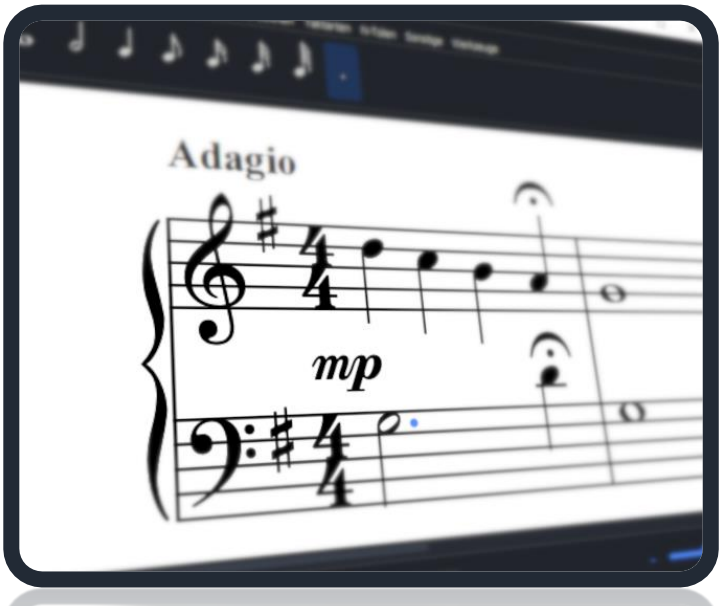
Dokumentation Abschlussarbeit Informatik:

Notensatzprogramm (Musik)



Software und Dokumentation erstellt von März bis Juni 2023 am
Wilhelm-Ostwald-Gymnasium Leipzig

Inhaltsverzeichnis

1 Kurzreferat

Als Abschlussarbeit der Sekundarstufe 1 im Fach Informatik wurde von Noah Weiler der Klasse 10/1 ein Notensatzprogramm für das Fach Musik erstellt. Es ermöglicht Schülern und anderen Benutzern, ein leeres Notenblatt mit Musikzeichen zu befüllen, um somit musiktheoretische Inhalte zu üben und Musikstücke zu schreiben. Nach der Bearbeitung kann das erstellte Dokument auf verschiedene Wege gespeichert und vervielfältigt werden, was den Austausch zwischen Schülern und Lehrkraft ermöglicht. Bei der Erstellung wurde darauf Wert gelegt, dass das Programm sehr nutzerfreundlich ist, gleichzeitig aber nicht zu viel vorgegeben wird, um für die schulische Anwendung geeignet zu sein. Das Programm wurde mit der Programmiersprache Python sowie dem GUI-Toolkit PyQt5 entwickelt.

2 Einleitung

2.1 Themenwahl

Das erstellte Notensatzprogramm ist eine Musiksoftware, die darauf ausgelegt ist, Notenblätter digital statt auf Papier zu befüllen, um ein gut lesbares Endresultat zu erhalten. Dabei liegt der Fokus auf dem Komponieren und nicht dem Erstellen von hörbarer Musik.

2.2 Projektziel

Die Zielstellung des Projekts beinhaltete neben dem Befüllen von Notenblättern mit Musikzeichen wie zum Beispiel Noten, Pausen, Artikulationszeichen und Dynamikzeichen auch das Verändern, also Löschen und Bewegen von Musikzeichen und insgesamt eine benutzerfreundliche und simple Gestaltung.

Die genauen Ziele mit Ordnung der Kriterien nach der Wichtigkeit sind in den Planungsunterlagen zu finden, welche sich ebenfalls, wie diese Dokumentation, im Dokumentationsordner des Projekts befinden.

2.3 Motivation

Für viele Menschen ist Musik ein wichtiger Bestandteil ihres Lebens. Dazu gehört auch, selber Musik zu produzieren und zu schreiben, weswegen Musik auch in der Schule unterrichtet wird, um die wichtigsten Grundlagen zu vermitteln. Dabei wird fast immer auf Stift und Papier zurückgegriffen. In Zeiten der Digitalisierung stehen dafür aber bereits Computerprogramme wie zum Beispiel MuseScore zur Verfügung, die jedoch wegen einer aufwändig zu erlernenden Bedienung noch nicht im Schulalltag angekommen sind.

Um Schülern und anderen eine leicht zu bedienende Notationssoftware an die Hand zu geben, wurde eine solche eigens programmiert und dokumentiert und dann von Musikern getestet.

Da die Programmidee etliche Möglichkeiten des Hinzufügens von Funktionen bereitstellt, konnte an diesem Projekt optimal das Lösen ganz unterschiedlicher Probleme geübt werden. Daher wurde dieses komplexe Projekt einem vergleichbar vielleicht simpleren vorgezogen.

2.4 Projektplanung

Das Projekt wurde bis zum 01.03.2023 geplant, die Planungsunterlagen sind im angesprochenen Dokumentationsordner zu finden. Wie dort angemerkt, kann der gesamte Quellcode unter dem Link in den unten befindlichen Quellen eingesehen werden. [1]

Die Versionsverwaltung Git mit der Oberfläche GitHub wurde daher gewählt, weil es damit möglich ist, den Code in mehreren Teilschritten zu veröffentlichen, kostenlos online zu speichern und auf das gesamte Projekt über Visual Studio Code zuzugreifen. Über das lokal installierte Git konnte dann auch zu verschiedenen Teilschritten zurückgewechselt werden, was die Fehlersuche beim Hinzufügen neuer Funktionen erleichtert.

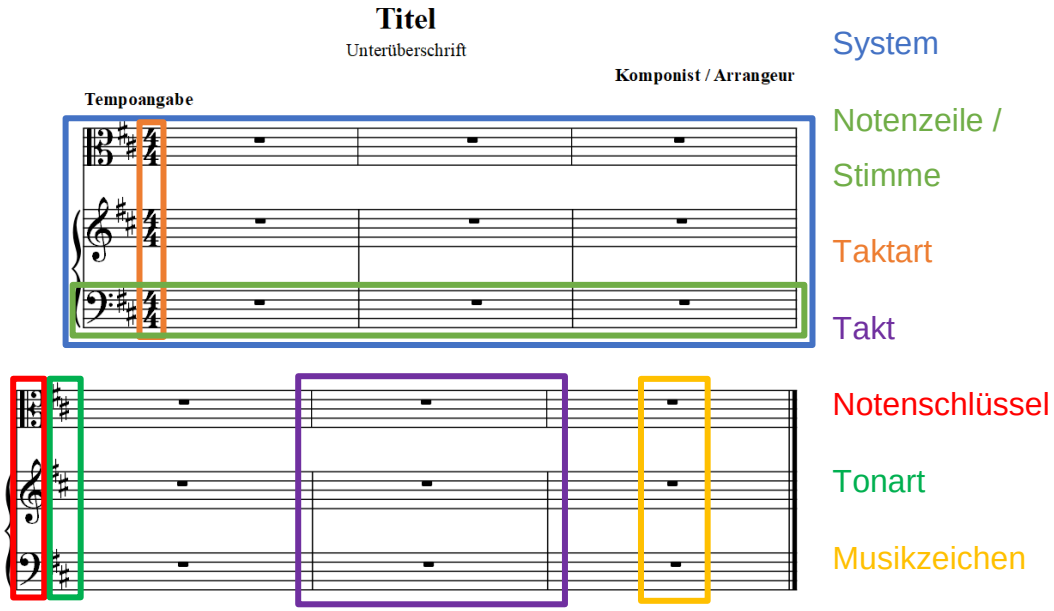
Das Projekt war bis kurz vor Abgabedatum privat und ist nun öffentlich, um den Code sowie die kompilierte Version leicht zugänglich zu machen.

3 Vorbetrachtung

3.1 Partituraufbau

Eine Partitur, wie sie mit dem Programm erstellt werden soll, stellt ein mehrstimmiges Musikstück dar.

Um die Musiknotation programmtechnisch umsetzen zu können, muss im Vorfeld die genaue Objektstruktur geplant werden. Im Folgenden ist eine Abbildung zu sehen, die später mit dem selbst erstellten Programm abgebildet wurde und die Struktur einer Partitur, also eines Notensatzes, verdeutlichen soll.



Dabei fällt auf, dass mehrere Objekte ineinandergreifen: So beinhaltet ein System mehrere Takte und mehrere Notenzeilen, wobei ein Takt mehrere Teilstücke von Notenzeilen enthält. Die für jede Zeile verschiedenen Notenschlüssel sowie die Tonart werden in jedem neuen System neu angezeigt, wobei die Taktart nur im ersten Takt und bei Taktwechseln angezeigt wird.

Dies erschwert die objektorientierte Umsetzung, da dabei genau darauf geachtet werden muss, welche Objekte welche anderen Objekte beinhalten. Dazu folgt später die Beschreibung der gefundenen Lösung.

3.2 Musikschriftarten

Beim Erstellen eines solchen Programms stellt sich schnell die Frage, wie es möglich ist, die vielen Musikzeichen dem Nutzer grafisch zur Verfügung zu stellen. Die Verwendung von Bilddateien würde lange Ladezeiten, eine schlechte Skalierbarkeit und uneindeutige Positionen verursachen. Daher wird für solche Anwendungen üblicherweise eine Schriftart verwendet, also ein Satz von Schriftzeichen, in diesem Falle Sonderschriftzeichen. Die Schriftzeichen einer Schriftart werden über Unicodes definiert und festgelegt. Ein Unicode besteht aus der Vorsilbe „U+“ und einer Hexadezimalzahl. Der Unicode für den Buchstaben „a“ ist zum Beispiel

U+0061. Lässt man sich nun von verschiedenen Schriftarten das Schriftzeichen zurückgeben, welches bei diesem Unicode hinterlegt ist, bekommt man in allen Fällen ein „a“ zurück, welches sich aber im Aussehen unterscheiden kann.

Das gleiche System wird bei Musikschriftarten verwendet. Dabei legt das Standard Music Font Layout, kurz SMuFL, fest, welchem Unicode welches Musikzeichen zugeordnet ist. Dieser offizielle Standard wurde von Daniel Spreadbury entworfen.

Es gibt viele verschiedene Musikschriftarten, aus denen, abhängig von den Ansprüchen, gewählt werden muss. Die Musikschriftart Bravura, die ebenfalls von Daniel Spreadbury entwickelt und 2013 veröffentlicht wurde, gilt dabei als generelle Referenzschriftart für den SMuFL-Standard und wurde daher gewählt. [2]

Bravura ist frei unter der Lizenz SIL OPEN FONT LICENSE Version 1.1 - 26 February 2007 herunterladbar. [3]

Mit dieser Schriftart wurde auch das blaue Icon des Programms erstellt.

Die Zuordnungen von Unicodes und Musikzeichen, sowie Implementierungshinweise wurden online bereits dokumentiert. [4] So muss zum Beispiel eine Note aus verschiedenen Schriftzeichen zusammengesetzt werden.

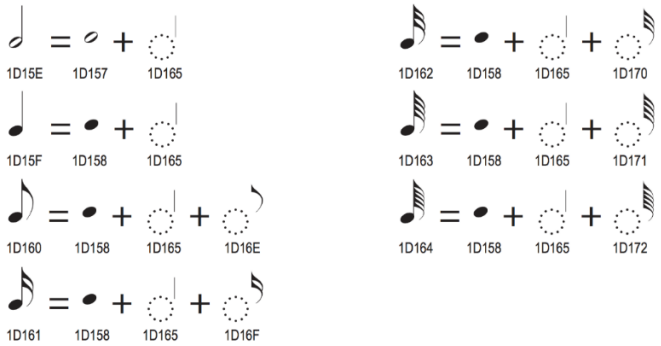


Abbildung 1: Implementierung von verschiedenen Notenwerten [1]

Dabei stellen die meisten Musikschriftarten, wie auch Bravura, Metadaten in JSON-Dateien zur Verfügung, die genaue Abmessungen und Koordinaten von wichtigen Verbindungspunkten der Schriftzeichen beinhalten, welche zur Implementierung essentiell sind.

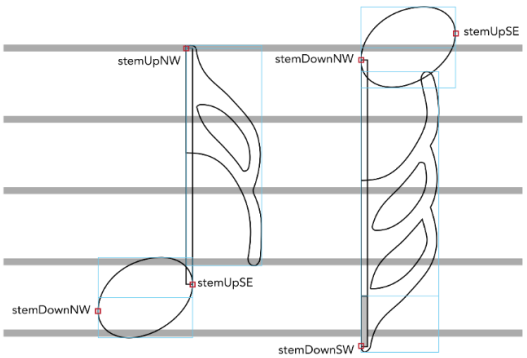
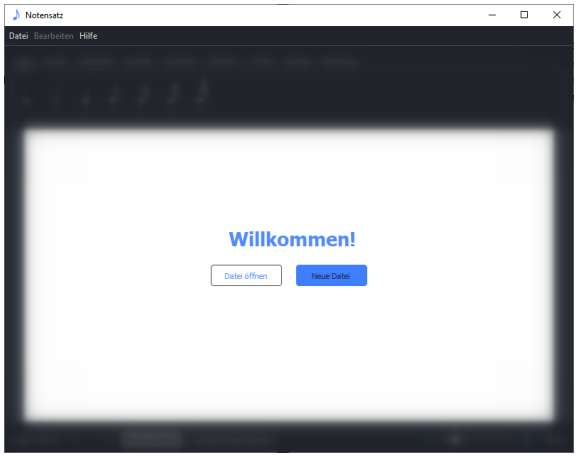


Abbildung 2: Ausdehnung (blau) und Verbindungspunkte (rot) von Noten [2]

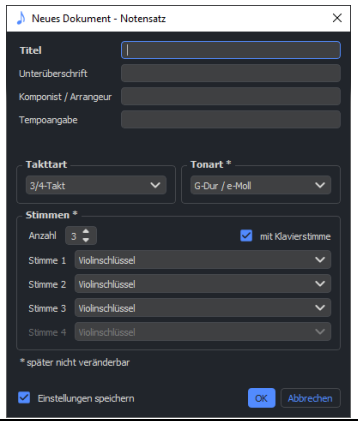
4 Ergebnisse

4.1 Programmbeschreibung (Anleitung)

4.1.1 Einführung



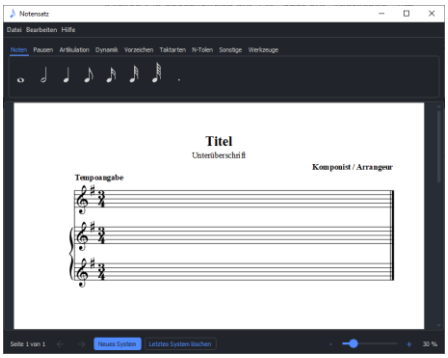
- Im Startbildschirm lässt sich zwischen der Option „Neue Datei“, um ein neues Notensatzdokument zu erstellen, und der Option „Datei öffnen“, um ein gespeichertes Notensatzdokument wieder zu öffnen, wählen.
- Bei einem Klick auf die Option „Neue Datei“ ist die sich öffnende Vorlage auszufüllen. Wenn die getätigten Einstellungen nicht verloren gehen sollen (auch nach Schließen des Programms), kann das Häkchen bei „Einstellungen speichern“ links unten gesetzt werden.



Die mit einem Sternchen gekennzeichneten Einstellungen können später nicht mehr geändert werden und sind im gesamten Dokument konstant. Mit dem Klick auf „OK“ wird ein neues Dokument erstellt, bei „Abbrechen“ gelangt man zurück zum Startbildschirm.

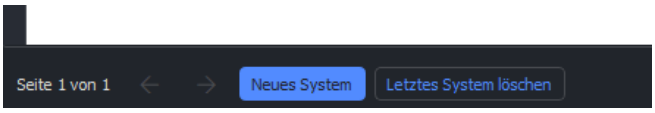
Nun kann das Notensatzdokument nach den jeweiligen Bedürfnissen befüllt, gespeichert und exportiert sowie bereits gespeicherte Notensatzdokumente geöffnet werden.

4.1.2 Bearbeiten



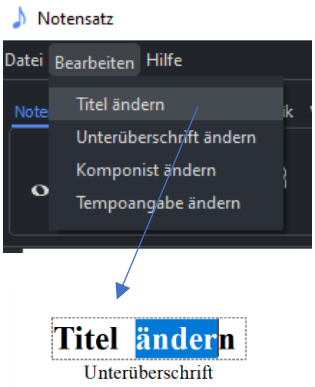
- Nachdem das neue Dokument erstellt wurde, befindet man sich nun in der Hauptumgebung des Programms.
  Unter der Menüleiste ist der „Notensatzbaukasten“, darunter die Dokumentansicht, deren Seiten, sowie die Skalierung und die Notenliniensysteme sich über die ganz unten befindliche Statusleiste ändern lassen.

- Über die zwei Buttons in der Statusleiste kann ein neues System hinzugefügt werden oder das aktuell letzte gelöscht werden.



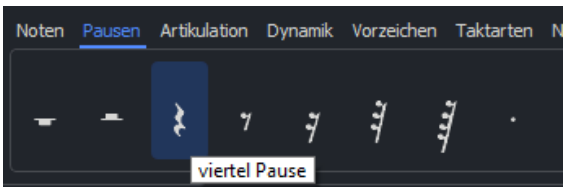
  Das Programm erstellt dabei automatisch neue Seiten, zwischen diesen lässt sich über die zwei Pfeile neben der Seiteninformation wechseln.

- Der Titel, der Untertitel, der Komponist und die Tempoangabe lassen sich mit einem direkten Klick auf das entsprechende Textfeld ändern. Generell, oder in Fällen, in denen das Textfeld leer und somit schwer zu finden ist, kann auch über die Menüleiste in die Textfelder navigiert werden.

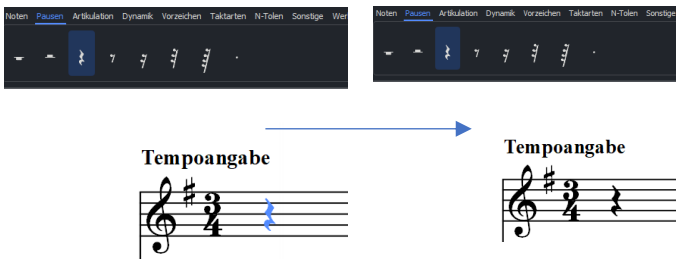


- Zum Eigentlichen Befüllen der Notenzeilen steht der „Notensatzbaukasten“ zur Verfügung. Dazu wählt man einfach den gewünschten Reiter (hier zum Beispiel „Pausen“) und klickt auf ein Symbol. Bei längerem Verweilen mit der Maus auf

einem Symbol wird eine Erklärung angezeigt, um welches musikalische Symbol es sich handelt.



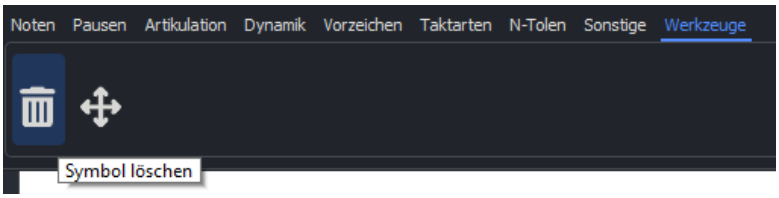
Das Programm schlägt nun automatisch freie Positionen für das ausgewählte Symbol vor, wenn man die Maus in die Dokumentansicht bewegt.



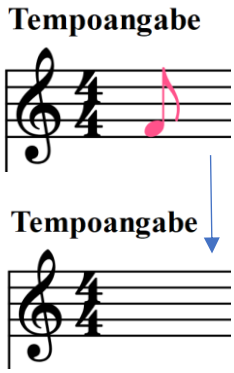
Mit einem Linksklick kann das Symbol an der gewählten Position platziert werden.

4.1.3 Löschen / Bewegen

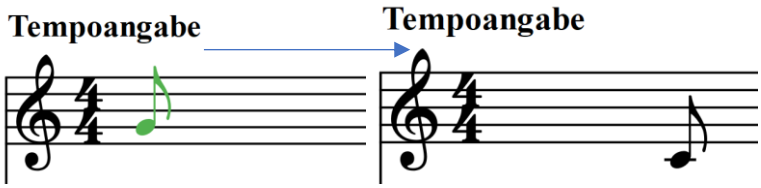
- Zum Verändern des erstellten Notensatzes können die Optionen unter dem Reiter „Werkzeuge“ genutzt werden.



- Wählt man die Option „Symbol löschen“ und fährt mit der Maus über das gewünschte Objekt, zum Beispiel eine Note, färbt sich dieses dann rosa und kann per Linksklick entfernt werden.

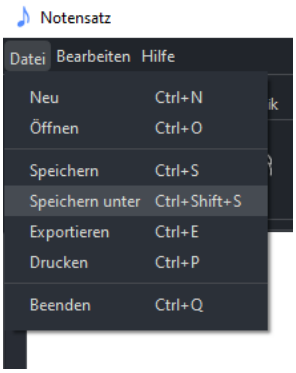


- Bei der Option „Symbol bewegen“ färbt sich das gewählte Objekt grün. Dieses wird per Linksklick beweglich und kann dann per erneutem Linksklick auf der neuen Position platziert werden.



4.1.4 Speichern / Öffnen

- Das erstellte Notensatzdokument kann über die Optionen „Speichern“, „Speichern unter“ und „Exportieren“ unter dem Reiter „Datei“ in der Menüleiste zur Vervielfältigung auf dem Gerät gespeichert werden. Dazu wählt man im sich öffnenden Dialog einfach den entsprechenden Ordner aus. Basierend auf dem Titel und dem Komponisten wird ein Dateiname mit der Dateiendung „nos“ oder „pdf“ erstellt, dieser kann entsprechend angepasst werden.



  Eine Datei im Format „nos“ kann vom Programm wieder geöffnet werden. Soll das Dokument fertiggestellt und zum Beispiel gedruckt werden, kann die Option „Exportieren“ gewählt werden, die den Inhalt des Dokuments in eine PDF-Datei exportiert. Diese kann nicht mehr vom Programm eingelesen werden.
  Ein Druckerdialog steht ebenfalls zur Verfügung.

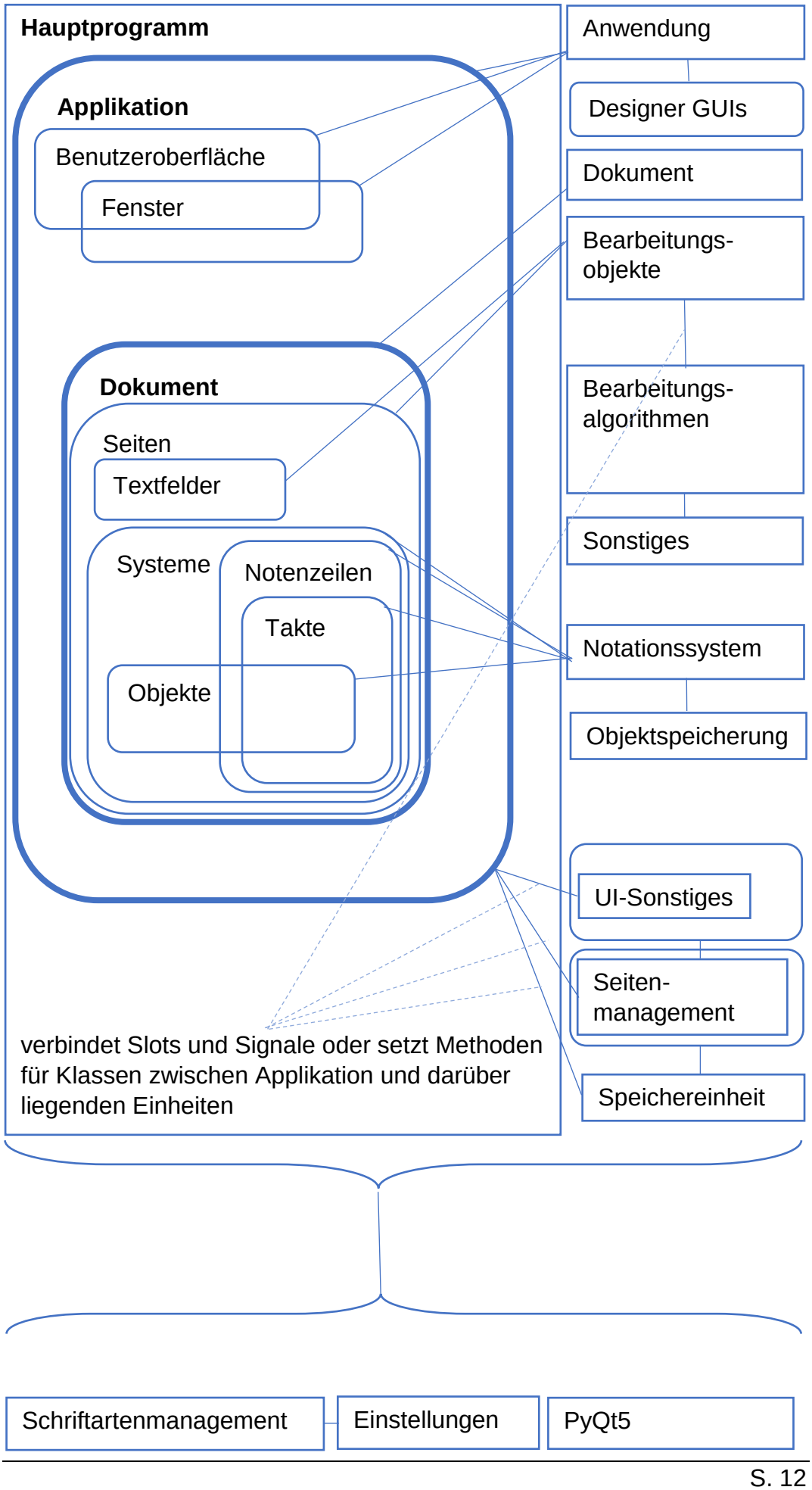
  Wie in der obigen Abbildung gezeigt, stehen für alle Optionen unter dem Reiter „Datei“ Tastenkombinationen zur Verfügung.
  Eine Datei kann über die Option „Öffnen“ unter dem Reiter „Datei“ oder über den linken Button auf dem bereits vorher beschriebenen Startbildschirm geöffnet werden.

5 Methoden

5.1 Programmstruktur

Das Programm wurde hauptsächlich objektorientiert programmiert.

Die Programmstruktur lässt sich wie folgt (vereinfacht) veranschaulichen. Zur Vereinfachung werden teilweise deutsche Übersetzungen verwendet, die so nicht im Quellcode wiederzufinden sind. Eckige Kästen symbolisieren Python-Dateien, die mehrere Funktionen und Klassen beinhalten können. Runde Kästchen symbolisieren tatsächliche Objekte.


Hauptprogramm
Applikation
Benutzeroberfläche
Fenster
Dokument
Seiten
Textfelder
Systeme
Notenzeilen
Takte
Objekte
verbindet Slots und Signale oder setzt Methoden für Klassen zwischen Applikation und darüber liegenden Einheiten
Anwendung
Designer GUIs
Dokument
Bearbeitungs-objekte
Bearbeitungs-algorithmen
Sonstiges
Notationssystem
Objektspeicherung
UI-Sonstiges
Seiten-management
Speichereinheit
Schriftartenmanagement
Einstellungen
PyQt5

5.2 Benutzeroberfläche

Die grafischen Benutzeroberflächen wurden mit dem Programm Qt-Designer erstellt und mit Hilfe des Tools pyuic5 in Python-Dateien konvertiert und später über den Programmcode noch modifiziert.

Das ansprechende Aussehen erlangt das Programm durch das PyQtDarkTheme, welches über pip installiert wurde. [5]

Es wurde sich für eine dunkelblaue Gestaltung entschieden, um dem Benutzer einen angenehmen Kontrast zum weißen Notenblatt zu bieten.

Der Notenbaukasten ist ein Tab-Widget, das mit Hilfe des erstellten Dictionary in *symbol_button.py* befüllt wird: Für jeden Eintrag in diesem wird ein `QPusButton` zum jeweiligen Tab hinzugefügt, auf den dann ein `QLabel` mit einem Musikzeichen als Text hinzugefügt wird.

Im Folgenden wird oft beschrieben, wie anstelle von Text ein Musikzeichen hinzugefügt wird, daher wird hier kurz beschrieben, wie die Funktionsweise dessen ist:

Wie in der Vorbetrachtung angemerkt, wird dafür die Musikschriftart Bravura verwendet. Die Schriftart wird über die Funktion `load_fonts()` aus *fonts.py* geladen und kann dann immer über

```
label.setFont(QFont("Bravura", 20))
```

auf ein `QLabel` angewendet werden. Soll dann der Text des Widgets auf ein Musikzeichen geändert werden, muss der Unicode bekannt sein. Dieser wird dann über die Funktion `unicode_to_string()` aus *fonts.py* in Text umgewandelt. Da es aber aufwändig und fehleranfällig wäre, für alle Musikzeichen den entsprechenden Unicode herauszusuchen, existiert eine JSON-Datei, die eine Zuordnung von Musikzeichenname und Unicode vornimmt. [6] Funktionen in *fonts.py* übernehmen hierbei das Auslesen dieser. Um beispielsweise eine Viertelnote auf einem Label darstellen zu können, muss folgendes aufgerufen werden:

```
label.setText(get_symbol(„noteQuarterUp“))
```

Für Textfelder im Dokument wurde die Schriftart Times New Roman verwendet, die zwar keine Open-Source-Lizenz besitzt, aber auf Windows-Geräten installiert sein sollte.

Im Tab „Werkzeuge“ des Notenbaukastens sind Buttons, deren Erscheinungsbild ebenfalls durch Sonderzeichen und nicht durch Bilder erreicht wird. Dafür wurde die Symbolschriftart Font Awesome verwendet.[7]

Font Awesome als auch Bravura haben Open-Source-Lizenzen, welche auch im *assets/fonts* – Ordner zu finden sind.

5.3 PyQt5 Grafiksystem

Über den Qt-Designer wurde im Hauptfenster ein `QGraphicsView` platziert. Auf diesem kann eine `QGraphicsScene`, kurz Scene, dargestellt werden, was eine Fläche für 2D-Objekte ist und auch noch bei tausenden Objekten effizient ist. Eine Scene repräsentiert also ein Notenblatt. Der Wechsel zwischen mehreren Scenes ermöglicht mehrere Notenblätter. Auf den Scenes können `QGraphicsTextItems`, `QGraphicsRectItems` und weitere `QGraphicstItems`, kurz Items, platziert werden. Außerdem sind `QGraphicsItemGroups`, kurz Groups, möglich, die mehrere Items beinhalten können. Alle Items einer Group sind relativ zur Position dieser positioniert. Dies ermöglicht, dass man mehrere Items, die immer den gleichen Abstand zu einander haben sollen, mit einem Befehl verschieben kann. Dabei kann eine Group immer auch eine weitere Group enthalten. [8]

5.4 Dokumentaufbau

Für das Dokument wurde die Klasse `DocumentUi` in *document.py* erstellt. Diese enthält eine Liste von Seiten. Die Objekte dieser Liste sind Instanzen der Klasse `Page` (*edit_items.py*), welche indirekt von der bereitgestellten Klasse `QGraphicsScene` erbt. Warum im Folgenden nur von indirekter Vererbung gesprochen wird, wird im Punkt „5.5.3 Speichereinheit“ erklärt. Das Dokument beinhaltet außerdem mehrere

`DocumentTextitems`, die von der Klasse `QGraphicsTextitem` indirekt erben, bearbeitbare Textfelder für Titel, Unterüberschrift, Komponist und Tempoangabe darstellen und nur auf der ersten Seite dargestellt werden. Alle Seiten beinhalten ein weißes `QGraphicsRectItem`, was den Hintergrund darstellt und, bis auf die erste, eine Seitenzahl, die logischerweise abwechselnd rechts und links zu sehen ist. Das Dokument beinhaltet außerdem eine Liste aller Systeme (der Klasse `System` (*notation_system.py*)) und ist für die Zuteilung dieser auf die Seiten zuständig. Im Folgenden wird die genauere Zusammensetzung dieser Systeme, also schon musikalischer Objekte, beschrieben.

5.5 Notationssystem

Alle folglich beschriebenen Klassen sind in der Datei *notation_sysem.py* zu finden.

Ein System ist eine Instanz der Klasse `System`, die indirekt von der Klasse `QGraphicsItemGroup` erbt und Items sowie Groups enthält. Wenn im Folgenden von Item beziehungsweise Groups gesprochen wird, sind Instanzen der Klasse `Musicitem` (erbt indirekt von `QGraphicsTextItem`) beziehungsweise `MusicitemGroup` oder `QGraphicsItemGroup` gemeint (eine `MusicitemGroup` ist eine spezielle `QGraphicsItemGroup`). Alle systemspezifischen Items sind: linker und rechter Taktstrich sowie die Klavierklammer. Ein System enthält eine Liste von Notenzeilen (der Klasse `Stave`), was wiederum eine Group ist und somit mehrere Items und Groups beinhaltet:

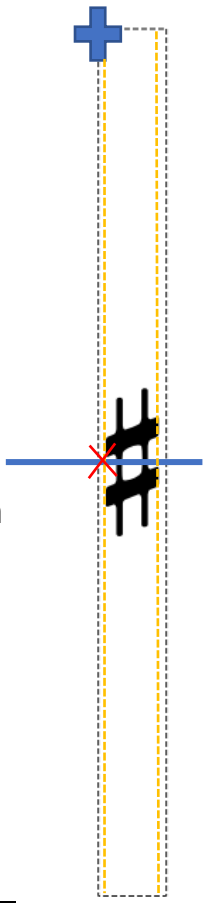
Eine Notenzeile enthält zu aller erst die fünf Notenlinien, welche indirekt Instanzen der Klasse `QGraphicsLineItem` sind, sowie einen Notenschlüssel als Item und mehrere Vorzeichen, auch Items, die die Tonart darstellen. Außerdem enthält sie eine Liste von Takten (der Klasse `Bar`), welche als Groups die strukturell kleinsten Untereinheiten bilden.

Ein solcher Takt beinhaltet manchmal eine Taktart der Klasse `TimeSignature`, immer einen linken Taktstrich als auch eine Liste zugehöriger Items und Groups, deren Klassen nicht genauer definiert sind,

da alle entweder Items oder Groups sind. So gibt es zum Beispiel die Klassen `Note`, `Rest`, `Clef` und normale `Musicitems`, die keine weiteren Eigenschaften besitzen oder spezifische Methoden benötigen.

5.6 Algorithmen

5.6.1 Positionierung



Um all diese Items und Groups genau positionieren zu können, bedarf es der Kenntnis über die genauen Maße und den Ursprungspunkt aller Musikzeichen. Eine interessante Information, die nur durch Zufall erlangt wurde, ist, dass alle Zeichen auf einer imaginären Notenlinie liegen, die genau in der Mitte der Höhe jedes Zeichens liegt (rechts im Bild blau markiert). Da aber alle Musikzeichen unterschiedliche Höhen haben und der von PyQt5 vorgegebene Ursprungspunkt (blaues Kreuz rechts) immer an der oberen linken Ecke befindlich ist, erhalten alle Zeichen eine komplett unzusammenhängende y-Positionierung. Die Lösung dessen liegt darin, von der gewollten y-Koordinate die Hälfte der Höhe des Begrenzungsrechtecks abzuziehen:

```
def set_real_y(self, ay: float):
      """get bottom stave line matching"""
      self.qt().setY(ay
self.qt().sceneBoundingRect().height() / 2)
```

Ein weiteres Problem tritt bei der x-Positionierung auf. Nach vielen Nachforschungen fiel auf, dass PyQt5 vor und hinter jedem Text in einem `QGraphicsTextItem` etwas Platz (gelb markiert) hinzufügt, welcher im Begrenzungsrechteck (schwarze Strichellinien) mit einberechnet wird. Dieser Abstand kann jedoch berechnet werden, da Bravura Metadaten bereitstellt, welche die Höhe und Breite des tatsächlich sichtbaren Zeichens über zwei Koordinatenpunkte definieren:

```
def get_character_width(key):
     box = get_specification("glyphBBoxes", key)
     return Musicitem.spec_to_px(box["bBoxNE"][0] -
box["bBoxSW"][0])


def get_qt_blank_space(self):
     key_ = self.key
     real_width = 0
     qt_width = self.qt().sceneBoundingRect().width()
     […]
     real_width = Musicitem.get_character_width(key_)
     real_width *= self.qt().transform().m11()
     return (qt_width - real_width) / 2
```

Daraus kann dann eine Funktion erstellt werden, die das Item an die richtige x-Koordinate, also bündig zum sichtbaren Teil des Zeichens, positioniert:

```
def set_real_x(self, ax: float):
        self.qt().setX(ax - self.get_qt_blank_space())
```

Jedes Musikzeichen kann nun relativ zu einem sinnvollen Punkt (voriges Bild rotes X) in x- und y-Richtung positioniert werden.

5.6.2 Speichereinheit

Aufgrund der enormen Komplexität des Programms ist es nicht trivial, einen Speicheralgorithmus und ein Dateiformat zu finden. Es gibt bereits internationale Standards im Bereich der Notensatzdateiformate, allerdings wäre eine Implementierung eines solchen zu zeitaufwändig gewesen. Eine manuelle Konvertierung der Objektstrukturen samt aller Attribute in ein Textformat wäre dies ebenfalls.

Daher wurde entschieden, die in Python integrierte Bibliothek pickle zu verwenden. Diese erlaubt es einem, Objekte in eine Binärdatei zu schreiben (serialisieren) und wieder zu laden (deserialisieren), ohne, dass jegliche Attribute verloren gehen. Dasselbe kann einfach mit dem Dokumentobjekt getan werden (*saving.py*):

```
def save_data(self, file_name):
        with open(file_name, "wb") as f:
            pickle.dump(self.app.document_ui, f)
```

Allerdings gibt es damit ein Problem: Einige Objekte sind nicht serialisierbar, was das Programm zum Absturz bringt. Dazu gehören primär alle `QGraphicsItems`, was bedeutet, dass gerade die grafischen Objekte nicht gespeichert werden würden. Um das zu verhindern wurde das System der „indirekten“ Vererbung und Verwendung dieser Klassen geschaffen. Die folgende Beschreibung bezieht sich auf *qt_saving_layer.py*. Alle nicht speicherbaren Klassen erhalten eine gleichnamige mit dem Prefix „N_“. Diese speichern lediglich einen Schlüssel für einen Itemcontainer, in dem die tatsächlichen `QGraphicsItems` gespeichert werden. Wird nun ein solches Objekt in eine Binärdatei geschrieben, wird die Methode `__getstate__` von der Bibliothek pickle ausgeführt und alle wichtigen Eigenschaften des Items werden an pickle übermittelt. Hier beispielhaft für eine `QGraphicsScene`:

```
class N_QGraphicsScene(N_GraphicsObject):
     def __init__(self, item: QGraphicsScene):
        super().__init__(item)

    def qt(self) -> QGraphicsScene:
        return self._qt()

    def __getstate__(self):
        self._rect = self.qt().sceneRect()
        return self.__dict__
```

```
    def __setstate__(self, d):
        self.__dict__ = d
        self._init(QGraphicsScene(self._rect))
```

Beim Lesen der Binärdatei wird die Methode `__setstate__` aufgerufen, die ein `QGraphicsItem` zum Itemcontainer hinzufügt und dessen Schlüssel speichert. [9]

Über die Methode `qt` kann dann von überall über dieses Hilfsobjekt auf das `QGraphicsItem` zugegriffen werden.

5.6.3 Editierungseinheit

Registriert die aktuelle Scene, dass sich die Maus bewegt hat, wird die Funktion `edit_update` in der Datei *editing.py* ausgeführt. Diese startet wiederum komplexe Algorithmen, die überprüfen, wo das aktuell ausgewählte Item platziert werden kann. Hierbei wurde darauf geachtet, dass beispielsweise Noten nur auf Notenlinien und Zwischenräumen sowie Hilfslinien platziert werden können und dass sich mehrere Items im Takt nicht überschneiden können. Andererseits soll das Programm Schülern, die die Hauptzielgruppe darstellen, nicht zu viel abnehmen.

Ein Item wird dann platziert, wenn der Nutzer die Maustaste drückt. Dann wird die Funktion `edit_pressed` ausgeführt.

Damit hier nicht jeder Algorithmus bis ins kleinste Detail dokumentiert werden muss, wurden für alle Variablen selbsterklärende Bezeichner genutzt sowie der Code zeilenanzahlentsprechend kommentiert.

Eine interessante Anmerkung ist, dass für den Algorithmus zum Bewegen von Objekten lediglich zwei bestehende Algorithmen kombiniert wurden: Klickt der Nutzer auf das gewünschte Objekt zum Bewegen, wird der Algorithmus zum Löschen eines Objekts ausgeführt und direkt danach der Algorithmus zum neuen Hinzufügen eines Objektes, bis der Nutzer wieder klickt und das neue Item auf der neuen Position platziert wird.

5.7 Kompilierung

Um das Programm Testpersonen und Nutzern einfach zur Verfügung stellen zu können, musste ein geeignetes Format zum Verschicken gefunden werden. Hätte man die bloßen Python-Dateien verschickt, hätte es einer aufwendigen Installation von der richtigen Python-Version und den verwendeten Bibliotheken wie PyQt5 und dem PyQtDarkTheme bedurft. Um dies zu umgehen, wurde das Skript *create_exe.py* erstellt, welches nach Installation des PyInstallers über pip beim Ausführen aus dem gesamten Projekt eine einzige ausführbare EXE-Datei erstellt. [10] Diese ist unabhängig von den Python-Dateien, aber nicht von den Ressourcen wie dem Icon oder den Schriftarten. Es kann aber ein Link zum Herunterladen des ZIP-Archivs vom GitHub Speicherort versendet werden. Nach dem Entpacken kann einfach aus dem Projektordner heraus auf die EXE-Datei geklickt werden, um das Programm zu starten. Nach dem Hinzufügen und Testen neuer Funktionen wird dieses Skript immer ausgeführt, um die EXE aktuell zu halten.

6 Diskussion

6.1 Diskussion des Ergebnisses

Im vorgegebenen Zeitraum wurden alle Muss-Kriterien sowie der Großteil der Soll-Kriterien erfüllt. Aufgrund fehlender Zeit wurden diese Kriterien gegenüber den vielen möglichen Kann-Kriterien priorisiert. Eine gute Nutzerfreundlichkeit wurde allemal erzielt, das Programm wurde von vier Musikern getestet. Beim Erstellen wurde auch darauf Wert gelegt, dass die hinzugefügten Funktionen einwandfrei funktionieren und nicht viele Fehler mit sich bringen, umso künftige Erweiterungen zu erleichtern.

Im Rahmen der Projektarbeit wurden so über 3000 Zeilen Code geschrieben.

Um die Inhalte der geleisteten Arbeit angemessen und verständlich zu behandeln und zu dokumentieren, musste die Vorgabe von 10-12 Seiten

für den Textteil leider überschritten werden, da auch das Hinzufügen von Bildmaterial essentiell war.

6.2 Erweiterungsmöglichkeiten

Zunächst sollten alle Kann-Kriterien implementiert werden, um das Projekt vollständig abzurunden. Diese sind in den Planungsunterlagen nachzulesen. Beim Erstellen der Software viel auf, dass in den Planungsunterlagen zwei wichtige Punkte vernachlässigt wurden, was zum einen das Hinzufügen von Akkorden sowie das Verbinden von Fähnchen der kürzeren Notenwerte waren. Es wäre für den Nutzer enorm praktisch, wenn diese zwei Punkte eine Integration in das Projekt erfahren würden.

7 Quellen

7.1 Textquellen und Links

[1] Noah Weiler. (2023). *notensatz*. GitHub. verfügbar unter: https://github.com/NO411/notensatz. [29.05.2023].

[2] Wikipedia. (2022). *SMuFl*. Wikipedia: The Free Encyclopedia. verfügbar unter: https://en.wikipedia.org/wiki/SMuFL. [29.05.2023].

[3] Daniel Spreadbury. (2021). *Bravura Releases*. GitHub. verfügbar unter: https://github.com/steinbergmedia/bravura/releases. [22.03.2023].

[4] Daniel Spreadbury. (2022). *Home*. Standard Music Font Layout (SMuFL). verfügbar unter: https://w3c.github.io/smufl/latest/index.html. [29.05.2023].

[5] PyPI. (2023). *PyQtDarkTheme*. PyPI. verfügbar unter: https://pypi.org/project/pyqtdarktheme/. [19.03.2023].

[6] Daniel Spreadbury. (2021) *glyphnames.json*. GitHub. verfügbar unter: https://github.com/w3c/smufl/blob/gh-pages/metadata/glyphnames.json. [13.04.2023].

[7] Font Awesome. (o. J.). *Downloads*. Font Awesome. verfügbar unter: https://fontawesome.com/download. [08.05.2023].

[8] The Qt Company Ltd. (2023). *Graphics View Framework*. Qt Documentation. verfügbar unter: https://doc.qt.io/qt-6/graphicsview.html. [21.03.2023].

[9] Python Software Foundation. (2023). *pickle – Python object serialization*. Python. verfügbar unter: https://docs.python.org/3/library/pickle.html. [18.04.2023].

[10] David Dortesi. (o. J.). *PyInstaller Manual*. PyInstaller. verfügbar unter: https://pyinstaller.org/en/stable/. [28.05.2023].

7.2 Abbildungen

[1] From Chapter 15 “Symbols”. The Unicode Standard. Version 6.2. Ed. Julie D. Allen et al. Mountain View. The Unicode Consortium. (2012). verfügbar unter: https://w3c.github.io/smufl/latest/media/precomposed-notes-unicode.png (abgerufen 29.05.2023)

[2] Daniel Spreadbury. (2022). Example of glyph registration for notes with flag. verfügbar unter: https://w3c.github.io/smufl/latest/media/notehead-stem-flag.svg (abgerufen 29.05.2023)

Alle anderen Abbildungen sind Bildschirmaufnahmen der erstellten Software oder wurden anderweitig selbst erstellt.

8 Selbständigkeitserklärung

Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne fremde Hilfe verfasst und keine anderen Hilfsmittel als die angegebenen verwendet habe. Insbesondere versichere ich, dass ich alle wörtlichen und sinngemäßen Übernahmen aus anderen Werken als solche kenntlich gemacht habe.

Ort: Leipzig

Datum: 30.05.2023 Unterschrift: