
Trabalho de INF1636

04/05/2020

Prof. Ivan Mathias Filho

# Instruções para a 1ª iteração

O software para jogar War será organizado segundo a arquitetura MVC. Na 1ª iteração será iniciada a especificação e codificação do componente Model dessa arquitetura. Esse componente será formado por uma ou mais classes que fornecerão uma API (Application Programming Interface) para que os demais componentes (View e Controller) solicitem, quando necessário, serviços ao componente Model.

# Implementação do componente Model

O componente Model deve ser um pacote Java. Todas as classes que representem elementos básicos do jogo War (por exemplo, Jogador, Território, Continente, Objetivo, Carta e etc) devem ser declaradas como não públicas, para que não possam ser diretamente referenciadas por classes dos pacotes de View e Controller.

Segundo o princípio da ocultação da informação, apenas as funções de um módulo que serão diretamente chamadas pelos demais módulos de uma aplicação – tendo em vista obter os serviços necessários à execução das tarefas desses módulos – deverão ser públicas. Isto é, o pacote Model terá de oferecer uma ou mais classes públicas que disponibilizarão uma API apara que as regras do jogo possam ser acessadas pelas classes externas a Model.

O componente Controller não começará a ser desenvolvido nesta 1ª iteração, mas é interessante, neste momento, apresentar algumas de suas características futuras.

Todo os procedimentos que concernem à realização de uma jogada serão controlados por classes do componente Controller. Isso pode ser exemplificado com um trecho do manual do War que lista as etapas que compõem uma jogada. Esse trecho diz o seguinte:

Cada jogador passa na sua vez, tanto na primeira como em todas as outras rodadas, pelas seguintes etapas, nesta ordem:

a) Receber novos exércitos e os colocar de acordo com a sua estratégia;

b) Se desejar, atacar os seus adversários;

c) Desloca seus exércitos se houver conveniência;

d) Receber uma carta se fizer jus a isto.

Isto é, será o componente Controller que controlará a execução das etapas referentes a uma jogada. Entretanto, para que as regras do jogo sejam cumpridas o Controller terá de solicitar serviços ao componente Model. Vejamos o seguinte exemplo:

O jogador de cor preta deseja atacar o território Califórnia a partir do território Colômbia/Venezuela. Para dar prosseguimento ao ataque, o Controller terá de se certificar de que

1. O território Colômbia/Venezuela pertence ao jogador de cor preta;
2. O Território Califórnia não pertence ao jogador de cor preta;
3. O território Colômbia/Venezuela faz fronteira com o território Califórnia;
4. O território Colômbia/Venezuela tem pelo menos dois exércitos.

Para validar as regras acima, a API do componente Model poderia ter um método chamado de **validaAtaque()**, que recebesse como parâmetros a cor do jogador da vez, o identificador do território de origem do ataque (nome ou número) e o identificador do território de destino do ataque (nome ou número). Esse método poderia retornar um valor booleano, indicando se o ataque poderia ou não ser realizado, ou mesmo um número inteiro que indicasse o número de dados que poderia ser usado no ataque. Caso o número de dados fosse zero ficaria determinado que ataque não poderia ser realizado.

O exemplo anterior fornece um caminho a ser seguido na implementação dos componentes Model e Controller. Ele deve ser visto como umas das possibilidades de organização dos componentes, e não como um modelo que tem de ser seguido.

## Regras abordadas na 1ª iteração

A seguintes regras do War terão de ser implementadas na 1ª iteração:

1. Inclusão de um jogador (nome e cor) na partida e definição, por meio de sorteio da sua ordem de jogada;
2. Sorteio dos objetivos que cada jogador terá de atingir;
3. Sorteio e distribuição das cartas entre os jogadores, com o consequente posicionamento de um exército em cada território recebido;
4. Recebimento e posicionamento dos exércitos correspondentes ao número de territórios (metade) que o jogador da vez possui;
5. Recebimento e posicionamento dos exércitos correspondentes à posse de um continente inteiro;
6. Recebimento e posicionamento dos exércitos correspondentes a troca de cartas.

Não será exigida a implementação de todas as regras do jogo nesta 1ª iteração, até porque quando os componentes View e Controller começarem a ser implementados surgirão, certamente, demandas por parte desses componentes que não foram inicialmente previstas. Sendo assim, o componente Model deverá permitir futuras alterações para atender a essas demandas.

Para tal, deve-se observar o princípio aberto/fechado, cuja formulação é atribuída a Bertrand Meyer, autor do livro **Object Oriented Software Construction**. Esse princípio diz o seguinte:

- Um módulo será dito aberto se ele ainda está disponível para extensão. Por exemplo, se for possível adicionar campos às estruturas de dados que ele contém, ou novos elementos ao conjunto de funções que executa.
- Um módulo será dito ser fechado se está disponível para uso por outros módulos. Isso pressupõe que o módulo tenha sido bem-definido. Isto é, que tenha uma descrição estável e que sua API abstraia suas características específicas.

Em termos de processo de desenvolvimento de software, é importante que o relatório da iteração diga o que foi implementado e testado, o que foi implementado, mas não foi totalmente testado, e o que não foi implementado no curso de uma iteração. Ele deve, também, relatar os problemas encontrados e os motivos pelos quais uma ou outra funcionalidade não foi implementada ou testada. Por último, esse relatório deve apontar o que será implementado na próxima iteração.

Um modelo de relatório de iteração está disponível na página da disciplina no EAD.

## 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.

Todas as funções que compõem um módulo têm de ser testadas individualmente. Para tal, crie um caso de teste para cada função a ser testada. Utilize o modelo de caso de teste que foi publicado no EAD.

Caso um método **m1** chame um método **m2**, deve-se, primeiro, testar **m2** para garantir que ela está funcionado segundo a sua especificação. Em seguida testa-se **m1**, pois mesmo que ela não esteja funcionando corretamente, não será necessário retestar **m2**, pois já se sabe que ele está funcionado de acordo com o especificado.

Quem tiver conhecimento da ferramenta JUnit pode usá-la sem maiores problemas, mas seu uso não é obrigatório.