Übung zur Vorlesung Programmieren 2
SoSe 2024

Blatt 4

Smart Pointer, Polymorphismus, Exceptions, Operatorüberladung

Erfordert die VL bis Kapitel	Operatorüberladung
Abgabe bis	28. Mai 2024, 16:00 Uhr

Richtlinien für Abgaben:

Lösungen, die diese Richtlinien nicht erfüllen, werden mit *0 Punkten* gewertet!
Abgegeben werden alle vorgegebenen, oder für die Aufgaben neu erstellten, Code/Text-Dateien.
Abgaben erfolgen über den Git-Server (↗) des Instituts.
Jeder in der Gruppe ist für die (pünktliche) Abgabe mitverantwortlich. Dafür gilt nicht die Commit-Zeit, sondern der Push-Zeitpunkt auf dem Server.

In Git ...
- wird für jedes Übungsblatt ein neuer Ordner angelegt: *Gruppe_XxYZ/blatt1*, */blatt2*, */blatt3*, ...
- dürfen keine temporären/automatisch generierten Dateien hinzugefügt werden (→ *.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, die in der Konsole ausgegeben werden, 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.

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

Hinweise zu Blatt 4:

Exkursionswoche: Wegen dem Aufschub für Blatt 3 verschieben sich Abgabefrist und Präsentationen für Blatt 4 entsprechend um eine Woche (siehe Deadline).

Good News: In diesem und einigen folgenden Aufgabenblättern wird das Schiffe-versenken-Projekt aus der letzten Übung weiterentwickelt. Der Großteil des Codes sollte euch also schon bekannt sein.

Aufgabe 1

8 Punkte

Smart Pointer
Bei der Übergabe von Objekten als Funktionsparameter, die als Smart Pointer erstellt wurden, sollten als Faustregel folgende Fragen bedacht werden:

A Wird das Objekt in der Funktion verändert?

B Verändert die Funktion die Lebenszeit des Objekts (wird zB. ein weiterer `shared_ptr` erzeugt)?

A	B	Funktionsparameter
Ja	Ja	`std::shared_ptr<Objekt> obj`
Ja	Nein	`Objekt & obj`
Nein	Ja	`std::shared_ptr<const Objekt> obj`
Nein	Nein	`Objekt const & obj`

Der folgende Artikel beschreibt das noch einmal etwas anschaulicher: `https://medium.com/@vgasparyan1995/pass-by-value-vs-pass-by-reference-to-const-c-f8944171e3ce`

Es werden in jeder Runde Raketen abgefeuert. Dazu wird zunächst entsprechend der eingegebenen Koordinaten eine Instanz der Klasse `Missile` erstellt. Diese wird anschließend an verschiedene andere Funktionen/Objekte übergeben, dort verarbeitet und zum Teil auch gespeichert. Dabei werden "unnötige" Kopien dieser Instanz erzeugt, zu sehen durch Terminal-Ausgaben der jeweiligen (Copy-)Konstruktoren.

Nutzt Smart Pointer (`std::shared_ptr`) und Referenzen für die Verwaltung, Weitergabe und Verarbeitung von `Missile`-Instanzen im gesamten Projekt, sodass keine unnötigen Kopien mehr erstellt werden.

Abgabe/Präsentation:

Je Spielrunde, bzw. je abgefeuerter Rakete ...

1. wird nur einmal `"(Create Missile | ...)"` ausgegeben (zweimal nach Bearbeitung von Aufgabe 3).
2. wird nie `"(Copy Missile)"` ausgegeben.

Aufgabe 2

8 Punkte

Exceptions
Bekommt eine Funktion ungültige Eingaben, ist es oft unpraktikabel oder sogar unmöglich einen entsprechenden "Fehlerwert" statt des eigentlichen Ergebnisses der Funktion zurückzugeben.
Dann kann mittels `throw ...` ein Fehler "geworfen" werden, der mittels `catch (...) {...}` später wieder "aufgefangen" und behandelt werden kann, zB. von der aufrufenden Funktion.

Entfernt den Rückgabewert der Funktion `addShip(..)` und ändert die Funktion selbst, sowie die aufrufende Funktion `initializeShip(..)` in *init.cpp* so, dass die beiden möglichen Fehlerfälle (*outsideSeaBounds* und *overlapOtherShip*) mithilfe von Exceptions behandelt werden. Erstellt dazu eine eigene Exception-Klasse, die von `std::exception` abgeleitet ist. Die entsprechenden Terminal-Ausgaben für die Spieler sollen unverändert bleiben.

Abgabe/Präsentation:

1. Eigene Exception-Klasse(n) definiert.
2. `addShip(..)` wirft unterscheidbare Exceptions für die beiden Fehlerfälle.
3. Exceptions werden korrekt auffangen und behandelt.

Aufgabe 3

9 Punkte

> *Polymorphismus*
> Das Konzept des Polymorphismus erlaubt es spezialisierte Funktionalitäten zu implementieren, und diese an anderer Stelle zu verwenden ohne dort überhaupt von diesen Spezialisierungen wissen zu müssen. Dazu kann eine Klasse die Funktionen ihrer Basis-Klasse (von der geerbt wurde) mit eigenen Implementierungen "überschreiben".
> *Tipp:* Innerhalb überschreibender (`override`) Funktionen einer abgeleiteten Kind-Klasse kann die Funktions-Implementierung der Basis-Klasse mithilfe des *Scope-Resolution-Operators* aufgerufen werden, z.B.:
>
> ```
> void ChildClass::test() { ... BaseClass::test(); ... }
> ```
>
> So können die existierenden Funktionen aus der Basis-Klasse genutzt werden, statt ähnlichen/identischen Code in der erbenden Klasse erneut zu implementieren.

Um etwas mehr Spannung in das Spiel zu bekommen, soll in jeder zweiten Runde eine "Taumelrakete" abgefeuert werden. Dazu ist in *TumblingMissile.h/cpp* bereits ein Großteil der Klasse `TumblingMissile` vorgegeben, die von `Missile` erbt. Diese berechnet beim Raketenstart eine Abweichung (`offsetX` und `offsetY`) in eine zufällige Richtung, die zu den eingegebenen Zielkoordinaten addiert werden soll.

Vervollständigt die Klasse `TumblingMissile` durch Überladung der virtuellen Funktionen `getX()` und `getY()` (aus der Klasse `Coordinates`) so, dass `TumblingMissile`-Referenzen im restlichen Code wie `Missile`-Referenzen verwendet werden können, aber automatisch die Abweichung der Koordinaten berücksichtigt wird. Überladet auch `toString()`, sodass sowohl die Zielkoordinaten ohne, als auch mit Abweichung ausgegeben werden.

Vervollständigt anschließend die Funktion `sendTumblingMissileTo(..)` in *game.cpp* analog zu der vorgegebenen Funktion `sendRegularMissileTo(..)`, nur dass eben jeweils eine `TumblingMissile` erzeugt wird.

Erweitert zuletzt die Funktion `gameTurn(..)`, sodass abhängig von der aktuellen Runde (Parameter `round`) immer abwechselnd normale und taumelnde Raketen (`sendTumblingMissileTo(..)`) gestartet werden.

Abgabe/Präsentation:

1. Überladung der drei virtuellen Funktionen von `Coordinates` in `TumblingMissile`.
2. Implementierung von `sendTumblingMissileTo(..)`
3. Erweiterung von `gameTurn(..)`

Aufgabe 4

6 Punkte

> *Operatorüberladung*

Implementiert zwei Operatoren für die Klasse `Coordinates` in *Coordinates.h/cpp*.

Zunächst soll die Funktion `samePositionAs(Coordinates ..)` durch den `operator==(..)` ersetzt werden, sodass Koordinaten im restlichen Code mittels `coordinatesA == coordinatesB` verglichen werden, statt mit `coordinatesA.samePositionAs(coordinatesB)`.

Mit dem zweiten `operator<<(..)` soll es möglich sein Koordinaten direkt per `cout << coordinatesA` auf dem Terminal auszugeben, ohne zusätzlich `toString()` aufrufen zu müssen. Dazu muss dieser eine `std::ostream`-Referenz und die auszugebenden `Coordinates` annehmen, und die `std::ostream`-Referenz zurückliefern.
(`std::ostream` kennen Sie bereits indirekt — es ist der Typ von `std::cout`)

Achtung: Damit der Operator überall verwendet werden kann, muss er im globalen Namespace, also außerhalb von `GameObjects`, deklariert werden.

Abgabe/Präsentation:

1. Implementierung von `operator==(..)` als Ersatz für `samePositionAs(Coordinates ..)`.
2. Implementierung von `operator<<(..)` zur direkten Ausgabe von Koordinaten nach `std::cout`.
3. Nutzung der beiden Operatoren im restlichen Code.

Aufgabe 5 (Bonus)

(3 Punkte)

Denkt euch einen weiteren Raketentypen aus, und implementiert diesen analog zur Klasse `TumblingMissile`. In *game.cpp* (`gameTurn(..)` und `send_____MissileTo(..)`) sollen dann drei statt zwei verschiedene Raketentypen abwechselnd genutzt werden.

Gesamt: 31 Punkte