Hasmusikell

Delving deeper into Haskell with Html

Luk, Wallace • wsluk@ucsc.edu
Cruz, Steven • sicruz@ucsc.edu
Li, Tommy • tocli@ucsc.edu

Plan

The idea behind our project was to delve deeper into the functionalities of Haskell in order to produce sheet music. Users would be able to generate their own music by using our own DSL (domain specific language). In order to display our content that represents what sheet music typically looks like, we use HTML to display the notes, time signatures, and all the other necessities in sheet music. In order to combine both the uses of Haskell and HTML, we use BlazeHTML, which is a blazingly fast HTML combinator library for the Haskell programming language. What BlazeHTML does is that it embeds HTML templates in Haskell code for optimal efficiency and composability.

Feature Specifications

- Requires a title of song in order for the program to run (will throw out error warnings)
- Optional uses of flats and sharps (can be used as a key signature or throughout the measures)
- Time signature is default as common time
- All notes must equate to 4 beats per measure unless time signature is changed to a different beat
- Sheet music always starts at middle C unless specified

Configuration Characteristics

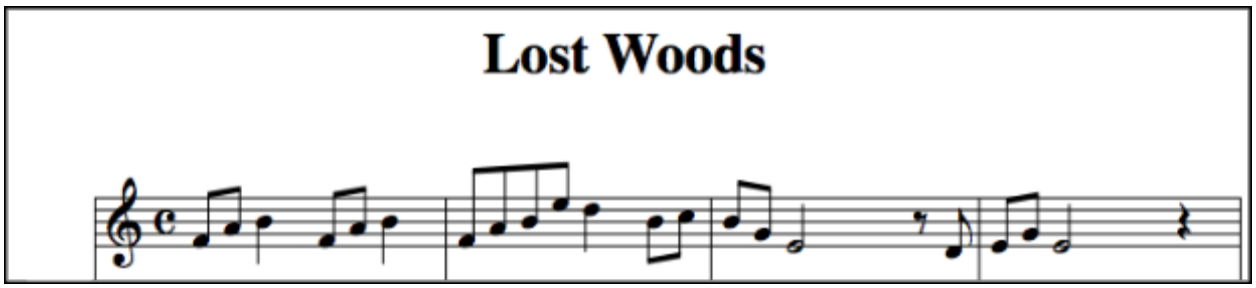
- Each note has a specific duration in terms of beats
- If sharps and flats are not defaulted, it can be added within the stanzas, and as well as naturals
- Notes can be played octaves up or down
- Filters out the last stanza of the sheet music to however played out

Definitions

- BPM (beats per measure)
- Measures consists of 4 beats
- Durations for each note are followed as:
    1. 4 is a quarter note (1BPM), 3 is dotted half (3BPM), 2 is half (2BPM), 1 is full (4BPM)
    2. C' (represents C an octave up), C_ (octave down)
    3. | separates each measure
    4. \n splits to new stanza
    5. nC is natural C, bC is flat C, and #C is sharp
    6. A4, D4, G4, C4 is a full chord note
    7. r denotes a rest

Example of a sheet generated by a user

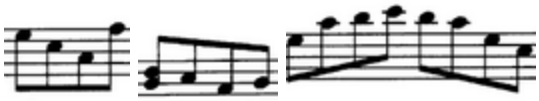
1. title: Lost Woods
2.
3. F5 A5 B4 FA5 A5 B4 | F5 A5 B5 E5 D4 B5 C5 | B5 G5 E2 r5 D5 | E5 G5 E2 r4



Changes from Original Plan

Because it is difficult to project an expected outcome from a programming language (both Haskell and HTML) never used before, there were some slight modifications and dropped proposals for the product. Due to Haskell being a gift where multiple features and libraries could be used as an advantage, it also is a curse where it throws you endless compiler nightmares.

Our original intent was for the user to create their own song by pressing the keys that correspond just how a piano was designed. Time restriction and Haskell had other plans for us. Also, dealing with the bridges on sheet music was cumbersome to deal with due to the numerous ways bridges can be drawn as shown below:



Worth noting that regardless of whether the bridges were not implemented, we still decided to put the sixteenth notes and eighth notes together with the same distance the bridges contain.

Methodology and Implementation

Because we learned about parsers rather late in the quarter, all of the parsing functionality was based on the short chapter in *Real World Haskell* on Parsec, the parsing library for Haskell. Then realizing that parsing for specific strings of the sequence [#bn]?[ABCDEFG][123456][_']? would be complicated, we opted to use regular expressions, which was detailed in another chapter of *Real World Haskell*.

Converting it to HTML at that point would just be a matter of dealing with a list of list of list of Strings to represent the list of stanzas which is a list of measures which is a list of notes/chords. To do this, many recursive functions were written, the conversion to look at it abstractly looks like this [[[String]]] -> [[[Sound]]] -> Html, where Sound is a data object we created to keep track of the specific qualities of the note/chord. See MusicSheet.hs for details.

Going from and triple list of Sound objects to Html involved carrying around a lot of values if we wanted to place an image on the a specific position in the Html, this required us to carry x y positions as well as the key signature around in another data object called Manager. See blazeSheets.hs for details. Our main loop is also there.

Challenges

Getting off the ground was pretty hard, since initially we still weren't entirely sure what the language would look like, or if it could encompass all the intricacies of sheet music. A lot of time was taken just to talk about the what the language would look like and consider specific cases. Dealing with a triple list object was also difficult to wrap your head around at first. Delegating tasks was also difficult as the program by nature wasn't all too modular. Also, printing the triple list object into HTML with blazeHTML was challenging since we needed to map all the notes to the right position with pure HTML and CSS code.

Conclusion / Future Plans

Seeing as this project offers no real practical purpose, there doesn't seem to be a reason to continue this onwards as there are many other decent programs that generate sheet music for you (that also aren't constrained to just commontime nor limited to using bridges). However, we learned a lot about Haskell and parsec throughout the group project. Once we had a strong understanding of how parsing worked, it helped give us a clearer understanding (coincidentally) of the homework required. However, we will leave the project as it is right now as we believe that there are many other programs that function the same way, just in a different language.