Tone Deaf-I: A Practical Application for Amusia and a Reflection on the App-Development Process

Trina Coleen Dayley
Utah State University

Follow this and additional works at: https://digitalcommons.usu.edu/gradreports

Part of the Teacher Education and Professional Development Commons

Recommended Citation
https://digitalcommons.usu.edu/gradreports/782
TONE DEAF-I
A Reflection on the App-Development Process

Trina Dayley
TEAL 6960: Master’s Creative Project
Utah State University
Tone Deaf-I:
A Reflection on the App-Development Process

When I began this project, I knew I wanted to develop something that could be used to help a small population of people—people who were deemed congenitally amusic, or “tone deaf.” I had read articles on the subject, collected information which I later compiled into a literature review, took suggestions from sage teachers, and formed guidelines and a work plan to direct my efforts in the app-development process. Having no previous experience in scripting, save the minimal html work I did and continue to do as an indie author, I began to prepare myself by developing small games via YouTube tutorial videos at the start of summer of 2015.

I knew it was going to be a difficult process when I began—the idea that I would have to learn something that was completely foreign to me was frightening. During that summer, I was introduced to terms like Coroutines and IEnumerators, while, for, and foreach loops, arrays, Vector3, Transform, and tags. I came across terms that had become antiquated, like Debug.Log (currently replaced by “print”) and Application.LoadLevel (now called with “SceneManager.LoadScene”). Although I still had much to learn before I could confidently start my creative project, I had gone far enough in my understanding by fall of that year that I laughed out loud when one online Unity forum contributor commented, “Lists are great. ArrayLists, on the other hand, make kittens cry.”

In my study, I have gained an appreciation and respect for computer languages because, as I discovered, like other languages across the world, computers have rules of speech. Like a delicate dinner conversation, computers need to be told in a specific way what you would like and how you would like it done. Leaving out a “;” or a “}” could have disastrous results. Consider the following employment of the rules of speech: If one would like the screen to display a calculated number, first declare a type “int” or “float” variable along with a name and an initial value (if any), then construct a method that will utilize that variable, and finally a reference, or call, of that method in a function. As one participates in the conversation of scripting, s/he must be sure to give mind to differences in capitalization. Both of the scripting languages I studied (C# and JavaScript) are case-sensitive, which means that there is a
significant difference between gameObject and GameObject. Computer languages also come with their own variation in dialect, which is dependent on the program being used by the developer (I am using C# on Unity 5.3.2.f1).

Figure 1: Script Example

In this paper, I will reflect on the guidelines I had previously formed and the efforts made to bring the final result to fruition. Many of the successes I’ve had in the recent months of my application development I credit to the patience of my ITLS 5245 adjunct professor, Marc Midura. I know I would not have been able to do what I have done if this particular course had not been made available this semester. It has aided my progress in both correcting assumptions
from my early YouTube training and widening my understanding of the tasks scripting can achieve.

**Implementation**

What follows is a depiction of the thoughts and actions used to design and format the exercises as seen in the application as well as other issues that arose in the app-development process.

**Baseline Finder**

Once I had done some independent study on how to use the program and coding language, and had begun taking the class with Mr. Midura, I realized that I could not simply start off by coding the exercises themselves. It became apparent as I was developing the first levels of the application that I would have to form a scene wherein a user could calibrate his or her vocal range. By achieving this, the app could then be personalized to the comfortable range of each user. But this was not an easy task. I had to create a scene that would produce a meaningful number. That number would then be used within all subsequent levels to form what
I called a “baseline” note. But how could I get this number? What kind of operations and methods and variables would I need? How would I be able to easily share this number between scripts?

I began to put together the Baseline Finder scene. It had all the basics—a staff, a moveable character, and a plugin that enabled movement via pitch received through the microphone. The only thing it lacked was the ability to produce a meaningful number. For this, I searched the Internet, knowing that I wanted to create a script that would find not the average (that was easy to find), but the mode. In the few solutions I found, there was an inconsistency and I couldn’t determine whether I should use “System.Linq” (which was negatively reviewed for its CPU usage) or another method that used generic lists and more lines of, what appeared to me at that time, confusing code. I had little clear understanding regarding lists and arrays so I was grateful that Marc Midura was able to help me navigate through this new territory after class one night. When we were able to produce a resultant number based on the mode, we discovered that Marc produced a number that was consistent with the mode, but I did not. At first I wondered if it was a difference in range that affected the script in someway. After much searching, we discovered that the last note that was sung to find the baseline was not being calculated in the mode formula. Because Mark had a more fluctuating approach in his pitch to test the mode and I had a stable note throughout, we were producing differing results—since my last note and my first note and every pitch in-between were the same pitch, the formula wasn’t recording my pitch at all. After realizing the cause of the error, we were able to extend the array to include the last recorded pitch.

Now that the Baseline Finder scene produced a meaningful number (mode), I created a shared script and implemented that number in a separate script that read and interpreted pitches collected from the microphone into midi notes. By comparing read midi notes to the baseline mode, I essentially created, what we call in music, a “moveable do” wherein the baseline note became the new middle C.

In determining the length of iterations the app should detect before producing the mode, or baseline note, I decided to leave that length of time up to the user by placing that ability in a button component. Also, nearly every scene has a way to get back to reconfigure the baseline
note. This way, if the user feels strained when trying complete an exercise, s/he has the option to recalibrate the baseline note.

![Vertical Slider scene](image)

**Vertical Slider**

After developing the Baseline Finder, I began work on the Vertical Slider. This first exercise produced the foundation for the rest of the application. It was important to get it to respond as accurately as I could to the pitchDetector, a plugin that interprets incoming pitch from the microphone and returns a midi note value. At first, this plugin was presented me with a significant issue—I could not get the pitchDetector to get data from the microphone. After many hours of trial and error, and with great assistance from Marc Midura, we were able to determine that the volume was muted within the script, preventing the plugin from interpreting the incoming pitches. However, when we unmuted the volume, we heard our voices regurgitated and distorted in high pitches through the computer’s speaker. To compensate, we lowered the volume enough to respond, but not enough to create a loop of sound that reacted negatively to itself. This limitation means that a user must have her/his volume set to a medium to low level when using the application.

After I was able to get the pitchDetector to respond, I began working on targets, how I wanted to script their behavior, and how I wanted to use the amount of targets collected to determine when the player could move on to the next scene in the level.
Figure 4: This is an example of a script in C# that I wrote for the Vertical Slider exercise. In it, I describe the behavior I want the targets to have (i.e., self-destruct after 3 seconds if they aren’t hit by the user, spawn at specific locations, rotate for a spinning effect).
Speech Inflection

Thankfully, the app-development process started to pick up speed after the Vertical Slider and Baseline Finder scenes were established. For this Speech Inflection exercise, I knew I needed to reset the position of the character after each word/target was destroyed in order to situate the user to attempt to collect the next target. I also needed to enable the camera to track the character’s movement—an ability that wasn’t necessary with both the Vertical Slider and Baseline Finder scenes.

I discovered a few issues that were quickly identified and solved. For example, I noticed that the character kept unintentionally hitting the next few targets before returning to the ground. I solved this by adding a Coroutine that allowed the character to wait before moving to the next position on the x-axis. Another example involved two characters appearing in the same scene after progressing to another exercise. Although the error was entertaining, I removed the issue by deleting a DontDestroyOnLoad command I had previously written in the script.
Interval Training

Having completed the Speech Inflection exercise, I was able to move on to Interval Training. The most notable difference between this exercise and the others was that the character was in constant forward motion. To write a script that specified this behavior wasn’t difficult, but I soon realized that it became cumbersome to constantly see my character drop to the ground after reaching a target. Although the same tactic was used in the Speech Inflection exercise, I believe that the act of singing made it more tiresome. Furthermore, I had concluded that it would be most beneficial to allow the character to stay where the user last put it so that the user can then use it’s y position as a reference for reaching the next target. This was accomplished by making the rigidbody component of the character kinematic when the pitchDetector read close to 0 decibels, and effected by gravity otherwise. Liking the final result, I returned to the Vertical Slider and employed the method there as well.

Scope and Sequence

Great thought went into arranging the various levels and exercises into a logical progression. For input, I asked Dr. Sylvia Munsen to provide me with a typical progression used in teaching children solfege (e.g., do, re, mi, etc.). With her list as a guide, I structured each of the exercises, picking songs and phrases that were appropriate to the task, and grouped them accordingly. As a result, there are ten levels with three variations of each level across the varying exercises.
The progression of the exercises is divided such that the simplest intervals are initiated in the early exercises (e.g., unison, minor third) and the more complex interval combinations are presented in the later levels (e.g., half-steps, perfect fourth jumps). For each level, there is an interval that is explored across all 3 exercises. For example, Level 2 practices a minor third. This is initiated in the Vertical Slider exercise and then reinforced throughout both the Speech Inflection and Interval Training exercises. To further establish the theme that connects these exercises together, I have created a context theme, which, in the case of Level 2, is in regards to the term *rain*. In the next version of the application, I plan to add a text that states the theme at the top of each exercise in hopes that by helping the user identify the theme of a level, s/he will understand the purposed objective.

Below is a breakdown of the levels and exercises.

<table>
<thead>
<tr>
<th>Scope &amp; Sequence</th>
<th>Vertical Slider</th>
<th>Speech Inflection</th>
<th>Interval Training</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Level 1</strong></td>
<td>mi-mi</td>
<td>“I was born at the age of twelve on an MGM lot.” (Judy Garland)</td>
<td>Irving Berlin’s “Mr. Monotony” (segment)</td>
</tr>
<tr>
<td><strong>Level 2</strong></td>
<td>sol-mi</td>
<td>“The way I see it, if you want the rainbow, you gotta put up with the rain.” (Dolly Parton)</td>
<td>“Rain, Rain” (without la)</td>
</tr>
<tr>
<td><strong>Level 3</strong></td>
<td>la-sol-mi</td>
<td>“Do not go gentle into that good night. Rage, rage against the dying of the light.” (Dylan Thomas)</td>
<td>“Silent Night” (segment)</td>
</tr>
<tr>
<td><strong>Level 4</strong></td>
<td>mi-re-do</td>
<td>“I don’t know where I’m going from here but I promise it won’t be boring.” (David Bowie)</td>
<td>David Bowie’s “Heroes” (segment)</td>
</tr>
<tr>
<td><strong>Level 5</strong></td>
<td>sol-mi-do</td>
<td>“Life is what happens to you while you're busy making other plans.” (Adage, quoted by John Lennon)</td>
<td>Beatles’ “Ob-La-Di, Ob-La-Da” (segment)</td>
</tr>
<tr>
<td><strong>Level 6</strong></td>
<td>sol-mi-do</td>
<td>“I am sure my music has a taste of codfish in it.” (Edvard Grieg)</td>
<td>Morning Mood by Edvard Grieg (segment)</td>
</tr>
<tr>
<td><strong>Level 7</strong></td>
<td>sol,-do</td>
<td>“I always felt like a marriage works best on, say, a farm…” (Ethan Hawke)</td>
<td>Mendelssohn’s “Wedding March” and folksong “Old MacDonald” (segments)</td>
</tr>
<tr>
<td><strong>Level 8</strong></td>
<td>sol-do’-re-mi</td>
<td>“One should always have something sensational to read in the train.” (Oscar Wilde)</td>
<td>“I've Been Working on the Railroad” (segment)</td>
</tr>
</tbody>
</table>
Comparison

When I wrote my project proposal for this application, I included a list of seven guidelines that were derived from a literature review, which was also included in the proposal. The following is a comparison between the proposed guidelines and the exercises found within the application.

Proposed Guidelines and Actualized Application

1. Software applications will not have background music, or a backtrack.
   a. **Reason(s):** Music-listening experiences often result in noise to an amusic; scholars have found that a singer may perform well until supplemented by an instrument, which can subsequently cause her/him to lose control of the melody (Stewart, 2006, pp. 904-906; Sloboda, 2005, p. 256; Pfordresher & Brown, 2007).
   b. **Result:** All exercises (including the Baseline Finder) do not include accompanying music so as to not confuse those deemed amusical. To be thorough, I opted to not include sound effects when targets are collected.

2. Exercises will include live, real-time interaction with pitch production practices.
   a. **Reason(s):** Singers with amusia exhibit a discrepancy between what the “brain knows” and the “mind does not” (Mitchell, 2011, para. 6); most poor-pitch singers do not experience a perceptual deficiency, but rather, the mismapping in the
conversion from what is heard to what is produced (Pfordresher & Brown, 2007, p. 114).

b. **Result:** All play interaction within the application uses real-time response with minimal lagging in order to increase user-awareness of his/her own pitch discrepancy. Due to the volume issue of the plugin as noted earlier, a user may need to be close to the device and in a quiet space to achieve the best response.

3. Whenever possible, participants will need the use of words while singing.
   a. **Reason(s):** Symptoms of amusia include failure to recognize familiar tunes and the inability to distinguish one tune from another without lyrics present (Stewart, 2006, pp. 904-906). Amusics, therefore, are more apt to distinguish one tune from another with lyrics present.
   
   b. **Result:** With the exception of both the Baseline Finder and the Vertical Slider, the exercises use words to help the user distinguish between pitches. The Baseline Finder and the Vertical Slider instruct the user to sing using “la.”

4. Invoke the abilities that the tone deaf may be able to do well already.
   a. **Reason(s):** They are able to process speech intonation, recognize variations in articulation, tempo, timbre, and the emotionality of a selection of music (Sloboda, 2005).
   
   b. **Result:** Through the Speech Inflection exercise, I can access abilities that a tone deaf person can already process naturally, such as speech intonation and tempo variation. Due to limitations, I was not able to explore more of this aspect, such as affecting articulation, and timbre. Also, I do not yet know how to implement the emotionality of a piece of music within the context of this application.

5. Use a range of pitch intervals, quartettones included.
   a. **Reason(s):** Amusics may be able to decipher quartettones, which was previously thought unattainable (Hutchins, Zarate, Zatorre, & Peretz, 2010; Peretz, Brattico, Jarvenpaa, & Tervaniemi, 2009).
   
   b. **Result:** The application contains the ability to move according to whole and half-step pitch adjustments. Although the plugin has the ability to detect and return frequencies, I could not determine how to script it to that end. As a result, most times
a user will need to sing up or down a whole step before the character will show visible movement. I consider this to be a great limitation to my application.

6. Design pitch-capture abilities that allow for small movements in frequency.
   a. **Reason(s):** Amusics may be able to perform the general contour of a song, though they may be unable to report later whether an interval was moving up or down (Lebrun, Moreau, McNally-Gagnon, Goulet, & Peretz, 2011).
   b. **Result:** The minimum movement that can currently be detected is a half-step. As I described under Guideline #5, I could not script the plugin (pitchDetector) to move according to frequency, which would have been ideal since it would’ve allowed a user to see the result of his/her effort in adjusting pitch. As it is, it moves according to midi note, so if one sings a C and a C#, the character doesn’t move—it stays on the same line. There were other limitations with the plugin itself, namely, the volume. The company that produced the plugin has recently issued a newer version that may have alleviated some of these problems since it is built for the same version of Unity I used to create the application.

7. Don’t make participant response subject to a beat in time.
   a. **Reason(s):** Amusia sometimes includes the inability to keep a beat (Sloboda, 2005, p. 256).
   b. **Result:** There are no beats present in the application, with the exception of the timing used in the Interval Training exercise, which could be construed to be subject to a very slow tempo, depending on user preference. The absence of beat constraint allows the user to sing with little to no pressure of a timed completion.

**Findings**

I have learned many things throughout this experience that it is difficult to summarize them. In short, I better relate now to people when they say that they can read or hear a studied language and understand it before they can speak it themselves. I can look at a script and say, “I see what they are doing here” and I can imagine what the end result will look like, but it is still a struggle at times to replicate the result without first consulting my own personal “language” dictionary (i.e., the Unity API online reference guide). Regardless of my amateur attempts, my
logical approach to solving problems in the scripting world has greatly improved; I am now able to solve most of my issues on my own by breaking it down step by step, distinguishing between the known and the unknown. For instance, watching people like Marc Midura comb through a scripting problem has not only taught me how to work through issues independently, but it has also led me to believe that if Sherlock Holmes were real and living today, he would be an app-developer.

After all my effort, there are still limitations with the application. The ability to move the character in smaller increments according to frequency instead of by midi note is a result I would like to see come to completion, and the issue with the volume needing to be lowered is problematic. I also believe that the flow between menus and the transition between scenes and levels could be improved. These are issues that I will continue to address as I work to bring this application to a level that is worthy to be distributed.

**Implications**

Although this application is meant for individual use and for those who are clinically tone deaf, there are implications that this application, and the principles learned from it, can be used in other ways. Many people believe that they are tone deaf when, in fact, they are not. With this application, one can learn common intervals and build confidence singing these intervals in a private setting. Even those who are confident in their singing ability, such as an ardent choir member, may find it useful to not only strengthen their sight singing ability, but also strengthen and widen their vocal range by altering the baseline to a higher or lower pitch.

Implications also reach the classroom setting. From the principles found within the guidelines, a music teacher might best serve a student by visually showing, with gestures, the current pitch of a singer and the progression of that pitch towards a desired end result. Instead of playing the accompaniment louder to direct the wayward pitch center of a student, a teacher could have the student sing with others in unison, utilizing similar timbres to direct pitch. A teacher may also refer a student to a particular level within the app to help with a problematic interval. Those that teach languages might find it beneficial to teach a song in the new language using the method found in both the Interval Training and the Speech Inflection exercises, where
words are relative to notes. The same concept can also be applied to exhibit the proper speech inflection of a new language (e.g., the lift at the end of a question, the musical component of the Mandarin language, etc.).

Conclusion

I started my creative project with the intention of developing an app to help people with amusia. I still hope that, with a bit of refinement, this will still be the case. However, I have learned so much more than I ever though I would. Not only have I learned a coding language, and a software program I can use to develop learning games, I have discovered how perseverance and determination can be used to accomplish your goals. This process was extremely challenging. I had to work hard and struggle. I had to seek out help from outside resources and other people. By sticking to it, I was able to open up a whole new world and a great new resource I can use to help me as I teach my future students. I can also use my creative project experience to help my students understand that dedication to a goal can pay off in unexpected ways. This project has been a great way to conclude my Masters degree program. I am looking forward to applying all I have learned to my future endeavors as an educator.
References


