

# **Tutorial zum Porten und Schreiben von 16 Bit Songs mit MML**

**Ahrion**

22. April 2021

# Inhaltsverzeichnis

<b>1</b>	<b>Einführung</b>	<b>3</b>
1.1	Vorwort . . . . .	3
1.2	Überblick . . . . .	4
1.3	Musiktheorie und MML Code Analogie . . . . .	5
<b>2</b>	<b>Ersten Song schreiben</b>	<b>6</b>
2.1	Standard- und Spezialbefehle . . . . .	6
2.1.1	Tempo und Lautstärke . . . . .	7
2.1.2	Standardlänge . . . . .	8
2.1.3	Loops . . . . .	10
2.1.4	Panning . . . . .	13
2.1.5	Instrumente setzen . . . . .	14
2.1.6	Lokale Lautstärke . . . . .	17
<b>3</b>	<b>Song aus Midi konvertieren</b>	<b>19</b>
3.1	Vergleich der Konverter . . . . .	19
3.2	Installation und Benutzung von mmltk . . . . .	20
3.3	Midi aufbereiten und konvertieren . . . . .	21
<b>4</b>	<b>Instrumente und Samples</b>	<b>23</b>
4.1	Custom Samples . . . . .	24
4.2	Instrumente definieren . . . . .	25
4.2.1	ADSR . . . . .	25
4.2.2	Gain . . . . .	27
4.3	Sample Groups . . . . .	29
4.4	Eigene Samples erstellen . . . . .	30
<b>5</b>	<b>Ticks, Song Stats und Hexbefehle</b>	<b>34</b>
5.1	Ticks . . . . .	34
5.2	Stats . . . . .	35
5.3	Hexbefehle . . . . .	35
5.3.1	Pan Fading . . . . .	35
5.3.2	Tempo Fading . . . . .	35
5.3.3	Vol Fading . . . . .	35
5.3.4	Pitch Bendings . . . . .	35
5.3.5	Vibrato . . . . .	35
5.3.6	Tremolo . . . . .	35
5.3.7	Legato . . . . .	35
5.3.8	Light Staccato . . . . .	35
5.3.9	Echo . . . . .	35
5.3.10	Yoshi Drums . . . . .	35
<b>6</b>	<b>Weitere Standardbefehle und Makros</b>	<b>35</b>
6.1	Loops . . . . .	35
6.2	Intros . . . . .	35
6.3	Makros . . . . .	35
6.4	Remote Codes . . . . .	35

<b>7</b>	<b>Global Songs</b>	<b>35</b>
<b>8</b>	<b>FAQ</b>	<b>36</b>
<b>9</b>	<b>Anhang</b>	<b>37</b>

# 1 Einführung

## 1.1 Vorwort

Dieses Tutorial richtet sich hauptsächlich an Anfänger, die entweder noch keine Musik geportet haben oder gerade damit beginnen. Außerdem sollten einige Kapitel interessant für diejenigen sein, die zwar nicht vorhaben selber Musik zu porten, aber Musik in eigene Roms patchen möchten oder einfach nur wissen wollen, wie der allgemeine Ablauf des Portens aussieht. Die nötige Musiktheorie ist gering und wird in diesem Tutorial abgedeckt. Vorkenntnisse in diesem Bereich sind also keine Voraussetzung.

## 1.2 Überblick

Musik Ports für den SNES schreiben wir in MML (Macro Musik Language). Das Audioformat des SNES heißt SPC, benannt nach dem in der Konsole verbauten Soundchip SPC700. Obwohl eine SPC Datei das Audioformat ist, wird diese nicht für das Patchen oder Porten von Musik benötigt. Unser Ziel ist es, ein Textdokument in MML zu erstellen, das wir letztendlich mit AddMusicK in eine ROM patchen.

### Allgemeiner Ablauf

Wenn wir einen Song porten wollen, gibt es verschiedene Möglichkeiten dies zu tun. Diese sind:

- Einen Song vollständig in einer Textdatei in MML schreiben
- Eine MIDI Datei bearbeiten/erstellen, mit einem Konverter in eine Textdatei mit MML Code konvertieren und Textdatei weiter bearbeiten
- Eine SPC Datei mit einem Konverter in eine Textdatei mit MML Code konvertieren und Textdatei weiter bearbeiten

Die erste Methode ist verglichen zur zweiten aufwändiger und werden wir deshalb nur selten verwenden. Die dritte Methode ist eine Notlösung, da die bisher einzigen SPC Konverter sehr unsauber konvertieren. Die zweite Methode ist die gängigste die wir am häufigsten benutzen werden.

Zu Übungszwecken werden wir trotzdem im weiteren Verlauf einen kurzen Song vollständig im Texteditor schreiben.

### Einschränkungen

AddMusicK und der SPC700 Chip kommen nicht ohne Einschränkungen daher. Diese sollten wir kennen, bevor wir einen Song schreiben um zu verstehen, wie dieser auszusehen hat.

- 8 Soundkanäle: Der SPC700 besitzt 8 Kanäle die wir belegen können. Diese werden in AddMusicK mit #0 bis #7 durchnummeriert. Pro Kanal kann maximal eine Note gleichzeitig gespielt werden, wodurch wir maximal 8 Noten zur selben Zeit im gesamten Song platzieren können.
- Arbeitsspeicher: Der SPC700 besitzt 64kB Audio RAM. Ein Song muss vollständig in den Arbeitsspeicher geladen werden können.
- Oktaven: AddMusicK verwendet einen Bereich von knapp 6 Oktaven, angefangen mit o1c (C Note der 1. Oktave) bis o6a (A Note der 6. Oktave). Wichtig hierbei ist nur der Bereich den wir verwenden können und nicht die maximale Oktave. Wir können nämlich jedes einzelne Instrument quasi beliebig im Pitch ändern (Pitch bis 16 kHz) und somit auch in niedrigen Oktavräumen akustisch hohe Noten spielen.
- Auflösung: Die Zeitschritte sind quantisiert. Dies hat zur Folge, dass Noten und Pausen sowohl eine Mindestlänge besitzen als auch nur ein vielfaches dieser Mindestlänge lang sein und nur in einem festen Raster platziert werden können. Diese Mindestlänge nennt sich Tick.

### 1.3 Musiktheorie und MML Code Analogie

Noten befinden sich in Oktavräumen. Eine Oktave besteht aus 12 Halbtönen die wie folgt angeordnet werden: C, C#, D, D#, E, F, F#, G, G#, A, A#, B

Der Wechsel um genau eine Oktave bedeutet eine Frequenzverdopplung bzw Frequenzhalbierung. In MML geben wir die Oktavlage mit o, gefolgt von einer Zahl an. So beschreibt z.B. o3 den dritten Oktavraum.

Wird am Anfang des Kanals keine Oktave angegeben, wird diese von AddMusicK auf die vierte Oktave gelegt.

Eine Note besitzt eine Tonhöhe (z.B. C, D, E usw.) und einen Notenwert (ganze Note, halbe Note, Viertelnote usw.) der die Länge beschreibt. In MML wird eine Note gekennzeichnet, indem man zuerst die Tonhöhe, gefolgt von seiner Länge schreibt. c1 beschreibt ein C mit der Länge einer ganzen Note, ein e4 ist ein E mit der Länge einer Viertelnote.

C# D# F# G# und A# werden in MML mit einem + geschrieben, da # für eine Vielzahl von Befehlen reserviert ist. c+8 ist also eine c# Achtelnote.

Möchten wir eine Note beschreiben die eine andere Länge besitzt, können wir dies mit einem Punkt oder einer Tilde tun. (Eine weitere Möglichkeit ist das Gleichheitszeichen was einen Tick-Wert angibt, später dazu mehr) Eine Punktierte Note wird mit einem oder mehreren Punkten markiert. Jeder Punkt hinter einem Notenwert addiert den halben Notenwert dem vorherigen Notenwert dazu.

Beispiel:

c1. hat die Länge  $1 + 1/2$   
c1.. hat die Länge  $1 + 1/2 + 1/4$   
c8... hat die Länge  $1/8 + 1/16 + 1/32 + 1/64$

Mit Punktierten Noten können aber nicht beliebige Längen dargestellt werden. Dafür können wir einen Haltebogen (^) verwenden. Dieser addiert alle darauf folgenden Notenwerte zusammen.

Beispiel:

c1^4 hat die Länge  $1 + 1/4$   
c2^4^16 hat die Länge  $1/2 + 1/4 + 1/16$

Wenn wir Triolen (Noten zur Basis 3) schreiben wollen, haben wir auch hier mehrere Möglichkeiten. Ohne Triolen könnten wir beispielsweise nicht mit 3 gleich langen Noten einen 4/4 Takt vollständig ausfüllen. Eine Möglichkeit ist den Notenwert als Zahl zu schreiben.

Beispiel:

c6 c6 c6 c6 c6 c6 sind 6 Sechstelnoten, die zusammen eine ganze Note lang sind.

Noten, die in {} stehen, werden mit dem Wert  $2/3$  multipliziert, ähnlich wie eine Triole in der Musiktheorie beschrieben wird. Das vorherige Beispiel kann man auch schreiben als:

{c4 c4 c4 c4 c4 c4}

Aus den Viertelnoten werden so Sechstelnoten ( $1/4 \cdot 2/3 = 1/6$ )

Pausen werden mit einem r (Rest) gekennzeichnet, gefolgt von einer Länge

r1 ist eine Ganze Pause

r4 ist eine Viertelpause

r1<sup>8</sup> ist eine Pause Mit Länge  $1 + 1/8$

Wird keine Länge für eine Note oder Pause angegeben, wird diese von AddMusicK als Länge 1 interpretiert.

## 2 Ersten Song schreiben

Damit wir Musik porten können, benötigen wir zuallererst folgende Programme:

- Einen Texteditor unserer Wahl
- AddMusicK, um die zu portierende Textdatei in eine Rom zu patchen und zusätzlich eine SPC Datei zu kompilieren.
- Einen SPC Player (am besten SPC700 Player) zum abspielen von SPC Dateien.

In späteren Kapiteln werden noch weitere Programme benötigt.

Ein Song ist in 2 Teile aufgeteilt. Im ersten Bereich werden hauptsächlich Instrumente und Makros definiert. Alle Befehle die sich hier befinden sind so genannte Spezial Befehle (beginnen mit # – nicht zu verwechseln mit einem Kanal) und einige wenige Standard Befehle. Im zweiten Bereich befinden sich die Kanäle, die mit Noten und Hex-, sowie Standard Befehlen gefüllt sind.

### 2.1 Standard- und Spezialbefehle

Wir fangen nun ein neues Textdokument an. Wo sich dieses befindet ist egal, ich empfehle einen neuen Ordner custom anzulegen unter AddMusicK/music/ der neben dem Ordner originals liegt. Unser erster Song wird das populäre Prelude aus der Final Fantasy Reihe werden. Der Song soll parallel zu diesem Tutorial selbst geschrieben werden.

Damit wir ein Song porten können, muss dieser nur die zwei folgenden Dinge besitzen:

- Den Spezial Befehl #amk 2 – die 2 gibt an, welchen Song Parser AddMusicK benutzen soll. Dieser Wert kann sich mit einer neueren Version von AddMusicK ändern. AddMusicK schreibt den Befehl entweder am Anfang oder am Ende des Dokumentes auch selbstständig hin, falls der Befehl nicht im Textdokument steht.
- Den 1. Kanal, Kanal #0

### 2.1.1 Tempo und Lautstärke

Mit dem Standard Befehl `t` geben wir das Tempo an. Dieses wird nicht wie üblich in BPM (Beats per Minute, also Viertelnoten pro Minute) angegeben, lässt sich aber wie folgt umrechnen:  $BPM \cdot 0,4096 = tWert$

Unser Song besitzt einen BPM Wert von 78. Daraus wird also 31,9488. Weil nur Integer Werte (ganze Zahlen) verwendet werden können, runden wir das Tempo auf `t32` auf.

Mit dem Standard Befehl `w` geben wir die globale Lautstärke des Songs an. Der Wertebereich liegt bei `w0` bis `w255`. Wir legen diesen auf `w180`. Wird kein `w` Wert festgelegt, wird er von AddMusicK auf `w200` gesetzt. Beide Befehle müssen sich entweder über dem 1. Kanal oder direkt am Anfang des 1. Kanals befinden.

Im folgenden Bild ist eine so genannte Piano Roll zu sehen. Die höchste Note ist ein `o8c`, was zu hoch für die Rahmenbedingung ist. Wie vorher erwähnt ist unsere absolute Oktavenhöhe für die maximale Tonhöhe aber nebensächlich. Daher verschieben wir gedanklich alle Noten um 2 Oktaven nach unten. Somit beginnt die erste Note im zweiten Oktavenraum.

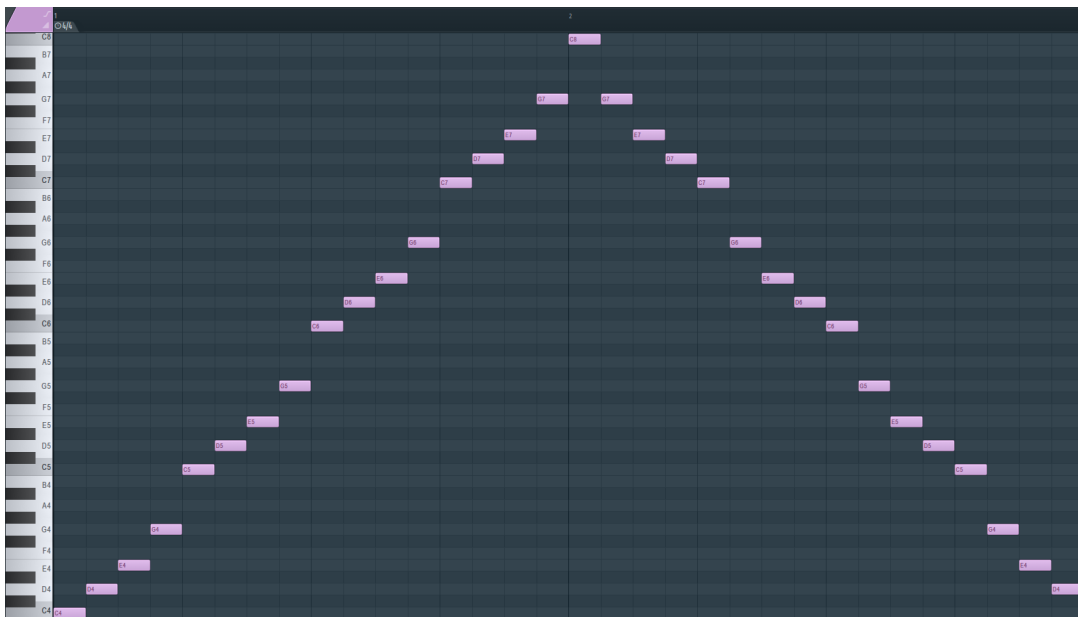


Abbildung 1: Notenabfolge A

Unser Dokument sieht bisher wie folgt aus:

```
1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2
```



Wir füllen nun den 1. Kanal mit Noten. Im Bild sehen wir die Noten der ersten 2 Takte (engl. Bar. Ein Takt besteht bei einem 4/4-Takt aus 4 Viertelnoten, bei einem 3/4-Takt aus 3 Viertelnoten usw.) Die einzelnen Noten sind daher 16tel Noten. Die gezeigte Notenfolge nennen wir zur Veranschaulichung A.

Nun schreiben wir die Noten in den Kanal: c16 d16 e16 g16

Ein Wechsel in einen anderen Oktavraum können wir entweder durch erneutes setzen des o Befehls erreichen oder durch > (Wechsel in den nächst höheren Raum) bzw < (Wechsel in den nächst niedrigeren Raum). Die bevorzugte Variante sollte > und < sein. Der Vorteil liegt darin, dass > und < den Oktavraum in Relation zum vorherigen angeben, während der o Befehl einen absoluten Wert festlegt. Durch das Ändern der ersten Oktave eines Kanals bewegen sich daher alle Noten mit dem gleichen Oktavabstand mit, während bei der Nutzung von o Befehlen jede Oktave manuell geändert werden müsste.

Für die ersten zwei Takte ergibt sich dadurch:

```
1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2
8 ; Notenfolge A (Takt 1 + 2)
9 c16 d16 e16 g16 > c16 d16 e16 g16
10 > c16 d16 e16 g16 > c16 d16 e16 g16
11 > c16 < g16 e16 d16 c16 < g16 e16 d16 c16
12 < g16 e16 d16 c16 < g16 e16 d16
```

Leerzeichen und Zeilenumbrüche werden von AddMusicK nicht interpretiert und dienen nur der besseren Übersicht. Mit einem Semikolon ; können Kommentare eingefügt werden. Ein Kommentar wird von AddMusicK vollständig ignoriert und nimmt keinen Platz im Arbeitspeicher ein.

### 2.1.2 Standardlänge

Mit dem l Befehl (length) kann die Standardlänge von Noten und Pausen definiert werden, also der Länge, die von AddMusicK angenommen wird, wenn keine Länge hinter einer Note bzw. Pause steht (normalerweise 1). Da alle Noten in diesem Kanal aus 16tel Noten bestehen, können wir den Code mit l16 übersichtlicher gestalten. Hierbei sei gesagt, dass der l Befehl nicht die Größe des Songs verringert, sondern nur der Übersicht dient. Das Textdokument sieht nun folgendermaßen aus:

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 l16
8
9 ; Notenfolge A (Takt 1 + 2)
10 c d e g > c d e g > c d e g > c d e g
11 > c < g e d c < g e d c < g e d c < g e d

```

Im folgenden Bild sehen wir die Noten der nächsten 2 Takte. Die Notenabfolge nennen wir B. Zu Übungszwecken soll diese selbst geschrieben werden.

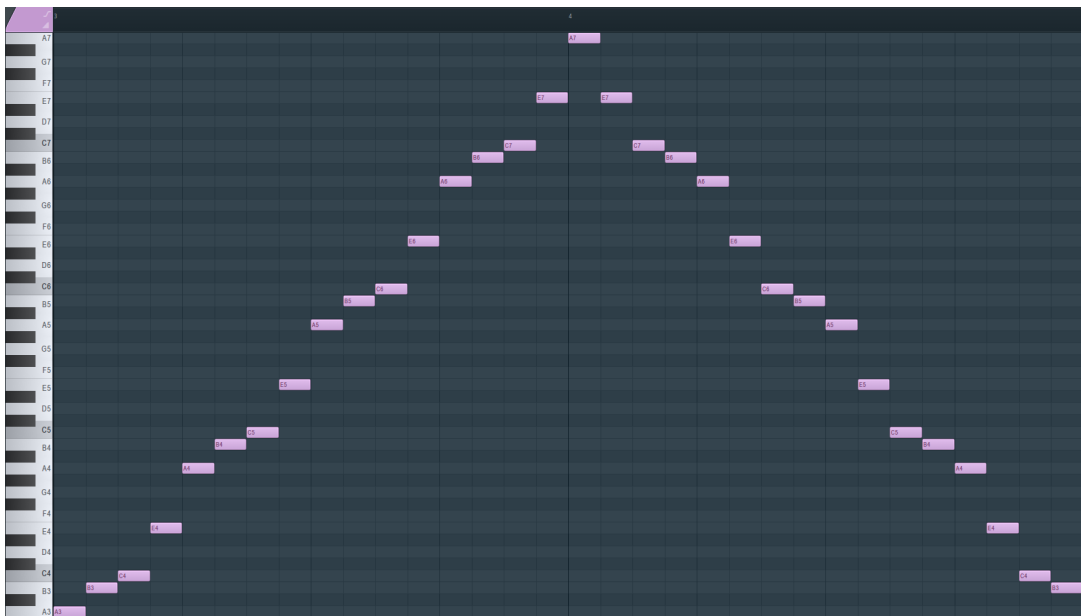


Abbildung 2: Notenabfolge B

Wenn alles geklappt hat, sieht der Song nun wie folgt aus:

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 l16
8
9 ; Notenfolge A (Takt 1 + 2)
10 c d e g > c d e g > c d e g > c d e g
11 > c < g e d c < g e d c < g e d c < g e d
12
13 ; Notenfolge B (Takt 3 + 4)

```

```

14 < a b > c e a b > c e a b > c e a b > c e
15 a e c < b a e c < b a e c < b a e c < b

```

Dies ist eine gute Gelegenheit, um den Song Probe zu hören. Dafür öffnen wir *AMKGUI.exe*, scrollen im unteren Fenster Local Songs so weit nach unten wie es geht und klicken auf den letzten Song. Als nächstes gehen wir auf *Add new Song* und wählen unsere Textdatei aus. Diese sollte jetzt markiert sein. Als nächstes setzen wir den Haken bei *Porter mode* und klicken auf *Run*. AddMusicK erzeugt aus der Textdatei eine SPC Datei die automatisch abgespielt wird, sofern der SPC Player als Standardprogramm für SPC Dateien gesetzt wurde. Falls dies nicht der Fall ist, wird eine Fehlermeldung erscheinen und wir müssen die SPC manuell öffnen. Eine SPC Datei wird nach dem Porten automatisch im Ordner SPCs abgelegt.

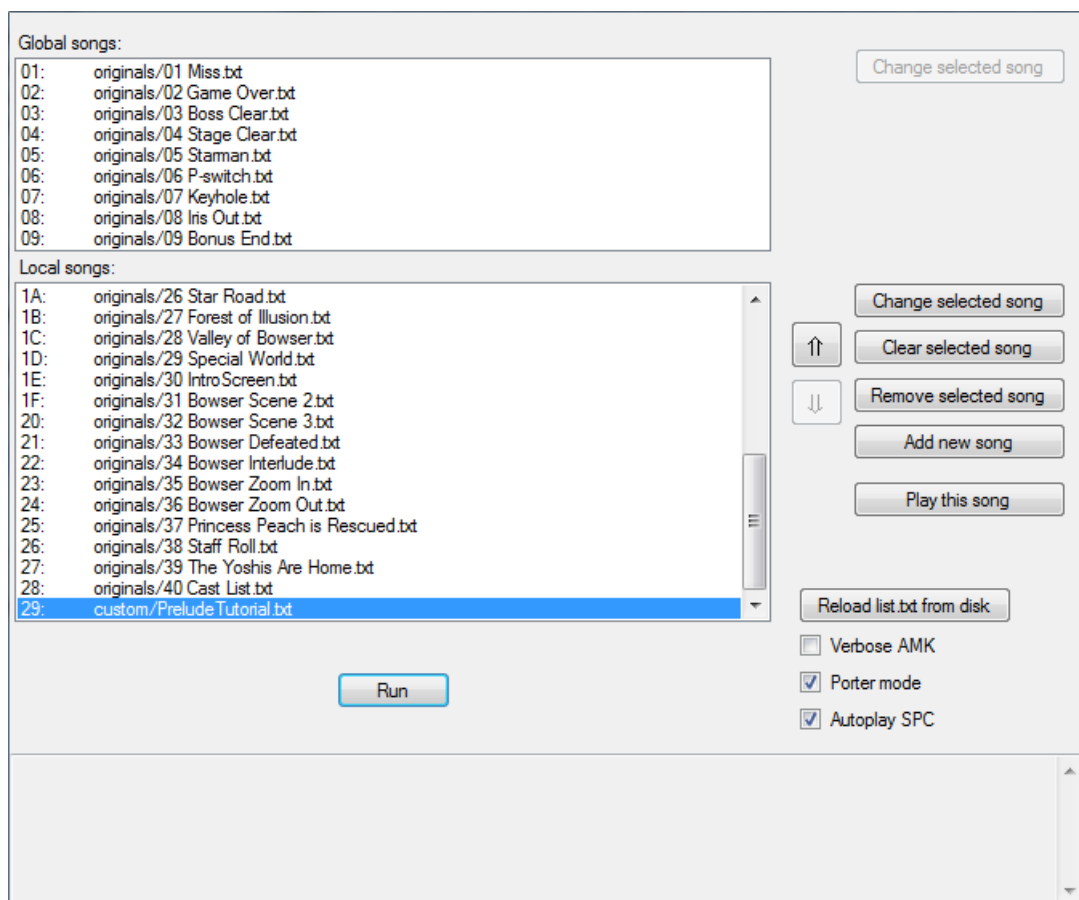


Abbildung 3: AMKGUI.exe

Als nächstes kommen wieder Notenfolge A und B. Anstatt die Notenfolgen zu kopieren, benutzen wir ein neues Werkzeug und zwar Loops.

### 2.1.3 Loops

Jede Note die wir schreiben vergrößert den Song. Mit Loops können wir wiederkehrende Notenabfolgen zusammenfassen, einmalig speichern und aufrufen wann wir wollen. Das Speichern eines Loops verbraucht zwar ebenso Speicherplatz, ist der Loop aber groß genug und wird mehrfach aufgerufen, kann man dadurch eine Menge Platz im Arbeitsspeicher sparen. Hierbei sei gesagt, dass man auf keinen Fall extrem kurze oder gar nur einzelne Noten in einen Loop speichern sollte, weil dies

den umgekehrten Effekt hat und den Song vergrößert.

Es gibt 3 Arten von Loops: Normale Loops, Labeled Loops und Superloops. Wir verwenden hier einen normalen Loop. Dieser speichert alles was sich in eckigen Klammern [] befindet in einem Loop ab, eine Zahl dahinter gibt an, wie oft dieser Loop gespielt werden soll. Unsere Notenabfolge A B A B wird dadurch zu [A B]2

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 116
8
9 [
10 ; Notenfolge A (Takt 1 + 2 + 5 + 6)
11 c d e g > c d e g > c d e g > c d e g
12 > c < g e d c < g e d c < g e d c < g e d
13
14 ; Notenfolge B (Takt 3 + 4 + 7 + 8)
15 < a b > c e a b > c e a b > c e a b > c e
16 a e c < b a e c < b a e c < b a e c < b >
17 ]2

```

Wichtig bei Loops im allgemeinen ist folgendes zu beachten: Ein sauberer normaler Loop hat immer gleich viele > und <. Dadurch ist sichergestellt, dass wir uns am Ende eines Loops in der gleichen Oktave befinden wie am Anfang des Loops. Ist dies nicht der Fall, laufen die Noten beim erneuten Durchlauf aus. Daher muss wie im Codebeispiel am Ende der Notenabfolge B ein > platziert werden. Täten wir dies nicht, würde beim zweiten Durchlauf des Loops das erste C nicht in der zweiten Oktave beginnen, sondern in der ersten, weil das die letzte Oktave war in der wir uns befanden als das letzte b gespielt wurde.

Als nächstes kommen die Notenabfolgen C, D, E und F. Wer mag, kann diese zu Übungszwecken selbst schreiben, die Bilder dazu befinden sich im Anhang.

Der Song sieht nun wie folgt aus:

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 116
8 [
9 ; Notenfolge A (Takt 1 + 2 + 5 + 6)
10 c d e g > c d e g > c d e g > c d e g
11 > c < g e d c < g e d c < g e d c < g e d
12
13 ; Notenfolge B (Takt 3 + 4 + 7 + 8)

```

```

14 < a b > c e a b > c e a b > c e a b > c e
15 a e c < b a e c < b a e c < b a e c < b >
16 ]2
17
18 ; Notenfolge C (Takt 9 + 10)
19 a > c f g a > c f g a > c f g a > c f g
20 a g f c < a g f c < a g f c < a g f c
21
22 ; Notenfolge D (Takt 11 + 12)
23 < b > d g a b > d g a b > d g a b > d g a
24 b a g d < b a g d < b a g d < b a g d
25
26 ; Notenfolge E (Takt 13 + 14)
27 < g+ > c d+ g g+ > c d+ g g+ > c d+ g g+ > c d+ g
28 g+ g d+ c < g+ g d+ c < g+ g d+ c < g+ g d+ c
29
30 ; Notenfolge F (Takt 15 + 16)
31 < a+ > d f a a+ > d f a a+ > d f a a+ > d f a
32 a+ a f d < a+ a f d < a+ a f d < a+ a f d

```

Diese gesamte Notenabfolge A B A B C D E F bzw. [A B]2 C D E F wird 3 mal hintereinander abgespielt. Es bietet sich daher an, einen weiteren Loop zu verwenden. Einen normalen Loop können wir aber nicht in einen anderen normalen Loop verschachteln. Dafür gibt es Superloops, gekennzeichnet durch doppelte Eckige Klammern [[]]. Dadurch wird unser Song zu [[ [A B]2 C D E F ]]3.

In eine weitere Ebene zu verschachteln ist nicht möglich, es können jedoch beliebig viele normale und labeled Loops parallel in einem Superloop zusammengefasst werden.

Nachdem der Superloop platziert wurde, sieht unser Song wie folgt aus:

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 116
8 [[
9 [
10 ; Notenfolge A (Takt 1 + 2 + 5 + 6)
11 c d e g > c d e g > c d e g > c d e g
12 > c < g e d c < g e d c < g e d c < g e d
13
14 ; Notenfolge B (Takt 3 + 4 + 7 + 8)
15 < a b > c e a b > c e a b > c e a b > c e
16 a e c < b a e c < b a e c < b a e c < b >
17 ]2
18
19 ; Notenfolge C (Takt 9 + 10)

```

```
20 a > c f g a > c f g a > c f g a > c f g
21 a g f c < a g f c < a g f c < a g f c
22
23 ; Notenfolge D (Takt 11 + 12)
24 < b > d g a b > d g a b > d g a b > d g a
25 b a g d < b a g d < b a g d < b a g d
26
27 ; Notenfolge E (Takt 13 + 14)
28 < g+ > c d+ g g+ > c d+ g g+ > c d+ g g+ > c d+ g
29 g+ g d+ c < g+ g d+ c < g+ g d+ c < g+ g d+ c
30
31 ; Notenfolge F (Takt 15 + 16)
32 < a+ > d f a a+ > d f a a+ > d f a a+ > d f a
33 a+ a f d < a+ a f d < a+ a f d < a+ a f d
34 ]]3
```

Alle Noten für Kanal #0 sind gesetzt. Wir können diesen Kanal nun weiter bearbeiten.

#### 2.1.4 Panning

Panning (oder kurz Pan) bezeichnet die Aufteilung der Lautstärke auf unterschiedliche Lautsprecher. Der entsprechende Befehl in MML ist y. Gültige Werte liegen zwischen 0 und 20, dabei gilt:

y0 : 100% Lautstärke auf rechten Lautsprecher

y20 : 100% Lautstärke auf linken Lautsprecher

y10 : 50% Lautstärke auf rechten und linken Lautsprecher (Standardwert)

Panning hilft uns dabei, einen Song räumlicher klingen zu lassen.

Es ist auch möglich, Panning kontinuierlich zu ändern. Dafür verwenden wir unseren ersten Hex-Befehl.

Hex-Befehle erkennt man daran, dass diese immer mit einem \$ beginnen, gefolgt von einem zweistelligen Hexwert. Der Befehl für das so genannte Pan Fading ist \$DC \$XX \$YY, wobei XX und YY Hexwerte sind, die das Fading weiter beschreiben. \$XX gibt die Dauer des Prozesses an, \$YY gibt den Panning Wert an, auf den geändert werden soll.

Wir möchten in Kanal #0 das Panning so haben, dass es mit 75% links und 25% rechts beginnt (y15) und nach einem Takt mit 75% rechts und 25% links aufhört (y5). Danach soll das Panning für den nächsten Takt den genau entgegengesetzten Verlauf haben, sodass wir nach insgesamt 2 Takten wieder bei y15 sind. Dieser Vorgang soll sich in Kanal #0 durch den gesamten Song ziehen.

Als erstes müssen wir unseren Startwert festlegen, also y15. Danach kommt der erste Hexbefehl \$DC. Als nächstes kommt die Dauer. 1 Takt bzw. eine ganze Note als Hexwert ausgedrückt ist \$C0. Diesen Wert kann man entweder ausrechnen (siehe Ticks) oder in der entsprechenden Tabelle im Anhang nachschauen. Unser Zielwert ist 5. Der erste Befehl um die Lautstärke vom linken zum rechten Lautsprecher zu verschieben ist somit fertig definiert: \$DC \$C0 \$05.

Wollen wir wieder auf y15 zurück, machen wir dies mit \$DC \$C0 \$0F, wobei 15 als Hexwert \$0F ist. Wer Probleme beim Umrechnen von Dezimalzahlen zu Hexwerten

hat kann auch einen Taschenrechner benutzen und diesen auf Programmierer einstellen. Die beiden Befehle schreiben wir nun abwechselnd nach jeweils einem Takt in den ersten Kanal.

Der Song sollte nun folgendermaßen aussehen:

```

1 #amk 2
2
3 t32
4 w180
5
6 #0
7 o2 116 y15
8 [[
9 [
10 ; Notenfolge A (Takt 1 + 2 + 5 + 6)
11 $DC $C0 $05 c d e g > c d e g > c d e g > c d e g
12 $DC $C0 $0F > c < g e d c < g e d c < g e d c < g e d
13
14 ; Notenfolge B (Takt 3 + 4 + 7 + 8)
15 $DC $C0 $05 < a b > c e a b > c e a b > c e a b > c e
16 $DC $C0 $0F a e c < b a e c < b a e c < b a e c < b
17 ]2
18
19 ; Notenfolge C (Takt 9 + 10)
20 $DC $C0 $05 a > c f g a > c f g a > c f g a > c f g
21 $DC $C0 $0F a g f c < a g f c < a g f c < a g f c
22
23 ; Notenfolge D (Takt 11 + 12)
24 $DC $C0 $05 < b > d g a b > d g a b > d g a b > d g a
25 $DC $C0 $0F b a g d < b a g d < b a g d < b a g d
26
27 ; Notenfolge E (Takt 13 + 14)
28 $DC $C0 $05 < g+ > c d+ g g+ > c d+ g g+ > c d+ g g+
29 > c d+ g
30 $DC $C0 $0F g+ g d+ c < g+ g d+ c < g+ g d+ c < g+ g d+ c
31
32 ; Notenfolge F (Takt 15 + 16)
33 $DC $C0 $05 < a+ > d f a a+ > d f a a+ > d f a a+ > d f a
34 $DC $C0 $0F a+ a f d < a+ a f d < a+ a f d < a+ a f d
35 ]]3

```

### 2.1.5 Instrumente setzen

Instrumente werden mit @ ausgewählt, gefolgt von einer Nummer. Mit 0 - 18 und 21 - 29 werden Instrumente bestehend aus Super Mario World Samples mit bestimmten Voreinstellungen angewählt. Instrumentnummern 30+ sind Instrumente, die in #instruments definiert wurden. Dabei ist es egal, ob die dafür verwendeten Samples aus SMW kommen oder eigene – sogenannte custom Samples – verwendet werden. Das Standardinstrument ist @0 was automatisch gewählt wird, wenn in einem Kanal

kein Instrument platziert wird. Eine vollständige Liste der SMW Samples befindet sich im Anhang.

Zum Testen empfehle ich in Zeile 7 verschiedene Instrumente auszuwählen, z.B. @2, @3 oder @5 und diese anzuhören.

Das Sample des Standardinstruments @0 soll beibehalten werden, allerdings ändern wir dessen ADSR Werte (Attack, Decay, Sustain, Release). Dafür gibt es zwei Möglichkeiten. Entweder mit dem Hex-Befehl \$ED oder mit dem Spezial Befehl #instruments. Wir verwenden letzteres, die genaue Funktionsweise und Unterschiede werden im Kapitel ADSR weiter erläutert.

Wir definieren uns nun ein neues Instrument auf Basis des Samples von @0 mit anderen Eigenschaften. Da es sich hierbei um einen Spezial Befehl handelt, kommt dieser in den ersten Teil des Dokuments. Das neu definierte Instrument rufen wir mit @30 auf.

```

1 #amk 2
2
3 #instruments
4 {
5     @0 $9E $D5 $B8 $06 $00 ; @30
6 }
7
8 t32
9 w180
10
11 #0
12 o2 l16 y15 @30

```

Wir bauen unseren Song nun weiter. Im Folgenden Bild sehen wir einige Begleitakkorde.

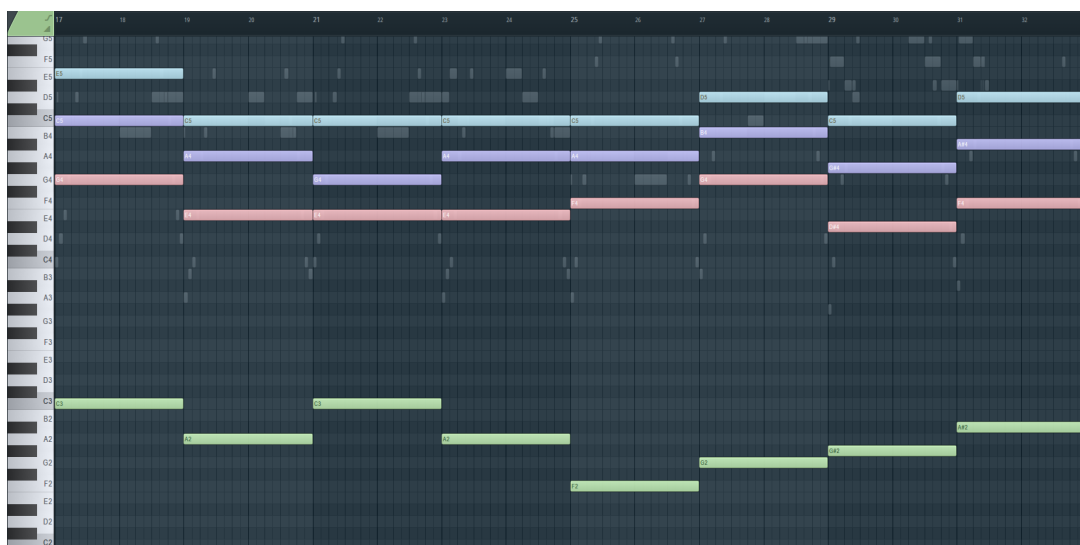


Abbildung 4: Kanal 2 bis 5

Der erste Kanal ist vollständig und wird nicht weiter bearbeitet. Parallel zum 1. Kanal folgt nach 16 Takten eine Begleitung. Die Noten erstrecken sich jeweils über



2 Takte und haben deshalb einen Notenwert von  $1^{\wedge}1$ . Pro Kanal kann aber nur eine Note gleichzeitig platziert werden. Deshalb brauchen wir für diese Begleitung 4 Kanäle. Die Noten ordnen wir auf die Kanäle wie folgt zu:

Kanal #1 blaue Noten, angefangen bei Oktave 5

Kanal #2 lila Noten, angefangen bei Oktave 5

Kanal #3 rote Noten, angefangen bei Oktave 4

Kanal #4 grüne Noten, angefangen bei Oktave 3

Die Begleitung soll außerdem ein weiteres mal wiederholt werden, sodass sie von Takt 17 bis 32 und dann nochmal von Takt 33 bis 48 gespielt wird. Zu Übungszwecken sollen diese 4 Kanäle selbst geschrieben werden.

Kanal #1 bis #4 sollten wie folgt aussehen:

```
1 #1
2 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
3 o5
4 [e1^1 c1^1 c1^1 c1^1 c1^1 d1^1 c1^1 d1^1]2
5
6 #2
7 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
8 o5
9 [c1^1 < a1^1 g1^1 a1^1 a1^1 b1^1 g+1^1 a+1^1]2
10
11 #3
12 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
13 o4
14 [g1^1 e1^1 e1^1 e1^1 f1^1 g1^1 d+1^1 f1^1]2
15
16 #4
17 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
18 o3
19 [c1^1 < a1^1 > c1^1 < a1^1 f1^1 g1^1 g+1^1 a+1^1]2
```

Als nächstes definieren wir uns ein neues Instrument auf Basis des Samples von @1 für die vier Kanäle. Wir fügen es in #Instruments ein und können es mit @31 auswählen. Hierbei ist auf die Reihenfolge der Instrumente zu achten, weil diese die Nummerierung vorgibt. Das @30 und @31 hinter den Semikolons sind nur Kommentare.

```
1 #instruments
2 {
3     @0 $9E $D5 $B8 $06 $00 ; @30
4     @1 $F6 $E1 $00 $03 $00 ; @31
5 }
```

Wir weisen allen Noten in Kanal #1, #2, #3 und #4 das neue Instrument mit @31 zu und hören den Song probe. Nach ungefähr 47 Sekunden setzt unsere Begleitung ein.

### 2.1.6 Lokale Lautstärke

Wie wir bereits gelernt haben, wird mit dem `w` Befehl die globale Lautstärke des Songs eingestellt. Mit dem `v` Befehl können Noten der einzelnen Kanäle zusätzlich individuell eingestellt werden. Auch hier liegt der Wertebereich zwischen `v0` und `v255`.

Die Noten in den vier neuen Kanälen stellen wir etwas leiser ein, indem wir in jeden der Kanäle ein `v220` schreiben. Zusätzlich dazu stellen wir das Panning ein, damit die Akkorde ein wenig auf den Lautsprechern aufgeteilt werden.

Die Kanäle mit der Begleitung sehen nun so aus:

```
1 #1
2 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
3 o5 @31 v220 y8
4 [e1^1 c1^1 c1^1 c1^1 c1^1 d1^1 c1^1 d1^1]2
5
6 #2
7 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
8 o5 @31 v220 y9
9 [c1^1 < a1^1 g1^1 a1^1 a1^1 b1^1 g+1^1 a+1^1]2
10
11 #3
12 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
13 o4 @31 v220 y11
14 [g1^1 e1^1 e1^1 e1^1 f1^1 g1^1 d+1^1 f1^1]2
15
16 #4
17 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
18 o3 @31 v220 y12
19 [c1^1 < a1^1 > c1^1 < a1^1 f1^1 g1^1 g+1^1 a+1^1]2
```

Als letztes fehlt noch ein weiteres Instrument was in Kanal #5 zum Einsatz kommen soll. Wir definieren es wie folgt:

```
1 #instruments
2 {
3     @0 $9E $D5 $B8 $06 $00 ; @30
4     @1 $F6 $E1 $00 $03 $00 ; @31
5     @13 $CC $E7 $B2 $06 $00 ; @32
6 }
```

Da Kanal #5 extrem kurze Noten enthält die nur schwer ablesbar sind, kann unser letzter Kanal der Vollständigkeit halber einfach aus dem Codebeispiel übernommen werden.

Wie wir in Zeile 8 und 9 sehen, kann ein Instrument in einem Kanal beliebig oft gewechselt werden. Das ist auch notwendig für komplexere Songs um mehr als 8 Instrumente in einem Song benutzen zu können.

```

1 #5
2 o5 @32 v240
3 r1^1^1^1^1^1^1^1^1^1^1^1^1^1^1^1
4 [
5 c32 d32 c2... < b2 > d2 < b32 > c2.... d4 c4 < b4 > d4
6 c32 d32 c2... < b2 > d2 d8 e8 c2. e4 d4 c4 < b4
7 g32 a2.... g2 a2 b2. > c4 d2 g2
8 d+32 f8.. d+8 d8 c2 @31 g+4 g4 f4 d+4
9 @32 d+32 g8.. f8 d+96 f64^192 d+96 f64^192 d+16 d1.
10 ]2

```

An dieser Stelle wird an dem Song nicht mehr weiter geschrieben. Er soll aber dennoch abgespeichert werden, da wir in späteren Kapiteln diesen für andere Beispiele benutzen werden.

## 3 Song aus Midi konvertieren

Bisher haben wir manuell einen Song in einem Texteditor geschrieben. Für komplexere und längere Songs eignet sich das Konvertieren aus einer MIDI Datei. Ein Konverter übernimmt allerdings nur einen Teil unserer Arbeit. Die MIDI Datei muss vor dem Konvertiervorgang aufbereitet und das Ergebnis weiterhin im Texteditor bearbeitet werden.

### 3.1 Vergleich der Konverter

In diesem Kapitel werden drei Konverter vorgestellt, die alle Vor- und Nachteile mit sich bringen. Dabei handelt es sich um PetiteMM von gocha und loveemu, MIDI2MML von NeutronCat und mmltk von Nobody-86.

#### PetiteMM

PetiteMM ist der mit Abstand älteste der drei Konverter mit einem simplen GUI.

##### Vorteile

- Java Programm lässt Benutzung auf Windows und MacOS zu
- Schneller Konvertiervorgang
- Konvertiert fast fehlerfrei

##### Nachteile

- Keine schöne MML Code Formatierung

#### MIDI2MML

MIDI2MML ist ein neuerer Konverter aus Ende 2020.

##### Vorteile

- Benutzerfreundliches GUI
- Schneller Konvertiervorgang
- Schöne MML Code Formatierung
- Überlappende Noten werden über 8 Kanäle hinaus in separate Kanäle abgespeichert
- MIDI Events sind einsehbar

##### Nachteile

- C# Programm, daher nur auf Windows zu benutzen
- Konverter interpretiert gewisse Notenwerte falsch
- Einige Notenwerte werden nicht intuitiv dargestellt.

## mmltk

mmltk ist ein weiterer neuerer Konverter aus Ende 2020, in Python geschrieben und als einziger Konsolenbasiert.

### Vorteile

- Schöne MML Code Formatierung
- Überlappende Noten werden über 8 Kanäle hinaus in separate Kanäle abgespeichert
- Einziger Konverter der konstant fehlerfrei konvertiert
- Kann zusätzlich MML in eine MIDI Datei zurück konvertieren (experimentell)

### Nachteile

- Programm momentan nur auf Windows zu benutzen
- Langsamer Konvertiervorgang
- Einmalige Installation aufwändiger als bei den anderen Konvertern

Da mmltk bei weitem das beste Ergebnis liefert, empfehle ich an dieser Stelle den einmaligen Mehraufwand der Installation.

## 3.2 Installation und Benutzung von mmltk

mmltk basiert auf Python, der erste Schritt ist also Python zu installieren. Python kann hier heruntergeladen werden: <https://www.python.org/downloads/>

Als nächstes öffnen wir die Konsole und geben `python --version` ein um zu testen, ob Python korrekt installiert wurde. Daraufhin sollte die Konsole mit der Versionsnummer von Python antworten.



```
D:\Dokumente
λ python --version
Python 3.7.1

D:\Dokumente
λ
```

Abbildung 5: Versionsnummer von Python aufrufen

mmltk benötigt 3 weitere Pakete die wir nun installieren und zwar `mido`, `numpy` und `pandas`. Um die Pakete jeweils zu installieren, geben wir nacheinander

```
python -m pip install mido
python -m pip install numpy
python -m pip install pandas
```

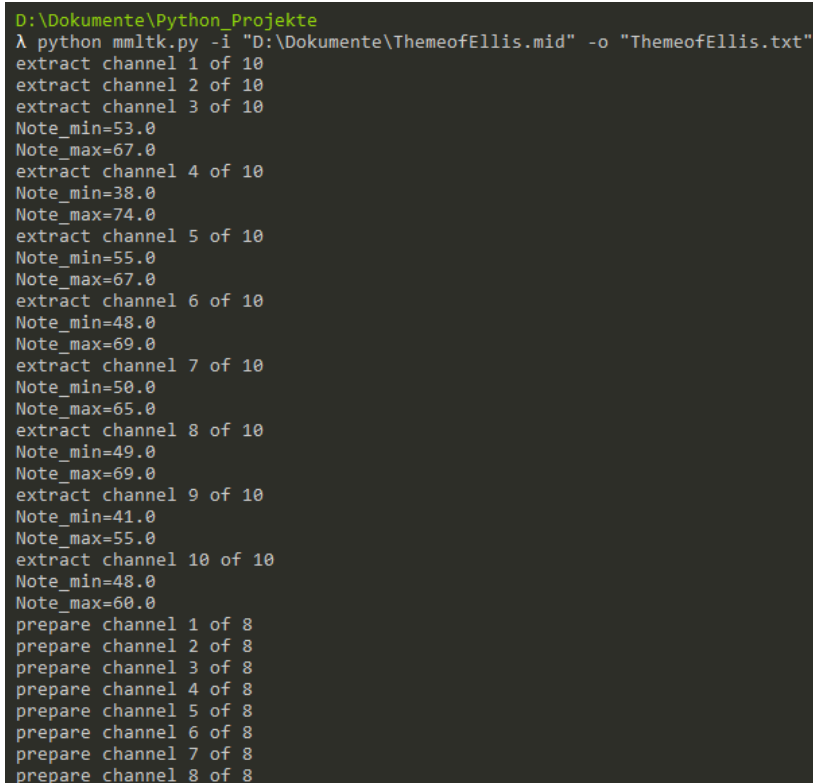
ein. Ab jetzt können wir mmltk benutzen. Es kann unter <https://gitlab.com/Nobody-86/mmltk> heruntergeladen werden.

Nachdem wir die Datei an einem Ort unserer Wahl entpackt haben, öffnen wir wieder die Konsole. Wir navigieren mit dem Befehl `cd` zu dem Verzeichnis, in dem `mmltk.py` liegt. Mit dem Befehl

```
python mmltk.py -i "MIDINAME.mid" -o "MMLNAME.txt"
```

wird eine Midi in ein MML Dokument konvertiert, die im selben Verzeichnis liegt wie `mmltk.py`.

Alternativ kann auch der Pfad angegeben werden, falls die Midi nicht im selben Verzeichnis liegt wie `mmltk.py`. Tipp: Per Drag & Drop wird automatisch der Dateipfad in die Konsole geschrieben.



```
D:\Dokumente\Python_Projekte
λ python mmltk.py -i "D:\Dokumente\ThemeofEllis.mid" -o "ThemeofEllis.txt"
extract channel 1 of 10
extract channel 2 of 10
extract channel 3 of 10
Note_min=53.0
Note_max=67.0
extract channel 4 of 10
Note_min=38.0
Note_max=74.0
extract channel 5 of 10
Note_min=55.0
Note_max=67.0
extract channel 6 of 10
Note_min=48.0
Note_max=69.0
extract channel 7 of 10
Note_min=50.0
Note_max=65.0
extract channel 8 of 10
Note_min=49.0
Note_max=69.0
extract channel 9 of 10
Note_min=41.0
Note_max=55.0
extract channel 10 of 10
Note_min=48.0
Note_max=60.0
prepare channel 1 of 8
prepare channel 2 of 8
prepare channel 3 of 8
prepare channel 4 of 8
prepare channel 5 of 8
prepare channel 6 of 8
prepare channel 7 of 8
prepare channel 8 of 8
```

Abbildung 6: MIDI mit `mmltk` in ein Textdokument mit MML konvertieren

### 3.3 Midi aufbereiten und konvertieren

Neben dem Konverter wird zusätzlich noch ein Programm benötigt, mit dem MIDIs (in einer Piano Roll) bearbeitet werden können. Beispiele dafür sind FL Studio, LMMS oder Anvil Studio. Die Trial Version von FL Studio reicht für unsere Zwecke vollkommen aus und beschränkt uns hinsichtlich des Arbeitens an Midi Dateien nicht. In diesem Tutorial benutze ich FL Studio und beschreibe den allgemeinen Umgang damit.

Wir öffnen eine beliebige MIDI Datei mit FL Studio, am einfachsten geht dies, indem die Datei per Drag & Drop in das offene Programm gezogen wird. Wichtig hierbei ist, dass FL Studio nur MIDI Dateien mit der Endung `.mid` und nicht `.midi` öffnen kann.

Das folgende Fenster wird sich öffnen:



Abbildung 7: MIDI mit FL Studio öffnen

Wir stellen den Channel type auf *MIDI Out*. Falls zur MIDI Datei eine Soundbank in Form einer DLS (Downloadable Sound) Datei vorliegt, kann auch *MIDI Out with Fruity LSD* ausgewählt werden. *FLEX* sollte vermieden werden, besonders wenn man nur die Trial Version besitzt und keine FL Studio Projekte öffnen kann. FLEX Kanäle können notfalls auch im Nachhinein noch durch MIDI Out Kanäle ersetzt werden.

Als nächstes muss der MIDI Port geändert werden und zwar auf den gleichen, den auch die MIDI Out Kanäle verwenden (standardmäßig auf 0). Mit F10 öffnen sich die MIDI Settings, dort stellen wir den Port auf 0.



Abbildung 8: MIDI Port wählen

Zusätzlich kann über den Reiter *Project* die Timebase auf 48 PPQ eingestellt werden. Dadurch sind alle Noten und Pausen im richtigen Maß quantisiert, wodurch garantiert ist, dass die zeitliche Auflösung nie überschritten wird.

Ab jetzt sind alle Vorkehrungen getroffen und die MIDI Datei kann nach den bereits erwähnten Einschränkungen bearbeitet werden (pro Kanal nur eine Note gleichzeitig, maximal 8 Kanäle, Noten zwischen den Oktaven o1c bis o6a). Zusätzlich sollte darauf geachtet werden, welche Noten in den Kanälen 7 und 8 liegen. Kanal 8 ist in SMW hauptsächlich für den Sprung Soundeffekt reserviert (und einige weitere die aber eher selten benutzt werden), Kanal 7 für die restlichen Soundeffekte. Das hat zur Folge, dass vor allem lange Noten und Noten, die für eine markante Melodie zuständig sind, nicht in diesen beiden Kanälen untergebracht werden sollten. Jedes mal wenn ein Soundeffekt in SMW abgespielt wird, werden die Noten des entsprechenden Kanals stumm geschaltet.

Die vollständigen Listen aller Soundeffekte befinden sich in den Ordnern 1DF9 (Kanal 7) und 1DFC (Kanal 8).

Entspricht die MIDI Datei den Anforderungen, sollte das Projekt als .mid exportiert werden falls man nicht die Trial Version von FL Studio besitzt, weil man keine Projektdaten öffnen kann, MIDI Dateien aber schon (wie wir offensichtlich gesehen haben).

Hier sei noch einmal gesagt, dass man darauf achten soll, nicht FLEX zu benutzen, da es passieren kann, dass die exportierte MIDI Datei keine Noteninformationen enthält und alle Arbeiten verloren gehen.

## 4 Instrumente und Samples

In diesem Kapitel wird erklärt, was der Unterschied zwischen so genannten sampled und unsampled Songs ist, wie einzelne Instrumente definiert werden, was Sample Groups sind und wie wir eigene Samples erstellen und einbinden können.

Außerdem benötigen wir für das Kapitel folgende Programme:

- Audacity
- OpenMPT
- BRRPlayer von Vitor Vilela
- BRRTools von Bregalad und nyanpasu64 oder das C700 VST Plugin

Unsampled Songs sind Songs, die ausschließlich Samples aus SMW benutzen. Ein Beispiel für ein unsampled Song ist der von uns erstellte Song aus Kapitel 2.

Von sampled Songs ist die Rede, sobald mindestens ein custom Sample (ein Sample, das aus einem anderen Spiel kommt oder selbst erstellt wurde) für den Song verwendet wird.

Unsampled Songs klingen oft nach SMW und haben den Vorteil, dass sie nur wenig Platz im Audio RAM einnehmen. Dafür ist man jedoch stark eingeschränkt in der Instrumentauswahl. Sampled Songs haben dieses Problem nicht, können aber schnell den gesamten Arbeitsspeicher belegen, weswegen besonders auf den Speicherplatz geachtet werden muss.



## 4.1 Custom Samples

Der SPC700 arbeitet mit so genannten .brr Samples (Bit Rate Reduction). Es gibt verschiedene Möglichkeiten an diese Samples zu gelangen:

- Ein Sample Pack herunterladen
- Samples aus einer SPC Datei extrahieren, z.B. mit split700, dem C700 Plugin oder SPC2MML
- Ein Sample aus einer .wav Datei selbst erstellen

Ein sehr umfangreiches Sample Pack ist samples of insanity von musicalman. Es ist hier verfügbar: <https://bin.smwcentral.net/u/29022/soi-2019-07-12.zip>

Sollen Custom Samples für einen Song verwendet werden, müssen sich diese in einem Unterverzeichnis von AddmusicK/samples/ liegen. Am besten legt man für jeden Song einen eigenen Ordner für Samples an. Mit dem Spezial Befehl #path weiß AddMusicK, in welchem Verzeichnis Samples gesucht werden sollen.

Wir fügen am Beispiel unseres Songs aus Kapitel 2 ein Custom Sample ein. Als erstes erstellen wir einen Ordner mit dem Namen *Prelude* in AddmusicK/samples/. Danach kopieren wir uns das Sample Harp 2.brr, welches in samples of insanity/individual samples/other liegt und fügen es in den von uns zuvor erstellten Ordner *Prelude* ein. Außerdem öffnen wir das beiliegende Textdokument *!patterns.txt* worauf wir gleich noch zurückgreifen werden.

Jetzt öffnen wir wieder unseren Song und geben den Pfad mit #path "Prelude" an. Danach bestimmen wir mit #samples{}, welche Samples aus dem Ordner benutzt werden sollen. Wir tragen neben dem Samplename noch eine Samplegroup ein die wir verwenden möchten, was entweder die Samplegroup #default oder #optimized ist, falls wir keine eigene erstellen.

AddMusicK legt im Hintergrund die Samplegroup #default automatisch an wenn kein #Samples{} Block vorhanden ist, weswegen wir bisher keine Samplegroup selber anlegen mussten.

```
1 #amk 2
2
3 #path "Prelude"
4
5 #samples
6 {
7     #default
8     "harp 2.brr"
9 }
```

Als letztes definieren wir uns das neue Instrument. Wir ersetzen dafür das alte Instrument an die Position @30, indem die gesamte Zeile gelöscht wird und dafür der Samplename plus den ersten 5 Hexwerten aus *!patterns.txt* die hinter *harp 2.brr* stehen.

Der Anfang des Songs sollte nun wie folgt aussehen:

```
1 #amk 2
2
3 #path "Prelude"
4
5 #samples
6 {
7     #default
8     "harp 2.brr"
9 }
10
11 #instruments
12 {
13     "harp 2.brr"    $BF $B0 $00 $02 $00    ; @30
14     @1              $F6 $E1 $00 $03 $00    ; @31
15     @13             $CC $E7 $B2 $06 $00    ; @32
16 }
```

## 4.2 Instrumente definieren

Möglicherweise möchten wir ein Instrument in seinen Eigenschaften ändern oder haben gar keine Standardwerte, weil wir das Sample beispielsweise selbst erstellt haben. Was es mit den einzelnen Hexwerten auf sich hat wird im Folgenden erklärt.

```
1          "harp 2.brr"    $BF $B0 $00 $02 $00    ; @30
```

Als erstes wird entweder die Nummer des SMW Samples oder der Custom Sample Name angegeben, der definiert werden soll. Danach beschreiben die Hexwerte der Reihe nach:

Attack und Decay (AD), Sustain und Release (SR), Gain, Pitch, Fine Pitch.

ADSR und Gain beschreiben die Hüllkurve bzw. Anschlags- und Abklingverhalten des Instruments, wobei immer nur ADSR oder Gain gleichzeitig aktiv ist. Es ist aber möglich, mit weiteren Befehlen zwischen ADSR und Gain zu wechseln und auch die Werte zu verändern.

Mit Pitch und Fine Pitch stimmen wir das Instrument auf den richtigen Pitch, ähnlich wie eine Gitarre. Soll das Instrument um eine Oktave vergrößert oder verkleinert werden, müssen Pitch und Fine Pitch verdoppelt bzw. halbiert werden.

### 4.2.1 ADSR

Allgemein lassen sich verschiedene Parameter mit ADSR ansteuern, im Falle von AddMusicK allerdings nur der Verlauf der Lautstärke. Außerdem ist ADSR in AddMusicK leicht anders definiert als üblich. Das kommt daher, dass es beim SPC700 kein echtes Release Event gibt. Zwar gibt es so genannte Off Keying Events (immer wenn eine Note endet), diese lösen aber nicht das herkömmliche Release Event aus, weil das Instrument sofort aufhört zu spielen, sobald das Off Keying Event eintritt.

(Es gibt allerdings die Möglichkeit, ein künstliches Release Event nach Ende einer Note selbst zu erzeugen)

Die 4 Parameter beschreiben hier folgende Eigenschaften:

- **Attack (Anstieg)** – gibt die Dauer an, bis die Lautstärke das Maximum erreicht hat.
- **Decay (Abfall)** – gibt die Dauer an, die zwischen Maximum und Sustain Wert liegt.
- **Sustain (Halten)** – gibt das Verhältnis zwischen abfallenden Wert nach der Decay Dauer und dem Maximum an.
- **Release (Loslassen)** – gibt die Dauer an, bis die Lautstärke 0 erreicht.



Abbildung 9: ADSR

Wert	Attack (Sek.)	Decay (Sek.)	Sustain (Verhältnis)	Release (Sek.) Wert $\leq F$	Release (Sek.) Wert $> F$
00	4.1	1.2 (\$ED)	1/8	Unendlich	1.2
01	2.6	0.74 (\$ED)	1/8	38	0.88
02	1.5	0.44 (\$ED)	2/8	28	0.74
03	1.0	0.29 (\$ED)	2/8	24	0.59
04	0.64	0.18 (\$ED)	3/8	19	0.44
05	0.38	0.11 (\$ED)	3/8	14	0.37
06	0.26	0.074 (\$ED)	4/8	12	0.29
07	0.16	0.037 (\$ED)	4/8	9.4	0.22
08	0.096	1.2	5/8	7.1	0.18
09	0.064	0.74	5/8	5.9	0.15
0A	0.04	0.44	6/8	4.7	0.11
0B	0.024	0.29	6/8	3.5	0.092
0C	0.016	0.18	7/8	2.9	0.074
0D	0.01	0.11	7/8	2.4	0.055
0E	0.006	0.074	1	1.8	0.037
0F	0	0.037	1	1.5	0.018

Die Tabelle und die beiden unterschiedlichen Befehle zur ADSR Beschreibung benötigen eine genauere Erklärung. Diese erfolgt am Beispiel unseres Custom Instruments.

```
1 #instruments
2 {
3     "harp 2. brr"    $BF $B0 $XX $YY $ZZ ; @30
4 }
5                     $ED $3F $B0
```

Beide Befehle beschreiben die gleichen ADSR Werte. Als erstes sei gesagt, dass die Reihenfolge der ADSR Werte nicht \$AD \$SR, sondern \$DA \$SR entspricht.

Als nächstes fällt auf, dass sich der Decay Wert unterscheidet. Der Decay Wert in #instruments{} muss immer um 8 aufaddiert werden.

Sustain und Release Werte hängen voneinander ab. In diesem Beispiel ist die Release Dauer nicht unendlich, obwohl der Release Wert auf 0 steht. Für eine Release Dauer von unendlich muss der Sustain Wert immer eine gerade Zahl sein. Hier müsste der Sustain Wert auf A gesetzt werden, damit das Sustain Verhältnis gleich bleibt, die Release Dauer aber unendlich beträgt.

Falls \$DA in #instruments{} kleiner als \$80 ist, wird automatisch Gain aktiviert. Genauso wird Gain aktiviert, falls im \$ED Befehl der \$DA Wert  $\geq 80$  ist.

Das Instrument wird durch ADSR wie folgt beschrieben:

**Attack:** 0 Sek.

**Decay:** 0.29 Sek.

**Sustain:** 6/8 vom Maximum

**Release:** 1.2 Sek.

#### 4.2.2 Gain

Neben ADSR kann ein Instrument auch durch Gain beschrieben werden. Wie bei ADSR gibt es zwei Möglichkeiten, Gain zu definieren. Zum einen durch das dritte Argument in #instruments{} (wird dort aber nur aktiv, falls \$DA in #instruments{} kleiner als \$80 ist), zum anderen durch die Hexbefehle \$FA \$01 \$XX oder \$ED \$80+ \$XX.

Gain beschreibt keine vollständige Hüllkurve sondern nur das Anschlag- oder Abklingverhalten eines Instruments. Im Vergleich zu ADSR bietet Gain allerdings auch exponentielle Verläufe an, wohingegen durch ADSR nur lineare Verläufe möglich sind.

Es gibt fünf verschiedene Gain Modi:

- **Direct**
- **Decrease**
- **Exponential Decrease**
- **Increase**
- **Bent Line Increase**

Bei Gain Werten zwischen 00 bis 7F ist der Modus Direct aktiv (direkter Anschlag). Die weiteren Modi inklusive Anstiegs- und Abklingzeiten stehen in der folgenden Tabelle:

Zeit von Wert 7F bis 0:				Zeit von Wert 0 bis 7F:			
Wert	Decrease	Wert	Exp Decrease	Wert	Increase	Wert	Bent Line Increase
80	Unendlich	A0	Unendlich	C0	Unendlich	E0	Unendlich
81	4.1 s	A1	38 s	C1	4.1 s	E0	7.2 s
82	3.1 s	A2	28 s	C2	3.1 s	E2	5.4 s
83	2.6 s	A3	24 s	C3	2.6 s	E3	4.6 s
84	2.0 s	A4	19 s	C4	2.0 s	E4	3.5 s
85	1.5 s	A5	14 s	C5	1.5 s	E5	2.6 s
86	1.3 s	A6	12 s	C6	1.3 s	E6	2.3 s
87	1.0 s	A7	9.4 s	C7	1.0 s	E7	1.8 s
88	770 ms	A8	7.1 s	C8	770 ms	E8	1.3 s
89	640 ms	A9	5.9 s	C9	640 ms	E9	1.1 s
8A	510 ms	AA	4.7 s	CA	510 ms	EA	900 ms
8B	380 ms	AB	3.5 s	CB	380 ms	EB	670 ms
8C	320 ms	AC	2.9 s	CC	320 ms	EC	560 ms
8D	260 ms	AD	2.4 s	CD	260 ms	ED	450 ms
8E	190 ms	AE	1.8 s	CE	190 ms	EE	340 ms
8F	160 ms	AF	1.5 s	CF	160 ms	EF	280 ms
90	130 ms	B0	1.2 s	D0	130 ms	F0	220 ms
91	96 ms	B1	880 ms	D1	96 ms	F1	170 ms
92	80 ms	B2	740 ms	D2	80 ms	F2	140 ms
93	64 ms	B3	590 ms	D3	64 ms	F3	110 ms
94	48 ms	B4	440 ms	D4	48 ms	F4	84 ms
95	40 ms	B5	370 ms	D5	40 ms	F5	70 ms
96	32 ms	B6	290 ms	D6	32 ms	F6	56 ms
97	24 ms	B7	220 ms	D7	24 ms	F7	42 ms
98	20 ms	B8	180 ms	D8	20 ms	F8	35 ms
99	16 ms	B9	150 ms	D9	16 ms	F9	28 ms
9A	12 ms	BA	110 ms	DA	12 ms	FA	21 ms
9B	10 ms	BB	92 ms	DB	10 ms	FB	18 ms
9C	8 ms	BC	74 ms	DC	8 ms	FC	14 ms
9D	6 ms	BD	55 ms	DD	6 ms	FD	11 ms
9E	4 ms	BE	37 ms	DE	4 ms	FE	7 ms
9F	2 ms	BF	18 ms	DF	2 ms	FF	3.5 ms

Tabelle 1: Gain Table von ggamer77

Decrease und Exponential Decrease werden nur wirksam, wenn sie während einer gespielten Note aktiviert wird. Dies können wir mittels Haltebogen erreichen.

Im SPC700 Player werden die einzelnen Modi wie folgt angezeigt:

Direct	Ga D - -
Decrease	Ga A 0 0
Exponential Decrease	Ga A 0 1
Increase	Ga A 1 0
Bent Line Increase	Ga A 1 1

Die geänderten ADSR Werte durch \$ED bzw. Gain Wert durch \$FA \$01 oder \$ED werden mit erneutem Aufrufen des Instruments (@) wieder zurückgesetzt.

### 4.3 Sample Groups

Wer bereits häufiger Musik in eine Rom gepatcht oder sampled Songs selbst geschrieben hat, wird wahrscheinlich schon auf Sample Groups gestoßen sein.

Sample Groups sind – wie der Name schon vermuten lässt – eine vordefinierte Gruppe an Samples. AddMusicK stellt die Sample Groups #default und #optimized zur Verfügung. Diese beinhalten alle SMW Samples, die für Local Songs (normale Hintergrundmusik), Global Songs (Hintergrundmusik, die in allen Levels geladen sind, z.B. Stage Clear oder P-Block) und Soundeffekte verwendet werden. Sie befinden sich in *samples/default/* bzw. *samples/optimized/*.

Sofern kein #samples{} Block verwendet wird, setzt AddMusicK versteckt die Sample Group #default. Sobald #samples{} benutzt wird, muss eine Sample Group gesetzt werden. Ohne Sample Group wird nur das Standardinstrument @0 geladen, alle anderen SMW Samples können dann nicht mehr verwendet werden.

In eine Rom gepatcht hat dies außerdem zur Folge, dass weder Global Songs noch Soundeffekte richtig abgespielt werden. Das liegt daran, dass jeder Soundeffekt in SMW aus den gleichen Samples erzeugt werden, die auch für die Musikinstrumente benutzt werden. Stattdessen werden Custom Samples die im #samples{} Block liegen verwendet. Sobald ein sampled Song geschrieben wird, muss also eine Sample Group gesetzt werden.

Bei der Sample Group #default handelt es sich um die original SMW Samples, die Samples aus #optimized beinhalten nur die Hälfte der Informationen, wodurch diese auch nur halb so groß sind. Der Informationsverlust bedeutet eine geringere Soundqualität, diese ist aber kaum wahrnehmbar. Die Wahl der Sample Group hängt also nur davon ab, wie viel freier Speicher noch im Arbeitsspeicher vorhanden ist.

#### Custom Sample Groups

Es ist auch möglich eine eigene Sample Group anzulegen. Hauptmotivation ist, etwas Arbeitsspeicher einzusparen, indem einzelne Samples aus der Sample Group durch die Datei *EMPTY.brr* ersetzt werden. Dabei handelt es sich um ein Sample ohne Informationen und dient nur als Platzhalter, damit alle anderen Samples ihre richtige Position in der Liste beibehalten.

In *Addmusic\_sample\_groups.txt* werden alle Sample Groups definiert. Zwei Dinge fallen auf: Zum einen scheint es, als würden Samples fehlen. Die Samples @11, @16, @18, @23 - @28 werden jedoch nur durch die Samples aus der Liste erzeugt, die aber andere ADSR Einstellungen besitzen.

Zum anderen stehen hinter den Samples @14, @17 und @21 keine Ausrufezeichen. Ein Ausrufezeichen hinter einem Samplennamen bewirkt, dass diese global (also dauerhaft) geladen sind, solange die Sample Group verwendet wird. Die Samples @14, @17 und @21 sind die einzigen Samples, die weder für Global Songs, noch Soundeffekte und ausschließlich für einige Local Songs verwendet werden und daher nicht global geladen werden müssen.

Unter <https://www.smwcentral.net/?p=viewthread&t=97787> ist eine vollständige Liste, welche Samples für welche Soundeffekte und Global Songs verwendet werden. Die populärsten Samples die durch *EMPTY.brr* ersetzt werden sind *12 SMW @15.brr* (wird nur für Yoshi Sound verwendet), *09 SMW @7.brr* (wird nur für den Global Song Bonus End verwendet) und *13 SMW Thunder.brr*. Eine Sample Group ohne *12 SMW @15.brr* wäre daher nicht kompatibel mit Leveln, die Yoshi benutzen. Daher muss immer darauf hingewiesen werden, welche Sounds oder Global Songs nicht mehr funktionieren, wenn man einen Song teilt oder hochlädt.

Falls eine neue Sample Group angelegt werden soll, kopiert man dafür am einfachsten die Sample Group #optimized aus *Addmusic\_sample\_groups.txt*, fügt diese im Textdokument ein mit einem anderen Namen als optimized und ersetzt Samples, die nicht verwendet werden sollen durch *"EMPTY.brr"*!. Im Song selbst wird die Sample Group schließlich mit #SampleGroupName geladen.

## 4.4 Eigene Samples erstellen

Jede Audioquelle, die sich in eine .wav Datei konvertieren lässt, sind potentielle Samples die sich in Songs benutzen lassen. Als erstes öffnen wir Audacity und laden unsere Sounddatei ein (z.B. .mp3 oder .wav). Mit dem Selection Tool (*F1*) markieren wir nun alle Bereiche, die entfernt werden sollen und löschen diese mit *Entf*. Danach klicken wir links auf den kleinen Pfeil neben dem Dateinamen und wählen dort *Split Stereo to Mono* aus.

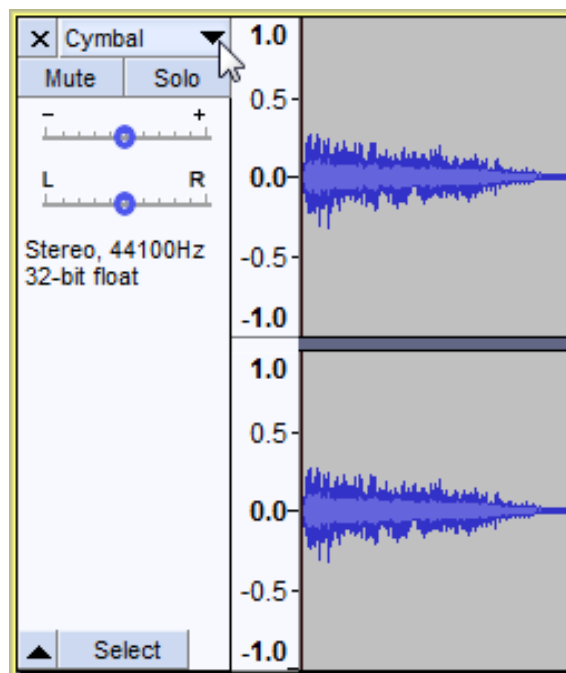


Abbildung 10: Track aufteilen in Audacity

Dadurch beugen wir vor, dass das Sample im nächsten Schritt nicht nur auf einem Lautsprecher zu hören ist. Außerdem wird dadurch die Größe des Samples ungefähr halbiert.

Es ist auch möglich, 2 Stereo Samples zu erzeugen. Dafür muss *Split Stereo Track* anstatt *Split Stereo to Mono* ausgewählt werden. Das hat allerdings den Nachteil, dass 2 Samples für einen Sound benötigt werden, was wiederum bedeutet, dass doppelt so viel Speicher verbraucht wird und im Song selbst 2 Kanäle belegt werden müssen.

Die Datei wird nun als .wav Datei exportiert mit einer Signed 16-bit PCM Codierung.

## OpenMPT

Als nächstes öffnen wir OpenMPT, laden unsere exportierte .wav ein und klicken oben auf den Reiter *Samples*. Es öffnet sich folgendes Fenster:

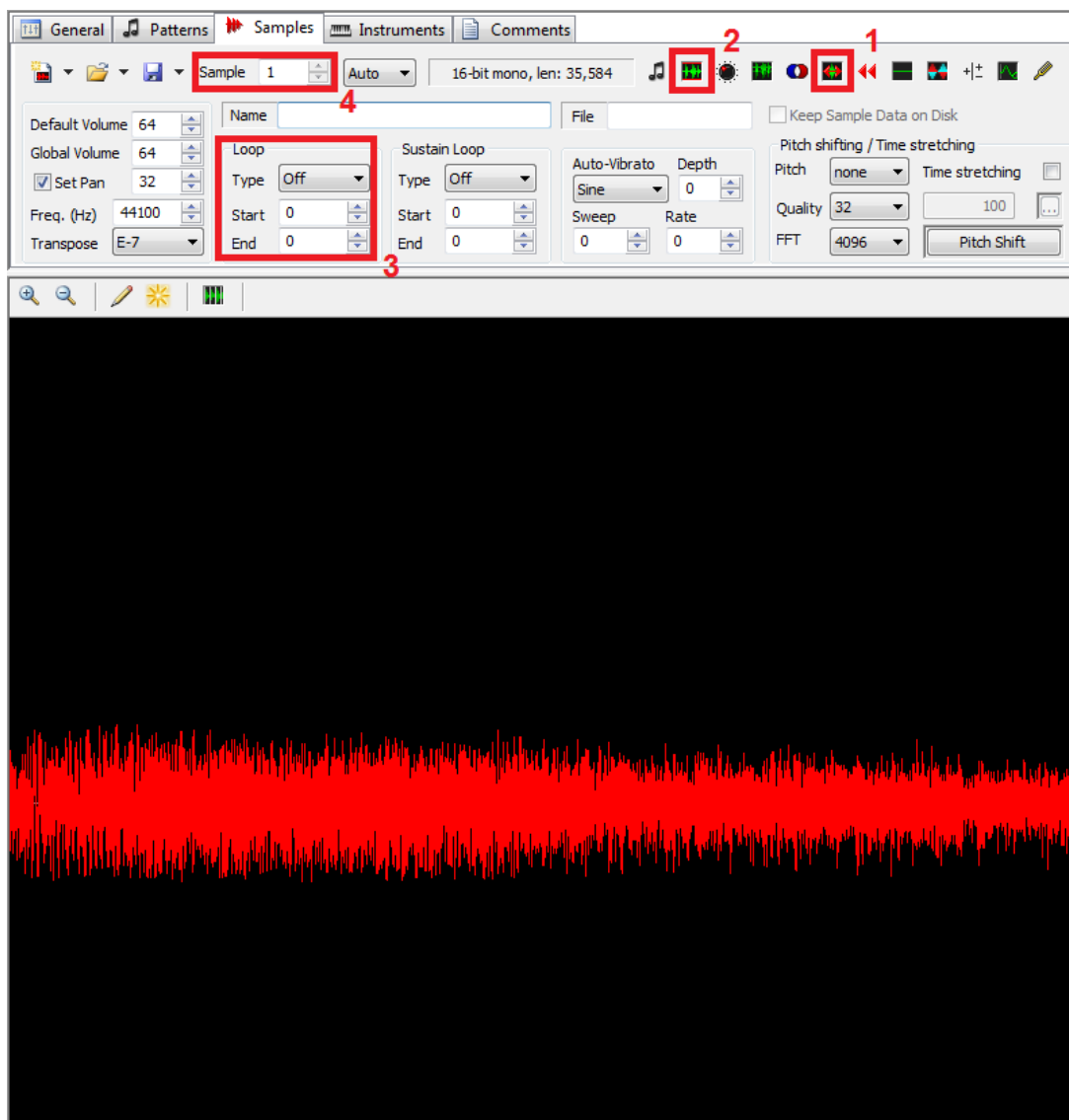


Abbildung 11: Samples bearbeiten in OpenMPT

Wir benötigen nur die 4 markierten Funktionen. Diese sind:



- **1. Up-/Down Sampling**
- **2. Normalize**
- **3. Loop Points**
- **4. Sample Auswahl**

Punkt 1 benutzen wir zum Downsamplen. Damit ist gemeint, dass die Samplerate – also die Abtastrate – verringert wird. Voreingestellt ist eine Samplerate von 44.1 kHz, halbieren wir diese durch downsamplen auf 22.05 kHz wird unser Signal nur noch mit der halben Anzahl an Samples (damit sind die Abtastpunkte gemeint) abgetastet. Die Konsequenzen sind, dass die Hälfte an Informationen gelöscht werden, was in ein halb so großes Signal mit Qualitätsverlust resultiert. Außerdem wird die akustisch wahrnehmbare Frequenz bei gleichem Pitch verdoppelt was wir an anderer Stelle ausnutzen können.

In der Regel sollte man ein- bis maximal zweimal downsamplen um das Sample klein genug zu bekommen.

Punkt 2 verstärkt unser gesamtes Sample im gleichen Maß ohne Clipping zu erzeugen (Abschneiden der Amplitude). Es ist immer einfacher, ein Sample im nachhinein leiser einzustellen als lauter, weshalb wir diese Option immer benutzen sollten.

Punkt 3 benötigen wir nur für Samples die loopen, also unendlich lang zu hören sind, solange eine Note nicht aufhört zu spielen. Dieser Punkt ist recht anspruchsvoll, da es nicht einfach ist, Start und Endpunkte zu finden, die ein Sample sauber loopen zu lassen. Besonders bei stark heruntergesampten Signalen fängt ein Sample schnell an zu "schwingen".

Wichtig hierbei ist, dass beide Looppunkte ein vielfaches von 16 sein müssen, falls wir im nächsten Schritt zum Konvertieren ein anderes Programm als BRRTools oder das C700 Plugin benutzen sollten.

Sustain Loop benötigen wir nicht. Dieses beschreibt einen Loop der verlassen wird, sobald das Sample nicht mehr gespielt wird und dadurch das Ende zu hören ist. Da es kein echtes Release Event in AddMusicK gibt und ein Sample mit Ende einer Note sofort aufhört zu spielen, kann in einem geloopten Sample der Rest des Signals hinter dem End Loop Punkt gelöscht werden, weil dieses nie erreicht wird.

Punkt 4 wird nur benötigt, falls wir vorher in Audacity das Sample auf 2 Stereo Tracks aufgeteilt haben. Hier kann zwischen den einzelnen Samples hin und her gewechselt werden.

Ist das Sample fertig bearbeitet, kann es über das Diskettensymbol neben Punkt 4 wieder als .wav Datei gespeichert werden.

Als letzten Schritt muss die .wav Datei nur noch in eine .brr Datei konvertiert werden. Dafür benutzen wir entweder das C700 Plugin (bevorzugt) oder BRRTools.

## **C700**

Das C700 Plugin muss in ein DAW (Digital Audio Workstation) unserer Wahl installiert werden. Unter anderem unterstützen FL Studio, LMMS und OpenMPT das Plugin.

Wir öffnen das Plugin in unserer DAW und legen per *Drag & Drop* die .wav Datei in die Mitte des Plugins. Danach klicken wir auf *Save Smpl...* woraufhin das Plugin eine .brr und eine .smpl Datei erstellt. Letztere kann gelöscht werden. Das erstellte .brr Sample kann nun über den #samples{} Block in einen Song geladen werden.

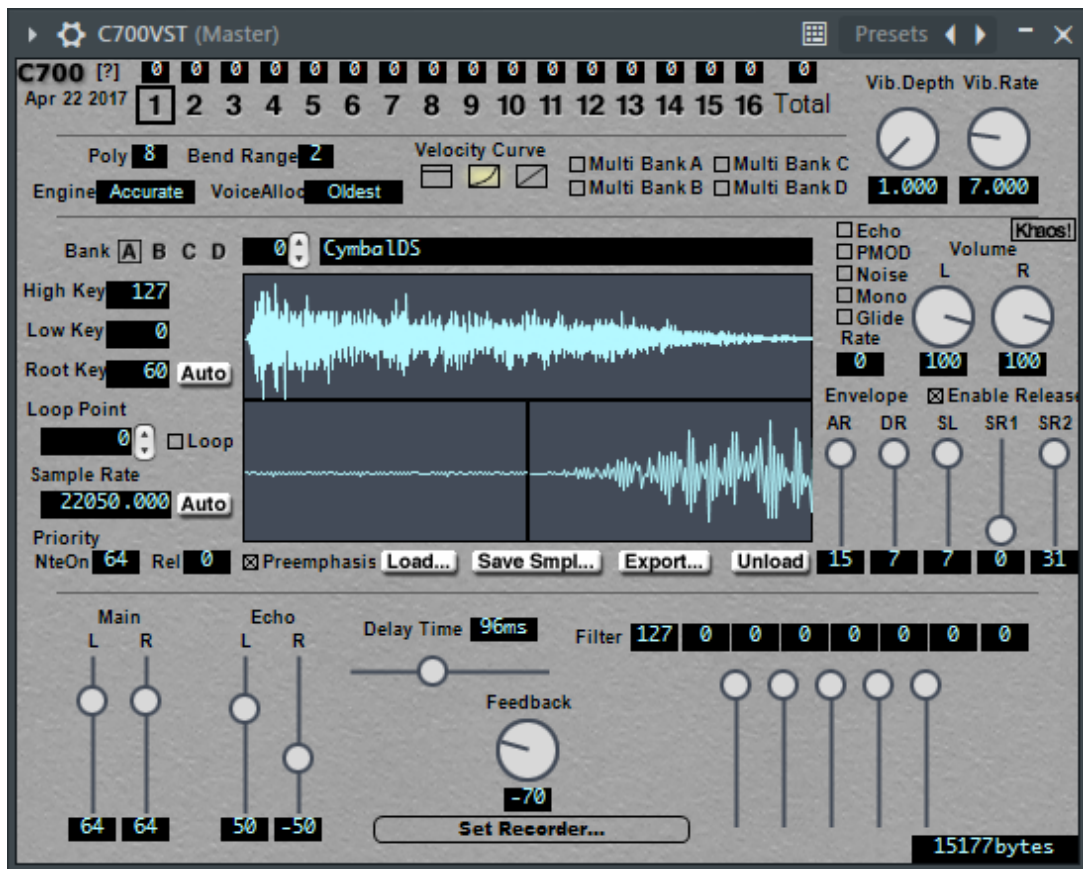


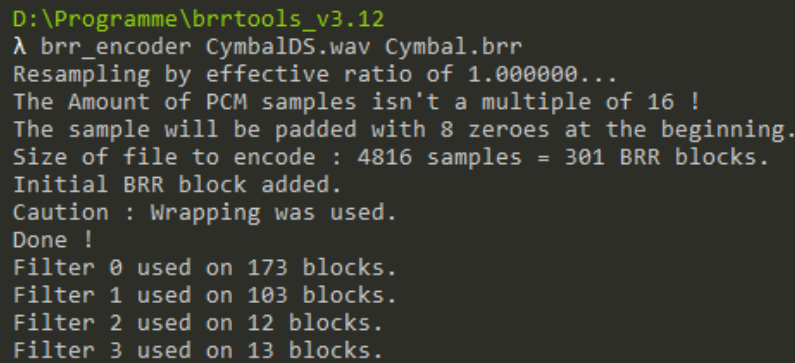
Abbildung 12: C700 VST Plugin in FL Studio

## BRRTools

Alternativ zu C700 kann auch BRRTools verwendet werden. BRRTools kommt mit 3 konsolenbasierten Programmen daher. Wir benötigen *brr\_encoder.exe*. Zunächst legen wir das Sample in das gleiche Verzeichnis wie *brr\_encoder.exe*, danach öffnen wir die Konsole und navigieren in das selbige Verzeichnis und geben folgendes ein:

```
brr_encoder Samplename.wav Samplename.brr
```

Hinter *brr\_encoder* können verschiedene Optionen hinzugefügt werden. Diese und weitere Erklärungen finden sich in der ReadMe.



```
D:\Programme\brertools_v3.12
λ brr_encoder CymbalD5.wav Cymbal.brr
Resampling by effective ratio of 1.000000...
The Amount of PCM samples isn't a multiple of 16 !
The sample will be padded with 8 zeroes at the beginning.
Size of file to encode : 4816 samples = 301 BRR blocks.
Initial BRR block added.
Caution : Wrapping was used.
Done !
Filter 0 used on 173 blocks.
Filter 1 used on 103 blocks.
Filter 2 used on 12 blocks.
Filter 3 used on 13 blocks.
```

Abbildung 13: brr encoder über die Konsole starten

## 5 Ticks, Song Stats und Hexbefehle

Dieses Kapitel beschäftigt sich rund um das Thema Hexwerte und Hexbefehle. Es werden die meisten, aber nicht alle Hexbefehle vorgestellt, es handelt sich hierbei jedoch um die wichtigsten.

Unter [AddmusicK/readme\\_files/hex\\_command\\_reference.html](#) befindet sich eine vollständige Liste aller Hexbefehle.

### 5.1 Ticks

Songs die wir mit AddMusicK schreiben sind in Notenwerten, Pausen und Events zeitlich quantisiert. Die Auflösung beträgt 48 PPQ (pulses per quarter note, deutsch Impulse pro Viertelnote) bzw. 48 TPQN (ticks per quarter note, deutsch Ticks pro Viertelnote). In FL Studio kann der PPQ Wert unter *Options/Project General Settings* eingestellt werden.

Der kleinste zeitliche Schritt – unabhängig vom Tempo – ist 1 Tick. Dieser hat die Länge einer 192stel Note. Zwischen klassischen Notenwerten und Tick Werten kann also über das Verhältnis 192/Notenwert umgerechnet werden. Anstatt einem Notenwert hinter einer Note oder Pause kann mit einem Gleichheitszeichen = ein Tick Wert angegeben werden. Beispiel:

$$c2 \equiv c=96 \text{ (Rechnung: } \frac{192}{c2} \text{)}$$
$$r4^{16} \equiv r=60 \text{ (Rechnung: } \frac{192}{r4} + \frac{192}{r16} \text{)}$$

Die Auflösung von 48 PPQ hat zwei Dinge zur Folge: Unsere kleinste Note ist eine 192stel Note, eine 256stel Note ist daher nicht möglich darzustellen. Außerdem wird beispielsweise eine 128stel Note nicht richtig aufgelöst, da diese bei 48 PPQ 1.5 Ticks lang ist ( $\frac{192}{128} = 1.5$ ). Stattdessen wird eine 128stel Note auf 1 Tick abgeschnitten, was wiederum einer 192stel Note entspricht.

Tick Werte treffen wir oft in Dateien an, die mit einem SPC/MML Konverter konvertiert wurden. Für extrem kurze Noten, beispielsweise um einen Swing Rhythmus zu erzeugen, können Tick Werte intuitiver sein als die klassischen Notenwerte.

Am häufigsten jedoch benutzen wir Tick Werte in Hexbefehlen. Sobald eine Dauer für einen Hexbefehl angegeben wird, ist diese ein Tick Wert als Hexzahl ausgedrückt.

Am Beispiel unseres Panning Befehls aus Kapitel 2 sehen wir, dass bei \$DC \$C0 \$05 die Länge \$C0 als Dezimalzahl 192 entspricht, was als Tick Wert wiederum eine ganzen Note ist.

Alternativ befindet sich im Anhang eine Tabelle mit einigen Längen die als Hexwerte dargestellt sind.

## **5.2 Stats**

## **5.3 Hexbefehle**

### **5.3.1 Pan Fading**

### **5.3.2 Tempo Fading**

### **5.3.3 Vol Fading**

### **5.3.4 Pitch Bendings**

### **5.3.5 Vibrato**

### **5.3.6 Tremolo**

### **5.3.7 Legato**

### **5.3.8 Light Staccato**

### **5.3.9 Echo**

### **5.3.10 Yoshi Drums**

## **6 Weitere Standardbefehle und Makros**

### **6.1 Loops**

### **6.2 Intros**

### **6.3 Makros**

### **6.4 Remote Codes**

## **7 Global Songs**

## 8 **FAQ**

## 9 Anhang

### Notenfolgen C - F Kapitel 2

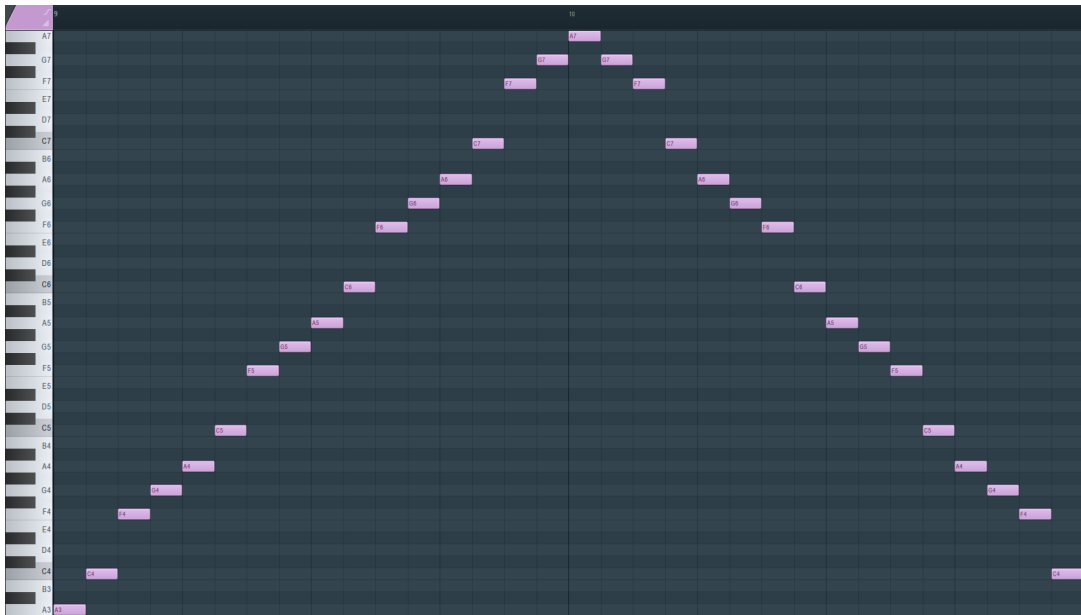


Abbildung 14: Notenabfolge C

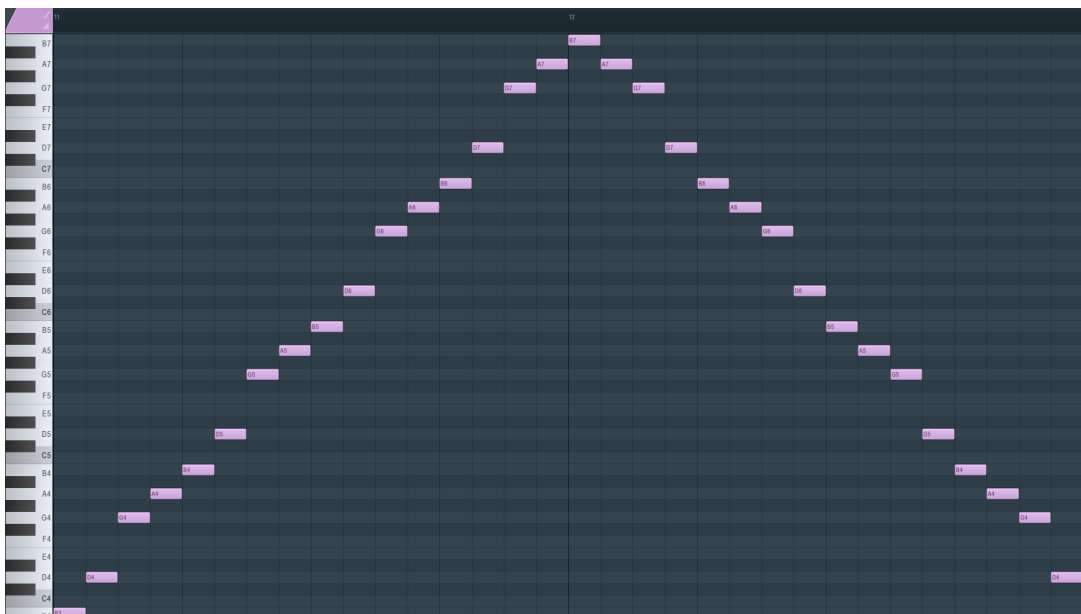


Abbildung 15: Notenabfolge D

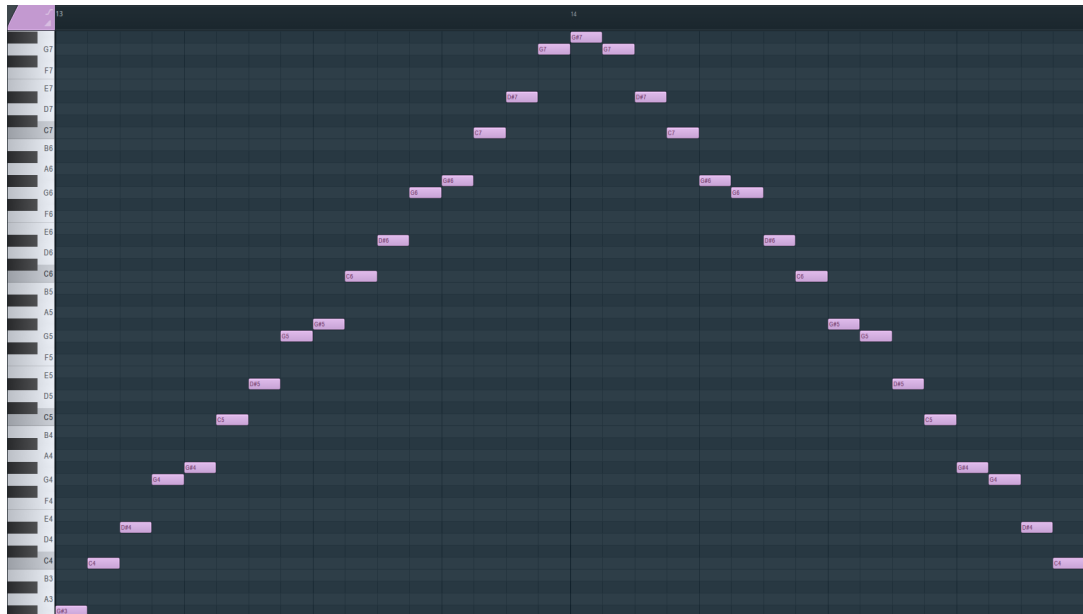


Abbildung 16: Notenabfolge E

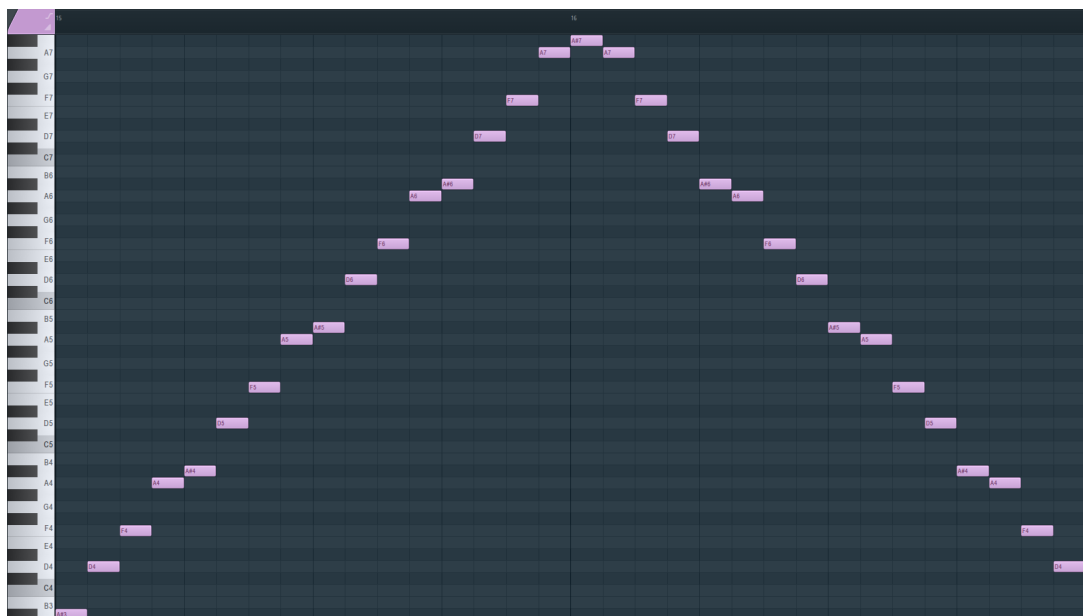


Abbildung 17: Notenabfolge F

\$C0	Ganze Note
\$80	Ganze Triole
\$60	Halbe Note
\$40	Halbe Triole
\$30	Viertelnote
\$20	Viertel Triole
\$18	Achtelnote
\$10	Achtel Triole
\$0C	16tel Note
\$09	16tel Triole
\$06	32tel Note
\$04	32tel Triole
\$03	64tel Note
\$02	64tel Triole

Tabelle 2: Längen in Hexwerte