UNIVERSIDADE ESTADUAL DE CAMPINAS
INSTITUTO DE COMPUTAÇÃO

Relatório - Projeto Final

MC322: Programação Orientada a Objetos

Professor: Marcos Raimundo

PokeJava

Enzo Ferreira (RA 243161)
Larissa Fazolin (RA 217395)
Vitor Takahashi (RA 231740)

1. Descrição Geral

O software a ser desenvolvido neste projeto é uma versão simplificada do jogo Pokémon. Para aplicar os conceitos de Programação Orientada a Objetos usando a linguagem Java, foi definido um jogo em que o usuário participa de um loop de batalhas contra pokémons selvagens, podendo capturá-los, ganhar itens, formar sua própria equipe pokémon e aumentar seus níveis conforme o progresso, podendo inclusive ver as evoluções de seus pokémons favoritos!

Para apresentar o projeto, este relatório foi organizado em seis seções (esta inclusa). Em **Planejamento**, são descritos os passos iniciais, incluindo o uso de User Flow e prototipagem. Em **Classes e Relacionamentos**, explica-se a estrutura das classes e funcionalidades do software. Em **Design Patterns** e **Tratamento de Exceções**, é feita uma análise das decisões de design e tratamento de erros. Finalmente, em **Software Final**, é apresentada a versão concluída do projeto.

2. Planejamento

Antes de iniciar o projeto, foi necessário realizar um planejamento prévio para decidir quais ações o usuário poderia realizar durante o jogo. Para isso, foi estruturado um User Flow, que auxiliou na organização das demais tarefas (definição de classes, atributos, métodos e relacionamentos). A *Figura 1*, por exemplo, apresenta o fluxo do Menu Inicial do

jogo. Também foram criados fluxos completos do turno do jogador e do Pokémon selvagem, que podem ser vistos no link do FigJam.

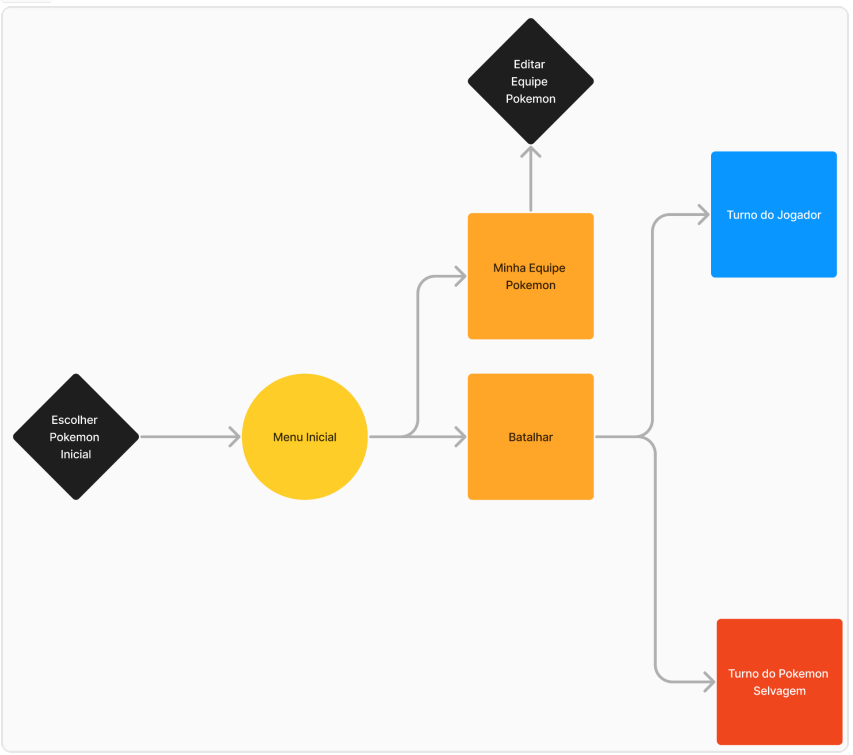

Figura 1: User Flow do fluxo do Menu Inicial do Projeto PokeJava.

Além do User Flow, também foi criado um protótipo das telas do jogo, que seriam desenvolvidas usando a interface gráfica JavaFX. A *Figura 2* apresenta a tela de batalha, e o protótipo completo pode ser acessado pelo link do Figma.

Figura 2: Protótipo da tela de batalhas do Projeto PokeJava.

3. Classes e Relacionamentos

Após o planejamento do escopo do projeto, o próximo passo foi organizar e estruturar o diagrama UML para servir como guia. A *Figura 3* apresenta este diagrama, que também pode ser acessado pelo link do Lucid para melhor visualização.

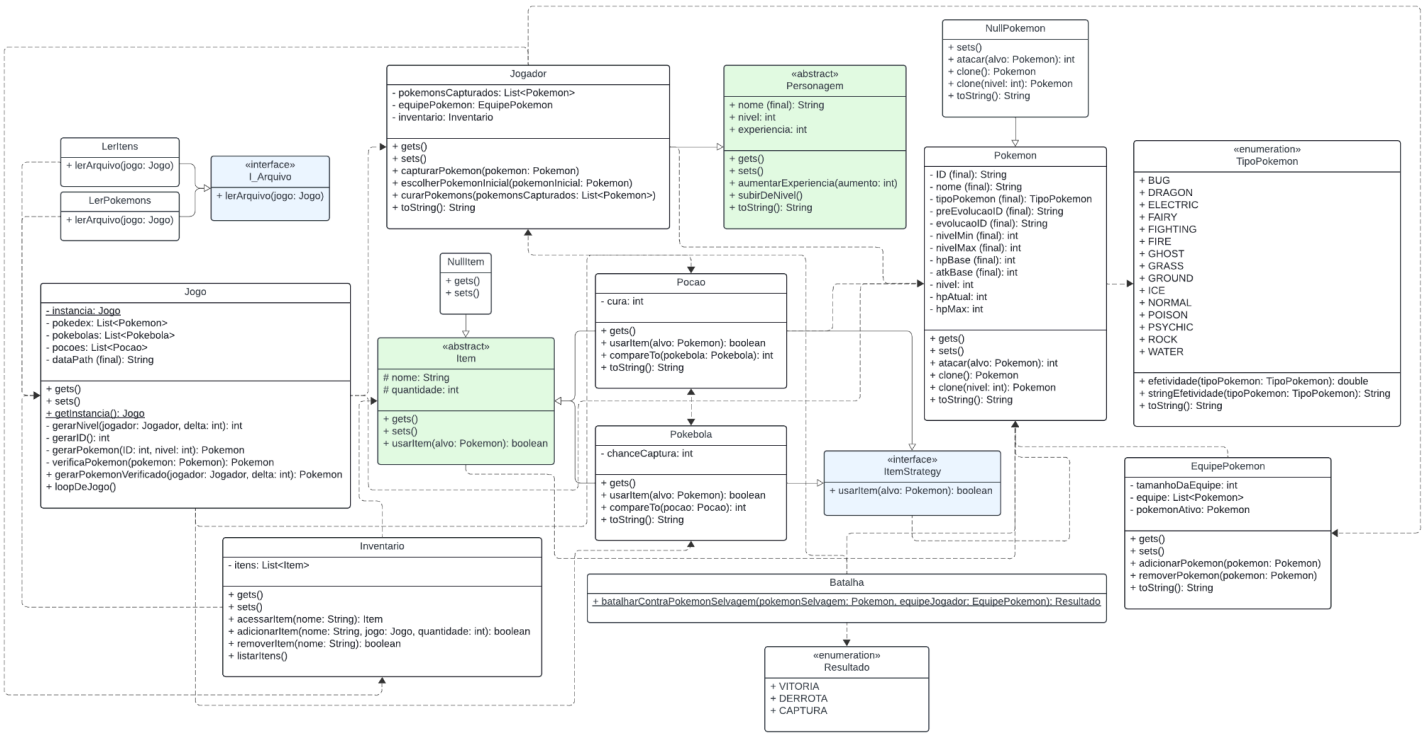

Figura 3: Diagrama UML do Projeto PokeJava.

A partir do diagrama, foram definidas as classes e seus relacionamentos:

- **I_Arquivo:** interface com um método abstrato para a leitura dos arquivos do jogo;
- **LerPokemons** e **LerItens:** classes que implementam a interface I_Arquivo, concretizando o método para a leitura do XML com os dados dos pokémons e dos itens, respectivamente;
- **Jogo:** classe singleton que inicializa o jogador, os pokémons e os itens do jogo, dando início ao loop do jogo. Também gera pokémons selvagens aleatórios baseado no nível médio dos pokémons do **Jogador**;
- **Item:** classe abstrata que possui como atributo o nome do item e um método para usá-lo;
- **NullItem:** classe herdeira de **Item**, utilizada para o uso do design pattern Null Object;
- **ItemStrategy:** interface utilizada para o uso do design pattern Strategy;

- **Pocao** e **Pokebola:** classes herdeiras de **Item**, que implementam o método para usar cada um dos itens à partir da interface **ItemStrategy**, curando ou tentando capturar um pokémon, respectivamente;
- **Inventario:** classe que representa o conjunto de itens que o **Jogador** possui. É responsável pelo gerenciamento desses itens, com métodos para adicionar, remover, acessar ou listar;
- **Personagem:** classe abstrata com atributos comuns a um personagem, como nome, nível e pontos de experiência. Também possui métodos para aumentar os pontos de experiência e subir de nível;
- **Jogador:** classe herdeira de **Personagem**, com atributos e métodos próprios para o jogador, incluindo a sua lista de pokémons capturados, a sua equipe pokémon e seu inventário, além de funções para escolher o pokémon inicial, capturar pokémons e curar todos os pokémons capturados (que é ativada quando o jogador sobe de nível);
- **Pokemon:** classe que contém atributos referentes aos pokémons, como seu nome, nível, hp e evoluções, além de um método próprio para seu ataque, que será utilizado durante as batalhas;
- **NullPokemon:** classe herdeira de **Pokemon**, utilizada para o uso do design pattern Null Object;
- **EquipePokemon:** classe que representa o conjunto de pokémons selecionados que o **Jogador** utilizará durante as batalhas. É responsável pelo gerenciamento desse pokémons, com métodos para adicionar e removê-los da lista;
- **TipoPokemon:** classe enum com todas as possibilidades de tipo de pokémons, além de um par de métodos para calcular e imprimir a efetividade de um tipo contra outro;
- **Batalha:** classe com um único método estático, que leva como argumentos um pokémon selvagem e o jogador e retorna o **Resultado** da batalha;
- **Resultado:** classe enum com três possibilidades de resultado. Ao fim da batalha, o jogador pode capturar o pokémon selvagem e ganhar pontos de experiência (CAPTURA). Por outro lado, é possível que o jogador derrote o pokémon selvagem, resultando na obtenção de pontos de experiência e itens aleatórios (VITORIA). Além disso, há o caso em que a equipe de pokémons do jogador perde a batalha (DERROTA).

4. Design Patterns

Foram utilizados três design patterns no desenvolvimento do projeto, o Singleton, o Strategy e o Null Object.

O Singleton foi aplicado à classe **Jogo** para garantir que apenas uma instância dela seja criada durante toda a execução do programa. Isso faz sentido porque essa classe é responsável por inicializar elementos fundamentais, como o jogador, os pokémons e os itens. Ter múltiplas instâncias da classe **Jogo** poderia levar a inconsistências e dificuldades de controle, já que esses elementos precisam ser únicos e compartilhados por todo o sistema. Portanto, ao usar o Singleton, garantimos que apenas uma instância da classe seja criada, mantendo a consistência e facilitando o gerenciamento dos recursos do jogo.

O padrão Strategy, por outro lado, foi empregado no projeto nas subclasses de **Item**, para permitir que o comportamento de utilização dos itens, como poções e pokébolas, seja definido por classes separadas, tornando o sistema mais flexível e de fácil extensão. Com essa abordagem, novos tipos de itens com comportamentos distintos podem ser facilmente adicionados sem a necessidade de modificar as classes existentes, garantindo um código mais coeso, modular e de fácil manutenção.

Por fim, o padrão Null Object foi utilizado nas classes **NullPokemon** e **NullItem** para evitar a necessidade de verificações explícitas de nulidade ao longo do código. Ao invés de retornar null quando um **Pokemon** ou **Item** não é encontrado, uma instância de uma classe Null específica é retornada. Isso simplifica o fluxo do programa, pois elimina a necessidade de múltiplas verificações de null, reduzindo a chance de NullPointerExceptions e tornando o código mais robusto e fácil de ler. Além disso, ao fornecer comportamentos padrão para essas instâncias Null, garantimos que o sistema continue a operar de maneira previsível, mesmo quando certos elementos estão ausentes.

5. Tratamento de Exceções

Quanto ao tratamento de exceções, foram escolhidas pelo grupo as exceções **InvalidPokemonLevelException** e **InvalidItemException**, considerando a necessidade de lidar com situações em que o nível de um Pokémon ou o nome de um item são inválidos, respectivamente. Essas exceções permitem que o código identifique e trate esses casos de forma adequada, fornecendo uma mensagem de erro específica para lidar com a situação.

O benefício desse tratamento de exceções no projeto é a melhoria da robustez e da legibilidade do código. Ao lançar exceções específicas para casos inválidos de nível de Pokémon ou de nome de item, o código se torna mais fácil de entender e manter, pois a lógica de validação e tratamento de erros está encapsulada nas próprias exceções. Isso também facilita a identificação e a correção de problemas, uma vez que a exceção lançada fornece informações precisas sobre o erro ocorrido, contribuindo para a confiabilidade e a qualidade do sistema.

6. Software Final

O software final desenvolvido pelo grupo para o Projeto PokeJava está disponível no link do GitHub.[1] A classe Main do projeto contém um loop do jogo, onde o usuário é guiado pelas funcionalidades disponíveis nesta versão simplificada de Pokémon.

Os passos iniciais solicitam ao jogador que insira seu nome e escolha seu pokémon inicial, assim como no jogo original. Em seguida, inicia-se um loop do Menu Inicial. Neste menu, o jogador pode escolher uma das quatro ações: batalhar contra um pokémon selvagem, editar sua equipe pokémon, ir ao Centro Pokémon para curar seus pokémons ou sair do jogo.

Ao escolher batalhar, o jogador pode atacar o inimigo com seu pokémon ativo, usar um item (pokébola ou poção) ou trocar o pokémon ativo por outro de sua equipe. Em cada batalha, o jogador enfrenta um pokémon selvagem aleatório, com dificuldade progressiva e recompensas para cada vitória ou captura. O nivelamento do pokémon selvagem é feito a partir da média de nível dos pokémons do jogador, e a efetividade de cada tipo de ataque também é calculada, tornando a experiência o mais funcional possível.

O jogador também pode, a qualquer momento, conferir sua lista de pokémons capturados e editar sua equipe pokémon como preferir, experimentando diferentes combinações e tipos de pokémon.

[1] Infelizmente, não foi possível implementar a interface gráfica devido a dificuldades na configuração do JavaFX. Apenas um dos membros conseguiu rodar o programa em seu computador e criar a primeira tela, enquanto os demais não conseguiram. Por isso, decidimos concentrar nossos esforços em garantir o funcionamento do jogo pelo terminal, como nos últimos laboratórios.