UNIVERSITÄT
LEIPZIG

Softwaretechnikpraktikum

Entwurfsbeschreibung

Gruppe: nw19a

Mitglieder: Thomas Pause, Sabine Lorius, Arik Korte, Martin George, Josephine Lange, Esther Prause, Anh Kiet Nguyen, Bärbel Hanle

Verantwortlich: Sabine Lorius & Anh Kiet Nguyen

Betreuer: Dr. Nicolas Wieseke

Tutor: Martin Frühauf

Abgabedatum: 02.03.2020

Version: 0.7 (Release 3)

Stand: 2. März 2020

Inhaltsverzeichnis

1 Visionen und Ziele

Ziel ist die Entwicklung einer Anwendung als *Proof-Of-Concept*, die es *Poi*-Spielern ermöglicht zu berechnen, auf welche Weise beliebige *Poi*-Figuren erzeugt werden können (beschränkt auf den 2-dimensionalen Raum). Als einfachste Stufe wurde im Vorprojekt eine Funktionalität erstellt die es ermöglicht mit beliebigen Eingabeparametern sog. *Flowers* zu berechnen. Dies wird durch eine Zeicheneingabe über das Touchpad erweitert und um eine Bibliothek zum Speichern und Laden von *Flowers* und Kurven ergänzt.

2 Rahmenbedingungen und Produktübersicht

2.1 Rahmenbedingungen

Die Anwendung wird als Android App mit einer kleinsten unterstützten Display-Auflösung von 768x1366 Pixel (HD) und dem Minimum *API-Level* 25 (Android 7.1.1) umgesetzt. Als Programmiersprache kommt maßgeblich *Kotlin* zum Einsatz.

2.2 Produktübersicht

Die im Release 1 umgesetzte App bietet dem User die Möglichkeit, über die frei veränderlichen Eingabeparameter **radius** (Verhältnis der Radien Poikreis zu Handkreis), **freq1** (Frequenz des Handkreises), **freq2** (Frequenz des Poikreises) und **rotation** (Drehung der *Flower*) beliebige *Flowers* zu generieren. Die zur Darstellung einer solchen *Flower* nötige Handbewegung eines *Poi*-Spielers kann daraufhin (zunächst ohne jede Berücksichtigung physikalischer Aspekte) berechnet und gemeinsam mit der *Flower* graphish präsentiert werden. Im Release 2 wird über die Touchpadeingabe die Funktion bereitgestellt frei Hand eigene geschlossene Linienzüge zu zeichnen und diese als parametrisierte Kurven, ebenso wie erzeugte Flowers, in einer Bibliothek abzuspeichern und zu verwalten. Um eine fließende Navigation in der an Funktionalität gewachsenen App zu gewährleisten wurde ein *Navigation Drawer* implementiert. Release 3 ergänzt den Funktionsumfang der App zusätzlich um einen Live-Modus. Farbige Animationen der errechneten Poi- sowie Handkurven können in auswählbaren Geschwindigkeiten abgespielt und an beliebigen Stellen angehalten werden.

3 Grundsätzliche Struktur- und Entwurfsprinzipien

Die verschiedenen Funktionen der App sind in separaten Paketen organisiert. So gibt es die Pakete **flowers**, **drawByHand**, **livemode** und **library**. Jedes dieser Paket enthält ein Fragment, das die grafische Oberfläche der entsprechenden Funktion der App bereitstellt und dafür alle UI-Elemente instantiiert und auf diese, die für die User-Interaktion benötigten, EventHandler angefügt. Zu jedem dieser Fragments existiert eine `layout_<paket>_fragment.xml`, die das Layout des jeweiligen Fragments vorgibt. Des Weiteren sind im Paket **utility** einige Methoden und Klassen zusammengefasst, die eine Implementierung von häufig genutzten Konzepten vereinfachen. Im **curvetohand** Paket ist die `CurveToHand`-Klasse, die Funktionen zur Berechnung von Handkurven bereitstellt.

Flowers-Menü Dieses Menü wird neben dem `FlowersFragment` durch zwei weitere Klassen realisiert: Die `FlowersCalc`-Klasse berechnet auf Grundlage der vom User eingegebenen Werte für Radius, Hand- und Poifrequenz sowie Rotation, die Koordinaten der entsprechenden *Flower*. Das Ergebnis wird in der Klasse `FlowersFragment` entgegengenommen und an eine Instanz der `FlowersCanvasView` weitergeleitet. Diese Klasse stellt einen Canvas auf der Benutzeroberfläche zur Verfügung, auf dem die

Flower gerendert werden kann. Eine *Flower* kann nun auch durch Betätigung des **save**-Buttons in der Bibliothek gespeichert werden.

DrawByHand-Menü Das `DrawByHandFragment` hält eine Instanz der `PaintView`, die es dem User erlaubt, auf dem Bildschirm des Gerätes zu zeichnen. Die dafür nötigen TouchEventHandler werden innerhalb des Fragments zur Verfügung gestellt. Der vom User gezeichnete Pfad wird nach Anheben des Fingers automatisch geschlossen, da die Berechnung einer Handbahn auf Grundlage dieses Pfades perspektivisch nur mit geschlossenen Kurven funktionieren kann. Der Pfad kann durch Betätigung des **save**-Buttons als parametrisierte Kurve in der Bibliothek gespeichert oder durch Betätigung des **clear**-Buttons zurückgesetzt werden.

Bibliothek-Menü Zur Bibliothek gehören neben dem `LibraryFragment` noch der `FileHandler.kt` sowie die `LibraryEntryView` und ein `Converter`. In der Bibliothek können sowohl gespeicherte *Flowers* als auch vom User gezeichnete und abgespeicherte Pfade aufgerufen werden. Eine Filterfunktion erlaubt das Ein- und Ausblenden von Dateien aus den einzelnen Kategorien. Ebenso ist es möglich den Dateinamen einzelner Einträge zu verändern oder einen Eintrag ganz zu löschen. Das Layout einzelner Bibliothekseinträge wird in der xml-Datei `library_entry_view.xml` definiert. Die benötigten Funktionen zum Laden, Bearbeiten und Löschen werden von der `LibraryEntryView.kt`-Klasse bereitgestellt. Der `FileHandler` ist die Schnittstelle der Bibliothek zum internen Speicher der App. Er beinhaltet Methoden zum Anlegen, Löschen, Schreiben und Lesen von Dateien. Ein `Converter`-Objekt stellt Funktionen bereit, um aus Strings Objekte zu erstellen und umgekehrt. Diese Funktionen können genutzt werden, um z.B. eine gespeicherte Touchpad Kurve aus der Bibliothek aufrufen zu können.

LiveMode-Menü Das `LiveModeFragment` hält eine Instanz der `AnimationCanvasView`, die es erlaubt, eine *Poi*-Bahn und die dazugehörige Hand-Bahn sowohl statisch, sowie auch als Animation darzustellen. Im `livemode`-Packet ist des Weiteren die Klasse `LiveModeViewModel` zu finden, die sowohl im `LiveModeFragment`, als auch im `DrawByHandFragment` und im `FlowersFragment` referenziert wird. Diese Klasse erlaubt es Daten über den Lebenszyklus eines Fragmentes hinaus zu erhalten und somit Informationen zwischen Fragmenten auszutauschen. Diese Eigenschaft wird genutzt, um dem `LiveModeFragment` mitzuteilen, welche `Flowers`, bzw. welche parametrisierte Kurve als Animation dargestellt werden soll.

curvetohand-Paket Die `curveToHand`-Klasse stellt Funktionen bereit um mit Hilfe komplexer *Fourieranalyse* aus einer gegebenen Poikurve die zugehörige Handbewegung zu berechnen. Insofern ist sie die Schnittstelle zwischen dem DrawByHand- und dem LiveMode-Menü.

4 Struktur- und Entwurfsprinzipien einzelner Pakete

4.1 Struktur (Arbeitspaket 1)

Die Struktur der Anwendung wurde mit der Implementierung des *Navigation Drawer*s noch einmal grundlegend überarbeitet. Es gibt nun nur noch eine Activity, die `MainActivity`, welche die einzelnen *Fragment*s mit Ihren zugehörigen Layouts lädt und verwaltet. Die Navigation, also die Wegeführung durch die App, wird nun durch die Datei `navigation/mobile_navigation.xml` definiert.
Arbeitspaket 1 enthält unter anderem die Erstellung von Activities. Diese wurden in Release 2 entsprechend durch Fragments abgelöst. Aus Release 1 betrifft das die `FlowerActivity`-Klasse, welche jetzt durch das `FlowersFragment` repräsentiert wird. Das zugehörige Layout wird in der Datei `layout_flowers_fragment.xml` festgelegt. Dieses Fragment nimmt als User-Eingaben die Parameter

Radius, Handfrequenz, Poifrequenz und den Rotation (Drehung der Flower) entgegen und stellt dazu eine *Flower* auf dem Bildschirm dar. Der Nutzer kann dabei zwischen einer Darstellung mit und ohne Handkreis wählen.

Entscheidungen:

- Mit Release 3 wurde die Bibliothek als Startfragment und Einstieg in die App festgelegt.

4.2 Kurveneingabe (Arbeitspaket 2)

Aus Arbeitspaket 2 wurden die Eingabe mittels Touchpadzeichnung und die Erzeugung einer parametrisierten Kurve aus dieser Touchpadeingabe implementiert. Die `layout_drawbyhand_fragment.xml` legt das Layout der Zeichenfläche fest. Die entsprechenden Funktionen befinden sich in der `PaintView`-Klasse. Die folgenden Tabellen geben eine Übersicht über die Variablen eines `PaintView`-Objekts sowie die von ihm bereitgestellten Funktionen.

Variable	Bedeutung
`drawPaint`	Stil und Farbe einer Kurve
`path`	Koordinaten der Kurve
`pathMeasure`	messbare Eigenschaften eines Path-Objekts
`lastPosX`	Position an der x-Koordinate
`lastPosY`	Position an der y-Koordinate
`TOLERANCE`	Mindestdistanz zum letzten Punkt
`INCREMENT_LENGTH_FOR_SAMPLING`	Länge für das Sampling der Punkte aus dem Path-Objekt
`pointList`	Liste der Koordinaten des Path-Objekts

Funktion	Ausgabe
`onDraw(canvas)`	Rendern des gezeichneten Path-Objekts auf der Canvas
`onActionDown(posX,posY)`	Path-Objekt wird initialisiert und mittels posX und posY die Position auf dem Touchpad gesetzt
`onActionDrag(posX,posY)`	verbindet die Punkte posX und posY mit `lastPosX` und `lastPosY`
`onActionUp()`	schließt das Path-Objekt, damit eine geschlossene Kurve entsteht
`onClear()`	löscht die Informationen des Path-Objekts
`onCalculate()`	gibt die Liste der Koordinaten des Path-Objekts zurück
`samplePath()`	erstellt die Liste der Koordinaten und sampled das Path-Objekt
`Path.connectPointsViaQuadraticBezier (fromX,fromY,toX,toY)`	Erweiterung der Path Funktionalität, um die Punkte mittels *Bézierkurve* zu verbinden

4.3 Berechnungen (Arbeitspaket 3)

Aus Arbeitspaket 3 wurden im Flowers-Paket die Berechnung der *Flower*- und der Handkoordinaten für den *Flowers*-Bereich umgesetzt sowie im curvetohand-Paket die Berechnung einer Handkurve aus einer vorgegebenen Poikurve.

4.3.1 Berechnungen im Flowers-Menü

Die Funktionen zur Berechnung von Poi- und Handbewegungen für Flowers finden sich in der `FlowersCalc`-Klasse. Die folgenden Tabellen geben eine Übersicht über die Variablen eines `FlowersCalc`-Objekts, sowie die von ihm bereitgestellten Funktionen.

Variable	Bedeutung
`radius`	Verhältnis von Poikreisradius zu Handkreisradius
`freq1`	Frequenz des Handkreises (ganzzahlig)
`freq2`	Frequenz des Poikreises (ganzzahlig)
`rotation`	Drehung der *Flower* in Grad
`steps`	Anzahl der berechneten Punkte auf der Kurve

Funktion	Ausgabe
`greatestCommonDivisor()`	größter gemeinsamer Teiler der Frequenzen (s.u.)
`xHand(t)`	x-Koordinate der Hand im Schritt t (ohne Drehung)
`yHand(t)`	y-Koordinate der Hand im Schritt t (ohne Drehung)
`xFlower(t)`	x-Koordinate des *Poi*-Kopfs im Schritt t (ohne Drehung)
`yFlower(t)`	y-Koordinate des *Poi*-Kopfs im Schritt t (ohne Drehung)
`calcHand()`	Liste der Koordinaten, die von der Hand durchlaufen werden (nach Drehung)
`calcFlower()`	Liste der Koordinaten, die vom *Poi*-Kopf durchlaufen werden (nach Drehung)

Entscheidungen:

- Wie implizit im Lastenheft festgelegt rechnen wir mit dem Radius des Handkreises als Längeneinheit.
- Aus mathematischen Gründen haben wir uns dafür entschieden, mit zwei ganzzahligen Frequenzen für Hand- und Poikreis zu rechnen, statt wie im Lastenheft vorgesehen mit einem reellen Parameter ω, der das Verhältnis beider Frequenzen darstellt (siehe dort `/LFMF3/`). Auf diese Weise können wir sicherstellen, dass wir immer periodische Kurven erhalten. Da für die Kurven von Hand- und *Poi*-Bewegung nur das Verhältnis dieser Frequenzen von Bedeutung ist dividieren wir an jeder Stelle, wo diese Frequenzen verwendet werden, durch deren größten gemeinsamen Teiler. Dies betrifft die Funktionen `xHand`, `yHand`, `xFlower` und `yFlower`. Dadurch können wir sicherstellen, dass in der vorgegebenen Anzahl Schritte die *Flower* genau einmal durchlaufen wird.
- Da interessante *Flowers* für Radien nahe 1 erwartet werden, schränken wir den Bereich für die Werte, die dieser Parameter entgegen nimmt, auf Werte zwischen 0 und 3 ein.
- Im Gegensatz zu früheren Versionen geben wir die Drehung der *Flower* in Grad an. Die Kurven werden erst durch die Funktionen `calcHand` und `calcFlower` gedreht. In die Berechnungen der einzelnen Koordinaten in den Funktionen `xHand`, `yHand`, `xFlower` und `yFlower` geht die Drehung noch nicht ein.

4.3.2 Berechnungen im curvetohand-Paket

Die Klasse `CurveToHand.kt` im `curvetohand`-Paket stellt Funktionen bereit, um mit Hilfe von *Fourieranalyse* wie in [1] beschrieben aus einer gegebenen Poikurve einen Vorschlag für die Handbewegung zu errechnen. Dazu werden zu einer gegebenen Kurve die Fourierkoeffizienten berechnet. Der Betragsmäßig größte Koeffizient wird als Hauptradius bezeichnet, der zugehörige Index als Hauptfrequenz und das Argument (Winkel zur reellen Achse) dieses Koeffizienten als Hauptphasenverschiebung. Wir nehmen an, dass sich der Poikopf relativ zur Hand auf einer Kreisbahn mit dem Hauptradius als Radius, der Hauptfrequenz als Frequenz und der Hauptphasenverschiebung als Phasenverschiebung bewegt. Diese Kreisbahn heißt dann Hauptkreis. Die Handbahn ergibt sich dann als Differenz aus der gewünschten Poibahn und dem Hauptkreis.

Variable	Bedeutung
`poiCurve`	Kurve, zu der die Handbewegung berechnet werden soll
`freqMin`	kleinste Frequenz des Bereichs, in dem nach der Hauptfrequenz gesucht wird
`freqMax`	größte Frequenz des Bereichs, in dem nach der Hauptfrequenz gesucht wird
`periodDuration`	Zeit zum vollständigen Durchlaufen der Poikurve
`pi`	π als Float-Zahl
`delta`	Genauigkeit mit der Float-Zahlen auf Gleichheit getestet werden sollen

Funktion	Ausgabe
`fourier(freq)`	Fourierkoeffizient als komplexe Zahl in kartesichen Koordinaten
`mainCircle()`	Frequenz, Radius und Phasenverschiebung des Hauptkreises
`calcHand()`	Liste der berechneten Handkoordinaten

Entscheidungen:

- Wenn mehrere Koeffizienten den gleichen Betrag haben, wird der kleinere der Indizes als Hauptfrequenz gewählt.
- Die Zeiteinheit wählen wir so, dass ein Punkt in der Koordinatenliste der Poikurve in einem Zeitschritt durchlaufen wird.

Komplexe Zahlen werden in dieser Klasse als Paare von Float-Zahlen dargestellt. Dabei können der erste und zweite Eintrag entweder als Realteil und Imaginärteil oder als Betrag und Argument interpretiert werden. Im ersten Fall spricht man von einer komplexen Zahl in kartesischer Darstellung, im zweiten von einer komplexen Zahl in Polarkoordinaten. Für Umrechnungen zwischen diesen Darstellung gibt es im `utility`-Paket ein `Complex`-Objekt. Die folgende Tabelle gibt eine Übersicht über die Funktionen dieses Objekts.

Variable	Bedeutung
`pi`	π als Float-Zahl
`delta`	Genauigkeit, mit der Float-Zahlen auf Gleichheit getestet werden sollen (wichtig für Fallunterscheidungen)

Funktion	Ausgabe
`calcRadius(numberCart)`	Betrag der in kartesischen Koordinaten angegebenen komplexen Zahl `numberCart`
`calcAngle(numberCart)`	Argument (Winkel zur reellen Achse) der in kartesischen Koordinaten angegebenen komplexen Zahl `numberCart`
`calcRe(numberPolar)`	Realteil der in Polarkoordinaten gegebenen komplexen Zahl `numberPolar`
`calcIm(numberPolar)`	Imaginärteil der in Polarkoordinaten gegebenen komplexen Zahl `numberPolar`

4.4 Live-Modus (Arbeitspaket 4)

In der generischen Klasse `CanvasView` werden Methoden definiert, die die statische graphische Darstellung einer parametrisierten Kurve auf einem Canvas ermöglichen. Diese Funktionalitäten bilden den Grundstein für eine Animation von Poi- und Handkurve.
`CanvasView` erbt von SurfaceView. Statt des Android Runtime-Systems kann ein separater Thread innerhalb der Anwendung die `onDraw`-Methode einer SurfaceView aufrufen. Dadurch lassen sich später die Zeitpunkte für die schrittweise graphische Anzeige der Animationen kontrollieren.
Die Funktion `pointsToPath` übersetzt eine Liste von Koordinaten in ein Path-Objekt. Dieses kann dann mit den klasseneigenen Methoden `scale` und `translate` flexibel auf die Größe der Canvas angepasst werden, ohne dass eigens komplexe Funktionen geschrieben werden müssen, die diese Berechnungen durchführen. Path stellt zudem nicht nur für Animationen eine Reihe von Instrumenten zur Verfügung, sondern bietet sich auch als mögliche Datenstruktur für das Abspeichern graphischer Objekte an.
Die im `LiveModeFragment` instantiierte Klasse `AnimationCanvasView` erbt von `CanvasView` und implementiert darüber hinaus Methoden, um den Thread zu steuern, der für nebenläufige Verarbeitung der Animation zuständig ist. Die `update`-Methode wird vom aufrufenden Fragment aufgerufen und erwartet zwei Listen von Koordinaten, eine für die *Poi*-Bahn, die andere für die Handbahn. Die einzelnen Punkte werden dann zum einen in ein Pfadobjekt umgewandelt, auf das die Transformationsmethoden der vererbenden Klasse angewendet werden. Zum anderen werden die Punkte mit der Methode `transformPointsInPointList` einzeln skaliert und verschoben. Das Speichern dieser Punkte in Listenform ermöglicht für die Animation darüber zu iterieren. Dies geschieht in der `updatePos`-Methode, die 60 mal in der Sekunde über den Thread aufgerufen wird und je nach eingestellter Wiedergabegeschwindigkeit mit n-großen Sprüngen über die Koordinatenlisten iteriert. Auf der aktuellen Position auf der Handbahn wird ein Kreis dargestellt, die aktuelle Position auf der *Poi*-Bahn wird in einer separaten Liste (`poiTrace`) abgelegt. Dies erlaubt die letzten x Punkte der *Poi*-Bahn als Schweif darzustellen. Die `interpolateColors` interpoliert zwischen zwei gegebenen Farben, um den *Poi*-Schweif als Farbverlauf darstellen zu können. Des Weiteren sind noch einige Methoden implementiert, die die Animation selbst steuern und die Möglichkeit bieten, diese unabhängig vom Thread zu pausieren oder zu stoppen.

4.5 Bibliothek (Arbeitspaket 5)

Aus Arbeitspaket 5 wurden die meisten Features, die die Bibliothek bieten soll, umgesetzt.

FileHandler Die `FileHandler`-Klasse bildet die Schnittstelle zum internen Speicher der App. Es werden Methoden zum Schreiben, Lesen, Löschen und Verändern von Dateien bereitgestellt. Zusätz-

lich gibt es Funktionen, die Informationen über abgespeicherte Dateien liefern um diese in der Bibliothek darstellen zu können. Die folgenden Tabellen geben eine Übersicht über die Variablen eines `FileHandler`-Objekts, sowie die von ihm bereitgestellten Funktionen.

Variable	Bedeutung
`context`	Kontext des `FileHandlers`
`directory`	Name des Verzeichnisses, aus dem die Dateien gelesen werden/ in das sie gespeichert werden

Funktion	Aufgabe
`writeFile(fileName, data)`	legt eine neue Datei an und füllt sie mit Daten
`appendFile(fileName, data)`	fügt einer existierenden Datei neue Daten hinzu
`readFile(fileName)`	liest Inhalt einer Datei
`deleteFile(fileName)`	löscht eine Datei
`renameFile(oldName, newName)`	benennt eine Datei um
`getFileNames()`	gibt eine Liste aller abgespeicherten Dateien zurück
`fileExists(fileName)`	prüft, ob eine Datei mit diesem Namen vorhanden ist
`isValidFileName (intendedName)`	prüft, ob ein eingegebener String einem gültigen Dateinamen entspricht
`getFileTitle(fileName)`	gibt den Namen einer Datei zurück (ohne Endung)
`getFileFormat(fileName)`	gibt Format (Flower oder Kurve) einer Datei zurück

LibraryEntryView Die Layout-Datei `library_entry_view.xml` legt das Layout eines einzelnen Bibliothekseintrages fest, welcher aus einem Thumbnail, dem Dateinamen, einem Edit- und einem Delete-Button besteht. Die `LibraryEntryView`-Klasse hinterlegt diese Elemente mit Funktionalität. Bei Klick auf den Dateinamen wird je nach Dateiformat eine Flower mit den entsprechenden Parametern im Flowers-Menü geladen oder eine durch Touchpadeingabe erstellte Kurve im DrawByHand-Menü angezeigt. Die folgende Tabelle gibt einen Überblick über Variablen, die zu einem Bibliothekseintrag gehören.

Variable	Bedeutung
`context`	Kontext des Bibliothekeintrages
`title`	Dateiname
`format`	Dateiformat (z.B. Flower, Kurve)
`library`	Bibliothek, zu der der Eintrag gehört
`textView`	TextView zur Darstellung des Dateinamens
`editButton`	Edit-Button zur Umbennung der Datei
`deleteButton`	Delete-Button zum Löschen der Datei

LibraryFragment Alle Bibliothekseinträge erscheinen in einer Scroll-View, die in der `layout_library_fragment.xml`-Datei erstellt wird. In dieser Datei sind außerdem zwei Checkboxen zur Auswahl der in der Bibliothek angezeigten Dateiformate angelegt. Die zugehörige Kotlin-Datei `LibraryFragment.kt` definiert die Funktion dieser Elemente. Die folgenden Tabellen geben einen Überblick über Variablen und Funktionen der `LibraryFragment`s.

Variable	Bedeutung
`libraryList`	Liste die die `LibraryEntryViews` beinhaltet
`flowerCheckBox`	Checkbox für *Flower*-Dateien
`touchPadCheckBox`	Checkbox für Touchpad-Dateien.
`fileHandler`	Schnittstelle zum internen Speicher

Funktion	Aufgabe
`createEntryViews`	füllt die `libraryList` mit Einträgen (alphabetisch sortiert)
`startFlowersFragment`	zeigt die abgespeicherte *Flower* im *Flower*s-Menü an
`startDrawByHandFragment`	zeigt die abgespeicherte Kurve im `DrawByHandMenü` an
`showEditDialogue`	öffnet den *Dialog* zur Umbenennung des Dateinamens
`showDeletionDialog`	öffnet die Abfrage, ob der Eintrag tatsächlich gelöscht werden soll

4.6 Sonstiges (Arbeitspaket 6)

Die Dateistruktur einer Android-App erlaubt es bestimmte Werte global zu definieren, um an ihnen leichter Anpassungen vornehmen zu können.

Dateien	Inhalt
`colors.xml`	speichert alle in der App benutzten Farben als Hex-Werte
`styles.xml`	definiert Themes (derzeit DarkTheme) und das Erscheinungsbild von InputBoxes und TextBoxes
`strings.xml`	setzt Strings für alle Textelemente des UI

Diese Dateien sind im Ordner **res/values** zu finden.

Um die Implementierung häufig genutzter Konzepte zu vereinfachen, wurden im Paket **utility** bereits Klassen angelegt, die Blaupausen für die Erstellung von *Dialogen* und *Toasts* zu Verfügung stellen. Um sie nutzen zu können müssen lediglich die Methoden aus `FragmentExtensnFunct.kt` in ein Fragment importiert werden und können fortan von dort aufgerufen werden.

5 Datenmodell

Wird im `FlowersFragment` oder im `DrawByHandFragment` der save-Button betätigt öffnet sich ein *Dialog* der es ermöglicht einen Dateinamen einzugeben, unter dem die vom Nutzer generierten Daten abgespeichert werden sollen. Alle Daten werden im internen Speicher der App, im Verzeichnis **files/library**, abgelegt. Das `FlowersFragment` speichert die Parameter der `FlowersCalc`, separiert je durch ein Leerzeichen. Das `DrawByHandFragment` generiert aus der gezeichneten Kurve eine Liste von Koordinatenpaaren, wobei jedes Koordinatenpaar, durch ein Leerzeichen getrennt, in eine eigene Zeile geschrieben wird. Um später zu wissen, welches Format in einer Datei zu erwarten ist, enden die Dateinamen des `FlowersFragments` auf `.flw` und die des `DrawByHandFragments` auf `.tp`.
Das `LibraryFragment` holt sich durch den `FileHandler` eine Liste aller sich im designierten Verzeichnis befindlichen Dateinamen und erstellt daraus die `LibraryEntryViews`. Diese ermöglichen es Dateien zu öffnen, löschen, oder ihnen eine neuen Bezeichner zuzuweisen. Wird eine Datei geöffnet, liest das `LibraryFragment` den Dateiinhalt und erstellt daraus die entsprechenden Daten, die zur Anzeige an das Fragment übergeben werden.

6 Glossar

Fourieranalyse Bei der Fourieranalyse wird eine periodische Kurve in der Ebene in eine Summe aus Kreisbahnen zerlegt, deren Frequenzen Vielfache von $\frac{1}{T}$ sind, wobei T die Periodendauer ist. Dazu werden die sogenannten Fourierkoeffizienten berechnet - komplexe Zahlen deren Betrag dem Radius und deren Argument der Phasenverschiebung der zugehörigen Kreisbewegung entspricht. Genaueres wird in [1] beschrieben.

Proof-Of-Concept Proof-Of-Concept oder Proof-Of-Principle ist ein Begriff aus dem Projektmanagement. Er ist ein Beleg dafür, dass ein Vorhaben prinzipiell realisierbar ist. Die Kriterien dafür können in technischen oder betriebswirtschaftlichen Faktoren liegen. In der Regel ist mit dem Proof-of-Concept meist die Entwicklung eines Prototyps verbunden, der die benötigte Kernfunktionalität aufweist.

Poi Ein Poi besteht konzeptuell aus einer Kugel, an der sich eine Schnur mit einem Griff oder einer Schlaufe am anderen Ende befindet, von wo aus die Kugel im Kreis geschwungen werden kann. Durch zusätzliche Bewegungen der Hand kann eine Vielzahl an unterschiedlichen kreisähnlichen Bahnen erzeugt werden.

Flower Eine Flower ist eine spezielle parametrisierte Kurve, die durch eine Gleichung der Form

$$x(t) = cos(2\pi f_1 t) + r * cos(2\pi(f_2 t + \phi)), \quad y(t) = sin(2\pi f_1 t) + r * sin(2\pi(f_2 t + \phi))$$

dargestellt werden kann. Dabei sind r, f_1, f_2 und ϕ frei wählbare Parameter.

Ein *Poi*-Spieler kann solch eine Figur erzeugen indem er die Hand, welche den *Poi* hält, in einer zusätzlichen Kreisbahn bewegt. Dabei entstehen in Abhängigkeit von Rotationsrichtung und Geschwindigkeit unterschiedliche Muster. Sie werden Flowers genannt, da viele dieser Figuren schlaufenförmige Blütenblätter, sog. 'Petals', aufweisen.

API-Level Der API-Level gibt den Entwicklungsstand der eingebauten Funktionen (API, Application Programming Interface) an, die durch einen Entwickler, bspw. bei der Entwicklung von Apps, verwendet werden können. Die Angabe ist besonders dann entscheidend, wenn es um die Kompatibilität einer App mit einer auf einem Gerät installierten Android-Version geht. Verlangt die App einen höheren API-Level, als die Android-Version implementiert, bspw. weil spezifische Funktionen der Android-Version verwendet werden, so kann die App nicht installiert werden.

Kotlin Kotlin ist eine statische, typisierte Programmiersprache, welche sich in JavaScript-Quellcode transformieren und in Form von Bytecode für die JVM (Java Virtual Machine) übersetzen lassen lässt. Die wichtigsten Ziele bei der Entwicklung waren eine hohe Kompilier-Geschwindigkeit und möglichst wenig Code. Kotlin lässt sich außerdem zur Entwicklung von Android-Apps verwenden und wird dafür seit 2017 offiziell von Google unterstützt. Seit Mai 2019 ist Kotlin die von Google bevorzugte Programmiersprache für Android App-Entwicklung.

Dialog Ein Dialog in einer Android-App ist ein kleines Fenster, das den User über etwas informiert, die Bestätigung eines Vorgangs einfordert oder vom User eine Texteingabe, etwa einen Dateinamen, verlangt. Ein Dialog verschwindet erst nach Betätigung des Users.

Toast Ein Toast gibt eine einfache Rückmeldung zu einem Vorgang in einem kleinen Pop-up-Balken am unteren Bildschirmrand. Toasts verschwinden nach einer bestimmten Zeit automatisch.

Navigation Drawer Der Navigation Drawer stellt eine effiziente und dennoch recht simple Art der Wegeführung innerhalb der App dar. Offiziell von Google zur Verfügung gestellt bietet er eine Sidebar als Navigationsmöglichkeit, wie sie bereits aus anderen Apps - wie zum Beispiel der YouTube-App - bekannt ist.

Bézierkurve Die Bézierkurve ist eine parametrisch modellierte Kurve, die ein wichtiges Werkzeug bei der Beschreibung von Freiformkurven und -flächen darstellt.

Quellenverzeichnis

[1] *Berechnung der Handbewegung zu einer gegebenen Poibahn.* URL: `https://pcai042.informatik.uni-leipzig.de/~nw19a/Website/` (besucht am 28. 02. 2020).