UNIVERSIDADE FEDERAL DE VIÇOSA CAMPUS UFV-FLORESTAL

Disciplina: Programação Orientada a Objetos, CCF 313

Docente: Philipe de Freitas Melo

Discentes: Gustavo Eufrausino de Medeiros, 5062
Henrique Fernandes de Andrade, 5071
Eduardo Antunes dos Santos Vieira, 5076
Carlos Márcio Moreira Costa, 5079
Ana Lúcia de Souza, 5084
Lucas Fonseca Sabino Lana, 5105

Documentação Trabalho Prático de POO

Truco-java

Florestal - MG

2023

SUMÁRIO

INTRODUÇÃO

A proposta do trabalho consistia na elaboração de um sistema arbitrário na linguagem Java, utilizando os conceitos do paradigma da orientação a objetos. O sistema escolhido pelo grupo foi um jogo de truco mineiro, um tradicional jogo de estratégia utilizando cartas de baralho. As regras do truco estão inclusas na entrega parcial do trabalho, mas, para os efeitos desta introdução, o jogo consiste na execução de rodadas, mãos, jogos e uma partida, de forma retroativa. A cada rodada, cada um dos quatro jogadores dispostos em duas duplas recebe 3 cartas. O objetivo da dupla é vencer o maior número de pontos, com o auxílio de propostas para aumentar o valor das mãos, que inicialmente valem 2 pontos. Ao obter-se 12 pontos, um jogo acaba. Uma equipe é vencedora ao vencer 2 jogos.

Os encontros ocorreram pelo Discord, onde os membros reuniram-se e elaboraram um modelo UML do programa e suas classes. Em seguida, após um brainstorming inicial e com as classes elaboradas, houve a divisão de tarefas, em que cada dupla de alunos ficou responsável por um grupo delas. Após a elaboração das classes, houve uma segunda divisão de tarefas mais amplas, que incluía o teste do programa, o desenvolvimento das classes e os ajustes ao UML inicial que, naturalmente, foi morfando-se ao longo da linha do tempo do projeto.

MODELAGEM UML

Abaixo está a imagem do diagrama UML do projeto. O UML foi a primeira etapa do desenvolvimento do jogo, porém, devido a mudanças no código ao longo do desenvolvimento do sistema, foi necessário uma mudança bastante radical no diagrama em relação ao primeiro modelo criado.

Link para acesso ao diagrama:

- https://drive.google.com/file/d/1c-EE-MFBjt4Kd-7pWbJeNKCy9kEaABIf/view?usp=sharing

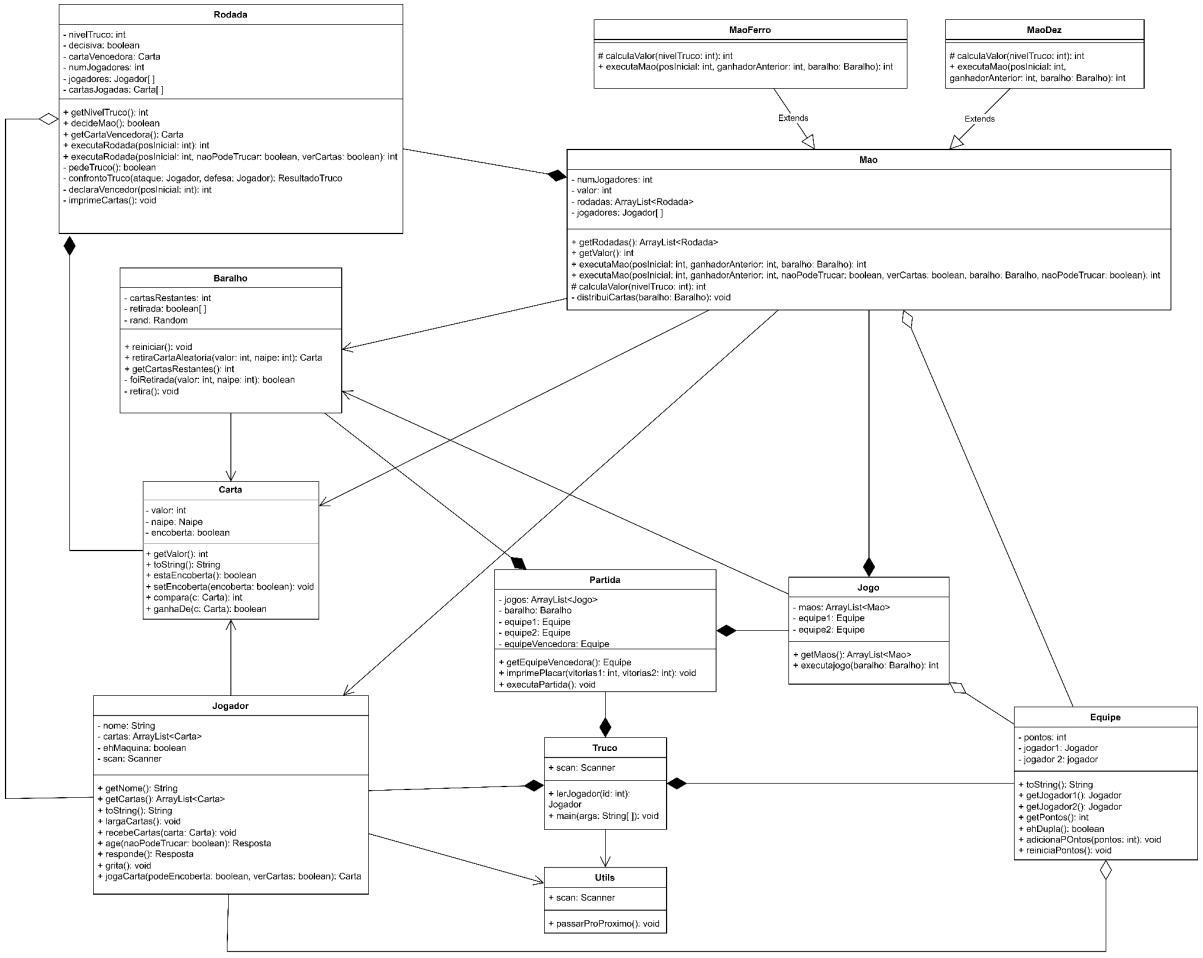

Figura 1: Diagrama UML atualizado

PRINCIPAIS FUNCIONALIDADES

Quanto às funcionalidades, o programa conta com as classes Baralho, Carta, Equipe, Jogador, Jogo, Mao, MaoDez, MaoFerro, Naipe, Partida, Resposta, ResultadoTruco, Rodada e Truco, este último sendo nosso main(). De forma mais abstrata, o programa é capaz de simular uma partida(**Partida.java**) de truco comum, com quatro jogadores(**Jogador.java**) em duplas (**Equipe.java**) e um baralho(**Baralho.java**) de quarenta cartas, utilizando doze delas para cada mão, distribuídas por um gerador de números aleatórios com correção de repetições. Para cada jogo(**Jogo.java**), o programa também conta com um contador de pontos, que variam a cada pedido de truco, seis, nove e doze, bem como todas as possibilidades de resposta. É também possível gritar, parte fundamental de um jogo de truco real. Abaixo, exibições de tais classes e funcionalidades.

```java
// Determina a quantidade de pontos a partir de um "nível" de truco
protected int calculaValor(int nivelTruco) {
    if(nivelTruco <= 1) return 2 + nivelTruco * 2;
    else return 4 + nivelTruco * 2;
}

// Distribui cartas aleatorias a todos os jogadores
private void distribuiCartas(Baralho baralho) {
    // Embaralha o baralho e descarta quaisquer cartas que os jogadores já tenham
    baralho.reiniciar();
    for(Jogador j : jogadores) j.largaCartas();

    // Distribui as cartas
    for (int i = 0; i < 3; i++) {
        for (Jogador jogador : jogadores) {
            // Verifica se ainda há cartas no baralho
            if (baralho.getCartasRestantes() > 0) {
                // Remove a carta do topo do baralho e a dá ao jogador
                Carta carta = baralho.retiraCartaAleatoria();
                jogador.recebeCarta(carta);
            }
        }
    }
}
```

Figura 2: Trecho da classe Mao{} que determina o valor e distribui as cartas aleatoriamente.

```java
public class Partida {
	private Equipe equipe1;
	private Equipe equipe2;
	private Equipe equipeVencedora;
	private ArrayList<Jogo> jogos;
	private Baralho baralho = new Baralho();

	public Partida(Equipe equipe1, Equipe equipe2) {
		jogos = new ArrayList<Jogo>();
		this.equipe1 = equipe1;
		this.equipe2 = equipe2;
	}

	public Equipe getEquipeVencedora() {
		return equipeVencedora;
	}

	public void executaPartida() {
		int vitorias1 = 0, vitorias2 = 0;
		while(vitorias1 < 2 && vitorias2 < 2) {
			Jogo jogo = new Jogo(equipe1, equipe2);
			int eq = jogo.executajogo(baralho);
			jogos.add(jogo);
			if(eq == 1) ++vitorias1;
			else ++vitorias2;
		}
		equipeVencedora = vitorias1 > vitorias2 ? equipe1 : equipe2;
	}
}
```

Figura 3: Totalidade da classe Partida{} que executa as partidas, contabiliza vitórias e inicia os jogos.

INSTRUÇÕES DE COMPILAÇÃO E EXECUÇÃO

Fizemos uso de um fluxo de trabalho ligeiramente inconvencional no que diz respeito à compilação e execução do programa e seus testes. Como muitos de nós não fizemos uso de uma IDE durante o desenvolvimento, acabamos por não utilizar um sistema de compilação, como Maven ou Gradle. Em vez disso, escrevemos alguns simples *scripts* em linguagem shell para facilitar os processos de invocação manual dos executáveis *java* e *javac*.

O primeiro e mais utilizado desses *scripts* é o *run.sh*, que simplesmente compila e executa todos os arquivos no pacote **br.ufv.truco**. Ele pode ser invocado sem a especificação de nenhum argumento de linha de comando. Os arquivos *.class* gerados pela compilação são colocados na pasta out.

O segundo *script* utilizado é o *test.sh*, que compila e executa os testes utilizando a

biblioteca JUnit. Os testes se encontram no pacote **br.ufv.truco.testes**. O *script test.sh* deve ser invocado especificando como argumento de linha de comando o nome do teste que se deseja executar, que em geral corresponde ao nome de um dos arquivos *.java* no pacote de testes. Para que ele funcione corretamente, é necessário antes ter compilado o pacote principal usando o *run.sh* e também ter baixado o JUnit e suas dependências.

Para simplificar o processo de fazer o *download* dos arquivos *.jar* do JUnit e suas dependências, criamos um terceiro *script, download-deps.sh.* Ele utiliza a ferramenta *wget*, padrão no Linux, para baixar os arquivos. Então, ele coloca-os na pasta deps, onde o *test.sh* espera que eles estejam. Dessa forma, ao tentar executar os testes pela primeira vez, tendo antes executado o projeto com *run.sh*, deve-se executar o comando *./download-deps.sh && ./test.sh <teste>*, onde qualquer teste serve.

DESENVOLVIMENTO

O início do desenvolvimento do projeto foi a elaboração e construção de um diagrama UML, sendo necessário para planejar o funcionamento do jogo e quais classes haveriam. Após o término desta etapa, seguimos para a codificação, nos baseando fortemente no UML.

Dividimos conjunto de classes para duplas, sendo os conjuntos classes que são mais conectadas umas às outras do que com o resto do sistema, por exemplo; as classes Mão, Rodada e Jogo. Mas, no decorrer do desenvolvimento cada integrante acabou assumindo funcionalidades inteiras, onde acabava tendo que modificar o funcionamento de não apenas uma classe.

Sequencialmente, à medida que foi sendo implementadas as ideias e classes do diagrama, foi percebido que teríamos que nos distanciar do UML, já que acabamos não pensando/ ou errando a hierarquia e funcionamento de métodos, além de outros casos onde não estavam previstos no diagrama.

Dessa forma, tivemos que realizar uma reestruturação do diagrama UML, para corresponder com o rumo que o projeto tomou. Além disso, com a finalização dos códigos, começou a realização de testes, para não apenas ver se o projeto funcionava, mas também para procurar possíveis erros (bugs) em ações do jogo.

Imediatamente, após realizar os testes e encontrar os erros, houve a etapa de correção destes erros. Ademais, foi necessário realizar estes dois processos, procura e correção de erros, mais de uma vez a fim de ter o projeto rodando da melhor forma possível.

Posteriormente, devido haver ainda um tempo para melhorias no projeto, foi

implementado duas “novas” funcionalidades, primordiais para o truco, a mão de 10 e a mão de ferro. Com todas as funcionalidades desejadas implementadas e funcionando, era interessante haver uma melhoria na interface do jogo. Embora, não havendo tempo hábil para o desenvolvimento de uma interface gráfica, foi realizada uma melhoria na interface do jogo, uma interface de terminal, sendo o foco destas melhorias uma melhor experiência e maior interatividade para os jogadores.

Finalmente, houve a realização dos testes unitários, sendo a última etapa,envolvendo código, de desenvolvimento do projeto. Sendo a verdadeira última etapa do projeto a realização desta documentação.

DESAFIOS

Os principais desafios enfrentados pelo grupo consistiam na quantidade de segmentação necessária para um bom fluxo de ideias, bem como o tamanho do código desenvolvido para cada classe, já que há catorze delas, cada uma variando de 8 a 200 linhas, exigindo sempre a declaração de cada variável e atributo, bem como seus muitos métodos, que, apesar de altamente reutilizados, trazem a sensação de que o fim do desenvolvimento está distante no horizonte. Outra dificuldade foi a modelagem do jogo com suas regras e nuances para a programação orientada a objetos. Aplicar conceitos de abstração, herança, polimorfismo e encapsulamento nos ajudou a fazer um código mais flexível, mas administrar as ideias para escrever um código eficiente não foi simples. A divisão das tarefas entre os membros do grupo também se provou um desafio. Em disciplinas anteriores, a quantidade de pessoas por grupo era menor, de modo que essa dificuldade não era tão perceptível.

CONCLUSÃO

Por fim, o trabalho em grupo fez com que os membros se familiarizassem com os conceitos do paradigma da orientação a objetos, seus algoritmos e estruturas, assim como sua segmentação e abstração e como isso afeta o resultado final do código. As estruturas estudadas, apesar de descontraídas quanto à execução, foram de extrema importância para a prática do paradigma, suas propriedades e seus pontos fracos e fortes, além de evidenciar quando e como utilizar seus conceitos. A chance de desenvolver um jogo é igualmente única e valiosa, principalmente no ambiente acadêmico. Foi, de fato, extremamente gratificante e esclarecedor.