1. **Introduction** Our final project will be to create a language that has expressions in the form of notes, chords, and sine functions, and combines them in various ways to output a tone or a chord and a printed sine graph. The problem of representing music through a programming language is not a new problem. However, while a lot of computer interpretations of music exist, a lot of them translate the language of music into sheet music with automatic playback or randomized improvisation of notes. Both of these solutions offer useful ways for a user to creatively interact with computer simulated music or instruments. In our language, we solve the more original problem or question of the relationship between mathematical functions and musical notes. Our language takes in combinations of notes and translates those notes into a sine function. The sine function displayed can also be edited to be either a sum of functions or a sum of sine functions and chords. The function graphed is also played through audio representation of sinusoid. This relationship need its own language because a musical language already exists, so it can be translated into a language that can be both a musical language and a programming language. Our language, by representing musical output as a result of adjustable keyboard notes, a sum of sine functions, and a combination of note values and sine functions, is a new method of presenting the relationship between musical notes. As most people think of the distinction between pitches as the difference between black dots on a staff or distance on a keyboard, our language gives people a new way to think about the relationship between notes and chords by translating them directly from the notes on a keyboard to the visual change in amplitude of a sine curve. The visual output of our language also has the educational benefit of explaining the relationship between sine functions with different coefficients, what amplitude is, and what happens when sine functions are added together.

2. **Design Principle** The guiding aesthetic goal of our design will be to create a smooth integration of musical notes and the math behind them. The two things are in some sense the same, but are very difficult to relate. Part of the problem is that it is very difficult for our brains to imagine a conversion between the audio of notes to a visual mathematical representation. Therefore a major goal of our

language and representation will be to have simultaneous visual and audio representation of a note or notes. The user should be able to hear any sine wave and see any note. In extension, it is important for the language itself to facilitate connection between notes and sine waves. To help the user connect the math to the sound, they should be able to manipulate notes and sine equations as if they are the same thing and then be able to instantly hear the result. In line with the philosophy that languages are not just a tool but how you think, the goal of the language is to allow people to think in math and music at the same time.

3. **Example 1** Play $sin(440t) + A + 440Hz$ Writes a .wav file to the directory, that when played separately by the user, is the note A This example can be run by the following: dotnet run example-1.gb OR in the repl (dotnet run)

4. **Example 2** Let am = A + C + E Draw am + B + sin(220t)

   am evaluates to A + C + E, so the end result is these three sine waves plus the sine wave of B plus sin(220t), so it graphs that in a popup window. This can be run by the following: dotnet run example-2.gb OR in the repl

5. **Example 3** Let x = C + E + G Draw x Play x

   This program stores a variable x as a C major chord, then draws it in a pop up window and writes a .wav file to the directory for the corresponding chord. This can be run by the following: dotnet run example-3.gb OR in the repl

6. **Language Concepts** At a very basic level, everything reduces to sine waves in the form sin(ft) where f is the frequency (Hz) of the note produced. The primitives, then, are the 3 different ways to represent that sine wave: a musical note, a sin(fx) function, or f Hz. The combining forms are chords that are made up of these primitives, which take the form of mathematical expressions when the primitives are represented as sin(fx) expressions, musical chords (Am, A,C,E, A + C + E) when the primitives are expressed as notes, or sound outputs when the primitives are expressed as frequencies. When writing in this language, the user needs to know that these three

forms can all be expressed synonymously, but does not need to know how to convert between them.

7. **Syntax** The syntax of this language, in its final form, will be represented very visually because the UI will consist of a keyboard with keys that can be pressed and a sum of functions that can be manipulated.
   Variables
   Sine function - the primitive value in this language, sine functions will be expressed in the form $sin(A(2\pi t))$ where t is in seconds
   Hz - another primitive value of the language is Hertz, represented by an int A
   Note - a note is composed of a char that represents the name of the note and a sine function
   Chord - a chord is represented by a string (the name of the function) and a list of notes
   Expressions - expressions are either sine functions, notes, chords, or a combination of two expressions
   Functions
   Play - the playback feature is the process of evaluation, that will take an input expression from a chord or a variable, convert it into a single sine function, and then play back the corresponding sound: either single note or chord.
   Draw - the drawing evaluation function does the same as play, but the output is in the form of a popup window graph
   Add - addition will be represented by + and will sum together chords, notes, and sine functions to create compound expressions to be evaluated
   Updated BNF
   $< expr >::=< chord > | < op > | < var > | < expr >< expr >$
   $< chord >::=< note > | < note >< chord >$
   $< note >::=< Hz > | < sin > | < char >$
   $< op >::=< play > | < draw > | < assign > | < add > | < mult >$
   $| < div >$
   $< play >::=< expr >$
   $< draw >::=< expr >$
   $< assign >::=< string >< expr >$
   $< add >::=< expr >< expr >$

$< mult >::=< float >< expr >$
$< div >::=< expr >< float >$
$< var >::=< string >$

8. **Semantics**
   Note: There is not precedence in any operation as they are all commutative.
   Also, every type evaluates to a chord * context type

Syntax	Abstract Syntax	Type	Meaning
nHz | sin(nt) | A	Hz of Int, Sine of Int, or Key of char	Note	Notes are a pimitive in the language
c | (n + n)	SingleNote of Note or Notes of Note list	Chord	Chords are combinations of one or more notes that are by default added together
c	ChordVal of Chord	Chord	Stores one chord as an expression
play e	PlayOp of Expr	Expr	Creates a WAV file of the chord resulting from evaluating the expression it is given
draw e	DrawOp of Expr	Expr	Creates a gnu plot of the chord resulting from evaluating the expression it is given
e + e	AddOp of Expr * Expr	Expr → Expr	Adds the two expressions and returns a new Expr stored as an Expr list
e/float	DivideOp of Expr * float	Expr → Expr	Divides the sin function form of the chord by the given float
e*float	MultOp of Expr * float	Expr → Expr	Multiplies the sin function form of the chord by the given float
let s = e	AssignOp of string * Expr	string * Expr	Adds a new assignment from a variable s and an expression e
s	Variable of string	string	Stores a variable as an expression
e	Expr	Expr	One of the above types, any valid form of syntax in the language

9. **Remaining Work** An additional feature that could be added is

initializing the language with a Map of variables assigned to chord names (major, minor, etc), but since the feature exists for the user to declare those chords when necessary, it is left to the user to declare chords. As in our initial concept of the language, having a GUI that presented the user with a keyboard to play would abstract this language to more of an interactive program whereas its current state allows the user to write a sequence of commands that are operated as a sequence instead of just being an interactive platform. It would still be an interesting and fun challenge to create a GUI, but serves a somewhat different purpose as the language currently.