
            && quadro[i+1][j].caractere==' ' && quadro[i-1][j].caractere==' ' ){  //verifica se não há colisões

            pixel = inicializaCruz(*id);
            quadro [i][j]=*pixel;  //atribui cada pixel ao quadro
            quadro[i][j-1]=*pixel;
            quadro[i][j+1]=*pixel;
            quadro[i+1][j]=*pixel;
            quadro[i-1][j]=*pixel;
            count+=1;
            (*id)++;
        }
    }

}
```

Figura 3 - implementação da função de inserção da cruz.

Ademais, a função *sorteiaQuantFiguras* é utilizada para retornar um número aleatório no intervalo de 1 a 100 e é utilizada no caso de o usuário indicar uma quantidade aleatória de figuras através do menu. Já a função *sorteiaFigura* está relacionada à opção 4 do menu, que deve gerar figuras aleatoriamente até atingir a

quantidade indicada, então, ela faz *n* sorteios para garantir a inserção aleatória das figuras até atingir a quantidade total desejada.

Além disso, uma implementação extra adicionada às quatro primeiras opções do menu foi a aleatoriedade das cores de cada figura e esse procedimento é realizado pelas funções *sorteiaCores* e *selecionaCor.* Então, durante a impressão do quadro, é feito o sorteio entre oito possibilidades de cores para cada figura contida na arte e sempre que é feita a impressão dessa figura, seu id, que já está associado à uma posição de uma lista de cores obtida pela função *sorteiaCores*, é utilizado para recuperar a coloração correta daquela imagem e isso pode ser observado na figura 4 que retrata a função de impressão do quadro.

```
void imprimeQuadro(int quantFiguras, tipoFigura quadro[LINHAS][COLUNAS]){

    int listaCores[quantFiguras];

    sorteiaCores(quantFiguras,listaCores); //determina uma cor para cada id de figura

    for (int i=0;i<LINHAS;i++){
        for (int j=0;j<COLUNAS;j++){
            if (quadro[i][j].tipoFig=='n'){
                printf("%c",quadro[i][j].caractere);
            }
            else{
                selecionaCor(listaCores[quadro[i][j].idFigura]); //recupera a cor associada ao id
                printf("%c",quadro[i][j].caractere);
                printf("\033[0m");
            }
        }
    printf("\n");

    }
}
```

Figura 4 - implementação da função que imprime o quadro no terminal.

Por fim, há uma função *resetarQuadro* que é responsável por reiniciar a matriz que guarda as figuras para que uma nova arte possa ser gerada.

### 3.1 Mesa de cartas

A arte escolhida para ser implementada neste projeto foi a representação de uma mesa com cartas de baralho em que os naipes e valores das cartas são definidos aleatoriamente. Além disso, para maximizar a quantidade de cartas que poderiam ser distribuídas, foi estabelecida uma regra de que elas deveriam permanecer alinhadas, com a possibilidade de no máximo três linhas de cartas, entretanto, dentro dessa limitação das linhas, as cartas podem ser posicionadas em diferentes colunas para garantir a aleatoriedade de posições. Para justificar essa configuração, foi definido que cada linha de cartas representa a “mão” de um mesmo jogador, logo, a arte pode representar a distribuição de cartas entre 1,2 ou 3 jogadores, a depender da quantidade de figuras informadas pelo usuário que pode

ser no máximo 30 para garantir uma execução eficiente da aplicação. A distribuição é feita tentando sempre maximizar a quantidade de jogadores a partir de uma divisão aproximadamente igualitária de cartas e considerando uma quantidade mínima de 3 cartas para cada um deles.

Para a implementação dessa funcionalidade, dois tipos abstratos de dados foram criados para atender às suas necessidades uma vez que envolve operações mais complexas e pode ser observado na figura 5.

```
typedef struct Carta{
    char tipoCarta;
    char cartaDesenho[6][8];
} Carta;

typedef struct Mesa{
    char caractere;
    Carta* carta;
} Mesa;

void mesaVazia(Mesa mesa[LINHAS][COLUNAS]);
Carta* inicializaPaus(char simbolo);
Carta* inicializaCopas(char simbolo);
Carta* inicializaOuro(char simbolo);
Carta* inicializaEspada(char simbolo);

Carta** sorteiaCartas(int quantidade);
int sorteiaQuantCartas();

int* divideCartas(int quantidade, int* qntJogadores);
void insereCartas(Carta** deck, int quantidade, Mesa mesa[LINHAS][COLUNAS], int* divisaoCartas,int* qntJogadores);
int desviaColisao(int colunas, int usadas[], int qtdUsadas);
void desenhaCarta(Mesa mesa[LINHAS][COLUNAS], Carta* carta, int linha, int coluna);


void imprimeMesa(Mesa mesa[LINHAS][COLUNAS]);
void resetarMesa(Mesa mesa[LINHAS][COLUNAS], Carta*** deck, int** divisaoCartas,int qntCartas);
```

Figura 5 - implementação do TAD carta.

Para realizar a operação de gerar uma obra de arte original foi definida uma *struct* Mesa que será justamente o tipo da matriz que armazenará as figuras, cada posição dessa matriz irá guardar o seu respectivo caractere além de um ponteiro para uma carta, que se trata de outra struct que armazena o tipo da carta (paus, espada, ouro, copas) além de uma matriz compacta que tem o desenho da carta. Esse ponteiro será nulo naquelas posições da mesa em que não há nada desenhado.

De maneira similar ao que foi descrito para as operações anteriores, existe uma função responsável por inicializar a matriz vazia (*mesaVazia*), com exceção das bordas que já serão automaticamente preenchidas.

A função *sorteiaCartas* é a responsável por definir aleatoriamente os naipes e símbolos das cartas que irão compor a obra de arte sem que haja repetições (mesmo naipe e símbolo), isso é feito por meio de estruturas *switch case*, inicialmente para definir um símbolo sorteado dentre 12 possibilidades e posteriormente para chamar a inicialização a depender do naipe. Cada carta inicializada é armazenada em um vetor *deck* que ao final da função é retornado.

Em relação às funções de inicialização de cada carta, elas são responsáveis por alocar memória para cada uma e atribuir os seus respectivos valores assim como o desenho, como pode ser observado na figura 6 para a inicialização de uma carta do naipe “paus”, nesse caso, o símbolo anteriormente definido é recebido como parâmetro e incluído na posição correta da ilustração da carta

```
Carta* inicializaPaus(char simbolo) {

    Carta* item = malloc(sizeof(Carta));
    item->tipoCarta = 'P';

    char desenho[6][8] = {
        " sssss ",   //'s' indica borda superior/inferior, representada por '_'
        "l  _  l",   //'l' indica borda lateral, representada por '|'
        "l ( ) l",
        "l(_'_)l",
        "l  |  l",
        "lsssssl"
    };

    desenho[1][1] = simbolo;

    memcpy(item->cartaDesenho, desenho, sizeof(desenho));

    return item;
}
```

Figura 6 - implementação da função que inicializa uma carta do naipe “paus”.

A função *divideCartas* é responsável por determinar a distribuição de cartas no quadro, seguindo os critérios citados anteriormente de maximizar a quantidade de linhas e realizar uma divisão aproximadamente igual de cartas entre cada uma delas. Então, ao final de sua execução teremos a quantidade de linhas de cartas do quadro e também a quantidade de cartas em cada linha. A partir disso, a função *insereCartas* é utilizada para auxiliar o processo de inserção de todas as cartas na matriz a partir das informações obtidas anteriormente. Além disso, ela chama a função *desviaColisao* que irá encontrar as melhores colunas para posicionar cada carta de forma a incluir todas elas, caso essa verificação ultrapasse 500 tentativas para a inserção de uma mesma carta, toda a matriz é reiniciada e novas posições de coluna são buscadas para todas as cartas de forma a tentar encontrar uma configuração de posicionamentos mais eficiente. Uma vez que uma posição disponível é encontrada, a função *desenhaCarta* faz a inserção da carta na matriz que representa o quadro.

Uma vez que todas as cartas foram inseridas, a função *imprimeMesa* pode ser chamada para exibir no terminal a configuração final do quadro gerado, processando também as cores de cada caractere, que adotarão tons diferentes para

representar os naipes das cartas, sendo vermelho para copas e ouro e cinza escuro, no caso de espada e paus.

Por fim, têm-se a função *resetarMesa* que é responsável por liberar todas as memórias alocadas e desfazer os valores utilizados durante a geração do quadro.

### 3.3 Main

O arquivo principal main.c é responsável por gerenciar todas as funções desenvolvidas a partir do processo de interação com o usuário para que o funcionamento da aplicação se dê de maneira correta. Para isso, foi desenvolvido um menu que representa a interface de opções que o usuário irá utilizar para interagir com o programa e a partir das suas escolhas e dados informados, funções referentes a execução dessas partes serão chamadas para que seja criada uma obra de arte condizente às escolhas feitas.

## 4 Compilação e execução

Para executar o programa foi criado um arquivo Makefile que irá auxiliar nesse processo (figura 7).

```
compile: src/cartas.c src/figuras.c src/main.c
    @gcc src/cartas.c src/figuras.c src/main.c -Iinclude -Wall -Wextra -g -o bin/main.exe

run:
    @bin/main.exe
```

Figura 7 - arquivo Makefile da aplicação.

Logo, uma vez que se esteja localizado no mesmo diretório em que o *Makefile* se encontra, ou seja, na pasta *tp_00*, basta executar os comandos *make compile* para realizar a compilação e *make run* para fazer a execução da aplicação no caso de sistema operacional linux. Para o sistema windows, os comandos *mingw32-make compile* e *mingw32-make run* realizam respectivamente a compilação e execução do código.

## 5 Resultados

### 5.1 Figuras genéricas

Uma vez que o desenvolvimento da aplicação foi finalizado e a sua execução realizada, houve a possibilidade de testar o seu funcionamento. A figura 8 mostra a interface do programa que solicita informações do usuário.

```
PROGRAMA GERADOR DE OBRA DE ARTE:
================================
Escolha o tipo de figura basica a ser usada para criar a obra:
1 - asterisco simples.
2 - simbolo de soma com asteriscos.
3 - letra X com asteriscos.
4 - figuras aleatorias
5 ou qualquer outro numero - opcao de obra de arte criada pelo aluno
================================
Digite o tipo de figura basica desejada:5
Digite a quantidade de figuras (menor ou igual a zero para aleatorio):15
```

Figura 8 - interface do menu da aplicação.

Já as figuras 9,10,11 e 12 retratam a utilização das opções 1 a 4 do menu, mostrando duas artes diferentes geradas a partir de mesmos valores escolhidos para demonstrar a aleatoriedade de cores e posições das figuras geradas, demonstrando um resultado satisfatório que atingiu às expectativas da especificação.

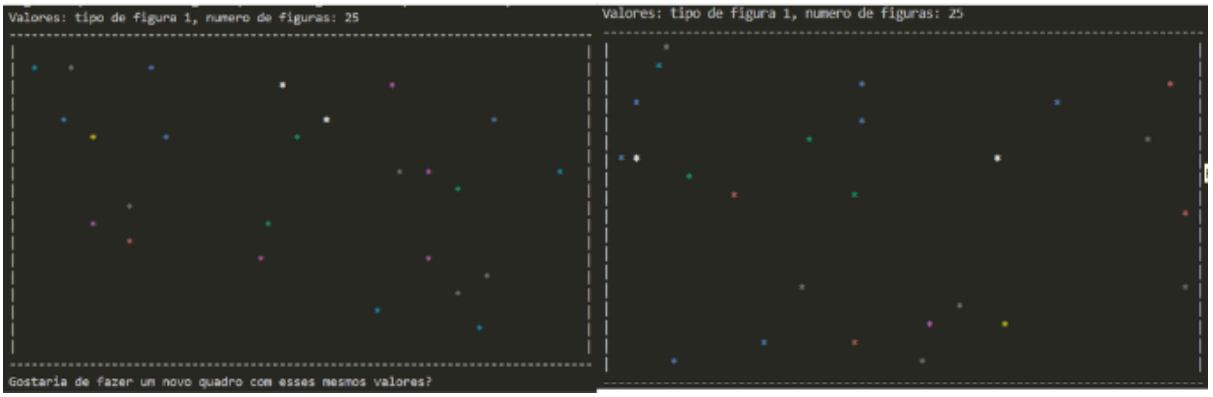


Figura 9 - artes geradas pela opção 1.

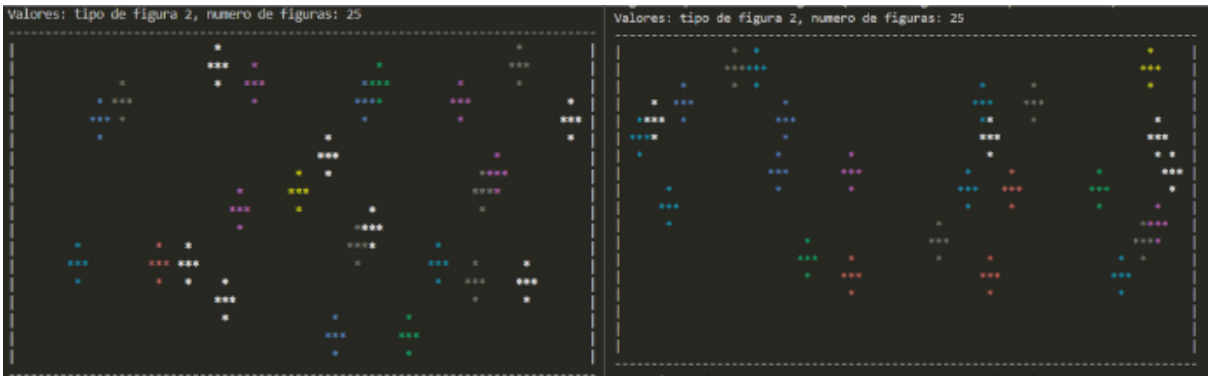


Figura 10 - artes geradas pela opção 2.

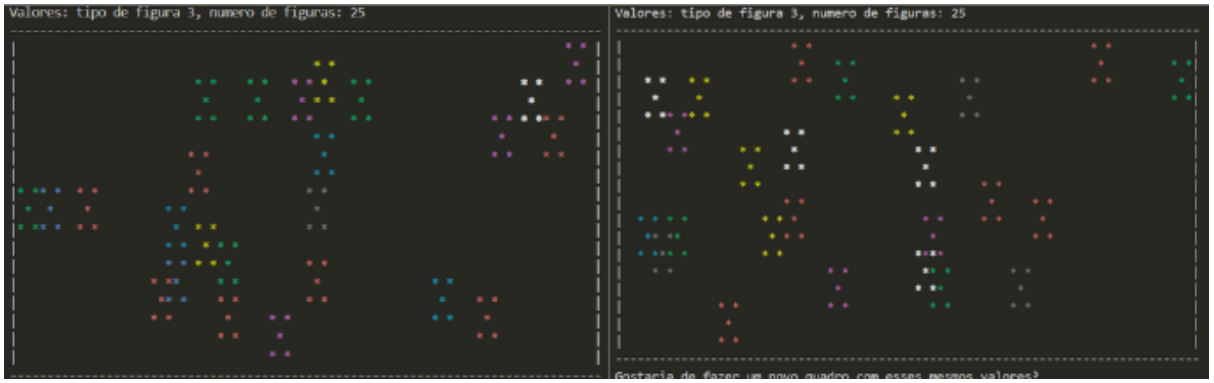


Figura 11 - artes geradas pela opção 3.


Figura 12 - artes geradas pela opção 4.

### 5.2 Mesa de cartas

A implementação da opção 5 também funcionou como o esperado, a figura 13 representa saídas obtidas a partir de sua utilização para mesmos valores e é possível observar a aleatoriedade das cartas geradas e também de suas posições.

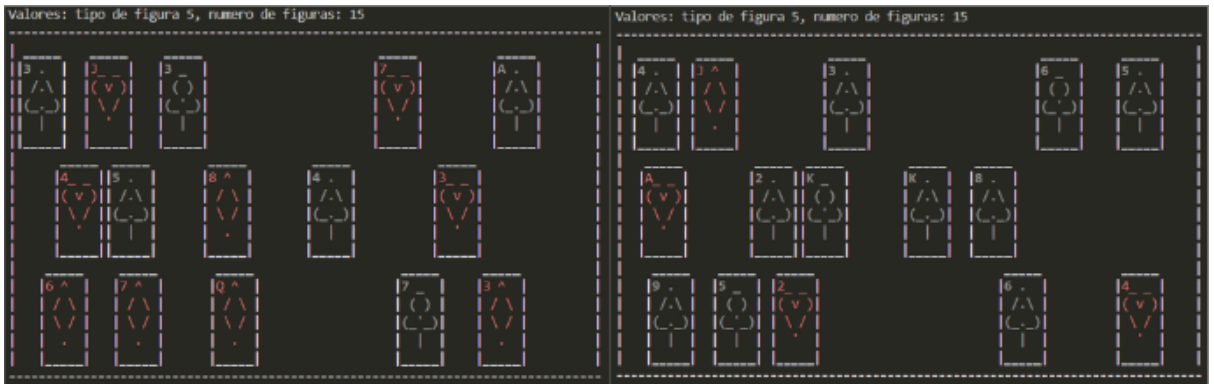


Figura 13 - artes geradas pela opção 5.

Como foi destacado anteriormente, existe um critério de que o mínimo de cartas por linha seja três, então para desenhar 8 cartas o programa faz a distribuição em apenas duas linhas como indicado na figura 14. Isso se configura como uma situação em que estariam sendo representados apenas dois jogadores,

uma vez que a ideia é que cada linha indique o conjunto de cartas de um mesmo jogador.


Figura 14 - artes geradas pela opção 5.

## 6. Conclusão

Diante as implementações realizadas e os resultados obtidos, é possível concluir que o projeto atendeu às especificações fornecidas uma vez que ele é capaz de gerar todas as figuras em posições aleatórias do quadro, além de possibilitar a formação de uma arte exclusiva que é retratada como uma mesa de cartas, em que as cartas escolhidas têm o naipe e valor aleatórios e assumem posições também aleatórias na sua distribuição, dentro dos critérios estabelecidos. O desenvolvimento desse projeto também cumpriu com o seu objetivo de servir como oportunidade para relembrar a utilização da linguagem C, mas além disso foi importante para o aprendizado de novas maneiras de implementação, principalmente no que se refere ao preenchimento de matrizes utilizando matrizes menores, que representam os desenhos. Ademais, o tratamento de colisões também exigiu um determinado esforço criativo para que fosse feito e isso se configura também como uma experiência enriquecedora no que se refere à obtenção ou desenvolvimento de novos raciocínios lógicos. Outrossim, a utilização de cores no terminal do programa foi outra novidade descoberta durante a criação desse projeto e tal funcionalidade se mostrou extremamente interessante e de significativa utilidade para futuras implementações.

## 7. Referências

[1] Medium. Disponível em: <https://medium.com/@selvarajk/adding-color-to-your-output-from-c-58f1a4dc4e75>

[2] IDE Visual Studio Code. Disponível em: <https://code.visualstudio.com/>

[3] Microsoft. Disponível em: <https://learn.microsoft.com/pt-br/cpp/c-language/?view=msvc-170>

[4] ASCII Art Archive. Disponível em: <https://www.asciiart.eu/miscellaneous/playing-cards>