Documentação do jogo de cartas

1. Introdução

O trabalho desenvolvido consiste em um jogo de cartas colecionáveis, onde dois jogadores alternam turnos para posicionar monstros no campo, equipá-los e atacar o oponente com o objetivo de reduzir sua pontuação até chegar a zero (cada um começa com 10.000 pontos). O jogo simula um duelo estratégico, exigindo o gerenciamento eficiente de cartas e o planejamento de jogadas.

Cada jogador possui um deck inicial, a partir do qual cinco cartas são distribuídas para sua mão no início da partida. Durante o turno, o jogador pode realizar ações como posicionar monstros, equipá-los, mudar seus estados, atacar ou passar a vez. A vitória ocorre quando a pontuação de um dos jogadores chega a 0 ou quando as cartas no baralho acabam.

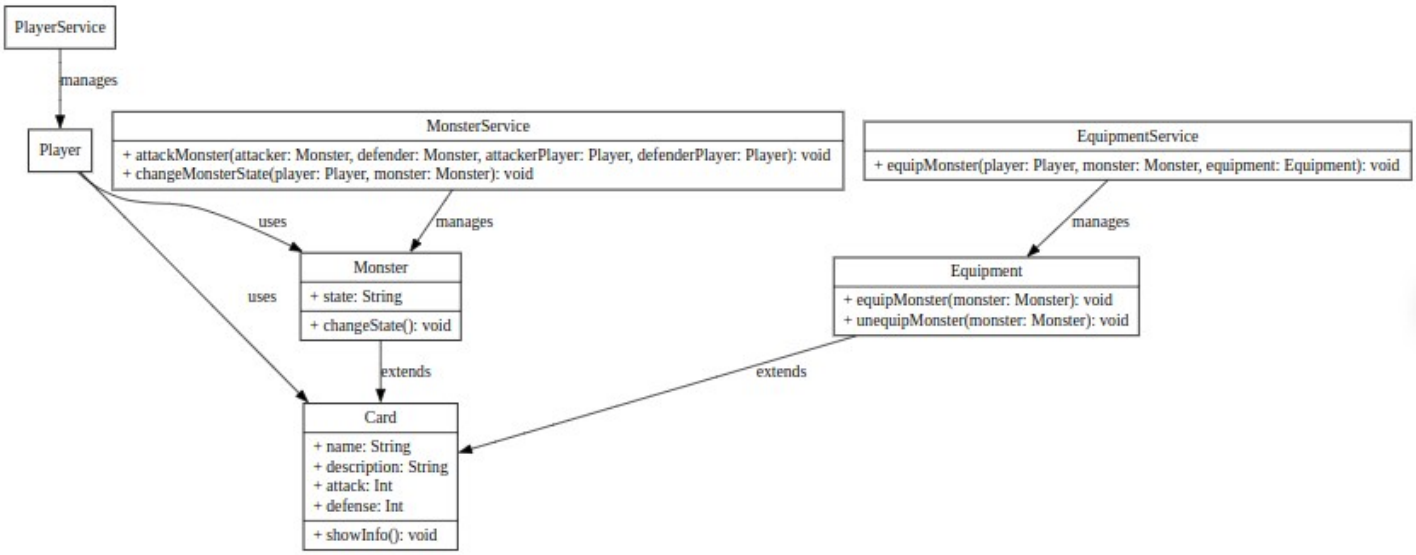
1.2 Implementação

Estrutura de Dados

O jogo foi implementado utilizando a linguagem Kotlin, com foco nos conceitos de Orientação a Objetos (POO). Os principais elementos do programa estão representados pelas classes e objetos a seguir:

- **Card**: Classe abstrata base para representar cartas genéricas.
    - **Monster**: Subclasse que representa cartas de monstros, com atributos como ataque, defesa e estado (ofensivo ou defensivo).
    - **Equipment**: Subclasse que representa cartas de equipamentos, que podem ser atribuídos a monstros para aumentar seus atributos.
- **Player**: Representa os jogadores, armazenando informações como pontuação, mão de cartas, monstros posicionados no campo e o nome do jogador.
- **Serviços**:
    - **PlayerService**: Gerencia as ações dos jogadores, como distribuir cartas e posicionar monstros.
    - **MonsterService**: Lida com ações específicas de monstros, como ataques e mudanças de estado.
    - **EquipmentService**: Gerencia a aplicação de equipamentos em monstros.

1.3 Diagrama de Classes:



1.4 Funcionamento das funções:

Distribuição de Cartas:

- Função “distributeCards” distribui cartas do baralho inicial para os jogadores no início do jogo.
- Cada jogador recebe um número definido de cartas.

Posicionar Monstros no Campo:

- O jogador seleciona um monstro da mão e define seu estado inicial (offensive ou defensive).
- O monstro é removido da mão e adicionado ao campo.

Ataques:

- Um monstro posicionado pode atacar diretamente o oponente (se este não tiver monstros) ou atacar um monstro adversário.
- O dano causado é calculado com base nos atributos de ataque e defesa.

Compra de Cartas:

- Após cada ação, o jogador compra uma nova carta aleatória do baralho. Se o baralho estiver vazio, nenhuma carta é comprada.

Finalização do Jogo:

- A cada turno, verifica-se se a pontuação de algum jogador chegou a 0 ou menos. Se isso ocorrer, o jogo é encerrado, e o vencedor é anunciado.

Formato de Entrada e Saída de Dados

- **Entrada**:
    - As cartas são lidas de um arquivo CSV contendo informações sobre nome, ataque, defesa e tipo (monstro ou equipamento).

    - Durante o jogo, as entradas do jogador são realizadas via teclado.

- **Saída**:
    - As informações do estado atual do jogo, incluindo as cartas na mão, monstros no campo e pontuação, são exibidas no console.
    - Mensagens informativas são exibidas após cada ação para indicar o resultado (e.g., dano causado, cartas compradas).

Decisões e Ajustes

- **Compra de cartas**: Implementada para garantir que o jogador receba cartas de forma aleatória após cada ação.
- **Finalização do jogo**: Adicionada verificação explícita no loop principal para identificar e encerrar o jogo caso algum jogador atinja 0 ou menos pontos.

2. Conclusão

O trabalho proposto foi desenvolvido com sucesso, implementando um jogo de cartas funcional que utiliza conceitos importantes de programação orientada a objetos.

Durante o desenvolvimento, foram encontrados os seguintes desafios principais:

1. **Gerenciamento do estado das cartas**:
    - Inicialmente, houve dificuldades em diferenciar adequadamente os tipos de cartas e garantir que as ações fossem realizadas apenas com as cartas apropriadas.

2. **Controle do fluxo do jogo**:
    - A lógica para alternar turnos entre os jogadores e verificar as condições de término do jogo exigiu ajustes e testes.

3. **Manuseio do baralho**:
    - Foi necessário corrigir erros que faziam com que uma única carta fosse comprada repetidamente.

Esses desafios proporcionaram uma oportunidade valiosa de aprendizado, especialmente no que diz respeito à depuração e à implementação de testes incrementais. Como melhorias futuras, seria interessante adicionar uma interface gráfica para o jogo, tornando-o mais acessível e visualmente atraente. Seria possível também implementar novas funções como comprar cartas, criar cartas do tipo “magia”, etc;