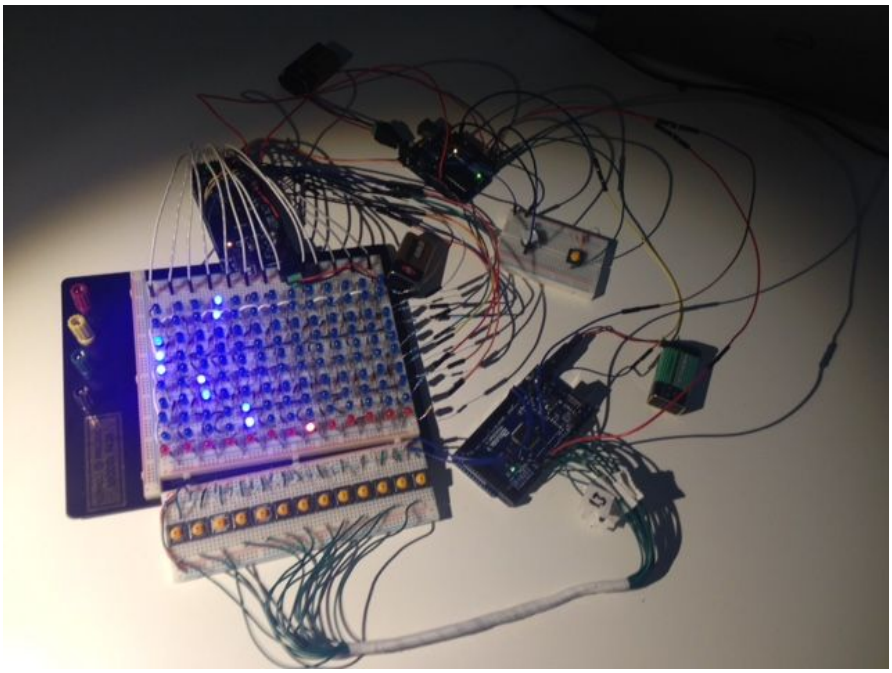
Synth-Hero



Group Members:

Deon Zoss, Kathleen Mancillas, Dale Wesa

Background:

Discussion of our project began with a desire to make something musical. All musicians, we were drawn to answer the question: “what can arduino accomplish in the world of music?” -- Our first instinct was to make an instrument. One of us wanted to build a synthesizer, the others wanted to make a music type game. So, we decided to put the ideas together and build “Synth Hero” -- an Arduino based synthesizer version of the popular 00’s video game *Guitar Hero.* We would leverage a large array of LEDs for our display, push button switches for the piano keys, and a square wave oscillator for

the sound itself.

Settling on our final design:

Initially we wanted to make the synthesizer from scratch, the core of which would be a 555 timer rigged to an array of different resistors, and played via an arduino over serial/flip-flop. We Opted however for a simpler design, choosing to use the built in library *tone*. We will talk more about this specific library later. The ultimate reason for this design choice was scope. Were this a project to only build a synthesizer, we would want to have a keyboard with more functionality. I.e. different waveforms, multiple voices etc. However, considering the point of our project was to build a game, as long as the synthesizer played the correct notes, we are happy. Adding the requirement to build a whole synthesizer from scratch adds unnecessary hardware complexity that wouldn't add real value to the end user.

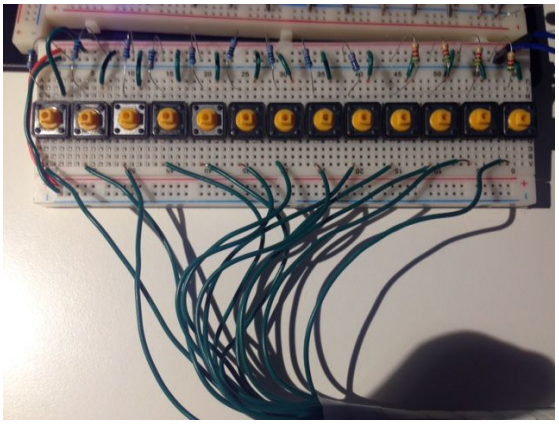
Another reason to change a design usually involves availability of materials. If Apple could affordably put leica cameras in each iPhone and still charge the same for them, you bet they would. We hit a bump in the construction of our project where we weren't able to get our hands on a proper circuit board large enough to fit our massive LED array, all of our buttons in the shape we had hoped, and had copper lanes on the back for us to solder to. We ordered what we hoped would be an adequate board, but after careful consideration of the fact that it shipped to us without a copper backing, we decided it would be best for the overall success of the build our product on a breadboard. This came with the "feature not a bug" decision to change our original

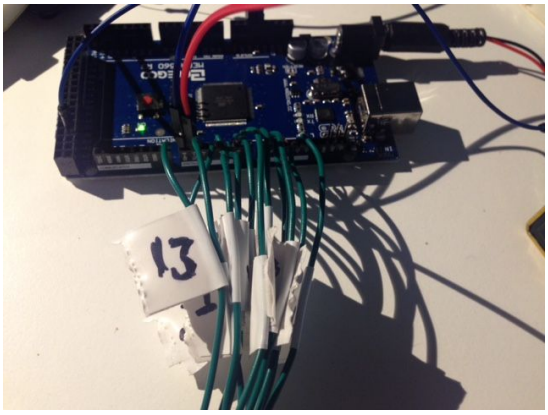
button layout to be all in a line instead of in two rows as you would find on a piano. We believe that while this makes our game less piano like, the end experience is easier to understand and more fun for the user.

The purpose of our project was to teach the user how to play various songs. The songs would be displayed as trailing LEDs in a matrix. When the LED that was lit became the bottom LED (the red LED) that is the signal for the user to press the button on time. The button then produced a certain tone from the buzzer, and if the user correctly pressed the right buttons at the right time, the desired song would be played.

Individual components:

Buttons:

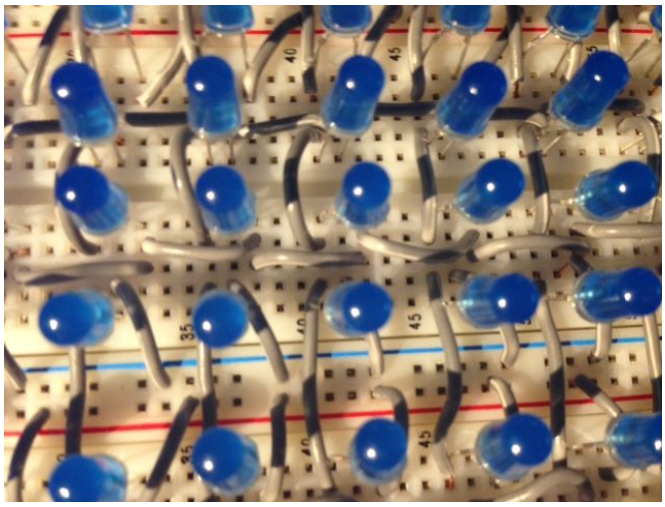




Our keyboard consisted of 13 buttons. We decided on this amount of buttons to use because that was exactly how many buttons we could fit along our breadboard. Due to the relatively small amount of buttons, we chose to make the buttons play a major scale (as opposed to the standard chromatic scale, where each note is a semi-tone

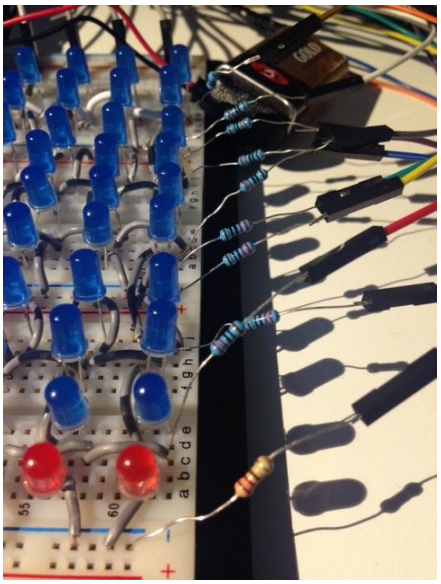
apart from each other). Choosing this scale prohibited us to only display songs that fit the major scale.

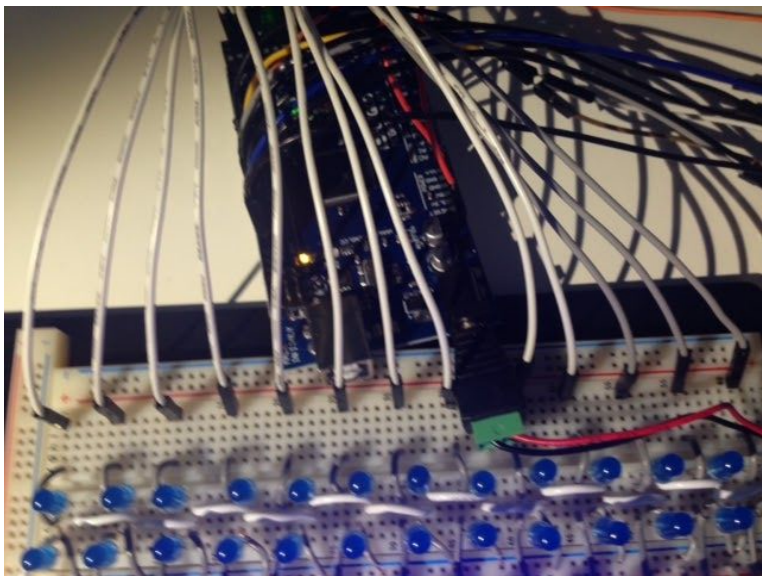
LED array:



The array was the most time consuming of what we had to do for our project. The number of columns that we allowed for the LED array was dependent on how many buttons we could fit on the breadboard; 13. Deciding how many rows we should have was a matter of trial and error, and ended up deciding that 10 rows spaced about a centimeter apart was the best to display a relatively smooth experience for the user. The LED array was connected by multiplexing the rows and columns. The anodes of the LED's in the same column were all connected together, and the cathodes of the LED's in the same row were all connected together. In order to make the matrix easily visible, and also due to a limited number of jumper wires, we cut and stripped wire to connect these LED's, which took a considerable amount of time. Resistors were placed on the end of each row as well to limit the current going through each row. We used 23 pins for

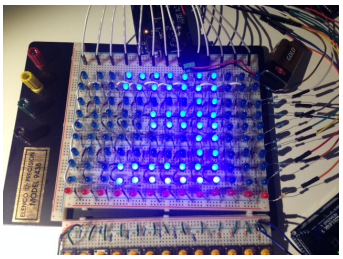
the 130 LED's, and purchased an arduino Mega for this reason, which has 54 pins. Multiplexing the array allowed us to choose the row and column of the LED we wanted to light. Lighting up multiple LED's at once had to be done through the process turning on and off the desired LED's sequentially, and very rapidly. Doing this quickly tricks the eye into thinking that the LED's are all on at the same time, but are actually done back-to-back. Red LED's were placed at the very bottom of the matrix, to help the user know exactly when to press the note on time.

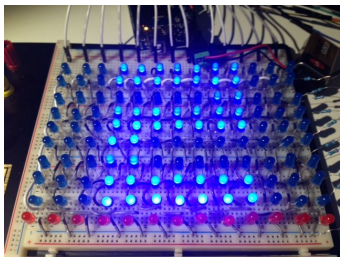


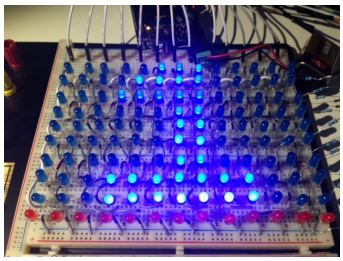


As far as the code is concerned, a dynamic array was malloced depending on the number of rows and buttons that we had. This array then stored integer values. Notes could be added to the array with a function that changed the value of the first row of the array. The value had to be greater than one in order to be displayed, anything less that one was regarded as the LED at that index in the array being turned off. The value resembled the duration of the note to be played. A value of one signified that the note should be played for one beat, and was simply one LED

turned on in the array. A value of two would display a trail of two LED's coming down the proper column in the array, and etc. for values higher than two. Another function would then update the array to be displayed for the next beat in the song. LED's at an index of value 1 would be turned off by decrementing its value and setting the note below it to one. LED's of value 2 would be done the same way, except now the old index is now decremented to 1, it will still be displayed, and this process repeats for the entire column until the note is at the last row. A variable for song speed was also implemented to tell the array how fast it should update. The intro effect was done by telling the matrix to display multiple notes in a pattern using a loop and with fast song speed. The countdown effect was done by setting the entire matrix equal to a matrix of ones and zeros that resembled the desired number, and then these number matrixes were run through the same updatedisplay() function used for the trailing notes.



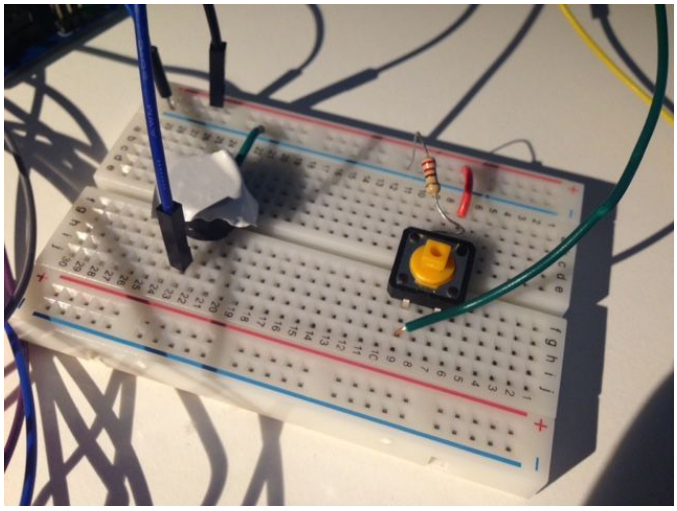




There was a comment from a roommate that the rudimentary synthesizer that we made sounded like a bagpipe, so we chose the song to display to be the common

bagpipe-played song “Scotland the Brave”. The song also conveniently provided the right amount of difficulty and variation that we were looking for. The notes of the song were individually added with function calls to add the specific note at a specific time. This process could have definitely been performed better, perhaps reading from a song array which has encoded notes that are read and added automatically. However, this level of abstraction is something that we did not have time to implement in the final project.

Speaker/Reset Button:



The third arduino received information from the arduino using the keys, and produced a tone via the buzzer. We put a piece of tape on top of the buzzer to change

the sound to be more full and more pleasant sounding, The button also outputs to the arduino handling the LED matrix to refresh the screen.