UNIVERSIDADE FEDERAL DE PELOTAS
Curso: Ciência da Computação
Disciplina: Programação Orientada a Objeto
Professor: Felipe Marques

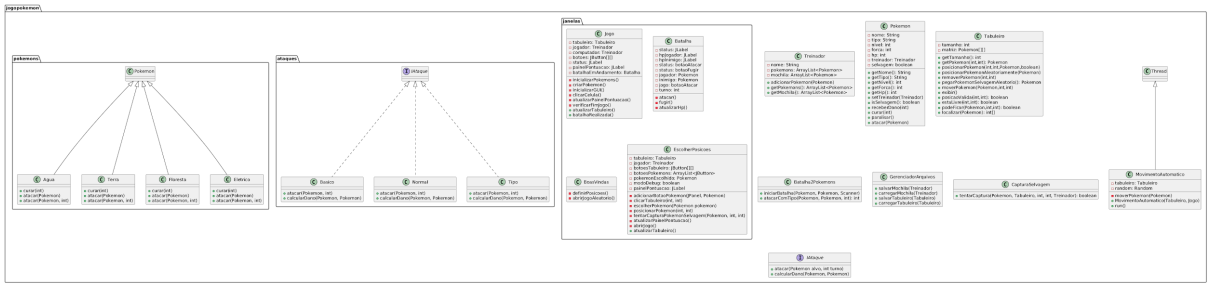
Relatório do Trabalho Final — Jogo Pokémon

Discentes: Larissa Gabriela Barrozo e Vinicius Campos

INTRODUÇÃO:

O tema do jogo é inspirado na franquia Pokémon, em que o jogador assume o papel de um treinador que captura e treina os Pokémons para competir em batalhas. O objetivo principal é completar a Pokédex, capturando todos os Pokémons disponíveis, e treinar o time para obter a maior pontuação de experiência possível. A implementação foi realizada seguindo os princípios de **Programação Orientada a Objetos (POO)**, garantindo que as classes fossem independentes e as interações entre objetos ocorressem nas regras do jogo pedidas.

DIAGRAMA DE CLASSES



CONCEITOS:

A parte central do jogo é a manipulação de um tabuleiro dividido em uma grade NxN, subdividida em quatro regiões específicas para cada tipo de Pokémon: Água, Floresta, Terra e Eletricidade. Cada célula da grade representa um espaço que pode conter um Pokémon, um objeto do cenário ou estar vazio.

Tratar as exceções: RegiaoInvalidaException onde teremos certeza de que cada tipo de pokémon fique em sua respectiva área e NenhumaPosicaoDisponivelException **extends** RuntimeException que define uma exceção personalizada para tratar posicionamento inválido. Foi criada usando captura de exceção com throw/try-catch e aplicada nos ***.java*** necessários.

A classe **Main.java** é o ponto de entrada do programa. Ele simplesmente instancia a janela BoasVindas, que provavelmente abre a primeira interface gráfica do jogo. Onde iremos compilar e executar o jogo. O main só inicia a aplicação, enquanto o resto da lógica fica em outras classes.

Usamos de **classes abstratas (Pokemon.java) .** Onde a classe abstrata Pokemon contém atributos comuns como força, nível, experiência, energia e tipo, bem como métodos gerais, enquanto as subclasses (PokemonAgua, PokemonTerra, PokemonFloresta, PokemonEletrico) implementam a

interface IAtaque para definir estratégias de ataque específicas e habilidades únicas de cada tipo, como foi solicitado no pdf, isso permitiu aplicar o **polimorfismo.** O encapsulamento (Atributos hp, ataque, defesa com métodos getters/setters em Pokemon) que protege o acesso direto aos atributos, permitindo controle sobre a leitura e alteração deles.

Método atacar() é sobrescrito nas subclasses de Pokemon, onde objetos de diferentes subclasses podem ser tratados de modo uniforme durante a execução das batalhas.

A interface gráfica **- IAtaque** - foi construída utilizando botões para representar cada célula, permitindo que o jogador realize ações de captura, batalhas ou movimentos. Inclusive a região inválida do qual, por exemplo, Pokémon da terra não pode ir na região do Pokémon da água.

Interface **HabilidadeEspecial** implementada pelas classes Água, Terra, Fogo, Floresta) Obriga cada tipo a implementar sua própria habilidade especial, garantindo um contrato de métodos.

A interface **EscolherPosicoes.java**, cria a interface gráfica inicial para o jogador escolher e posicionar seus Pokémons no tabuleiro antes de iniciar o jogo. Permite selecionar entre Pikachu, Bulbasaur e Squirtle, clicar em células do tabuleiro para posicioná-los, além de capturar Pokémons selvagens. A classe centraliza a fase de configuração do jogo, permitindo ao jogador montar seu time de forma interativa. O uso de exceções personalizadas assegura regras de posicionamento, enquanto a associação com outras classes garante que a lógica do jogo (captura, tabuleiro, movimento) seja reutilizada. Em termos de conhecimento de POO, temos :
Encapsulamento → uso de atributos privados (tabuleiro, jogador, botoesTabuleiro) com acesso controlado por métodos.
Associação entre classes → integração com Tabuleiro, Treinador, Pokemon e suas subclasses (Agua, Eletrico, Floresta, Terra).
Polimorfismo → o método posicionarPokemon() recebe qualquer objeto que herde de Pokemon
Eventos (GUI com Swing) para interação do jogador
Exceções personalizadas → RegiaoInvalidaException garante que cada Pokémon só ocupe sua região válida.

Falando um pouco mais de Batalha2Pokemons ,além das batalhas entre dois Pokémons, foi implementada a funcionalidade de batalha entre somente um Pokémon do jogador contra um Pokémon adversário (Batalha1Pokemon). Essa versão simplificada permitiu focar na interação direta entre tipos de ataque e HP, utilizando cálculos de dano que consideram força, nível e multiplicadores de tipo. O código também incorpora efeitos especiais específicos de cada tipo de Pokémon, como paralisação, regeneração parcial de HP ou aumento de dano em determinadas condições, aproximando a mecânica das batalhas do universo Pokémon.

Sobrecarga de métodos (método batalhar() em Batalha2Pokemons), sobrecarregado com diferentes parâmetros, permite usar o mesmo nome de método para variações de batalha, mudando somente os parâmetros.

A classe **Jogo.java,** gerencia a fase principal do jogo em um tabuleiro 6x6 e se concentra na lógica central da partida, ela representa a janela principal onde o jogador interage com o tabuleiro. O uso de herança e polimorfismo permite lidar com diferentes tipos de Pokémon de maneira unificada, enquanto a thread dá autonomia ao computador. Controla o jogador e o computador, permite capturar Pokémons selvagens, iniciar batalhas entre treinadores e atualiza a interface gráfica. Também implementa o movimento automático do computador. Também usa o **Encapsulamento** para atributos

privados tabuleiro(instância do tabuleiro que guarda os Pokémon.), jogador e computador são treinadores, botoes (matriz de JButton que representa o tabuleiro na GUI.), tatus → JLabel que mostra mensagens (ex.: captura ou batalha) e batalhaEmAndamento (controla se há uma batalha ativa) e estes atuam protegendo o estado do jogo. **Herança** nas classes de Pokémons (Agua, Terra, Floresta, Eletrico) derivam de Pokemon. **Polimorfismo** no uso de switch e objetos de diferentes subclasses tratados de forma genérica como Pokemon. **Associação entre classes** que integra Tabuleiro, Treinador, Pokemon, CapturaSelvagem e Batalha.**Threads** no método iniciarMovimentoComputador() roda em paralelo para simular o computador capturando Pokémon e o **Tratamento de eventos (GUI)** na interação por cliques em botões usando. Seus métodos:
ActionListener.inicializarPokemons() → cria Pokémon iniciais do jogador e computador + Pokémon selvagens aleatórios.
criarPokemon(tipo, nome, selvagem) → fábrica simples para instanciar Pokémon pelo tipo.
inicializarGUI() → monta a interface gráfica com botões e layout.clicarCelula(linha, coluna) → lógica de interação ao clicar: Mostra mensagem se não há Pokémon. e Captura Pokémon selvagem (50% de chance) ou inicia batalha com Pokémon adversário.
atualizarTabuleiro() → atualiza os nomes (ou símbolos) dos Pokémon na GUI.
batalhaRealizada() → chamada quando uma batalha termina, limpa o estado de batalha.
iniciarMovimentoComputador() → thread que faz o computador capturar Pokémon selvagens a cada 4s automaticamente.
verificarFimJogo() → avalia se ainda existem Pokémon selvagens; se não, declara o vencedor.

Ou seja, a classe **Jogo.java**: mostra o tabuleiro com os Pokémon., permite clicar nas células para batalhar ou capturar, controla a pontuação do jogador e do computador, Inicia o movimento automático do computador (IA).

A classe Pokedex.java gerencia a lista de Pokémons capturados pelo jogador. Permite adicionar novos registros, listar detalhes dos capturados e consultar o total registrado , com ideia de completar a Pokédex, capturando todas as espécies Pokémon disponíveis. Usou de **Encapsulamento** na lista capturados é privada, acessada apenas por métodos da classe.**Associação** ,pois relaciona a Pokedex com objetos da classe Pokemon. A ideia central foi manter organizado o histórico de capturas, centralizando informações do progresso do jogador. O encapsulamento garante que apenas métodos definidos controlem os registros, evitando manipulação direta e preservando a integridade dos dados.

A classe **GerenciadorArquivos.java** é a responsável por salvar e carregar o estado do jogo em arquivos de texto. Permite gravar/restaurar tanto a mochila do treinador (Pokémons capturados) quanto o tabuleiro (posições e status dos Pokémons).

O **Tabuleiro.java** implementa **Grid N x N para** armazena os Pokémons no tabuleiro. Teve de separar as **regiões por tipo (** Água: superior-esquerdo, Floresta: superior-direito, Terra: inferior-esquerdo ,Elétrico: inferior-direito). **Posicionar e movimentar com ,** posicionarPokemon e posicionarPokemonAleatoriamente e moverPokemon com checagem de região e se a posição está livre . A ideia de **Remoção:** removerPokemon(linha, coluna) remove do grid, ao ser capturado. **Busca e seleção de Pokémons selvagens:** temPokemonsSelvagens() e pegarPokemonSelvagemAleatorio() .**Debug / exibição do grid:** exibir() imprime os tipos no console .**Validações :**posicaoValida e estaLivre. Também Mantém **listas separadas por região** (regiaoAgua, regiaoFloresta, regiaoTerra,

regiaoEletrico) além do grid, e, **Integração com captura**: ao capturar pokemon, remove do grid e da lista da região. **Integração com treinador**: mantém referência ao treinador dono do Pokémon.

A classe **CapturaSelvagem.java,** Define a lógica de captura de Pokémon selvagens. Dá uma chance (50%) de adicionar o Pokémon ao time do treinador e removê-lo do tabuleiro. Utilizando da aleatoriedade que simula o lançamento da Pokébola. Caso a captura falhe, o Pokémon selvagem é reposicionado em uma célula adjacente não ocupada. Para batalhas entre treinadores, o cálculo de dano considera a força, o nível, a experiência e a habilidade especial de cada Pokémon. As batalhas ocorrem em rounds, e a energia dos Pokémons é restaurada após cada confronto. Em termos de conteudos de POO, o **encapsulamento** é toda a lógica de captura está em um único método (tentarCaptura), sem expor detalhes para outras classes e garante que a mecânica de captura seja consistente em todo o jogo. **Associação** interage diretamente com Pokemon, Tabuleiro e Treinador. O **Uso de estado** que modifica atributos de objetos (setSelvagem(false), setTreinador…), alterando o papel do Pokémon dentro do jogo e Método estático Centralizar a captura em uma classe estática evita duplicação de código e facilita manutenção.

Inclusão de pausas com Pressione ENTER para continuar... para melhorar a experiência do usuário e a leitura dos turnos;

A classe **JogadorComputador.java** foi implementada com **Threads**, permitindo que o treinador virtual faça suas ações junto do ao jogador humano. Pequenos atrasos (Thread.sleep()) são o tempo de decisão do computador. Ou seja, controla o comportamento automático do treinador computador. Ele age em paralelo (thread), atacando Pokémons selvagens aleatórios a cada 4 segundos e capturando-os quando nocauteados. Usamos de **Herança** , herda de `Thread` para criar um jogador que roda em paralelo ao humano. **Encapsulamento** onde a lógica da jogada do computador está toda isolada dentro do run(). **Polimorfismo** quando usa o método `atacar` dos diferentes tipos de Pokémon, que podem sobrescrever o comportamento de ataque.

O sistema de experiência e evolução de Pokémons foi implementado, incluindo incremento de nível ao atingir experiência, e isto potencializa os ataques. A interface exibe a pontuação dos times em tempo real, e mensagens informam quando a energia de um Pokémon chega a zero, encerrando ou avançando a rodada conforme o caso.

EXTRA: Padronização de Ataques (Strategy Pattern)

- O padrão Strategy encapsula cálculos de dano e efeitos especiais, permitindo alterar o comportamento de ataque sem modificar a lógica central das batalhas. Isso facilita manutenção, expansão do jogo e aplicação uniforme de polimorfismo. Foi aplicado em formas de Ataque: AtaqueTipo, Basico e Normal.

PASSOS PARA EXECUTAR

- Sistema Operacional: Windows
- IDE: IntelliJ

Compilação e execução:

1. Abrir o projeto no IntelliJ.
2. Configurar SDK e nível de linguagem
3. Build → Build Project (Ctrl+F9).
4. Executar main da classe principal (Run 'NomeDaClasse.main()').

EXEMPLOS DE UTILIZAÇÃO:

Figura 1 - Tela de boas-vindas do jogo. O jogador possui três opções.

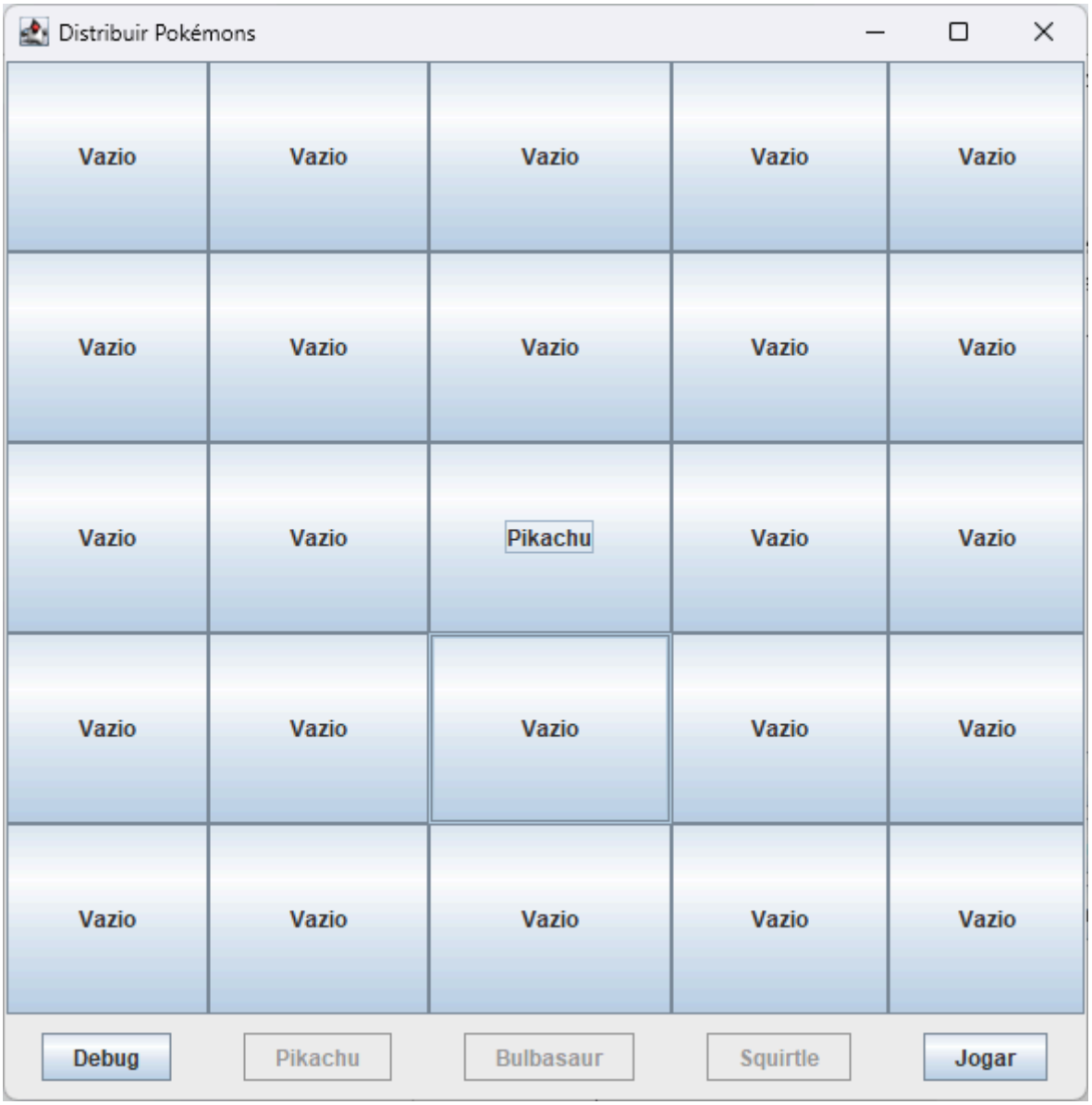


Figura 2 - Tela de posicionamento manual. O jogador possui uma escolha entre três Pokémons para o seu time.

DESAFIOS:

Entre os desafios, teve a implementação das regras de movimentação e posicionamento de Pokémons e do treinador, quando teve de tirar da posição fixa, garantindo que não houvesse sobreposição de posições e respeitando as restrições de região inválida, exigindo atenção à comunicação entre objetos (do qual constantemente apresentava algum erro).

Também foi desafiador fazer funcionar a **Batalha2Pokemons** progressivamente conforme o tipo de Pokémon, pois cada tipo possuía diferentes danos, ataques e efeitos especiais. Transformar isso em uma batalha completa por turnos, com atualização de HP, pontuação e aplicação de habilidades especiais, exigiu muito esforço e ainda, foi necessário garantir que, ao final da batalha, o sistema

voltasse corretamente ao menu principal, respeitando pausas e acionamento do botão ENTER pelo usuário. Do qual, inicialmente, não dava certo.

Outro desafio e um dos mais importantes foi a **sincronização entre múltiplas threads** — principalmente na movimentação automática do treinador computador e na captura aleatória de Pokémons selvagens.

A criação da **interface principal** do jogo também exigiu atenção: integrar botões do tabuleiro, status do jogo, pontuação, captura e batalhas em tempo real, enquanto a lógica de POO permanecia modular e independente, foi um desafio de design e implementação. Além disso, o gerenciamento de eventos do usuário, como cliques em células ou execução de batalhas, teve que se comunicar corretamente com todas as classes envolvidas.

CONCLUSÃO

O projeto implementa com sucesso a lógica central do jogo Pokémon, incluindo captura, batalha, evolução e controle de pontuação, aplicando conceito de POO de forma prática. Embora algumas partes tenham sido limitadas por tempo, a execução principal não foi comprometida.

Melhorias e lições aprendidas:
Ficou claro que, mesmo com código funcional, a manutenção e expansão exigem planejamento cuidadoso e muito demorado. Em futuras versões, seria interessante adicionar novos tipos de Pokémons, habilidades especiais mais complexas, animações mais elaboradas na interface e maior interação entre jogadores,uma mochila com inventário. Além disso, otimizar threads e eventos para reduzir possíveis conflitos melhoraria a experiência.