Rhythmr, Or Semi-Automated Audio Loops

Gregory Michael Travis
Independent Researcher
greg.m.travis@gmail.com
https://github.com/GregoryTravis

Introduction

Rhythmr is an interactive system, implemented in Haskell, for creating ear-pleasing rhythm loops from randomly-acquired audio files. The user iterates through an unending stream of randomly combined music loops, swiping left or right on each one to indicate whether they like it or not. Rhythmr derives a model of which combinations sound good based on user input, and from that, produces a complete song by inserting the chosen combinations into a predefined score.

In a previous project, I created a tool for generating songs from a pool of musical loops completely automatically. After generating thousands of these songs, I hand-picked sixteen and called it an album. However, Rhythmr takes this concept a step further, placing the user in the middle of the process to select individual elements before they are combined randomly.

Aesthetic choice cannot be automated, but it can be optimized. One way to look at creativity is to view it as a kind of generate-and-test process, albeit a largely unconscious and invisible one. Rhythmr makes this process explicit: actually generate material, and let the user select what to keep.

Haskell was crucial to this project, for it allows efficient construction of reliable software, and pure state handling made implementing features like ‘undo’ very easy.

Aesthetic Model

It could be said that Rhythmr is based on a narrow, Western definition of what sounds good: regular beats, 4/4 time, and so on. To that end, Rhythmr is useful for finding musical loops that fit nicely together, but uniformity and consistency are by no means the only aesthetic goals. Sometimes you want things to be weird, and Rhythmr supports this as well: you, as the user, can simply swipe right on weird things, and just see what happens.

Model

We say there is an affinity between a set of loops if the user swipes right on that set, or if we can guess that the user would, based on previous choices. We use the notation a ~ b to mean that a and b sound good together: they have affinity.

In addition to user choices, we can derive affinities using four rules:

The Subset Rule: If a set of loops is good, then any subset is good too. This doesn't mean that it's as pleasing to the ear, just that it's unlikely that removing a loop will cause the remaining loops to somehow clash.

The Superset Rule: If a set of loops is bad, then any superset of those loops is bad too. Adding a loop is unlikely to hide an existing clash between loops (although that is possible).

The Transitive Rule: If a ~ b and b ~ c, then a ~ c.

The Exception Rule: If the other rules deem a mix good, but the user deems them bad, the user choice takes precedence. After all, exceptions to the first three rules are certainly possible, but they are too tricky to identify by anything but user choice.

These rules are heuristic. Consider the possibility that a group of three loops sounds good together, but any two of these loops do not. This seems unlikely, but it is possible. The rules listed above are based on the idea that this is unlikely but possible, and will incorporate these odd judgments into the affinities it derives.

The transitive rule, in particular, can produce affinities that don't sound good. If two groups of harmonious beats intersect at all, the transitive rule combines them into an affinity group, and it's entirely possible that this group as a whole does not sound good. Additional user choices will sever this bad connection.

Process

Rhythmr leads the user through a series of mixes, asking them to swipe left or right on each one. The mixes are derived in several different ways, outlined below. At each step Rhythmr will pick the strategy, or the user can also choose the strategy.

Random: Rhythmr picks two to four loops randomly from the pool. Because the pool is a subset of the entire library, there is a moderate chance that the randomly chosen set will intersect with previously judged loops, which allows the model to combine smaller affinity groups into larger ones.

Incremental: If the previous group was judged good, Rhythmr replaces just one of the loops with another loop. The resulting group thus has a higher likelihood of being good as well, since the group is similar to the last one.

Subsets: If the previous group is judged bad, then from that group of n, Rhythmr generates all possible subsets of size n-1. This approach has the downside that it generates a lot of very

similar proposals, but it tends to find better and larger affinity groups simply because more intersecting combinations are tried, and the user is asked to make finer distinctions.

Divide And Conquer: If the previous group was judged bad, Rhythmr divides it in half and proposes each half in turn. If either of these halves are also judged bad, this is repeated. The idea is that a mix might be compromised by a single beat that is generally out of step with any regular rhythm, or two loops that compromise the rest of the mix, and this strategy swiftly eliminates them.

User Interface

Rhythmr has a graphical user interface, although it dispenses with buttons, menus, and so on. Instead, the application window shows the state and behavior of the model, and choices are made using the keyboard. (When I port Rhythmr to run on mobile devices, I will definitely implement a Tinder-like swiping interface.)

The window is divided into three sections: pool, affinities, and song.

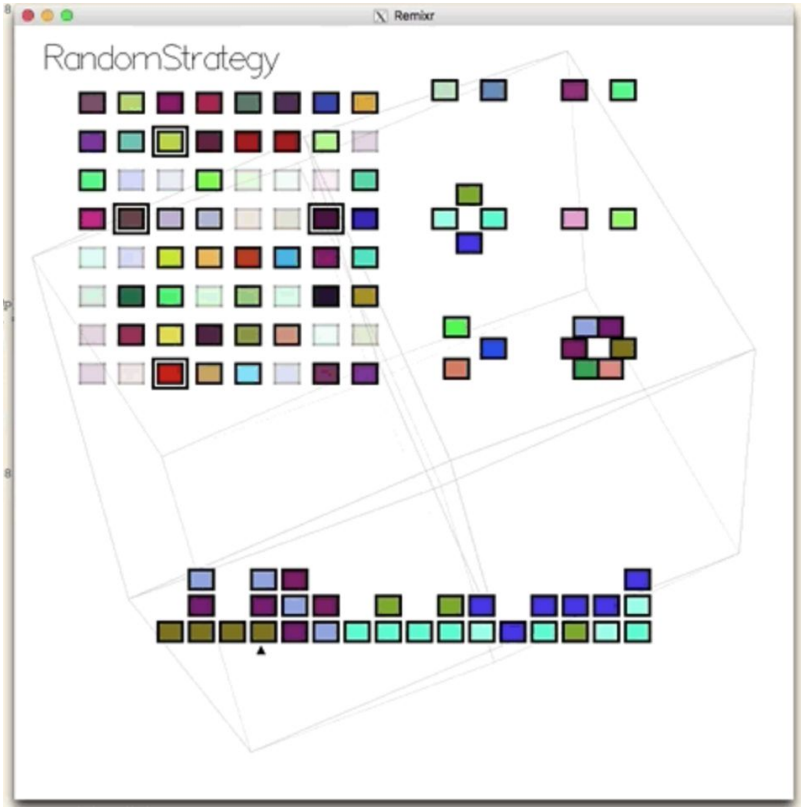
Pool: The pool is a representation of 128 loops from which all mixes are taken. When the loops are judged good, they move from here to the affinities section.

Affinities: The affinities section shows clusters of loops -- these are loops that are considered to work well together based on the user's choices. You can see clearly which loops have been judged, either by the user or the model, to have affinity.

Song: If there are enough affinities, they can be used to generate a song, which is displayed in this area and played out loud.

Rhythmr also supports saving, loading, undo and redo.

Screenshot



Terminology

- Good / Bad
    - A mix is judged good or bad by the user; these judgments are used to make further proposals, and to generate affinities that can be used to generate songs from scores
- Affinity
    - The quality of two or more loops sounding good together
- Affinity Group
    - A set of loops that are judged to sound good together, either by direct user judgment, or by deriving such a judgment from previous user judgments
- Left / Right
    - Directions to “swipe” on a mix, to judge it bad or good (resp.)
- Judgment / Choice
    - A user decision about whether a mix sounds good or not
- Proposal
    - A set of loops combined into a mix and played out loud, with a subsequent judgment by the user
- Group

    - A set of loops
- Loop
    - A single piece of audio, generally a single measure
- Beat
    - A rhythmic loop
- Mix
    - A set of loops mixed together into a single loop
- Pool
    - A set of loops that are used to put together proposals for the user to judge
- Library
    - A large set of loops from which a pool is chosen on startup
- Model
    - A method for determining affinity groups from user judgments; also used to pick new mixes to propose
- Strategy
    - One of four different ways to pick the next mix to propose to the user
- Score
    - A composition schematic into which loops can be inserted to produce a song
- Song
    - A full, automatically generated piece of music that is constructed by selecting groups of loops to insert into slots in a score
- Subset
    - A subset of a set of loops
- Superset
    - A superset of a set of loops

Source Code

https://github.com/GregoryTravis/rhythmr