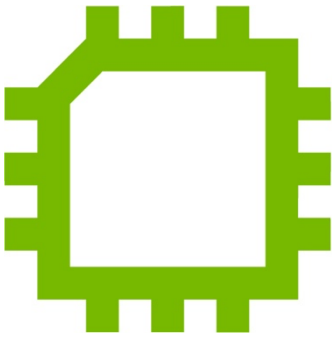
Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Datum: 26.06.18



Softwareprojekt: Asteroids in c++ mit der Library von SFML

Sören Größer - s0554662

Ahmed Jenbaz – s0564915

Fabian Rod – s0546837

Hochschule für Technik und Wirtschaft Berlin
University of Applied Sciences

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Inhaltsverzeichnis

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Einleitung

Das folgende Projekt beschäftig sich mit dem Re-Engineering eines Programms, geschrieben in der Objekt orientierten Programmiersprache C++, sowie der Implementierung neuer Funktionen. Zum Verständnis und arbeitsteiligen Entwicklung der Implementierung wurden verschiedene Methoden genutzt um den reinen Code abstrakt darzustellen und zu begreifen. Wir entschieden uns ein Klassendiagramm, ein Zustandsdiagramm, sowie das Domain Modell & die UseCases darzustellen um dies zu gewährleisten.
Dafür genutzte Programme:

- Sourcetrail um einfacher die Beziehungen der Klassen untereinander zu erfassen.
- Yakindu zur Darstellung des Zustandsdiagramms
- Umlet zur Dastellung des Klassendiagramms, der UseCases und dem Domain Modell

Abschließend konnten die geforderten Veränderungen implementiert werden. Dazu gehörte:

- Beenden des Programms durch ‚ESC'
- Mehrere Gegner fliegen eine Kreisformation
- Es gibt Power Up, die einem Leben oder andere Vorteile geben

Am Ende dieser Belegarbeit resümieren wir, welche Sachen wir dadurch lernen konnten und legen den Source Code im Anhang bei.

Beschreibung

Asteroids in C++ mit SFML Library

Asteroids ist ein interessantes Spiel bzw. Konsole Programm, geschrieben in der Objekt orientierten Programmiersprache C++ mit der Nutzung der SFML Library. Es ist ein solides Beispiel für die Nutzung von C++ im Bereich der Spiele Entwicklung. Es demonstriert die grundlegenden Kommandos, Syntaxe, Funktionen, Strukturen, sowie das Konzept von der Dateiverwaltung in C++. Es nutzt die Multimedia Bibliothek von SFML, wodurch Computer Ressourcen, audiovisuelle Ausgaben und Peripherie Abfragen einfacher zu verarbeiten und implementieren sind. Hinzu ist SFML „multi-platform" und „multi-language", was System freies arbeiten und kompilieren, als auch die Konvertierung in andere Programmiersprachen zulässt.

Weitere Informationen zu SFML – „Simple and Fast Multimedia Library" findet man unter www.sfml-dev.org. Für dieses Projekt wurde die SFML Version 2.3.2 genutzt.

Die Idee beruht auf das klassische Arkade-Game von 1979 von Atari. Das Spiel startet in einer TableTop Ansicht eines Weltraums mit 5 Asteroiden und einem Raumschiff. Der Spieler beginnt mit einem eigenen Raumschiff und befindet sich im „Spawning-Modus", wodurch er unverwundbar ist. Nach einer definierten Zeit verlässt er diesen. Als Spieler kann man sich nun über die WASD-Tasten bewegen und über die Leertaste schießen. Selber wird man vom gegnerischen Raumschiff beschossen. Ziel des Spiels ist, weder von den Schüssen des

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Gegner, noch von den Asteroiden getroffen zu werden. Gewonnen hat man, wenn alles abgeschossen wurde und man steigt ein Spielelevel auf. Die Asteroiden werden durch Beschuss kleiner und vermehren sich in 3 abnehmenden Stufen. Die Punkte dafür werden in der Highscore gespeichert, sofern man am Ende des Spiels den alten Wert übersteigt. Das Spiel endet erst, wenn man alle seine Leben verloren hat. Mit zunehmenden Level erhöht sich die Anzahl der Asteroiden.

Asteroids Abstrakt

Im Folgenden sind die Header Dateien aufgelistet mit grober Zuordnung.

Headerdatei	Kurze Beschreibung
Game.h	Spielstatus aktualisieren; GameObjects ans laufende Spiel hängen
GameObjects.h	Mainclass für alle Objekte mit grundlegenden Methoden; beinhaltet hinzu alle erbenden Objektdeklarationen (Bullet;Asteroid;Enemy;Player;
GameState.h	Deklarationen für den Highscore
HighScore.h	Deklarationen zum Einlesen der Highscore und Abspeichern in eine Datei
StarParticleSysrem.h	Deklaration zur Erstellung der Hintergrundsterne

Des Weiteren die C++ Dateien.

Cpp Datei	Weitere Dateien:
Asteroid.cpp	Zur sofortigen Implementierung mit VisualStudio liegt eine Engine.vcxproj mit bei. Die Engine.Cpp ist die „Main“ Quelldatei. In der ScoreSheet.bin wird der Highscore gespeichert.
Bullet.cpp	
Enemy.cpp	
Engine.cpp	
Game.cpp	
GameObjects.cpp	
GameState.cpp	
HighScore.cpp	
Player.cpp	
StarParticleSystem.cpp	

Die SFML Version 2.3.2 wurde genutzt und liegt im entsprechenden Ordner zum Linken bei.

Menü/Optionen

Es gibt für dieses Spiel kein Menü. Es lässt sich lediglich zwischen zwei Programm Modi schalten. Mit TAB stoppt man das Spiel und lässt sich die aktuelle Highscore anzeigen. Wird TAB nicht gedrückt, läuft das Spiel bis man 4 Leben verloren hat und kann anschließend nur über X oder ALT+F4 geschlossen werden.

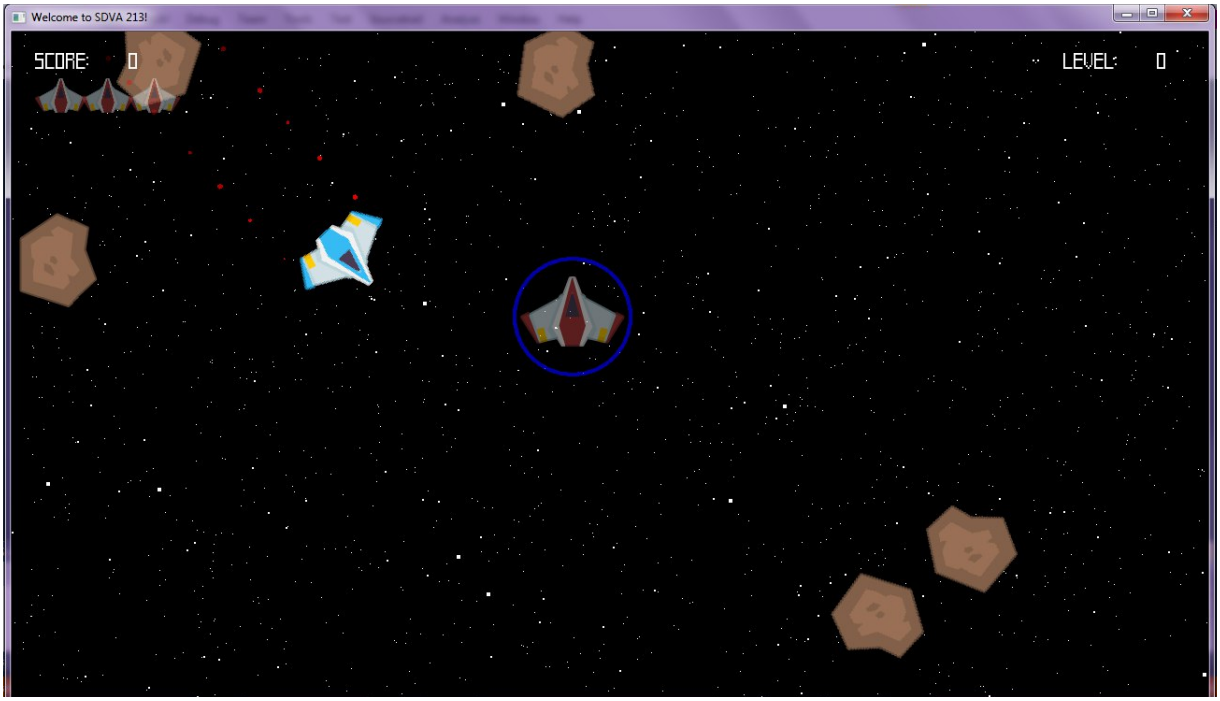
Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Features

- Durch die Nutzung von SFML-Library wurde eine Musik Schleife implementiert, Sprites geladen, ein Fenster geöffnet und Events Abfragen gestartet um Interaktionen abzufragen
- Hinzu wurde die SFML-Library genutzt um mathematische Berechnungen (Vektoren, ect.) anzustellen.
- Der Gegner verfolgt den Spieler und schießt auf ihn. Wenn der Spieler tot ist, beginnt er einen eigenen Weg zu fliegen.
- Alle Spieleobjekte können über den Fensterrand fliegen und erscheinen am gegenüberliegenden Rand
- Der Hintergrund besteht aus dutzenden sich bewegenden „Sternen“ und bringt dem Spiel mehr tiefe.
- Weitere Grafiken stehen in Unterordnern für Erweiterungen zur Verfügung.

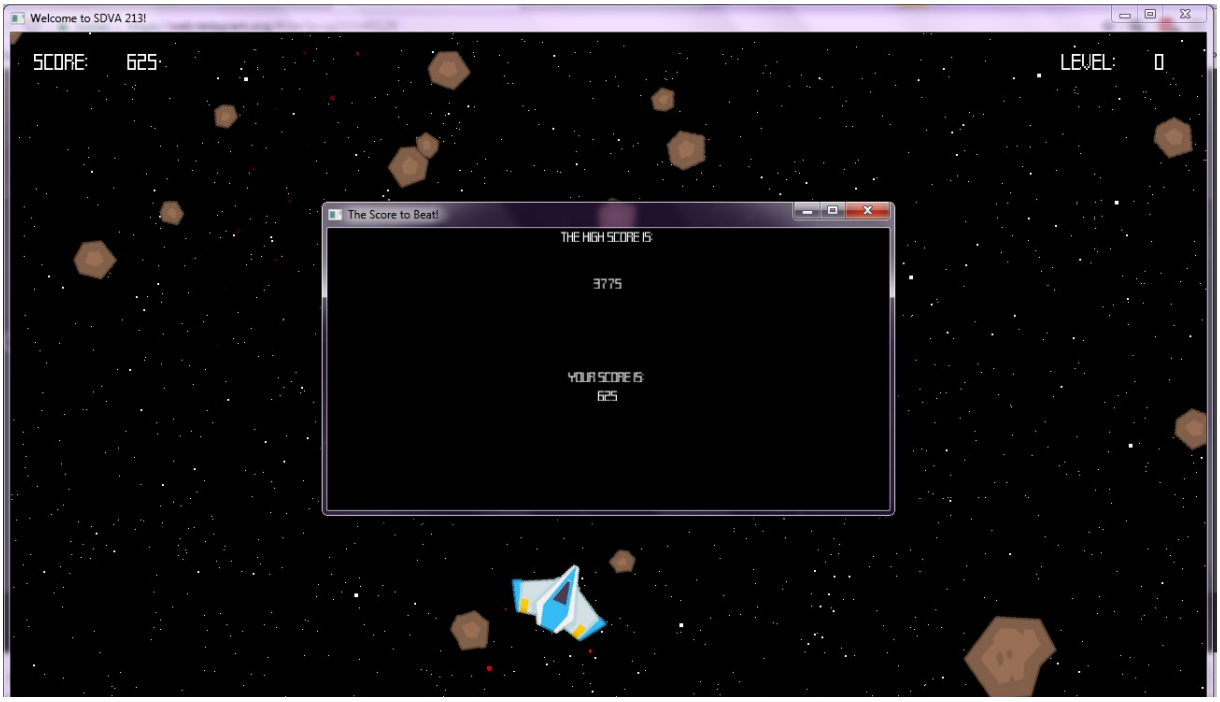
Screenshots



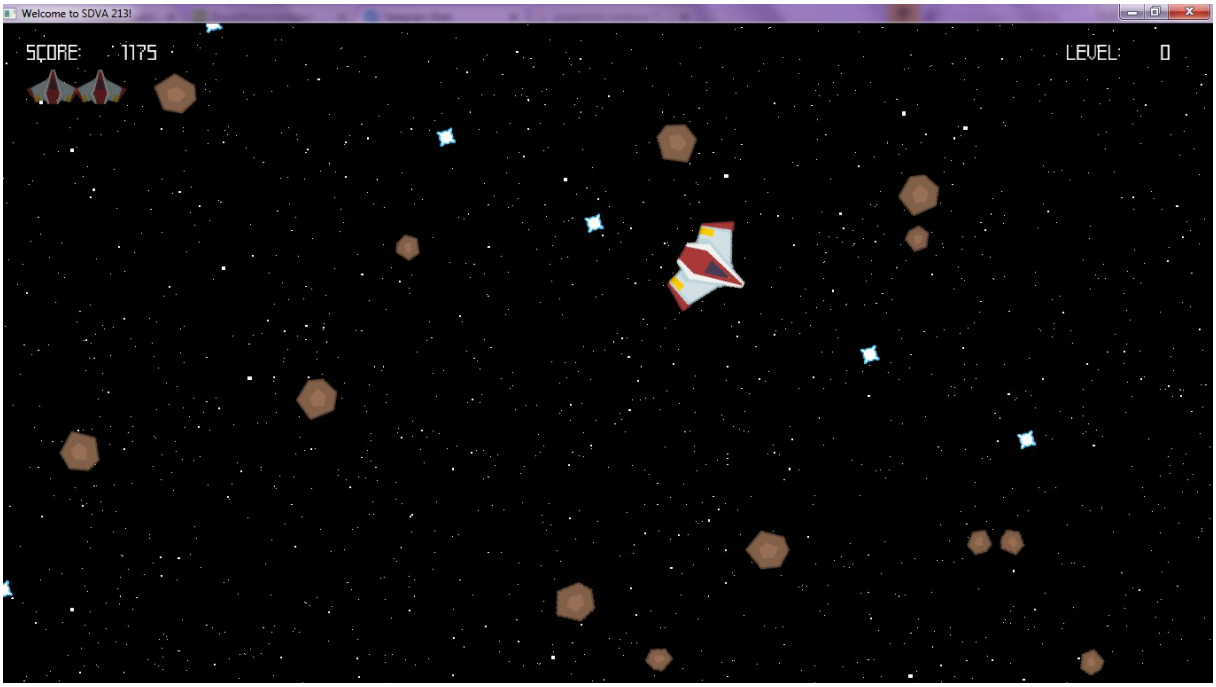
Spiel startet und man ist unverwundbar

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18



Mit der ‚Tab' – Taste kann man die Highscore einsehen



Gegner und große Asteroiden wurden getroffen und es fliegen mittlere und kleine umher

Credits/ Lizenzen

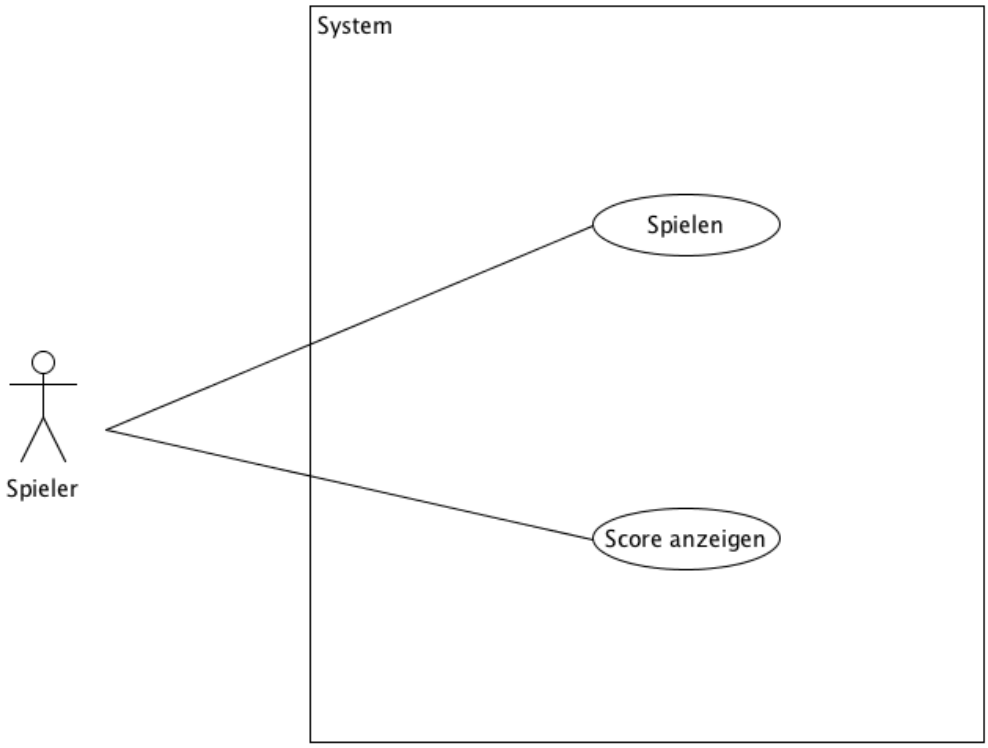
Das Spiel wurde von „UchihaKite" in C++ mit SFML programmiert.

Die Grafiken und Sounds stammen von „Kenny Vleugels" – www.kenney.nl und stehen unter License CC0 und somit zur freien privaten und kommerziellen Nutzung

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Use Cases



Use Case : Spielen.

Primärer Aktor: Spieler

Die Schritte 2-6 können in beliebiger Reihenfolge
ausgeführt werden.

Erfolgsszenario:
1. Der Spieler startet das Spiel.
2. Die Asteroiden und das Raumschiff vom Gegner bewegen sich in zufälligen Bahnen.
3. Der Spieler schießt und der Schuss bewegt sich in eine Richtung.
4. Der Spieler bewegt das Raumschiff nach rechts, links oder vorne.
5. Der Spieler bremst.
6. Das Raumschiff des Gegners schießt auf den Spieler.

Erweiterungen:
(2-6)a. Der Schuss berührt einen der vier Ränder des Spielfensters.
(2-6)a.1. Der Schuss erscheint wieder vom anderen Rand gegenüber und setzt seine Bewegung fort in dieselbe Richtung.
(2-6)a.1. Der Schuss verschwindet nach 5 Sekunden
(2-6)a.2. Das Spiel läuft weiter

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

3b. Der Spieler trifft einen großen Asteroiden.
3b.1. Der Asteroid wird in drei kleinere Asteroiden zerstört.
3b.2. Die neu entstandenen Asteroiden bewegen sich auseinander in zufällige Richtungen.
3b.2. Der Score erhöht sich um 100.
3b.3. Das Spiel läuft weiter

3c. Der Spieler trifft einen mittleren Asteroiden.
3c.1 Der Asteroid wird in zwei kleinere Asteroids zerstört.
3c.2 Die neu entstandenen Asteroiden bewegen sich auseinander in zufällige Richtungen.
3c.2 Der Score erhöht sich um 50.
3c.3 Das Spiel läuft weiter.

3d. Der Spieler trifft einen kleinen Asteroiden.
3d.1 Der Asteroid wird zerstört.
3d.2 Der Score erhöht sich um 25.
3d.3 Das Spiel läuft weiter.

(2-6)f. Der Spieler trifft ein Raumschiff.
(2-6)f.1 Der Raumschiff wird zerstört.
(2-6)f.2 Der Score erhöht sich um 100.
(2-6)f.3 Der Spiel läuft weiter.

(2-6)g. Der Spieler berührt einen Asteroiden, einen Raumschiff oder wird von anderen Rauschiffen erschossen.
(2-6)g.1 Der Spieler stirbt.
(2-6)g.2 Der Spieler verliert ein Leben.
(2-6)g.2 Der Spieler erscheint wieder und bleibt unverwundbar für 2 Sekunden.
(2-6)g.3 Das Spiel läuft weiter.

(2-6)h. Der Spieler berührt einen Asteroiden, ein Raumschiff oder wird von anderen Rauschiffen erschossen.
(2-6)h.1 Der Spieler stirbt.
(2-6)h.2 Der Spieler verliert ein Leben.
(2-6)h.2 Der Spieler hat keine Leben mehr.
(2-6)h.3 Der Spieler verliert.
(2-6)h.4 Das Spiel endet

(2-6)u. Der Spieler zerstört alle Asteroiden und Raumschiffe
(2-6)u.1 Das Level erhöht sich um eins.
(2-6)u.2 Es entstehen neue Asteroiden und Raumschiffe
(2-6)u.2 Das Spiel läuft weiter

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Usecase: Score anzeigen

Kurzbeschreibung: Der Spieler möchte sich den Score anzeigenlassen
Primärer Aktor: Spieler

Erfolgsszenario:
1. Der Spieler drückt die Taste TAB
2. Das Spiel wird gestoppt.
3. Der Score wird gezeigt.
4. Use Case endet erfolgreich.

Use Case: Schießen

Kurzbeschreibung: Der Spieler möchte Asteroiden oder andere Raumschiffe angreifen.

Primärer Aktor: Spieler.
Vorbedingung: Das Raumschiff ist nicht tot oder unverwundbar.
Nachbedingung: Der Schuss bewegt sich in die Richtung, auf die der Raumschiff zeigt.

Erfolgsszenario:
1. Der Spieler drückt die Taste Leertaste.
2. Das Raumschiff schießt.
3. Use Case endet erfolgreich.

Erweiterungen:
a.1. Das Raumschiff berührt einen Asteroiden oder wird von einem oder mehreren Raumschiffen erschossen.
a.2. Das Raumschiff wird zerstört.
a.3. Das System stellt fest, dass der Spieler mindestens ein Leben hat, Der Spieler verliert ein Leben.
a.4. Use Case endet erfolglos.

a.1b Das Raumschiff berührt den Asteroiden oder wird von einem oder mehreren Raumschiffen geschossen.
a.2b. Das Raumschiff wird zerstört.
a.3b. Das System stellt fest, dass der Spieler kein Leben mehr hat.
a.4b. Der Spieler hat verloren
a.5b. Use Case endet erfolglos.

Use Case: Den Score anzeigen

Kurzbeschreibung: Der Spieler möchte sich die Score anzeigen lassen. .

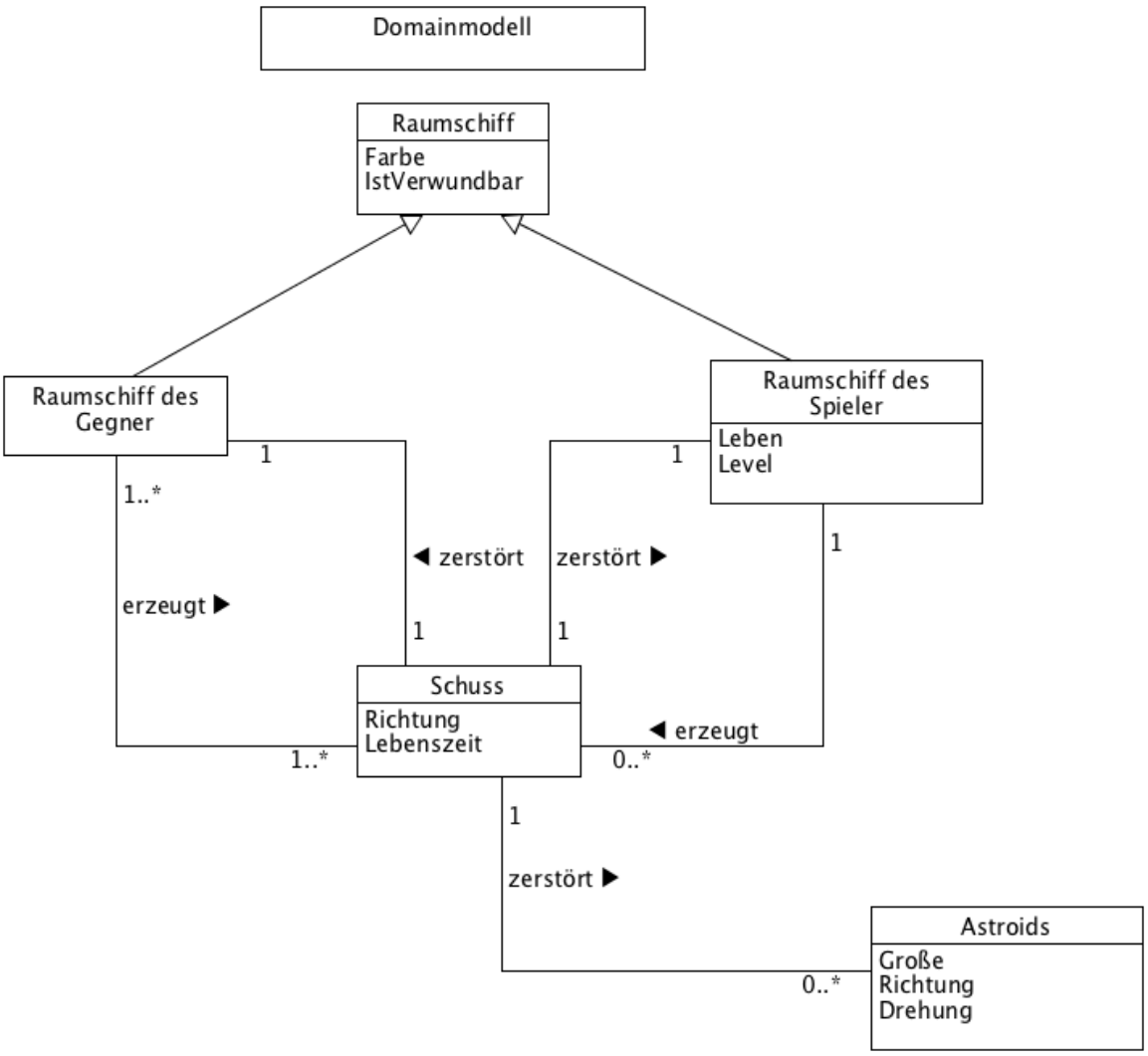
Erfolgsszenario:
C.1 Der Spieler drückt die Taste Tab.
C.2 Das Spiel stoppt und ein Fenster wird geöffnet, auf dem der Score steht.
C.3 Use Case endet erfolgreich.

Use Case: Das Spiel beenden

Kurzbeschreibung: Der Spieler möchte das Spiel beenden.

Erfolgsszenario:
C.1 Der Spieler drückt ALT+F4
C.2 Das Spiel ist beendet.
C.3 Use Case endet erfolgreich.

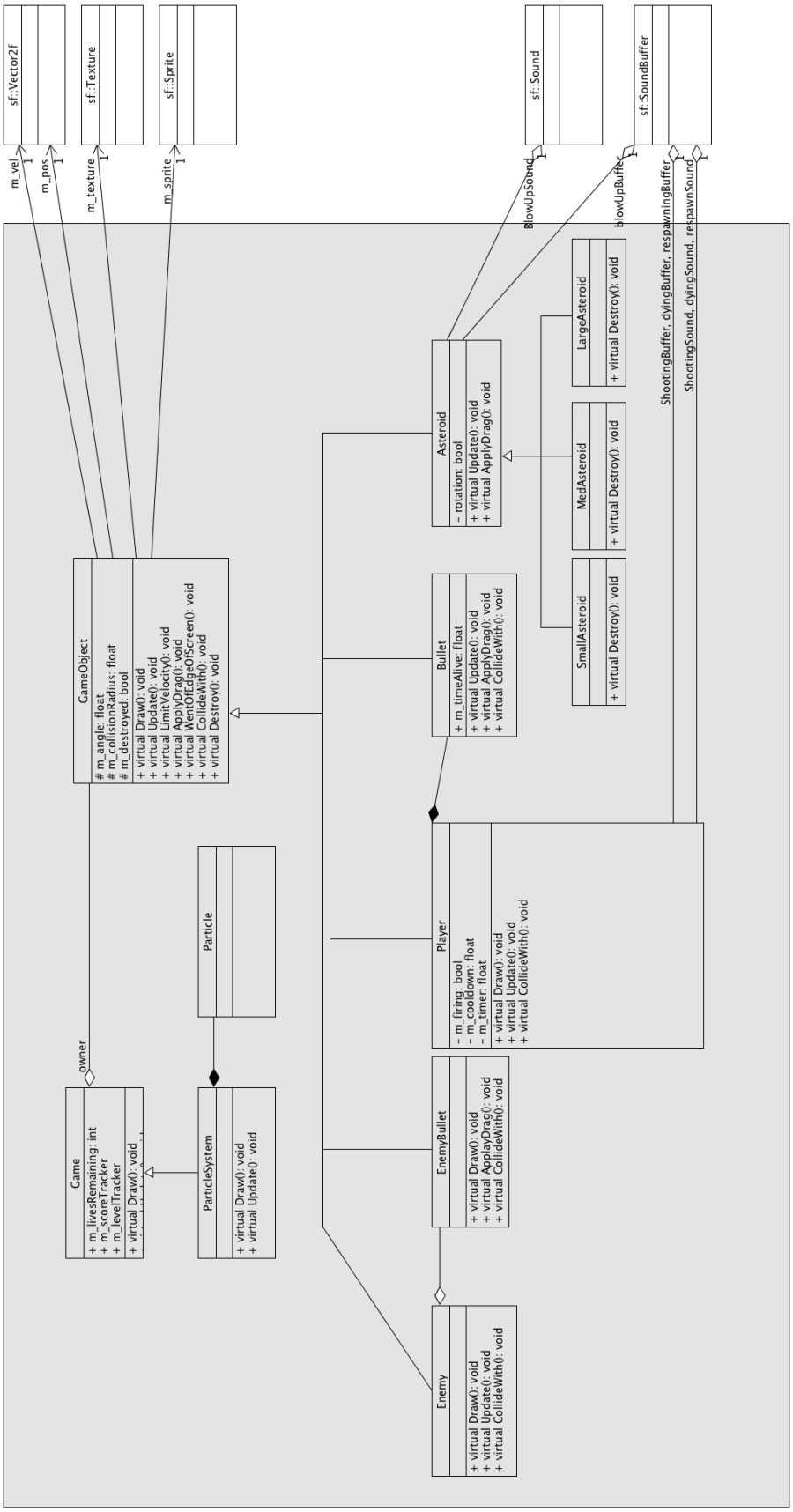
Domain Model



Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

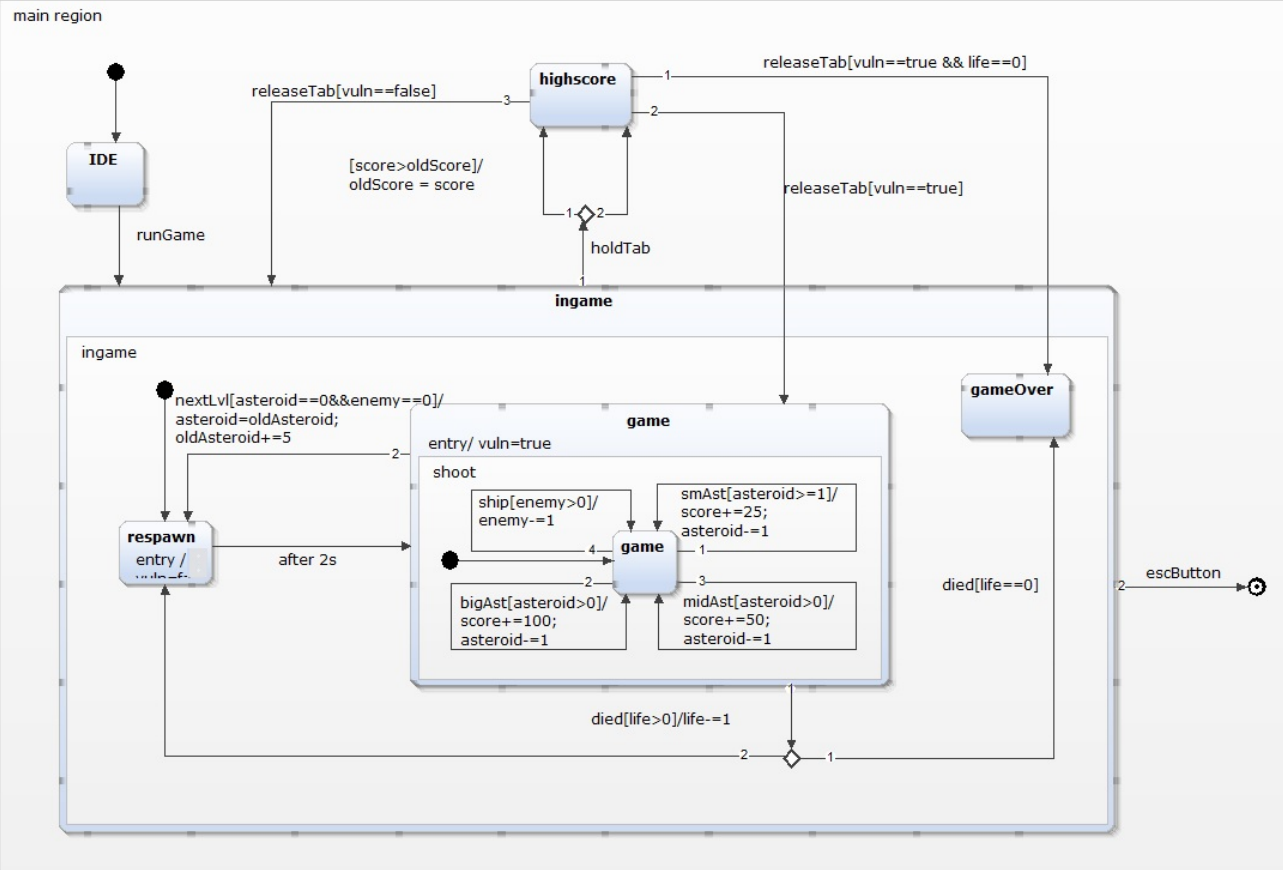
Klassendiagramm



Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Zustandsdiagramm



Beschreibung:

Das Spiel wird mit der Exe-Datei, bzw. vom Compiler aus gestartet. Daraufhin landet man in dem Respawn-Zustand und erhält die Kontrolle über sein Raumschiff. In diesem Modus kann man sein Raumschiff zwar bewegen, jedoch noch nicht schießen. Nach 2 Sekunden wird die Unverwundbarkeit aufgehoben wodurch die Benutzung der Waffen nun möglich ist.

Es sind Counter für die Gegner, sowie den Asteroiden vorhanden. Sind all diese Werte durch Abschüsse auf null reduziert, gelangt man wieder in den Respawn-Zustand. Nun gelangt man jedoch in das nächste Level und die Anzahl der großen Asteroiden wird erhöht. Außerdem werden einem pro Abschuss Punkte gutgeschrieben, abhängig von der Größe des Asteroiden.
Wird man von einem Gegner abgeschossen oder aber von einem Asteroiden getroffen, stirbt man und ein Life-Counter reduziert sich um 1 und man gelangt erneut in den Respawn-Zustand. Erreicht der Life-Counter 0 endet die Abzweigung in den GameOver Zustand.

Sowohl vom Respawn, Ingame als auch GameOver-Zustand gelangt man jederzeit in den Highscore-Zustand. Dabei wird überprüft, ob die aktuelle Punktzahl höher ist als der alte Highscore und daraufhin gegebenenfalls durch den neuen ersetzt.

Mit ALT+F4 wird das Spiel beendet und der Endzustand erreicht.

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Implementierung

ESC-Taste

```
74                  if (event.key.code == sf::Keyboard::Escape)
75                  {
76                      event.type = sf::Event::Closed;
77                      //window.close();
78                      return 0;
79                  }
```

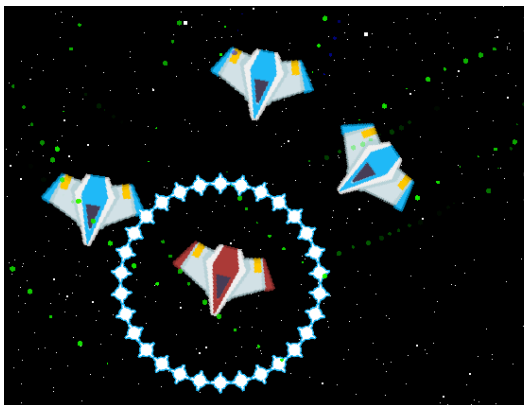
Die Implementierung des Escape-Events konnte recht einfach mit der SFML Bibliothek realisiert werden. Dafür wurde eine Bedingung in die Enginge.cpp (Main Datei) eingebracht, die abfragt ob der *event.key.code* jemals ein Keyboard Escape enthält. Sollte dies der Fall sein, wird ein event.type gesetzt mit dem Event Closed. Es wird aus der Funktion gesprungen und im späteren Code der Typ abgefragt. Das führt dann zum Schließen des Fenster und zum Beenden.

Power-Up

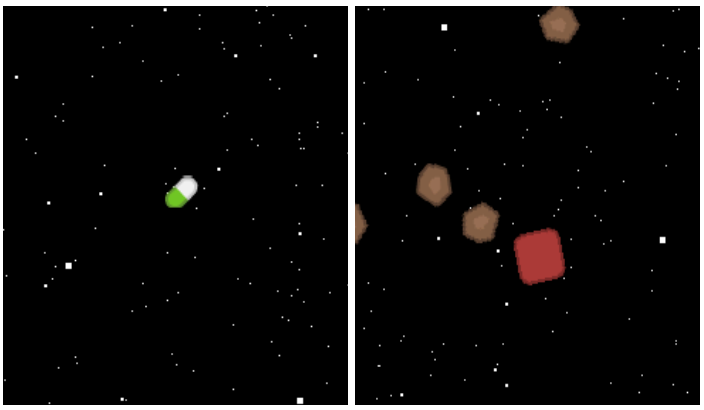
In jedem Spiel gibt es Bonus Gegenstände. Dadurch steigern die Möglichkeiten und die Spieltiefe. Um dies bei Asteroids zu realisieren, musste eine neue Klasse erstellt werden die wiederum auch von dem GameObjects erbt. Die Methoden Update(), Draw() und CollideWith() wurden neu definiert, sowie ein Generator im Konstruktor erstellt.
Dieser entscheiden um was für ein Power-Up es sich handelt und somit welche Fähigkeiten es besitzt.

```
31      switch (randomValue(therandomGenerator)) {
32          case 1:
33              m_skill = 1;
34              this->reload("Sprites/PNG/Power-ups/pill_green.png");
35              break;
36          case 2:
37              m_skill = 2;
38              this->reload("Sprites/PNG/Power-ups/shield_silver.png");
39              break;
40          case 3:
41              m_skill = 3;
42              this->reload("Sprites/PNG/Power-ups/bold_silver.png");
43              break;
44          case 4:
45              m_skill = 4;
46              this->reload("Sprites/PNG/Power-ups/powerupRed.png");
47              break;
48      }
```

Die Sprites wurden dafür neu geladen und die Memberfunktion m_skill belegt.
In der CollideWith Funktion wurde dann entsprechend dem m_skill die Methode bei Berührung ausgeführt. Das Power-Up wird dann in jedem zerstört.



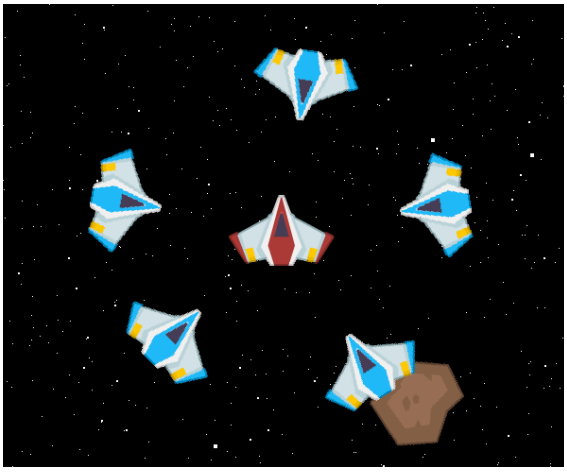
Ein Blitz führt zu mehreren Kugeln

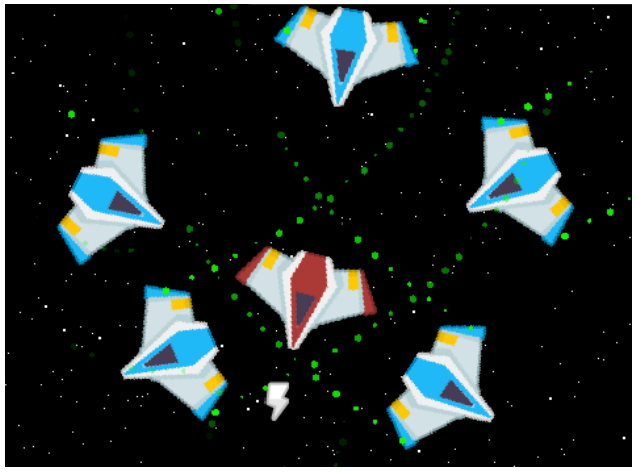


Die Tablette erhöht, der rote Kasten kostet ein Leben

Des weiteren gibt es noch ein Schild, welches den Spieler für 3 Sekunden unverwundbar macht.

Formationsflug





Dies war mitunter die schwierigste Umsetzung. Mehrere Gegner sollten um den Spieler fliegen und sich trotz seiner Bewegungen immer wieder im Kreis anordnen. Sie mussten sich stetig korrigieren und Ihre Position musste in Abhängigkeit der anderen Gegner sein.

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

```
156     float dist_player = length(playerLocation - m_sprite.getPosition());
157     
158     if(m_gotchha){
159         SetVelocity(0);
160         if (m_angle < m_angle_set-10 ){
161             SetAngle(m_angle-70);
162             SetVelocity(150);
163             fly();
164         } else if (m_angle > m_angle_set+10) {
165             SetAngle(m_angle+70);
166             SetVelocity(150);
167             fly();
168         } else {
169             SetVelocity(0);
170             m_gotchha = false;
171         }
172     }else {
173         if (dist_player <= 175 && dist_player > 150) {
174             m_gotchha = true;
175             //m_tracktimer = dt;
176         } else if (dist_player <= 150){
177             SetAngle(m_angle-180);
178             SetAccel(25.f);
179             fly();
180         } else {
181             if (!isPlayerSpawning) {
182                 SetAccel(dist_player-100);
183                 fly();
184             } else {
185                 SetAccel(25.f);
186                 fly();
187             }
188         }
```

Dafür wurde die Klasse erweitert und anstatt den Gegner nur einfach zu tracken und zu beschießen wurde das Flugmanöver eingefügt. Es besteht aus einer verschachtelten Abfrage. Zu erst wird berechnet wie weit der Spieler entfernt ist. Sollte dies weit genug sein, wird in seine Richtung geflogen. Ab einer bestimmten Nähe stoppen die Gegner und suchen ihre Position um den Spieler. Diese wird beim Initialisieren einer Runde und somit der Festlegung der Gegner in m_angle_set gespeichert. Sollten Sie den passenden Winkel haben beginnen Sie die Entfernung zu korrigieren. Dies wiederholt sich stets in Abhängigkeit vom Spieler.

Die Funktion Fly() beinhaltet nur die fliegenden Antriebspartikel, welche bei Bewegung berechnet und gezeichnet werden müssen.

Erkenntnisse

Man sagt Programmierung ist etwas, was niemals zu 100% fertig wird. Dementsprechend steigt mit der Erfahrung die Erkenntnis über alten Programmcode den man so nicht nochmal schreiben würde. Da C++ parallel zu diesem Projekt gelehrt wird sind hier die Sprünge noch größer. Toten Code in Klassen zu bauen, die nicht bei jedem Objekt gebraucht werden ist immer ungünstig. So muss man gestehen, dass die **PowerUps** im besten Fall keine große Klasse sind, sondern die unterschiedlichen Gegenstände voneinander erben und eine eigene Klasse bilden sollten.
Die **ESC** Implementierung bedenkt wiederum nicht, dass bei Beendigung, alle erstellten Objekte gelöscht werden sollten. Dies passiert jedoch auch nicht beim UrCode. Besser wäre

eine Lösung wo die erstellten Objekte, die ja alle dem Game zugeordnet wurden, via list wieder gelöscht werden.
Allgemein wäre es in Programmen sinnvoll, wichtige Iterale (Level/ Leben/ Gegneranzahl) in eine extra Datei zu schreiben um einfacher Veränderungen umzusetzen. Der **Positionsflug** wäre besser implementiert, wenn die Abfrage der Positionen der Formation nicht nur zu Level Erstellung erfolgt, sondern auch bei dem Verlust eines Gegners. Auch könnte sicherlich die If/Else Verschachtelung etwas „schöner“ gelöst werden.

Jedoch muss man irgendwann den Moment abpassen, wo Aufgaben erfüllt sind und Programme laufen. Optimierung und BugFixes wird es immer geben, es ist mehr eine Annäherung an 100%

Programmauswertung

Asteroids ist in sich eine solide Programmierarbeit. Die Vererbung der Spielobjekte von einer Hauptklasse dem Gameobjekt, macht es einfach neue Objekte zu implementieren, welche einfach in die Game Klasse integriert und dadurch geupdatet werden. Die File Hierarchie bietet stets die passenden Informationen. Im GameObjekt.h sind alle SpielObjekte vertreten und können mit den anderen Objektenklassen interagieren. Auch die Logik der Vektor Abfrage zur Bestimmung von Entfernungen, Bewegungen und dem Driften sind in sich schlüssig und schlang geschrieben.
Betrachtet man jedoch die Sound Implementierung fragt man sich, warum jedes Objekt sich einen eigenen SoundBuffer schreiben muss und nicht auf eine andere Klasse mit allen SoundBuffern und Sounds zugreifen kann. Dadurch war es auch recht umständlich, wenn man ein Geräusch abspielen wollte es zu implementieren. Man musste es extra mit in die Klasse einfügen oder besser auf die Sounds eines anderen Objektes zugreifen. In unserem Fall wurde Player als Jukebox missbraucht. Genauso unübersichtlich ist es, keine Konstanten zu verwenden die alle an einem Ort hinterlegt sind. Dadurch wären Test Veränderungen viel einfacher möglich. Auch manche Methoden die wiederholt in vererbenden Klassen überschrieben worden sind, hätte man allgemeiner gefasst sauber von der Main Klasse erben können. Die ganze Engine.cpp Datei scheint eher wie eine Teststrecke zu sein. Die ersten Asteroids werden direkt in einer separaten Schleife erstellt, bei Level Update in einer anderen.

Was also an der einen Stelle sauber getrennt wurde, wurde an einer anderen Stelle wieder festgeklebt. Aber das ist auch wiederum das Gute. Man konnte viele Erkenntnisse vom Code lernen, wie man mit Objekten umgehen sollte. Er war dabei nicht minimalistisch und perfekt, was aber wohlmöglich das Lesen und Verstehen vereinfacht hat. Problemzonen in fremden Code zu entdecken schafft Interesse es besser machen zu können und zu wollen.

Fazit: Ein Programm mit mehr Potential

Name: Sören Größer
Ahmed Jenbaz;
Fabian Rod

Datum: 26.06.18

Lesson Learned

Auf der Suche nach einem geeigneten Programm für unser Projekt ging es zunächst darum, ein Programm zu finden und dieses auf dem eigenen Rechner lauffähig zu bekommen. Dazu war es oftmals erforderlich ein neues Projekt zu erstellen. Dabei haben wir gelernt, welche Einstellungen am Compiler vorgenommen werden müssen, um aus den einzelnen C++ Dateien ein ganzes Projekt zu erstellen.

Schwierigkeiten gab es vor allem bei dem Einbinden von externen Dateien wie z.B. Dateien, welche für die Grafikausgabe erforderlich sind. Einige hatten so viele Updates, dass sie mit der damals genutzten Version nicht mehr kompatibel sind, oder aber andere Dateien existierten gar nicht mehr.

Zudem haben wir bei dem Bearbeiten der Aufgaben die Stärken und Schwächen der einzelnen Gruppenmitglieder feststellen können, und dementsprechend die erforderlichen Aufgaben sinnvoll verteilt. Durch diese Rollenverteilung war eine effizientere Arbeitsweise möglich. Die Erfahrung der Rollenverteilung wird noch bei weiteren Gruppenprojekten hilfreich sein.

Für unser Projekt war es erforderlich, verschiedenste Diagramme zu erstellen. Somit war es uns möglich, im Unterricht erlerntes Wissen praxisnah an unser ausgewähltes Projekt umzusetzen. Dabei haben wir unsere Kenntnisse in der Erstellung von Use-Cases, Klassendiagrammen sowie Zustandsdiagrammen vertieft. Auch haben wir gelernt, sinnvoll erforderliche Programme auszuwählen, welche uns hilfreich erscheinen, und sich in diese einzuarbeiten und sicher umzugehen. Darunter fallen z.B. Umlet, Yakindu oder Sourcetrail.

Schlussendlich ging es auch darum, ein bereits bestehendes Programm zu verändern und zu erweitern. Dabei mussten wir uns in einem bereits bestehenden Code einarbeiten. Auf der Suche nach einem geeigneten Programm haben wir viele verschiedene Stile der einzelnen Entwickler feststellen können. Dabei waren manche gut lesbar, andere wiederum hatten einen schlechten Stil. Die Herausforderung war es zunächst, den Code zu verstehen und die Zusammenhänge zu erkennen.

Dazu haben wir viel herumexperimentiert um ein besseres Gefühl zu bekommen und die Auswirkungen zu beobachten. Nach und nach wurde der Code verständlicher und wir konnten neue Funktionen implementieren. Vor allem das Beschäftigen mit bereits bestehender Software wird uns für zukünftige Projekte, sowohl in der Universität als auch im Berufsleben, von Hilfe sein. Gerade im Berufsleben ist es wichtig, sich in bereits bestehenden Projekten einzuarbeiten und diese zu verändern.