Trabalho de INF1636

08/06/2020

Prof. Ivan Mathias Filho

Instruções para a 3ª iteração

O objetivo da 3ª iteração é implementar as funcionalidades do jogo que tratam da realização de jogadas até os ataques, incluindo a ocupação de territórios. Todas essas funcionalidades terão de ser acionadas por meio da interface gráfica com o usuário (componente View).

Implementação do componente Model

Na 3ª iteração apenas pequenos ajustes, caso sejam necessários, serão feitos no componente Model, uma vez que sua implementação foi concluída na 2ª iteração.

Implementação do componente View

São os seguintes os requisitos de interface gráfica com o usuário (GUI) que terão de ser implementados na 3ª iteração:

1. Definição da ordem dos jogadores – a definição da ordem para efeito de jogadas, será feita por meio de simulação de sorteio, realizada pelo próprio programa. Os jogadores serão apenas informados do resultado do sorteio.
2. Recebimento e distribuição de exércitos – no início da jogada, o participante deve receber exércitos a partir do número de territórios possuídos, por possuir um continente por inteiro e por troca de cartas.
3. Ataques – para simplificar a implementação do jogo os ataques serão feitos com o maior número possível de dados, assim como as defesas. Não caberá ao jogador escolher com quantos dados irá atacar ou se defender, pois o programa tomará essa decisão por ele.
4. Ocupação de território conquistado – para simplificar a implementação do jogo, o programa irá deslocar o máximo de exércitos possível como resultado da conquista de um território.
5. Escolha dos valores dos dados – tendo em vista a realização de testes das funcionalidades do jogo, nesta iteração a interface gráfica deverá prover um mecanismo para a escolha dos valores dos dados como resultado de um lançamento.

Observações

a) Os objetivos que vocês irão disponibilizar podem variar de acordo com as regras da versão do jogo que vocês escolherem, desde que eles incluam número de territórios a serem conquistados, continentes a serem conquistados e exércitos a serem eliminados.

b) O mecanismo para que um jogador escolha como distribuir os exércitos que recebeu no início de uma rodada (item 2 da lista anterior) é de livre escolha de cada grupo. Entretanto, mecanismos muito rudimentares ou que exijam excesso de digitação (por exemplo, baseado apenas em digitação de texto) perderão pontos. Sugiro que os territórios que um jogador possui sejam exibidos em uma list box de múltipla escolha. Após a escolha dos territórios que receberão exércitos ter sido encerrada, o programa percorreria a lista perguntando ao jogador, para cada território, quantos exércitos ele deseja posicionar. Este método é a penas uma sugestão.

Testes Unitários

Com a implementação de parte da interface gráfica com o usuário (GUI) será necessário criar casos de teste para funcionalidades completas, disparadas por meio de ações sobre a GUI.

Design Patterns e Estruturas de Dados

Na avaliação do trabalho será levada em consideração a aplicação correta das técnicas de design e programação vistas durante o curso. Isso inclui a observação dos critérios de acoplamento e coesão, a organização do aplicativo em pacotes e a utilização **obrigatória** dos seguintes Design Patterns:

- **Observer** – toda a comunicação e transferência de dados entra o Model e a View terá de ser feita por meio de classes e interfaces que implementem este padrão (**vide observação abaixo**).
- **Façade** – a organização do Model sugerida na 1ª iteração já segue os conceitos relativos ao padrão Façade. Isto é, classes não públicas para representar os elementos do jogo, juntamente com uma ou mais classes que compõem uma API por meio da qual serviços do Model são requisitados.

- **Singleton** – as classes que formam a API do Model são candidatas naturais a serem implementadas segundo este padrão.

Todas as coleções que forem usadas no programa devem ser implementadas por meio de classes pertencentes ao framework de coleções de Java.

Observação: a classe **Observable** e a interface **Observer** se tornaram obsoletas a partir do Java 9. Vocês devem implementar suas próprias classes para contornar essa limitação. Elas são muito simples e estão bem documentadas no exemplo que se encontra nas transparências do Capítulo 7.