UNIVERSITÄT OSNABRÜCK

Dr. Thomas Wiemann

12. Übungsblatt zur Vorlesung „Einführung in die Programmiersprache C++“

Wintersemester 2018 / 2019

Aufgabe 1: Template Meta Programming (40 Punkte)

Beantworten Sie die folgenden Fragen schriftlich:

1. Nehmen Sie an, sie haben eine Software programmiert, mit deren Hilfe sich ähnlich wie in unserer Software 2D Sprites rendern lassen. Anders als bei unserem Ansatz haben Sie sich allerdings dazu entschieden, keine eigene Template-Klasse zur Repräsentation von 2D-Koordinaten (analog zu unserer Klasse Vector) zu schreiben. Stattdessen sollen die Renderable-Klassen direkt über den Typ des Vektors vertemplatet werden, d.h., der Typ zur Vektor-Repräsentation ist ein Template-Parameter der Klasse `Renderable` und ihrer Unterklassen. Erläutern Sie vor- und Nachteile dieses Ansatzes.
2. Neben der Geometrie sollen die in 1.) genutzten Typen auch Felder mit Farbinformationen enthalten. Skizzieren Sie wie sich mit Hilfe von Traits feststellen lässt, ob der Typ, mit dem ein wie in 1) vertemplatetes Renderable instanziert wird, Komponenten zum Speichern der `r`, `g` und `b` Werte einer Farbe unterstützt. Vervollständigen Sie das in der Datei `Traits.cpp` gegebene Programmgerüst sinnvoll.
3. Betrachten Sie den Code in der Datei `Foo.cpp`. Erklären Sie die Funktionsweise und Intention in der Verwendung der dort implementierten Funktionalität.

Aufgabe 2: Refactoring der Interfaces / Mehrfachvererbung (30 Punkte)

In der Version der Software, die Sie in Ihrem Repository finden, sind in der abstrakten Klasse `Renderable3D` verschiedene Funktionalitäten vermischt, nämlich das Rendern von Objekten sowie deren Transformationen. Trennen Sie diese Funktionalitäten nun sauber, indem Sie ein Interface `Transformable` schreiben, in dem die Transformationsroutinen gekapselt sind.

Implementieren Sie nun eine Klasse `PhysicalObject` die von `Transformable` und `Renderable` erbt und zudem folgende Funktionalitäten zur Verfügung stellt:

- Einen `float`-Member `m_radius` der den Radius des Bounding-Balls um das Objekt repräsentiert
- Einen Pointer auf ein Renderable, der intern benutzt wird, um das Objekt darzustellen
- Die Implementierung der `render()`-Methode
- Die Methode `bool collision(PhysicalObject::Ptr& p)`, die prüft ob eine Kollision zwischen dem aktuellen Objekt und dem im Parameter `p` übergebenen Objekt stattgefunden hat. Prüfen Sie dazu, ob sich die Bounding-Balls (Kugeln mit Radius `m_radius` um die aktuelle Position) der Objekte schneiden.

In unserem Fall sollen die Klassen `Asteroid` und `Bullet` von `PhysicalObject` erben. Alle anderen verwendeten `Renderables` sollen bei Bedarf von `Renderable` und / oder `Transformable` erben.

Aufgabe 3: Implementierung von Collisionen zwischen Bullets und Asteroiden (30 Punkte)

Im Ordner `physics/` finden das Gerüst für eine `PhysicsEngine`, die alle aktiven Objekte verwaltet, zwischen denen Kollisionen auftreten können. Implementieren Sie in der Klasse die im Header geforderten Funktionalitäten. Wenn Sie nach der Implementierung die markierte Stelle in der Klasse `MainWindow` einkommentieren, sollen Sie in der Lage sein, mit abgefeuerten Bullets Asteroiden abzuschießen. Verwenden Sie diesmal keine Threads (die Synchronisation und Absicherung aller Objekte ist nicht trivial...).

Abgabe

Checken Sie ihre Abgabe bis Montag, 28.01.2019, 08:00 Uhr in den Ordner "uebung12" Ihres git-Repositories ein.