# Dou Shou Qi - Padrões de Projeto

**Ivens Diego Müller**[1]

[1]CEAVI - Universidade do Estado de Santa Catarina(UDESC) Ibirama - SC - Brasil

ivens.muller@edu.udesc.br

## 1. Informações Gerais

Jogo Dou Shou Qi implementado utilizando a linguagem de programação Java para a disciplina de Padrões de Projeto para o curso de Bacharelado em Engenharia de Software no Centro de Educação Superior do Alto Vale do Itajaí na Universidade do Estado de Santa Catarina(UDESC). Foi utilizado o padrão de arquitetura MVC e os padrões de projeto Abstract Factory, Builder, Command, Observer, Singleton, Visitor, Strategy, State e Decorator.

## 2. Model-View-Controller(MVC)

O padrão de arquitetura MVC foi implementado no jogo. Implementou-se 3 pacotes.

### 2.1. Model

Contém todas as classes de modelo do sistema, como peças, animais, fábricas abstratas(Abstract Factory) e as classes do Builder.

### 2.2. View

Contemos apenas 3 classes, uma delas é a Tabuleiro, que herda de um JFrame e é a view onde contém a JTable que é utilizada para implementar o tabuleiro.

A outra, chamasse TabuleiroRenderer, que é a classe responsável renderização do JTable, desenhando as imagens nas suas posições.

A terceira, chamada de TabuleiroTableModel, é a classe responsável por buscar os dados de cada posição do controller e implementa o método do Click no mouse para realizar a movimentação das peças (chamando o controller para validade).

### 2.3. Controller

No controller, foi implementado a classe TabuleiroController, que tem todas as regras de criação dos objetos, movimentação das peças e implementação do tabuleiro em si. Também, há a interface do Command e todas as classes do padrão command.

## 3. Observer

O padrão observer foi implementado da seguinte maneira: Criado a interace ObservadorTabuleiro no pacote Model, onde o TabuleiroTableModel implementa essa interface. O controller tem uma lista de observadores do tabuleiro que, quando o TabuleiroTableModel cria um objeto do controller, ele se cadastra como observador. A criação do controller e o cadastramento como observador acontecem no construtor do TabuleiroTableModel.

Essa interface tem 5 métodos. Que são chamados pelo controller. Um é chamado para notificar o carregamento do tabuleiro, outro para enviar mensagens, notificar alteração no tabuleiro, notificar a troca da imagem de uma peça e notificar o fim do jogo.

## 4. Abstract Factory

No pacote Model, foram criados duas Fábricas Abstratas, uma que é chamada de FabricaDePeca, essa fábrica possui dois métodos abstratos, um utilizado para cria uma peça para o jogador 1 e a outra para o jogador 2. Ambos os métodos recebem como parâmetro uma linha e uma coluna e retornam um objeto do model Peca. As fábricas concretas são: FabricaDeArmadilha, FabricaDeCachorro, FabricaDeElefante, FabricaDeGato, FabricaDeLeao, FabricaDeLeopardo, FabricaDeLobo, FabricaDeRato, FabricaDeTigre e FabricaDeToca.

A outra chamasse FabricaDeObjetoTabuleiro, que tem um método abstrato chamado criaObjetoTabuleiro que retorna uma Peca da classe Lago. A fábrica concreta é: FabricaDeLago.

## 5. Builder

Há 2 Builder's implementados. Um para criar as pecas de cada jogador, e outro para criar os objetos do tabuleiro. O Builder das peças, tem a classe abstrata BuilderJogador, que tem os métodos para criar cada tipo de peça, Tem duas classes concretas chamadas ConcretBuilderJogador1, ConcretBuilderJogador1Horizontal, ConcretBuilderJogador2Horizontal e ConcretBuilderJogador2 que implementam os métodos abstratos. A classe DiretorJogador, recebe um Builder por parâmetro e chama o método de criação das peças. Esses métodos implementados de criação das peças, chamam o abstract factory para criar a peça desejada. O Builder dos objetos do tabuleiro tem uma classe abstrata BuilderObjetoTabuleiro que tem um método abstrato para criar Lagos, e tem as classes chamadas ConcretBuilderTabuleiro e ConcretBuilderTabuleiroHorizontal que implementam esse método, chamando a fábrica de lagos passada por parâmetro para criar um novo lago. O builder é chamado através das classes de Strategy de criação do tabuleiro, nas classes CriarTabuleiroHorizontal e CriarTabuleiroVertical.

## 6. Command

O padrão command, possui uma interface chamada Command que fica no pacote do Controller, essa interface tem um método chamado execute que recebe uma matriz de objetos do tabuleiro, um animal e um objeto padrão. Foram criadas as classes: EntrarArmadilha, EntrarLago, SairArmadilha, SairLago e MovimentarPeca. Todas essas classes implementam a interface Command. Dentro do método execute, na posição atual do animal vindo por parâmetro é setado o objeto padrão, na posição definida pelo tipo do command é setado o animal e é alterado a posição atual da classe do Animal. A chamada do command pode ser visto na classe Jogo no método realizaMovimentacao na linha 124.

## 7. Singleton

O padrão singleton foi implementado na View Tabuleiro, onde a classe Tabuleiro tem um objeto de si mesma chamado instance, um construtor privado que faz a criação do

Frame, e um método síncrono chamado getInstance() que retorna um Tabuleiro, que verifica quando o objeto instance está nulo, ele chama o construtor privado e sempre no final retorna o instance. Isso pode ser visto na classe Tabuleiro, no pacote View, onde o método getInstance() começa na linha 29.

## 8. Visitor

O padrão visitor, implementado no jogo, possui duas interfaces Visitor, uma para verificar o ataque em algumas peças e outra para visitar os animais do jogador. As interfaces são VisitorAnimaisJogador e VisitorAtaque. O VisitorAnimaisJogador é implementado nas classes VisitorAnimaisDefinirPosicoes, VisitorAnimaisJogadorSetarJogador, VisitorAnimaisNaoDeixarEntrarToca, VisitorAnimaisNaoDeixarMatarMesmoTime, VisitorAnimaisQuantidade, VisitorAnimaisSetarTabuleiro, VisitorAnimaisSetarImagemArmadilha e VisitorAnimaisVerificarImagens. Todas essas classes realizam uma função diferente, autoexplicativa com seu nome. O VisitorAtaque é implementado na classe VisitorAtacanteAnimal que, por sua vez, é ancestral da classe VisitorAtacanteRato. A classe VisitorAtacanteAnimal verificar a força e se a peça a ser atacada está ou não na armadilha, para adicionar em uma lista de posições possíveis para um determinado animal andar. Já o VisitorAtacanteRato, que herda de VisitorAtacanteAnimal, é utilizado para fazer a mesma funcionalidade com os Animais, mas quando é um elefante, o tratamento é diferente. As chamadas dos visitors podem ser identificadas nas seguintes classes:

- CriarTabuleiroVertical: no método criarTabuleiro, onde o primeiro Visitor é instanciado na linha 26.
- CriarTabuleiroHorizontal: no método criarTabuleiro, onde o primeiro Visitor é instanciado na linha 23.
- JogoIniciado: no método movimentaPeca, onde o primeiro Visitor é instanciado na linha 53.
- Animal, Rato, Tigre e Leão: no método verificarPosicoesPossiveis;

## 9. Strategy

O padrão Strategy foi utilizado para criação do tabuleiro e, por isso também sendo necessário para definir as posições possíveis de movimentação de um Animal. Há uma interface CriarTabuleiro, onde as classes CriarTabuleiroVertical e CriarTabuleiroHorizontal implementam o método criarTabuleiro. É definido a estratégia de criação do tabuleiro através de uma pergunta ao usuário no momento de iniciar o jogo, quando o usuário definir o tipo de tabuleiro que ele quer, é passado para o controller um boolean dizendo se é vertical ou horizontal. Isso pode ser visto na classe TabuleiroController, no construtor da classe, onde a estratégia é definida da linha 42 a 44. Sendo chamado seu método criarTabuleiro na linha 61. A Interface DefinirPosicoesPeca foi criada e tem sua implementação nas classes DefinirPosicoesPossiveisHorizontal e DefinirPosicoesPossiveis Vertical, há também outras classes que decoram elas, que será explicado no tópico sobre Decorator. As classes que definem as posições possíveis são instanciadas na classe Animal, no método verificarPosicoesPossiveis, começando na linha 49.

## 10. State

O padrão State foi utilizado para definir o estado do jogo. Quando o jogo é criado, o estado vem por padrão JogoAguardandoInicio, quando é tentado fazer a primeira movimentação

de peça, é trocado para JogoIniciado e, quando não tiver mais peças disponíveis ou algum jogador chegou na toca adversária, o estado é trocado para JogoFinalizado. A interface do State criada é JogoEstado, onde as classes JogoAguardandoInicio, JogoIniciado e JogoFinalizando herdam. É possível verificar o inicio da utilização dos estados na classe Jogo, setando JogoAguardandoInicio no construtor e, no método movimentaPeca que começa na linha 106, chama os estados para realizar a movimentação e verificar se deve ou não trocar para o próximo estado.

## 11. Decorator

O padrão Decorator foi utilizado para decorar as estratégias de definição de posições possíveis de cada Animal, sendo as classes base: DefinirPosicoesPossiveisVertical e DefinirPosicoesPossiveisHorizontal. A interface é DefinirPosicoesPeca, a classe abstrata do Decorator é DefinirPosicoesPecaDecorator e as classes que são as decoradoras são DefinirPosicoesPecaEntraLago, DefinirPosicoesPecaPulaLago e DefinirPosicoesPecaPulaLagoHorizontal. Foi implementado esse padrão para esta ocasião pois, todos os animais tem as mesmas movimentações, para cima, esquerda, direita e para baixo. Porém, há animais que tem movimentações a mais, como o rato, que pode entrar no lago. Por isso foi utilizado o padrão decorator, pois o rato faz as posições normais e também entra no lago. A chamada dos decorators pode ser vista no método verificarPosicoesPossiveis nas classes Rato, Tigre e Leao. Este método foi sobrescrito da classe ancestral Animal para poder aplicar o decorator da maneira correta.