



SERVIÇO PÚBLICO FEDERAL - MINISTÉRIO DA EDUCAÇÃO  
CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS  
UNIDADE CONTAGEM  
ENSINO MÉDIO

MEL RAPOSEIRAS RICALDONI, NÚBIA TORRES DE OLIVEIRA, SARAH DOS  
SANTOS OLIVEIRA

**RELATÓRIO REFERENTE AO “TP - JOGO BATALHA NAVAL EM C++ E SFML”**

Contagem

Abril/2023

MEL RAPOSEIRAS RICALDONI - 20222011071

NÚBIA TORRES DE OLIVEIRA - 20222011106

SARAH DOS SANTOS OLIVEIRA - 20222011151

## **RELATÓRIO REFERENTE AO “TP - JOGO BATALHA NAVAL EM C++ E SFML”**

Relatório técnico do trabalho proposto pela disciplina LLPI do Curso de Informática do Centro Federal de Educação Tecnológica de Minas Gerais para o desenvolvimento do jogo Batalha Naval.

Orientador: Alisson Rodrigo dos Santos

Contagem

Abril/2023

## **AGRADECIMENTOS**

Em primeiro plano, a Deus, por fornecer força e perseverança durante a realização do projeto.

Ao professor Alisson Rodrigo dos Santos pela atenção e apoio, ao se disponibilizar para tirar nossas dúvidas explicando como solucionar as adversidades, assim como a orientação.

A nossos colegas de classe, pelo apoio e amizade, bem como a contribuição com seus conhecimentos e dúvidas durante as aulas, que auxiliaram para o melhor entendimento da matéria e resolução de algumas tarefas.

Enfim, aos nossos familiares, por não duvidarem de nossas capacidades e estarem sempre presentes diante de qualquer contratempo. Além de outras fontes, que contribuíram de forma direta ou indireta para a conclusão deste trabalho.

## RESUMO

Os jogos estão ganhando cada vez mais espaço na vida das pessoas, assim surge a necessidade do aparecimento de mais indivíduos capacitados para a sua produção. O atual trabalho tem o intuito de aplicar e desenvolver os conhecimentos sobre programação dos alunos do segundo ano de informática, além do aprendizado do uso e instalação de bibliotecas externas para a manipulação de sons, imagens e dinâmicas. A criação do jogo foi realizada através da IDE Eclipse, utilizando as linguagens de programação c/c ++, bem como a biblioteca SFML. Foram efetuadas pesquisas na internet e dúvidas com o professor da disciplina para a construção do algoritmo. Como resultado obteve-se não só a recreação simplificada do jogo Batalha Naval, como também a expansão dos conhecimentos das linguagens usadas e da realização da procura de conteúdos. Concluiu que é um projeto complicado, o emprego de novos conceitos e a elaboração de uma lógica complexa são tarefas que demandam esforço e bastante pesquisa.

**Palavras-chave:** Batalha Naval. SFML. Programação.

## SUMMARY

Games are gaining more and more space in people's lives, which creates the need for the emergence of more skilled individuals for their production. The purpose of the current work is to apply and develop the programming knowledge of second-year computer science students, as well as to learn the use and installation of external libraries for sound, image, and dynamic manipulation. The game creation was done using the Eclipse IDE, using the programming languages C/C++ as well as the SFML library. Internet research and consultation with the course professor were conducted for algorithm construction. As a result, not only a simplified recreation of the Battleship game was obtained, but also an expansion of knowledge of the languages used and content search. It was concluded that this is a complicated project, as the application of new concepts and the development of a complex logic require effort and extensive research.

**Keywords:** Battleship. SFML. Programming.

## SUMÁRIO

INTRODUÇÃO .....	6
OBJETIVOS .....	7
DESENVOLVIMENTO .....	8
ORIENTAÇÕES DO FUNCIONAMENTO DO JOGO .....	10
TÓPICOS CONCLUÍDOS E NÃO CONCLUÍDOS DO TRABALHO .....	11
PÁGINA PRINCIPAL (TABULEIRO) .....	21
CONCLUSÕES E RECOMENDAÇÕES .....	22
REFERÊNCIAS BIBLIOGRÁFICAS .....	23
BIBLIOGRAFIA RECOMENDADA .....	24

## INTRODUÇÃO

Este documento apresenta o relatório técnico referente ao projeto “TP - jogo Batalha Naval”, por meio da programação em c/c++ , utilizando a biblioteca de interface gráfica SFML. Foi desenvolvido pelas alunas Mel Raposeiras Ricaldoni, Núbia Torres de Oliveira e Sarah dos Santos Oliveira, de informática 2, mediante as orientações do professor Alisson Rodrigo dos Santos do Centro Federal de Educação Tecnológica de Minas Gerais (CEFET-MG).

A prática tem o objetivo de estar um nível acima da estudada pelos alunos atualmente, assim nos incentivando a estudar e fazer pesquisas em outras fontes, fora a escola. Ao final é esperado alcançar maiores conhecimentos a respeito da matéria, assim como a recreação simplificada do popular jogo “Batalha Naval”.

Nos anos 20, o passatempo se tornou popular entre prisioneiros e soldados no intervalo dos combates. Em 1931, apareceu nos EUA a primeira versão comercial, ainda em papel, chamada Salvo. Durante a 2ª Guerra Mundial, em 1943, foi lançado o jogo com o nome que ficou mais conhecido nos EUA: Battleship. Em 1967, durante a Guerra Fria, veio a primeira versão de tabuleiro, com as clássicas maletinhas e navios de plástico encaixáveis – lançada no Brasil em 1988. (ORTEGA, 2018, s/p)

Ao realizar este trabalho, utilizamos a IDE Eclipse para o desenvolvimento do código, bem como a incrementação da biblioteca SFML ao editor. Para melhor funcionamento do algoritmo optamos pela manipulação de funções, assim o programa exibe uma melhor organização, facilitando não só na apresentação, como também para a execução de futuras alterações.

## OBJETIVOS

- Uso da programação juntamente com o uso da biblioteca SFML;
- Explorar aspectos da biblioteca no desenvolvimento de aplicações multimédia;
- Construir o jogo Batalha Naval com:
  - ☒ Um tabuleiro de 10x10, utilizando imagens e sons para melhorar a interação;
  - ☒ Barcos posicionados aleatoriamente no tabuleiro: 1 barco de 5 quadrados, 2 barcos de 4 quadrados, 3 barcos de 3 quadrados e 4 barcos de 2 quadrados;
  - ☒ Permitir a interação do usuário com o tabuleiro, ao clicar sob um quadrado significa que uma bomba está sendo jogada ali;
  - ☒ O resultado deve ser sinalizado no tabuleiro – Se atingiu parte de um barco, se um barco foi completamente abatido e se o tiro acertou a água;
  - ☐ Ao final do jogo, aparece a possibilidade de jogar novamente sem ter que reiniciar o jogo.

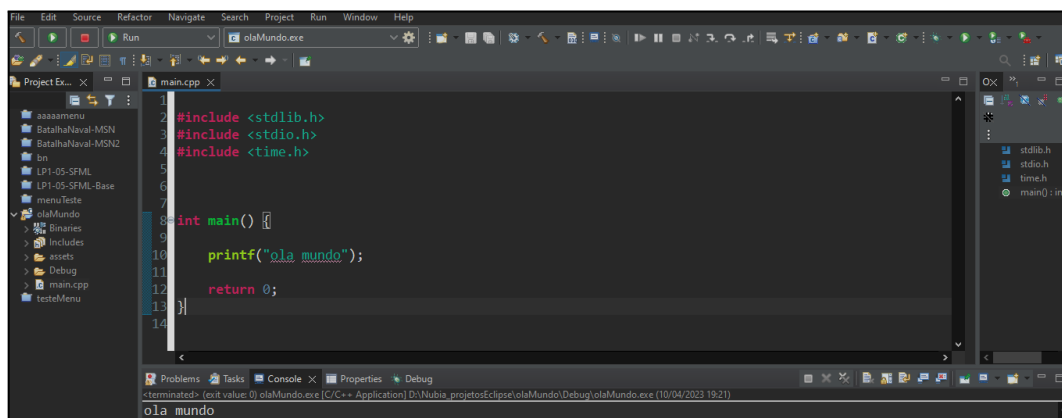


## DESENVOLVIMENTO

Após a disponibilização das instruções do trabalho para a turma, o trio prontificou-se para pesquisar a respeito do game, suas regras e como jogar, bem como alguns exemplos da aplicação das lógicas a serem desenvolvidas. Foram encontrados exemplos da jogabilidade no site: <https://rachacuca.com.br/jogos/batalha-naval/>.

Posteriormente, nas aulas seguintes da matéria de Laboratório de Linguagens e Técnicas de Programação, foi ensinado a instalar e utilizar a IDE Eclipse, que pode ser acessada através do site: <https://www.eclipse.org/downloads/>. Por ser a primeira vez manuseando o editor, o professor nos auxiliou com o passo a passo da utilização, durante a realização de outras práticas.

## TESTE NO EDITOR



Primeiro programa na IDE (fonte: autora)

Em seguida, realizamos a aplicação do Mingw64, a fim de utilizar o GCC para compilar nossos programas. Modificamos as propriedades da IDE para a inclusão do compilador. O link disponibilizado para o acesso do Mingw64 aplicado: [https://www.dropbox.com/s/qoxc2tm7zl6iumq/x86\\_64-6.4.0-release-posix-seh-rt\\_v5-rev0-7.3.0.7z?dl=0](https://www.dropbox.com/s/qoxc2tm7zl6iumq/x86_64-6.4.0-release-posix-seh-rt_v5-rev0-7.3.0.7z?dl=0).

O próximo passo foi instalar a biblioteca SFML. No moodle da disciplina foram disponibilizados um tutorial do professor a respeito da integração da SFML ao Eclipse, o link para baixar os arquivos da biblioteca e um exemplo de código simples, para testar a operação. Aplicamos a versão SFML 2.5.1, a qual era compatível para nossos segmentos, acesso em:

[https://ava.cefetmg.br/pluginfile.php/43214/mod\\_assign/introattachment/0/SFML-2.5.1-windows-gcc-7.3.0-mingw-64-bit.zip?forcedownload=1](https://ava.cefetmg.br/pluginfile.php/43214/mod_assign/introattachment/0/SFML-2.5.1-windows-gcc-7.3.0-mingw-64-bit.zip?forcedownload=1).

### Orientações do funcionamento do jogo

Será criado um tabuleiro (matriz) 10 x 10, onde ocorrerá a dinâmica. No início, deverão ser posicionados 10 barcos, com os tamanhos definidos: 1 barco de 5 quadrados, 2 barcos de 4 quadrados, 3 barcos de 3 quadrados e 4 barcos de 2 quadrados. Logo após, o jogador começará a jogar bombas no tabuleiro (clique nos quadrados) a fim de acertar os barcos. O objetivo é afundar todos os barcos. Para ganhar, o jogador precisa acertar todos os barcos posicionados no tabuleiro.

Ao desenvolver o algoritmo dividimos as tarefas em partes:

- a. Construir a matriz interativa;
- b. Posicionar os barcos aleatoriamente (sem que eles se encostassem);
- c. Construir a lógica das bombas;
- d. Adicionar os mecanismos restantes.

Na organização do projeto utilizamos **headers** (cabeçalhos) para a declaração das funções, e no arquivo principal (o main.cpp) realizamos as definições destas funções. Logo, temos uma pasta para cada parte do algoritmo, assim, durante a realização de modificações ficará mais fácil de manusear. As pastas são denominadas da seguinte forma: nome.h. Exemplos criados para o trabalho: **barcos.h**; **enum.h**; **star.h**.

Ao decorrer do código utilizamos de algumas **funções** (um bloco de código que executa alguma operação) para o melhor funcionamento, e para a **função main** (a principal) não encher de informações. A principal foi utilizada apenas para “chamar” as outras funções.

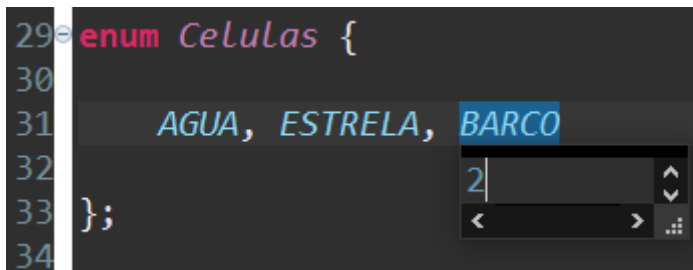
Uma das funções utilizada foi a **enum** (abreviação de "enumeration") que permite definir um tipo de dados que representa um conjunto fixo de valores constantes. Esses valores são listados explicitamente na definição do enum, e cada um é associado a um identificador único. Abaixo a aplicação no programa:

## FUNÇÕES EXEMPLOS

```

29 enum Celulas {
30
31     AGUA, ESTRELA, BARCO
32
33 };
34

```



Função enum (fonte: autora)

Neste caso, os dados são definidos como: *AGUA=0*, *ESTRELA=1* e *BARCO=2*. Assim, ao decorrer da lógica desenvolvida, foram utilizados estes valores na manipulação das funções.

Encontramos algumas dificuldades no cumprimento de todos os objetivos, assim, alguns tópicos não foram concluídos para a finalização do projeto.

## Tópicos concluídos e não concluídos do trabalho

### Concluídos

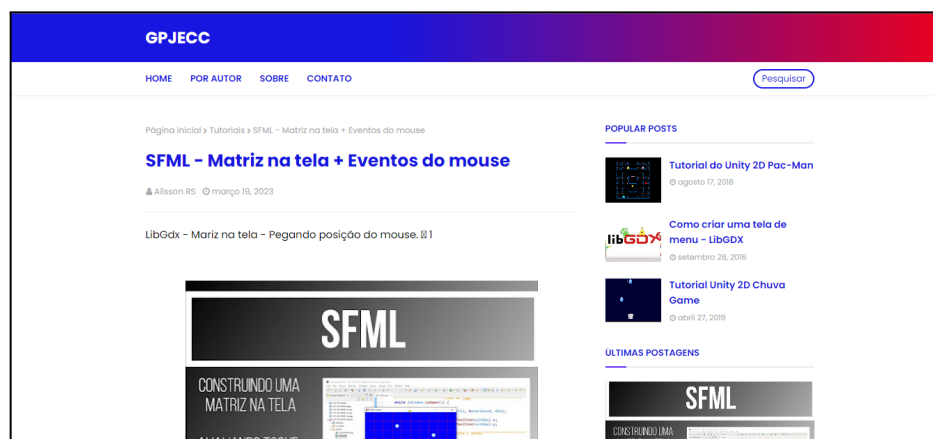
**1.1** Um tabuleiro 10 x 10, com imagens (animações) e sons, onde é possível interagir (clicar) nos espaços da matriz;

Para a construção da matriz interativa, utilizamos como referência o código disponibilizado pelo professor em seu site. Foi incluída no algoritmo uma classe da biblioteca, responsável por desenhar retângulos na tela: *sf::RectangleShape boxes[10][10];*

Outro recurso importante era a interação do usuário com o tabuleiro (ou seja, a matriz), para isso usamos opções da SFML que permitia essa conexão com o jogador, um exemplo é o toque do mouse: *sf::Event::MouseButtonPressed*.

Este programa está dividido em partes, que são “chamadas” pela função principal. Dessa forma, temos criadas as seguintes funções: *sf::RectangleShape criaRetangulo; enum Celulas; void trataEventos; void desenhaRetangulos;* e a principal *int main();*

## SITE REFERÊNCIA



Site do código de referência

(fonte: <https://gpjecc.blogspot.com/2023/03/sfml-matriz-na-tela-eventos-do-mouse.html>)

Para customizar o tabuleiro utilizamos uma animação de água ao fundo.

## CUSTOMIZAÇÃO DO TABULEIRO

```

/***** Declaracao das texturas para a animacao da agua no jogo *****/

sf::Texture fundo[21];

fundo[0].loadFromFile("assets/ocean01.png");
fundo[1].loadFromFile("assets/ocean02.png");
fundo[2].loadFromFile("assets/ocean03.png");
fundo[3].loadFromFile("assets/ocean04.png");
fundo[4].loadFromFile("assets/ocean05.png");
fundo[5].loadFromFile("assets/ocean06.png");
fundo[6].loadFromFile("assets/ocean07.png");
fundo[7].loadFromFile("assets/ocean08.png");
fundo[8].loadFromFile("assets/ocean09.png");
fundo[9].loadFromFile("assets/ocean10.png");
fundo[10].loadFromFile("assets/ocean11.png");
fundo[11].loadFromFile("assets/ocean12.png");
fundo[12].loadFromFile("assets/ocean13.png");
fundo[13].loadFromFile("assets/ocean14.png");
fundo[14].loadFromFile("assets/ocean15.png");
fundo[15].loadFromFile("assets/ocean16.png");
fundo[16].loadFromFile("assets/ocean17.png");
fundo[17].loadFromFile("assets/ocean18.png");
fundo[18].loadFromFile("assets/ocean19.png");
fundo[19].loadFromFile("assets/ocean20.png");
fundo[20].loadFromFile("assets/ocean21.png");

int textura_atual = 0; //para percorrer o vetor de texturas
int frame = 0; //para contar os frames da animacao
  
```

Adicionando as texturas de oceano (fonte: autora)

Esta foi adicionada à matriz através de um vetor de texturas (uma classe que representa uma imagem ou uma textura), em que, em cada posição foram armazenadas uma imagem que, posteriormente, fariam a composição da animação.

Logo abaixo do preenchimento dos espaços do vetor, estão declaradas duas variáveis importantes para a animação. A variável *int textura\_atual* é a que vai percorrer o vetor e irá mostrar as imagens. Ela é inicializada com 0 para que comece da primeira imagem. Já a

variável *int frame* é para contar quantas vezes o programa vai ficar na mesma imagem, por exemplo, se a imagem deverá ficar na tela por 2 ciclos ou por 50 ciclos.

```

/***** Sprites para a animacao da agua no jogo *****/

//linha1
sf::Sprite agua(fundo[textura_atual]);
agua.setScale(1.052, 1);

sf::Sprite agua2(fundo[textura_atual]);
agua2.setScale(1.052, 1);
agua2.setPosition(360, 0);

sf::Sprite agua3(fundo[textura_atual]);
agua3.setScale(1.052, 1);
agua3.setPosition(720, 0);

//linha2
sf::Sprite agua4(fundo[textura_atual]);
agua4.setScale(1.052, 1);
agua4.setPosition(0, 180);

sf::Sprite agua5(fundo[textura_atual]);
agua5.setScale(1.052, 1);
agua5.setPosition(360, 180);

sf::Sprite agua6(fundo[textura_atual]);
agua6.setScale(1.052, 1);
agua6.setPosition(720, 180);

```

Aplicando sprites para animação da água (fonte: autora)

## ANIMAÇÃO DA ÁGUA

```

//linha5
sf::Sprite agual3(fundo[textura_atual]);
agual3.setScale(1.052, 1);
agual3.setPosition(0, 720);

sf::Sprite agual4(fundo[textura_atual]);
agual4.setScale(1.052, 1);
agual4.setPosition(360, 720);

sf::Sprite agual5(fundo[textura_atual]);
agual5.setScale(1.052, 1);
agual5.setPosition(720, 720);

if (frame == 2)
{
    frame = 0;

    if (textura_atual == 20)
    {
        textura_atual = 0;
    } else {
        textura_atual++;
    }
}

```

Lógica da animação (fonte:autora)

```
frame++;
```

Lógica da animação (fonte:autora)

Nesta parte foram criados Sprites() para preencher toda a matriz. A variável *textura\_atual* é a que vai percorrer o vetor de texturas. No final é possível perceber um *if*, nele está praticamente toda a lógica da animação. este *if* está programado para quando *frame = 2*, *frame* deve voltar a ser 0, porque, depois que uma imagem já passou por 2 ciclos – de 0 a 2 – é a vez a próxima imagem passar por 2 ciclos, por isso ela volta a ser 0.

Entrando no *if* temos que se *textura\_atual == 20* então *textura\_atual = 0*. O vetor de texturas têm posições de 0 a 20; este comando faz com que, quando chegar na imagem que está na posição 20, volte para a imagem na posição 0, assim teremos um loop e a animação não vai parar, até que o programa feche.

Após isso temos um *else*, que é para o caso de não estar na imagem de posição 20, nesse caso a variável *textura\_atual* é incrementada em mais 1, dessa forma passando para a próxima posição do vetor e consequentemente para a próxima imagem.

No final do código a variável *frame* é incrementada, esta parte é o penúltimo comando do *main*.

**1.2** Gerar todos os 10 barcos de tamanhos diferentes em coordenadas aleatórias, sem encostarem um no outro, em direções alternadas (vertical e horizontal), além de individualizá-los com um sistema de ID.

O algoritmo responsável pelo cálculo dessa etapa será dividido em 4 funções principais:

- **void** *gen\_game\_matrix* (**int** g[][10], **sf::RectangleShape** boxes[10][10], **int** dim)

Esta função sem retorno receberá os seguintes argumentos: uma matriz 10x10 do tipo inteiro — a matriz lógica —, uma matriz de retângulos — a matriz gráfica — e a dimensão dos retângulos. Sua execução desencadeará o uso das demais funções, portanto, é a única a ser chamada no arquivo principal *main.cpp*. Inicialmente temos a função *srand*:

“Para cada valor de semente diferente usado em uma chamada para `srand`, pode-se esperar que o gerador de números pseudoaleatórios gere uma sucessão diferente de resultados nas chamadas subsequentes para `rand`. [...] A geração de números aleatórios toda vez que o programa é rodado, exige que `srand` seja inicializada com algum valor de tempo de execução distinto, como por exemplo o valor retornado pela função `time` (declarado no cabeçalho `<time.h>`). Isso é distinto o suficiente para as necessidades de randomização mais triviais.” (CPLUSPLUS, s/p, tradução nossa)

```
srand(time(NULL));
```

Função `srand` utilizando a função `time` como argumento (fonte: autora)

`Null` é utilizado para que a função `time` retorne o número total de segundos decorridos desde 01/Janeiro/1970.

Em seguida a variável `placing_fail` do tipo `bool` é declarada, seu nome significa “falha na colocação dos barcos”. Basicamente: se for definida como `true` é porque um erro ocorreu durante a geração dos elementos, veremos o que a define como verdadeira ou falsa posteriormente. Para garantir um resultado correto, é preciso que enquanto houver falha o algoritmo seja recalculado. Dessa forma, um *loop do while* é construído:

```
do{
    placing_fail = repeat(g, boxes, dim);
}while(placing_fail);
```

Loop (fonte: autora)

O *loop* será finalizado quando `placing_fail` for igual a `false`, ou seja, quando os erros pararem de ocorrer. O valor dessa variável é dinâmico e obtido através da atribuição com diversas funções durante a execução do programa.

- **`bool repeat(int grid[][10], sf::RectangleShape boxes[10][10], int dim)`**

Esta função retornará um valor do tipo `bool` e tem como argumentos a matriz lógica, a matriz gráfica e a dimensão dos retângulos. As variáveis desse escopo são: `int totalboats = 0`, `sub_count = 0`, `ctorp_count = 0`, `tank_count = 0`, `plane_count = 0`, respectivamente o total de barcos e quantidade de cada tipo de barco (submarino, contratorpedeiros, navios-tanque e

porta-aviões). A quantidade de cada tipo de barco será o elemento chave para individualizar cada barco futuramente. **Como barcos não voam**, a matriz lógica deverá ser inicializada como **água**. Dois *for* foram utilizados para percorrer a matriz e, simultaneamente, definir os valores de seus elementos e criar os retângulos da parte gráfica através de outra função incluída no cabeçalho `tile.h`.

```
//Inicializa matriz como agua
for(int l = 0; l < 10; l++){
    for(int c = 0; c < 10; c++){
        grid[l][c] = agua;
        boxes[l][c] = criaRetangulo(l * dim, c * dim, dim, dim);
    }
}
```

(fonte: autora)

Encerrado o *loop*, iniciam-se as chamadas da próxima função: *gen\_boat*. Para cada barco com aparições maiores do que 1 (submarinos, contratorpedeiros e navios-tanque), é feito um *for* que utiliza como contador as variáveis que representam a quantidade de cada tipo de barco.

```
for(sub_count; sub_count < 4 ; sub_count++){
    placing_fail = gen_boat(grid,2,sub_count);
    if(placing_fail == true){
        break;
    }
}
```

(fonte: autora)

Sendo o único de seu tipo e exceção, o barco porta-aviões foi construído dessa forma:

```
placing_fail = gen_boat(grid,5, plane_count);
if(placing_fail != true){
    plane_count++;
}
```

(fonte: autora)

Caso os *loops* não forem interrompidos por algum erro, a quantidade total de barcos será 10, condição necessária para que *placing\_fail* seja falsa.



```

totalboats = sub_count + ctorp_count + tank_count + plane_count;
if(totalboats != 10){
    placing_fail = true;
    return placing_fail;
}else{
    placing_fail = false;
    return placing_fail;
}

```

(fonte: autora)

- **bool** *gen\_boat*(**int** g[][10], **int** len, **int** id)

Outra função com retorno do tipo bool, recebe a matriz lógica, o comprimento do barco e a id — o número atual do barco gerado na contagem — . Possui as seguintes variáveis em seu escopo: **int** *x*, *y* (coordenadas do quadrado inicial), **int** *i* (que receberá posteriormente o valor do *comprimento do barco - 1*), **Celulas** *barco* (uma variável baseada na enum **Celulas** que corresponderá ao tipo do barco gerado), **direcao** *dir* (uma variável baseada na enum **direcao** que corresponderá à direção do barco, vertical ou horizontal), **int** *error* (mapeia erros de posicionamento e obriga o computador a recalculá-los até certo limite), **int** *it\_count* (conta o número de iterações que o *loop* de tentativas de posicionamento fez) e **bool** *placing\_fail* (uma espécie de broadcast que interliga as principais funções e detecta erros que exigem o reposicionamento desde o primeiro barco).

No início, relacionamos a variável *barco* com o argumento *len* (comprimento do barco) através de um *switch case*:

```

switch(len){

    case 2:
        barco = submarino;
        break;
    case 3:
        barco = ctorpedo;
        break;
    case 4:
        barco = tanque;
        break;
    case 5:
        barco = porta_avioes;
        break;
};

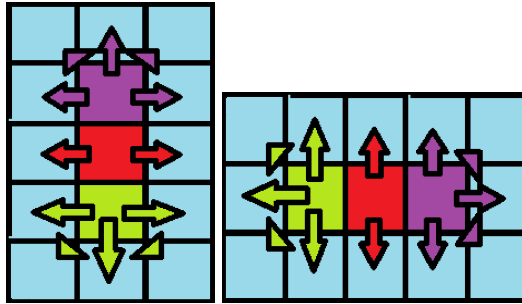
```

(fonte: autora)

A partir disso o *loop* é construído: preliminar aos cálculos, a variável *it\_count* incrementa a cada vez que o *loop* se repete, ou seja, a cada vez que ocorre um erro possível de ser resolvido. Se o número de iterações chegar a 100, significa que o computador verificou praticamente a matriz inteira e mesmo assim não achou um lugar possível para gerar o barco, uma falha que exige o recomeço de todos os cálculos para ser corrigida. Assim, o *loop* quebra e *placing\_fail* assume o valor *true* antes de ser retornada pela função. As coordenadas *x* (linha) e *y* (coluna) são randomizadas pela função *rand*, em um *range* de 0 a 9 para cada. Juntamente, *error* é zerado e a variável *dir* também é randomizada entre 0 ou 1, que correspondem a vertical ou horizontal na enum **direção**. Outro *switch case* é utilizado para definir os procedimentos específicos de cada caso da variável *dir* (VER ou HOR): por exemplo, se *dir* = VER, *x* não poderá ser 0, *error* se torna -1 e o *loop* continua (gera as coordenadas novamente), enquanto que se *dir* = HOR, *y* não poderá ser 9, repetindo o sistema de mapeamento de erro e reiniciando o loop. Para coordenadas que não ultrapassem os limites da matriz, um *for* é feito para checar se todos os quadrados na determinada direção estão livres: caso um quadrado for diferente de **água**, um erro é gerado e o *for* quebra. O *do-while* sempre detecta se *erro* é = -1, o que faz com que ele continue. Se o mar estiver para peixe, o possível barco passará por mais um teste, apelidado de “periféricos”, esse será feito por uma função de mesmo nome e *error* mapeará seu retorno. No final do *switch*, se algum caso correr bem e chegar até o final, *error* assume o valor 1 e o *loop* é finalizado por essa condição. O término do loop significa que tudo deu certo e que o barco foi posicionado, fazendo com que *placing\_fail* torne-se *false* antes de ser retornada pela função.

- **int** *perifericos*(**int** g[][10], **int** i, **int** x, **int** y, Celulas barco, **direcao** dir, **int** id)

A última função do algoritmo, graças a ela os barcos não ficam grudados e possuem identificação única. Retorna um valor inteiro e recebe como argumento a matriz lógica, as coordenadas, o tipo do barco, a direção e a id do barco a ser gerado. As variáveis no seu escopo são **int** *error* = 0, *i2* = *i* (uma cópia de *i*, que equivale a *len-1*), *cont* = 0 (conta os quadrados que passaram em todos os testes). Um *switch case* é feito para cada direção, pois o cálculo dos periféricos muda dependendo da orientação do barco. Para esclarecer melhor a teoria dos periféricos, seguem as imagens:



(fonte: autora)

Os quadrados roxos representam os quadrados finais do barco, os verdes os iniciais e os vermelhos os do meio. Cada um desses quadrados possuem testes de periféricos exclusivos, por exemplo: um barco não poderá ser posicionado caso a diagonal superior esquerda do quadrado final for igual a um barco. O problema da questão da identificação foi fazer com que esses testes reconhecessem todos os tipos de barco acompanhados de suas ID de contagem. A solução foi um *if* que comparava cada periférico dos quadrados com todas as probabilidades de barcos possíveis. Após os testes serem aprovados por meio da variável *cont*, que conta os quadrados válidos e é comparada com a *len*, os **periféricos** (representados por 1 na enum **Celulas**) são definidos na matriz, evitando a ultrapassagem dos limites. Para finalizar, a utilização da ID consiste na soma desta com o inteiro que representa o tipo do barco na enum **Celulas**, o que permite a diferenciação entre sequências de quadrados do mesmo tipo.

Exemplo (os periféricos foram impressos como água):

### IMPRESSÃO DOS BARCOS

0	0	0	0	0	0	22	22	0	0
0	0	0	0	0	0	0	0	0	31
0	41	0	0	0	0	0	0	0	31
0	41	0	51	51	51	51	51	0	31
0	41	0	0	0	0	0	0	0	0
0	41	0	21	21	0	42	42	42	42
0	0	0	0	0	0	0	0	0	0
0	0	0	0	32	32	32	0	23	23
0	0	0	0	0	0	0	0	0	0
0	24	24	0	33	33	33	0	0	0

<span style="color: red;">■</span> Primeiro submarino	<span style="color: green;">■</span> Terceiro submarino
<span style="color: orange;">■</span> Segundo submarino	<span style="color: purple;">■</span> Quarto submarino

(fonte: autora)

### Não concluídos

- ☐ Reiniciar o jogo sem precisar sair;
- ☐ Mecanismo de bombas;
- ☐ Sinalizar destruição de um barco.

### Criação do menu (tópico opcional)

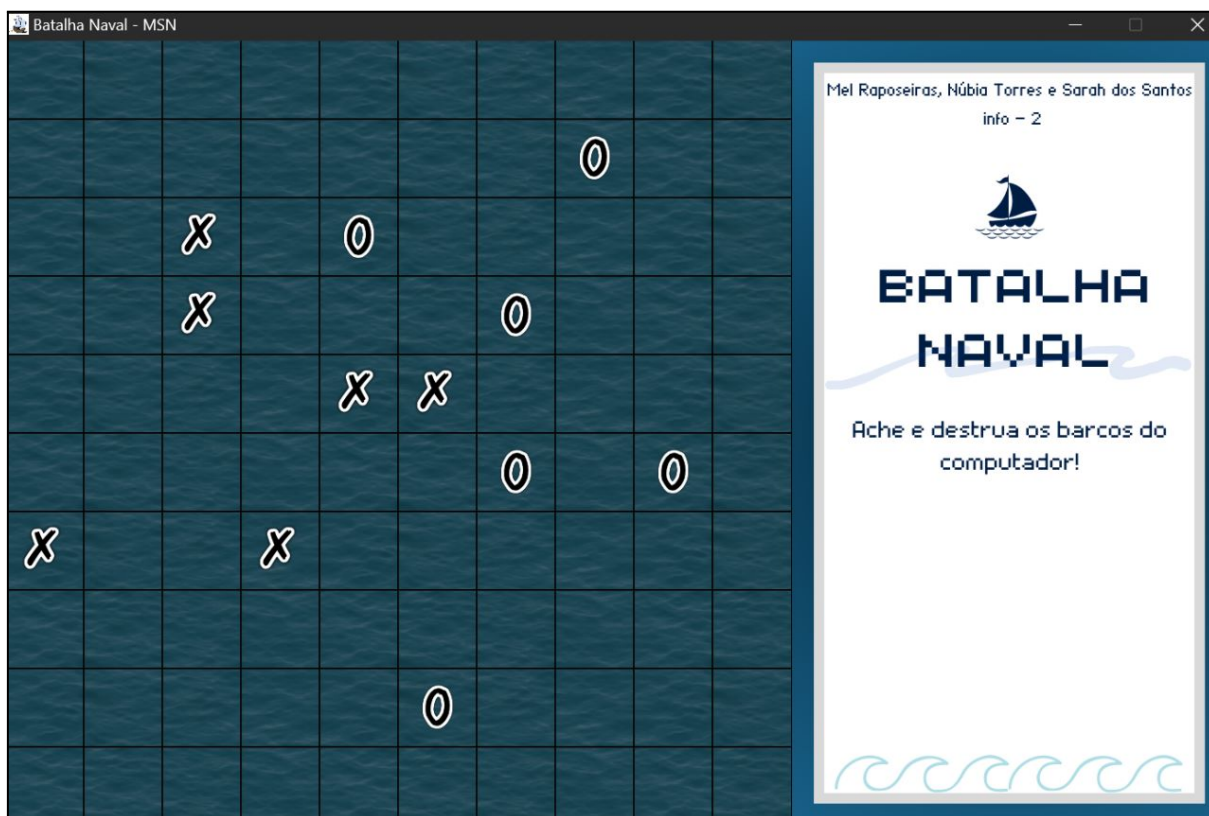
Tentamos construir um menu inicial, com um botão escrito “jogar”, assim, quando o jogador pressionar será direcionado para a tela principal do Batalha Naval. Contudo, a tarefa não foi concluída, visto que a lógica do botão não foi finalizada. Abaixo tem-se o exemplo final da primeira tela, na qual contamos com o título do jogo, em uma fonte maior, o nome das autoras logo abaixo, com a fonte um pouco menor, e, no meio, a palavra “jogar” que serviria como um botão. Quando aberta, esta janela toca uma música de fundo (música: He’s a pirate).

### MENU INICIAL



Menu inicial (fonte: autora)

## PÁGINA PRINCIPAL (TABULEIRO)



Tabuleiro Batalha Naval (fonte: autora)

## CONCLUSÕES E RECOMENDAÇÕES

Neste trabalho abordamos alguns dos objetivos presentes na especificação. Dividimos as tarefas para resolver cada lógica/problema separado e aos poucos (para assim termos uma melhor organização). Encontramos dificuldades na realização de algumas partes, assim conseguimos solucionar algumas das dúvidas com o professor e colegas, contudo não foram todas que obtivemos sucesso na conclusão. Manuseamos o tabuleiro do jogo (matriz 10x10), de modo que o jogador consegue interagir com os elementos dele, atirando as bombas a fim de acertar os barcos. Completamos o raciocínio do posicionamento dos barcos na matriz, são alocados 10 barcos, com os tamanhos conforme informados nas instruções do projeto (1 barco de 5 quadrados, 2 barcos de 4 quadrados, 3 barcos de 3 quadrados e 4 barcos de 2 quadrados), que não encostam uns nos outros (um dos requisitos da tarefa). Ao lado da matriz colocamos um painel (imagem), explicando que existem 10 barcos aleatórios espalhados e número infinito de bombas. Não foi possível criar o botão ao final do game para que o jogador pudesse jogar novamente sem precisar sair da janela. Gostaríamos de fazer um menu inicial, mas não o concluímos.

Por fim, apesar de alguns obstáculos, conseguimos realizar alguns dos objetivos do trabalho. Elevamos nossos conhecimentos a respeito da lógica de programação e utilização de interface gráfica. Aprendemos não só a utilizar uma nova IDE, como também a instalar uma nova biblioteca, e empregar algumas de suas classes e funções. Não conseguimos entregar o Batalha Naval completo, do modo como planejamos, entretanto, trabalhamos em equipe até chegar em nosso produto final.

## REFERÊNCIAS BIBLIOGRÁFICAS

Animações com Sprites | SFML & C++ | Tutorial 07. Disponível em: <[https://www.youtube.com/watch?v=ic5YaX\\_V2TA](https://www.youtube.com/watch?v=ic5YaX_V2TA)>. Acesso em: 09 abr. 2023.

*Como CRIAR uma JANELA com SFML & C++ - Tutorial 02.* 2021. Disponível em: <<https://www.youtube.com/watch?v=J9iv9GKzEas>>. Acesso em: 21 março. 2023.

OpenGameArt.org. Disponível em: <<https://opengameart.org/>>.

ORTEGA, Rodrigo. Como surgiu o jogo Batalha Naval? 2018. Disponível em: <<https://super.abril.com.br/mundo-estranho/como-surgiu-o-jogo-batalha-naval/>>. Acesso em: 08 de abril. 2023.

*SPRITES e TEXTURAS | SFML & C++ | Tutorial 05.* 2021. Disponível em: <<https://www.youtube.com/watch?v=HfdJ6P7C6NA>>. Acesso em: 21 março. 2023.

## BIBLIOGRAFIA RECOMENDADA

Animações com Sprites | SFML & C++ | Tutorial 07. Disponível em: <[https://www.youtube.com/watch?v=ic5YaX\\_V2TA](https://www.youtube.com/watch?v=ic5YaX_V2TA)>. Acesso em: 09 abr. 2023.

Cplusplus. Disponível em: <<https://cplusplus.com/>>. Acesso em: 09 abr. 2023.

C progressivo. *Jogo: Batalha Naval em C*. 2012. Disponível em: <<https://www.cprogressivo.net/2012/09/batalha-naval-em-c.html>>. Acesso em 09 abril. 2023

SANTOS, Alisson. *Bibliotecas de jogos para apoio das disciplinas de LP1 e LP2*. 2016. Disponível em: <<http://gpjecc.blogspot.com/2016/02/bibliotecas-de-jogos-para-apoio-das.html>>. Acesso em: 03 abril. 2023.

SANTOS, Alisson. *SFML - Matriz na tela + Eventos do mouse*. 2023. Disponível em: <<https://gpjecc.blogspot.com/2023/03/sfml-matriz-na-tela-eventos-do-mouse.html>>. Acesso em: 21 de março. 2023