Le but de ce moteur de jeux sera de gérer:

\*la logique du jeu

\*les règles

\*les mécanismes de jeu,

\*les interactions entre les cartes,

\*les phases de jeu

\*maintient l'état actuel du jeu ( points de vie, les effets en cours,) etc.

\*gérer les actions des joueurs,

\*les effets de cartes,

\*les attaques

\*etc

le moteur => catalogue( pour créer les decks de chaque joueur en fonction des cartes sélectionnées.)

Interfaces :

**CardInterface** : represente les propriétés et méthodes communes à toutes les cartes du jeu, telles que le coût, les dégâts, etc.

getCost(): int : Renvoie le coût de la carte.

getName(): string : Renvoie le nom de la carte.

getDamage(): int : Renvoie les dégâts infligés par la carte.

**ActionInterface** : pour représenter une action effectuée par un joueur. Les actions peuvent être jouer une carte, attaquer, défendre (jsp si ca existe)

playCard(CardInterface card, PlayerInterface targetPlayer): Permet à un joueur de jouer une carte sur un joueur cible.

attack(PlayerInterface attacker, PlayerInterface target): Permet à un joueur d'attaquer un autre joueur.

**PlayerInterface :**

getHealth(): int : Renvoie la santé actuelle du joueur.

getMana(): int : Renvoie la quantité de mana disponible pour le joueur.

drawCard(): CardInterface : Permet au joueur de piocher une carte.

takeDamage(int amount): Fait subir des dégâts au joueur.

playAction(ActionInterface action): Permet au joueur d'effectuer une action spécifique.

**DeckInterface :**

getCards(): List<CardInterface> : Renvoie la liste des cartes présentes dans le deck.

addCard(CardInterface card): Ajoute une carte au deck.

shuffle(): Mélange les cartes dans le deck.

**GameInterface :**

startGame(): Démarre une nouvelle partie.

endGame(PlayerInterface winner): Termine la partie et déclare un gagnant.

**1. Principe de Responsabilité Unique (Single Responsibility Principle - SRP) :**

* **Application :**
  + Chaque classe a une seule raison de changer.
  + Game est responsable du contrôle global du jeu.
  + Player gère les actions spécifiques d'un joueur.
  + Card représente une carte et gère ses effets.
  + Deck gère le deck d'un joueur.

**2. Principe Ouvert/Fermé (Open/Closed Principle - OCP) :**

* **Application :**
  + L’interface (Action) permett d'ajouter de nouvelles actions et effets sans modifier le code existant.
  + Les classes peuvent être étendues (par exemple, en ajoutant de nouveaux types de cartes) sans modification du code existant.

**3. Principe de Substitution de Liskov (Liskov Substitution Principle - LSP) :**

* **Application :**
  + Les sous-classes peuvent être utilisées de manière interchangeable avec leurs classes de base.
  + Les différentes cartes (par exemple, créatures, sorts) peuvent être utilisées où une carte est attendue, respectant ainsi le principe de substitution de Liskov.

**4. Principe de Ségrégation d'Interface (Interface Segregation Principle - ISP) :**

* **Application :**
  + L’interface (Action) sont spécifiques à un ensemble restreint de méthodes.
  + Cela évite aux classes d'implémenter des méthodes qui ne sont pas pertinentes pour elles.

**5. Principe d'Inversion de Dépendance (Dependency Inversion Principle - DIP) :**

* **Application :**
  + Les classes dépendent d'abstractions (par exemple, Action) plutôt que de détails concrets.
  + Les détails concrets (par exemple, les différentes implémentations d'effets) dépendent des abstractions.