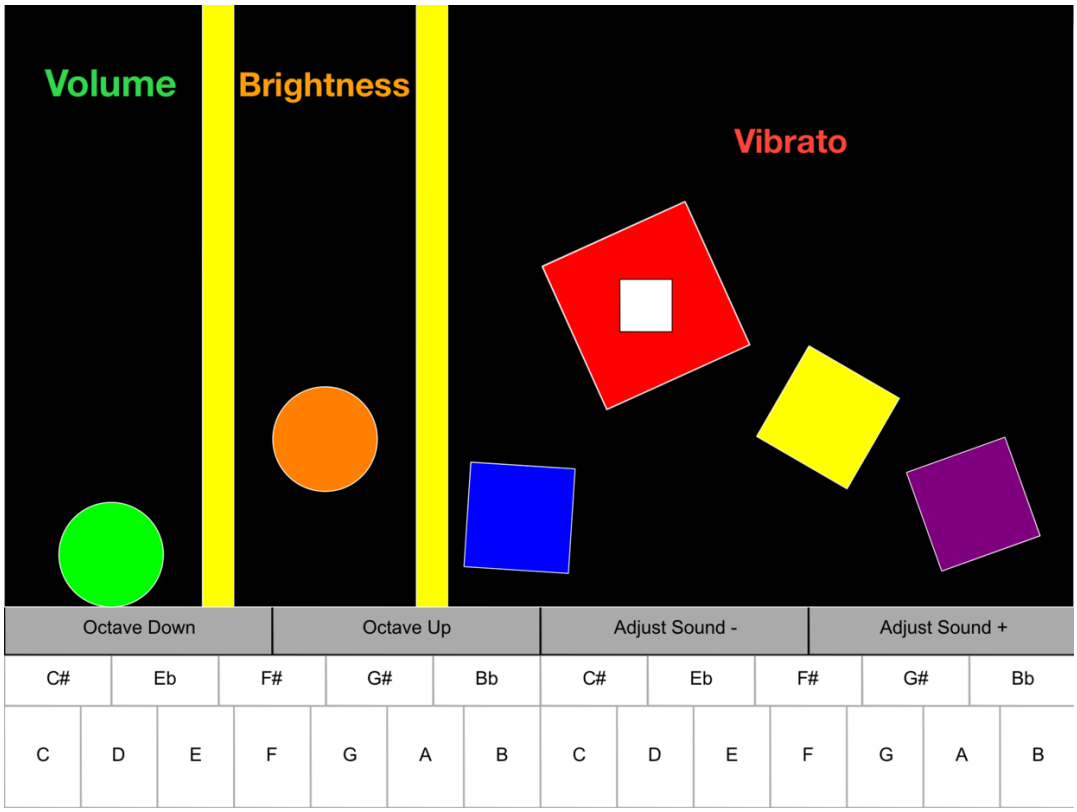
PhysiBoard



iPad OS Audio App (for 9th Gen iPad)

by
Jonathan Cooke

19th January 2023

For iOS Audio Programming ELE00051H

1. App Features

PhysiBoard is a simple single view app for the 9th Generation iPad which uses sandbox physics to control expression parameters for a touch-friendly keyboard-style instrument. The layout resembles that of a traditional keyboard so that experienced musicians can pick up the instrument quickly. However, the intended audience is non-musicians with a knowledge equivalent to the core national music curriculum for secondary age students in the United Kingdom (Key Stage 3) [1]. The app is a toy rather than a serious instrument, which can introduce simple synthesis concepts visually in an interactive way.

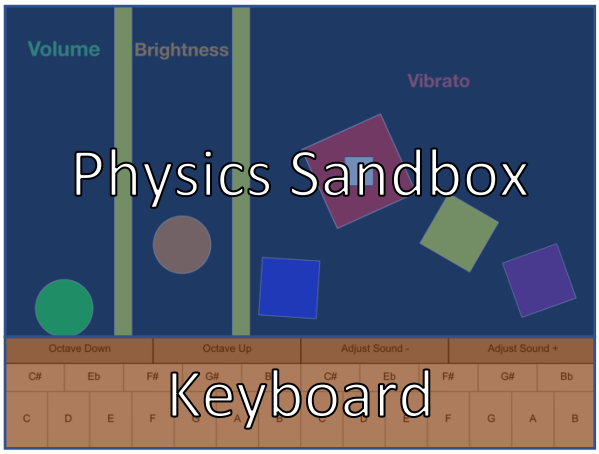


Figure 1: App Screen Layout

The Keyboard

The app screen is split into two main sections, which are laid out for the landscape view on a 9th Generation iPad. Illustrated in Figure 1, the smaller lower section comprises the 'keyboard', which has note keys and a small number of control buttons. There are two octaves of western chromatic notes, split between upper and lower sections: larger keys for notes within the C major scale, and the smaller upper keys comprising of the sharps and flats. The notes are laid out geometrically as opposed to using a traditional keyboard layout. The keyboard is capable of handling multiple simultaneous touches. Seated directly above the keyboard keys are additional buttons allowing the user to shift the keyboard octave or adjust the sound.

The Physics Sandbox

The larger section of the screen comprises of a physics sandbox with individual physics objects which the user can control through touch. The sandbox has three main labelled sections: 'Volume', 'Brightness' and 'Vibrato'. The names of these sections were chosen to use commonly understood terms and avoid direct reference to the synthesis parameters they control. These names refer to control parameters for a simple monophonic FM synthesiser, which is also controlled by the keyboard.

Within the sandbox three types of interactive physics objects are found: circles which control synthesis parameters, a spinning square which also controls synthesis parameters, and a set of free-moving coloured squares which do not directly control synthesis. The two circles each have their vertical positions tied to synthesis parameters controlling 'Volume' and 'Brightness'. The spinning square is mapped to two synthesis parameters by its angle of rotation and the horizontal position which together control 'Vibrato'. All physics objects will move around and interact with one another when the user is not controlling them. All physics objects are bound by a 'gravity' which pulls them towards the lower edge of the screen.

Synthesis Control

When a keyboard key is touched the green 'Volume' circle jumps upwards. As its height increases, so does the volume of the note being played. When the spinning square is dragged or hit by another object it will spin in place around the inner white square, adding a pitch modulation to the synthesiser in the manner of a Low Frequency Oscillator (LFO).

The three coloured squares which do not directly control synthesis parameters have been added for the user

to play around with and place as they like, changing how other objects behave. Each of the three have been given different simulated mass values and so will transfer different magnitudes of momentum to other objects during collisions.

The display can be interacted with using multitouch interactions, with the user holding or controlling multiple objects, keys or buttons at the same time. This allows the user to play the keyboard whilst changing its expression using the physics sandbox.

2. Design

PhysiBoard has been written using the Xcode application (13.4.1), for iPad OS 15.7 using the SpriteKit framework and the graphics capabilities therein for display elements. The SpriteKit scene created is hosted within a UIKit view. In addition, the AudioKit (5.2.2) framework has been used to generate all audio within the app. Most of the design is constructed within the Swift code, although section labels have been placed using the Scene editor.

The structure of code files within the app (shown in Figure 2) has been derived from the 'iOS Game' app template provided by Apple within Xcode, with SpriteKit as the chosen 'Game Technology'. The SpriteKit scene is hosted within the GameViewController class, using UIKit. Visual elements of the app and user-interactions are handled by the SynthLayoutScene class, and all audio is controlled from within the AudioController class. The SynthLayoutScene class also uses a SpriteKit Scene file also named SynthLayoutScene.

During development an alternative method of presenting the app was explored which would have hosted multiple SpriteKit scenes using the newer SwiftUI framework. However, no way of multi-touch capable SpriteKit scenes within the framework was found, hence the SpriteKit scene is rendered using the older UIKit framework.

LFOs Through Physics

To test user-interaction with physics objects as a method of controlling audio parameters, a number of prototype apps were written. These prototype apps tested free-moving physics bodies, as well as bodies attached to spring, pin and sliding joint physics joints (all from the physics capabilities of the SpriteKit framework). Early ideas included using springs or pendula to represent Low Frequency Oscillators (LFOs). However, due to a combination of the complexity of programming touch interactions, and the limitations of SpriteKit physics, simpler and less ideal implementations were used. In the final app, as an example, a triangle-wave shaped LFO is generated by the spinning square to control vibrato. A smoother sounding vibrato might have been achieved by using a sine-wave LFO, and it was originally planned that this would be achieved using springs or pendula, which oscillate sinusoidally in nature.

Classes and Functions

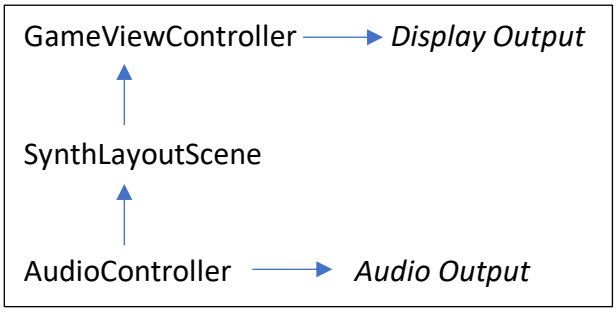


Figure 2: Design Hierarchy

Within the SynthLayoutScene class are a small number of custom subclasses, functions for layout, adding objects to the screen and functions defining touch interactions with objects. In addition there are also overridden SKScene functions including didMove(), the four touches functions and update(). Most objects when added, for example keyboard keys and physics objects, are added to arrays as they are added to the SpriteKit scene

so that they can be accessed and updated after initialisation. Functions are grouped according to whether they pertain to the keyboard or physics sections of the screen.

The monophonic synthesiser audio within the app is generated using the FMOscillator object within AudioKit which uses a single sinewave to modulate a carrier sinewave. This takes place within the AudioController class which can accept control values when called within another class. Individual functions have been written for each control parameter to be changed by the user's interactions, e.g. updateFMAmplitude(). In the current app build an instance of AudioController is created from within SynthLayoutScene and used to pass physics object position parameters to the audio parameter functions whenever the update() function is called.

3. Future Development

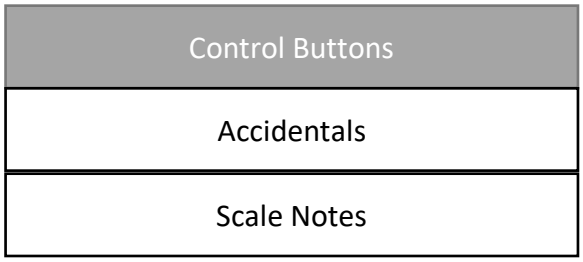


Figure 3: Keyboard Row Layout

PhysiBoard has been designed with several future developments in mind, many of which were not added to the first release due to time constraints. Originally intended to follow a more modular design, in future developments the keyboard should be isolated in a separate class so that the physics sandbox can be more easily swapped out for other arrangements of physics objects. Additionally, using the current app design it should be possible to add more synthesiser voices with minimal restructuring of code.

The keyboard keys and buttons have been designed to be deliberately adaptable such that the number of each in a row is adaptable. Figure 3 shows the layout scheme of keys and buttons on the keyboard. Rather than starting on C with the natural notes (C Major) in the scale notes row, the scale notes could hold the notes of any western scale, with the remaining notes placed in the Accidentals row. Though in the current build there are two octaves of eight natural notes each in the scale notes row, this could be adapted to hold a five note pentatonic scale, with the remaining 12-tone notes held in the Accidentals row. The rows could also be adapted to fit microtonal tunings, including those making use of more than twelve notes.

The keyboard keys and buttons are already designed to adapt the number of requested keys to the size of the display, however additional functions would need to be written to automatically map different scales onto the chosen numbers of notes. The current mapping of notes to frequencies relies on the use of the AudioKit function midiNoteToFrequency() to convert MIDI note values to frequencies. To use more complex tuning systems than western twelve-tone, it would be required to migrate this to a more elaborate framework. The AudioKit framework Tonic may be capable of such calculations, but a successful implementation of this was not found during development.

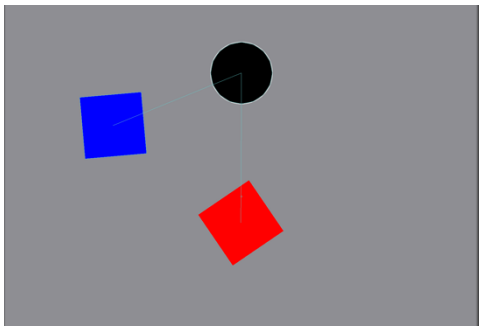


Figure 4: Spring Joint Test

In the future additional physics sandboxes could be added to the app in the form of free updates. These new sandboxes could use different arrangements of physics objects to control

either the same or different synthesis parameters in order to give users more ideas to play with. One concept which was prototyped but not added to the app due to time constraints was the model of two dynamic squares attached to a static central circle by springs as shown in Figure 4. In this sandbox the squares would rotate around the circle like clock hands and collide if they met.

4. Marketing Strategy

Monetisation

The monetisation strategy of PhysiBoard must be balanced against the level of barrier to access or inconvenience caused by the presence of monetisation mechanisms. For an app intended to be used as an instrument it would be inappropriate to include intrusive advertisements which in any way impeded the ability of the user to use the app at any time. For example, the use of pop-up video adverts with audio would disturb the sound coming from the app and interfere with the ability of the user to use the app as a performance instrument. The loading or changing of banner adverts could also lead to temporarily worse performance of the app and cause artefacts in the output sound.

Although PhysiBoard is intended more as a toy than a serious performance instrument, some users may choose to use the app for performance. In addition, some customers of the app may not become the end users. For example, a school music teacher may choose to buy the app as a teaching aid. As musicians, such customers may be dissuaded from using the app if it is not appropriate to musicians. Therefore, a monetisation strategy which doesn't use advertisements at all would be preferable. As the current release version of the app is very simple, but is intended to be developed to include more features in the future, it is recommended to begin distribution of the app for free, with in-app purchases for extra features being added later.

Using the free with in-app purchases monetisation model, free from advertisements the app should be able to grow its userbase despite its relative simplicity. As the number of active users of the app increases, the chances of users choosing to make small in app purchases for additional features will also increase. Examples of features which could be offered as low cost in-app purchases (of under 1GBP each) include: different scales, a microtonal capable keyboard, additional physics sandboxes and additional synthesiser voices.

If after some time on the App Store it is found that users are downloading the free app but are not willing to pay for in app purchases an alternative approach could be taken to add non-intrusive banner advertisements to the top edge of the screen above the physics sandbox section. These adverts would not significantly disturb the intended audience, but could be removable by more serious musicians by paying a small in-app purchase of 1GBP or under).

Standing Out from Competitors

The market for synthesiser apps on the Apple App Store is highly competitive and saturated with high quality offerings. However, many of these are designed specifically as performance or writing tools, or as teaching aids, often with comparatively high purchase prices. PhysiBoard will stand out from competitors using its novel approach to synthesiser control, appealing to both non-musicians and musicians due to its playful interface. Offering the app as a free download will also help to differentiate it from apps marketed as serious instruments and remove barriers to users trying it out.

References

[1] Department for Education (2013). *Music Programmes of Study: key stage 3.* gov.uk [Online]. Available at: https://assets.publishing.service.gov.uk/government/uploads/system/uploads/attachment_data/file/239088/SECONDARY_national_curriculum_-_Music.pdf [Accessed: 19 Jan. 2023].

Bibliography

The following sources were referred to during the development of PhysiBoard in addition to the Apple Developer Documentation:

- Apple Inc. (2022). *The Swift Programming Language.* docs.swift.org. [Online]. Available at: https://docs.swift.org/swift-book/ [Accessed: 19 Jan. 2023].
- R. Wenderlich and B. Eller (2018, Apr. 11). *SpriteKit Tutorial for Beginners.* Kodeco. [Online]. Available at: https://www.kodeco.com/71-spritekit-tutorial-for-beginners [Accessed: 19 Jan. 2023].
- L. Joswiak (2018, Mar. 23). *Drag and Drop Sprites in SpriteKit*. Lukas Joswiak [Online]. Available at: https://lukasjoswiak.com/drag-and-drop-sprites-in-spritekit/ [Accessed: 19 Jan. 2023].
- M. Faarkrog and C. Begbie (2017, Dec. 5). *Introduction to the SpriteKit Scene Editor*. Kodeco. [Online]. Available at: https://www.kodeco.com/620-introduction-to-the-spritekit-scene-editor [Accessed: 19 Jan. 2023].