### 

### 

### 

### 

### 

### 

### **Relatório Jogo de Cartas**

Lucas Da Costa Miranda

Luiz Filipe Ferreira Ramos

#### **Descrição Geral**

O projeto é um jogo de cartas baseado em turnos, desenvolvido em Kotlin, onde dois jogadores competem utilizando cartas de monstros e equipamentos. O jogo inclui mecânicas de combate, estratégia, e gerenciamento de cartas, com um fluxo dinâmico e funcionalidade robusta.

### **Componentes e Classes Principais**

* **Card**
  + Representa uma carta no jogo.
  + Atributos:
    - name: Nome da carta.
    - description: Descrição da carta.
    - type: Tipo da carta (MONSTER ou EQUIPAMENT).
    - attack: Valor de ataque.
    - defense: Valor de defesa.

data class Card(

val name: String,

val description: String,

val type: CardType,

val attack: Int,

val defense: Int

)

* **CardType**
  + Enumeração para classificar os tipos de cartas:
    - MONSTER: Representa monstros.
    - EQUIPAMENT: Representa equipamentos.
  + Inclui um método estático fromString para converter strings em valores da enumeração.

**Codigo**:  
 enum class CardType {

MONSTER, EQUIPAMENT;

companion object {

fun fromString(type: String): CardType? {

return when (type.uppercase()) {

"MONSTRO", "MONSTER" -> MONSTER

"EQUIPAMENTO", "EQUIPMENT" -> EQUIPAMENT

else -> null

}

}

}

}

* **Deck**
  + Gerencia o baralho de cartas, realizando operações como leitura de um arquivo CSV e embaralhamento.
  + Atributos:
    - cardList: Lista de cartas no baralho.
    - filePath: Caminho para o arquivo CSV de cartas.
    - currentCard: Índice da próxima carta a ser distribuída.
  + Métodos:
    - getCard(): Retorna a próxima carta do baralho.
    - hasCard(): Verifica se há cartas disponíveis.
    - readCardsFromCSV(): Lê e valida as cartas do arquivo CSV.

**Codigo:**  
 class Deck {

private val cardList: MutableList<Card> = mutableListOf()

private val filePath = "cartas.csv"

private var currentCard = 0

init {

readCardsFromCSV()

cardList.shuffle()

}

fun getCard(): Card? {

if (currentCard >= cardList.size) return null

return cardList[currentCard++]

}

fun hasCard(): Boolean = cardList.isNotEmpty()

private fun readCardsFromCSV() {

val resourceURL = javaClass.getResource(filePath)

if (resourceURL == null) {

println("Arquivo de cartas não encontrado.")

return

}

val file = File(resourceURL.toURI())

file.forEachLine { line ->

val tokens = line.split(";")

if (tokens.size < 5) return@forEachLine

val card = Card(

name = tokens[0].trim(),

description = tokens[1].trim(),

attack = tokens[2].toIntOrNull() ?: return@forEachLine,

defense = tokens[3].toIntOrNull() ?: return@forEachLine,

type = CardType.fromString(tokens[4].trim()) ?: return@forEachLine

)

cardList.add(card)

}

}

}

* **GameManager**
  + Controla o fluxo do jogo, gerencia turnos e interações entre os jogadores e o baralho.
  + Atributos:
    - players: Lista de jogadores.
    - currentTurn: Índice do jogador atual.
    - drawPile: Instância do baralho.
    - NUMBER\_OF\_PLAYERS: Número de jogadores (fixado em 2).
    - DEFAULT\_PLAYER\_LIFE: Vida inicial de cada jogador (1000).
  + Principais funcionalidades:
    - Inicia o jogo e realiza o loop de turnos.
    - Executa ações como atacar, descartar cartas, equipar monstros, e mudar estados de monstros.
    - Gerencia condições de vitória e impressão do estado do tabuleiro.

**Codigo:**  
class GameManager {

private val players: MutableList<Player> = mutableListOf()

private var currentTurn = 0

private val drawPile = Deck()

fun processGame() {

if (!drawPile.hasCard()) {

println("Erro ao carregar cartas. Jogo abortado.")

return

}

println("===== JOGO DE CARTAS TABLEMONSTER =====")

for (i in 1..2) {

println("Digite o nome do jogador $i: ")

val name = readln()

players.add(Player(name, 1000))

}

while (true) {

val currentPlayer = players[currentTurn]

val opponent = players[1 - currentTurn]

if (currentPlayer.getCurrentLife() <= 0) {

println("${opponent.name} venceu!")

return

}

printTableState(players)

println("Vez de ${currentPlayer.name}. Escolha sua ação:")

// Fluxo de ações: atacar, posicionar monstros, etc.

currentTurn = (currentTurn + 1) % players.size

}

}

}

### **Pontos de Destaque**

* **Estrutura modular**: Cada classe é responsável por um aspecto específico do jogo.
* **Validações robustas**: Durante a leitura do CSV, dados inválidos são descartados com mensagens de erro claras.
* **Experiência de jogo fluida**: O sistema de turnos, junto com a impressão dinâmica do estado da mesa, cria um fluxo de jogo imersivo.