# Relatório do Projeto Final de POO

Batalha Naval – 2020/2 – Profs. Felipe de Souza Marques e Rafael Burlamaqui

Juathan Duarte - 19200989
Henrique Garcia – 19200433
Leonardo Madruga – 17299192

# Introdução

O projeto final da disciplina de programação orientada à objetos consiste na criação e desenvolvimento de um jogo baseado no clássico Batalha Naval, que surgiu em sua forma mais rudimentar por volta do período da primeira guerra mundial. O jogo deve apresentar dois modos, um aleatório e outro personalizado, qual o jogador pode posicionar suas peças onde preferir, enquanto o adversário (computador) sempre terá sua estratégia completamente baseada em diversos *random number generators*. O projeto é executado em turnos, variando entre jogador e computador, e consiste na utilização de diferentes tipos de tiros, disponíveis através da sobrevivência das peças do jogador. A partida termina quando um dos lados fica sem navios, e caso o jogador humano vença, seu tempo de partida é gravado em um arquivo de texto, onde será comparado com uma lista de 15 melhores tempos, funcionando assim, como uma *leaderboard*.

# Diagrama de Classes

Para preservar a formatação do relatório e não ter perda de detalhes relacionados ao diagrama UML, foi anexado ao envio, dentro do arquivo zipado, a imagem do diagrama (pasta raiz).

# Como Compilar e Executar o Programa

1. Extrair o arquivo compactado fornecido pelo grupo;
2. Caso seja de preferência executar facilmente o arquivo:
    a. *Executar o arquivo *.jar localizado na pasta raiz;*
3. Caso seja de preferência executar manualmente o programa:
    a. *Instalar o editor de texto VSCode com o pacote de extensões Java (Java Extension Pack, Code Runner);*
    b. *Abrir o arquivo Main.java e dar Run (F5) ou executar o comando javac Main.java no PowerShell embutido;*

# Descrição breve das Classes

**Main.java**: Serve para invocar a *MainScreen* e, também. reiniciar o jogo na tela de sair;

**MainScreen.java**: Exibe a tela inicial do jogo, onde é possível selecionar "jogo aleatório" ou "definir seu jogo", na qual o usuário também insere seu nome.

**GameScreen.java**: Nessa tela, dividimos em 3 *GridLayouts,* de forma horizontal, renderizados separadamente;

**PlayerSide.java**: Conforme descrito anteriormente, *PlayerSide* é o painel da esquerda da *GameScreen*, contendo o título da página, tipos de tiro (separados em *JButtons* por embarcações), e o tabuleiro do lado do usuário;

**MiddlePanel.java**: O setor mediano do *GameScreen*, onde é possível inserir as embarcações (caso seja selecionado um jogo definido); caso contrário, a área de inserção de peças será desativada (*greyed out)*. Logo abaixo, está posicionado o timer responsável por registrar o tempo da sessão atual, junto com 4 botões:

- *Dica*: são disponibilizadas 3 dicas por partida, onde, ao indicar posição, o programa informa se há embarcações na linha ou coluna do ponto indicado;
- *Limpar Tabuleiro*: Essa opção estará disponível somente se o modo de jogo selecionado for "definido", possibilitando o *reset* das posições das embarcações antes do começo do jogo (em caso de arrependimento/erro do usuário);
- *Jogar*: Iniciará uma partida do modo "definido". Caso contrário, a partida inicia-se automaticamente, tornando o JButton *greyed out*; esse botão também é responsável por iniciar o *timer*;
- *Sair*: Abre uma janela de opções que permite reiniciar o programa, criar um novo jogo, ou sair do programa completamente;

**GameController.java**: Classe que aplica a lógica interna do jogo, sendo um compilado de métodos essenciais para o funcionamento e aplicação das regras do jogo e programa;

**GameState.java**: Armazena o estado atual do jogo (se já foi perdido/vencido, iniciado, ou em ocorrência);

**GameTimer.java**: Responsável pelas funcionalidades do timer (parar, iniciar, resetar);

**GridMaker.java**: Onde são renderizados os tabuleiros de ambos jogador e máquina, diferenciando ambos através de uma flag vinda do construtor chamado pelas classes *PlayerSide* e *ComputerSide*;

**PlayerController.java**: Centraliza as informações do tabuleiro dos jogadores;

**Span.java**: Tela usada para passar informações ao jogador (erros, derrota);

**AudioClip.java**: Classe que, ao ser invocada, executa os áudios (localizados no pacote *Resources*);

**Constants.java**: Lista as letras do grid;

**Context.java**: Serve para instanciar os objetos do jogador e da máquina;

**EventEmitter.java**: Essa função serve pra notificar todos os listeners que aconteceu uma ação;

**Player.java**: Encapsula diversas funções para o funcionamento do jogo, e retornar ao jogador diversas informações relacionadas ao contexto atual da partida, alguns status como disponibilidade de radar, tiros etc. Também é responsável pela "IA" do jogador máquina, onde determina as regras do RNG relacionados aos tiros dela;

**Shot.java**: Classe responsável pelo funcionamento da ação "atirar" do jogo. Consiste em administrar as informações do tiro, coletar informações tais como tipo de tiro, se acertou ou errou etc.;

**ShotType.java**: Forma a silhueta dos tiros, recebe e retorna informações para que seja efetivado o tiro desejado no tabuleiro;

**Vector2.java**: Forma a "estrutura" da definição de um ponto cartesiano (x, y). Responsável como agente direcional para a indexação de qualquer ação que requer um local;

**Vessel.java**: Em primeiro momento, recebemos o tipo e a vida das embarcações (sendo a "vida" determinada pelo tamanho da mesma em espaços do tabuleiro). A *hitbox* (espaço ocupado) da embarcação é determinada pelo intervalo definido pela expressão (ponto inicial da embarcação + tamanho em espaços – 1), em *isInHitbox*. Essa classe também é responsável pelos status individuais das embarcações, indicando se estão vivas (para verificação de disponibilidade de tiro), se foram detectadas pelo sonar (*isInCross*, sendo a "*cross"* (cruz) o alcance da dica (sonar)) etc.;

**VesselTypes.java**: Responsável por armazenar os tipos de embarcações, contendo nome, tamanho e tipo de tiro;

**Score.java**: Serve para instanciar o objeto do jogador no ranking para ser organizado pelo *ScoreManager*;

**ScoreManager.java**: Recebe o *score* atual e organiza o mesmo na lista do arquivo *score.txt*;

**SpanWin.java**: Executa um JFrame para indicar que o jogador venceu e exibe o score;

# Dificuldades Encontradas

- **Tempo**: dois integrantes do grupo são estagiários e tiveram dificuldades em conciliar a carga horária do estágio, a demanda do trabalho e as tarefas de suas demais cadeiras.
- ***Kickstart Logic***: Visto de um plano amplo, em um ponto de vista inicial, a forma em que o trabalho foi proposto foi assustadora. Em primeiro momento, programamos a interface gráfica e, posteriormente, tivemos extrema dificuldade em integrar a lógica com ela.
- **Conciliação de turnos**: Houve dificuldade em administrar a separação entre jogadas do jogador humano e máquina, e de continuidade de jogo, devido a problemas técnicos relacionados à lógica interna de atualização de tabuleiro (por exemplo, a máquina ainda ter um turno próprio após a vitória do jogador).
- **Administração das embarcações em nível individual**: travamos múltiplas vezes perante problemas apresentados na lógica de redução de vida por *vessel*, tendo que lidar inúmeras vezes com *vessels* que morriam antes da hora, a disponibilidade dos tiros não sendo executada corretamente (*greying out* o botão, e não mudando o tipo de tiro baseado nas *vessels* disponíveis etc.).
- **Saturação**: devido ao primeiro problema, citado no início dessa seção, ocorreu que o grupo estava exausto, resultando num índice maior de erros por falta de atenção, fadiga, e mau humor.

# Conclusão

Em um primeiro momento, ficamos com bastante receio da alta demanda do trabalho, tendo visto que nossa real base foram exercícios semanais dos quais a escala é incomparável com o trabalho final. Ao longo das semanas de desenvolvimento, foram aparecendo diversos empecilhos dos quais foram necessárias várias horas de pesquisa e networking com os colegas. Porém, apesar da desconexão entre sala de aula e projeto, devemos admitir que houve uma enorme quantia de aprendizado conforme superamos as barreiras encontradas. O projeto foi desenvolvido ao longo de duas semanas, contabilizando um total superior a 30 horas de desenvolvimento. Foram necessárias diversas refatorações de porções separadas do código (o que nunca é divertido ou estimulante), várias canecas de café e cuias de chimarrão.

# Sugestões para Trabalhos Futuros

Houve um certo desequilíbrio entre a matéria dada em aula de forma expositiva e a porção de conteúdo buscado de forma autodidata, aprendido “na marra”, devido o fato de que, por consequência da proposta do trabalho, maior parte do nosso tempo não foi focando diretamente nos conceitos de programação orientada a objetos (por mais que estivéssemos ativamente trabalhando em cima deles), mas sim, o aprendizado de múltiplas ferramentas quais nunca tínhamos sequer tido algum contato.

# Screenshots

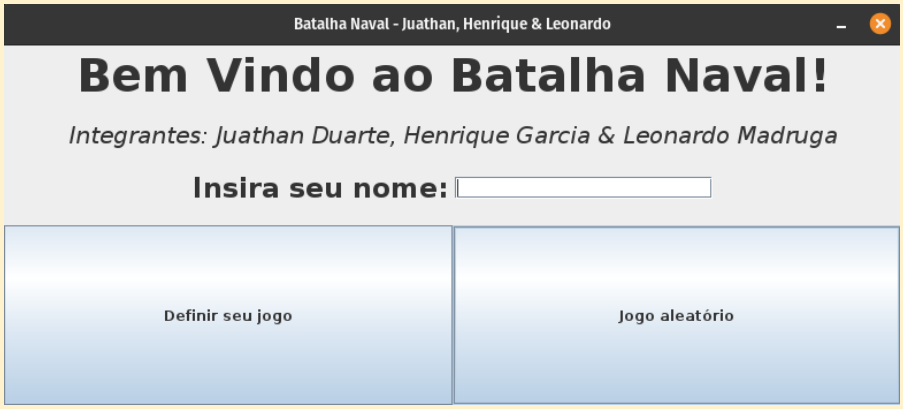


*Figura 1: Tela Inicial*

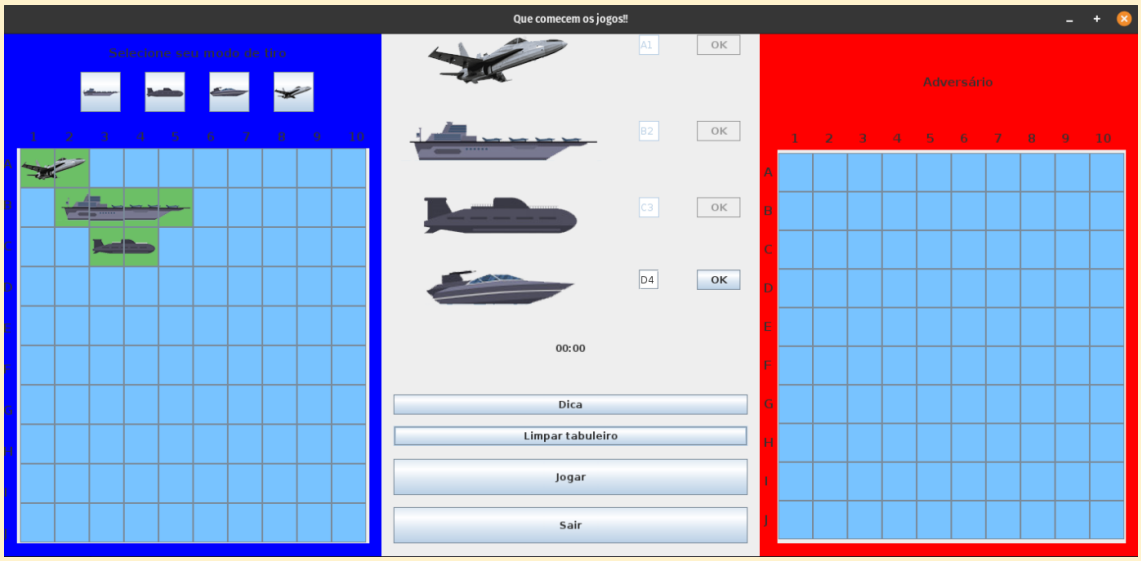


*Figura 2: Jogo Definido*

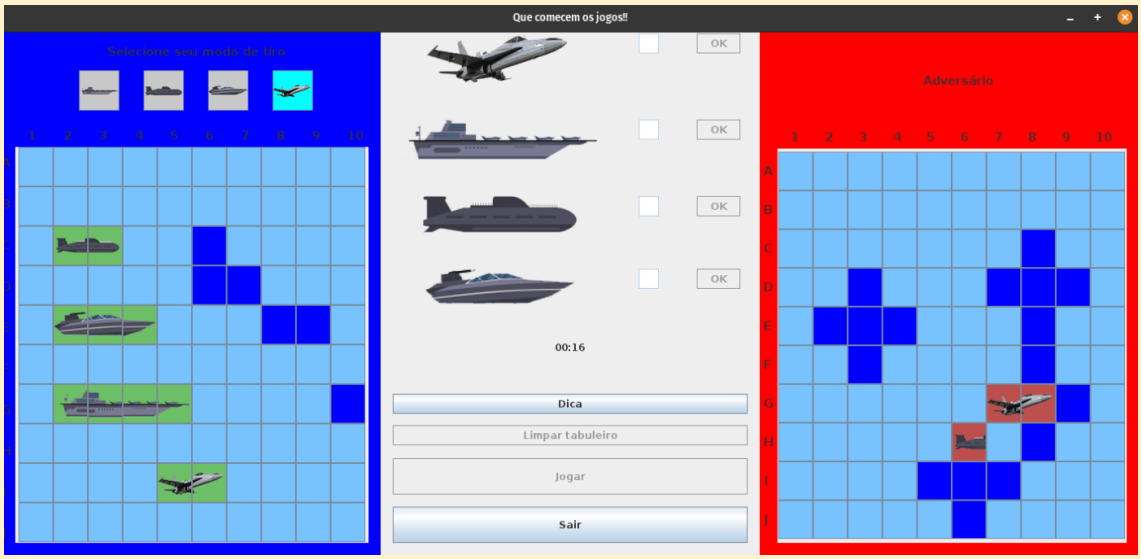


*Figura 3: Jogo Aleatório*

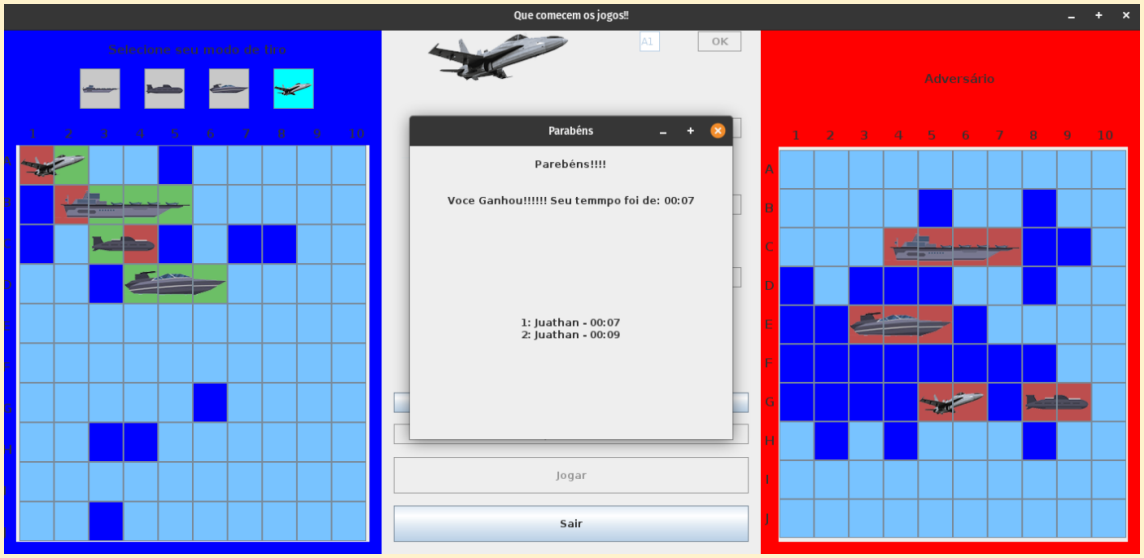


*Figura 4: Tela final/Ranking*