# Projekt im Kurs “Entwicklung von Multimediasystemen”

Lukas Hodel Robert Kasseck Richard Remus

29. Januar 2015

# Contents

# 1 Idee

# 2 Anforderungen

# 3 Einarbeitung

## 3.1 GameDesignPatters

## 3.2 Qt/Qml

## 3.3 Projektstrukturen

# 4 Spiellogik

Ein sehr kritischer Teil des Projektes besteht in der inneren Spiellogik. Sie bestimmt die Regeln, den Ablauf und auch jeden möglichen Zustand des Spiels. Ihre Aufgabe ist es auch, den Zustand der einzelnen Akteure des Spiels konsistent zu halten. Um die Umsetzung der Logik unabhängig von Qt zu halten, haben wir uns entschieden die Spiellogik völlig von den anderen Bestandteilen des Spiels zu entkoppeln.

## 4.1 GameActor

Die Klasse GameActor repräsentiert jedes denkbare Objekt, das Teil des Spielgeschehens ist. Dazu zählen die Raumschiffe der Spieler, Planeten, Sonnen, Weltraumschrott, Asteroiden aber auch jegliche Geschosse der Waffen über die die Schiffe verfügen. Dies lässt sich gut am in *Figure 1* abgebildeten Diagramm ablesen, in welchem wir die Vererbung der GameActors in unserem Spiel darstellen.

Die wichtigsten Eigenschaften eines GameActors sind seine Position, Geschwindigkeit, Beschleunigung, Masse, Lebenspunkte, Gravitationskraft sowie die Reichweite, über welche seine Gravitation andere GameActors beeinflussen kann. Seine Positionierung und Bewegung werden mit elementarer Vektorrechnung realisiert. Wann immer eine Kraft auf einen GamActor wirken soll, wird die Methode **applyForce()** mit einem entsprechenden Kraftvektor verwendet. Die angewendete Kraft wirkt sich zunächst nur auf seinen Beschleunigungsvektor addiert, erst beim Aktualisieren des GameActors über **update()** wird der Beschleunigungsvektor mit dem Geschwindigkeitsvektor verrechnet und mit diesem dann schließlich die neue Position bestimmt. Falls GameActors miteinander Kollidieren wird dies mithilfe der **collisionDetection()** aus der

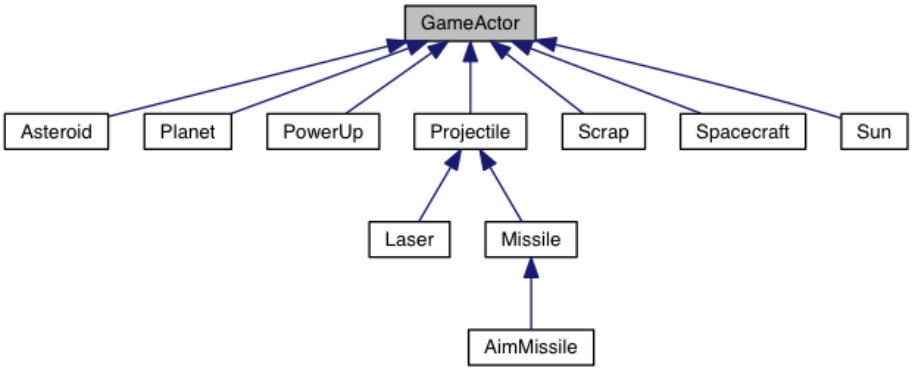


Figure 1: GameActor Klassenhierarchie

**Physics**-Bibliothek ermittelt. Durch Implementation der virtuellen Methode **handleCollision()** kann festgelegt werden, wie der jeweilige GameActor mit einem Zusammenstoß umgehen soll. Sollte er schaden nehmen, kann dies mit **dealDamage()** realisiert werden, diese Methode bestimmt dabei auch, ob der zugewiesene Schaden die maximale Zahl von Lebenspunkten überschreitet und aktualisiert das Feld **killed** dementsprechend. **addHealth()** agiert in analog entgegengesetzter Weise zu **dealDamage()** wobei wir jedoch davon ausgehen, dass ein getöteter oder zerstörter GameActor nicht "wiederbelebt" werden sollte. Dies sollte, falls nötig über einen anderen Weg geregelt werden. Zusätzlich definieren wir mit einer Lebenspunktzahl von -1 einen GameActor, der nicht durch das Zuteilen von Schaden durch **dealDamage()** sterben kann. Falls ein GameActor durch seinen Tod einen Effekt auf das Spielgeschehen haben, zum Beispiel eine Explosion oder das setzen eines PowerUps, kann dies in der virtuellen Methode **handleKill()** definiert werden.

## 4.2 Projectile

Mit Projectile sollen alle möglichen Arten von Geschossen realisiert werden. Das könnten Laser und Raketen, aber auch Bomben, Minen oder andere Dinge sein, welche durch eine Waffe im Spielfeld platziert werden können. Projectile erbt von GameActor, bringt aber ein paar Erweiterungen. Zum einen kann es den Punktestand von befreundeten Raumschiffen beeinflussen, wobei "befreundet" bedeutet, dass dieses Projectile sie nicht in negativer weise beeinflussen kann. Zum anderen kann für jedes Projectile eine maximale Lebensdauer (time to live), gemessen in Update-Zyklen. Dadurch soll simuliert werden, dass die meisten Geschosse keine unendliche Reichweite haben, eine Rakete könnte zum Beispiel eine begrenzte Menge Treibstoff haben. Erreicht ein Projectile eine **timeToLive** von null, wird es als **killed** behandelt. In analoger Weise zu den Lebenspunkten des GameActors definieren wir, dass ein Projectile unbegrenzte Lebenszeit hat,

wenn es eine **timeToLive** von -1 hat. Darüber hinaus gehorcht das Projectile allen durch GameActor definierten Regeln, sofern diese nicht überschrieben werden.