Trabalho Prático: Implementação e Teste de um (Pseudo)Jogo de Batalha por Turnos

Disciplina: Teste de Software

Professor: Eiji Adachi

Descrição do Entregável

Os alunos deverão desenvolver e entregar um conjunto completo de artefatos que demonstram a implementação e o teste do jogo de batalha por turnos conforme as especificações fornecidas anteriormente.

O entregável deve incluir, para cada regra de negócio identificada no enunciado do trabalho, os seguintes componentes:

1. Documentação das Regras de Negócio

Para cada regra de negócio, os alunos devem fornecer:

1.1. Descrição das Partições Identificadas

- **Objetivo:** Identificar e descrever as regras que influenciam a regra de negócio e que são usadas para particionar o domínio de entrada. Devem ser especificadas as partições que levam a situações excepcionais, quando aplicável.

1.2. Análise dos Valores Limites Identificados

- **Objetivo:** Identificar os valores de fronteira que se aplicam a cada partição, determinando os valores que devem ser testados.

1.3. Tabela de Decisão

- **Objetivo:** Representar de forma tabular as condições e ações que especificam cada regra de negócio, facilitando a compreensão e a criação de casos de teste.

2. Projeto e Implementação dos Casos de Teste

2.1. Casos de Teste

- **Objetivo:** Criar casos de teste detalhados que cobrem todas as partições, valores limites e regras da tabela de decisão.
- **Componentes:**
    - **Identificador do Caso de Teste:** Nome único ou número.
    - **Descrição:** Breve descrição do que o caso de teste verifica.
    - **Entradas:** Valores de entrada utilizados no teste.

    - **Saídas Esperadas:** Resultados esperados após a execução do teste.
    - **Pré-condições:** Estado necessário antes da execução do teste.
    - **Pós-condições:** Estado esperado após a execução do teste.

2.2. Implementação dos Testes Automatizados

- **Ferramenta:** Utilizar **JUnit 5** para implementar os testes.
- **Requisitos:**
    - Cada caso de teste deve ser implementado como um método de teste separado. Alternativamente, podem ser usados testes parametrizados.
    - Utilizar asserções apropriadas para verificar os resultados esperados.
    - Nomear os métodos de teste de forma clara e descritiva.
    - Incluir comentários explicativos quando necessário.
    - Testar casos excepcionais.
    - **Relatório de Cobertura de Testes:**
        - **Objetivo:** Demonstrar que os testes alcançaram **100% de cobertura de arestas** no código.
        - **Requisitos:**
            - Utilizar uma ferramenta de análise de cobertura, como **JaCoCo** ou **Cobertura**, para gerar relatórios de cobertura.
            - Incluir no `README.md` do projeto instruções sobre como rodar os testes com a análise de cobertura e como gerar o relatório de cobertura.
            - O relatório deve evidenciar que todas as arestas do código foram cobertas pelos testes.

2.3. Aplicação do Critério MC/DC

- **Objetivo:** Aplicar o critério **MC/DC (Modified Condition/Decision Coverage)** ao **if** mais complexo do projeto.
- **Requisitos:**
    - Identificar o **if** mais complexo no código do projeto.
    - Elaborar casos de teste que atendam ao critério MC/DC para este **if** específico.
    - **Documentação:**
        - Incluir no relatório de testes a identificação do **if** escolhido.
        - Descrever as condições e decisões envolvidas.
        - Apresentar uma tabela mostrando os casos de teste, as combinações de condições e quais resultados são esperados para atingir a cobertura MC/DC.
    - **Implementação:**
        - Implementar os casos de teste no código, garantindo que o critério MC/DC seja satisfeito.
        - Utilizar comentários ou documentação adicional nos testes para indicar que eles fazem parte da cobertura MC/DC.

2.4. Rastreabilidade

- **Objetivo:** Garantir que cada partição, valor limite, regra da tabela de decisão e critérios de cobertura estejam cobertos por pelo menos um caso de teste.

- **Estratégia Sugerida:**
    - Criar uma matriz de rastreabilidade que mapeia cada regra de negócio e critérios de cobertura para os casos de teste correspondentes.
    - Incluir referências cruzadas entre a tabela de decisão, critérios de cobertura (incluindo MC/DC) e os casos de teste.
    - Utilizar identificadores únicos para facilitar a rastreabilidade.

Partição / Critério	**Casos de Teste Relacionados**
Resistência ≥ Ataque (Guerreiro)	CT01, CT02
Ataque ≥ Velocidade (Assassino)	CT03, CT04
MC/DC no **if** da função X	CT15, CT16, CT17, CT18
...	...

3. Implementação das Regras de Negócio

- **Objetivo:** Implementar o código que realiza as regras de negócio conforme especificadas.
- **Requisitos:**
    - **Clareza e Manutenção:** Código bem estruturado, com uso adequado de classes, métodos e padrões de design quando aplicável.
    - **Documentação:** Comentários explicativos nas partes complexas do código.
    - **Conformidade:** Garantir que todas as restrições e regras detalhadas no enunciado sejam respeitadas.

4. Programa Demonstrativo (*main*)

Os alunos devem implementar um método *main* que demonstre o funcionamento completo e correto do jogo. Este programa deve incluir:

4.1. Criação dos Personagens

- **Funcionalidades:**
    - Permitir ao usuário escolher a classe de cada personagem (Guerreiro ou Assassino).
    - Distribuir os 20 pontos entre os quatro atributos, respeitando as restrições da classe escolhida.
    - Validar a distribuição e exibir os atributos finais dos personagens após a validação, incluindo os pontos de vida.
- **Interface:** Via console, com prompts claros para o usuário.

4.2. Início da Batalha

- **Funcionalidades:**
    - Determinar e exibir qual personagem ataca primeiro.
    - Caso haja empate, decidir aleatoriamente e informar quem ataca primeiro.

4.3. Execução dos Turnos

- **Para cada turno:**
  - Exibir qual personagem está atacando.
  - Calcular e exibir o **Dano Base**, indicando se foi um golpe crítico.
  - Exibir o **Dano Infringido** e atualizar o **HP** do defensor.
  - Mostrar o **HP** atual de ambos os personagens.
  - Indicar se um ataque foi evitado devido à **Evasão**.

4.4. Finalização da Batalha

- **Funcionalidades:**
  - Detectar quando o **HP** de um personagem é esgotado.
  - Anunciar o vencedor.
  - Permitir reiniciar a batalha ou encerrar o programa.

5. Documentação do Projeto

5.1. README

- **Conteúdo:**
  - **Autores:** Nome completo dos autores, em ordem alfabética do primeiro nome.
  - **Introdução:** Breve descrição do projeto e seus objetivos.
  - **Instruções de Compilação e Execução:** Passos detalhados para compilar e executar o programa.
  - **Como Executar os Testes:** Instruções para rodar os testes automatizados.
  - **Como Gerar o Relatório de Cobertura:** Instruções para gerar o relatório de cobertura dos testes.
  - **Dependências:** Listar todas as dependências e como instalá-las.
  - **Uso do Programa:** Exemplos de uso e funcionalidades disponíveis.

5.2. Comentários no Código

- **Objetivo:** Explicar as principais funções, classes e decisões de design diretamente no código.
- **Requisitos:**
  - Comentários claros e concisos.
  - Evitar redundâncias; comentar apenas partes que necessitam de explicação adicional.

5.3. Relatório de Testes

- **Conteúdo:** Para cada funcionalidade:
  - **Partições**
  - **Valores Limites**
  - **Tabelas de Decisão**
  - **Casos de Teste**
  - **Rastreabilidade**
  - **Cobertura de Testes**, descrevendo brevemente qual cobertura foi alcançada inicialmente após aplicar os critérios de testes funcionais e se foram necessários criar mais testes para alcançar a cobertura exigida.
  - **Aplicação do Critério MC/DC:**

        - Descrição detalhada do **if** mais complexo escolhido.
        - Tabela demonstrando as condições, decisões e os casos de teste que satisfazem o critério MC/DC.
        - Discussão sobre como os casos de teste implementados atendem ao critério.
- **Formato:** O relatório deverá ser obrigatoriamente em formato Markdown (extensão `.md`) e deverá ser disponibilizado na raiz do projeto com o nome `Relatorio_Testes.md`. Podem ser usadas planilhas com extensão `.xlsx`, mas elas devem ser usadas de modo complementar ao relatório em formato Markdown, ou seja, mesmo com as planilhas, o relatório ainda é necessário.

6. Apresentação e Entrega

- **Apresentação:**
    - O grupo completo deverá apresentar o trabalho em horário previamente agendado. Nesta ocasião, o professor irá: inspecionar o trabalho feito, arguir os membros do grupo sobre as decisões tomadas e irá testar manualmente a *main* implementada.
- **Entregável:**
    - Deverá ser entregue o projeto base disponibilizado pelo professor com os devidos artefatos exigidos neste enunciado. O projeto deverá ser entregue em formato `.zip` via **SIGAA** até a data prevista no sistema. O projeto disponibilizado pelo professor é um projeto Maven e deverá ter o seu arquivo `pom.xml` modificado da seguinte forma:
        - `artifactId`: deve seguir o formato `batalha-nomesobrenome1-nomesobrenome2-nomesobrenome3`, sendo que os nomes devem estar em ordem alfabética pelo primeiro nome e basta o primeiro e último nome de cada membro;
        - `description`: deve constar o nome completo de todos os membros em ordem alfabética seguindo o padrão `Este trabalho foi feito por: nomesobrenome1, nomesobrenome2, nomesobrenome3.`
- **Importante:**
    - Quaisquer dependências extras devem estar devidamente configuradas no arquivo `pom.xml`.
    - **Não aceitarei o uso do Project Lombok.**

Checklist do Entregável

Para garantir que todos os componentes foram incluídos, os alunos podem utilizar o seguinte checklist:

1. **Documentação das Regras de Negócio:**
    - ☐ Descrição das partições identificadas para cada regra.
    - ☐ Análise dos valores limites para cada partição.
    - ☐ Tabela de decisão para cada regra.
2. **Casos de Teste:**
    - ☐ Projeto detalhado dos casos de teste.
    - ☐ Implementação dos testes automatizados em JUnit 5.
    - ☐ Aplicação do critério MC/DC ao **if** mais complexo.

- [ ] Relatório de Cobertura de Testes demonstrando 100% de cobertura de arestas.
- [ ] Matriz de rastreabilidade demonstrando cobertura completa.

3. **Implementação do Jogo:**
   - [ ] Código que implementa as regras de negócio.
   - [ ] Método *main* que demonstra o funcionamento completo do jogo.
4. **Documentação:**
   - [ ] README detalhado com todas as instruções necessárias.
   - [ ] Comentários adequados no código.
   - [ ] Relatório de Testes com inclusão da aplicação do critério MC/DC.
5. **Formato e Organização:**
   - [ ] Arquivos organizados de forma clara e lógica.
   - [ ] Nomenclatura consistente e apropriada.
   - [ ] Upload correto na plataforma de entrega.

Critérios de Avaliação

Para assegurar uma avaliação justa e objetiva, os seguintes critérios serão utilizados:

1. **Completude e Correção:**
   - Todos os requisitos do entregável foram atendidos.
   - As regras de negócio estão corretamente implementadas e testadas.
2. **Qualidade dos Testes:**
   - Cobertura completa das partições, valores limites e regras de negócio.
   - **100% de cobertura de arestas** demonstrada no relatório de cobertura.
   - Aplicação correta do critério **MC/DC** ao **if** mais complexo.
   - Testes automatizados bem estruturados e eficazes.
3. **Qualidade do Código:**
   - Código limpo, bem organizado e comentado.
   - Uso adequado de princípios de programação orientada a objetos.
4. **Documentação:**
   - Clareza e detalhamento das instruções no README.
   - Documentação completa das regras de negócio e casos de teste.
   - Inclusão do relatório de cobertura de testes e aplicação do critério MC/DC.
5. **Funcionamento do Programa Demonstrativo:**
   - O método *main* funciona conforme especificado.
   - A interação com o usuário é intuitiva e sem erros.

6. **Rastreabilidade:**
    - A matriz de rastreabilidade está presente e demonstra cobertura completa.
    - Facilita a verificação de que todos os critérios de cobertura foram atendidos.