Trabalho de INF1636

20/05/2020

Prof. Ivan Mathias Filho

Instruções para a 2ª iteração

O objetivo da 2ª iteração é completar os componentes Model com as regras remanescentes da 1ª iteração, além de iniciar a construção da interface gráfica com o usuário (componente View).

Implementação do componente Model

Na 2ª iteração o componente Model deve ser completado. Obviamente, alterações futuras podem ser feitas nesse componente em função de necessidades surgidas no decorrer da implementação dos demais componentes (View e Controller), que serão os elementos centrais das duas iterações seguintes. Entretanto, tais alterações deverão ser de pequena magnitude, devido às propriedades da arquitetura MVC.

Regras abordadas na 2ª iteração

A seguintes regras do War terão de ser implementadas na 2ª iteração:

1. Ataques
    1.1. Simulação, por meio de geração de números randômicos, do lançamento dos dados de ataque e de defesa;
    1.2. Comparação de ambos os conjuntos de dados;
    1.3. Conquista de territórios;
    1.4. Deslocamento de exércitos como resultado de conquistas;
    1.5. Obtenção de carta como resultado de conquistas;
    1.6. Deslocamento de exércitos ao término de uma jogada.
2. Eliminação de Concorrente
    2.1. Possível mudança do objetivo relacionado com o jogador eliminado (X), caso o responsável pela eliminação não seja o jogador cujo objetivo seria justamente eliminar X.
3. Cumprimento de objetivo e encerramento de uma partida.

Implementação do componente View

São os seguintes os requisitos de interface gráfica com o usuário (GUI) que terão de ser implementados na 2ª iteração:

1. Cadastro dos jogadores – janela inicial em que serão definidos o número de jogadores, o nome e a cor de cada um deles.
2. Exibição do tabuleiro do jogo, sorteio dos territórios entre os jogadores participantes e posicionamento dos exércitos nesses territórios.
3. Sorteio e exibição dos objetivos. A exibição de um objetivo deve ser solicitada pelo jogador da vez por meio de uma operação sobre um elemento da GUI. Cada grupo tem liberdade para implementar este requisito da maneira que achar mais adequada.
4. Exibição do número de exércitos estacionados nos territórios. Este requisito pode ser implementado de várias maneiras. Por exemplo:
    4.1. Uso de JLabels em cada um dos territórios – o número de exércitos e a cor do jogador que possui o território podem ser exibidos por meio de texto HTML inserido em um JLabel. O uso de HTML permite definir o tipo de fonte dos caracteres, o tamanho e a cor do texto exibido.
    4.2. Clique sobre um território – após o clique seria aberto um diálogo que informaria o número de exércitos estacionados e a cor do jogador que possui o território.
    4.3. Uso de listabox – uma janela separada conteria uma listbox em que seriam exibidos os nomes de todos os territórios, o número de exércitos estacionados em cada um deles e a cor do jogador que possui cada um dos territórios.

Testes Unitários

Testes unitários são realizados tendo em vista testar unidades individuais de código fonte. Tais unidades podem ser funções, métodos, classes, módulos e etc. Eles têm por objetivo verificar se cada unidade atende corretamente à sua especificação.

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.
- **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.