Erstellung einer benutzerfreundlichen graphischen Benutzerschnittstelle zur Erzeugung wohlklingender periodischer Melodien

Paul V. Guigas (PG), Matr.-Nr. 70018 Dennis Debeye (DD), Matr.-Nr. 71040

Hochschule für Technik, Wirtschaft und Kultur

Zusammenfassung

Zielstellung war die Erarbeitung eines Melodien-Erzeugers. Unter Ausnutzung musikalischer Phänomene und Gesetzmäßigkeiten lassen sich leicht Ton- und Akkordfolgen konstruieren, die *gefällig* klingen. Mithilfe dieser Möglichkeiten wurde ein Werkzeug erstellt, das es einem musikalisch unerfahrenen Anwender erlaubt ohne große Einarbeitung über eine grafische Benutzeroberfläche *angenehm* empfundene Melodien und ggf. ganze Musikstücke zu generieren. - DD

Einleitung (DD)

Der Mensch ist schon seit Jahrtausenden technisch und geistig in der Lage, Musik zu machen. Bereits 150.000 v.Chr. hatte Musik einen festen Platz im Alltag des Menschen und stellte ein wichtiges Element bei der Durchführung von rituellen Zeremonien dar. Später, mit dem Aufkommen der ersten Hochkulturen, d.h. ab ca. 11.000 v.Chr., kam Musik in unterschiedlichen Funktionen und abhängig von der gesellschaftlichen Stellung ihrer Hörer zum Einsatz: Neben ihrer Verwendung im spirituellen Rahmen fand man Musik nun auch in Form von *Volksmusik* bei den einfachen Bauern und Arbeitern und als *Kunstmusik*, d.h. als Statussymbol in Palästen und an Höfen. (vgl. [ler]).
Ganz ähnlich stellte sich die Verbreitung von Musik noch im Mittelalter dar: Auch hier war ein wesentlicher Aspekt die Repräsentation von Wohlstand und Macht, da nur die gut Betuchten sich professionelle Musiker und teure, qualitativ hochwertige Instrumente leisten konnten. Die breite Bevölkerung kam - vor allem auf dem Land - hingegen nur selten in den Genuss von Musik. Wenn dies der Fall war, so wurde diese von fahrenden oder gering qualifizierten Gelegenheitsmusikanten aufgeführt. Die Weitergabe von Liedern war nur schwer möglich.
Trotz der immer größeren Verbreitung von Musik, auch für die breite Bevölkerung, war bis ins frühe 20. Jahrhundert hinein die einzige Möglichkeit der Darbietung die Liveaufführung. Dies änderte sich mit dem Aufkommen des Rundfunks und des Grammophons. Ab den 1920er Jahren wurden Schallplatten und Abspielgeräte auch für die einfache Bevölkerung erschwinglich. Gleichzeitig war die Tonqualität auf ein akzeptables Niveau gestiegen.
Im Laufe des 20. und mit dem Beginn des 21. Jahrhunderts wurde Musik dann mehr und mehr zum Medium der Massen. Zunächst löste die CD die Schallplatte als gängiges Trägermedium ab, mit der fortschreitenden Entwicklung der Digitaltechnik und dem Aufkommen des Internets war Musik schließlich nicht mehr an ein Trägermedium gebunden. Heute steht Milliarden von Menschen eine schier endlose Auswahl von Musiktiteln über Streaminganbieter jederzeit und überall auf der Welt zur Verfügung. (vgl. [wik])
Dieser kurze Überblick über die Geschichte verdeutlicht eindrucksvoll, wie sich der *Konsum* von Musik im Laufe der Jahrzehnte und Jahrhunderte geändert hat: Einst war er nur den Eliten (seien es Geistliche, Adlige oder Reiche) vorbehalten, so steht Musik heute einem Milliardenpublikum zur Verfügung. Gleichzeitig hat sich jedoch die *Produktion* von Musik weniger deutlich verändert: Zwar ist es ein beträchtlicher Unterschied, ob ein Musiker aus ästhetischen, religiösen oder schlicht aus effizienzsteigernden Gründen (Feld- oder Fabrikarbeit; zum Takt halten und steigern) musiziert, was aber allen gemein ist, ist, dass Talent und Wissen zur Erzeugung von Musik erforderlich sind. Ganz anders als beim Konsumenten, der für den einfachen Musikkonsum im Laufe der Geschichte immer weniger *mitbringen* musste, verkomplizierte sich die Rolle der Komponisten und Musiker sogar, da der Anspruch stetig stieg. So wurde das *Musikhören* zum Massenphänomen, während das *Musikmachen* einigen wenigen vorbehalten blieb.
Die von uns hier vorgestellte Software in Kombination mit einer einfachen grafischen Benutzeroberfläche soll

das musizieren deshalb auch Menschen ohne vorhandene Vorkenntnisse verfügbar machen. Durch Ausnutzung musiktheoretischer Phänomene und Gesetzmäßigkeiten, die dazu führen, dass bestimmte Konstellationen von Tönen und Akkorden durch den Menschen stets als wohlklingend wahrgenommen werden, kann dem Musiker eine musikalische Ausbildung erspart bleiben, wodurch auch musizieren *massentauglich* werden kann.
Im folgenden werden wir die erwähnten musiktheoretischen Phänomene kurz erläutern, anschließend aufzeigen, wie wir bei der technischen Umsetzung vorgegangen sind und schließlich unseren Ansatz mit anderen existierenden Produkten vergleichen.

Musiktheoretischer Hintergrund (DD)

Konsonanz und Dissonanz

Der Begriff Konsonanz (aus dem lateinischen: *con* = zusammen und *sonare* = klingen) lässt sich im Bereich der Musik nicht eindeutig definieren. Jedenfalls findet man keine Definition, die alle Musiktheoretiker gleichermaßen zufrieden stellt (s.[Dal14], S.104ff.). Aus diesem Grunde sei an dieser Stelle - stark vereinfacht - Konsonanz als eine Eigenschaft definiert, die beschreibt, wie gut zwei Töne zusammen klingen, d.h. wie harmonisch ihre Hörwahrnehmung ist.
Es ist schnell einsichtig, warum Konsonanz im Kontext dieser Arbeit von Bedeutung ist: Will man ein Werkzeug entwickeln, mit dem ausschließlich wohlklingende Tonfolgen erzeugt werden können, so muss also das Ziel sein, möglichst immer Konsonanz zwischen den Tönen zu erreichen. Das Problem ist jedoch, dass Konsonanz sich im Allgemeinen weder mathematisch noch psychologisch einwandfrei beschreiben lässt, wie die nachfolgenden Abschnitte zeigen werden. Trotz der unscharfen Definitionen hat man es geschafft den Intervallverhältnissen zwischen zwei Tönen einen Konsonanzgrad zuzuordnen (s. [int16]). Auf diese Festlegung wird sich in den anschließenden Abschnitten immer wieder bezogen.

Definition nach *Hermann von Helmholtz*

Im 19. Jahrhundert erklärte der deutsche Wissenschaftler Hermann von Helmholtz die Konsonanz mit Hilfe sogenannter Schwebungen (s.u.), die beim Zusammenklingen zweier Töne mit unterschiedlicher Frequenz entstehen (s.[Dal14], S.105).
Helmholtz postulierte, dass diese Schwebungen die zentrale Ursache für das Dissonanzempfinden darstellen. Besonders Schwebungsfrequenzen zwischen 10 und 40 Hz seien für das menschliche Gehör sehr unangenehm (s. [Wal19],S.66).

Exkurs: Schwebung Eine Schwebung ist ein physikalisches Phänomen, das bei der Addition periodischer Schwingungen entsteht. Addiert man z.B. zwei Sinus-Schwingungen mit sich leicht unterscheidender Frequenz, so wird das Ergebnis wieder eine Sinus-Schwingung sein, die jedoch eine periodisch an- und abschwellende Amplitude besitzt. Die Frequenz dieser als *Schwebung* bezeichneten Überlagerungsschwingung berechnet sich durch

$$f_{Schwebung} = |f_1 - f_2|$$

Definition nach *Carl Stumpf*

Carl Stumpf entkräftete die von Helmholtz aufgestellte Theorie jedoch mir der Argumentation, dass sich Schwebungen auch bei zwei konsonant zusammen klingenden Tönen künstlich erzeugen lassen. Umgekehrt war es ihm gelungen, dissonante Zusammenklänge zu erzeugen, die völlig frei von Schwebungen waren.
Stumpf vertrat die Meinung, dass in erster Linie die Tonverschmelzung die Ursache für konsonant zusammen klingende Töne war. Damit war die Unterscheidbarkeit zweier gleichzeitig erklingende Töne gemeint, d.h. je eher ein Mensch in der Lage ist, zwei zugleich gehörte Töne akustisch zu unterscheiden (je geringer also ihre Verschmelzung ist), desto dissonanter würden diese wahrgenommen.

Wahl einer Tonmenge

Akkorde

Wie bereits im Abschnitt zur Konsonanz erläutert, ist die Wahl zueinander konsonanter Töne ein erstrebenswertes Ziel bei der Erstellung eines Werkzeugs für musikalisch völlig ungeschulte Anwender. Durch Vermeidung

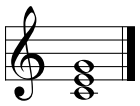

Abbildung 1: C-Dur Akkord

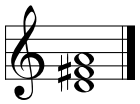

Abbildung 2: D-Dur Akkord

von Dissonanzen werden nämlich Tonfolgen sowie -zusammenklänge umgangen, die potentiell als unangenehm wahrgenommen werden. Durch dieses Vorgehen steigt also die Wahrscheinlichkeit erheblich, dass - selbst bei völlig zufälliger Auswahl von gleichzeitig oder nacheinander abgespielten Noten aus der von uns vorgegebenen Menge - das Ergebnis im allgemeinen noch als *wohlklingend* beschrieben werden kann. Die Aufgabe besteht nun also darin eine geeignete Menge von Noten zu finden, die diese o.g. Forderung erfüllt.
Ein guter Kandidat dafür ist der Dur-Akkord. Er besteht aus dem jeweiligen Grundton, der großen Terz darüber und der kleinen Terz darüber (s.Abb.1). [int16] attestiert den beiden Terz-Intervallen zwar eine lediglich *unvollkommene Konsonanz*, jedoch lässt sich subjektiv sehr schnell feststellen, dass diese drei Noten im Zusammenklang ein sehr harmonisches Klangbild erzeugen. Funktioniert dies jedoch auch, wenn nicht alle drei Noten zugleich gespielt werden? Ja, denn das einzige *Nicht-Terz-Intervall* in dieser Konstellation ist jenes zwischen dem Grundton und dem höchsten Akkordton (im Falle des C-Dur-Akkords sind dies **C** und **G**). Diese bilden jedoch - mit einem Abstand von 7 Halbtönen - eine reine Quinte, welche ihrerseits ein vollkommen konsonantes Intervall darstellt (vgl.[int16]). Somit haben wir mit dem Dur-Akkord bereits eine geeignete Notenmenge für unseren Anwendungsfall identifiziert.
Drei Noten allein genügen einem potentiellen Anwender jedoch vermutlich nicht. Vielmehr sollte dem Benutzer ein Umfang von mindestens zehn Tönen zur Verfügung stehen, um eine ansprechende Melodie mit einer gewissen Dynamik erzeugen zu können. Die gefundene Notenmenge ist also entsprechend zu erweitern. Es ist jedoch nicht möglich, einfach einen beliebigen weiteren Dur-Akkord hinzuzunehmen, da die Töne der zwei Akkorde dann möglicherweise dissonante Intervalle bilden. Ein Beispiel: Nutzt man die Töne eines C-Dur-Akkordss (C-E-G) zusammen mit denen eines D-Dur-Akkord (D-Fis-A; Abb.2), so erhält man zusätzlich die in Abb.3 dargestellten Intervalle. Neben den zwei reinen Quarten (E-A und D-G), die als vollkommen konsonant zu bewerten sind und der großen Sexte (C-A), welche zumindest eine unvollkommene Konsonanz leisten kann, erhalten wir vier mild dissonante große Sekunden (C-D, D-E, E-Fis und G-A) sowie eine kleine Sekunde und einen Tritonus, die beide als scharf dissonant zu klassifizieren sind (s.[int16]). Dies zeigt, dass eine willkürliche Wahl eines weiteren Dur-Akkordes die Forderung nach konsonanten Intervallen keineswegs erfüllt.
Eine mögliche Lösung für dieses Problem ist die Wahl desselben Dur-Akkords in unterschiedlichen Oktaven. Z.B. zwischen den Intervallen C-E (4 Halbtöne) und C-E' (16 Halbtöne) ändert sich die Konsonanzwahrnehmung nicht, da es sich weiterhin um den selben Ton handelt, der lediglich in einer anderen Oktave gespielt wird. Einzig neu hinzukommendes Intervall ist die reine Quarte zwischen der höchsten Note des tieferen Akkords und der tiefsten Note des höheren Akkords. Die reine Quarte ist jedoch vollkommen konsonant, so dass unsere Forderung nach konsonanten Intervallen stets erfüllt bleibt. Bestimmt man nun nach diesem Prinzip eine Menge von Noten, so ergeben sich (im Beispiel für C-Dur) die in Abb.4 dargestellte Auswahl. Analog kann natürlich auch eine Notenmenge in Moll bestimmt werden, indem man statt Dur-Akkorden das Moll-Äquivalent wählt, also zunächst den Schritt von einer kleinen Terz geht und erst dann die große Terz weiterspringt (s. Abb.5). Das Rahmenintervall und die Akkordabstände verändern sich dadurch nicht.

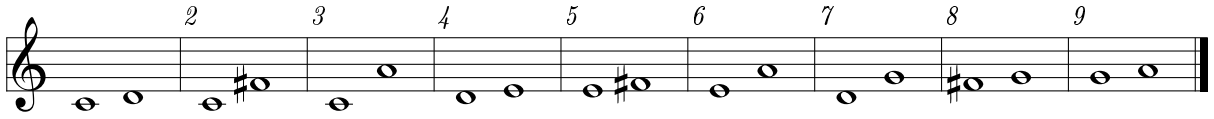

Abbildung 3: Intervalle, die sich aus der Kombination eines C-Dur-Akkordes mit einem D-Dur-Akkord ergeben

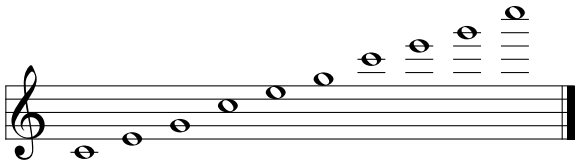

Abbildung 4: Auswahl von 10 Tönen für unseren Sequencer nach dem Dur-Akkord-Verfahren (C-Dur)

Pentatonik

Pythagoras konstruierte um 500 v.Chr mit folgender Berechnungsvorschrift eine Tonmenge:

Sei f_0 die Grundfrequenz

$$f_{n+1} = \begin{cases} f_n \cdot \frac{3}{4} \text{ , falls } f_n \cdot \frac{3}{2} > 2 \\ f_n \cdot \frac{3}{2} \text{ , sonst} \end{cases} \quad (1)$$

Auf diese Weise erhält man für die ersten fünf Frequenzverhältnisse $1, \frac{3}{2}, \frac{9}{8}, \frac{27}{16}, \frac{81}{64}$. Aufsteigend sortiert entsprechen diese fünf Verhältnisse jenen der heutigen Noten C,D,E,G und A (vgl. [Wal19], S.69). Sie bilden eine pentatonische Skala (*griechisch* penta = fünf).
Umgekehrt kann man diese Skala konstruieren, indem man aus der gängigen heptatonischen Skala - in diesem Beispiel der C-Dur-Tonleiter - die Halbtonschritte entfernt. Diese befinden sich jeweils vor den Noten F und H, weshalb sie gestrichen werden. Was man hierdurch erreicht, ist, dass mit den verbleibenden Noten keine scharf dissonanten Intervalle (große Septime, kleine Sekunde und Tritonus) mehr gebildet werden können (vgl. [pena])). Schlimmstenfalls kann noch eine kleine Septime (z.B. D-C') oder eine große Sekunde (z.B. C-D) auftreten, die lediglich als milde Dissonanz zu klassifizieren sind (s.[int16]).
Aufgrund dieser Eigenschaften findet die Pentatonik in unterschiedlichsten Bereichen Verwendung, in denen es darauf ankommt, ohne viel konzeptionelle Arbeit wohlklingende Melodien zu erzeugen, wie z.B. bei der Improvisation u.a. im Jazz ([Pro]) oder beim Musizieren mit Kindern im Rahmen der Waldorfpädagogik ([penb]). Auch in Kinderliedern (z.B. *Backe, backe Kuchen*) und Jingles (z.B. *HARIBO*) findet man häufig pentatonische Melodien.
Die entsprechende pentatonische Skala für C-Moll bilden die Töne C-Es-F-G-B. So kann für unsere Anwendung nun also eine Menge von Tönen für die Dur- und die Moll-Pentatonik zu einem Basiston (im Beispiel weiterhin C) gewählt werden. diese sind in Abb.6 und 7 zu sehen.

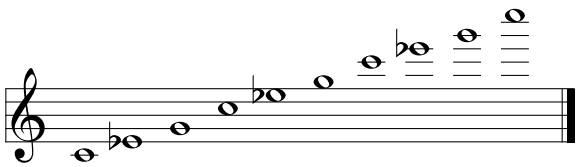

Abbildung 5: Auswahl von 10 Tönen für unseren Sequencer nach dem Moll-Akkord-Verfahren (C-Moll)

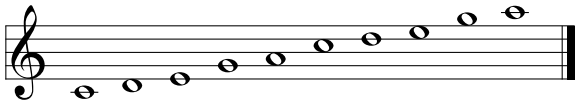

Abbildung 6: Auswahl von 10 Tönen für unseren Sequencer nach dem Dur-Pentatonik-Verfahren (C-Dur)

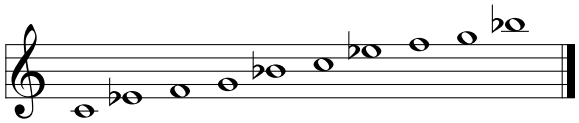

Abbildung 7: Auswahl von 10 Tönen für unseren Sequencer nach dem Moll-Pentatonik-Verfahren (C-Moll)

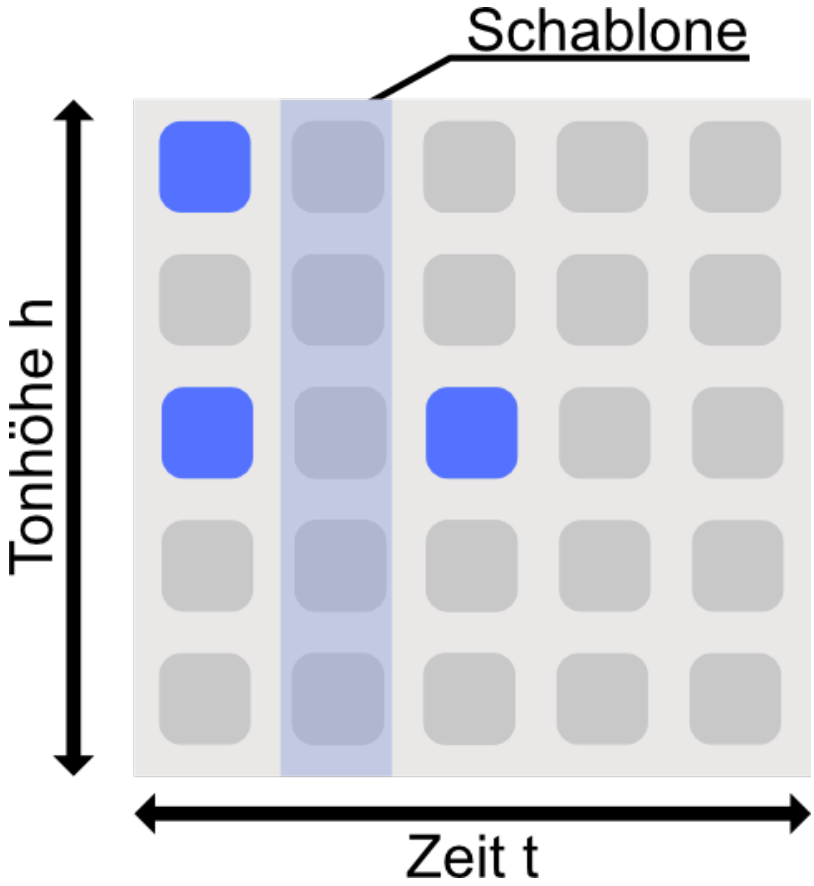

Abbildung 8: Konzept für die Umsetzung der GUI

Implementierung (PG)

Konzept

Der Nutzer soll auf einer 2-dimensionalen Matrix eine Melodie konstruieren. Dabei repräsentieren die X-Achse die Zeit und die Y-Achse die Tonhöhe. Gibt ein Nutzer auf der Vertikalen zwei Töne an, so werden diese gleichzeitig gespielt. Das Tool durchläuft dabei alle Töne entlang der Horizontalen bis zum Ende des durch den Nutzer definierten Rahmens. Nachdem ein Iterationsschritt durchlaufen wurde, springt das Programm wieder zurück an den Anfang, sodass eine Endlosschleife realisiert wird.
Dem Nutzer wird visualisiert, an welcher Stelle gerade Töne gespielt, beziehungsweise nicht gespielt werden. Dies erfolgt über eine Art Schablone, die in einem vom Nutzer festgelegten Zeitabstand entlang der Zeitachse über die Töne geschoben wird. Die Abfrage der Töne erfolgt in Echtzeit, sprich die Änderungen des Nutzers werden innerhalb des nächsten Iterationsschrittes registriert und umgesetzt.
Darüber hinaus soll der Nutzer die Möglichkeit haben, Matrizen für jeweils verschiedene Instrumente und Drums dynamisch hinzuzufügen.

Nichtfunktionale Anforderungen

Das von uns entwickelte Programm hat eine breite Zielgruppe. Um möglichst vielen Nutzern entgegenzukommen sind folgende nicht-funktionale Anforderungen notwendig.

Performanz

Die Umsetzung sollte so ressourcensparend wie möglichst sein. Sollte sich die Programmausführung auch nur um wenige Millisekunden verzögern, bestünde die Gefahr, dass der angestrebte Rhythmus nicht mehr gehalten werden kann und die Melodie aus dem Takt fällt. Das Resultat wäre eine Art Disharmonie. Dies gilt es zu vermeiden.

Plattformunabhängigkeit

Das Programm sollte auf möglichst vielen Plattformen lauffähig sein. Die Benutzereingaben sollen auf jeder Plattform zu dem gleichen Resultat, sprich derselben Melodie führen.

Benutzbarkeit

Dem Nutzer wird durch eine simple, einfach zu verstehende Oberfläche ein erleichterter Einstieg geboten. Angestrebt wird hierbei eine minimalistische GUI.

Veränderbarkeit

Alle Veränderungen sollen zur Laufzeit der Melodie direkt umgesetzt werden. Die Session wird dabei nicht unterbrochen. Dadurch, dass dem Nutzer die Manipulationen der Melodie direkt aufgezeigt werden und somit ein natürlicher Übergang zwischen zwei Melodien erfolgt, erhoffen wir uns einen erhöhten Lerneffekt.

Java-Umsetzung

Zunächst wurde sich für eine Umsetzung in Java entschieden. Hierbei war angedacht, die GUI mithilfe des JavaFX-Frameworks zu bauen, während die Töne mittels des JavaX.Sound.Midi-Pakets (vgl.[Ora18]) erzeugt werden. Die Umsetzung scheiterte nicht nur an den mangelnden JavaFX Kenntnissen, sondern vor allem an der Verfahrensweise des Midi-Paketes.
Ein Track, der erstellt wird, ist einem Sequencer zugehörig. Nachdem die Noten des Tracks gesetzt wurden, kann der Sequencer unter Angabe eines festgesetzten Tempos gestartet werden. Sollte sich nun aber die Melodie durch neue Benutzereingaben verändern, so muss der Sequencer neu gestartet werden, sprich die Session wird unterbrochen. Dies widerspricht jedoch der nichtfunktionalen Anforderung der Veränderbarkeit, da so kein natürlicher Übergang zwischen der alten und der neuen Melodie gewährleistet werden kann.

```
1  Track track = sequence.createTrack();
2  // Adding some events to the track
3  for (int i = 5;
4       i < (4 * numOfNotes) + 5;
5       i += 4)
6  {
7       // Add Note On event
8       track.add(makeEvent(144, 1, i, 100, i));
9
10      // Add Note Off event
11      track.add(makeEvent(128, 1, i, 100, i + 2));
12 }
13 // Setting our sequence
14 //so that the sequencer can
15 // run it on synthesizer
16 sequencer.setSequence(sequence);
17
18 // Specifies the beat rate
19 // in beats per minute.
20 sequencer.setTempoInBPM(220);
21
22 // Sequencer starts to play notes
23 sequencer.start();
```

Listing 1: Typische Umsetzung der MIDI-Soundwiedergabe in JAVA unter Verwendung eines Sequencers, Quelle: [gre]

Unity-Umsetzung

Das Scheitern der Java-Umsetzung hat zu einem Umdenken unseres Ansatzes geführt. Um die Veränderungen des Nutzers zur Laufzeit der Musik umzusetzen, haben wir uns für den Einsatz einer Spiel-Engine entschieden. Ein großer Vorteil besteht bei diesem Ansatz darin, dass durch den Einsatz einer Game-Loop die Anforderung der Veränderbarkeit nicht verletzt wird (vgl.[Wik18]). Da viele Spiele-Engines inzwischen auch plattformunabhängig sind und darüber hinaus auch plattformunabhängige Frameworks für die GUI-Erzeugung mitliefern, sind diese für die programmatische Umsetzung unseres Konzepts sehr geeignet.
Da wir bereits Erfahrung mit der Unity-Engine (vgl.[Tec18i]) sammeln konnten entschieden wir uns für eine Umsetzung innerhalb dieser Laufzeits- und Entwicklungsumgebung. Die entsprechende Codebasis wurde mithilfe C# unter Verwendung des Entwurfsmusters *Objektorientierung* realisiert.

Aufbau des Unity-Projekts

Unity baut auf das Konzept, dass möglichst viele Objekte schon vor der Laufzeit vordefiniert werden. Diese sogenannten *GameObjects*(vgl.[Tec18f]) werden mit verschiedenen Attributen (im Unity Fachjargon *Components*(vgl.[Tec18d])) versehen. Man hat dabei die Möglichkeit, auf die in der Engine vordefinierten Standard Components zurückzugreifen oder per C#-Skript eigene Components zu definieren.
Die folgenden Betrachtungen beziehen sich auf Abb. 9. Die vor der Laufzeit angelegten GameObjects innerhalb der Szene sind im Blauen Rahmen dargestellt und werden in einer Vorschau vorgerendert (rechter Abschnitt des Bildes). Dem Nutzer werden dabei nur Elemente innerhalb des Sichtfeldes der MainCamera präsentiert (grüner Rahmen), die restlichen Elemente außerhalb der Szene dienen zum Kopieren derselben während der Laufzeit (roter Rahmen).
Prinzipiell wurde bei den GameObjects in folgende funktionalitäten Unterschieden:

Canvas

Innerhalb des Canvas-Objektes (vgl.[Tec18c]) werden die UserInterface-Elemente registriert und gerendert. Das Canvas-Objekt wird mit der MainCamera (vgl.[Tec18b]) verknüpft, sodass die jeweiligen Elemente innerhalb des Nutzersichtfensters positioniert werden. Die in der Abbildung vorliegenden UI-Elemente wurden bereits vor

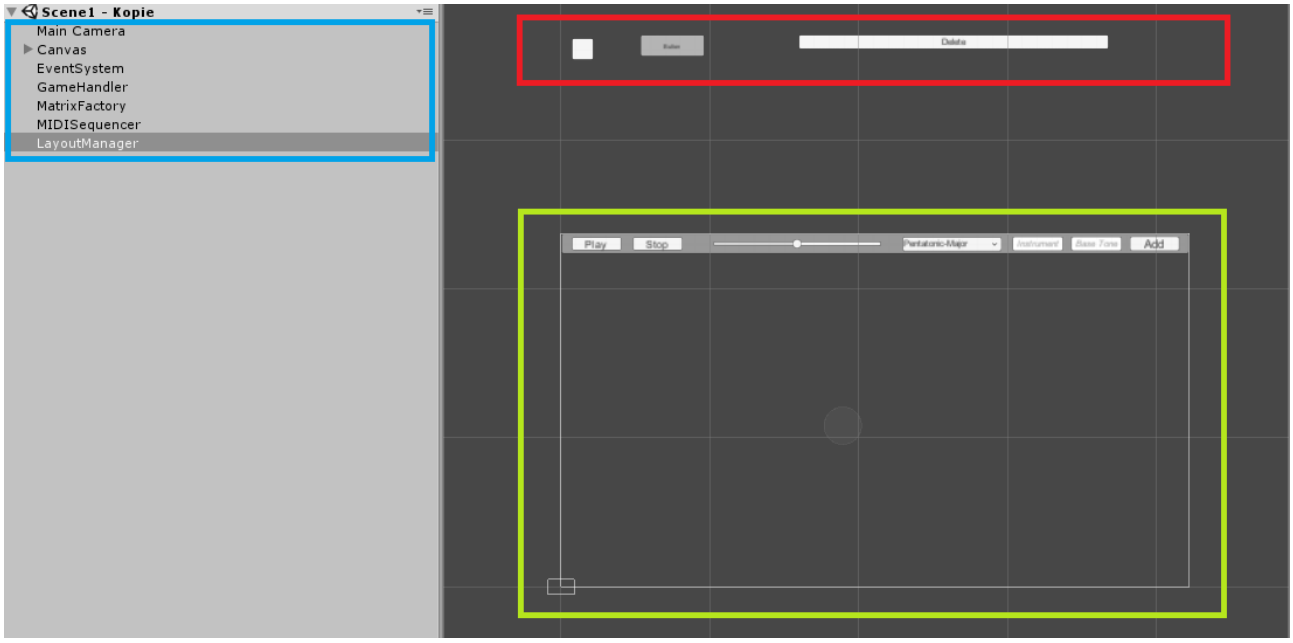

Abbildung 9: Benutzeroberfläche von Unity

der Laufzeit im Editor definiert, die zur Laufzeit erzeugten Elemente werden ebenfalls in das Canvas-Objekt gespeist.

GameHandler

Dem GameHandler-Objekt unterliegt die selbstgeschriebene GameHandler-Component. Dieses Objekt ist für die Realisierung der Spiellogik zuständig - Hier wird definiert, wie häufig die Game-Loop frequentiert wird. Darüber hinaus steuert der GameHandler den MIDISequencer an und übergibt ihm die zum aktuellen Zeitpunkt angespielt/gestoppt werden.

MatrixFactory

Dem MatrixFactory-Objekt unterliegt die selbstgeschriebene MatrixFactory-Component. Dieses Objekt ist für die Erstellung der MatrixPanel verantwortlich.

MIDISequencer

Dem MIDISequencer-Objekt unterliegt die MIDISequencer-Component. Dieses Objekt ist für die Steuerung des Midi-Sequencers verantwortlich, es spielt die vom GameHandler übergebenen Töne zum Zeitpunkt X ab. Der MIDISequencer basiert auf der Codebasis des Projektes CSharpSynthProject (vgl.[pau14]), hierbei wurde die Adaption der Codebasis innerhalb der Unity-Entwicklungsumgebung des Autors Chad Carter (vgl.[Car17]) übernommen. Die Codebasis als auch die Adaption unterliegt der MIT-Lizenz, die Verwendung des Sequencers ist somit sowohl für Open- als auch ClosedSource Projekten grundlegend erlaubt.
Die Verwendung der CSharpSynthProject-Codebasis erlaubt eine Plattformunabhängige implementierung, da wir hier nicht auf proprietäre MIDI-Treiber angewiesen sind. Die Ausgabe erfolgt innerhalb der unityinternen SoundEngine, folglich wird die plattformspezifische Audio-Ausgabe von Unity selbst übernommen.

LayoutManager

Dem LayoutManager-Objekt unterliegt die selbstgeschriebene LayoutManager-Component. Dieses Objekt ist für die Erzeugung und Verwaltung der zur Laufzeit erstellten UserInterface-Elemente zuständig. Diese werden in den Entsprechenden Container-Objekten innerhalb des Canvas-Objektes erstellt.

Abbildung 10: Benutzeroberfläche unserer Anwendung unmittelbar nach dem Öffnen

Funktionsablauf

Zunächst läuft sämtliche Interaktion mit dem Programm im Header ab. Der Nutzer kann über die Art der Konsonanz entscheiden. Zur Auswahl stehen dabei die Pentatonik in jeweils Dur und Moll als auch ein klassischer Dur- und Moll-Akkord (s.Abb 10).
Der Nutzer übergibt dabei innerhalb eines Input-Feldes eine MIDI-Instrumenten-Nummer und einen Basis-Ton, über dem die Konsonanz errechnet wird. Darüber hinaus kann der Nutzer im Dropdown-Feld eine Drum-Spur auswählen, aufgrund CSharpSynth spezifischer Eigenheiten sind die Instrumente als auch der Basis-Ton innerhalb der Programmlogik vordefiniert, weswegen die Input-Felder für Instrumenten-Nummer und Basis-Ton hier standardmäßig deaktiviert sind. Hat der Nutzer alle notwendigen Eingaben getätigt, kann durch Drücken des *Add*-Buttons ein MatrixPanel erzeugt werden.

Ein MatrixPanel-Objekt selbst, beziehungsweise dessen Component ist dabei aufgebaut, wie in Abb. 11 zu sehen.

Ein MatrixPanel besteht grundlegend aus einem GameObject, das eine UI-Panel-Component inne hält. Innerhalb der MatrixPanel-Klasse werden die einzelnen MusicTiles definiert und verwaltet. Diese MusicTiles sind eigens geschriebene Components, die eine Button-Component (vgl.[Tec18a]) beinhalten. Sie werdem dem Nutzer letztendlich als die Buttons präsentiert, auf die er klicken muss, um einen Ton Höhe Y an der Stelle X anzumelden. Innerhalb der Ereignisverwaltung des Buttons selbst wird mit einem Click-Event die Anmeldung/Abmeldung beim GameHandler-Objekt realisiert, sodass der dem Button zugrundeliegende Ton Höhe Y zum Zeitpunkt X gespielt/ nicht gespielt werden kann. Welcher Ton beziehungsweise welches Instrument gespielt wird ist im jeweiligen MusicTileData-Objekt definiert. Dieses wird bei der Anmeldung/Abmeldung durch das Click-Event des MusicTiles an den GameHandler mit übergeben.

Hat der Nutzer nun einen Ton per Klick auf das entsprechende MusicTile angemeldet, kann er die Abspiellogik mit dem Start/Stop-Button ausführen/anhalten. Dabei läuft ein Marker über entlang der Horizontalen X-Achse über die Menge der entsprechenden Tiles, um dem Nutzer zu verdeutlichen, welche MusicTiles aktuell angespielt werden. Per Slider kann hierbei das Tempo der Abspielreihenfolge angepasst werden.

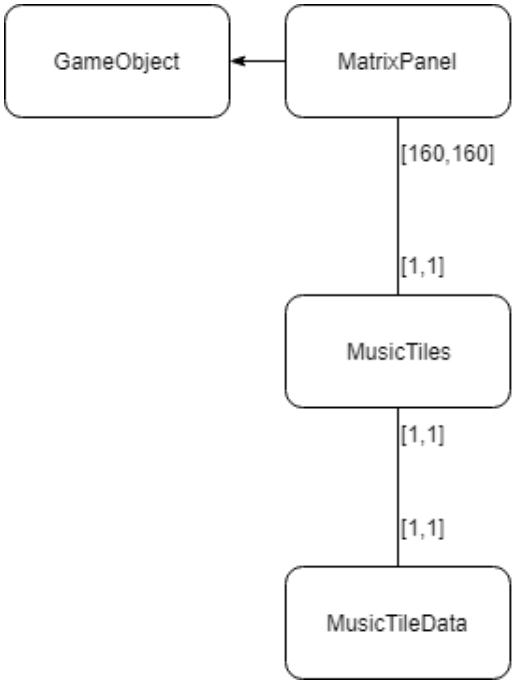

Abbildung 11: Aufbau der Component eines MatrixPanel-Objektes

Programmatische Anpassungen

MIDISequencer

Der MIDISequencer selbst beziehungsweise dessen Codebasis in Form des CSharpSynth selbst bedarf einer Erläuterung: Der MIDISequencer verwendet den General-MIDI Standard GM1, folglich muss die Instrumentennummern diesem entnommen werden (vgl.[pau18]). Darüber hinaus ist die Funktionalität der Codebasis selbst von Plattform zu Plattform unterschiedlich. Zunächst einmal ist zu erwähnen, dass der MIDISequencer auf allen bisher getesteten Plattformen (Windows,Linux,Android) lauffähig ist. Jedoch ist der Sound-Output gestört, sodass der ausgegebene Ton rauscht. Auf Linux hingegen funktioniert die `StreamSynthesizer.NoteOff()`-Methode der CSharpSynth-Codebasis nicht sachgemäß, sodass auf die `StreamSynthesizer.NoteOffAll()` Methode zurückgegriffen werden muss. Dies hat zur Folge, dass die entsprechenden Noten nicht ausgespielt, sondern abrupt abgebrochen werden. Um trotzdem ein für die Plattform optimales Nutzererlebnis zu bieten, wurde für die Windows-Version die `NoteOff()`-Methode und für die Linux-Version die `NoteOffAll()`-Methode implementiert. Per Präprozessor-Makro werden beim Kompilieren des Objektes die plattformspezifischen Anpassungen in den Build übernommen (siehe hierzu Listing 2 und 3).

Game-Loop

Damit das Programm entsprechend der zuvor aufgestellten Spezifikationen lauffähig ist, mussten einige Anpassungen vorgenommen werden. Zunächst wäre hier die Umsetzung der Game-Loop zu erwähnen. Unity hat diese zwar standardmäßig in Form der Update-Methode (vgl.[Tec18h]) umgesetzt, jedoch ist die Frequenz ihrer Aufrufe an die Prozessorgeschwindigkeit gebunden, somit variabel und für ein Plattformunabhängiges Nutzererlebnis nicht geeignet. Zwar gibt es die Unity-Methode *FixedUpdate*(vgl.[Tec18g]), bei der man die Anzahl der Aufrufe pro Sekunde steuern kann, jedoch ist dies nur bis zur Kompilierung des Projektes möglich, liegt das Programm als .exe vor kann die Geschwindigkeit im vorliegenden Build nicht nachwirkend geändert werden. Um jedoch dem Nutzer zu ermöglichen, während der Laufzeit des Programms die Geschwindigkeit zu ändern, mussten wir eine eigene Game-Loop schreiben. Dies wurde mit einer Endlosschleife realisiert, die Geschwindigkeit des Programms ist in dieser Herangehensweise die Zeit, in der die GameHandler-Klasse den Ton registriert. Hierbei wird solange ein Ton beim MIDISequencer abgespielt, wie der Nutzer die Geschwindigkeit des Programms per Slider angegeben hat. Der GameHandler wird per Coroutine (vgl.[Tec18e]) währenddessen für die Dauer des Tones pausiert, bis er nach Abschluss der Zeit den entsprechenden Ton beim MIDISequencer abmeldet.

```
1  // GameHandler.cs
2  #if UNITY_STANDALONE_LINUX
3      midiSequencer.stop ();
4  #endif
5
6  #if UNITY_STANDALONE_WIN
7      foreach (MusicTileData tileData in verticalRowsPlaying[counterVerticalRow])
8      {
9          midiSequencer.stop(tileData.getNote());
10     }
11
12     foreach (MusicTileData tileData in verticalRowsDrumsPlaying[
           counterVerticalRow])
13     {
14         midiSequencer.stopDrum (tileData.getNote());
15     }
16 #endif
```

Listing 2: Die Datei GameHandler.cs

```
1  // MIDISequencer.cs
2  public void stop()
3  {
4      midiStreamSynthesizer.NoteOffAll (true);
5  }
6
7
8  public void stop(int note)
9  {
10       midiStreamSynthesizer.NoteOff(0, note);
11 }
```

Listing 3: Die Datei MIDISequencer.cs

Vergleich mit anderen Lösungen (DD)

Die Grundidee unseres Programms ist nicht neu. Das Grundprinzip entspricht im Wesentlichen einem Reproduktionsklavier, das bereits seit dem Jahr 1905 bekannt ist. Reproduktionsklaviere verwenden Lochstreifen aus Papier als Trägermedium für das Musikstück. Der Papierstreifen wird durch eine mechanische Lesevorrichtung am Klavier geführt. Abhängig von der Position der Löcher erklingen dann unterschiedliche Töne (vgl. [wel]).
Heute findet man Hard- und Softwarelösungen, die unserem System ähnlich sind, unter dem Begriff *Matrix-* oder *Step-Sequencer*. Als Beispiel für Hardware-Geräte wäre die *Mode Machines SEQ-12* (s.[mod]) anzuführen. Hierbei handelt es sich im um eine MIDI-Steuerung, also ein Gerät zur Erzeugung von Midi-Signalen, welches zur Klangerzeugung noch einen entsprechenden Synthesizer benötigt.
Ein anderes Hardwareprodukt ist der *Synthstrom Deluge*. Es vereint die Funktionalität von Synthesizer, Sampler und Sequencer in einem Gerät. Die Bedienoberfläche sieht unserer Software bereits recht ähnlich. Was aber beiden genannten Geräten gemein ist, ist, dass sie sich eher an erfahrene Musiker richten, da der Umfang der spielbaren Töne nicht eingeschränkt wird und der Funktionsumfang sowohl technisch als auch musikalisch einiger Einarbeitung bedarf.
Für einen nähren Vergleich eigenen sich daher eher Softwarelösungen. Besonders solche, die mit dem gleichen Ziel erstellt wurden: das Musizieren einem breiteren Publikum zu ermöglichen. Zwei Anwendungen, die sich tatsächlich gut mit unserer Lösung vergleichen lassen„ sind zum einen die *Tone Matrix* (s.Abb.14 und [ton]), die von André Michelle entwickelt wurde und die Android App *RollingTones* von Vít Hotárek (s.Abb.13 und [Hot13]). Diese werden im folgenden näher betrachtet und unserer Lösung gegenübergestellt. *Tone Matrix* wurde dabei ausgewählt, weil es sich um einen der frühsten Vetreter solcher Anwedungen handelt. Der Protoyp ist mittlerweile ca. zehn Jahre alt (s. [ton09]) und hat sich seitdem kaum verändert. Für ein aktuelleres Beispiel fiel die Wahl auf *RollingTones*, da diese App unter dem Suchbegriff *Step Sequencer* im *Google PlayStore* unter den relevantesten Ergebnissen auftauchte und - im Gegensatz zu ähnlichen Konkurrenzprodukten - eine ansehnliche Zahl von Installationen (50.000+) und (größtenteils positiven) Bewertungen (683) aufwies (vgl.[Hot13]).

Instrumente

Die *Tone Matrix* - als ältestes der drei Programme - bietet lediglich ein einzelnes melodisches Instrument an. Dieses kann nicht gewechselt werden. Aktiviert man die Kacheln der Benutzeroberfläche, so beginnt das Tool unmittelbar mit dem Abspielen der zugehörigen Töne. *RollingTones* bietet hingegen zumindest die Möglichkeit, die Wellenform zu variieren. Zur Auswahl stehen Sinus-, Dreieck-, Sägezahn- und Rechteckschwingung jeweils in bipolarer oder normalisierter Form. Ein konkretes Instrument kann jedoch nicht gewählt werden.
Unser System bietet die Möglichkeit, ein beliebiges MIDI-Instrument zu wählen Somit stehen 128 melodische Instrumente zur Verfügung. Durch die Option, mehrere Matrizen einzufügen, kann man des Weiteren sogar mehrere melodische Instrumente gleichzeitig erklingen lassen. So ist es theoretisch denkbar (abhängig von der Rechenleistung des Anwenderrechners), ein ganzes Orchester zu imitieren. Zwischen den einzelnen Matrizen kann über *Tabs* gewechselt werden.
RollingTones bietet außerdem rhythmische Instrumente an. Es ist möglich, zwischen Kick-Drum, Snare, closed und opened Hi-Hat, Crash, Tabourine und Claps zu wählen und für jedes dieser Instrumente unterschiedliche Samples aus einer Liste zu selektieren. Die Positionierung der rhythmischen Instrumente in der Matrix spielt nur auf der Zeitachse eine Rolle. Beim Positionieren eines perkussiven Instruments auf unterschiedlichen Tonhöhen-Ebenen (vertikale Achse) gibt es keine Klangunterschiede. Es sei gesagt, dass die gleiche Matrix verwendet wird, die auch für die Melodie genutzt wird. Dadurch kann es vorkommen, dass eine Kachel, die für ein melodisches Instrument gewünscht ist, bereits durch ein rhythmisches belegt ist oder umgekehrt.
Tone Matrix bietet überhaupt keinen rhythmischen Teppich für erzeugte Musikstücke an. Zwar zeigt die Benutzeroberfläche im Browser einen Button mit der Aufschrift *ADD DRUMS* (s.[ton] oder Abb.14), Klickt man diesen jedoch an, wird man auf eine Fehlerseite (*404 Not Found*) weitergeleitet. Dies legt nahe, dass das Feature mal vorhanden war, dann aber wieder entfernt wurde.
Unser System lässt es zu, separate Matrizen für das Schlagwerk hinzuzufügen. Als Instrumente stehen wieder alle MIDI-Schlaginstrumente zur Verfügung, die sich auf fünf unterschiedliche Drum-Kit Matrizen verteilen. So hat der Nutzer die Möglichkeit, das volle Schlagzeug- und Perkussions-Spektrum auszuschöpfen und das völlig unabhängig von der Melodie. Allerdings ist bei der Verwendung der rhythmischen Instrumente ein zumindest grundlegender musikalischer Sachverstand von Nöten.

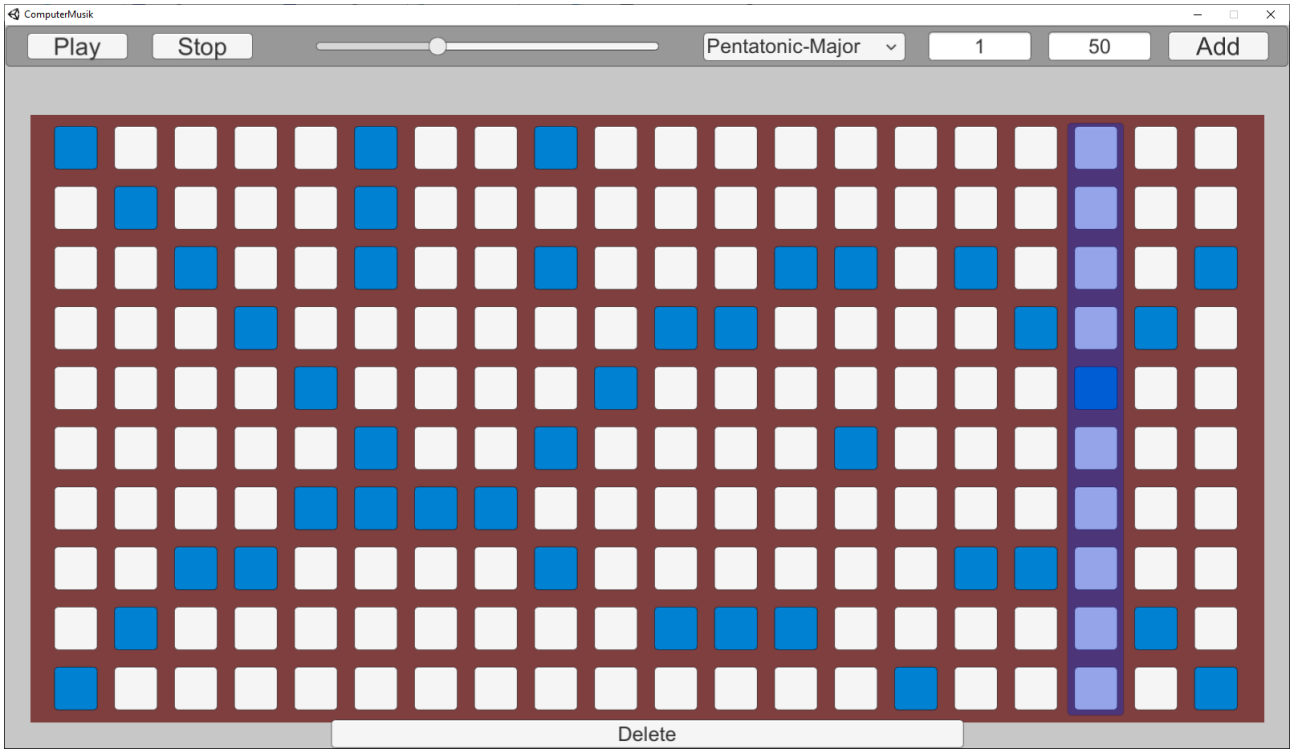

Abbildung 12: Benutzeroberfläche der von uns implementierten Software

Töne und Skalen

Auch hier bietet die *Tonematrix* wieder den geringsten Funktionsumfang: Es steht lediglich die Standardeinstellung zur Verfügung. Dabei handelt es sich um 16 Noten entlang einer pentatonischen Dur-Skala mit dem Grundton C. Weder die Oktave, noch die Art der Skala kann geändert werden.
RollingTones hingegen bietet solche Möglichkeiten, versteckt diese jedoch im Einstellungsmenü. Die Standardeinstellung für die Skala ist eine H-Moll Pentatonik, es können jedoch noch weitere pentatonische Moll- und Dur-Skalen ausgewählt werden. Außerdem stehen neben einigen klassisch heptatonischen Skalen auch diverse *natürliche* und *harmonische* Skalen zur Auswahl. Was dies im einzelnen für die Wahl der Noten bedeutet, wurde für diesen Vergleich jedoch nicht näher beleuchtet. *RollingTones* bietet außerdem die Möglichkeit, die Matrix um eine Oktave nach oben oder unten zu verschieben. Auch diese Einstellung findet sich im separaten Einstellungsmenü.
Die von uns entwickelte Anwendung bietet ebenfalls die Option, zwischen verschiedenen Skalen auszuwählen. Es stehen die pentatonischen Dur- und Moll-Skalen und die Akkord-basierten *Dreiton-Skalen* zur Verfügung (s.oben). Der Grundton der jeweiligen Skalen lässt sich frei aus der *MIDI-Range* auswählen. Hier stehen theoretisch alle Noten von A0 bis C8, also praktisch eine vollständige Pianoklaviatur zur Verfügung. Bei der Wahl des Grundtons ist allerdings einschränkend zu beachten, dass alle Töne der dazu ausgewählten Skala ebenfalls noch innerhalb der *Range* für MIDI-Noten liegen müssen.

Sonstiges

Unsere Anwendung bietet zusätzlich noch die Funktion, die Wiedergabegeschwindigkeit der erstellten Melodie über einen Schieberegler zu variieren (s.Abb.12).
Auch *RollingTones* hat eine solche Einstellmöglichkeit, die jedoch auch in diesem Fall im Einstellungsmenü untergebracht wurde. Die Android-Applikation bietet des Weiteren eine Funktion zur Erstellung ganzer Songs durch Kombination verschiedener Matrix-Konfigurationen. Da es hierzu jedoch keine Dokumentation gibt und die Bedienung nicht sehr intuitiv ist, konnte diese Funktion nicht getestet werden. Die App bietet außerdem die Möglichkeit, das aktuelle Projekt abzuspeichern.

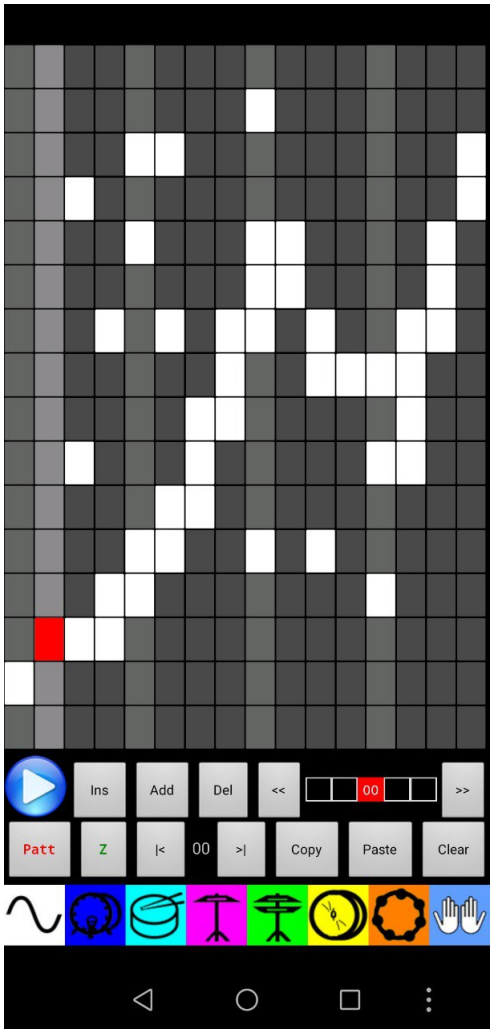

Abbildung 13: Benutzeroberfläche von *RollingTones*

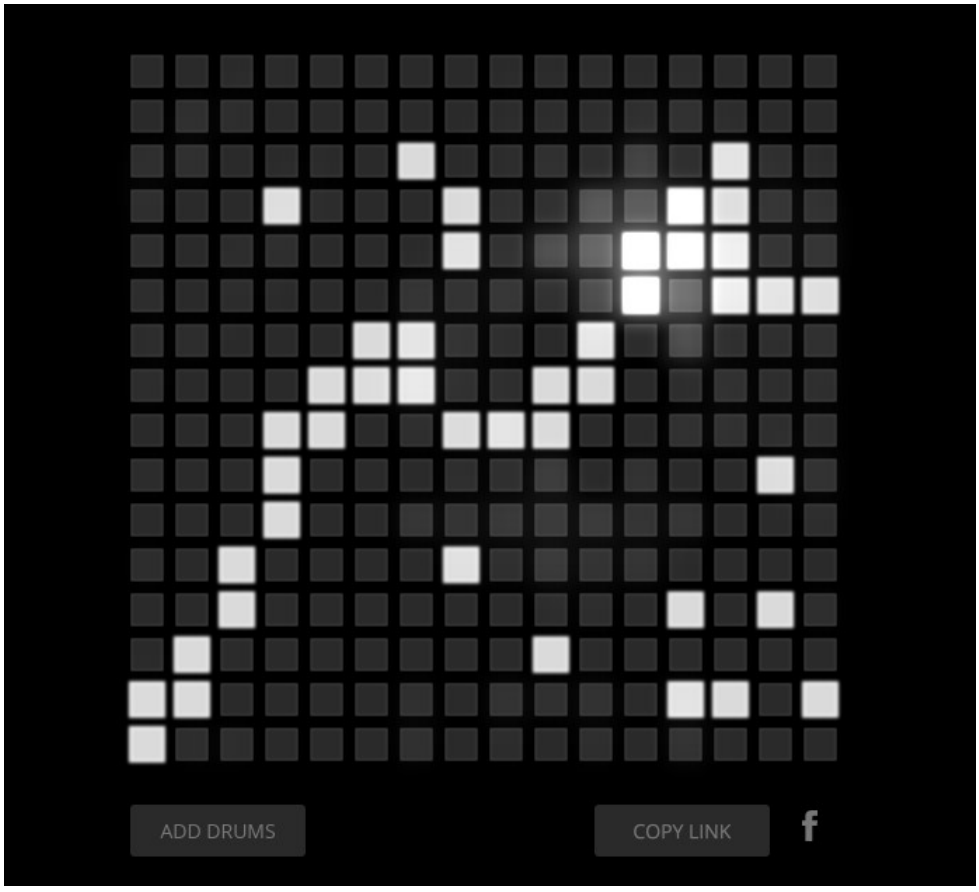

Abbildung 14: Benutzeroberfläche von *ToneMatrix*

Fazit und Ausblick (PG)

Nach anfänglichen Startschwierigkeiten bei der Umsetzung sind wir mit der fertigen Abgabe durchaus zufrieden. Die von uns gesteckten Anforderungen an das Programm wurden weitestgehend eingehalten und auch die Performanz konnte dank des eingesetzten CSharpSynthProjects sichergestellt werden. Unsere Erfahrungen mit der Unity-Engine waren ganz klar von Vorteil, da sie den durch die Startschwierigkeiten aufkommenden Zeitverlust gut kompensieren konnten. Auch die bereits in der Unity-Engine mitgelieferten UserInterface-Components haben den Zeitaufwand für die Erstellung der GUI minimiert. All die Vorteile, die eine Umsetzung in der Unity-Engine mit sich bringt, haben dazu geführt, dass wir ein für den Nutzer einfach zu benutzendes Programm erstellen konnten, das sich auch vor vergleichbaren Produkten nicht zu verstecken braucht.

Im Falle einer weiteren Bearbeitung des Projektes würden wir noch zunehmend an der Verbesserung der User-Experience arbeiten. Hierfür wären zunächst ein Usability-Test mit Probanden nötig, um basierend auf den Beobachtungen weitere Verbesserungen vornehmen zu können. Wünschenswert wäre ebenfalls die Möglichkeit, weitere MIDI-Standards wie den GM2-Standard zu verwenden, sowie eine Export/Import Funktion von erstellten Nutzerklängen.

Literatur

[Car17] Carter, Chad: *CSharpSynthForUnity*. https://github.com/kewlniss/CSharpSynthForUnity. Version: 2017. – Abgerufen am 02.02.2019

[Dal14] Dallmann, Dr. Niels-Constantin: Musikterminologie erklärt: Konsonanz und Dissonanz. (2014), März. – Erschienen in Sonic, S.104ff.

[gre] greendot: *Java | MIDI Introduction*. Quelle:https://www.geeksforgeeks.org/java-midi/. – Abgerufen am 20.01.2019

[Hot13] Hotarek, Vit: *RollingTones*. https://play.google.com/store/apps/details?id=cz.hotarekv.rtones. Version: Juni 2013. – Abgerufen am 02.02.2019

[int16] *Konsonanz und Dissonanz*. http://www.musicademy.de/index.php?id=2077. Version: 2016. – Abgerufen am 30.01.2019

[ler] *Die Anfänge – frühe Hochkulturen – Antike*. https://www.lernhelfer.de/schuelerlexikon/musik/artikel/die-anfaenge-fruehe-hochkulturen-antike. – Abgerufen am 20.01.2019

[mod] *Mode Machines SEQ-12; 12-Spur Matrix Sequenzer*. https://www.thomann.de/de/mode_machines_seq_12.htm?sid=686e26d5912840c3ba540127875a8dfc. – Abgerufen am 02.02.2019

[Ora18] Oracle: *Package javax.sound.midi*. https://docs.oracle.com/javase/7/docs/api/javax/sound/midi/package-summary.html. Version: 2018. – Abgerufen am 02.02.2019

[pau14] *CSharpSynthProject*. https://archive.codeplex.com/?p=csharpsynthproject. Version: 2014. – Abgerufen am 02.02.2019

[pau18] *GM 1 Sound Set*. https://www.midi.org/specifications-old/item/gm-level-1-sound-set. Version: 2018. – Abgerufen am 02.02.2019

[pena] *Pentatonik*. http://www.wikiwand.com/de/Pentatonik. – Abgerufen am 30.01.2019

[penb] *Pentatonik*. http://www.waldorfwilhelmsburg.de/front_content.php?idart=117. – Abgerufen am 30.01.2019

[Pro] Probst, Christian: *Pentatonik*. https://www.theorie-musik.de/tonleiter/pentatonik/. – Abgerufen am 30.01.2019

[Tec18a] Technologies, Unity: *Button*. https://docs.unity3d.com/ScriptReference/UI.Button.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18b] Technologies, Unity: *Camera*. https://docs.unity3d.com/ScriptReference/Camera.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18c] Technologies, Unity: *Canvas*. https://docs.unity3d.com/Manual/UICanvas.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18d] Technologies, Unity: *Component*. https://docs.unity3d.com/ScriptReference/Component.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18e] Technologies, Unity: *Coroutines*. https://docs.unity3d.com/Manual/Coroutines.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18f] Technologies, Unity: *GameObject*. https://docs.unity3d.com/ScriptReference/GameObject.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18g] Technologies, Unity: *MonoBehaviour.FixedUpdate()*. https://docs.unity3d.com/ScriptReference/MonoBehaviour.FixedUpdate.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18h] Technologies, Unity: *MonoBehaviour.Update()*. https://docs.unity3d.com/ScriptReference/MonoBehaviour.Update.html. Version: 2018. – Abgerufen am 02.02.2019

[Tec18i] Technologies, Unity: *Unity User Manual (2018.3)*. https://docs.unity3d.com/Manual/index.html. Version: 2018. – Abgerufen am 02.02.2019

[ton] *The tonematrix, a pentatonic step sequenzer by Audiotool*. http://tonematrix.audiotool.com/. – Abgerufen am 02.02.2019

[ton09] *Andre Michelle : Tone Matrix*. https://adage.com/creativity/work/tone-matrix/16020. Version: Mai 2009. – Abgerufen am 02.02.2019

[Wal19] Waldmann, Prof. Dr. J.: *Computermusik Vorlesung WS 18*. https://www.imn.htwk-leipzig.de/~waldmann/edu/ws18/cm/folien/. Version: Januar 2019. – Abgerufen am 20.01.2019

[wel] *Überblick über die Geschichte der Firma Welte & Söhne*. http://www.welte-mignon.de/wiki/geschichte_der_firma_m._welte_soehne. – Abgerufen am 02.02.2019

[wik] *Geschichte der Popular- und Popmusik*. https://de.wikipedia.org/wiki/Popmusik#Geschichte_der_Popular-_und_Popmusik. – Abgerufen am 20.01.2019

[Wik18] Wikipedia: *Game Loop*. https://de.wikipedia.org/wiki/Game_Loop. Version: 2018. – Abgerufen am 02.02.2019