Universidade Federal de Campina Grande

Centro de Engenharia Elétrica e Informática – CEEI
Unidade Acadêmica de Sistemas e Computação – UASC

Disciplina: Laboratório de Programação 2

Laboratório 06

Introdução

Neste laboratório queremos experimentar como entender e estender um software OO. Partiremos de um sistema pronto e iremos introduzir novas funcionalidades nesse sistema e garantir que o sistema esteja correto após as modificações. Vamos exercitar a leitura e extensão de código escrito por outra pessoa.

O uso de testes automáticos é essencial para evoluirmos um código grande. É a única forma prática de conferirmos que um grande número de funcionalidades ainda está OK depois de alguma modificação. Um ponto muito importante desse laboratório é que **usaremos os testes de unidade para entender o que você implementou.** Leremos os testes para entender o código. O que não estiver testado e compreensível nos testes não será considerado. Por isso, é melhor ir fazendo os testes na medida que você vai implementando as alterações. Não deixe para fazer os testes no final.

O sistema Dominó Brutal 1.0

O sistema no qual você trabalhará é um jogo de Dominó. Ele não é interativo: damos a ele as estratégias dos jogadores, ele simula o jogo e nos diz quem ganhou. É um sistema inspirado no Robocode, mas atualizado para um esporte mais relevante e complexo do que guerras de robôs, o dominó. Assim como no Robocode, qualquer pessoa pode programar novas estratégias e plugá-las no sistema para testá-la contra outras estratégias dela ou de outras pessoas.

Você baixará o código inicial do sistema **neste repositório no github** (clica em code, download zip). Sua primeira tarefa é entender o código atual e conseguir descrevê-lo. Explicamos mais na próxima seção. Depois desse entendimento, temos as extensões que queremos que você faça.

Evolução do sistema

Evolua o Dominó Brutal versão 1.0 para a versão 2.0 (Dominó Ainda Mais Brutal) com as seguintes funcionalidades:

US1: Desempate

Como jogador, desejo evoluir a avaliação de vencedor do jogo como dito a seguir, para ter um dominó com regras mais realistas. A modificação é que quando o jogo tranca porque chegou em uma situação onde ambos os jogadores passaram a vez (o atual empate), deve haver um vencedor da seguinte maneira:

- Se um dos jogadores tiver menos peças na mão que o outro, ele será vencedor.
- Se eles têm o mesmo número de peças na mão, se um dos jogadores tiver uma soma menor nos números das peças de sua mão que o outro, ele será o vencedor.

O jogo só deve ser empate se ambos tiverem o mesmo número de peças e a mesma soma no número das peças.

US2: Pontuação por vitória

Como jogador, quero que lá e lô e carroção sejam vitórias que valem mais, para ter regras mais realistas. A pontuação, retirada da Wikipedia, é a seguinte: *uma batida normal (em uma única "cabeça") vale 1 ponto, mesma pontuação quando o jogo trancar e acontecer a contagem, batida de "carroça"* (uma peça com 2 números iguais) *vale 2 pontos, o famoso "lá e ló" que significa bater com uma pedra simples* que encaixa *nas duas pontas, vale 3 pontos, já o "lá e ló" de carroça, também chamada de "quadrada", "cruzada" ou "carroça cruzada" vale 6 pontos.* Após implementar essa pontuação, altere a classe DominoBrutalRepetido para considerar pontuações além de vitórias, e para contar e imprimir quantas vezes cada jogador teve cada tipo de vitória (você escolhe o formato).

US3: Duas estratégias suas

Como jogador, quero ter duas estratégias de jogo novas, para que eu possa me divertir mais jogando. Para isso, você deve criar duas implementações de EstrategiaDeJogo. Exemplos de estratégias possíveis são guardar o número que mais se repete na mão para o final do jogo, jogar sempre o número que o oponente mais jogou até agora, ou jogar sempre primeiro os carroções (peças com 2 números iguais). Mas pode inventar a sua do jeito que quiser, nós estudaremos o que sua estratégia faz lendo os teste de unidade dela, então capriche nisso.

US4: EstratégiaComposta

Como usuário, quero criar uma nova EstrategiaDeJogo, a EstrategiaDeJogoComposta que recebe no seu construtor, uma lista de objetos do tipo EstrategiaDeJogo. A EstrategiaDeJogoComposta alterna a estratégia interna a ser usada em cada jogada, i.e. alterna a estratégia a ser usada a cada chamada do método decideJogada(List<Peca> mao, VisaoDaMesa mesa).

Veja o exemplo de uso dessa estratégia no DominoBrutal:

```
List<EstrategiaDeJogo> estrategias = new ArrayList<>();
estrategias.add(new JogaPrimeiraPossivel());
estrategias.add(new JogaMaior());
EstrategiaDeJogo estrategiaComposta = new EstrategiaDeJogoComposta(estrategias);
Jogo j = new Jogo("J1", new JogaPrimeiraPossivel(),  "J2", estrategiaComposta, 12);
```

Na primeira vez que o decideJogada de estrategiaComposta é invocada, esse método usa a estratégia JogaPrimeiraPossível (invocando o decideJogada desse objeto). Na segunda vez, irá usar a estratégia JogaMaior. Na terceira vez voltará a usar a estratégia de JogaPrimeiraPossivel, sempre invocando cada estratégia da lista em ordem.

US5: Estratégia de Jogo Comparadora

Como usuário de Jogo, quero poder criar uma estratégia, o JogaPrimeiraPossivelComparadora que recebe um Comparator<Peca> como parâmetro. No início de "decideJogada", essa estratégia deve ordenar a lista de peças da mão de acordo com a ordem recebida pelo Comparator e, no mais, jogar da mesma forma que o JogaPrimeiraPossivel funciona.

Para testar seu sistema, crie também a classe PecaPadraoComparator que é do tipo Comparator<Peca> e que ordena as peças primeiro pelo número direito, depois pelo número da esquerda, (ou primeiro pelo número da esquerda e depois da direita, a sua escolha). Exemplo ao ordenar primeiro pela esquerda e depois pela direita: [1/1, 1/2, 1/3, 1/4, 1/5, 1/6, 2/2, 2/3, 2/4, ...].
Dica: para ordenar uma lista com um objeto Comparator, use o método:

- java.util.Collections.sort(List<T> list, Comparator<? super T> c);
-

Veja o exemplo de uso dessa estratégia no DominoBrutal:

```
Comparator<Peca> cmp = new PecaPadraoComparator();
EstrategiaDeJogo estrategiaCmp = new JogaPrimeiraPossivelComparadora(cmp);
Jogo j = new Jogo("J1", new JogaPrimeiraPossivel(),  "J2", estrategiaCmp, 12);
```

Ao receber a mão, no decideJogada, ela será ordenada de acordo com o comparador passado como parâmetro da estratégia.