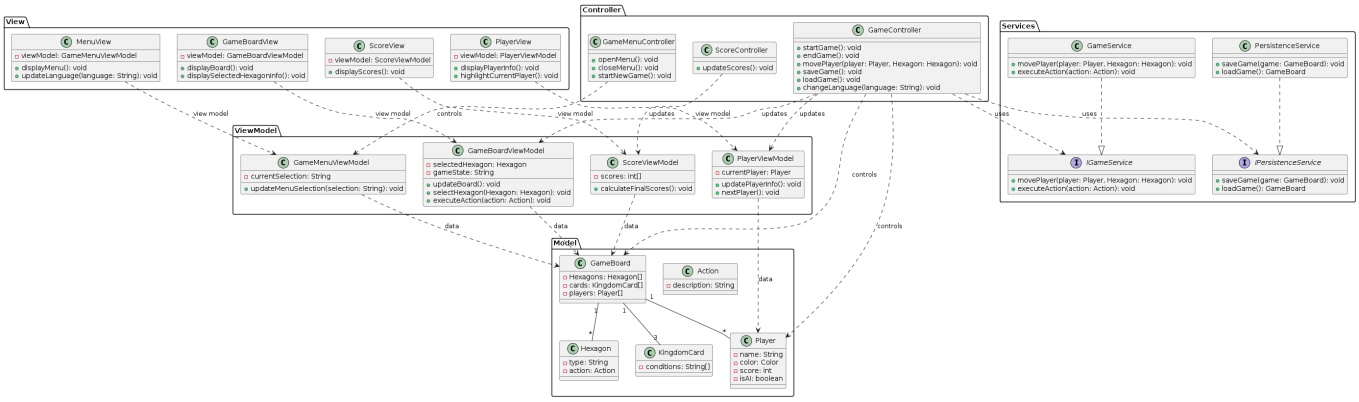
Architektur für Gruppe 16 - Undead Power Struggle (UPS)

UML-Klassendiagramm



Das Klassendiagramm stellt eine Architektur dar, das auf dem MVVMC (Model-View-ViewModel-Controller) Designmuster basiert.

Model

- **GameBoard**: Kern des Modells, das die Elemente des Spiels wie Haxagone (`Hexagons`), Karten (`KingdomCard`) und Spieler (`Player`) beinhaltet
  - **Hexagons**: Beinhaltet Eigenschaften wie Typ und eine zugehörige Aktion (`Action`)
  - **Player**: Beinhaltet Details zu den Spielern, einschließlich Name, Farbe, Punktestand und ob es sich um eine KI handelt
  - **KingdomCard**: Verwaltet spezifische Spielkartenbedingungen

View

- Verschiedene View-Klassen wie `MenuView`, `GameBoardView`, `PlayerView`, und `ScoreView` sind zuständig für die Darstellung der Benutzeroberfläche. Diese Komponenten sind eng mit den entsprechenden ViewModels verknüpft, von denen sie Daten beziehen und an die sie Benutzerinteraktionen weiterleiten

ViewModel

- **GameMenuViewModel**, **GameBoardViewModel**, **PlayerViewModel**, und **ScoreViewModel** dienen als Bindeglied zwischen den Views und dem Model. Jedes ViewModel verarbeitet die Logik, um auf Benutzeraktionen zu reagieren, Daten vom Model zu beziehen und die Views entsprechend zu aktualisieren

Controller

- **GameController**, **GameMenuController**, und **ScoreController** sind zentral für die Verarbeitung von Benutzeraktionen, das Initiieren von Spielen, das Beenden von Spielen, das Aktualisieren von Scores und andere spielbezogene Kontrollaufgaben. Diese Controller arbeiten eng mit den Services zusammen, um Aktionen auszuführen und den Spielzustand zu verwalten

Services

- **IGameService** und **IPersistenceService** bieten abstrahierte Interfaces für spielbezogene Dienstleistungen und Datenpersistenz
  - **GameService**: Implementiert das `IGameService` Interface, verwaltet Spielerbewegungen und führt Aktionen aus
  - **PersistenceService**: Implementiert das `IPersistenceService` Interface und ist verantwortlich für das Speichern und Laden des Spielzustands

Interaktionen

- Die `Controller` nutzen die `Services` zur Ausführung von Logiken und zur Datenpersistenz
- `ViewModels` reagieren auf Benutzereingaben durch die `Views` und aktualisieren die `Views` entsprechend, basierend auf Änderungen im `Model`
- Die `Views` sind direkt mit den `ViewModels` verbunden und stellen eine dynamische Darstellung des aktuellen Spielzustands bereit, wodurch eine reaktive Benutzeroberfläche ermöglicht wird

Designentscheidungen

- Die Verwendung von MVVMC ermöglicht eine klare Trennung der Zuständigkeiten, wodurch die Wartung und Erweiterbarkeit der Software verbessert wird
- Die Einbindung von Service-Interfaces fördert die lose Kopplung und erleichtert die Einbindung alternativer Implementierungen oder das Testen