**Batalha Naval**

11055716 – Vinicius Teodoro, 11084915 – Paulo Alexander Simoes, 11000516 – Claudio Quessada Cabello, 11091915 – Gabriel Bellomi Meale

31 de Julho de 2018

**Turma: A4 Noturno**

1. **Introdução**

Desenvolvimento do jogo **Batalha Naval** em Java com conceitos de Orientação a Objetos aprendidos em aula. Tivemos como objetivo o aperfeiçoamento do conhecimento na matéria, aplicando-o à algo real e ao mesmo tempo divertido, com interface gráfica intuitiva e ao mesmo tempo desafiadora no decorrer do desenvolvimento do projeto.

Tentamos implementar o máximo de conceitos de POO possíveis dentro do Projeto, com o intuito de faciliar o desenvolvimento e entendimento do código posteriormente.

1. **Descrição das classes**

* BattleMap: Criação de um mapa de Status na forma de Objeto Enum.
  + Construtor *BattleMap(...)*: Define o tamanho do mapa e seu status por meio de um objeto;
  + *clearMap()*: Limpa o mapa e deixa todos os campos como UNKNOWN;
  + *isValidShipPosition(*...): Valida se é possível um navio ser colocado em dada posição do mapa;
  + Outros 4 métodos são Getters e Setters de Status da posição do tabuleiro;
* Ship: Criação dos navios.
  + Construtor *Ship(...)*: Define o tamanho e id de cada navio;
  + *placeShip(...)*: Posiciona cada navio em determinada posição validando antes se é possível na dita posição;
  + *isSunk()*: Retorna true se todas as posições do navio estiverem com status HIT;
  + *takeShot(...)*: Determina se acertou ou não um navio, se acertou o status daquela posição vira HIT, senão MISS;
  + *blindCone()*: Retorna uma cópia do navio, sem sua posição;
  + Outros 3 métodos são Getters e Setters da posição do navio e Setter do navio afundado;
* Shot: Criação de um Construtor simples *Shot(...)* para setar as configurações iniciais do jogo, com o intuito de os métodos takeShot(...) e outros usem este método;
* Status: Criação de instâncias de status para configuração do Tabuleiro com o passar das rodadas;
* Vector2D: Classe criada para armazenar coordenadas usadas ao longo do jogo.
  + isInBounds(...): Verifica se dada coordenada está dentro do mapa de jogo;
  + *equals(...)*: Compara duas coordenadas;
  + Outros 2 métodos são devoluções (return) de se dada coordenada foi acertada e um Setter de acerto;
* GameController: Classe de controle do Jogo.
  + *start()*: Pede para o jogador setar as configurações iniciais, tais como número e tamanho de navios, etc;
  + *setup()*: Instancia os navios a serem usados;
  + *changeTurn(...)*: Muda os turnos de jogador para jogador;
  + *play()*: Cada jogador posiciona seus navios e continua num loop até verificar que o tiro escolhido na rodada fez tal jogador ganhar, assim finalizando o jogo;
  + *gameOver(*...): Termina o jogo, limpa a tela e pergunta se quer recomeçar;
* Player: Superclasse que configura e seta métodos gerais de jogadores tanto reais quanto IA.
  + Construtor *Player(...)*: Seta o mapa e navios;
  + *shotResults(...):* Atualiza a matriz de batalha com os resultados do tiro;
  + *takeShot(.*..): Se pergunta se algum navio foi atingido;
  + *isWinner():* Se pergunta se seu oponente ganhou;
* HumanPlayer*:* Subclasse de Player. Classe para o Jogador Humano.
  + Construtor *HumanPlayer(...)*: Inicializa atributos/objetos necessários, tais como navios, tamanho do mapa etc;
  + *placeShips()*: Solicita ao jogador que posicione seus navios;
  + *shoot()*: Solicita ao jogador que escolha coordenadas para disparar;
  + *takeShot(...)*: Avisa ao jogador que recebeu um tiro;
  + *startTurn()*: Carrega o mapa do jogador e da informações da rodada passada;
  + *endTurn()*: Limpa a tela e desloga;
* *IAPlayer*: Subclasse de Player. Classe Oponente, Inteligência Artificial.
  + Construtor *IAPlayer()*: Inicializa atributos/objetos necessários, tais como navios, tamanho do mapa etc;
  + *placeShips()*: Posiciona navios de forma aleatória e válida;
  + *putShipOnGrid()*: Posiciona navios no tabuleiro e atualiza status;
  + *shotResults()*: Alterna entre modo de procura e perseguição;
  + *updateStrategy()*: Descobre se há navios a serem perseguidos no mapa;
  + *shoot()*: Atira no ponto de maior chance de acertar um navio, sorteando em caso de empate;
  + *placeShips()*: Posiciona navios de forma aleatória e válida;
  + *endTurn()*: Fechar tela do IA;
  + *startTurn()*: Exibe informações enquanto IA joga;
* *ProbabilityMapDrawer*: Classe abstrata que fornece o mapa de probabilidades para a IA programar seus disparos..
  + Construtor *ProbabilityMapDrawer (...)*: Inicializa o mapa de probabilidades e armazena os objetos necessários, a lista de navios e o mapa de batalha.
  + *clearMap()*: Limpa o mapa de probabilidades, chamado antes de recalculá-lo;
  + *update()*: Método abstrato, recalcula e devolve à IA as probabilidades.
* *User*: Classe que associa um nome e senha a um jogador para este usar seu turno.
  + Construtor *User(...)*: Armazena os objetos necessários, jogador, nome e senha.
  + *isPasswordCorrect (...)*: Verifica se a String fornecida confere com a senha.
  + Outros métodos são 2 getters, do jogador e do nome, e 1 setter, da senha;
* *UserController*: Classe faz a comunicação direta com o usuário no console, configurando o jogo e alternando entre os jogadores através de login.
  + Construtor *UserController()*: Inicializa os objetos necessários.
  + Diversos métodos de interação com o usuário.

1. **Conceitos de orientação a objetos aplicados**

No decorrer do Projeto foram usados diversos conceitos aprendidos em aula e extraclasse também.

Talvez o conceito mais utilizado tenha sido o de **Encapsulamento**, para manter o código sempre o mais protegido de falhas possível.

Conceito de **Herança** apesar de pouco usado foi extremamente importante em seu uso por ajudar na eficiência do código, não necessitando a repetição de métodos; Para a criação dos *Jogadores* este conceito foi essencial, tendo uma **Superclasse** chamada *Player* que comanda e esta munida de métodos gerais que um Jogador possa ter, seja ele *Real* ou *IA*, e **Subclasses** para *Jogador Humano* ou *IA*, com métodos e atributos específicos para cada um.

Conceitos como **super** e **objeto** foram primordiais para a criação da Lógica *IA*, para poder usar os navios (*myShips*) da classe *Player*, por exemplo, e criar na váriavel *probabilityMap* um objeto de criação do mapa.

Outros conceitos não tão usados, mas que ajudaram no desenvolvimento foram **static**, que auxiliou no acesso de objetos como *GameController;* **final** para assegurarnos que este mesmo objeto *GameController* não seja alterado posteriormente e **abstract** para a criação da classe *Player.*

Foram usados também padrões de projeto, o *GameController* é um **Singleton**, pois é final, tem construtor privado, e método *public GameController getInstance()*. As classes que herdam de *ProbablilityMapDrawer* formam um padrão **Strategy**, pois a classe *IAPlayer* utiliza uma classe abstrata genérica, e alterna as instâncias de *ProbablilityMapDrawer* armazenadas, conforme necessidade, chamando um método implementado diferentemente em cada uma das subclasses.

O código segue indentação usual, e está bem explicada em seus comentários. A comunicação do usuário foi implementada diretamente no console. Estávamos desenvolvendo a aplicação em telas do *Java.swing*, mas ficamos sem tempo hábil.

1. **Participação de cada integrante do grupo**

Apesar de cada integrante do grupo ter feito uma parte distinta no decorrer do desenvolvimento do projeto para eficiência no prazo, foi de extrema importância a cooperação e entendimento do código inteiro por todos os membros, portanto todos participaram pelo menos um pouco em outras partes do código que não fez.

A parte de *modelagem de navios*, *tiros*, s*tatus do tabuleiro* e *tabuleiro* (trabalhoPOO/Naval Battle/src/br/edu/ufabc/poo/model) foram feitos primariamente pelo Vinicius;

A metodologia dos Jogadores por ser um pouco mais complexa por conta da IA foi feita pelo Claudio e Paulo (trabalhoPOO/Naval Battle/src/br/edu/ufabc/poo/players);

A *estrutura de estratégia* e *probabilidades de acerto no tabuleiro* (trabalhoPOO/Naval Battle/src/br/edu/ufabc/poo/strategy) foram feitas pelo Gabriel;

A parte final que comanda o código todo (classe *GameController*) foi feita em conjunto pelo Vinicius e Gabriel;

Em relação ao relatório, para não ficar exaustivo, foi feito em conjunto ao final do projeto por todos os membros.

Vale lembrar que o código todo teve a participação de todos os integrantes, ora alguém tirando dúvida de um e vice-versa.