VYTAUTAS MAGNUS UNIVERSITY
FACULTY OF INFORMATICS
DEPARTMENT OF APPLIED INFORMATICS

Quanchao Song

VIRTUAL HARP RUNNING ON MOBILE PLATFORMS

Term Paper

Informatics Systems study programme, state code 6121BX016
Study field Informatics

Supervisor Rita Valterytė
degree, name, surname signature, date

Defended prof. dr. Tomas Krilavičius
Dean of Faculty signature, date

Kaunas, 2022

CONTENT

LIST OF ABBREVIATIONS

RN	React Native
OC	Objective-C
2D	2-Dimensional
JS	JavaScript
IDE	Integrated Development Environment

ABSTRACT

Author	Quanchao Song
Title	Virtual Harp App
Supervisor	Rita Valterytė
Presented at	Vytautas Magnus University, Faculty of Informatics, Kaunas (2022.12.21)
Number of pages	15
Number of appendices	

Musical instruments have been playing an important role in people's daily life. There are many kinds of musical instruments, like guitar, piano, violin, etc. You can find many corresponding Apps related to these main-stream musical instruments on Google's Play Store or Apple's App Store. However, if you search "Harp", you can barely find any App, or even though you can find, it will not be that competitive to satisfy your need. Personally, I have a harp, and I have been wanting to make an App related to harp for long time. In this project, I use Flutter framework to develop a virtual harp which runs on mobile platforms like iOS and Android. So far, the virtual harp app has been completely done and it has a relatively good user experience.

1. INTRODUCTION

On Google's Play Store and Apple's App Store, you can find many beautiful and powerful Apps related to main-stream musical instruments like Piano, Guitar, Drums, etc. However, if you search "Harp" you will barely get any result, or even though you can, the current Apps about Harp is not that competitive. That is one of the reasons for me to want to solve this problem. Besides, When the requirement of the Term Paper was announced, I was once in a confusion since I had no idea about what I should do for my Term Paper. One time, when I was sitting and suddenly I glanced at my harp, something related to the Term Paper occurred to me: why not just choose the topic of making an App of a virtual harp? On the one hand, it would be sufficient for the requirement of the Term Paper, on the other hand, making an harp App would be of a lot of fun and romance.

That's where my idea came from, and that's what my **goal** is - to make an excellent Harp App.

To achieve this goal, i.e., to build a Harp App running on multi-platforms, we can separate this big task into a couple of small **tasks** and then accomplish them one by one:

- The first thing is to choose a proper framework (or language) and set up corresponding development environment .
- The second task should be to design the UI properly.
- The third task goes to implement the UI of the main page and basic functions like embedding harp sound.
- The fourth task is to add the plucking animation and effect.
- The fifth task is build the Add Notes page and implement the functions of adding/saving/playing notes.
- The final task is to integrate the aforementioned tasks and give a testing on a whole of these functions.

2. ANALYSIS OF TOOLS AND METHODS

As we are going to build an App running on both iOS and Android platforms, therefore it is essential to determine which framework we should choose to do it. After these years' boom of mobile development, now we have a couple of UI frameworks to choose, such as RN (React Native), Flutter, Qt, Xamarin, etc.

Of course, we can also use native codes for the development, in other words, we use OC (Objective-C) or Swift to code for the iOS App, and Java or Kotlin to code for the Android App, but correspondingly, the cost for 2-sides of codes will be hugely expensive, i.e., it will take a lot of time to develop, which will make the time for Term Paper insufficient. Thus, we have to choose one of the aforementioned multi-platform supported frameworks.

Among RN, Flutter, Qt and Xamarin, firstly, we can exclude Qt and Xamarin. For excluding Qt, it is because Qt is more used for building desktop based multi-platform software, like the Windows, MacOS, and Linux. It doesn't have much advantage when it goes to the mobile platform. Xamarin is excluded simply because it is a minority multi-platform UI framework, which means once we confront with some technological issues, it will be kind of hard to search solutions on the internet.

So now, we have to choose between Flutter and RN. Both UI frameworks have very large supporters, so we have to make a comparison between them. According to [1] and [2], we draw the Table 1., which illustrates the commons and difference of both UI frameworks.

Table 1. The Commons and Difference between Flutter and RN

	Popularity	**Documentation**	**Language**	**Hot Load**	**App Running Speed**	**3rd-Party Libraries**
RN	38%	Complete	JS	Support	High	Many
Flutter	42%	Complete	Dart	Support	Medium/Low	Many

From Table 1., we know that both frameworks have hot-reload functions, which enables the developers to test their codes more efficiently, and both frameworks have many 3rd-party libraries, which accelerate the development progress, and for both frameworks, we can search easily about their documents and find technological solutions on the internet.

However, for me, I personally prefer Dart language rather than JS, because Dart language is more convenient to build UI and animations. What's more, Flutter enables us to build very fast

Apps which can be even faster than native codes. This is a super advantage of Flutter. In comparison, RN often make relatively slow Apps, for which users sometimes complain that RN-built Apps have laggy responsive results. Since the Harp App requires high running efficiency due to the spring animations and effect, therefore it is better to choose Flutter UI framework.

By the way, according to [3], nowadays Flutter is a very trending UI framework nowadays, gradually gaining more attentions even than RN, attracting 500,000 developers using it monthly. It is easy and fast to use Flutter to develop beautifully designed UI.

Since I decide to choose Flutter UI framework, consequently, I will use Dart language as Dart is the official language for using Flutter framework. Dart language is an easy-to-learn language. Besides, it's available for all platforms. Similar to RN, dart maintains a kind of “state" mechanism to adapt to any change of the UI, as we are told in [4].

For the editor or IDE, there is no coercive answer. IntelliJ IDEA, VS Code, Emacs, VIM, or Atom are all fine. As long as the editor or IDE supports Flutter development mode, it is fine. Here, I personally prefer to use VS Code. According to [5], VS Code is easily customizable for the layout, the icons, fonts, etc. In the Extensions Market of VS Code, it is easy to find plug-ins to help you code for almost every programming language, not only these mainstream languages like C/C++, Python, JavaScript, PHP, etc, also including those infamous language, like Rust, Lisp, Erlang, Groovy, etc. Besides, VS Code supports git-related functions and a set of debugging functions. VSCode is the mostly used codes editor in my daily life. Therefore, I recommend VS Code.

3. HARP APPLICATION DESIGN AND IMPLEMENTATION

3.1 UI Design

This harp App contains 2 pages. The first page and also the main page, is the page for harp plucking. The second page is in charge of the music note sequence's creation, editing, and the command for starting to play.

To design the UI, I used the Sketch App on MacOS.

3.2 UI Implementation

3.2.1 Main Page

Firstly, the main page has such a layout structure as in Figure 1.

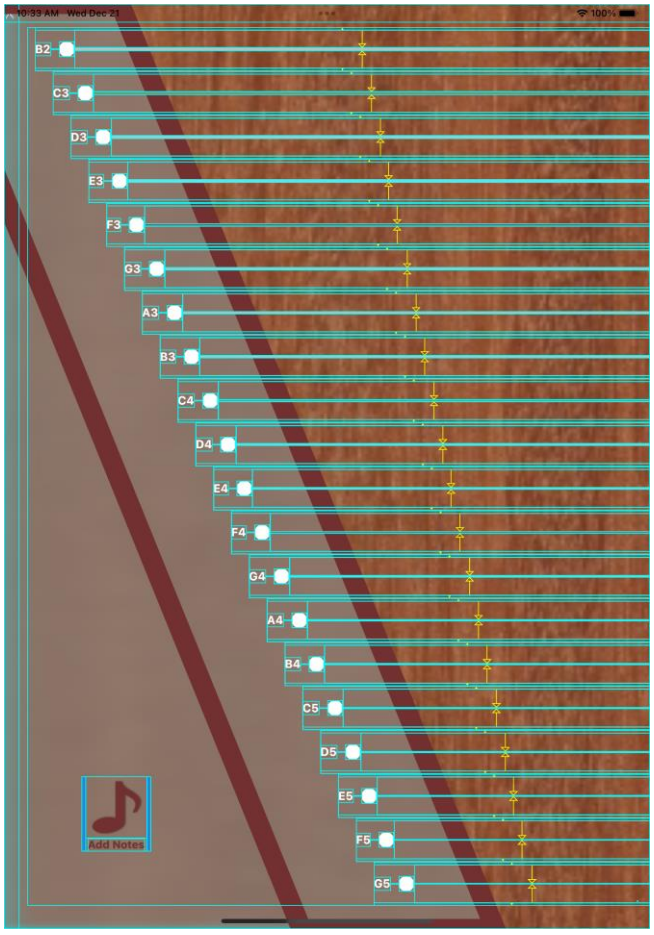


Figure 1. Layout Structure of Main Page UI

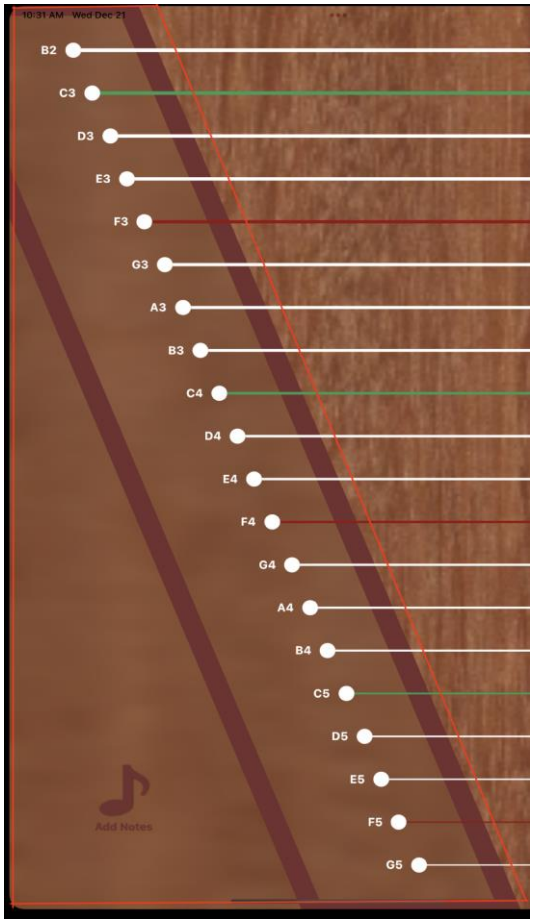


Figure 2. The Trapezoidal Part

And for the wrapped trapezoidal part as wrapped by the red path in Figure 2., two kinds of programming technologies are used.

The first programming technology is to use a clip path to wrap that trapezoidal part. To achieve this, I used the following codes.

```
class __MyPathClipper extends CustomClipper<Path> {

  @override
  Path getClip(Size size) {
    var path = Path();
    path.moveTo(150, 0);
    path.lineTo(size.width - 200, size.height);
    path.lineTo(0, size.height);
    path.lineTo(0, 0);
    path.close();
    return path;
  }

  @override
  bool shouldReclip(CustomClipper<Path> oldClipper) {
    return false;
  }
}
```

Codes for Making the Clip Path of Trapezoidal Part

The second programming technology is to blur the background image covered by the trapezoidal part. To achieve this, I combined three classes of Flutter: ClipRect, BackdropFilter, ImageFilter.blur, to make them work together to have the blur effect. The core codes are as the left one in the following two code fragments.

```
child: ClipRect(
  child: BackdropFilter(
    filter: ImageFilter.blur(
      sigmaX: this.blurX,
      sigmaY: this.blurY,
    ),
    child: childWidget ?? SizedBox(),
  ),
),
```

Codes for Blurring the Background

```
List<Widget> _generateHarpStringWidgets() {
  List<Widget> wgtList = [];

  for (int i = 0; i < 20; i++) {
    Widget w = _generateSingleHarpString(i);
    wgtList.add(w);
  }

  return wgtList;
}
```

Codes for Generating 20 Harp Strings

For the harp string layout part, I used a Column to be the organizer for the 20 harp strings. These 20 harp strings are generated by a simple for loop.

For the harp string widget, it is a Container filled with a Row organizer, which are filled with a Text and Container widgets.

3.2.2 Adding Notes Page

The core contents in this page are mainly two parts.

The first part is the notes input part, which allows users to input musical notes as in the Figure 3.

Figure 3. Input Notes Area in Adding Notes Page

The second part is to maintain a list, which contains the input musical notes by the user. In my case, I used `List<String> noteArr` to maintain users' input notes. With the `noteArr`, it is simple to display these musical notes on the top area of this page. What's more, with `noteArr`, we can easily save it to the local `sqlite` database, and pass the note sequence to the main page to let it play it sequentially.

3.3 Sound Embedding

A harp App should be able to perform harp sounds, absolutely. Thus, after the UI is implemented, it is time to employ harp sounds into the App. After some time's searching on the internet, I finally found an available resource on this website: [6].

After getting the .wav audio files, it is simple to load the corresponding local audio file and make it played at a certain string plucking. To load and play the local .wav file, I integrated a 3rd-party lib `SoundTool` to finish this job.

Apart from the way of loading and playing the local .wav files, there is another way to achieve the aim of playing harp sound. That way is to use an independent SoundFont (.sz2) file. Using a .sz2 file has more advantages than the previous way. For example, the .sz2 file is usually much smaller than the size of all those .wav files combined, and the .sz2 file can allow you to play a wider range of musical notes.

However, I didn't use the .sz2 file in the end, due to a technological problem. In Flutter, there are several 3rd-party libraries which handles the .sz2 files, but after I tried them, all of them didn't work fine when the harp .sz2 files were feeded to them. I tried these libs: flutter_sequencer, flutter_midi, and dart_melty_soundfont, all of which worked fine with a piano .sz2 file, but when they went to the harp .sz2 file, they didn't work well (i.e., they didn't play any sound). I also tested several more harp .sz2 files, all of them didn't work well. So finally, due to the limited time and energy, I abandoned this way. However, in the future, when I am going to have free time, I will try to figure out this problem.

Figure 4. A Spring in Real World

3.4 Plucking Animation

We know that, on a real harp, when we pluck a certain string, that string will have a plucking behavior. Therefore, we have to simulate this behavior by using some animation.

To implement the animation, I used the Spring Animation, which looks like a real spring (as shown in Figure 4.) with the physics of a spring.

In Flutter, we can use SpringSimulation class along with SpringDescription to implement such kind of animation with the physics of a spring.

Firstly, we need to init a SpringDescription instance, passing it with several parameters.

```
const spring = SpringDescription(
  mass: 2,
  stiffness: 2000,
  damping: 4,
);
```

Codes to Init A SpringDescription Instance

The following codes fragment represents that we want the spring physics of mass = 2, stiffness = 2000 and damping = 4. These parameters determine how the spring animation will be performed, for example, softly, hardly, slowly, or fiercely, etc. And of course, they have been properly tried and tested, and in the end, I chose these values for them to get a relatively good result of the harp spring plucking animation.

```
Container c = Container(
  height: 50,
  child: Align(
    child: s,
    widthFactor: 1,
    heightFactor: 1,
    alignment:
        Alignment(0, (index == _pluckingIndex ? _animController.value : 0)),
  ),
);
```

Codes for Defining the Animated Property Within the Harp Spring Widget

```
final simulation = SpringSimulation(spring, 0.05, 0, 300);
```

Codes to Init A SpringSimulation Instance.

Note that, for the string widget, it is a Container, whose alignment's y value is bound to the animation system.

Afterwards, we can initiate a SpringSimulation. instance as the following.

We can see that, the starting value for the harp spring widget's alignment's y value is 0.05, and the ending value is 0, which means after some spring physics behavior, the harp spring widget will finally go to the original position. And we assign the initial velocity at a relatively high value: 300, to make the animation more correspondent to the real harp spring's plucking behavior.

And finally, we simply run the animation.

```
_animController =
    AnimationController(vsync: this, lowerBound: -0.05, upperBound: 0.05);
_animController.addListener(() {
  setState(() {});
});

_animController.animateWith(simulation);
```

Codes to Run the Animation

3.5 Plucking Effect

It is a good user experience if the users can see which harp string they have plucked, therefore, we can add some plucking effect on the harp string that was plucked just now.

Originally, I tried to implement this function by using some particle effect. Flutter has a community-supported 2D game engine, called Flame. In Flame, there are some particle effect classes, like MovingParticle, ImageParticle, CirclePartilce and SpriteParticle, etc. I did use it in my project, however, I canceled it due to that this particle system required heavy computing resource, in other words, it costed a lot of memory and CPU resource, and made the computer run into a high-temperature status within 1 minute. For this App, the plucking effect should be some icing on the cake, so we should not let it consume so much CPU/memory resource.

Luckily, we have a light-weighted way to implement the plucking effect, which is to use the paint mechanism of Flutter. paint allows us to draw a line, path, text, image, or any shape onto the canvas , and it usually takes only very little computing resource. Besides, we can use a AnimationController to animate the content onto the canvas.

Thus, though some thinking and comparison, I decided to make a blinking effect as in Figure 5., on the plucked harp string.

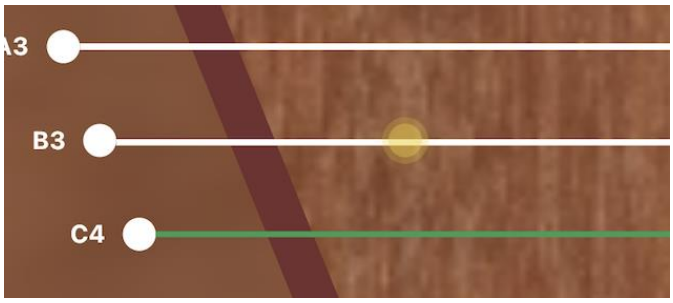


Figure 5. Plucking Effect on the Plucked Harp String

The core philosophy for this effect is to create a couple of filled circles with progressively increasing radius, and then animate their radius opacity at the same step and time. The core codes for this procedure can be like the following codes.

```
@override
void paint(Canvas canvas, Size size) {
  final Rect rect = Rect.fromLTRB(0.0, 0.0, size.width, size.height);
  for (int wave = 0; wave <= wavesCount; wave++) {
    circle(canvas, rect, minRadius, wave, _animation!.value, wavesCount);
  }
}

void circle(Canvas canvas, Rect rect, double? minRadius, int wave,
    double value, int? length) {
  Color _color;
  double r;
  if (wave != 0) {
    double opacity = (1 - ((wave - 1) / length!) - value).clamp(0.0, 1.0);
    _color = color.withOpacity(opacity);

    r = minRadius! * (1 + ((wave * value))) * value;
    final Paint paint = Paint()..color = _color;
    canvas.drawCircle(rect.center, r, paint);
  }
}
```

Codes for Painting Blinking Effect on The Canvas

3.6 Playing Music Note Sequence

The last procedure is to make the App able to play users' input musical notes automatically.

This function doesn't require something new to be created. Instead, it is more a function that combines the previous functions together.

The main methodology is that, firstly we obtain a list `noteArr` containing music notes from `AddNotesPage`, then we perform a `for` loop to iterate this `noteArr` to get each music note, and

transformed the music note to the form that we can use it for directly playing the corresponding .wav audio file. The core codes for this procedure is as following codes.

```
void _playSequenceNotes(List<String> noteArr) {
  for (int i = 0; i < noteArr.length; i++) {
    String noteStr = noteArr[i];
    int transformedNoteIndex = _transformNoteStrToIndex(noteStr);
    Future.delayed(Duration(milliseconds: 1000 * i), () {
      if (transformedNoteIndex >= 0) {
        _stringClicked(transformedNoteIndex);
      }
    }); // Future.delayed
  }
}
```

Codes for Playing The Sequential Musical Notes

Until now, the App plays these musical notes sequentially, along with plucking animation and effect.

4. RESULTS AND CONCLUSIONS

4.1 Results

After the aforementioned procedures, our project finally comes to an end, a good end. Now, we can perform plucking actions on the harp strings with fingers, and we can see corresponding spring animations and blinking effect. Besides, we can input a series of musical notes on the AddNotesPage, and after we press Play button, the App will fluently play these musical notes.

Here is a link for the recorded video of playing this harp App: [7].

4.2 Conclusions

After many researches and experiments on this project, I gained a deeper knowledge of the Flutter framework. Though a lot of time spending in the development of this App, I finally achieved everything that a Harp App should have. Of course, if I want to make this App perfect, there are still plenty of detailed things to do, for example, to improve the strings' plucking animation and effect, to make the musical notes be performed in a more natural way, to improve the UI of the main page, etc.

However, due to the limited time and energy, it is already a good job to finish this App until here. If I have more free time in the future, I will invest more time and energy to make this App a better one and publish it in the Play Store or App Store. In a word, my initial goals related to this harp App have been achieved.

5. REFERENCES

[1] Mohit Joshi, Flutter vs React Native: A Comparison, 2022. Available: https://www.browserstack.com/guide/flutter-vs-react-native. [Accessed 21 December 2022].

[2] Isaac Lyman, Flutter vs. React Native: Which is the right cross-platform framework for you?, 2022. Available: https://stackoverflow.blog/2022/10/31/comparing-frameworks-for-cross-platform-apps-flutter-vs-react-native/. [Accessed 21 December 2022].

[3] Evgeniy Fetisov and Anastasiya Talochka, Flutter vs. React Native in 2023: Which is Best for Your App?, 2022. Available: https://jaydevs.com/flutter-vs-react-native-which-is-best-for-your-app/. [Accessed 21 December 2022].

[4] Dart overview, 2022. Available: https://dart.dev/overview. [Accessed 21 December 2022].

[5] Visual Studio Code Official Website's Homepage, 2022. Available: https://code.visualstudio.com/. [Accessed 21 December 2022].

[6] ETHEREALWINDS HARP II: COMMUNITY EDITION, 2022. Available: https://vis.versilstudios.com/etherealwinds-harp.html. [Accessed 21 December 2022].

[7] Kitty Hello, Harp App Demonstration, 2022. Available: https://youtu.be/6iI5VLVhYcc. [Accessed 21 December 2022].

6. APPENDIXES

6.1 Topic formalization document (term paper)

Faculty of Informatics, Vytautas Magnus University

Bachelor studies, study programme “Informatics Systems”

_____Quanchao Song_____

(name, surname)

TERM PAPER (task description)

TITLE: Virtual harp running on mobile platforms

GOAL: to create an App of a virtual harp with around 20 strings, running on iPhone/iPad/Android Devices. This virtual harp allows users to press their fingers on the virtual strings and then perform rhythms.

TASKS:

1. Preparation of the Flutter development environment
2. Draw a draft sketch of UI
3. Codes for UI
4. Codes for functions of the strings
5. Trivial stuff such as animations, settings
6. Final testing & bug fixing

EXPECTED RESULTS:

- for midterm presentation:

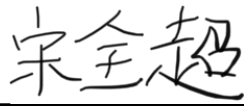
  Finish UI and basic functions of those virtual strings, i.e., users can press the strings and there will be corresponding sound

- for final presentation:

  An entire virtual harp App which runs on iOS & Android devices.

SCHEDULE (expand/reduce table upon need)

Week (or up to the date)	Activities	Results
1	1. Preparation of the Flutter development environment 2. Draw a draft sketch of UI	Finished UI draft sketch
2	1. Codes for UI	Completed UI
3	1. Codes for functions of the strings	Realized functions
4	1. Trivial stuff such as animations, settings	
5	1. Final testing & bug fixing	Finished App

Student Quanchao Song 

(name, surname, signature)

Supervisor Rita Valterytė

(title, name, surname, signature)

Study Programme Coordinator Daiva Vitkutė-Adžgauskienė

(title, name, surname, signature)

6.2 Sources

Harp App's Source Codes: https://github.com/AlbusSong/harp_app

6.3 Animations

String Plucking Animation: https://youtu.be/SajTDCTt5uA

String Plucking Effect: https://youtu.be/cHcGfJHSEQU