TECHNISCHE HOCHSCHULE DEGGENDORF

FAKULTÄT ANGEWANDTE INFORMATIK

Prüfungsstudienarbeit zum Thema

Java Programmierung FPW – 2D Plattformer

Abbildung 1: Titelbildschirm 2D-Plattformer

vorgelegt von

Anton Kraus - Matrikelnummer: 00804697

Simon Obermeier - Matrikelnummer: 00800498

Robin Prillwitz – Matrikelnummer: 00805291

Eingereicht: 04.07.2022

Betreuer: Bodenschatz

Inhalt

Abbildungsverzeichnis

Abkürzungsverzeichnis

JAR	Java Archive
JRE	Java Runtime Environment
JDK	Java Developement Kit
UML	Unified Modelling Language
HTML	Hyper Text Markup Language

Glossar

Javadoc	Javas Standard Dokumentationsmethode
JavaFX	Java Graphikbibliothek
Maven	Build System
MySQL	Relationale Datenbank

Ausarbeitung der Aufgabenstellung

Für das FWP Fach Java Programmierung sollte eine kleine Java-Anwendung für ein frei wählbares Szenario programmiert werden. Als Anforderungen der Prüfungsstudienarbeit wurde Vererbung, ein Interface sowie eine Javadoc Dokumentation gefordert. Der erste Aspekt der Vererbung wurde über die Nutzung von „JavaFX“ realisiert, welche zur Abstrahierung der zu zeichnenden Spielobjekte genutzt wurde. Die Umsetzung eines Interfaces erfolgte durch das Anlegen von zentralen Globalen Variablen in einem ausgelagerten Dokument. Die Javadoc wurde durch eine Interaktive HTML Dokumentation umgesetzt, wobei sich hier auf die wichtigsten Funktionen, sowie variablen aller Scopes und Konstruktoren beschränkt wurde.

Als besondere Herausforderung wurde eine GUI mithilfe von JavaFX und die Erstellung einer lauffähigen Jar-Datei gelistet. Zudem war die Implementation einer relationalen Datenbank in das Programm gewünscht. Die Umsetzung der GUI erfolgte mithilfe des Grafik-Frameworks „JavaFX“, womit sogleich die Implementierung der Hintergrundmusik, sowie der Spielsounds erfolgte. Zur Generierung einer lauffähigen Jar-Datei wurde das Framework Maven hinzugezogen. Für die alleinige Ausführung der Jar-Datei ist jedoch nur eine „JDK18“-Installation oder neuer erforderlich. Eine „MySQL“-Datenbank wurde Online umgesetzt, jedoch mit der Default-Option die Daten offline zu Speichern.

Programmübersicht

Wie bereits erwähnt wurde im Rahmen der Prüfungsstudienarbeit, für das FWP-Fach Java-Programmierung, ein 2D-Plattformer mithilfe des Grafik-Frameworks JavaFX umgesetzt. Mit dieser wurden individuelle graphische Elemente durch Sprites, sowie ein Soundtrack aus eigener Produktion implementiert. Grund hierfür war die Wahl des von uns erfundenen Settings.

„PizzaHut2077“

Als Pizza Hut Lieferant im Jahr 2077 muss man in einer dystopischen Umgebung Düsburgs gegen Dominos Pizza kämpfen um den Ruf der runden Fressalien von PizzaHut zu Retten.

Genauer ist dieser 2D-Plattformer ein Sidescolling-Jump’n’Run, weswegen man von zufällig generieten Plattformen zu Plattformen springen muss. Erschwert wird dies durch das zufällige Erscheinen von Gegnern in der Gestalt des Domino-Logos, welche sich auf den Spieler zubewegen und versuchen diesen durch Berührung an seiner Auslieferung zu hindern. Damit die Gegner den Spieler nicht erreichen, kann man versuchen diesen auszuweichen oder durch das Abfeuern von Projektilen in Form von Pizzen diese auslöschen. Es wäre ein leichtes durchgehend zu feuern, deswegen ist man durch eine links oberhalb im Spiel angezeigte „Heatbar“ beschränkt. Man kann also nur feuern, wenn die sogenannte „Heatbar“ niedrig genug ist. Graphisch wurde das Design des Spielers, ein Pizzabote im Pizza Hut Design, sowie das der Gegner in Form des Logos von Dominos angepasst. Ziel ist es Punkte zu sammeln, um nach dem Tod unter die Top 10 der Bestenliste zu kommen und seine Punktezahl mit Namen in die je nachdem Globale oder Lokale Datenbank zu übertragen. Punkte werden durch das Erreichen neuer Plattformen oder das Eliminieren von Gegnern erreicht.

UML Diagramm

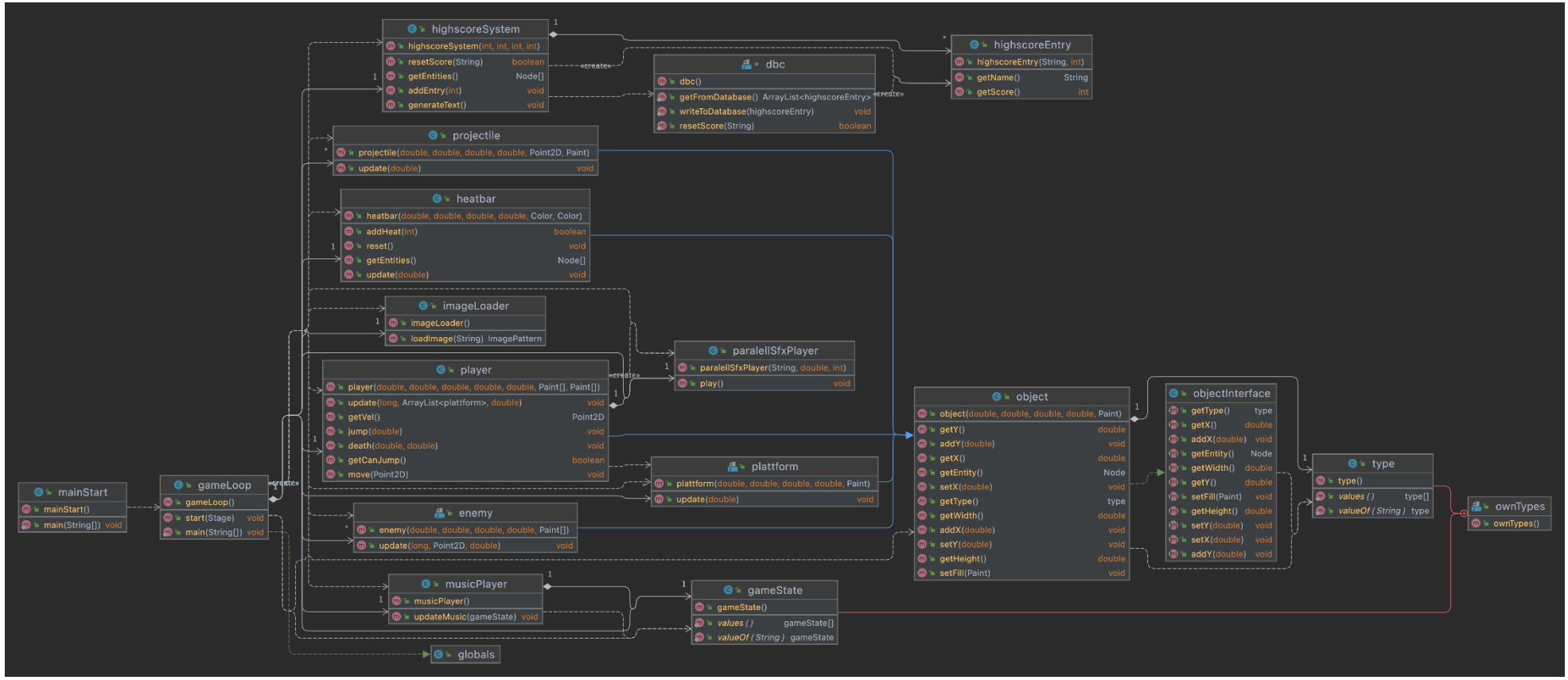

Abbildung 2: UML-Diagramm

Das hier in Abb.2 gezeigte UML Diagramm spiegelt den Inhalt der implementierten Pakete an, wobei jedoch auf äußere Abhängigkeiten verzichtet wurde. Aufgrund des Imports der Abbildung kann die Auflösung dieser beschränkt sein, weswegen ich hier darauf Verweise, das Diagramm ebenfalls im Projektordner zu öffnen. Die Textuelle Ausarbeitung des Diagramms wird im nächsten Abschnitt behandelt.

Klassen und Packages

Wie bereits im vorherigen Abschnitt referenziert, wird im Folgenden die Aufteilung der Klassen und Packages beschrieben. Hier verweise ich wiederum auf das UML Diagramm Abb.2.

Grundsätzlich basiert unser 2D-Sidescroller auf einer implementierten State-Machine, welche abhängig von den unterschiedlichen Inputs oder Aktionen des Users/Spielers Ihren State, sowie parallel dazu den Soundtrack, wechselt. Um nun auch die Graphische Orientierung des UML Diagramms beizubehalten, beginnen wir mit der „mainStart“, welche die Main-Mehtode enthält. Das Hauptprogramm hierbei ist der „GameLoop“. Dieses erbt von dem Interface „Globals“ sowie von der JavaFX Superklasse „Applikationen“. Das Interface „Globals“ vererbt extern, da hier alle Globalen Spieleeinstellungen wie z.B. die feuerrate definiert werden. Dies war besonders in den Anfängen hilfreich, da so schnell und übersichtlich wichtige Parameter eingestellt werden konnten. Für die graphische Erzeugung eines Fensters in dem Framework wurde die Superklasse „Applikationen“ benötigt. Die nächste implementierte Superklasse war „Object“ welche das Interface von „gameObject“ verwendet. Dadurch konnten nicht nur Fehler vermieden werden, sondern auch eine gleiche Grundfunktionalität des Spielers gewährleistet werden. Grundidee dahinter ist, dass der User optional alle Objekte in einem Array speichern kann. Implementiert wurde dies nicht, da sonst für jedes Objekt geprüft werden hätte müssen, ob das jeweilige Objekt auch ein Objekt ist. Da die State-Machine mein Aufgabenbereich war verweise ich bereits hier auf den Abschnitt der eigenen Leistungen. Grundsätzlich besteht die State-Machine aus einzelnen JavaFX-Panes, welche durch Aktionen im Spiel oder durch Tastatureingabe wechseln. Damit beim Laden einzelner Elemente eines neuen Zustandes wiederum alte nicht überschrieben werden, werden die jeweiligen Programmteile aus dem „GameRoot“ geladen, bzw. entladen. Sprich es werden bei einem Szenenwechsel nicht mehr benötigte Panes aus dem „GameRoot“ entladen und neue Panes geladen. Dadurch wird ermöglicht, dass Elemente zu dem vom „GameRoot“ graphisch dargestellten Fenster co-exestieren können.

Die States sind nicht nur and die Graphischen Elemente gebunden, sondern auch an den Music-Player, welche je nach State einen entsprechenden Clip spielt. Ebenfalls der Parallel Sfx-Player spielt Sound Effekte in einem Effizienten weg ab, in welchem die einzelnen Dateien nicht neu geladen werden müssen. Der Soundtrack wurde dabei mit den Programmen „Live“ von Ableton sowie „Audition“ von Adobe erstellt. Dazu aber im Kapitel eigene Leistungen mehr.

Zudem wurde eine interaktive Javadoc als HTML Dokumentation erstellt. Diese „index.html“ kann durch das Öffnen mit einem Browser aufgerufen werden. Zu finden ist diese unter dem Speicherpfad „./javadoc/apidocs“. Des Weiteren stellt diese Interaktive Dokumentation alle Abhängigkeiten zwischen Klassen für jedes Paket dar.

Eigene Leistung

Aufgrund hoher Präsenz an der Hochschule, einem regen Austausch innerhalb unserer Gruppe, sowie spaß an dem Projekt sind viele Teile des Programms, vor allem größere Klassen, wie der Game Loop durch Kollaboration entstanden. Jedes Mitglied der Gruppe hatte dabei seinen eigenen Aufgabenbereich. Aus diesem Grund und durch Abhängigkeit einzelner Teile, kommt es zu gelegentlichen Überschneidungen in der Ausarbeitung.

Da die Programmführung mit meiner eigenen Leistung zusammenhängt geht die Erklärung dieser mit der von mir Implementierten State Maschine einher.
Ebenfalls der dazu im Hintergrund laufende Musik-Player, sowie der Parallel-SFX-Player war Teil meiner Aufgabe.

State Maschine

Womit wir bereits zum ersten von drei Teilen kommen, welch ich der Gruppe beizutragen hatte. Hierzu habe ich ein abstraktes Diagramm erstellt, um die States besser graphisch nachvollziehen zu können. Einzelne Zeilen code wurden hier aufgrund des Umfangs nicht behandelt, können jedoch Im Verzeichnis nachgeschlagen werden.

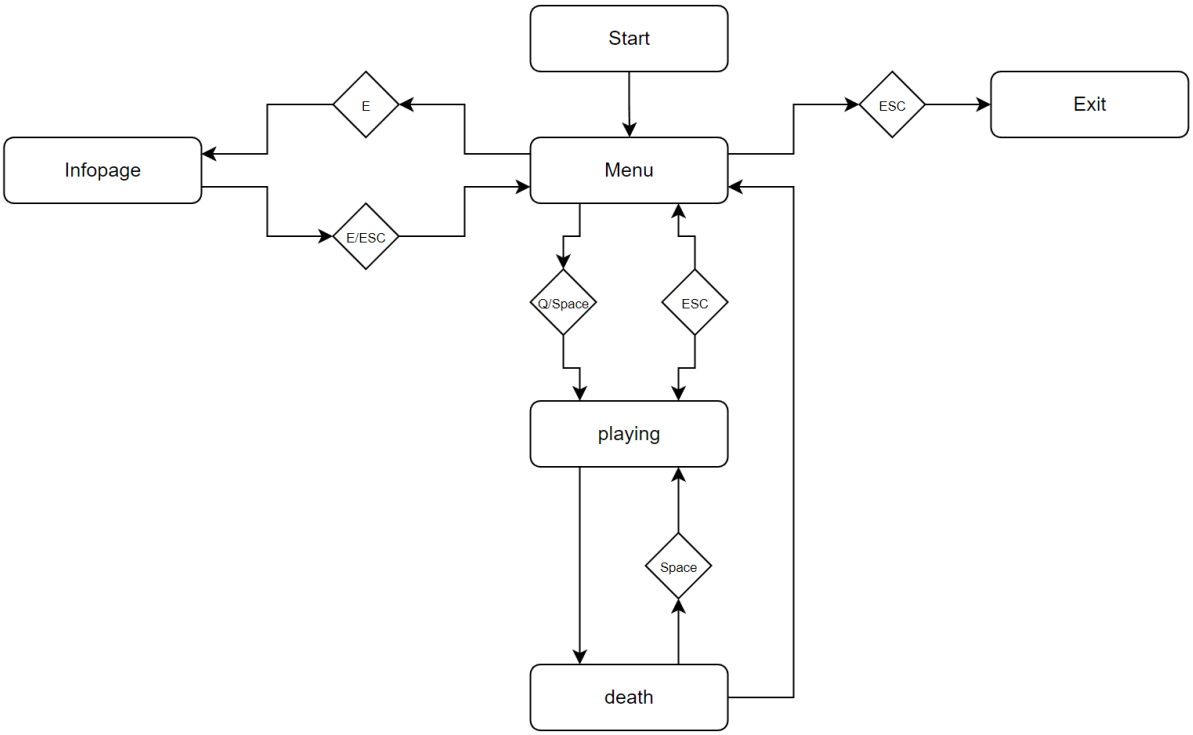

Abbildung 3: State-Machine

Wie in Abb.3 zu sehen war meine Aufgabe die Implementierung der verschiedenen States des Spieles und miteingehend dazu die „KeyActions“, sowie der vom State abhängige Music-Player, sowie der SFX-Player. Grundsätzlich lädt bzw. entlädt die State Maschine die jeweiliges Panes des States aus dem GameRoot.

Die State Maschine sowie die „KeyActions“-Funktion befinden sich im „GameLoop“ und können in diesem nachvollzogen werden.

Programmführung

Wie bereits im vorherigen Kapitel erwähnt, kommen wir nun zu der Programmführung, welche mein Teil des Projektes war. Ich beschreibe zudem auch hier gleich die Menüführung da diese auf der State Maschine basiert.

Bevor diese jedoch realisiert werden konnte, musste parallel dazu die Funktion „keyActions“ implementiert werden. Eingelesen werden die Tastaturanschläge durch einen „Event-Listener“ in der Hauptszene der Applikation, woraufhin diese dann in eine Globale „Hash-Map“ geschrieben werden und abgefragt werden können. Die Abfrage erfolgt durch die Methode „isPressedKey“ und kann an den benötigten stellen aufgerufen werden, wie in meiner Funktion „keyActions“. Im Menü wird dann je nach Tastenanschlag der State des Spieles gewechselt oder im Fall des aktiven Spieles die Bewegung des Spielers.

Nach dem Ausführen der Jar-Datei, welche durch den Maven Befehl „mvn package“ generiert wird, wird man von dem Titelbildschirm wie in Abb.1 zu sehen, mit dem passenden Logo „PizzaHut2077“, begrüßt. Jetzt hat man die Option über den Tastatur Input mit „E“ zu einer Informationsseite über die Autoren zu gelangen. Hier hat man zudem die Option, falls man das Admin Passwort kennt die Datenbank zurückzusetzen. Durch erneutes Drücken von „E“ oder „ESC“ gelangt man wieder zu dem Titelbildschirm. Siehe ABBXX.

Wieder im Titelbildschirm kann man den Sidescroller mit „Q“ starten. Auch der eher ruhige Soundtrack des Menüs wechselt zu einem schnelleren. Da sich der Bildschirm automatisch nach rechts bewegt wurde in der Funktion „keyActions“ die Bewegung nach links mit „A“ oder „D“ implementiert. Um zu springen, muss LEERTASTE gedrückt werden. Damit man sich gegen die dem Spieler zubewegenden Gegner im Design des Domino-Logos wehren kann, Zielt man mit dem Mauszeiger und feuert durch das Drücken der linken Maustaste Projektile ab. Diese sind jedoch an eine „Heatbar“ gebunden, welche sich mit jedem Schuss auflädt und bei häufiger Feuerrate erst wieder „abkühlen“ muss, bevor man erneut feuern kann. Parallel hierzu hört man für jede Aktion des Spielers einen Sound, einen SFX-Sound der parallel abgespielt wird. Für die Funktion und Implementation verweise ich auf Kapitel Parallel SFX-Player. Das Spiel kann im aktuellen Spiel-State jederzeit durch das Drücken von ESC beendet werden.

Der Spieler stirbt durch das erneute Bespringen einer bereits angesprungenen Plattform, das Erreichen eines Gegners oder durch das Fallen aus der Welt beim Verfehlen einer Plattform. Hier gleich als Beispiel: Stirbt man, so findet der Szenenwechsel in den „death“-State statt und das „death“-Pane wird im „GameRoot“ angezeigt und ein anderer Clip wird gespielt.

Sollte man nun einen Score unter den Top 10 erreicht haben, kann man seinen Namen mit seinem Score in der Datenbank verewigen. Ein leerer oder zu langer Name kann nicht eigegeben werden. Falls die Punktezahl zu niedrig war, wird diese Option nicht angezeigt. Dieses Highscore System und die dazugehörige Datenbank wurde jedoch nicht von mir Implementiert. Durch ESC gelangt man wieder zu dem Titelbildschirm und der Ablauf beginnt erneut.

Die Steuerung ist ebenfalls in der im Verzeichnis zu findenden README nochmals beschrieben

Music Player

Wie bereits in der Programmführung referenziert, läuft abhängig des jeweiliges States ein Music Player, welcher abhängig dieser verschiedenen Clips spielt. Die „.wav“-Clips entstanden wie in der Präsentation angemerkt durch Kooperation mit einem anderen Gruppenmitglied. Der Player jedoch wurde von mir Implementiert. Auch hier habe ich mir zur Vereinfachung ein abstraktes Diagramm erstellt. Umgesetzt wurde der Player in der gleichnamigen Klasse „MusicPlayer“.

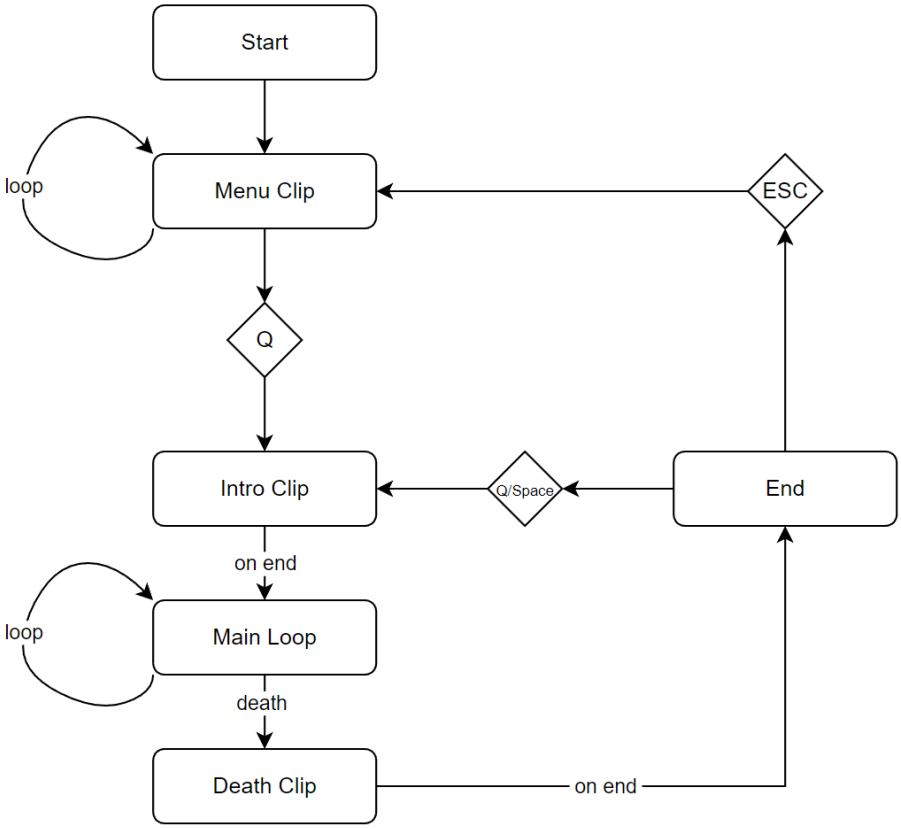

Abbildung 4: Music-Player

Beim Start des Programms wird ein ruhiger Clip gespielt. Durch den Wechsel in den Game-State startet ein Intro Clip, welcher abläuft und in einen Main Loop Clip übergeht. Dieser läuft so lange in einem loop, bis die Spieler stirbt oder den State durch ESC verlässt. Beim Tot spielt einmalig ein „Death“-clip. Je nach Input im „death“-State spielt der jeweilige Clip. In der ersten if-Schleife der Funktion wird abgefragt, falls kein Clip spielen sollte, ob sich ein Clip in der Warteschlange befindet. Ist dies der Fall, wird der sich in der Warteschlange befindende Clip gespielt. Die nächste Abfrage behandelt den optimalen State, in welchem gerade ein Clip spielt und sich kein Clip in der Warteschlange befindet. Zuletzt wird geprüft, falls ein Clip spielt und sich ein anderer Clip in der Warteschlange befindet, wird der aktuelle Clip sofort beendet und der sich in der in der Warteschlange befindende gespielt. Ein Beispiel hierfür wäre der Szenenwechsel vom „mainLoop“ zum Start des Spiels. Im zweiten Teil der else-Abfrage wird sich lediglich darum gekümmert, bei einem State- Wechsel etwas zu queuen.

Parallel SFX-Player

Der letzte Teil meines Aufgabenbereiches war die Umsetzung eines Players, welcher parallel zu dem Music Player, während des Spiels die Sound Effekte spielt. Diese werden durch den Input des Users oder Aktionen im Spiel getriggert. Beim Start des Programms werden im Konstruktor die Sound-Dateien vorgeladen. Dieser Kopiert/Lädt eine bestimmte Anzahl an identischen Clips/Media Objekten und speichert diese in einem lokalen Array. Zudem wird zu jedem Clip eine Lautstärke gesetzt.

Beispielsweise wird während des Spiels ein Projektil abgefeuert, so wird die private Instanz der klasse Parallel SFX-Player „missileSfx“ aufgerufen und mit der „play“-Funktion abgespielt. Diese Instanz beinhaltet den Namen der .wav Datei, sowei der Lautstärke, wobei 0.4 für 40% der Gesamt-Lautstärke steht. Ebenfalls wird die Anzahl der sich im Array geladenen Sounds festgelegt, in unserem Beispiel „5“. Wird nun ein Projektil abgefeuert wird der „missileSfx“-Sound abgespielt. Hierbei wird immer ein bereits nicht abgespielter oder ein bereits fertig abgespielter Sound verwendet. Sollten durch schnelles feuern, alle 5 Sound noch abspielen, so wird der am weitesten fortgeschrittene Sound gesucht, beendet und neu gestartet. Im Code habe ich dies in der letzten if-Schleife umgesetzt.

Der Sinn des ganzen ist das wir viele Soundeffekte spielen können, indem wir alte Soundeffekte einfach „Überschrieben“ also neu starten und nicht beliebig viele Soundeffekte zur „runtime“ laden müssen. Für jeden Sound im Effekt wird ein „parallelSfx“-Sound angelegt ausgenommen von Musik.