UNIVERSIDADE FEDERAL DE PELOTAS
ENGENHARIA DE COMPUTAÇÃO

MATHIAS EZEQUIEL MILBRATH
RAMON PELLE
ROBERTA SILVA DOS SANTOS

BATALHA NAVAL

Prof. Rafael Burlamaqui Amaral

Pelotas
2022

MATHIAS EZEQUIEL MILBRATH
RAMON PELLE
ROBERTA SILVA DOS SANTOS

BATALHA NAVAL

Trabalho semestral final para a matéria de Programação de computadores.
Prof. Rafael Burlamaqui Amaral

Pelotas
2022

SUMÁRIO

LISTA DE FIGURAS

1. Introdução

Como requisito para a entrega do trabalho final, solicitado pelo professor Rafael Burlamaqui Amaral, este relatório é imprescindível para a conclusão da matéria de Programação de Computadores, visto que será exposto todo o processo de aplicação dos conhecimentos obtidos durante todo o semestre letivo.

O presente relatório visa relatar a experiência de programar o jogo Batalha Naval. O tempo dedicado à ele se estendeu durante os meses de maio e junho, sendo que todas as etapas foram realizadas através de reuniões on-line com os membros do grupo.

Para compilar o nosso programa, basta adicionar todos os arquivos em um projeto no DEV C++ e compilar normalmente por lá. No vídeo anexado a este relatório é exposto uma forma diferente, usando o VSCODE.

2. Desenvolvimento

Quando começamos a pensar em formas de desenvolver o trabalho, decidimos dividi-lo em 3 partes. A primeira delas foi a gerência/manipulação de arquivos e estruturação do código (menus, save do jogo, limite de dicas), a segunda foi toda a parte que se refere ao jogo (definir os navios, os tiros), e a terceira desenvolver a parte que só seria possível depois das duas primeiras: fazer o computador jogar contra o usuário.

Depois de termos nos reunido várias vezes, decidimos iniciar o projeto mesmo sem ter muita ideia de qual rumo todo o projeto ia tomar. Assim, com a ajuda do monitor, desenvolvemos toda a parte de arquivos. A maior dificuldade encontrada nesta etapa foi a decisão de como armazenaríamos a matriz do tabuleiro no .txt.

Figura 1: Tabuleiro no .txt

Depois de vários testes, optamos por separar todos os elementos da matriz pela tecla espaço. Nesse arquivo, representado na Figura 1, cada número equivale a um elemento do tabuleiro. Por exemplo, o número 1 representa o mar, os números 4 a 7 representam as embarcações e os números 2 e 3 representam as posições que já foram atingidas por um tiro. Isso se aplica tanto na matriz que armazena o tabuleiro do usuário, quanto na que armazena as informações do computador.

```
for (linha = 0; linha < 8; linha++){
    for (coluna = 0; coluna < 8; coluna++){
        switch (tabuleiro[linha][coluna])
        {
        case mar:
            fprintf(salvarJogo, "%d ", mar);
            break;
        case submarino:
            fprintf(salvarJogo, "%d ", submarino);
            break;
        case aviao:
            fprintf(salvarJogo, "%d ", aviao);
            break;
        case escolta:
            fprintf(salvarJogo, "%d ", escolta);
            break;
        case caca:
            fprintf(salvarJogo, "%d ", caca);
            break;
        case errou:
            fprintf(salvarJogo, "%d ", errou);
            break;
        case acertou:
            fprintf(salvarJogo, "%d ", acertou);
            break;
        }
    }
}
```

Figura 2: Função salvar: guardando o tabuleiro no .txt

Outra coisa bastante trabalhosa foi decidir como armazenaríamos o número de dicas que o usuário utilizou. Para este problema, utilizamos de uma solução mais prática. Para evitar a utilização de ponteiros ou de ficar passando parâmetros para certas funções que não necessariamente precisam deles, optamos por utilizar mais o arquivo main.c. Nele, deixamos as funções de carregar ou iniciar um jogo para que, desta forma, o nosso main ficasse mais importante e conseguíssemos resolver o problema do limite de dicas.

Na hora de carregar um novo jogo, percebemos que seria um problema grande caso não houvesse uma verificação se existe um arquivo ou ainda não. Para resolver, utilizamos a variável “flag”, a qual serve como um contador que indica se certos eventos aconteceram ou não.

```c
if(flag==0){
    printf("Não existe jogo salvo");
}else if(flag ==1){
    printf("Erro ao carregar jogo");
}else if (flag==2){
    clean_stdin();
    mostraTabuleiro(tabuleiro);
    mostraTabuleiroIA(tabuleiroIA);
    menuJogo(tabuleiro, &limite, tabuleiroIA);
}
```

Figura 3: Uso da variável flag.

A partir da Figura 3, analisamos que “flag==0” indica que nem o arquivo que contém o tabuleiro do jogador, nem o do computador existe. Se “flag==1”, apenas um existe, e “flag==2”, ambos existem, portanto podemos dar início ao jogo, chamando todas as outras funções.

Além disso, encontramos uma solução muito interessante na hora de importar os dados do .txt para o jogo. Como armazenamos com espaço entre os valores, na hora de importar apenas metade da matriz seria preenchida com valores inteiros, o resto seria o char espaço. Depois de pesquisarmos bastante, encontramos no *Stack Overflow* este conteúdo:

“O sub-especificador ‘*’ [...] indica que é para ignorar qualquer tipo de dado do tipo especificado e que está sendo lido. Ou seja, não será armazenado e sim ignorado. ”

```c
if(existir){
    for (linha = 0; linha < 8; linha++){
        for (coluna = 0; coluna < 8; coluna++){
            fscanf(existir, "%d%*c", &tabuleiro[linha][coluna]);
        }
    }
    flag++;
}
```

Figura 4: Aplicação do sub-especificador *

Com o uso de %d%*c, conseguimos ler somente inteiros, ignorando todo char, seja espaços ou quebras de linha.

Quando começamos a estruturar o jogo em si (definir os navios, os tiros, tabuleiros, etc), estudamos como base o código do jogo que estava no pdf das especificações do trabalho, que foi passado pelo professor. Em um primeiro momento, a interpretação do código foi muito dificultosa, então decidimos começar pela parte que nos pareceu mais fácil, a estrutura dos tiros.

A princípio, começamos com um código bem simples, onde na função denominada “menuDisparar” o usuário escolheria qual embarcação queria atirar, e, após isso, digitaria a linha e a coluna do tiro. Uma vez que foi escolhida tanto a embarcação, quanto a linha e coluna desejada, seria chamada uma nova função, a qual adicionaria um valor a linha ou coluna, dependendo o tipo do tiro, para informar que além da posição desejada, haveria outras casas, ou não, do tabuleiro que seriam afetadas.

```
void menuDisparar(int tabuleiro[][8],  int navios[][2]){
    system("cls");
    mostraTabuleiro(tabuleiro);
    int menu, aux1, aux2;
    int tiro[2];
    printf("--- OPÇÕES ---\n1- Submarino\n2- Porta-avião\n3- Navil de Escolta\n4- Caça\n-----\n");
        puts("Digite a embarcação desejada: ");
        scanf("%d",&menu);

        while (menu<1 || menu>4)
        {
            puts("Digite um número válido [1-4]");
            scanf("%d",&menu);
        }

        switch (menu){
        case 1:{
        case 2:
            aux1=1;
            aux2=0;

            printf("Linha: ");
            scanf("%d",&tiro[0]);
            tiro[0]--;

            printf("Coluna: ");
            scanf("%d",&tiro[1]);
            tiro[1]--;

            acertou(tiro, navios);
            alteraTabuleiro(tiro, navios, tabuleiro, aux1, aux2);
            mostraTabuleiro(tabuleiro);
        break;
```

Figura 5: Primeiro esboço do menu disparar.

```
void alteraTabuleiro(int tiro[2], int navios[][2], int tabuleiro[][8], int aux1, int aux2){
    int i, j;
    if(aux1==-1 && aux2==-1){
        for(i=-1; i<=1; i++){
            if(acertou(tiro,navios)){
                tabuleiro[tiro[0]+i][tiro[1]]=1;
            }else{
                tabuleiro[tiro[0]+i][tiro[1]]=0;
            }
        }

            for(j=-1; j<=1; j++){
                if(acertou(tiro,navios)){
                    tabuleiro[tiro[0]][tiro[1]+j]=1;
                }else{
                tabuleiro[tiro[0]][tiro[1]+j]=0;
                }
            }

    }else{
        for(i=0; i<=aux1; i++){
            for(j=0; j<=aux2; j++){
                if(acertou(tiro,navios)){
                    tabuleiro[tiro[0]+i][tiro[1]+j]=1;
                }else{
                    tabuleiro[tiro[0]+i][tiro[1]+j]=0;
                }
            }
        }
    }
}
```

Figura 6: Primeiro esboço da função que avalia os tiros.

Esta função funcionaria da seguinte forma: Por exemplo, utilizaria o porta-aviões para atirar; na função “menuDisparar” digitaria a opção 2, consequentemente entraria no case 2, onde seria requisitado a linha e a coluna desejada, armazenando a linha na variável de vetor tiro[0] e a coluna no tiro[1]; Quando fosse chamada a função “alteraTabuleiro” (figura 6), além dos parâmetros das coordenadas, também seria enviado os valores do aux1 e do aux2, que, no exemplo dado, seriam 1 e 0, respectivamente; com esses valores, seriam marcados como alvo a linha e a coluna desejada mais a proxima linha com a mesma coluna, por exemplo: (1,1) e (2,1), deste modo, o disparo seguiria o exigido para o porta-aviões (atira no quadrado que foi o alvo e no quadrado inferior).

Posteriormente, utilizamos dois for no menu do jogo para definir 4 variáveis, que seriam chamados no menu disparar, para fazer a verificação se ainda existia a embarcação desejada para dar o disparo ou se já havia sido destruída, caso estivesse destruída, o jogador deveria escolher outra.

Para otimizar o código, fizemos, com o auxílio do monitor, uso de struct para definir as características de cada embarcação. Primeiramente chamamos o struct, denominado embarcacoes, e adicionando as características que seriam utilizadas: tamanho, aux1(referente a linha, como explicado anteriormente) e aux2(referente a coluna). Posteriormente, criamos uma função que retornava estruturas, onde seriam definidos os valores dos parâmetros utilizados.

```
typedef struct embarcacao{
    int tamanho;
    int aux1;
    int aux2;
}embarcacao;

embarcacao definir_str(int tipo){
    embarcacao nova;
    switch (tipo){
    case submarino:
        nova.tamanho = 2;
        nova.aux1 = 0;
        nova.aux2 = 0;
        return nova;
    case aviao:
        nova.tamanho = 4;
        nova.aux1 = 1;
        nova.aux2 = 0;
        return nova;
    case escolta:
        nova.tamanho = 3;
        nova.aux1 = 0;
        nova.aux2 = 1;
        return nova;
    case caca:
        nova.tamanho = 2;
        nova.aux1 = -1;
        nova.aux2 = -1;
        return nova;
    default:
        return nova;
    }
}
```

Figura 7: Estrutura das embarcações

Depois que finalizamos a construção dos tiros, começamos a estruturar o chamamento das embarcações dentro do tabuleiro. Certamente esta foi a etapa mais complicada do trabalho. Inicialmente, tentamos construir a inicialização das embarcações em uma única função. Nela fizemos dois vetores, um para linha e um para coluna, ambos com tamanho 4, nos quais seriam armazenados números aleatórios gerados por um rand()%8. Para os mesmos números não serem repetidos dentro de cada vetor, fizemos uma função sugerida pelo monitor e pesquisada pelos integrantes, encontrando o seguinte vídeo no youtube que explicava como garantia esse funcionamento.

```
int existe(int valores[], int tam, int valor){
    int i;
    for (i=0; i<tam; i++){
        if(valores[i]==valor){
            return 1;
        }else{
            return 0;
        }
    }
}
//--->inicia navios
void iniciaNavios( int tabuleiro[][8]){
    int linha[4], coluna[4], i, j, l, c, tam[4];
    srand(time(NULL));

    embarcacao submarino_str=definir_str(submarino);
    embarcacao aviao_str=definir_str(aviao);
    embarcacao escolta_str=definir_str(escolta);
    embarcacao caca_str=definir_str(caca);

        tam[0]=submarino_str.tamanho;
        tam[1]=aviao_str.tamanho;
        tam[2]=escolta_str.tamanho;
        tam[3]=caca_str.tamanho;


    for(i=0;i<4;i++){
        l=rand()%8;
        c=rand()%8;

        while(existe(linha,i,l)){
            l=rand()%8;
        }
        linha[i]=l;


        while(existe(coluna,i,c)){
            c=rand()%8;

        }
        coluna[i]=c;
    }
```

Figura 8: Primeiro esboço de inicialização das embarcações.

Após muitos testes e erros, acabamos por desistir da ideia de uma única função para todas as embarcações. Refizemos estas inicializações, porém foi muito mais fácil e menos trabalhoso desta forma e, apesar de ter mais funções, o código ficou menor.

Dois pontos foram críticos para essa decisão: primeiramente porque estava ocorrendo o erro de cair duas embarcações na mesma posição; segundamente porque quando a embarcação estava perto da ponta do tabuleiro e esta era maior que o espaço que havia, a embarcação simplesmente pulava para a linha de baixo, dividindo-se. Nesse sentido, utilizamos um while para que fossem gerados números aleatórios até que a embarcação pudesse ser estabelecida em alguma posição. Para isso, utilizamos um if, em que só seria aceito um navio em determinada posição, se a coordenada da linha e da coluna inicial mais x quadrados ao lado (dependendo do tamanho da embarcação), estivessem ocupados por “mar”.

Durante uma das reuniões do grupo, acabamos por descobrir um erro no programa: Como o código lia apenas números com uma variável de inteiros, acabava que, ao digitar um ou mais caracteres dentro de qualquer menu, o programa bugava, mostrando várias vezes a mesma tela, travando ou entrando em alguma opção aleatória. Para resolver o problema criamos uma variável do tipo char que armazenava o que o usuário digitava, e após isso, com a utilização do atoi (ferramenta de string para transformar caracteres em números), armazenamos o valor digitado para a opção oficial que seria utilizada no switch. Antes de entrar para o switch, ainda é feita uma verificação com um while para garantir que o valor digitado é válido. Tal resolução foi utilizada para todo e qualquer switch onde o usuário teria que digitar uma opção.

```
printf("--- OPÇÕES ---\n1- Novo jogo\n2- Carregar\n3- Sobre\n4-Sair\n----------------------\n");
scanf("%c", &letraMenu);
menu = atoi(&letraMenu);


while((menu!=1) && (menu!=2) && (menu!=3) && (menu!=4)){
    puts("\nTente novamente com um valor válido!");
    clean_stdin();

    printf("--- OPÇÕES ---\n1- Novo jogo\n2- Carregar\n3- Sobre\n4-Sair\n----------------------\n");
    scanf("%c", &letraMenu);
    menu = atoi(&letraMenu);
}
```

Figura 9: Verificação de números para menus.

Na implementação da parte do computador, ou seja, o adversário do player, utilizamos muitas funções já criadas anteriormente, fazendo apenas algumas alterações.

Nossa maior dificuldade foi definir o local onde o computador poderia atirar. Foi pré-definido pelo professor que ele não podia atirar em locais já atirados, mas solucionamos esse problema com algumas estruturas de condição. A forma do computador jogar é aleatória, usando “rand”.

A parte de testar o acerto e mostrar o tabuleiro apenas replicamos a versão do usuário, mas colocando em funções diferentes para não ocorrer erros de sobreposição de alguns dados importantes.

Além da parte do jogo, o jogador tem acesso a uma função a mais do que a IA: A dica. Nela, ele fornece ao software uma posição específica e o mesmo retorna

se existe ou não uma embarcação no local desejado. Além disso, mostra também se por acaso o usuário já atirou nesta mesma posição.

A parte mais difícil foi definir o limite de dicas, pois toda vez que executávamos o programa, ele zerava. Para isso, decidimos trabalhar com ponteiro. A solução foi fácil e prática. Para termos certeza que quando o usuário vai ter apenas 3 dicas ao fechar o jogo e carregar o mesmo, salvamos essa variável em um arquivo, para manter esse controle.

Por fim, para podermos definir a vitória ou a derrota do jogador, utilizamos uma função parecida com a verificação de destruição das embarcações do menu do jogo, que retornaria um valor para definir o resultado. Esta função seria utilizada da seguinte forma: Antes do disparo do usuário, verificando que, se ele perdeu, não poderia mais atirar, e antes do disparo da CPU, pois ela não poderia atirar se o usuário ganhasse. Se em algum desses chamamentos o jogador ganhasse ou perdesse, o programa retornaria para o menu do jogo que iria acessar direto o case 6, onde imprimiria o resultado do jogo e se o jogador quer jogar novamente ou não.

```
if( (checkSubmarino[1]==0)&&(checkAviao[1] == 0)&&(checkCaca[1] == 0)&&(checkEscolta[1] == 0)){
        return ganhou;
}else if((checkSubmarino[0]==0)&&(checkAviao[0] == 0)&&(checkCaca[0] == 0)&&(checkEscolta[0] == 0)){
        return perdeu;
}else {
        return sair;
}
```

Figura 10: Verificação do resultado do jogo.

3. Conclusão

O jogo de batalha naval foi uma ótima escolha para a disciplina, pois nos obriga a utilizar todo o conhecimento que adquirimos durante o semestre. Não é um tema super complicado, mas nos forçou a pensarmos em maneiras mais eficazes de desenvolver as ferramentas.

Temos certeza que o que fizemos está muito bom, mas ainda não perfeito. Depois de muitas horas dedicadas ao projeto, percebemos que por mais atenção que nós demos, sempre tem coisas a serem melhoradas.

Desenvolvemos mais atenção na hora de manipularmos matrizes, principalmente entre as funções. Por mais que nós tivéssemos feito uma grande gama de exercícios, asseguramos que não teríamos absorvido tanto o conteúdo como absorvemos depois deste trabalho.

REFERÊNCIAS

BATTISTI, Anselmo. **Programa em C não faz a leitura correta de um arquivo .txt com números inteiros**. 2019. Disponível em: https://pt.stackoverflow.com/questions/386828/programa-em-c-não-faz-a-leitura-correta-de-um-arquivo-txt-com-números-inteiros. Acesso em: 01 maio 2022.

TEAM, Edpresso. **Resolving the "a label can only be part of a statement..." error**. 2022. Disponível em: https://www.educative.io/edpresso/resolving-the-a-label-can-only-be-part-of-a-statement-error. Acesso em: 10 maio 2022.

TOLEDO, Gilberto. **Programação em C/C++ - Aula 70 - Números aleatórios não repetidos**. 2016. Disponível em: https://www.youtube.com/watch?v=QpMlwC36Y8o. Acesso em: 15 maio 2022.