Übung zur Vorlesung Programmieren 2
SoSe 2022

Blatt 3

CMake, Klassen, Clean Code

Erfordert die VL bis Kapitel	Clean Code 1 › Basics
Abgabe bis	01. Juni 2022, 16:00 Uhr

Richtlinien für Abgaben:

Lösungen, die diese Richtlinien nicht erfüllen, werden mit *0 Punkten* gewertet!

Abgaben (Code/Text) erfolgen als Gruppe (nicht einzeln) über den Git-Server (↗) des Instituts.
Für die Abgabe gilt nicht die Commit-Zeit, sondern der Push-Zeitpunkt.
In Git ...

- wird für die Dateien jedes Blatts ein Ordner angelegt: *Gruppe_XXX/blatt1*, *Gruppe_XXX/blatt2*, ...
- haben temporäre/automatisch generierte Datien nichts zu suchen (→ *.gitignore*).
- wird nur auf den Server gepusht, wenn der Code im aktuellen Commit lauffähig ist.
  Nicht lauffähige lokale Commits (für Zwischenstände) sind OK und sogar empfohlen. Diese sollten allerdings mit `"WIP:"` (work in progress) am Anfang der Commit-Nachricht gekennzeichnet werden.

Es wird kein vollständiger Code-Style vorgegeben, aber ...

- innerhalb jeder Abgabe muss die Code-Formatierung konsistent sein.
- es wird sinnvoll und konsistent eingerückt.
- Variablen haben aussagekräftige Namen.
- Compiler-Warnungen im eigenen Code gelten als Fehler.

Allgemeine Infos:

Die Aufgaben und Erklärungen orientieren sich an (C++17) CMake-Projekten in der CLion IDE (↗), für Studenten kostenlos verfügbar für Windows, macOS und Linux.

Details zum C++-Standard (zu Syntax, Klassen, Funktionen, ...) findet ihr auf *cppreference.com*.
Eine Umfangreiche Tutorialreihe gibt es auf *learncpp.com*, und weitere Codebeispiele zu vielen Aspekten auf *alandefreitas.github.io/moderncpp*.

Richtlinien & Infos werden bei Bedarf angepasst/erweitert. Sie gelten entsprechend je Übungsblatt.

Hinweise zu Blatt 3:

Feiertag: Wegen dem Aufschub für Blatt 2 verschieben sich Abgabefrist und Präsentationen für Blatt 3 entsprechend um eine Woche (siehe Deadline).

Exkursionswoche: Da während der Exkursionswoche die Präsentationstermine ausfallen müssen, verschieben sich die Präsentationen von Blatt 3 um eine weitere Woche. Das betrifft jedoch nicht die Abgabefrist.

Tests: Da der vorgegebene Code, und die Aufgaben dieses Blatts etwas umfangreicher sind (Nutzt die längere Bearbeitungszeit!), ist es schwierig zu sehen, ob einzelne Aufgaben korrekt gelöst wurden. Dazu sind Tests für die aufeinander aufbauenden Aufgaben 2–4 vorhanden. Aktiviert diese in der Datei *test.cpp* jeweils nachdem eine Aufgabe erledigt ist.

Aufgabe 1

4 Punkte

> *Projektstruktur mit CMake*

Entpackt und öffnet das C++-Projekt im Verzeichnis *Repo/blatt3*.

Das Projekt besteht aus einer Anwendung "Battleship" und den Bibliotheken "GameObjects" und "Sea". Vervollständigt die Projektstruktur in den *CMakeLists.txt* Dateien, sodass das Projekt kompiliert und ausgeführt werden kann.

Abgabe/Präsentation:

1. in *Repo/blatt3/*
   in *.../src/*
2. in *.../src/app/*
3. in *.../src/libGameObjects/*
4. in *.../src/libSea/*

Aufgabe 2

6 Punkte

> *Eigene und geerbte Memberfunktionen nutzen*

Implementiert die beiden leeren Funktionen der Klasse `Sea::Object` in *libSea/Object.cpp*.

Die Objekte haben eine Position (`Coordinates`), eine beliebige Länge (`size`) und eine Ausrichtung (`orientation`). Wie in Abbildung 1 zu sehen, geben die Koordinaten dabei die Position von einem Ende des Objekts auf dem Spielfeld an.
Die Ausrichtung gibt an, in welche Richtung der Rest des Schiffs auf dem Spielfeld steht (X: in X-Richtung nach rechts, Y: in Y-Richtung nach unten).

Nutzt zur Implementierung die in `Coordinates` und `Object` vorgegebenen Funktionen.

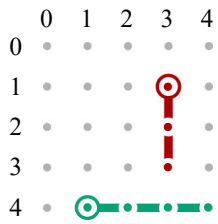

Abbildung 1:
Spielfeld mit Schiff A auf Position (1, 4) mit Länge 4 in X-Richtung,
und Schiff B auf Position (3, 1) mit Länge 3 in Y-Richtung.

Abgabe/Präsentation:

1. `Object::atCoordinates(..)` prüft, ob sich das Objekt auf den übergebenen Koordinaten befindet.
2. `Object::intersectsWith(..)` prüft, ob sich zwei Objekte überlappen.

Aufgabe 3

8 Punkte

> *Eine eigene Klasse mit Vererbung*

Implementiert die Klasse `Missile` in der Datei *libGameObjects/Missile.h* als Erweiterung der Klasse `Sea::Coordinates`, im Namensraum `GameObjects`.

Abgabe/Präsentation:

1. Klasse `Missile` inkl. Vererbung von `Sea::Coordinates`
2. Konstruktor `Missile::Missile(..)`
3. Funktion `Missile::hitSomething()`
4. Funktion `Missile::hasHitSomething()`

(Aktiviert die Zeile `#define CLASS_MISSILE_COMPLETE` in *Missile.h*, wenn die Aufgabe gelöst wurde.)

Aufgabe 4

3 Punkte

> *Nutzung der eigenen Klasse im restlichen Code*

Ergänzt die Funktionen der Klasse `Ship` in den Dateien *libGameObjects/Ship.h+.cpp*, die eure neue Klasse `Missile` benutzen.

Abgabe/Präsentation:

1. `checkAndUpdateIncomingMissile(..)`
2. `isDestroyed()`
3. `#define CLASS_MISSILE_COMPLETE` in *Missile.h*

(Aktiviert die Zeile `#define CLASS_SHIP_COMPLETE` in *Ship.h*, wenn die Aufgabe gelöst wurde.)

Aufgabe 5

3 Punkte

> *`static`-Variablen in Funktionen*

Ergänzt die Funktion `inputPlayerName(..)` in der Datei *app/init.cpp* so, dass eine statische Variable genutzt wird, die nur innerhalb der Funktion definiert ist, um bei jedem Aufruf den Namen des nächsten Spielers zu erfragen (`"Spieler 1:" ... "Spieler 2:" ...`).

Abgabe/Präsentation:

1. `inputPlayerName()`

Aufgabe 6

5 Punkte

> *Ergebnis/Status mittels* `enum class`

Definiert eine `enum class AddShipResult` für die Fälle "erfolgreich hinzugefügt", "außerhalb des Spielfelds" und "Überschneidung mit anderem Schiff" in der Datei *PlayerSea.h*.

Ergänzt die Funktion `addShip(..)` in den Dateien *libGameObjects/PlayerSea.h+.cpp* so, dass sie den entsprechenden Status als `AddShipResult` zurückgibt.

Erweitert die Funktion `initializeShip(..)` in der Datei *app/init.cpp* so, dass je nach erhaltenem Status eine passende Meldung ausgegeben wird. (Keine Ausgabe, wenn das Schiff hinzugefügt werden konnte.)

Abgabe/Präsentation:

1. `enum class AddShipResult`
2. `addShip(..)`
3. `initializeShip(..)`

Aufgabe 7

5 Punkte

> *Clean Code*

Bringt die `#include-` und `using`-Anweisungen in den Dateien *app/init.cpp* und *app/game.cpp* in eine sinnvoll Ordnung.

Erklärt, warum die `using`-Anweisung in der Datei *libGameObjects/Ship.cpp* unnötig ist.

Zerlege die Funktion `gameLoop(..)` in sinnvolle, kürzere Teilfunktionen.

Abgabe/Präsentation:

1. Includes in *app/init.cpp*
2. Erklärung als Kommentar in *libGameObjects/Ship.cpp*
3. Includes in *app/game.cpp*
4. `gameLoop(..)`

Aufgabe 8 (Bonus)

(3 Punkte)

Zeigt vor jeder Eingabe von Raketen-Koordinaten das Spielfeld mit den eigenen bisher abgefeuerten Raketen an. Gebt das Spielfeld dazu als Textzeilen (verschiedene Zeichen für Wasser, Raketentreffer und Wasser-Treffer) auf der Konsole aus.

Gesamt: 34 Punkte