OOAD
Aufgabenblatt 05

Prof. Dr.-Ing. Michael Uelschen
Hochschule Osnabrück
Laborbereich Technische Informatik
m.uelschen@hs-osnabrueck.de

Entwurfsmuster: BCE, MVC, Creator, Dependency Injection

In dieser Aufgabe wird der Einsatz von Entwurfsmustern insbesondere in Hinblick auf die Bedienschnittstelle und des Erzeugens der Klassen in Hinblick auf eine konkrete C++-Implementierung erprobt und vertieft.

Aufgabe 9: Vom Diagramm zum Programm

In dieser und der folgenden Aufgabe werden Teile des Spiels („Malefiz“ oder „Fang den Hut“) in C++ umgesetzt werden. Im Folgenden sollen die beiden Anwendungsfällen (diese können bei Ihnen anders benannt sein) zum „Spiel vorbereiten“ und „Würfeln und Spielfigur setzen“ von Ihnen auf Basis des bisherigen Designmodells in C++ umgesetzt werden. Hierzu wird ein Implementierungsmodell zuvor von Ihnen entworfen.

Aufgabe 9a) Erweiterung um eine Bedienschnittstelle entsprechend BCE und MVC-Muster
In den bisherigen Aufgaben ist die Schnittstelle zwischen Aktor und dem System wenig oder nur vereinfachend betrachtet worden. Im ersten Teil der Aufgabe soll diese Schnittstelle entsprechend dem BCE respektive dem MVC-Muster modelliert werden.

1. Entwickeln Sie mit Visual Paradigm (VP) aus Ihrem bisherigen „Design Model“ ein „Implementation Model“ (siehe Hinweis 1 zur Modelltransformation unten) und ergänzen oder erweitern Sie hierzu die bisherige Bedienschnittstelle entsprechend dem MVC-Muster.
2. Entwickeln Sie zwei Ansichten (View-Klasse): eine „Laufbahn“-Ansicht, die die eigentlichen Spielfelder und -figuren visualisiert und eine „Startbereich“-Ansicht, in der die Spielfiguren zu Spielbeginn sich befinden (auch als Versteck bei „Fang den Hut“ bezeichnet) und geschlagene Spielfiguren zurückgesetzt werden. Ergänzen Sie die beiden o. g. View-Klassen um eine `show()`-Methode, die eine (grafische) Ausgabe der Ansicht ausführt. Eine konkrete Implementierung (oder Idee dazu) ist an dieser Stelle noch nicht erforderlich.
3. UML bietet mit Stereotypen die Möglichkeit, die vorhandenen Modellelemente (hier: Klasse) zu erweitern und diesen speziellen Eigenschaften zuzuordnen. Ergänzen Sie die vorhandenen Stereotypen in VP um *Model*, *View*, *Controller* (siehe Hinweis 2). Weisen Sie den entsprechenden Klassen in Ihrem Modell den entsprechenden Stereotypen zu. Achten Sie hierbei darauf, dass die entsprechend dem MVC-Muster vorgesehenen Beziehungen zwischen den Klassen existieren. Anmerkung: nicht alle Klassen in Ihrem Modell müssen mit Stereotypen versehen werden.

4. Zeigen Sie mit Hilfe eines Sequenzdiagrammes, wie der Anwendungsfall „Würfel und ziehen“ unter Verwendung des MVC-Musters abläuft. Sie können hierbei auf Ihre Ergebnisse von Aufgabe 7 zurückgreifen und verwenden dabei jedoch Ihr überarbeitetes Klassendiagramm.

Hinweis 1: Visual Paradigm bietet mit der Modelltransformation (Kontextmenü unter Utility) eine elegante Möglichkeit (1-Click), aus einem bisherigen Modell ein neues Modell zu erzeugen (ohne Copy & Paste). Der Vorteil dieses Ansatzes ist es, dass das Originalmodell (hier Design) erhalten bleibt und ein neues Modell (hier Implementation) erzeugt wird. Durch die Möglichkeit zur Nachverfolgung (Tracebility) lässt sich jeder Zeit zwischen den Modellen hin- und herspringen und die Änderungen verfolgen. Hier steht's: https://www.visual-paradigm.com/tutorials/developdomainandimplclassmodel.jsp.

Hinweis 2: Über das Kontextmenü einer Klasse können Sie dieser einen Stereotypen zuweisen. Allerdings müssen zuvor einmalig die Stereotypen für das MVC-Muster ergänzt werden. Über die Auswahl von Edit Stereotypes/Edit Stereotypes (2x!) können Sie für das Modellelement Class die neuen Stereotypen einfach ergänzen. Weiteres zum Arbeiten mit Stereotypen in VP finden Sie hier: https://www.visual-paradigm.com/support/documents/vpuserguide/1283/177/6563_applystereot.html und/oder hier https://www.visual-paradigm.com/support/documents/vpuserguide/1283/177/6562_configureste.html. Eine Zuordnung von Tag-Werten etc ist für die Aufgabenstellung nicht gefordert.

Aufgabe 9b) Festlegung der Konstruktion der Objekte und Beziehungen
Auf der Basis des jetzt festgelegten Modells ist festzulegen, durch welche Instanzen die einzelnen Objekte erzeugt und die Beziehungen (Dependency) gesetzt werden. Diese auf den ersten Blick einfache Aufgabe ist auf den zweiten Blick komplizierter. Betrachten Sie hierzu, welche Klasse hat genügend Informationen, um andere Objekte zu erzeugen und zu initialisieren? Beispiel: für eine Komposition, eine „Ganzes-Teil-Beziehung“, ist für das „Ganzes“ nicht bekannt, aus wie vielen „Teilen“ diese bestehen soll, wenn diese Information in einer dritten Klasse vorliegt.

Das Entwurfsmuster Dependency Injection beschreibt, wie Abhängigkeiten (damit sind im Wesentlichen die Beziehungen zwischen Klassen gemeint) erzeugt, d.h. (von außen) injiziert werden. Eine kurze Einführung finden Sie hier: https://www.youtube.com/watch?v=IKD2-MAkXyQ (der Transfer von PHP nach C++ bleibt Ihnen überlassen).

Verwenden Sie hierzu in Ihrem Klassenmodell die „Usage“ -Beziehung (gestrichelte, gerichtete Linie). In UML können nicht nur Klassen sondern auch Beziehungen mit einem Stereotyp versehen werden. Ändern Sie den Stereotypen von *<<use>>* nach *<<create>>* (Auswahl über das Kontextmenü der Beziehung in VP).

Aufgabe 10: Implementierung

In der letzten Aufgabe sollen jetzt einige Teile der Brettspiel-Anwendung in der Programmiersprache C++ umgesetzt werden. Beschränken Sie sich auf die beiden Anwendungsfälle „Spiel vorbereiten“ und „Würfeln und ziehen“ o. ä. Das Schlagen/Fangen von Figuren anderer Spieler ist nicht zwingend zu implementieren (ein Zug auf ein besetztes Spielfeld ist nicht möglich; für das Malefiz-Spiel realisieren Sie keine Sperrsteine: diese können übersprungen werden). Realisieren Sie Beziehungen mit smarten Zeigern soweit angebracht (siehe Aufgabe 1). **Vermeiden** Sie die Verwendung der new und

`delete`-Operatoren und verwenden Sie stattdessen konsequent `std::make_unique` oder `std::make_shared`.

Realisieren Sie eine sehr einfache ASCII-basierte Ausgabe der `show()`-Methoden für die beiden Ansichten-Klassen. Diese muss nicht optisch mit dem eigentlichen Brettspiel übereinstimmen. Beispielsweise können Sie bei „Fang den Hut“ den äußeren Laufkreis als Linie (mit periodischer Fortsetzung) visualisieren. Lassen Sie sich etwas einfallen! Beachten Sie, dass insgesamt 1+*n* Objekte der Ansichten-Klassen erzeugt werden: eine „Laufbahn“-Ansicht und n „Startbereich“-Ansichten, wobei *n* die Anzahl der Mitspieler ist.

Sie können entweder die Klassen komplett „per Hand“ programmieren oder mit Visual Paradigm Klassengerüste erstellen lassen. Wie in der Vorlesung gezeigt, funktioniert das aber u. U. nicht immer fehlerfrei. Hier steht’s: https://www.visual-paradigm.com/support/documents/vpuserguide/276/330/7364_instantgener.html

Dokumentieren (beispielsweise durch einen Kommentar/eine Notiz) Sie jene Stellen im Modell, die Sie nicht 1-zu-1 umsetzen konnten, sondern wo Sie während der Implementierung festgestellt haben, dass Ihr Modell nicht vollständig oder fehlerhaft war (es gibt idealerweise nur wenige oder keine Fehler in Ihrem Modell 🤓🥂).