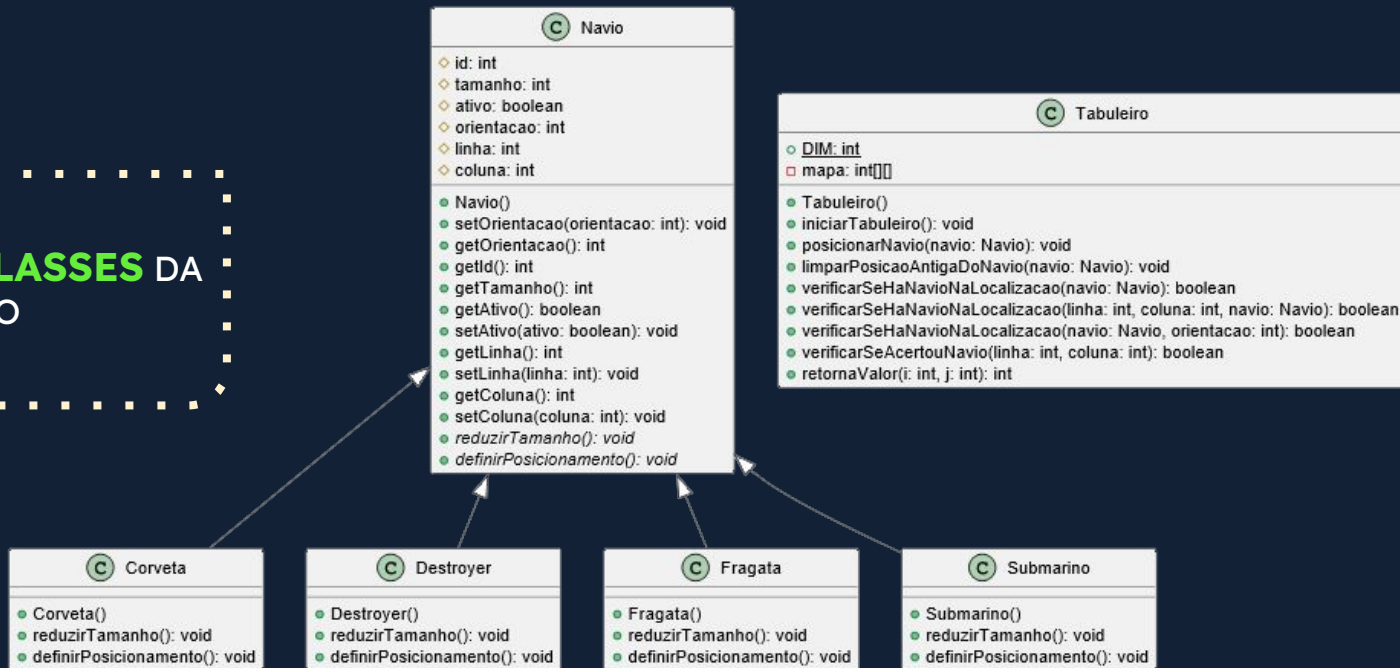


# LINGUAGEM DE PROGRAMAÇÃO II

Thiago Vitor Moreira Maia

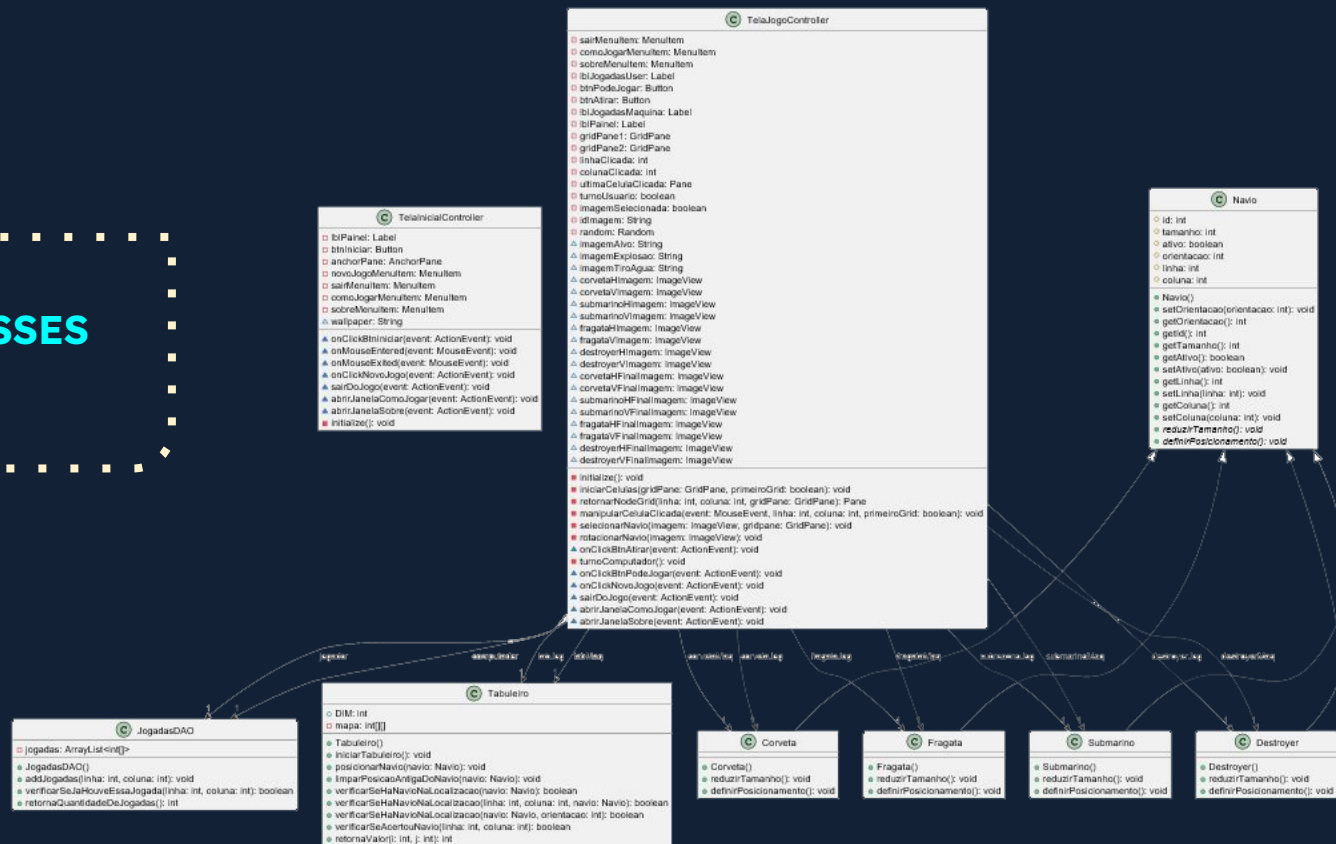
# DIAGRAMA DE CLASSES

## DIAGRAMA DE CLASSES DA CAMADA DE MODELO



# DIAGRAMA FINAL

## DIAGRAMA DE CLASSES FINAL DO PROJETO



# CLASSES MODELO - TABULEIRO

```
public class Tabuleiro {
    public static final int DIM = 10; // (dimensão do mapa)
    private int mapa[][]; // Array bidimensional, que representará um tabuleiro com os navios posicio

    public Tabuleiro() {
        mapa = new int[DIM][DIM];
    }

    * Método que inicializa o tabuleiro. A princípio, o tabuleiro será preenchido com o valor 0 em
    public void iniciarTabuleiro() {
        for (int i = 0; i < DIM; i++) {
            for (int j = 0; j < DIM; j++) {
                mapa[i][j] = 0;
            }
        }
    }

    * Método que posiciona um navio no tabuleiro.
    public void posicionarNavio(Navio navio) {
        if (navio.getOrientacao() == 0) { // horizontal
            for (int i = navio.getColuna(); i < navio.getColuna() + navio.getTamanho(); i++) {
                mapa[navio.getLinha()][i] = navio.getId();
            }
        } else { // vertical
            for (int i = navio.getLinha(); i < navio.getLinha() + navio.getTamanho(); i++) {
                mapa[i][navio.getColuna()] = navio.getId();
            }
        }
    }

    * Método que limpa a posição atual do navio.
    public void limparPosicaoAntigaDoNavio(Navio navio) {
        if (navio.getOrientacao() == 0) { // horizontal
            for (int i = navio.getColuna(); i < navio.getColuna() + navio.getTamanho(); i++) {
                mapa[navio.getLinha()][i] = 0;
            }
        } else { // vertical
            for (int i = navio.getLinha(); i < navio.getLinha() + navio.getTamanho(); i++) {
                mapa[i][navio.getColuna()] = 0;
            }
        }
    }
}
```

```
    } else {
        for (int i = linha; i < linha+navio.tamanho; i++) {
            if(mapa[i][coluna] != 0 && mapa[i][coluna] != navio.id) {
                return false;
            }
        }
    }

    return true;
}

* Método para verificar se há algum navio posicionado no trajeto em que se pretende rotacionar um ou
public boolean verificarSeHaNavioNaLocalizacao(Navio navio, int orientacao) {
    if(orientacao == 0) {
        for (int i = navio.coluna; i < navio.coluna+navio.tamanho; i++) {
            if(mapa[navio.linha][i] != 0 && mapa[navio.linha][i] != navio.id) {
                return false;
            }
        }
    } else {
        for (int i = navio.linha; i < navio.linha+navio.tamanho; i++) {
            if(mapa[i][navio.coluna] != 0 && mapa[i][navio.coluna] != navio.id) {
                return false;
            }
        }
    }

    return true;
}

* Este método verifica se um disparo realizado pelo jogador atingiu algum navio no tabuleiro.
public boolean verificarSeAcertouNavio(int linha, int coluna) {
    if (mapa[linha][coluna] != 0) {
        return true;
    }
    return false;
}

* Este método retorna um valor do tabuleiro, para permitir a identificação do navio que foi atingido
public int retornaValor(int i, int j) {
    return this.mapa[i][j];
}
}
```

# CLASSE ABSTRATA NAVIO

```
1 package br.ufrn.imd.modelo;
2
3 * Esta e uma classe abstrata que serve como modelo para as demais classes de navios que a herdarão.
4
5 public abstract class Navio {
6     protected int id;
7     protected int tamanho;
8     protected boolean ativo;
9     protected int orientacao;
10    protected int linha;
11    protected int coluna;
12
13    * Metodo construtor que cria um novo Navio.
14    public Navio() {
15        this.ativo = true;
16        this.orientacao = 0;
17    }
18
19    * Metodo que define a orientacao do navio.
20    public void setOrientacao(int orientacao) {
21        this.orientacao = orientacao;
22    }
23
24    * Metodo que retorna a orientacao do navio.
25    public int getOrientacao() {
26        return this.orientacao;
27    }
28
29    * Metodo que retorna o id do navio.
30    public int getId() {
31        return this.id;
32    }
33
34    * Metodo que retorna o tamanho do navio.
35    public int getTamanho() {
36        return this.tamanho;
37    }
38
39    * Metodo que retorna a condição do navio.
40    public boolean getAtivo() {
41        return this.ativo;
42    }
43
44    * Metodo que define a condição do navio.
45    public void setAtivo(boolean ativo) {
46        this.ativo = ativo;
47    }
48
49    * Metodo que retorna a linha em que o navio esta posicionado.
50    public int getLinha() {
51        return this.linha;
52    }
```

```
80
81 * Metodo que define a linha em que o navio devera ser posicionado.
82 public void setLinha(int linha) {
83     this.linha = linha;
84 }
85
86 * Metodo que retorna a coluna em que o navio esta posicionado.
87 public int getColuna() {
88     return this.coluna;
89 }
90
91 * Metodo que define a coluna em que o navio devera ser posicionado.
92 public void setColuna(int coluna) {
93     this.coluna = coluna;
94 }
95
96 * Metodo que reduz o tamanho o tamanho do navio, se ele for atingido.
97 public abstract void reduzirTamanho();
98
99 * Metodo que define o posicionamento do navio no tabuleiro.
100 public abstract void definirPosicionamento();
101 }
```

# SUBCLASSES DE NAVIO

```
1 package br.ufrn.imd.modelo;
2
3 * Esta a a classe que representa o navio Corveta. Ela e, portanto, uma subclasse da classe abstrata Navio.
4
5
6
7 import java.util.Random;
8
9 public class Corveta extends Navio {
10
11     public Corveta() {
12         this.id = 1; //Inteiro identificador do navio
13         this.tamanho = 2;
14         this.ativo = true;
15     }
16
17
18 * Reduz o tamanho da Corveta, caso ela seja atingida por um jogador.
19
20 public void reduzirTamanho() {
21     this.tamanho--;
22     if (tamanho == 0) {
23         this.ativo = false;
24     }
25 }
26
27
28
29 * Define o posicionamento inicial da Corveta.
30
31 public void definirPosicionamento() {
32     Random random = new Random();
33
34     this.orientacao = random.nextInt(2);
35     if (this.orientacao == 0) {
36         this.linha = random.nextInt(10);
37         this.coluna = random.nextInt(9);
38     } else {
39         this.linha = random.nextInt(9);
40         this.coluna = random.nextInt(10);
41     }
42 }
43
44 }
```



**Corveta**



# SUBCLASSES DE NAVIO

```
1 package br.ufrn.imd.modelo;
2
3 * Esta e a classe que representa o navio Submarino. Ela e, portanto, uma subclasse da classe abstrata Navio.
4
5
6 import java.util.Random;
7
8 public class Submarino extends Navio {
9
10
11 public Submarino() {
12     this.id = 2;
13     this.tamanho = 3;
14     this.ativo = true;
15 }
16
17 * Reduz o tamanho do Submarino, caso ele seja atingido por um jogador.
18
19 public void reduzirTamanho() {
20     this.tamanho--;
21     if (tamanho == 0) {
22         this.ativo = false;
23     }
24 }
25
26
27
28
29 * Define o posicionamento inicial do Submarino.
30
31 public void definirPosicionamento() {
32     Random random = new Random();
33
34     this.orientacao = random.nextInt(2);
35     if (this.orientacao == 0) {
36         this.linha = random.nextInt(10);
37         this.coluna = random.nextInt(8);
38     } else {
39         this.linha = random.nextInt(8);
40         this.coluna = random.nextInt(10);
41     }
42 }
43
44 }
```



Submarino

# SUBCLASSES DE NAVIO

```
1 package br.ufrn.imd.modelo;
2
3 * Esta a a classe que representa o navio Fragata. Ela e, portanto, uma subclasse da classe abstrata Navio.
4
5
6 import java.util.Random;
7
8 public class Fragata extends Navio {
9
10
11 public Fragata() {
12     this.id = 3;
13     this.tamanho = 4;
14     this.ativo = true;
15 }
16
17 * Reduz o tamanho da Fragata, caso ela seja atingida por um jogador.
18
19 public void reduzirTamanho() {
20     this.tamanho--;
21     if (tamanho == 0) {
22         this.ativo = false;
23     }
24 }
25
26 * Define o posicionamento inicial da Fragata.
27
28 public void definirPosicionamento() {
29     Random random = new Random();
30
31     this.orientacao = random.nextInt(2);
32     if (this.orientacao == 0) {
33         this.linha = random.nextInt(10);
34         this.coluna = random.nextInt(7);
35     } else {
36         this.linha = random.nextInt(7);
37         this.coluna = random.nextInt(10);
38     }
39 }
40 }
```



Fragata



# SUBCLASSES DE NAVIO

```
1 package br.ufrn.imd.modelo;
2
3
4 * Esta a a classe que representa o navio Destroyer. Ela e, portanto, uma subclasse da classe abstrata Navio.
5
6
7 import java.util.Random;
8
9 public class Destroyer extends Navio {
10
11     public Destroyer() {
12         this.id = 4;
13         this.tamanho = 5;
14         this.ativo = true;
15     }
16
17
18 * Reduz o tamanho do Destroyer, caso ele seja atingido por um jogador.
19
20 public void reduzirTamanho() {
21     this.tamanho--;
22     if (tamanho == 0) {
23         this.ativo = false;
24     }
25 }
26
27
28
29 * Define o posicionamento inicial do Destroyer.
30
31 public void definirPosicionamento() {
32     Random random = new Random();
33
34     this.orientacao = random.nextInt(2);
35     if (this.orientacao == 0) {
36         this.linha = random.nextInt(10);
37         this.coluna = random.nextInt(6);
38     } else {
39         this.linha = random.nextInt(6);
40         this.coluna = random.nextInt(10);
41     }
42 }
43
44 }
```



Destroyer

# REPOSITÓRIO DE JOGADAS

```
1 package br.ufrn.imd.dao;
2
3 * Esta classe e um repositorio que registra todas as jogadas realizadas por um jogador na partida.
4
5
6
7
8
9 import java.util.ArrayList;
10
11 public class JogadasDAO {
12     private ArrayList<int[]> jogadas; //ArrayList que armazena todas as jogadas realizadas por um jogador.
13
14     * Metodo construtor que cria um novo repositorio de jogadas.
15     public JogadasDAO() {
16         jogadas = new ArrayList<int[]>();
17     }
18
19     * Adiciona ao ArrayList 'jogadas' um vetor com dois inteiros, que representa a jogada realizada pelo jogador, no formato {linha, coluna}.
20     public void addJogadas(int linha, int coluna) {
21         int[] dados = new int[2];
22         dados[0] = linha;
23         dados[1] = coluna;
24         jogadas.add(dados);
25         System.out.println("Jogada inserida com sucesso!");
26     }
27
28     * Verifica se a jogada a ser feita pelo jogador ja foi realizada antes, ou seja, se ja esta cadastrada no ArrayList 'jogadas'.
29     public boolean verificarSeJaHouveEssaJogada(int linha, int coluna) {
30         for(int[] jogada : jogadas) {
31             if (linha == jogada[0] && coluna == jogada[1]) {
32                 return true;
33             }
34         }
35         return false;
36     }
37
38     * Retorna a quantidade de jogadas realizadas pelo jogador, que sera obtida a partir do tamanho do ArrayList jogadas.
39     public int retornaQuantidadeDeJogadas() {
40         return jogadas.size();
41     }
42 }
```

# CLASSES CONTROLLER

```
public class TelaInicialController {
    @FXML
    private Label lblPainel;

    @FXML
    private Button btnIniciar;

    @FXML
    private AnchorPane anchorPane;

    @FXML
    private MenuItem novoJogoMenuItem;

    @FXML
    private MenuItem sairMenuItem;

    @FXML
    private MenuItem comoJogarMenuItem;

    @FXML
    private MenuItem sobreMenuItem;

    String wallpaper = getClass().getResource("/img/wallpaper.png").toExternalForm();

    * Método para realizar a mudança de tela, quando o usuário clicar no botão btnIniciar.
    @FXML
    void onClickBtnIniciar(ActionEvent event) {
        MainApp.mudarTela(2);
    }

    * Método para alterar a cor do botão btnIniciar, quando o mouse estiver dentro do botão.
    @FXML
    void onMouseEntered(MouseEvent event) {
        btnIniciar.setStyle("-fx-border-color: white; -fx-border-width: 2px; -fx-background-color: #999999;");
    }

    * Método para alterar a cor do botão btnIniciar, quando o mouse sair do botão.
    @FXML
    void onMouseExited(MouseEvent event) {
```

```
import java.io.IOException;

public class TelaJogoController {
    @FXML
    private MenuItem sairMenuItem;

    @FXML
    private MenuItem comoJogarMenuItem;

    @FXML
    private MenuItem sobreMenuItem;

    @FXML
    private Label lblJogadasUser;

    @FXML
    private Button btnPodeJogar;

    @FXML
    private Button btnAtirar;

    @FXML
    private Label lblJogadasMaquina;

    @FXML
    private Label lblPainel;

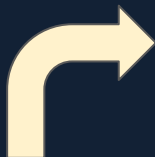
    @FXML
    private GridPane gridPanel;

    @FXML
    private GridPane gridPane2;

    private int linhaClicada;
    private int colunaClicada;
    private Pane ultimaCelulaClicada;
    private boolean turnoUsuario = true; // Variável para controlar o turno do jogador
    private boolean imagemSelecionada = false;
    private String idImagem;
    private final Random random = new Random();
```

# MOVIMENTANDO O NAVIO

```
destroyerVImagem.setOnMouseClicked(event -> {  
    if (event.getButton() == MouseButton.PRIMARY) {  
        selecionarNavio(destroyerVImagem, gridPanel);  
    } else if (event.getButton() == MouseButton.SECONDARY) {  
        rotacionarNavio(destroyerVImagem);  
    }  
});
```

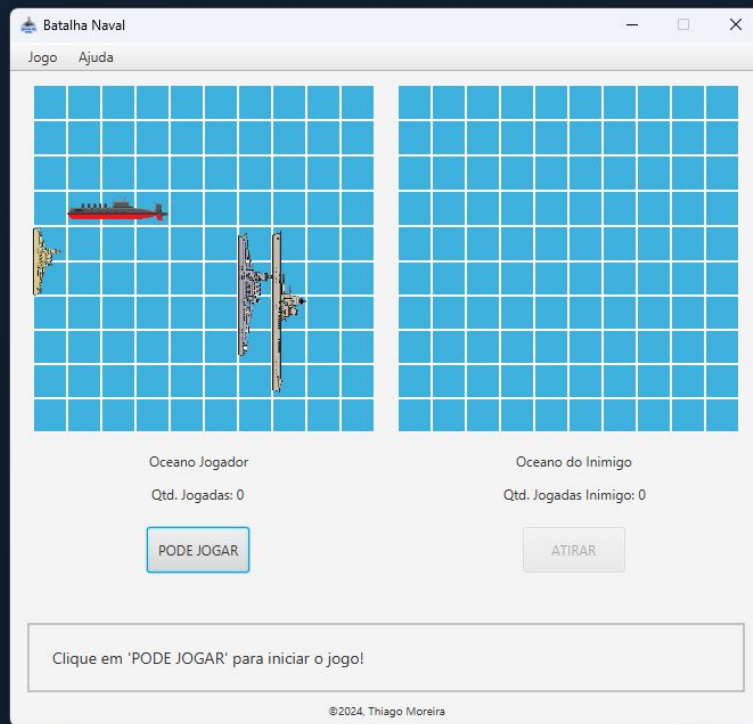
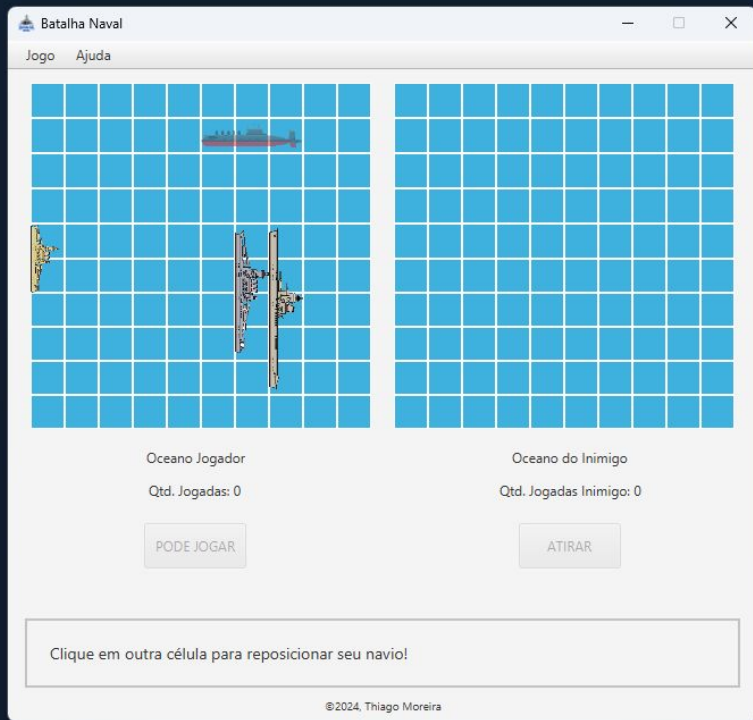


```
private void selecionarNavio(ImageView imagem, GridPane gridpane) {  
    if (imagemSelecionada) {  
        lblPainel.setText("Reposicione primeiro o outro navio!");  
    } else {  
        imagem.setOpacity(0.5);  
        imagemSelecionada = true;  
        idImagem = imagem.getId();  
        lblPainel.setText("Clique em outra célula para reposicionar seu navio!");  
        btnPodeJogar.setDisable(true);  
    }  
}
```

```
private void manipularCelulaClicada(MouseEvent event, int linha, int coluna, boolean primeiroGrid) {  
    if (imagemSelecionada) {  
        if (idImagem.equals("corvetaH")) {  
            if (coluna + corvetaJog.getTamanho() > 10 ||  
                !tabJog.verificarSeHaNavioNaLocalizacao(linha, coluna, corvetaJog)) {  
                lblPainel.setText("Não é possível posicionar a corveta aqui!");  
            } else {  
                tabJog.limparPosicaoAntigaDoNavio(corvetaJog);  
                corvetaJog.setLinha(linha);  
                corvetaJog.setColuna(coluna);  
                tabJog.posicionarNavio(corvetaJog);  
                GridPane.setColumnIndex(corvetaHImagem, corvetaJog.getColuna());  
                GridPane.setRowIndex(corvetaHImagem, corvetaJog.getLinha());  
                corvetaHImagem.setOpacity(1);  
                lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");  
                btnPodeJogar.setDisable(false);  
                imagemSelecionada = false;  
            }  
        } else if (idImagem.equals("corvetaV")) {  
            if (linha + corvetaJog.getTamanho() > 10 ||  
                !tabJog.verificarSeHaNavioNaLocalizacao(linha, coluna, corvetaJog)) {  
                lblPainel.setText("Não é possível posicionar a corveta aqui!");  
            } else {  
                tabJog.limparPosicaoAntigaDoNavio(corvetaJog);  
                corvetaJog.setLinha(linha);  
                corvetaJog.setColuna(coluna);  
                tabJog.posicionarNavio(corvetaJog);  
                GridPane.setColumnIndex(corvetaVImagem, corvetaJog.getColuna());  
                GridPane.setRowIndex(corvetaVImagem, corvetaJog.getLinha());  
                corvetaVImagem.setOpacity(1);  
                lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");  
                btnPodeJogar.setDisable(false);  
                imagemSelecionada = false;  
            }  
        } else if (idImagem.equals("submarinoH")) {  

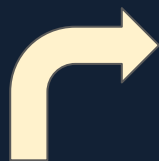
```

# MOVIMENTANDO O NAVIO





# ROTACIONANDO O NAVIO

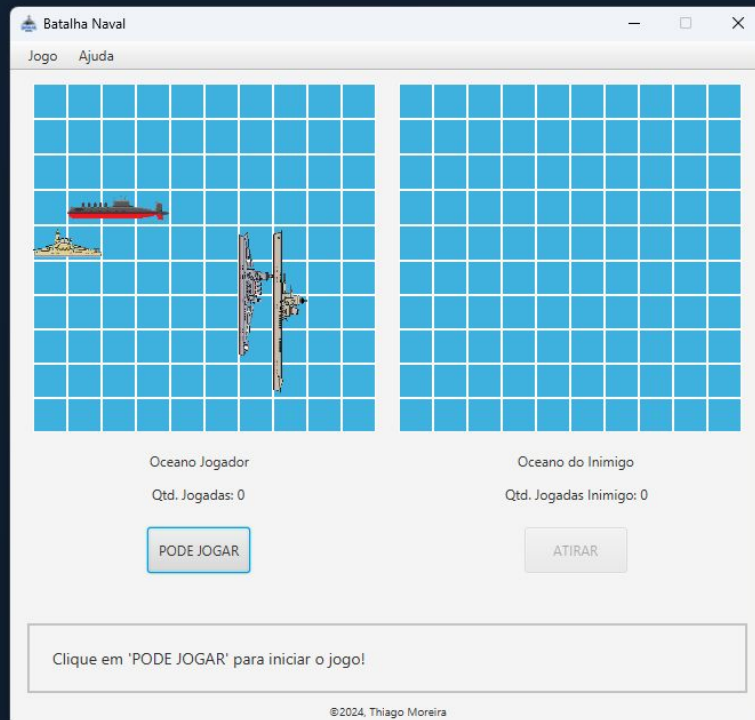
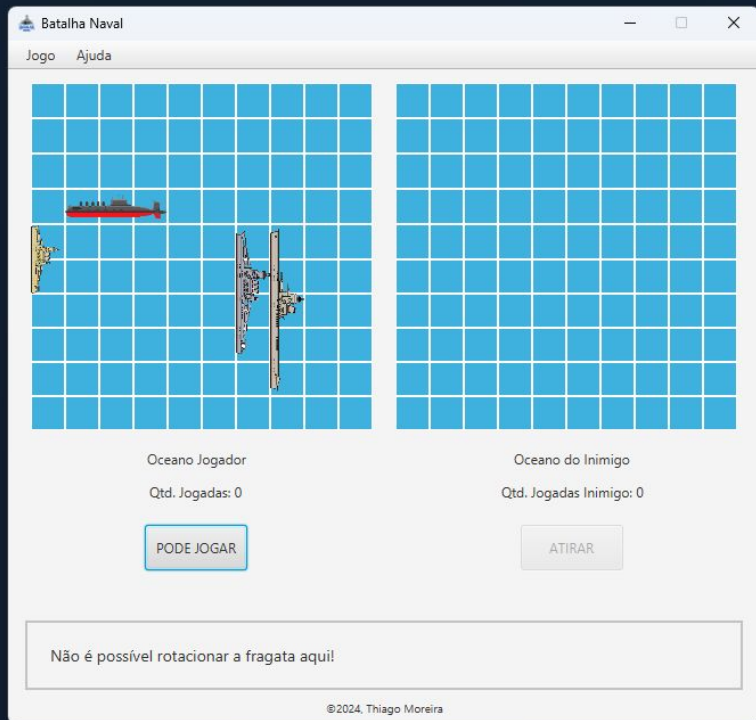


```
corvetaHImagem.setId("corvetaH");
corvetaHImagem.setOnMouseClicked(event -> {
    if (event.getButton() == MouseButton.PRIMARY) {
        selecionarNavio(corvetaHImagem, gridPanel);
    } else if (event.getButton() == MouseButton.SECONDARY) {
        rotacionarNavio(corvetaHImagem);
    }
});
```

```
private void rotacionarNavio (ImageView imagem) {
    if (imagem.getId().equals("corvetaH")) {
        if (corvetaJog.getLinha() + corvetaJog.getTamanho() > 10 ||
            !tabJog.verificarSeHaNavioNaLocalizacao(corvetaJog, 1)) {
            lblPainel.setText("Não é possível rotacionar a corveta aqui!");
        } else {
            tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
            corvetaJog.setOrientacao(1);
            tabJog.posicionarNavio(corvetaJog);
            gridPanel.getChildren().remove(corvetaHImagem);
            GridPane.setColumnIndex(corvetaVImagem, corvetaJog.getColuna());
            GridPane.setRowIndex(corvetaVImagem, corvetaJog.getLinha());
            gridPanel.getChildren().add(corvetaVImagem);
            lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
        }
    } else if (imagem.getId().equals("corvetaV")) {
        if (corvetaJog.getColuna() + corvetaJog.getTamanho() > 10 ||
            !tabJog.verificarSeHaNavioNaLocalizacao(corvetaJog, 0)) {
            lblPainel.setText("Não é possível rotacionar a corveta aqui!");
        } else {
            tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
            corvetaJog.setOrientacao(0);
            tabJog.posicionarNavio(corvetaJog);
            gridPanel.getChildren().remove(corvetaVImagem);
            GridPane.setColumnIndex(corvetaHImagem, corvetaJog.getColuna());
            GridPane.setRowIndex(corvetaHImagem, corvetaJog.getLinha());
            gridPanel.getChildren().add(corvetaHImagem);
            lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
        }
    } else if (imagem.getId().equals("submarinoH")) {
        if (submarinoJog.getLinha() + submarinoJog.getTamanho() > 10 ||
            !tabJog.verificarSeHaNavioNaLocalizacao(submarinoJog, 1)) {
            lblPainel.setText("Não é possível rotacionar o submarino aqui!");
        } else {
            tabJog.limparPosicaoAntigaDoNavio(submarinoJog);
            submarinoJog.setOrientacao(1);
        }
    }
}
```



# ROTACIONANDO O NAVIO



# POLIMORFISMO

```
public boolean verificarSeHaNavioNaLocalizacao(Navio navio) {
    if (navio.orientacao == 0) {
        for (int i = navio.coluna; i < navio.coluna + navio.tamanho; i++) {
            if (mapa[navio.linha][i] != 0) {
                return false;
            }
        }
    } else {
        for (int i = navio.linha; i < navio.linha + navio.tamanho; i++) {
            if (mapa[i][navio.coluna] != 0) {
                return false;
            }
        }
    }
    return true;
}
```

```
public boolean verificarSeHaNavioNaLocalizacao(Navio navio, int orientacao) {
    if(orientacao == 0) {
        for (int i = navio.coluna; i < navio.coluna+navio.tamanho; i++) {
            if(mapa[navio.linha][i] != 0 && mapa[navio.linha][i] != navio.id) {
                return false;
            }
        }
    } else {
        for (int i = navio.linha; i < navio.linha+navio.tamanho; i++) {
            if(mapa[i][navio.coluna] != 0 && mapa[i][navio.coluna] != navio.id) {
                return false;
            }
        }
    }
    return true;
}
```

```
public boolean verificarSeHaNavioNaLocalizacao(int linha, int coluna, Navio navio) {
    if(navio.orientacao == 0) {
        for (int i = coluna; i < coluna+navio.tamanho; i++) {
            if(mapa[linha][i] != 0 && mapa[linha][i] != navio.id) {
                return false;
            }
        }
    } else {
        for (int i = linha; i < linha+navio.tamanho; i++) {
            if(mapa[i][coluna] != 0 && mapa[i][coluna] != navio.id) {
                return false;
            }
        }
    }
    return true;
}
```

# VERIFICANDO SE HÁ NAVIO NA CÉLULA

```
// ----- POSICIONAMENTO DE NAVIOS DE FORMA ALEATÓRIA -----  
// ----- JOGADOR -----  
  
// posicionarNavio(corvetaJog);  
corvetaJog.definirPosicionamento();  
tabJog.posicionarNavio(corvetaJog);  
  
if(corvetaJog.getOrientacao() == 0) {  
    // Define a posição da célula para a ImageView  
    GridPane.setColumnIndex(corvetaHImagem, corvetaJog.getColuna());  
    GridPane.setRowIndex(corvetaHImagem, corvetaJog.getLinha());  
    // Adiciona a ImageView ao GridPane  
    gridPanel.getChildren().add(corvetaHImagem);  
} else {  
    GridPane.setColumnIndex(corvetaVImagem, corvetaJog.getColuna());  
    GridPane.setRowIndex(corvetaVImagem, corvetaJog.getLinha());  
    gridPanel.getChildren().add(corvetaVImagem);  
}  
  
// posicionarNavio(submarinoJog);  
submarinoJog.definirPosicionamento();  
  
while (!tabJog.verificarSeHaNavioNaLocalizacao(submarinoJog)) {  
    submarinoJog.definirPosicionamento();  
  
    tabJog.verificarSeHaNavioNaLocalizacao(submarinoJog);  
}  
  
tabJog.posicionarNavio(submarinoJog);
```

# VERIFICANDO SE HÁ NAVIO NA CÉLULA

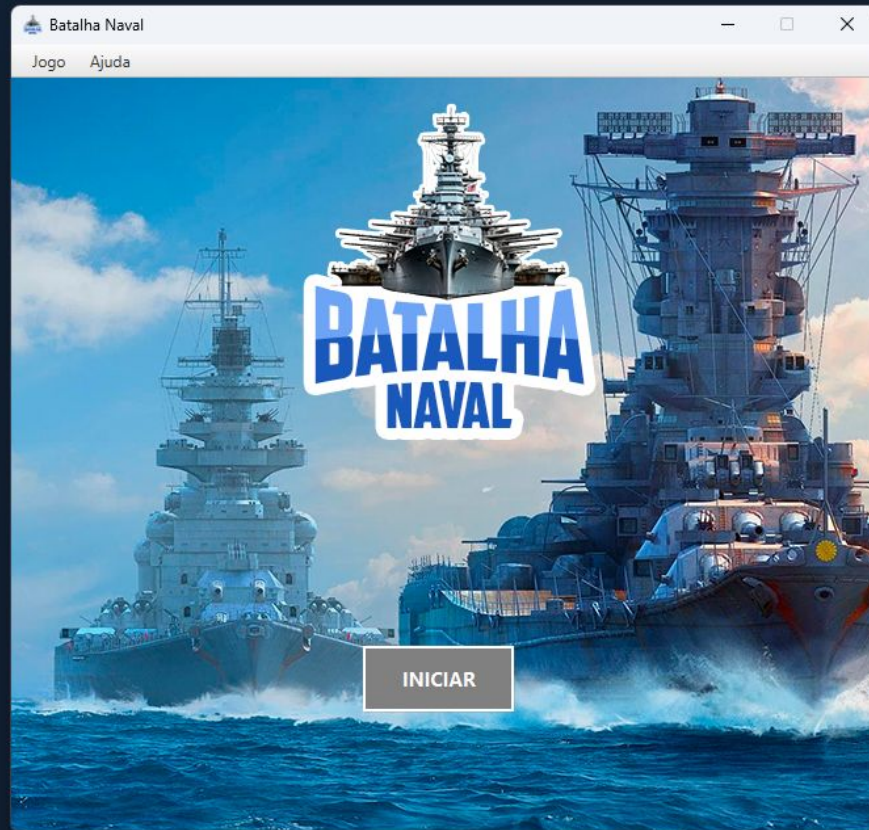
```
private void manipularCelulaClicada(MouseEvent event, int linha, int coluna, boolean primeiroGrid) {
    if (imagemSelecionada) {
        if (idImagem.equals("corvetaH")) {
            if (coluna + corvetaJog.getTamanho() > 10 ||
                !tabJog.verificarSeHaNavioNaLocalizacao(linha, coluna, corvetaJog)) {
                lblPainel.setText("Não é possível posicionar a corveta aqui!");
            } else {
                tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
                corvetaJog.setLinha(linha);
                corvetaJog.setColuna(coluna);
                tabJog.posicionarNavio(corvetaJog);
                GridPane.setColumnIndex(corvetaHImagem, corvetaJog.getColuna());
                GridPane.setRowIndex(corvetaHImagem, corvetaJog.getLinha());
                corvetaHImagem.setOpacity(1);
                lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
                btnPodeJogar.setDisable(false);
                imagemSelecionada = false;
            }
        } else if (idImagem.equals("corvetaV")) {
            if (linha + corvetaJog.getTamanho() > 10 ||
                !tabJog.verificarSeHaNavioNaLocalizacao(linha, coluna, corvetaJog)) {
                lblPainel.setText("Não é possível posicionar a corveta aqui!");
            } else {
                tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
                corvetaJog.setLinha(linha);
                corvetaJog.setColuna(coluna);
                tabJog.posicionarNavio(corvetaJog);
                GridPane.setColumnIndex(corvetaVImagem, corvetaJog.getColuna());
                GridPane.setRowIndex(corvetaVImagem, corvetaJog.getLinha());
                corvetaVImagem.setOpacity(1);
                lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
                btnPodeJogar.setDisable(false);
                imagemSelecionada = false;
            }
        }
    }
}
```

# VERIFICANDO SE HÁ NAVIO NA CÉLULA

```
private void rotacionarNavio (ImageView imagem) {
    if (imagem.getId().equals("corvetaH")) {
        if (corvetaJog.getLinha() + corvetaJog.getTamanho() > 10 ||
            !tabJog.verificarSeHaNavioNaLocalizacao(corvetaJog, 1)) {
            lblPainel.setText("Não é possível rotacionar a corveta aqui!");
        } else {
            tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
            corvetaJog.setOrientacao(1);
            tabJog.posicionarNavio(corvetaJog);
            gridPanel.getChildren().remove(corvetaHImagem);
            GridPane.setColumnIndex(corvetaVImagem, corvetaJog.getColuna());
            GridPane.setRowIndex(corvetaVImagem, corvetaJog.getLinha());
            gridPanel.getChildren().add(corvetaVImagem);
            lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
        }
    } else if (imagem.getId().equals("corvetaV")) {
        if (corvetaJog.getColuna() + corvetaJog.getTamanho() > 10 ||
            !tabJog.verificarSeHaNavioNaLocalizacao(corvetaJog, 0)) {
            lblPainel.setText("Não é possível rotacionar a corveta aqui!");
        } else {
            tabJog.limparPosicaoAntigaDoNavio(corvetaJog);
            corvetaJog.setOrientacao(0);
            tabJog.posicionarNavio(corvetaJog);
            gridPanel.getChildren().remove(corvetaVImagem);
            GridPane.setColumnIndex(corvetaHImagem, corvetaJog.getColuna());
            GridPane.setRowIndex(corvetaHImagem, corvetaJog.getLinha());
            gridPanel.getChildren().add(corvetaHImagem);
            lblPainel.setText("Clique em 'PODE JOGAR' para iniciar o jogo!");
        }
    }
} else if (imagem.getId().equals("submarinoH")) {
    if (submarinoJog.getLinha() + submarinoJog.getTamanho() > 10 ||
        !tabJog.verificarSeHaNavioNaLocalizacao(submarinoJog, 1)) {
        lblPainel.setText("Não é possível rotacionar o submarino aqui!");
    } else {
        tabJog.limparPosicaoAntigaDoNavio(submarinoJog);
        submarinoJog.setOrientacao(1);
        tabJog.posicionarNavio(submarinoJog);
        gridPanel.getChildren().remove(submarinoHImagem);
```

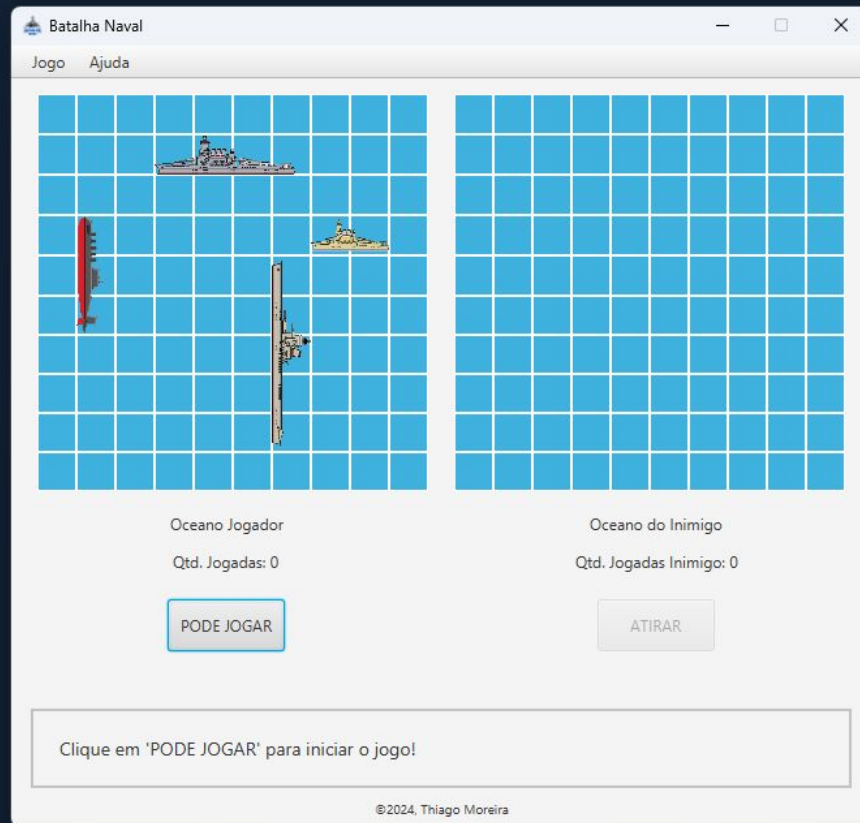


# TELA INICIAL






# TELA DO JOGO



# TELA COMO JOGAR

 Como Jogar

×

## Como Jogar Batalha Naval

**Objetivo do Jogo:**

O objetivo do jogo Batalha Naval é destruir todos os navios do oponente antes que ele destrua os seus.

**Regras:**

1. Tabuleiros:

Cada jogador possui dois tabuleiros de 10x10 células.

Um tabuleiro é usado para posicionar seus próprios navios.

O outro tabuleiro é onde você tenta acertar os navios do seu oponente.

2. Navios:

Cada jogador possui quatro navios de diferentes tamanhos:

- 1x2 (Corveta)
- 1x3 (Submarino)
- 1x4 (Fragata)
- 1x5 (Destroyer)

Os navios podem ser posicionados horizontalmente ou verticalmente, sem sobreposição entre eles.

3. Posicionamento:

Você posiciona seus navios clicando nas células do seu tabuleiro.

Clique para selecionar um navio e, em seguida, clique em outra célula para posicioná-lo.

Botão direito do mouse rotaciona o navio.

4. Atirando:

Para atirar, selecione uma célula do tabuleiro do seu oponente e clique no botão "ATIRAR"

O computador também atira aleatoriamente no seu tabuleiro quando você erra um tiro.

5. Vitória:

O jogo continua até que todos os navios de um jogador sejam destruídos.

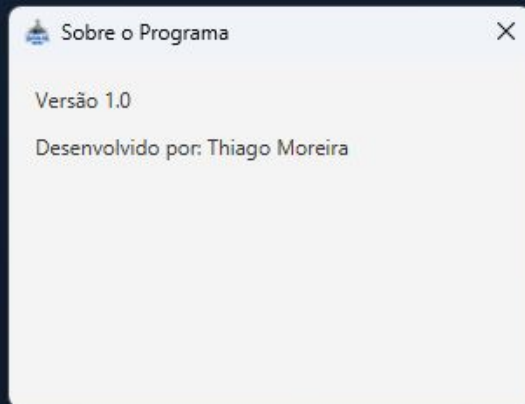
O primeiro jogador a destruir todos os navios do oponente vence.

**Controles:**

Novo Jogo: Reinicia o jogo, permitindo que você posicione seus navios novamente.

Sair: Fecha o jogo.

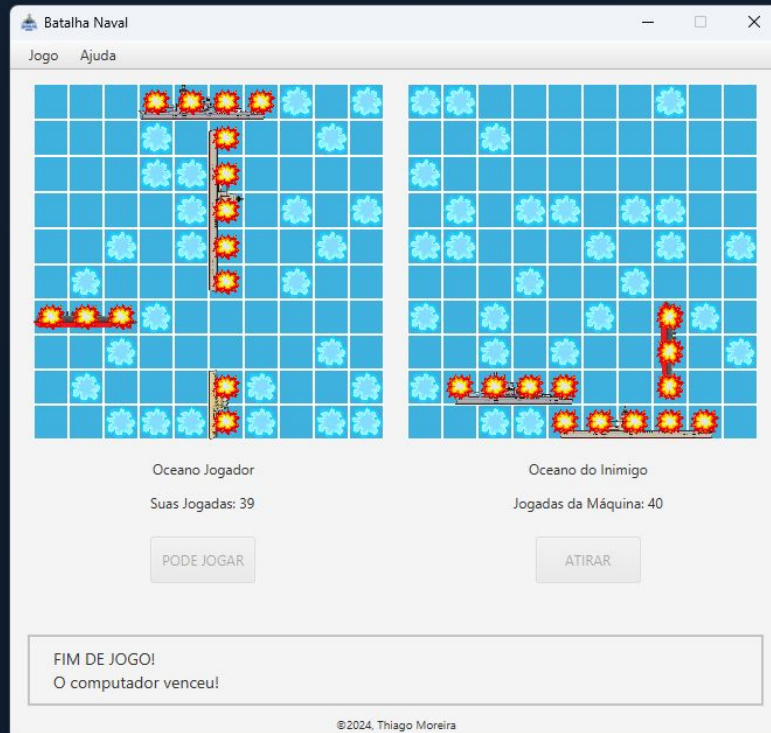
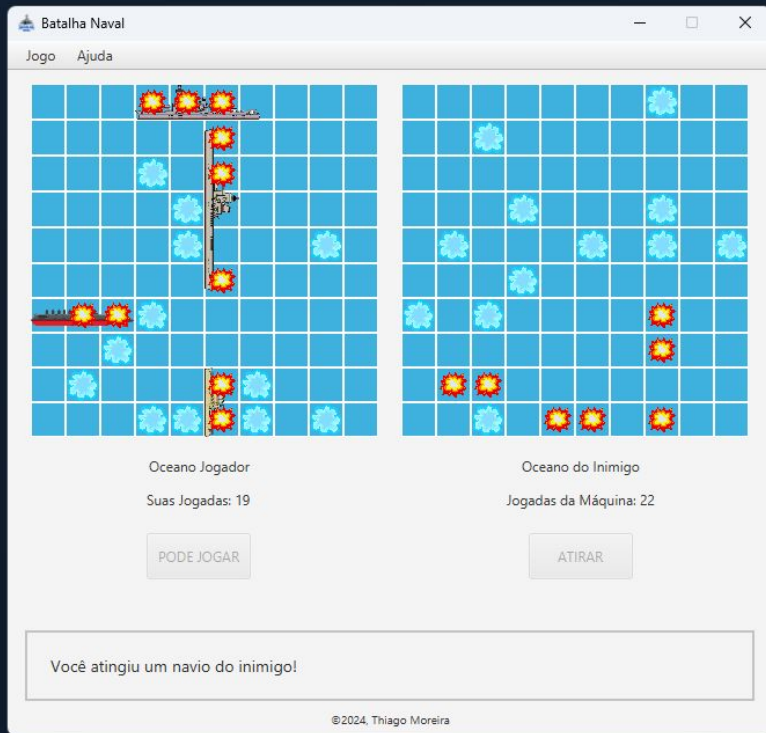
# TELA SOBRE



# ATIRANDO



# NAVIOS AFUNDADOS



# RESETANDO O JOGO

```
/**
 * Metodo para iniciar um novo jogo, ao clicar na opcao 'Novo Jogo' do menu Jogo.
 * @param event
 */
@FXML
void onClickNovoJogo(ActionEvent event) {
    MainApp.restart();
}
```

```
public static void restart() {
    stage.close();
    Platform.runLater(() -> {
        try {
            new MainApp().start(new Stage());
        } catch (Exception e) {
            e.printStackTrace();
        }
    });
}
```



# SAINDO DO JOGO

```
/**
 * Metodo para sair do jogo, ao clicar na opcao 'Sair' do menu Jogo.
 * @param event
 */
@FXML
void sairDoJogo(ActionEvent event) {
    Platform.exit();
}
```

# PONTOS A MELHORAR

- Fazer uma **abstração** melhor no código do **TelaJogoController.java**;
- Melhorar a **inteligência** da jogada do **Computador**.



**OBRIGADO!**