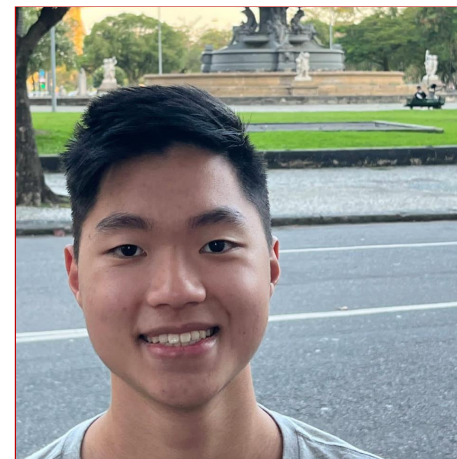
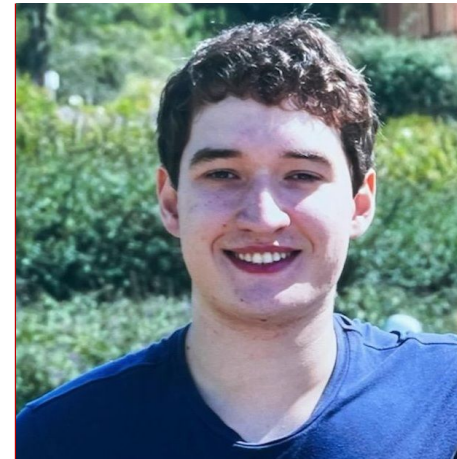


POKECLICKER

Carolina Wajner, Gabriel Lima, Luísa Bolzan,
Nicolas Lee e Rodrigo Feldens
Facilitador: Carolina Wajner

Técnicas de Construção de Programas – Turma A
Prof. Karina Kohl
2025/1

GRUPO 05



DESCRIÇÃO DO PROBLEMA

Contexto: Qual problema o projeto resolve?

- Muitos jogadores desejam gerenciar Pokémons de forma simples e casual, sem a complexidade dos jogos tradicionais.
- Experiência incremental e interativa.

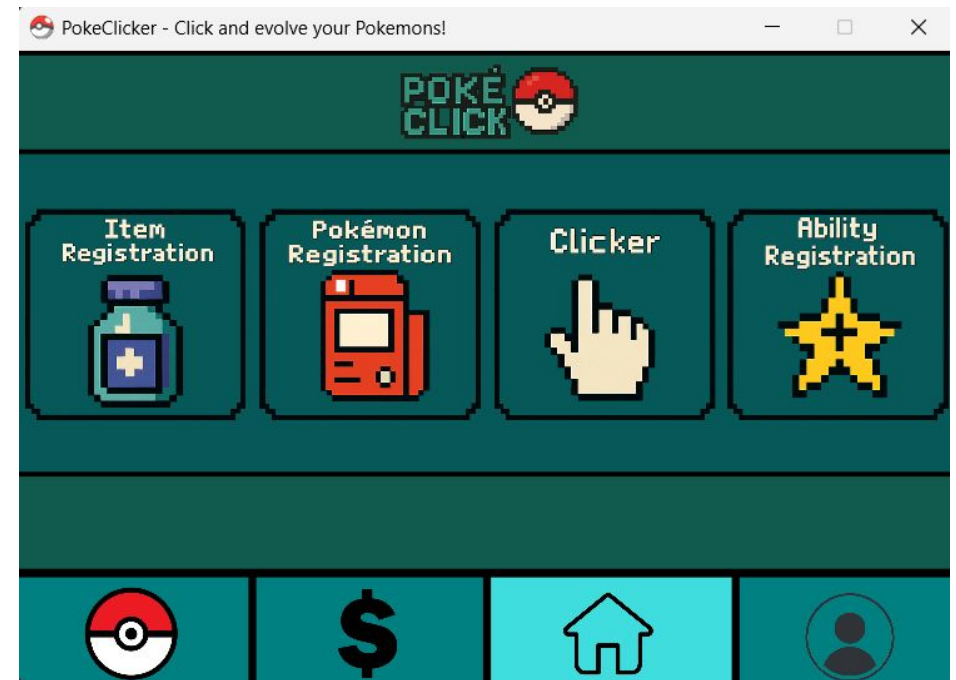
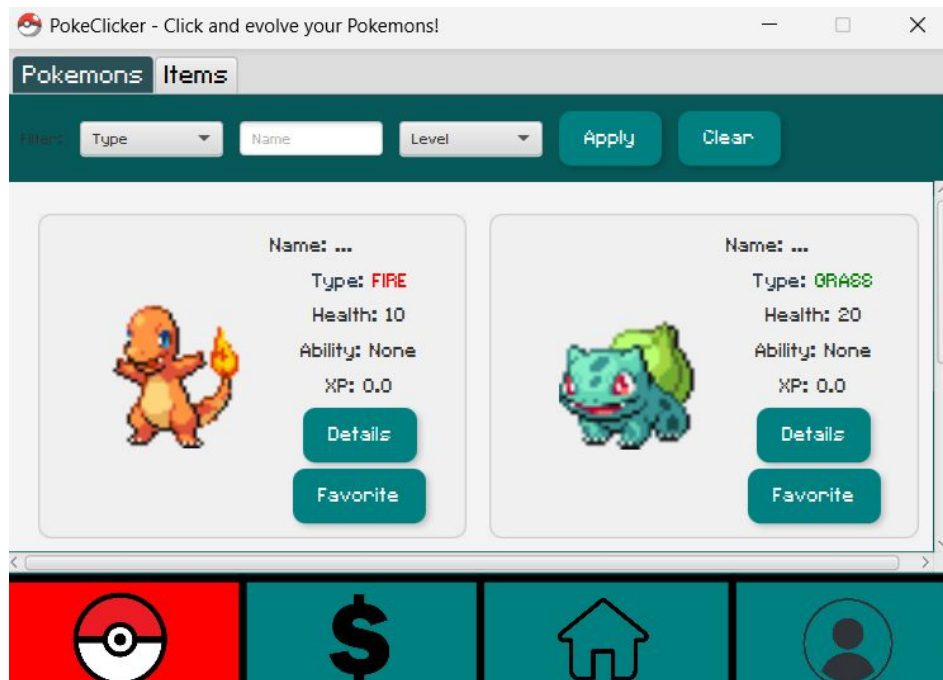
Motivação: Por que esse tema foi escolhido?

- O gênero “idle/clicker” cresce em popularidade, mas há poucas opções que tragam a temática Pokémon.
- Boa oportunidade para praticar programação (JavaFX, lógica de jogo e arquitetura de software).

DESCRIÇÃO DO PROBLEMA

Objetivo: O que o trabalho pretende alcançar?

- Criar uma interface gráfica simples e funcional que simule registros (pokémons, itens e habilidades), evoluções, loja e gerenciador (PC).



REQUISITOS FUNCIONAIS E NÃO FUNCIONAIS

Funcionais

- Cadastrar pokémons, itens e habilidades.
- Comprar pokémons e itens na loja.
- Visualizar e filtrar seus pokémons no PC.
- Detalhar seus pokémons no PC.
- Usar seus itens no PC.
- Apertar no botão do clicker e incrementar seu dinheiro.
- Comprar XP para um Pokémon no PC e evoluí-lo.

Não Funcionais

- Interface deve ser simples e intuitiva.
- Resposta a cliques e atualizações da interface em tempo real.
- Código deve ser modular e bem organizado.
- O jogo não pode corromper os dados salvos quando fechado (login e logout).
- Roda em sistema operacional que tenha suporte a Java e JavaFX.

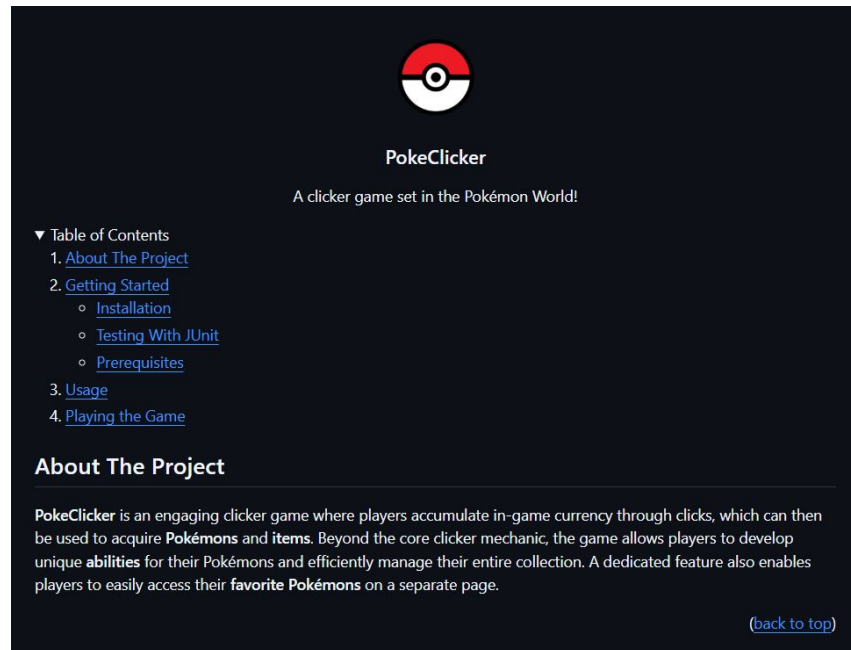
DIAGRAMA DE CLASSES

Por conta do tamanho, é preciso abrir o link para facilitar a visualização:

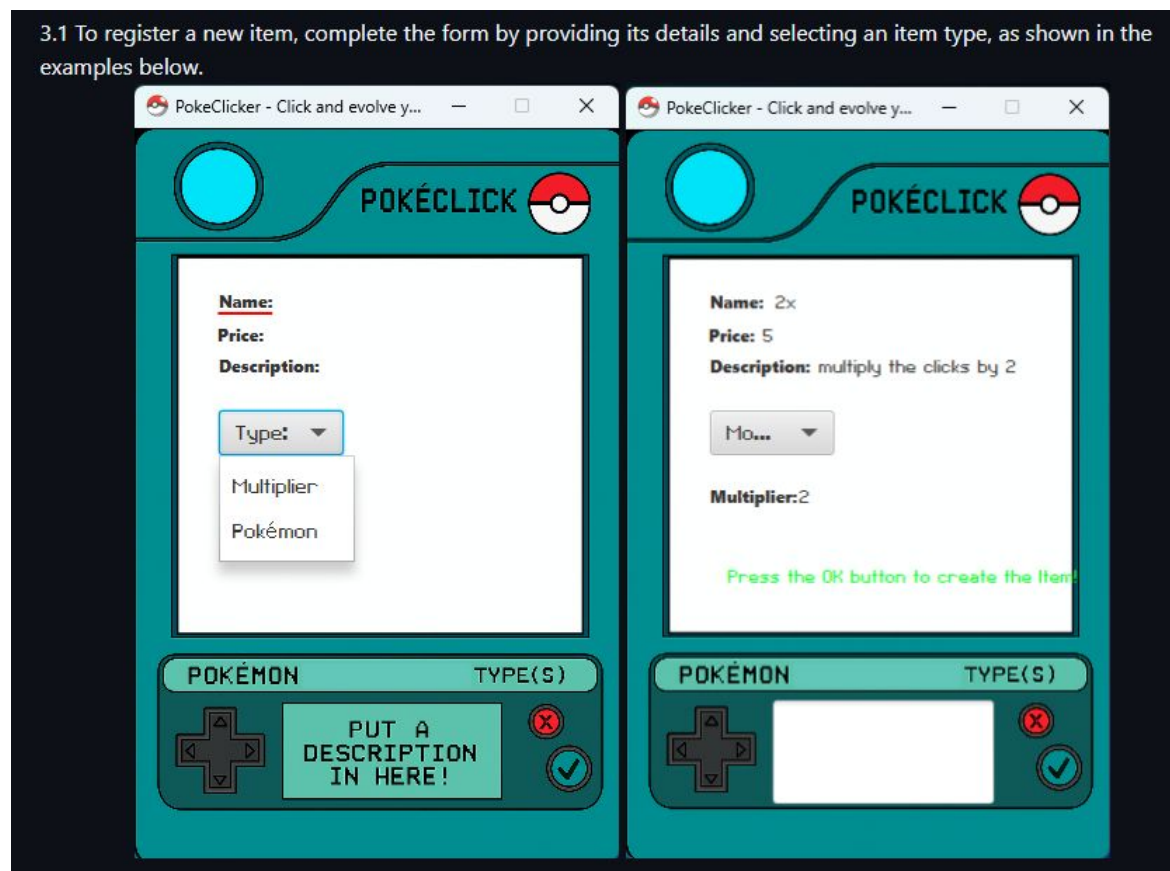
https://lucid.app/lucidchart/9da4de9f-e237-4ddf-bf41-82c79efb18f7/edit?page=0_0#

VISÃO DO REPOSITÓRIO

- **README.md:**




Exemplo de instrução para o usuário:




VISÃO DO REPOSITÓRIO

Exemplos de commits:


polished profile scene

 gslima33 committed last week


logica clicker/item

 carolwajner committed last week


added buy-item into ShopScene

 rsfeldens committed 4 days ago

Front: Substitute cyndaquil img to png

 nicolasclee committed 2 days ago

final tests

 luisabolzan committed 3 hours ago

Código evoluiu seguindo o princípio da **Single Responsibility**:

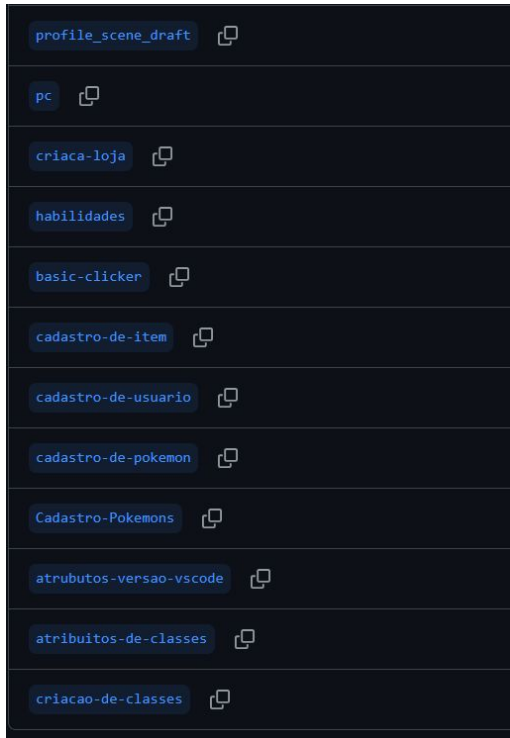
- Cada commit foi pensado para corrigir uma funcionalidade mais específica, facilitando para os outros desenvolvedores.

QR code com o link do repositório:

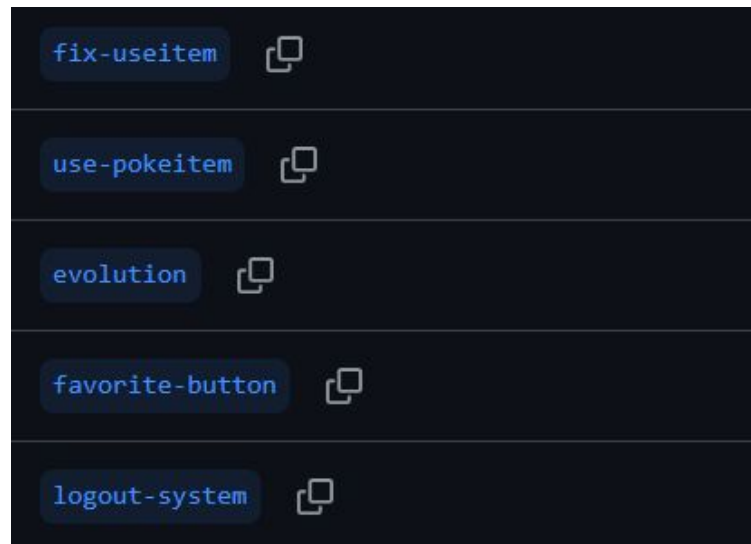


VISÃO DO REPOSITÓRIO

Exemplos de branches iniciais que ajudaram a estruturar o projeto:



Correções, funcionalidades aplicadas na interface gráfica etc.



Muitas branches, cada uma tratando de uma parte específica do código.

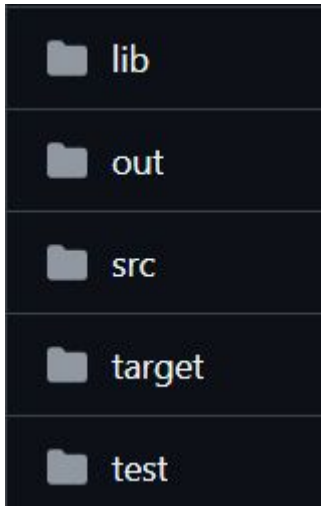


QR code com o link do repositório:

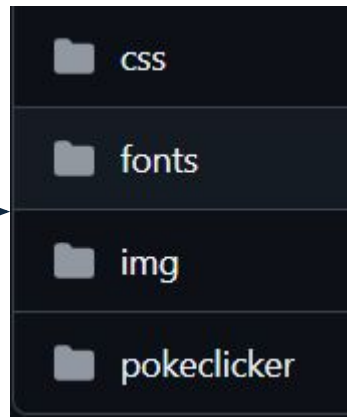


VISÃO GERAL DO CÓDIGO

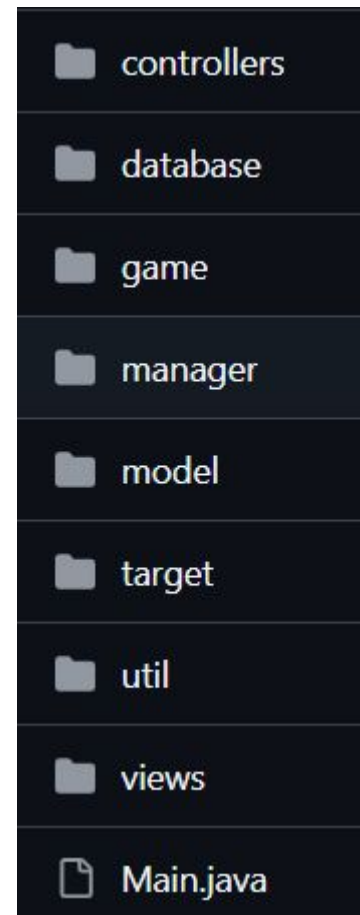
Pastas base



Implementação



classes modularizadas



Telas da interface gráfica

Base de dados (SQLite)

Shop, PC, Clicker

Gerencia a criação das classes na database

Classes base para tratamento modularizado

Trocadores de cena da interface gráfica

arquivos fxml da interface gráfica

VISÃO GERAL DO CÓDIGO

Database para armazenar:

- User
- Pokemon
- Item
- Ability



```
J AbilityDB.java
J ItemDB.java
J PokemonAbilityDB.java
J PokemonDB.java
J SQLiteConnection.java
J UserDB.java
```

Pokemon e seus tipos
(Fogo, Água e Grama)

```
J FirePokemon.java
J GrassPokemon.java
J LevelType.java
J Pokemon.java
J WaterPokemon.java
```

Aqui que ficam os **Getters**
e **Setters** cruciais para os
managers alterarem a base
de dados.

Base para as principais
funcionalidades que o
usuário pode acessar.

```
J Clicker.java
J PC.java
J Shop.java
```

VISÃO GERAL DO CÓDIGO

A aplicação possui uma interface gráfica que replica as funcionalidades do código de uma maneira intuitiva e visível.



```
AbilityController.java
ClickerController.java
HomeController.java
ImageSelectController.java
InitialController.java
ItemRegistrationController.java
PcController.java
ProfileController.java
RegistrationController.java
ShopController.java
```

- Seleção de telas que fornece autonomia para o usuário.
- Controle e tratamento de erros único para cada cena.

VISÃO GERAL DO CÓDIGO

Ex: manager (PokemonManager.java)

```
public static Pokemon createPokemon(String name, int totalHealth, List<Ability> habilidades, double
    pokeType type, String imagePath, String userName)
    throws IllegalArgumentException {
    if (PokemonDB.getPokemon(name) != null) {
        throw new IllegalArgumentException(s:"The Pokemon name already exists!");
    }

    Pokemon newPokemon;

    switch (type) {
        case FIRE ->
            newPokemon = new FirePokemon(name, LevelType.BEGINNER, xp:0.0, totalHealth, totalHea
                imagePath);
        case WATER ->
            newPokemon = new WaterPokemon(name, LevelType.BEGINNER, xp:0.0, totalHealth, totalHea
                imagePath);
        case GRASS ->
            newPokemon = new GrassPokemon(name, LevelType.BEGINNER, xp:0.0, totalHealth, totalHea
                imagePath);
        default -> throw new IllegalArgumentException(s:"Invalid Pokemon type!");
    }

    PokemonDB.insertPokemon(newPokemon, userName);
    return newPokemon;
}
```

Ex: database (ItemDB.java)

```
public static void updateItem(Item item) {
    String sql = "UPDATE item SET price = ?, available = ?, description = ?, multiplierOrDamage = ?";
    try (java.sql.Connection conn = SQLiteConnection.connect();
        java.sql.PreparedStatement pstmt = conn.prepareStatement(sql)) {
        pstmt.setDouble(parameterIndex:1, item.getPrice());
        pstmt.setBoolean(parameterIndex:2, item.isAvailable());
        pstmt.setString(parameterIndex:3, item.getDescription());
        pstmt.setDouble(parameterIndex:4, item.getMultiplierOrDamage());
        pstmt.setString(parameterIndex:5, item.getName());
        pstmt.executeUpdate();
        System.out.println(x:"Item updated successfully.");
    } catch (java.sql.SQLException e) {
        System.out.println("Error updating item: " + e.getMessage());
    }
}
```

Ex: model (User.java)

```
public class User
    implements Activatable {
    private String name;
    private double money = 0.0;
    private double moneyMultiplier = 1.0;
    private Pokemon favoritePokemon;

    public User(String name, double moneyMultiplier, double money, Pokemon favoritePokemon) {
        this.name = name;
        this.money = money;
        this.moneyMultiplier = moneyMultiplier;
        this.favoritePokemon = favoritePokemon;
    }
}
```

VISÃO GERAL DO CÓDIGO

Ex: game (Shop.java)

```
public class Shop {
    private final User user;
    private List<Purchasable> purchasables = new ArrayList<>();

    public Shop(User user, List<Purchasable> purchasables) {
        this.user = user;
        this.purchasables = purchasables;
    }

    public User getUser() {
        return user;
    }

    public List<Purchasable> getPurchasables() {
        return purchasables;
    }

    public Purchasable buyPokemonOrItem(Purchasable purchasable) {
        if (!purchasables.contains(purchasable)) {
            throw new IllegalArgumentException(s:"Item not found in the shop.");
        }
        if (user.getMoney() >= purchasable.getPrice()) {
            purchasable.setAvailable(available:false);
            return purchasable;
        } else {
            throw new IllegalArgumentException(s:"Not enough money to buy this item.");
        }
    }
}
```

Ex: views (arquivos da interface gráfica)

- abilityregistration.fxml
- clickerScene.fxml
- homeScene.fxml
- imageselection.fxml
- initialScene.fxml
- itemregistration.fxml
- pcScene.fxml
- pokemonregistration.fxml
- profileScene.fxml
- shopScene.fxml

Ex: controllers (ClickerController.java)

```
@FXML
public void clickButton(ActionEvent event) {
    Image gholdengoMoney = new Image(getClass().getResource(name:"/img/gholdengoMoney.gif").toExt

    gholdengoViewer.setImage(gholdengoMoney);
    gholdengoViewer.setFitWidth(240);
    gholdengoViewer.setFitHeight(190);
    clicker.registerClick();

    moneyLabel.setText("Money: " + user.getMoney());
    clickLabel.setText("Clicks: " + clicker.getTotalClicks());
}

@FXML
public void backButton(ActionEvent event) {

    UserManager.updateUser(user);
    SceneSwitcher.switchToHome(event, SceneSwitcher.getCurrentUsername());
}
```


TESTES - JUnit

Principais casos testados:

1. Criação, atualização e remoção
2. Busca e listagem
3. Validação de regras de negócio

```
Test run finished after 163610 ms
[      34 containers found      ]
[      0 containers skipped     ]
[      34 containers started    ]
[      0 containers aborted     ]
[      34 containers successful ]
[      0 containers failed      ]
[     185 tests found           ]
[      0 tests skipped          ]
[     185 tests started         ]
[      0 tests aborted          ]
[     185 tests successful      ]
[      0 tests failed           ]
```

```
@Test
void testAddAbilityToPokemonThrowsIfTypeMismatch() {
    Ability fireAbility = new Ability("TestEmber", "A test fire ability", PokeType.FIRE, 10, 0);
    AbilityDB.insertAbility(fireAbility);
    PokemonManager.createPokemon("Charmander", 39, List.of(), 100.0, PokeType.WATER, "img.png", "Ash");
    Exception ex = assertThrows(IllegalArgumentException.class, () ->
        PokemonManager.addAbilityToPokemon("Charmander", "TestEmber"));
    assertTrue(ex.getMessage().equalsIgnoreCase("Ability type does not match Pokemon type"));
}
```

```
@Test
void testCreatePokemonSuccess() {
    Pokemon pokemon = PokemonManager.createPokemon(
        "Charmander", 39, List.of(), 100.0, PokeType.FIRE, "img.png", "Ash");
    assertNotNull(pokemon);
    assertEquals("Charmander", pokemon.getName());
    assertEquals("FIRE", pokemon.getType());
}
```


TESTES - JUnit

Cobertura Funcional:

Os testes cobrem os principais fluxos de uso e erros esperados das entidades centrais.

Cobertura de Exceções:

Métodos críticos têm testes para exceções e mensagens de erro.

```
.
+-- JUnit Platform Suite [OK]
+-- JUnit Jupiter [OK]
| +-- AbilityTest [OK]
| | +-- testUpdateDamageNegativeThrows() [OK]
| | +-- testUpdateDamage() [OK]
| | '-- testAbilityConstructorAndGetters() [OK]
+-- PCTest [OK]
| +-- testRemoveItemThrowsIfNotEnough() [OK]
| +-- testAddItem() [OK]
| +-- testRemoveItemThrowsIfNotPresent() [OK]
| +-- testRemoveItemReducesCountAndRemovesWhenZero() [OK]
| +-- testAddPokemon() [OK]
| +-- testSetAndGetFavoritePokemon() [OK]
| +-- testGetItemCountForAbsentItem() [OK]
| '-- testConstructorInitializesCorrectly() [OK]
+-- ClickerTest [OK]
| +-- testSetMoneyPerClick() [OK]
| +-- testResetClicks() [OK]
| +-- testRegisterSingleClick() [OK]
| +-- testConstructorInitialization() [OK]
| +-- testRegisterMultipleClicks() [OK]
| '-- testRegisterClickWithMultiplier() [X] User's money should increase by 2.5. ==> expected: <2.5> but was: <6.25>
-- ShopTest [OK]
| +-- testBuyItemSuccess() [OK]
| +-- testBuyItemThrowsExceptionNotEnoughMoney() [OK]
| +-- testBuyItemSuccessDeductsMoney() [OK]
| +-- testBuyItemThrowsExceptionItemNotFound() [OK]
| '-- testConstructorInitializesCorrectly() [OK]
-- JUnit Vintage [OK]
```

DEMONSTRAÇÃO

Backup demonstração em vídeo:



CONCLUSÃO

Principais dificuldades:

Como armazenar os objetos e seus atributos de forma a facilitar a manipulação de dados?

- Depois de tentar .txt e .csv, o SQLite conseguiu entregar tudo aquilo que precisamos.

Comunicação entre back-end e front-end.

- Seguir os princípios da OOP, principalmente ao modularizar e encapsular o código, facilitou a implementação das funcionalidades cruciais para a aplicação.

→ O projeto reforçou a aplicação dos princípios da programação orientada a objetos e a importância da organização e modularidade do código.

→ Ademais, foi possível aprender sobre JavaFX e manipular base de dados de maneira bem estruturada.

CONCLUSÃO

Próximos passos: O que pode ser melhorado?

- É possível refatorar o código novamente e deixá-lo mais limpo, como por exemplo modularizar mais o controle de cenas da interface gráfica, distribuindo mais as responsabilidades de cada classe.
- A aplicação de uma maneira geral é bem sólida e possui alta capacidade de extensão, então novas features podem ser adicionadas no futuro para oferecer mais interação para o usuário (ex. batalhas pokemon).

CONCLUSÃO

Muito obrigado pela atenção! Perguntas?

