Programmentwurf
Age Of Sigmar: What can I do?

Name: Maike Barth
Matrikelnummer: 8038835

Abgabedatum: 29.5.23]

Allgemeine Anmerkungen:

Gesamt-Punktzahl: 60P (zum Bestehen mit 4,0 werden 30P benötigt)

die Aufgabenbeschreibung (der blaue Text) und die mögliche Punktanzahl muss im Dokument erhalten bleiben

es darf nicht auf andere Kapitel als alleiniger Leistungsnachweis verwiesen werden (z.B. in der Form "XY wurde schon in Kapitel 2 behandelt, daher hier keine Ausführung")

alles muss in UTF-8 codiert sein (Text und Code)

das Dokument muss als PDF abgegeben werden

es gibt keine mündlichen Nebenabreden / Ausnahmen – alles muss so bearbeitet werden, wie es schriftlich gefordert ist

alles muss ins Repository (Code, Ausarbeitung und alles was damit zusammenhängt)

die Beispiele sollten wenn möglich vom aktuellen Stand genommen werden

- *finden sich dort keine entsprechenden Beispiele, dürfen auch ältere Commits unter Verweis auf den Commit verwendet werden*
- *Ausnahme: beim Kapitel "Refactoring" darf von vorne herein aus allen Ständen frei gewählt werden (mit Verweis auf den entsprechenden Commit)*

falls verlangte Negativ-Beispiele nicht vorhanden sind, müssen entsprechend mehr Positiv-Beispiele gebracht werden

- ***Achtung: werden im Code entsprechende Negativ-Beispiele gefunden, gibt es keine Punkte für die zusätzlichen Positiv-Beispiele sondern 0,5P Abzug für das fehlende Negativ-Beispiel***
- *Beispiel*
  - *"Nennen Sie jeweils eine Klasse, die das SRP einhält bzw. verletzt." (2P)*

Antwort: Es gibt keine Klasse, die SRP verletzt, daher hier 2 Klassen, die SRP einhalten: [Klasse 1], [Klasse 2]

Bewertung: falls im Code tatsächlich keine Klasse das SRP verletzt: 2P ODER falls im Code mind. eine Klasse SRP verletzt: 0,5P

verlangte Positiv-Beispiele müssen gebracht werden – im Zweifel müssen sie extra für die Lösung der Aufgabe implementiert werden

Code-Beispiel = Code in das Dokument kopieren (inkl. Syntax-Highlighting)

falls Bezug auf den Code genommen wird: entsprechende Code-Teile in das Dokument kopieren (inkl Syntax-Highlighting)

bei UML-Diagrammen immer die öffentlichen Methoden und Felder angegeben – private Methoden/Felder nur angeben, wenn sie zur Klärung beitragen

bei UML-Diagrammen immer unaufgefordert die zusammenspielenden Klassen ergänzen, falls diese Teil der Aufgabe sind

Klassennamen/Variablennamen/etc im Dokument so benennen, wie sie im Code benannt sind (z.B. im Dokument nicht anfangen, englische Klassennamen zu übersetzen)

die Aufgaben sind von vorne herein bekannt und müssen wie gefordert gelöst werden – z.B. ist es keine Lösung zu schreiben, dass es das nicht im Code gibt

- *Beispiel 1*
  - *Aufgabe: Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten*
  - *Antwort: Es wurden keine Fake/Mock-Objekte gebraucht.*
  - *Punkte: 0P*
- *Beispiel 2*
  - *Aufgabe: UML, Beschreibung und Begründung des Einsatzes eines Repositories*
  - *Antwort: Die Applikation enthält kein Repository*
  - *Punkte*

falls (was quasi nie vorkommt) die Fachlichkeit tatsächlich kein Repository hergibt: volle Punktzahl

falls die Fachlichkeit in irgendeiner Form ein Repository hergibt (auch wenn es nicht implementiert wurde): 0P

- *Beispiel 3*
  - *Aufgabe: UML von 2 implementierte unterschiedliche Entwurfsmuster aus der Vorlesung*
  - *Antwort: es wurden keine Entwurfsmuster gebraucht/implementiert*
  - *Punkt: 0P*

Kapitel 1: Einführung (4P)

Übersicht über die Applikation (1P)

[Was macht die Applikation? Wie funktioniert sie? Welches Problem löst sie/welchen Zweck hat sie?]

Wenn man AOS (Age of Sigmar) spielt, verliert man sehr schnell den Überblick. Um dem entgegen zu wirken kann man in diesem Programm seine Armee eingeben. Das Programm gibt dem Anwender dann eine Textdatei mit Überblick, welche Fähigkeiten von wem, in welcher Phase genutzt werden können.

Die Auswahl passiert über die Eingabe in die Kommandozeile.

Starten der Applikation (1P)

[Wie startet man die Applikation? Was für Voraussetzungen werden benötigt? Schritt-für-Schritt-Anleitung]

Man startet die Anwendung über die Entwicklungsumgebung, wie Visual Studio 2022.

- Als Erstes wird der Spielmodus gewählt.
- Dann die Allianz
- Dann die Fraktion
- Dann die Subfranktion
- Je nachdem, ob Path to Glory gespielt wird. Wählt man die Tenets und Fähigkeiten aus.
- Dann die Battalion
- Dann kann der General ausgewählt werden - Ist er ein Zauberer wird noch ein Zauber ausgewählt
- Dann ein Artefakt
- Dann eine Kommandofähigkeit
- Es können nun weitere Units hinzugefügt werden oder der Prozess kann beendet werden, mit “f”
- Das fertige Dokument kann im Debug Ordner gefunden werden.

Technischer Überblick (2P)

[Nennung und Erläuterung der Technologien (z.B. Java, MySQL, …), jeweils Begründung für den Einsatz der Technologien]

C# .Net 6 : Langfristige Cross-Plattform Unterstützung

JSON: menschenlesbares und persistentes Format, einfache Serialisierung und Deserialisierung

Docker?

Kapitel 2: Clean Architecture (8P)

Was ist Clean Architecture? (1P)

[allgemeine Beschreibung der Clean Architecture in eigenen Worten]

Clean Architecture ist ein Software-Designmuster, das saubere, testbare und wartbare Anwendungen durch Trennung von Geschäftslogik und Infrastruktur erreichen will. Es besteht aus Schichten mit spezifischen Verantwortlichkeiten, um Kohäsion und Entkopplung zu maximieren. Das Ziel ist, die Wartbarkeit und Testbarkeit zu verbessern und Abhängigkeiten zu reduzieren.

Analyse der Dependency Rule (3P)

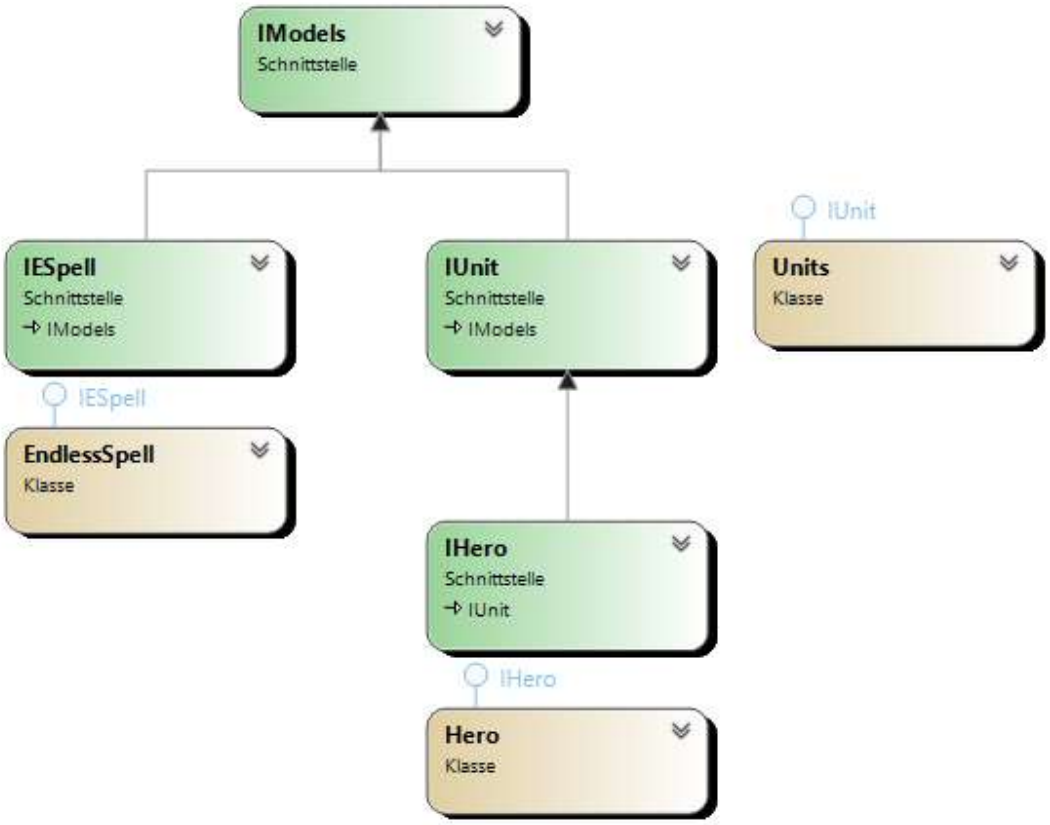
[1 Klasse, die die Dependency Rule einhält und 1 Klasse, die die Dependency Rule verletzt; jeweils UML (mind. die betreffende Klasse inkl. der Klassen, die von ihr abhängen bzw. von der sie abhängt) und Analyse der Abhängigkeiten in beide Richtungen (d.h., von wem hängt die Klasse ab und wer hängt von der Klasse ab) in Bezug auf die Dependency Rule]

Positiv-Beispiel: Dependency Rule

Da die dependency rule durch Interfaces implementiert werden:

Interface	Erbende Klassen
IModel	IUnit, IEndlessSpells
IUnit	Hero, Units

IUnit wird von CommandTrait verwendet, um einen Besitzer festzulegen.
IModel beinhaltet Methoden, die für alle Figuren benötigt werden. IUnit wiederum erweitert diese Methoden, somit wird ein hohes Level an Abstraktion geschaffen.



Negativ-Beispiel: Dependency Rule

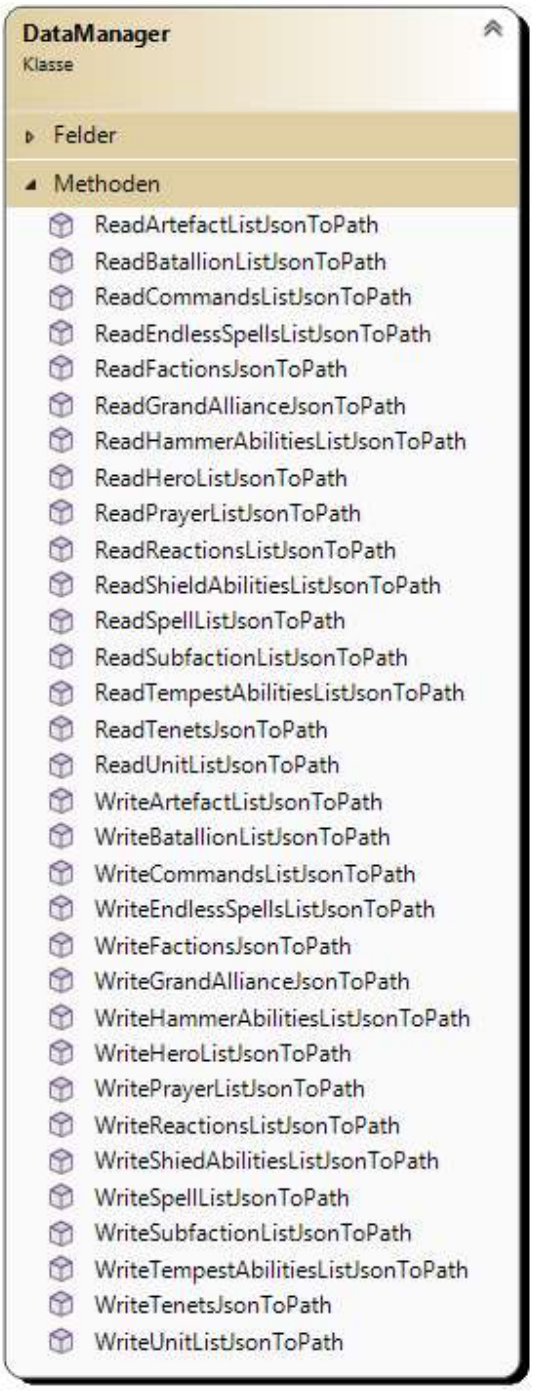
DataToWriteCollection: Es besteht eine sehr hohe Kopplung. Diese verletzt die dependency rule.

Analyse der Schichten (4P)

[jeweils 1 Klasse zu 2 unterschiedlichen Schichten der Clean-Architecture: jeweils UML (mind. betreffende Klasse und ggf. auch zusammenspielenden Klassen), Beschreibung der Aufgabe, Einordnung mit Begründung in die Clean-Architecture]

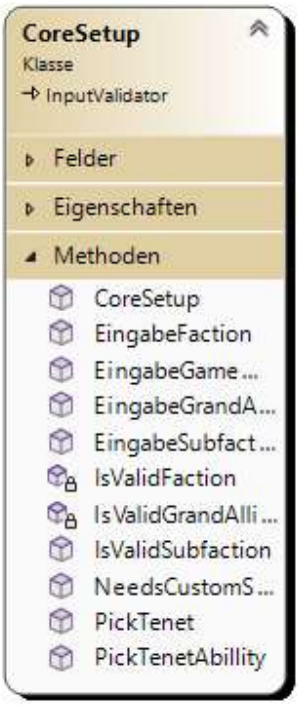
Schicht: Datenhaltung

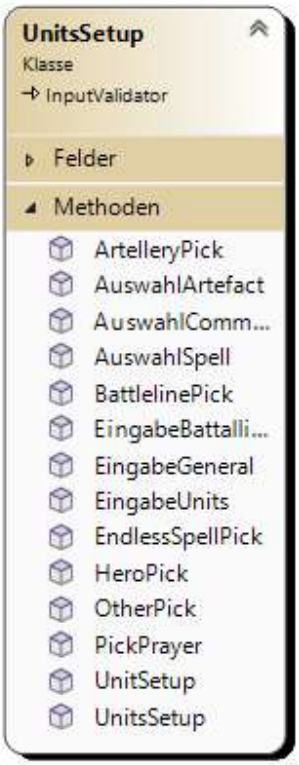
Der DataManager kann Objekte i JSON abspeichern und diese lesen. Das Lesen der Daten wird von der Zwischenschicht DataProvider ausgelöst.



Schicht: Logik

PlayerPicks wird genutzt um zur Laufzeit die Auswahl des Users zu speichern. Dafür übergeben die Methoden des CoreSetups und UnitSetus die Eingaben an die PlayerPick Instanz zur Speicherung.





Kapitel 3: SOLID (8P)

Analyse SRP (3P)

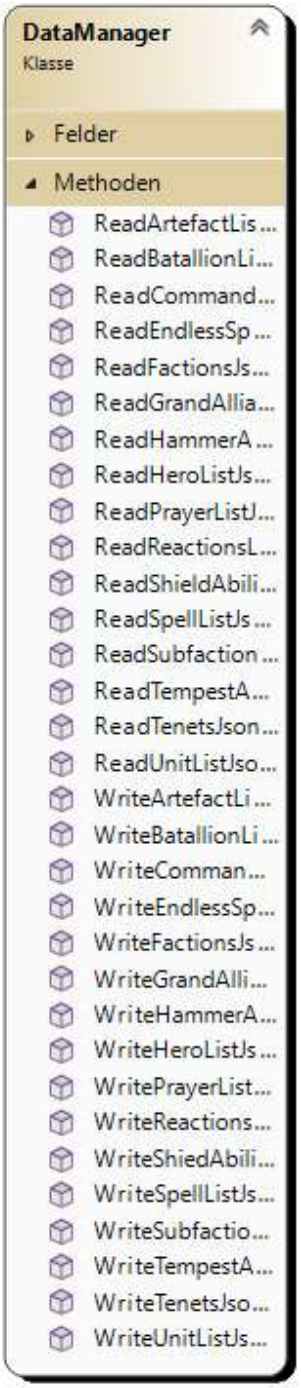
[jeweils eine Klasse als positives und negatives Beispiel für SRP; jeweils UML und Beschreibung der Aufgabe bzw. der Aufgaben und möglicher Lösungsweg des Negativ-Beispiels (inkl. UML)]

Positiv-Beispiel

ConsoleSpacer: Fügt Abstände und Zeilenumbrüche ein, um es lesbarer zu machen.

Negativ-Beispiel

DataManager: Speichert die Datenobjekte in die JSON Dateien und kann diese Dateien ebenfalls lesen.

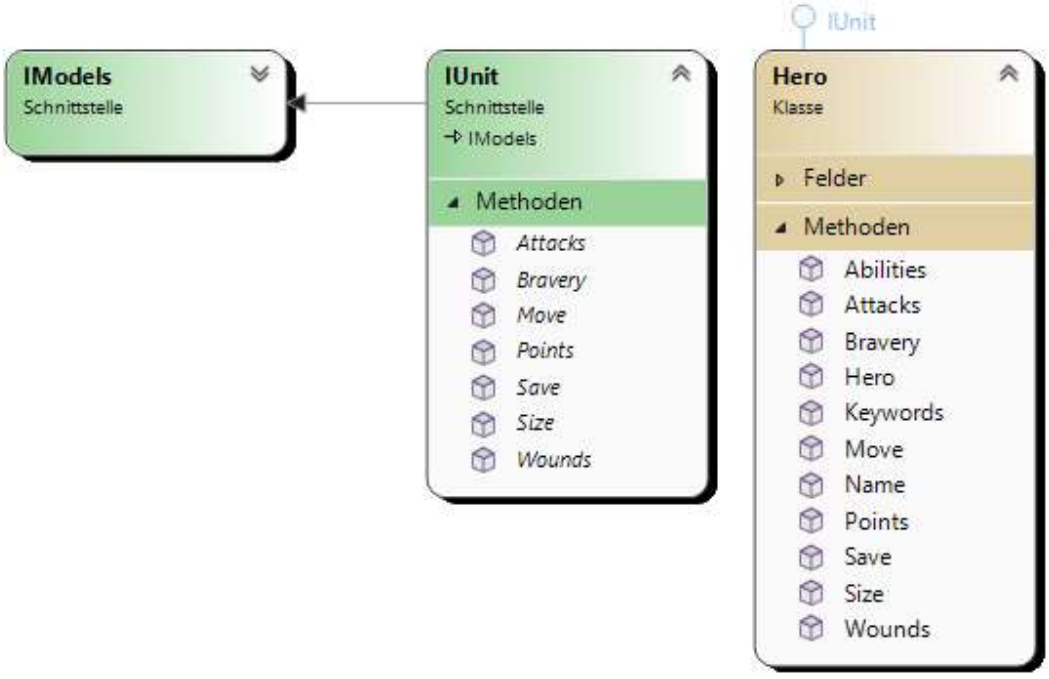


Analyse OCP (3P)

[jeweils eine Klasse als positives und negatives Beispiel für OCP; jeweils UML und Analyse mit Begründung, warum das OCP erfüllt/nicht erfüllt wurde – falls erfüllt: warum hier sinnvoll/welches Problem gab es? Falls nicht erfüllt: wie könnte man es lösen (inkl. UML)?]

Positiv-Beispiel

Hero: IUnit -> Der Hero implementiert das IUnit Interface. Soll eine Funktion hinzugefügt werden zu der Klasse Hero kann dies einfach gemacht werde, ohne dass IUnit oder IModels geändert werden müsste.



Negativ-Beispiel

Als Negativbeispiel kann angebracht werden, dass sobald eine Änderung in den Entity Klassen kommt, sowohl der DataProvider und DataToWriteCollection geändert werden muss.

Analyse [LSP/ISP/DIP] (2P)

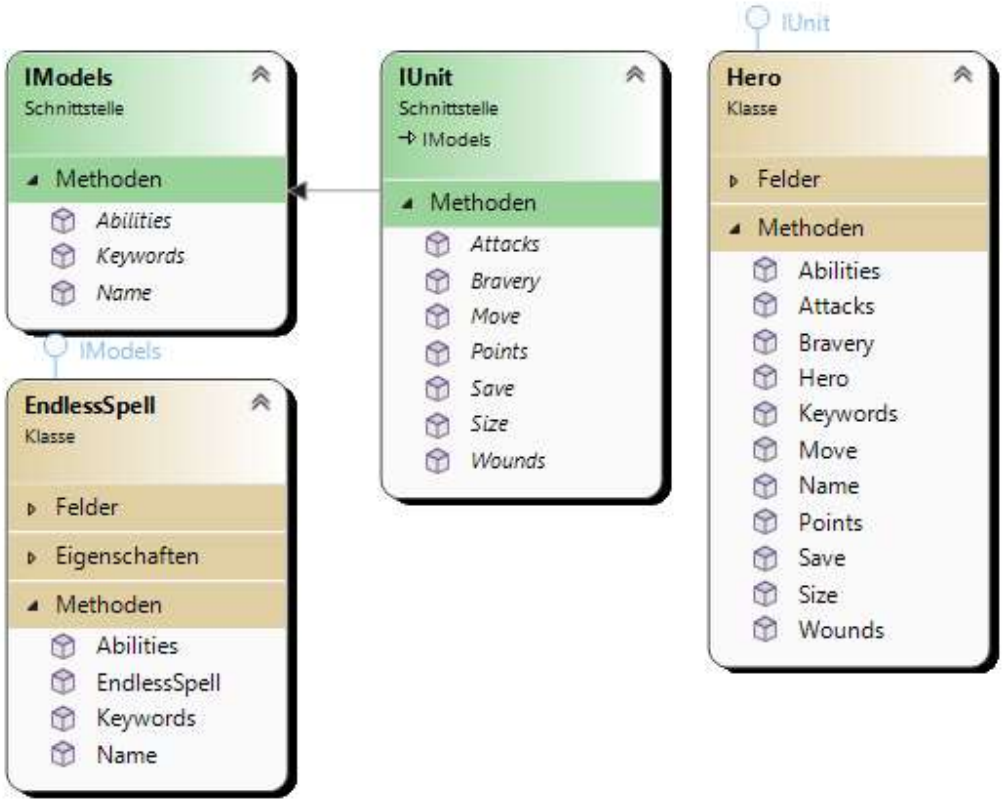
[jeweils eine Klasse als positives und negatives Beispiel für entweder LSP oder ISP oder DIP; jeweils UML und Begründung, warum hier das Prinzip erfüllt/nicht erfüllt wird; beim Negativ-Beispiel UML einer möglichen Lösung hinzufügen]

[Anm.: es darf nur ein Prinzip ausgewählt werden; es darf NICHT z.B. ein positives Beispiel für LSP und ein negatives Beispiel für ISP genommen werden]

ISP

Positiv-Beispiel

Hier ist zusehen, dass die Klasse EndessSpell nur die Methoden von IModel implementiert, da nicht mehr gebraucht wird. Jedoch hat die Klasse Hero eine kleine Schnittmenge mit der Klasse EndlessSpell. Deshalb wurde noch ein Zwischeninterface eingefügt, welches von IModel erbt und an Hero vererbt.



Negativ-Beispiel

Ich brauche ein interface das unötig viel deklariert und eine klasse die not implemented wirft-> meines Wissens nicht implementiert

Kapitel 4: Weitere Prinzipien (8P)

Analyse GRASP: Geringe Kopplung (3P)

*[eine **bis jetzt noch nicht behandelte** Klasse als positives Beispiel geringer Kopplung; UML mit zusammenspielenden Klassen, Aufgabenbeschreibung der Klasse und Begründung, warum hier eine geringe Kopplung vorliegt]*

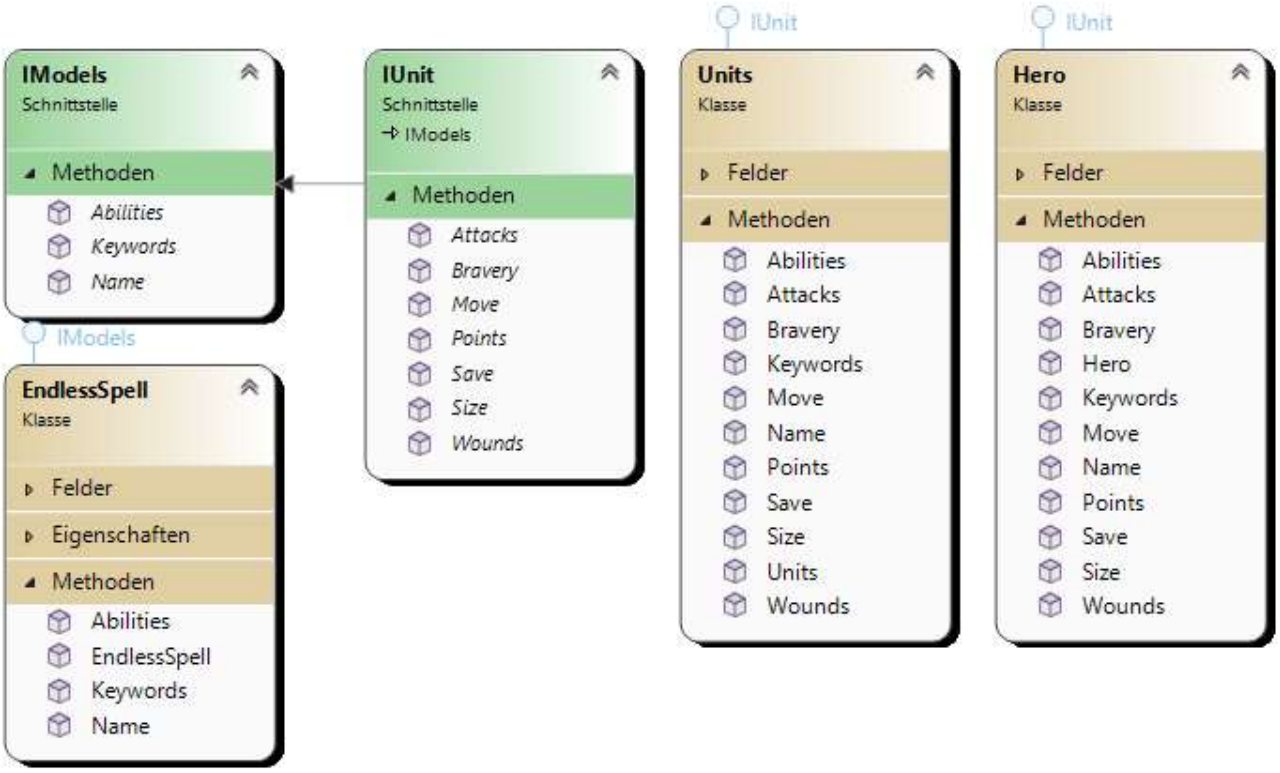
ConsolenReader: Wird verwendet, um die Eingabe in die Konsole zu lesen. Die Klasse selbst verwendet keine Klassen. Sie wird nur verwendet, daher hat sie eine sehr geringe Kopplung

Hierarchie	Wartbarkeitsindex	Zyklomatische Komplexität	Vererbungstiefe	Klassenkopplung
AOS_WCID (Debug)	84	460	2	49
Program	79	2	1	1
AOS_WCID	91	36	1	6
AOS_WCID.Data	78	109	1	28
AOS_WCID.Entities	91	132	1	10
AOS_WCID.Konsole	78	5	1	2
ConsolenReader	80	2	1	1
ConsoleSpacer	77	3	1	2
AOS_WCID.Konsole.Setup	60	101	2	30
AOS_WCID.Logic	77	75	1	28
WCID_UnitTests (Debug)	85	8	2	13

Analyse GRASP: [Polymorphismus/Pure Fabrication] (3P)

*[eine **bis jetzt noch nicht behandelte** Klasse als positives Beispiel geringer Kopplung; UML mit zusammenspielenden Klassen, Aufgabenbeschreibung der Klasse und Begründung, warum hier eine geringe Kopplung vorliegt]*

Polymorphismus: Hier werde gemeinsame Funktionen über Schnittstellen definiert und in den Klassen implementiert. Die Polymorphie besteht darin, dass die Implementierung jeweils unterschiedlich ist, jedoch die gleiche Funktion bereit stellt.



DRY (2P)

[ein Commit angeben, bei dem duplizierter Code/duplizierte Logik aufgelöst wurde; Code-Beispiele (vorher/nachher) einfügen; begründen und Auswirkung beschreiben – ggf. UML zum Verständnis ergänzen]

“Abstrahierung von validator” 20. März und “update custom faction” 21. März

Die Eingabevalidierung wurde an mehreren Stellen jeweils für die Methode implementiert. Dies wurde durch eine neue Klasse ersetzt, welche die Validierung übernimmt. Sie nimmt die zu validierende Eingabe und alle gültigen Eingabe entgegen und überprüft sie.

Vorherige Implementierung Beispiel:

```
218         -
219         -
220         -       }
221         -       public static bool IsValidPickTenet()
222         -       {
223         -           return false;
224         -       }
225         -       public static bool IsValidPickTenetAbility()
226         -       {
227         -           return false;
228     205         }
229     206
230     207     }
```

Folgende:

Methodenaufruf:

```
2 Verweise
public void PickTenet()
{
    StringBuilder chooseText = new StringBuilder();

    chooseText.AppendLine(PlayerPicks.Instance.Tenets.Count()==0 ? "What is your first Tenat?" : "What is your second Tenat?") ;

    int tenetID = -1;

    int tenetCount = initialStuff.TenetList.Count();
    bool isValidTenet = false;

    while (!isValidTenet)
    {
        Console.WriteLine(chooseText.ToString());
        for (int i = 0; i < tenetCount; i++)
        {
            Console.WriteLine($"{i} for {initialStuff.TenetList[i].Name}");
        }
        isValidTenet = IsValidInput(tenetCount, out tenetID);
    }
    PlayerPicks.Instance.Tenets.Add( initialStuff.TenetList[tenetID]);
    ConsoleSpacer.PrintSpacer();
}
```

Implementierung:

```
public bool IsValidInput(int maxInt, out int value)
{
    string input = consolenReader.GetLine();

    value = -1;

    if (string.IsNullOrEmpty(input) || !input.All(Char.IsDigit))
        return false;


    if (Int32.Parse(input) > maxInt)
        return false;

    value = Int32.Parse(input);
    return true;
}
```

Kapitel 5: Unit Tests (8P)

10 Unit Tests (2P)

[Nennung von 10 Unit-Tests und Beschreibung, was getestet wird]

Unit Test	Beschreibung
ConsolenReader - TestGetLine	Console lesen und Eingabe zurück geben ->early return
ConsolenReader - TestGetLineEmpty	Leere Console erkennen und zurück geben
DataManager-WriteGrandAllianceJsonToPath & ReadGrandAllianceJsonToPath	Es wird eine Liste von GrandAlliance-Objekten erstellt und in eine JSON-Datei geschrieben. Dann wird diese Datei mit der Methode ReadGrandAllianceJsonToPath eingelesen und überprüft, ob die erwarteten und tatsächlich gelesenen Objekte gleich sind.
InputValidator Valid Input	Dieser Test überprüft, ob ein True zurückkommt, wenn ein Wert innerhalb des

	Wertebereichs liegt.
InputValidator unvalid Input	Dieser Test überprüft, ob ein False zurückkommt, wenn ein Wert außerhalb des Wertebereiches eingegeben wird.
Grand Alliance selection TestValidSelection	Überprüft, ob die Auswahl der Grand Alliance richtig funktioniert, bei valider Eingabe
Grand Alliance selection TestUnvalidSelection	Test funktioniert nicht. Es kommt zu einem Timeout bei einer unvaliden Eingabe, da es in einer Endlosschleife ist. Dieser Timeout soll auch erwartet werden. Jedoch fehlt die richtige Methode dafür.
TestUnvalidAbilityFromValidPhase	Testet, ob ein false rauskommt beim Vergleichen einer unsinnigen Fähigkeit bei einer richtigen Phase
TestValidAbilityFromPhase	Testet ob, ein True rauskommt, wenn Keywords der Phase im Text sind
TestValidAbilityFromUnvalidPhase	Testet, ob False rauskommt, wenn Keywords im Text sind, aber keine valide Phase eingegeben wird

ATRIP: Automatic (1P)

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]

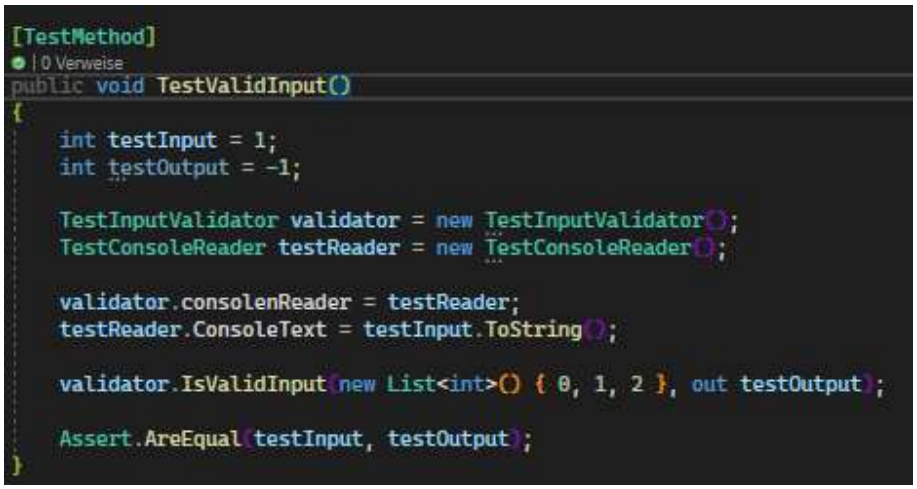
Innerhalb von Visual Studio können die Tests einfach ausgeführt werden, via einem Knopfdruck. Ebenso müssen keine Eingaben gemacht werden und es gibt nur bestanden oder fehlgeschlagen als Ergebnis.

ATRIP: Thorough (1P)

[Code Coverage im Projekt analysieren und begründen]

ATRIP: Professional (1P)

[Begründung/Erläuterung, wie 'Automatic' realisiert wurde]



```
[TestMethod]
0 Verweise
public void TestValidInput()
{
    int testInput = 1;
    int testOutput = -1;

    TestInputValidator validator = new TestInputValidator();
    TestConsoleReader testReader = new TestConsoleReader();

    validator.consolenReader = testReader;
    testReader.ConsoleText = testInput.ToString();

    validator.IsValidInput(new List<int>() { 0, 1, 2 }, out testOutput);

    Assert.AreEqual(testInput, testOutput);
}
```

Selbst im Test werden sprechende Namen verwendet und keine Abkürzungen. Gerade bei diesem Test ist es wichtig, dass getestet wird. Da es im Ablauf nach der Funktion zu Fehlern kommt.

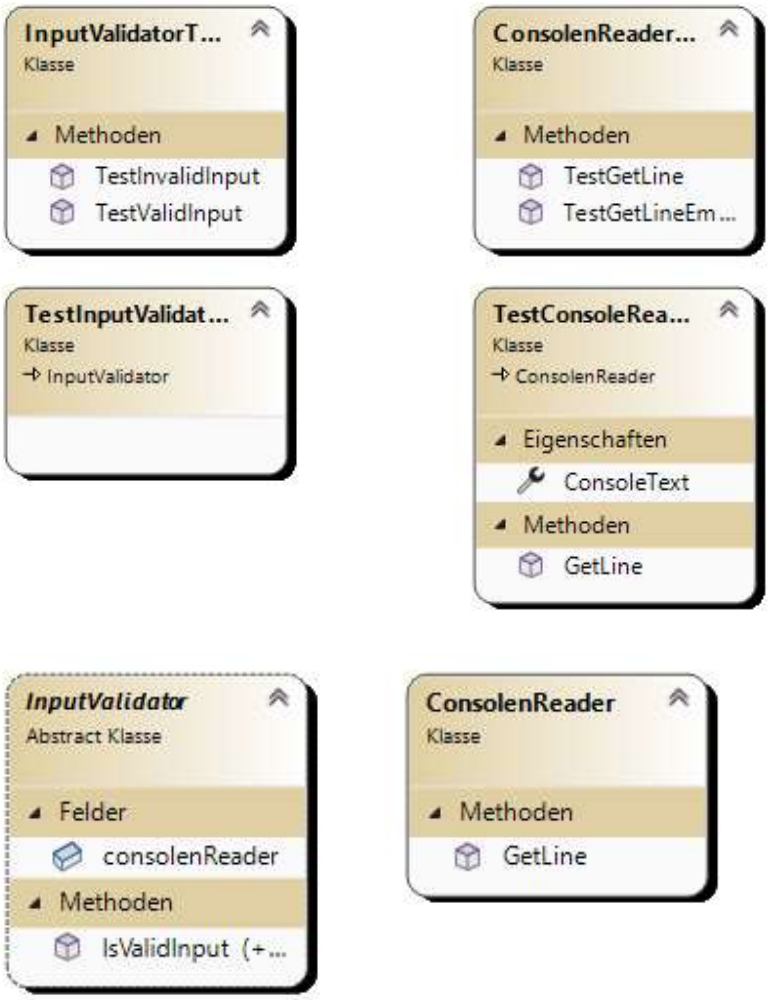
Fakes und Mocks (3P)

[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten (die Fake/Mocks sind ohne Dritthersteller-Bibliothek/Framework zu implementieren); zusätzlich jeweils UML Diagramm mit Beziehungen zwischen Mock, zu mockender Klasse und Aufrufer des Mocks]

TestInputValidator -> ruft die abstrakte Klasse auf.

TestConsoleReader -> erbt aus der Klasse ConsolenReader.

Beide werden benötigt, um künstliche Objekte zu generieren. Da beide Objekte nur während der Laufzeit innerhalb einer komplexen Struktur verwendet werden.



Kapitel 6: Domain Driven Design (8P)

Ubiquitous Language (2P)

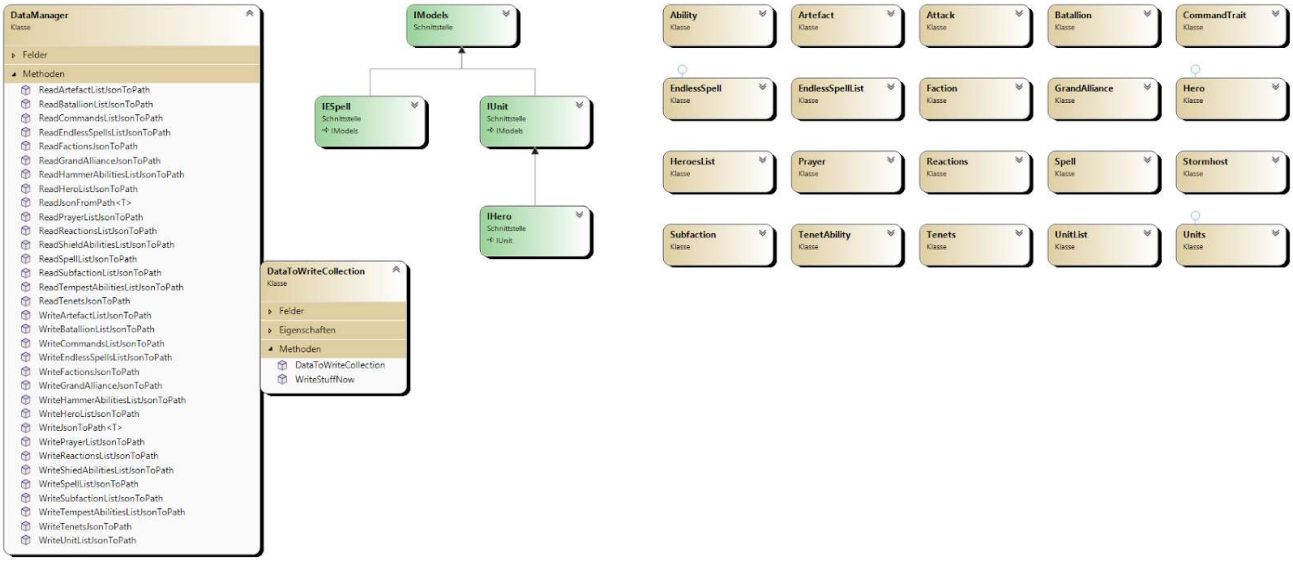
[Analyse und Begründung des Einsatzes von 2 Fake/Mock-Objekten (die Fake/Mocks sind ohne Dritthersteller-Bibliothek/Framework zu implementieren); zusätzlich jeweils UML Diagramm mit Beziehungen zwischen Mock, zu mockender Klasse und Aufrufer des Mocks]

Bezeichnung	Bedeutung	Begründung
InputValidator	Validiert Input	Sowohl Input als auch Validierung steht im Namen. Diese Funktion wird auch ausgeführt
PrintSpacer	Abstandshalter in Konsole schreibe	Drucke Abstandshalter (DE) Print Spacer (EN) 😺
UnitSetup/ CoreSetup	Auswahl der Aufstellung in Grundregeln	Hier wird jeweils die grundlegende Armeeauswahl aufgestellt und die Einheiten, mit denen gespielt werden soll. Diese Funktion

	und units	steht im Namen.
HeroPick/ BattlelinePick/ ArtelleryPick / ….	Auswahl von Held/ Front / Artellerie	Diese Methoden werden verwendet, um einen Helden/ etc. in auszuwählen. Diese Funktion steht im Namen.

Repositories (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Repositories; falls kein Repository vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist – NICHT, warum es nicht implementiert wurde]

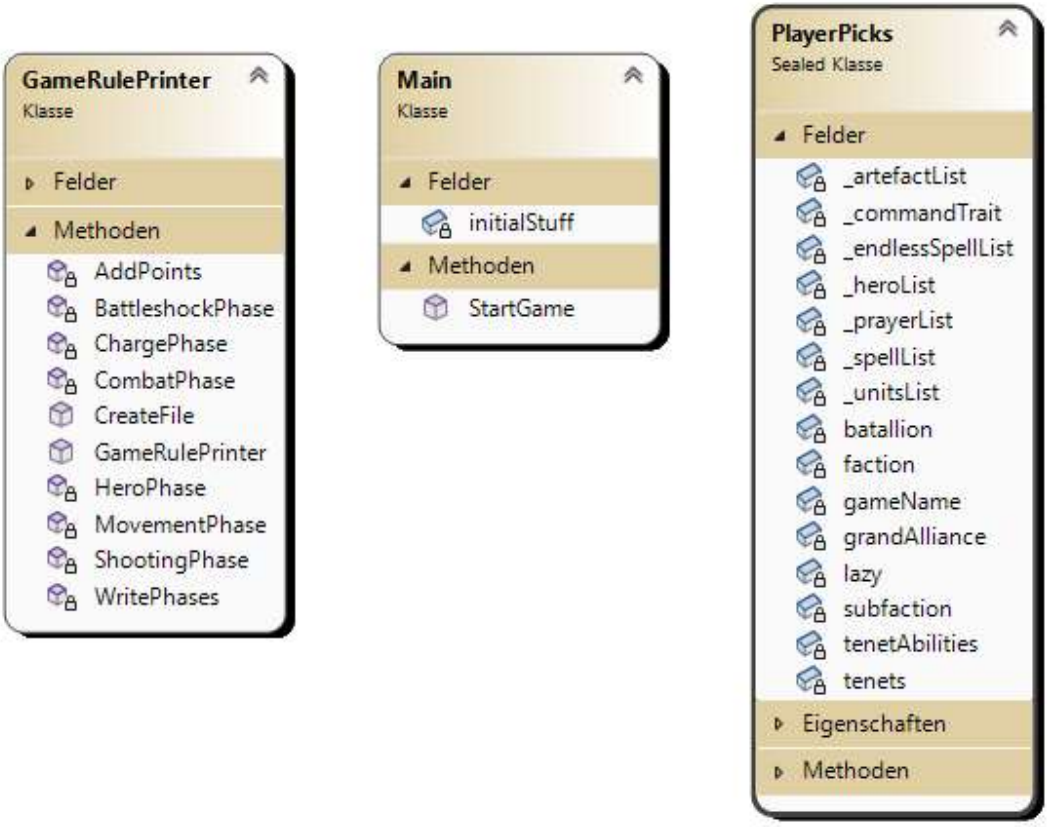


Hier wird der Datamanager als Art Repository verwendet. Er liest und schreibt Daten von oder in die JSON-Dateien. Es ist sinnvoll um alle Lese und Schreiboperationen an einem Ort zu haben.

Aggregates (1,5P)

[UML, Beschreibung und Begründung des Einsatzes eines Aggregates; falls kein Aggregate vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

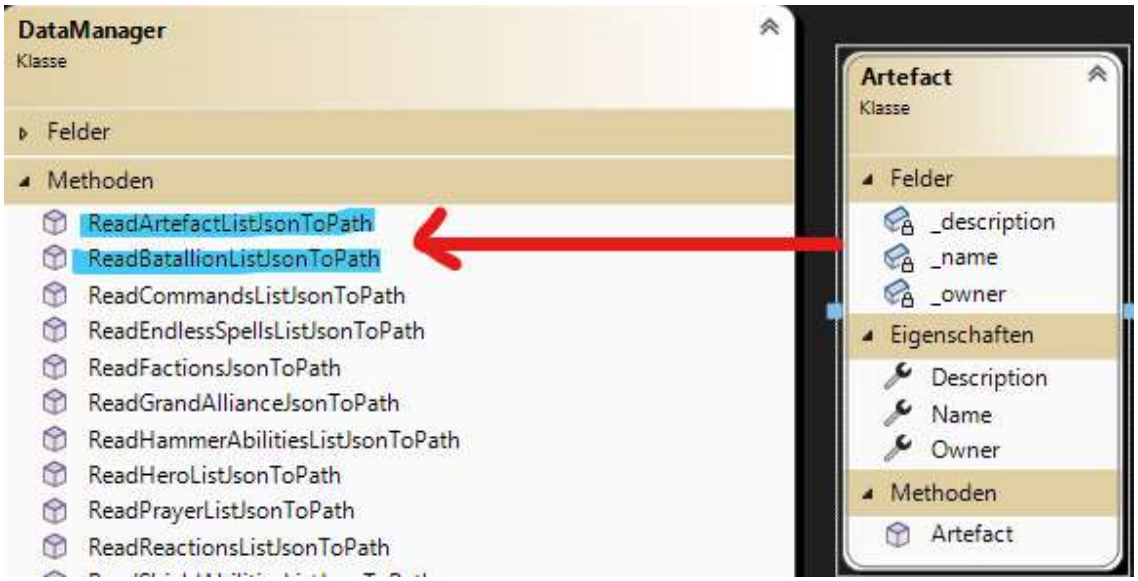
GameRule PRiter
Der GameRulePrinter ist hier ein Wurzelobjekt, welches verschiedene Objekte verwendet. Dies hat das Ziel, die Werte und Regeln in eine Textdatei zu drucken.



Entities (1,5P)

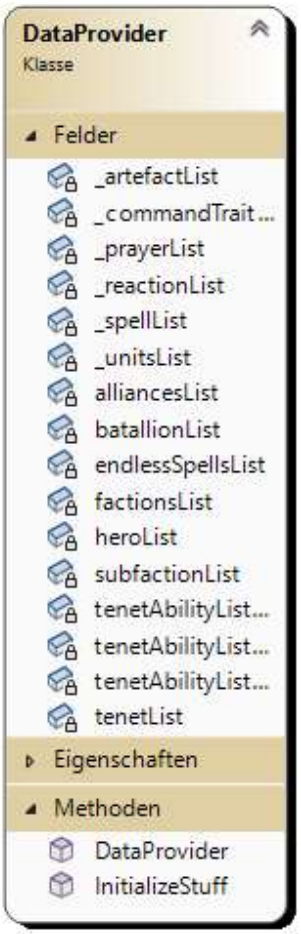
[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

Wird hier verwendet, da der JSON Serialisiere nur mit Entitätsklassen arbeitet.



Value Objects (1,5P)

Die Informationen zu den Figuren und Fähigkeiten / etc. verändern sich nicht. Deshalb stellt diese Klasse nur diese Informationen zur Verfügung. Diese können nicht verändert werden. (immutable)



Kapitel 7: Refactoring (8P)

Code Smells (2P)

[UML, Beschreibung und Begründung des Einsatzes eines Value Objects; falls kein Value Object vorhanden: ausführliche Begründung, warum es keines geben kann/hier nicht sinnvoll ist– NICHT, warum es nicht implementiert wurde]

[Long Class]

Die Klasse DataManager hat 333 Zeilen Code und übernimmt zwei Aufgaben. Die Lösung wäre hierfür, die Aufgabe des Lesens und des Schreibens in jeweils eine eigene Klasse zu exportieren.

```
public class DataWriter{
        public void IchSchreibeHero(){}
}

publlic class DataReader{
        public List<Hero> IchLeseHero(){
                return heroList;
        }
}
```

[Duplicate Code]

Die Klasse DataManager hat viel duplizierten Code. Der jeweils das Gleiche macht für verschiedene Objekte. Dies kann man Lösen in dem anonyme Typen verwendet werden.

```
public static void WriteJsonToPath<T>(List<T> entities, string path)
{
   var json = JsonSerializer.Serialize(entities);
   File.WriteAllText(path, json);
}
public static List<T> ReadJsonFromPath<T>(string path)
{
   using (StreamReader r = new StreamReader(path))
   {
      string json = r.ReadToEnd();
      return JsonSerializer.Deserialize<List<T>>(json);
   }
```

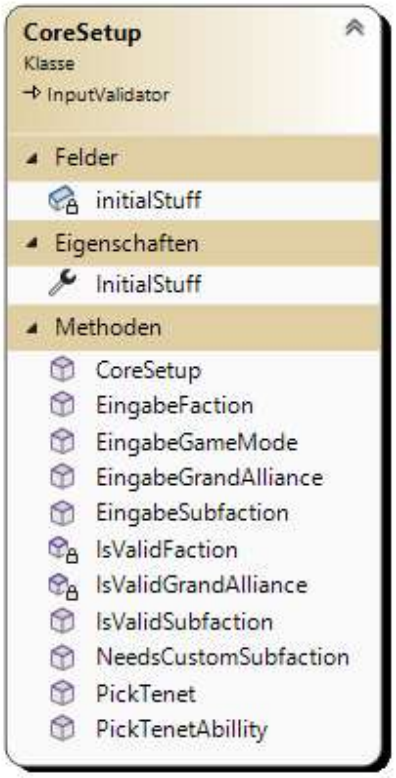
}

2 Refactorings (6P)

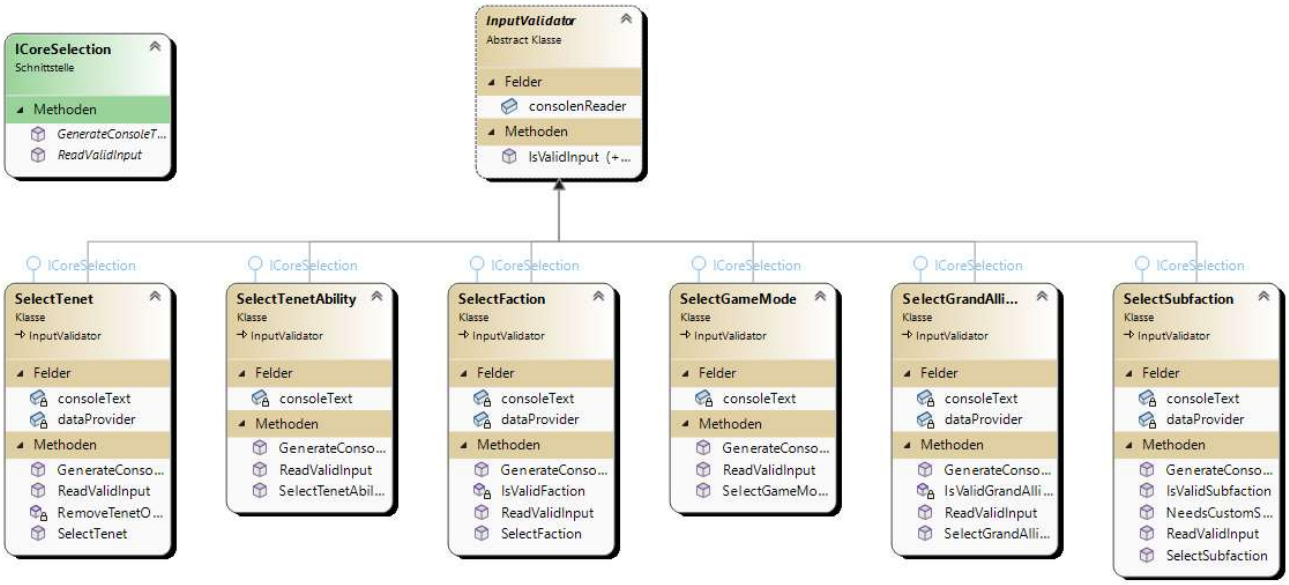
[2 unterschiedliche Refactorings aus der Vorlesung anwenden, begründen, sowie UML vorher/nachher liefern; jeweils auf die Commits verweisen]

Extract Method, Long class, god class: Hier wird beim ersten Refactoring die Klasse CoreSetup überarbeitet. Somit wird eine höhere Abstraktion erreicht. Dadurch wird der Code leserlicher und strukturierter. Refactoruing Godclass - 03b8b93

Vorher:

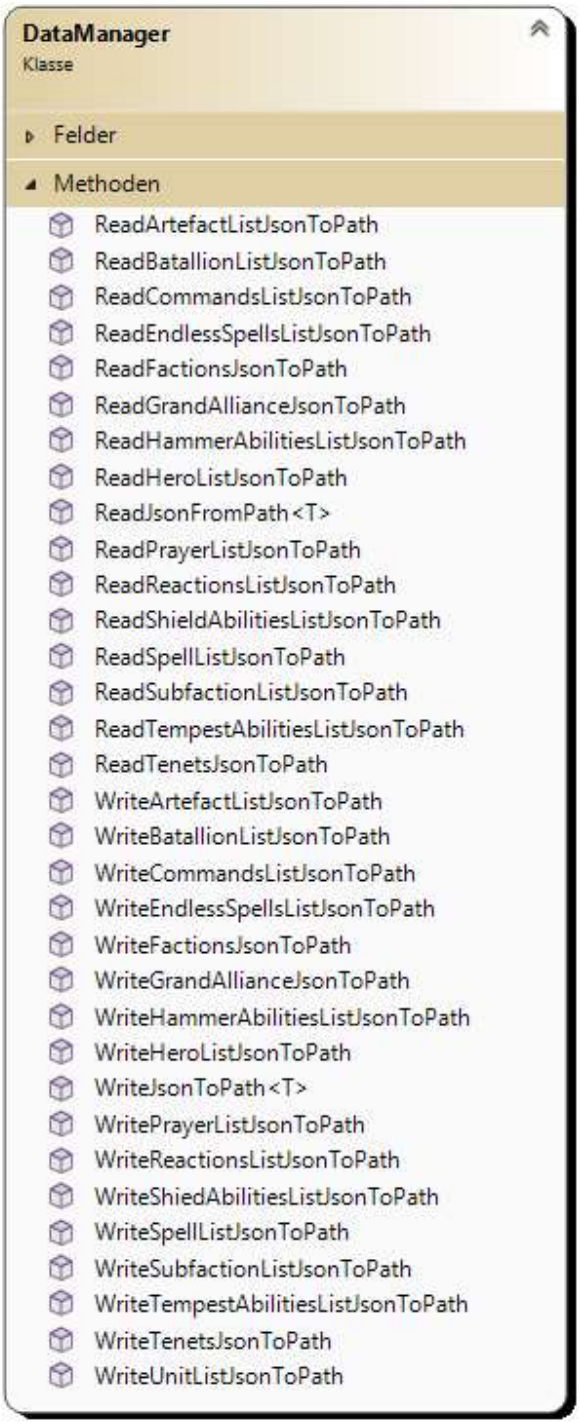


Nachher:

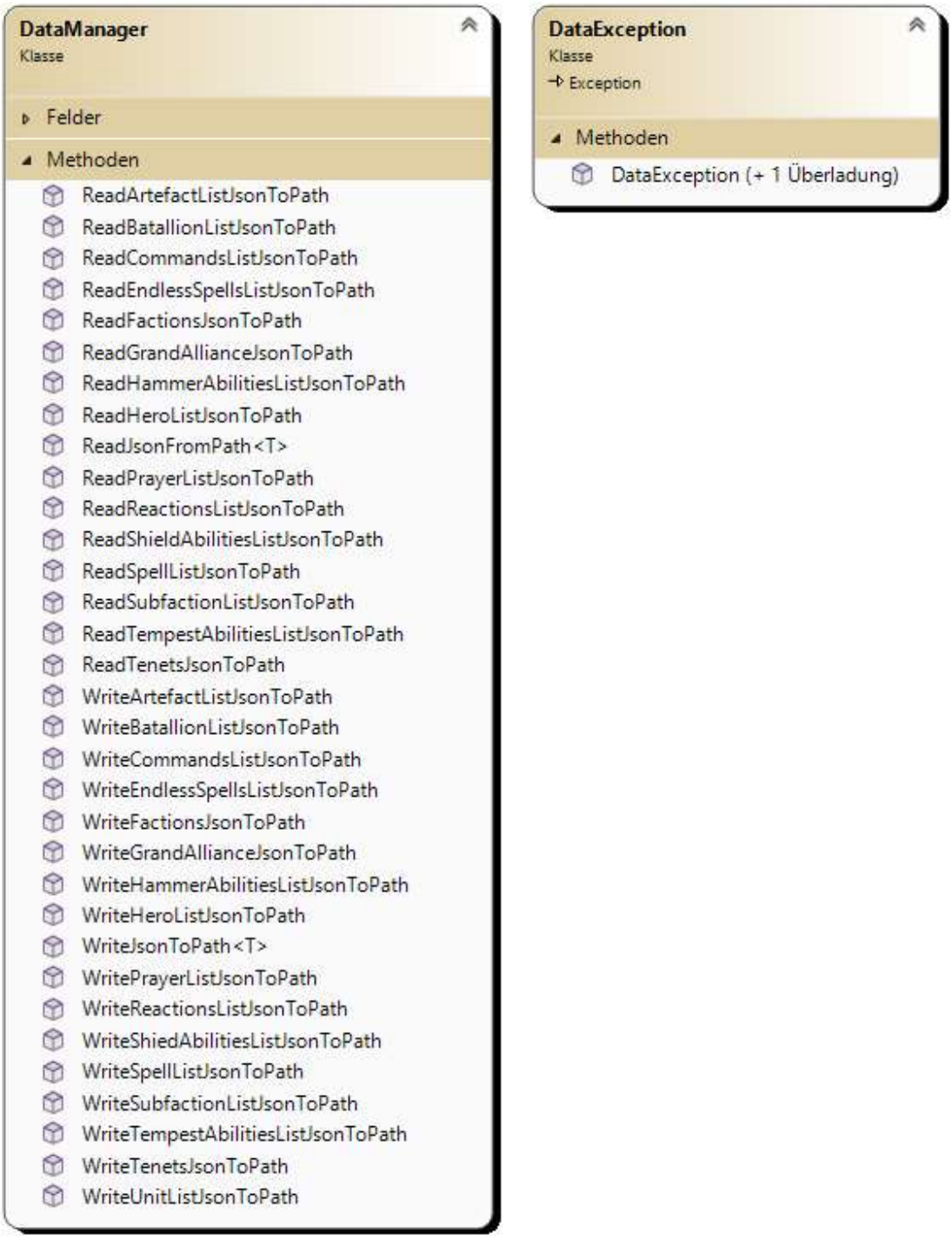


Replace error code with Exception: Beim zweiten Refactoring konnte null zurückgegeben werden. Das führte im späteren Verlauf zu Fehlern. Durch das Einfügen von Exceptions, wird ein beschreibender Text ausgegeben. Refactoring Exception JSON serialisiere - 56f8daf

Vorher:


DataManager
Klasse
Felder
Methoden
ReadArtefactListJsonToPath
ReadBatallionListJsonToPath
ReadCommandsListJsonToPath
ReadEndlessSpellsListJsonToPath
ReadFactionsJsonToPath
ReadGrandAllianceJsonToPath
ReadHammerAbilitiesListJsonToPath
ReadHeroListJsonToPath
ReadJsonFromPath<T>
ReadPrayerListJsonToPath
ReadReactionsListJsonToPath
ReadShieldAbilitiesListJsonToPath
ReadSpellListJsonToPath
ReadSubfactionListJsonToPath
ReadTempestAbilitiesListJsonToPath
ReadTenetsJsonToPath
WriteArtefactListJsonToPath
WriteBatallionListJsonToPath
WriteCommandsListJsonToPath
WriteEndlessSpellsListJsonToPath
WriteFactionsJsonToPath
WriteGrandAllianceJsonToPath
WriteHammerAbilitiesListJsonToPath
WriteHeroListJsonToPath
WriteJsonToPath<T>
WritePrayerListJsonToPath
WriteReactionsListJsonToPath
WriteShiedAbilitiesListJsonToPath
WriteSpellListJsonToPath
WriteSubfactionListJsonToPath
WriteTempestAbilitiesListJsonToPath
WriteTenetsJsonToPath
WriteUnitListJsonToPath

Nachher:

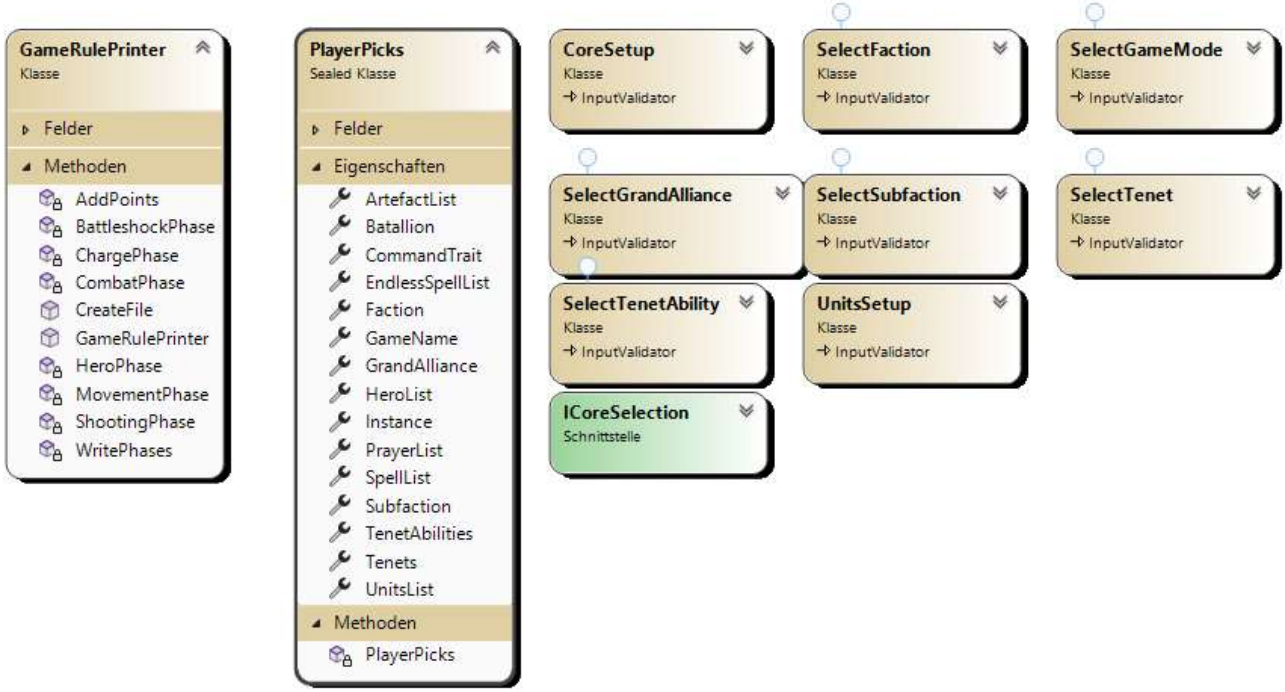


Kapitel 8: Entwurfsmuster (8P)

[2 unterschiedliche Entwurfsmuster aus der Vorlesung (oder nach Absprache auch andere) jeweils sinnvoll einsetzen, begründen und UML-Diagramm]

Entwurfsmuster: Singelton(4P)

PlayerPicks das alle Daten am richtigen Objekt gespeichert sind und das beim “ausdrucken” kein Fehler passiert.



Entwurfsmuster: Builder (4P)

data manager builder

data provider director