Universidade Federal de Minas Gerais
Instituto de Ciências Exatas
Departamento de Ciência da Computação
Disciplina Programação Modular (DCC0052) – 2016/01

Trabalho Prático -2 – UNO

Alexandre Alphonsos Rodrigues Pereira
Jerônimo Nunes Rocha

1.Introdução:

O trabalho proposto foi a implementação de um jogo de cartas com suas funcionalidades específicas. O jogo escolhido foi o UNO.

O objetivo do trabalho foi familiarizar o aluno com as etapas de análise, modelagem e o desenvolvimento de um sistema.

Seguem as regras do jogo:

O jogo deve ser jogado por maiores de 7 anos, e entre 2 e 10 jogadores(no trabalho são 4 jogadores, 3 máquinas contra o humano). O baralho é composto por cartas de quatro cores: verde, amarelo, vermelho e azul. As fileiras de cada cor variam entre 0 e 9. Existem três ações especiais para cada tipo de cor de carta, identificadas como "pular", "comprar duas" e "inverter". Há também cartas de ações especiais com fundo preto, "coringa" e "coringa comprar quatro". Para cada carta regular ou de ação, existem duas das mesmas no baralho, com exceção do 0, que só possui uma unidade. Há quatro "coringas que mudam de cor" e quatro "coringas comprar quatro", o que resulta num total de 108 cartas. Para diferenciar o 6 do 9, é utilizado um sublinhado embaixo da carta respectiva.

Para começar o jogo, são distribuídas sete cartas a cada jogador, e a carta que ficou em cima do baralho é virada para cima, sendo esta a primeira. Caso essa carta possua uma "habilidade especial" (nomeadamente pular, comprar duas e inverter), ela é tratada como se o jogador que deu as cartas tivesse jogado as mesmas. Se a carta for um coringa o jogador escolhe a cor que deve começar. Se for um coringa comprar quatro, deve ser devolvida ao baralho. O jogo começa com a pessoa posicionada ao sentido horário de quem distribuiu as cartas.

Em cada oportunidade, o jogador pode jogar uma carta de sua mão que seja igual à cor ou ao número (se for uma carta numérica) ou o símbolo (Se for uma carta que possui uma habilidade especial) ou uma carta idêntica (igual em ambos, a cor e o número ou símbolo) da última carta apresentada, ou então jogar um coringa ou coringa comprar quatro. Se a pessoa não possuir carta para jogar na ocasião, deve comprar e, caso ainda continue sem a carta precisa, perder seu turno, repetindo o processo até sair uma carta jogável. Se o jogador possuir a carta que precisa para ser jogada, mas não jogá-la e comprar outra, nenhuma penalização é aplicada. Depois de um jogador jogar a sua carta, o próximo ao sentido horário ou anti-horário - se estiver invertida a ordem - joga. As cartas podem ser jogadas na sequência (crescente ou decrescente) dos números, caso possuam a mesma cor.

Se as cartas que eram utilizadas para comprar esgotarem, as jogadas na mesa são embaralhadas novamente e colocadas como pilha. O jogo termina quando um jogador está sem nenhuma carta na mão.

2.Implementação:

A implementação do trabalho foi dividida em três pacotes seguindo o padrão **MVC**:

Um **controlador (controller**) envia comandos para o modelo para atualizar o seu estado por exemplo, editando um documento). O controlador também pode enviar comandos para a visão associada para alterar a apresentação da visão do modelo (por exemplo, percorrendo um documento).

Um **modelo** (**model**) armazena dados e notifica suas visões e controladores associados quando há uma mudança em seu estado. Estas notificações permitem que as visões produzam saídas atualizadas e que os controladores alterem o conjunto de comandos disponíveis. Uma implementação *passiva* do MVC monta estas notificações, devido a aplicação não necessitar delas ou a plataforma de software não suportá-las.

A **visão** (**view**) Gera uma representação (Visão) dos dados presentes no modelo solicita do modelo.

Cada pacote possui sua descrição (package-info.java), seu diagrama de classes (package-diagram.*) e suas respectivas classes. Uma descrição de cada classe, seus métodos e funcionalidades pode ser conferido em na pasta do tp ***doc/index.html***.

Além disso utilizamos 3 padrões de projetos conforme ensinados ao longo da disciplina:
-Builder
-Factory Method
-Singleton

DeckBuilder:
Utilizado para a construção do baralho de UNO que possui um representação complexa e particular. Caso o jogo de baralho fosse trocado seria possivel extender o Builder para contruir um outro tipo de baralho.

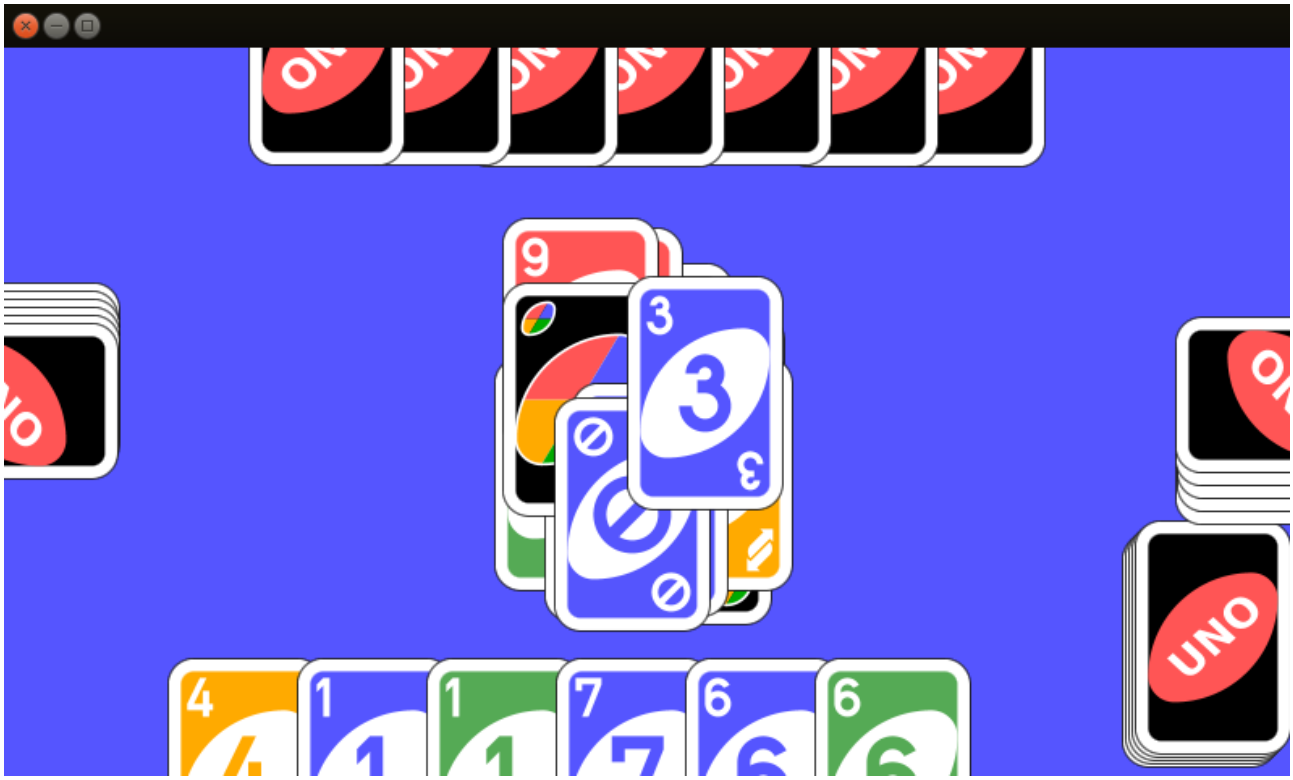
GraphicCardFactory:
Utilizado para delegar a instanciação das cartas para as subclasses apropriadas. O GraphicCardFactory também é singleton para garantir a existência de apenas uma instância dessa classe.

Os diagramas UML podem ser encontrados dentro dos pacotes.

3.Testes:

Os testes realizados foram a execução de diversas partidas verificando principalmente o efeito das cartas especiais, o sentido da rotação da mesa e os turnos dos jogadores.

Alguns bugs relativos as animações e redimensionamento da tela podem aparecer mas com baixíssima frequencia.



4.Conclusão:

A implementação foi feita com sucesso. A maior dificuldade encontrada foi o desenvolvimento da interface gráfica e a sua interface com o controlador do jogo. A execução do jogo se mostrou estável e com o comportamento dentro do esperado.

O trabalho se mostrou muito útil na consolidação dos conhecimentos aprendidos e discutidos durante as aulas.

5.Referências:

-Aulas
- https://docs.oracle.com/javase/7/docs/api/
-https://pt.wikipedia.org/wiki/MVC