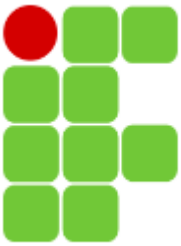

Instituto Federal de Minas Gerais

Campus Ouro Branco

Professor: Saulo Henrique Cabral Silva

Nome: Gabriel Oliveira Gomes

Apresentação:

O programa desenvolvido implementa a construção de um jogo de cartas como requisito da disciplina de Programação mobile.

Desenvolvimento do trabalho:

Foi utilizada a linguagem de programação Kotlin e o modelo de desenvolvimento model - view - controller. O primeiro passo foi a leitura do arquivo cartas.csv disponibilizado e então a criação de um objeto que pudesse conter todas as características necessárias.

Classes utilizadas:

Card:

```
class Card (var name:String, val description:String, var attack:Int, var defense:Int, val
cardClass:String, var attackMode:Boolean=false, var equipmentOn:Boolean=false, var
hasAttacked:Boolean = false) {
```

A Classe card é responsável por guardar cada linha do documento csv disponibilizado, com as informações de nome, descrição, ataque, ataque, defesa e tipo de monstro. Porém com as requisições de métodos específicos e mecânicas a se adotar no jogo, criou-se outros atributos para a classe, sendo eles: attackMode(Variável do tipo Boolean declara se a carta está em modo de ataque ou defesa), equipmentOn(Variável do tipo Boolean que declara se o monstro em questão já utiliza de algum equipamento, evitando que equipamentos fiquem acumulados em um mesmo monstro) e a variável hasAttacked(dita se a carta já atacou ou não durante o

turno, já que a mesma só pode realizar 1 ataque por turno, e não pode ser declarada para modo de deseja após atacar_

Player:

```
class Player (val name:String, var lifePoints:Int=10000, var hand:Array<Card?> = arrayOfNulls(10),
var field:Array<Card?> = arrayOfNulls(5))
```

A Classe Player é composta por seu nome, a quantidade de pontos de vida, e 2 arrays do tipo Card, que podem receber valores nulos. Um dos arrays trata-se das cartas "em mãos" do jogador, o outro array trata do campo de batalha onde o mesmo poderá posicionar as cartas desejadas.

Field:

```
class Field(var player1:Player, var player2:Player, var deck: MutableList<Card>)
```

A Classe Fiel é composta apenas de um array de do tipo Cards, que será o deck que recebe os valores lidos no documento CSV, e os 2 players que se enfrentarão.

Funcionamento e apresentação:

O programa não fez uso de interfaces, optando pro apresentar ao jogador as informações diretamente no console. As partes do jogo descritas na proposta do problema foram divididas em etapas de jogo, sendo elas:

roundStart:

Esta etapa é responsável pela compra de cartas pelo jogador atual

placePhase:

Esta etapa é responsável pela ativação de cartas da mão do jogador, para o seu campo de batalha..

changeMode

Nesta etapa, o jogador pode escolher monstros que ainda não atacaram, para alterar seu modo de ataque para defesa ou vice-versa

battlePhase

Nesta etapa é tratado a entrada dos inputs de batalha, monstro que irá atacar, monstro alvo, e dano aos pontos de vida, caso haja. Monstros que já atacaram 1 vez no round, não devem atacar mais, para isso, é declarado que apenas monstros com o atributos hasAttacked com valor false podem atacar, e aqueles que ataquem tem o valor alterado para true

endPhase

Nesta etapa é realizada o reset do estado das cartas, ou seja, todas as cartas que atacaram, devem poder atacar no próximo round, para isso, o atributo deles hasAttacked é alterado para false novamente, e o campo de batalha é invertido, para que o próximo player possa jogar.

Problemas encontrados:

Foi perdido uma quantidade excessiva de tempo ao tentar procurar o porque de equipamentos não estarem podendo ser utilizados, realizado testes e debugging, procurando por erros em comparações booleanas, para então descobrir que o erro se encontrava no método isMonster, que ao invés de comparar a palavra "monstro" herdada do arquivo cartas.csv, estava comparando a palavra "monster" sempre resultando em um valor de falso verdadeiro.

Conclusão:

Apesar de ser um trabalho relativamente simples, sua implementação trouxe alguns problemas, principalmente no que diz respeito as regras usadas no jogo, dito isso, alguns dos aspectos implementados foram baseados em outros card games, como por exemplo a separação de fases do turno, exemplo retirado do jogo de cartas Yu-Gi-Oh!