# Beat Genetics

Programming Assignment: Implement A Working Generative-Creativity System

Written Report

Candidate Number: 19329

Generative Creativity: G6004 (UG)

University of Sussex

May 2014

# Beat Genetics:

## 1.0 Overview

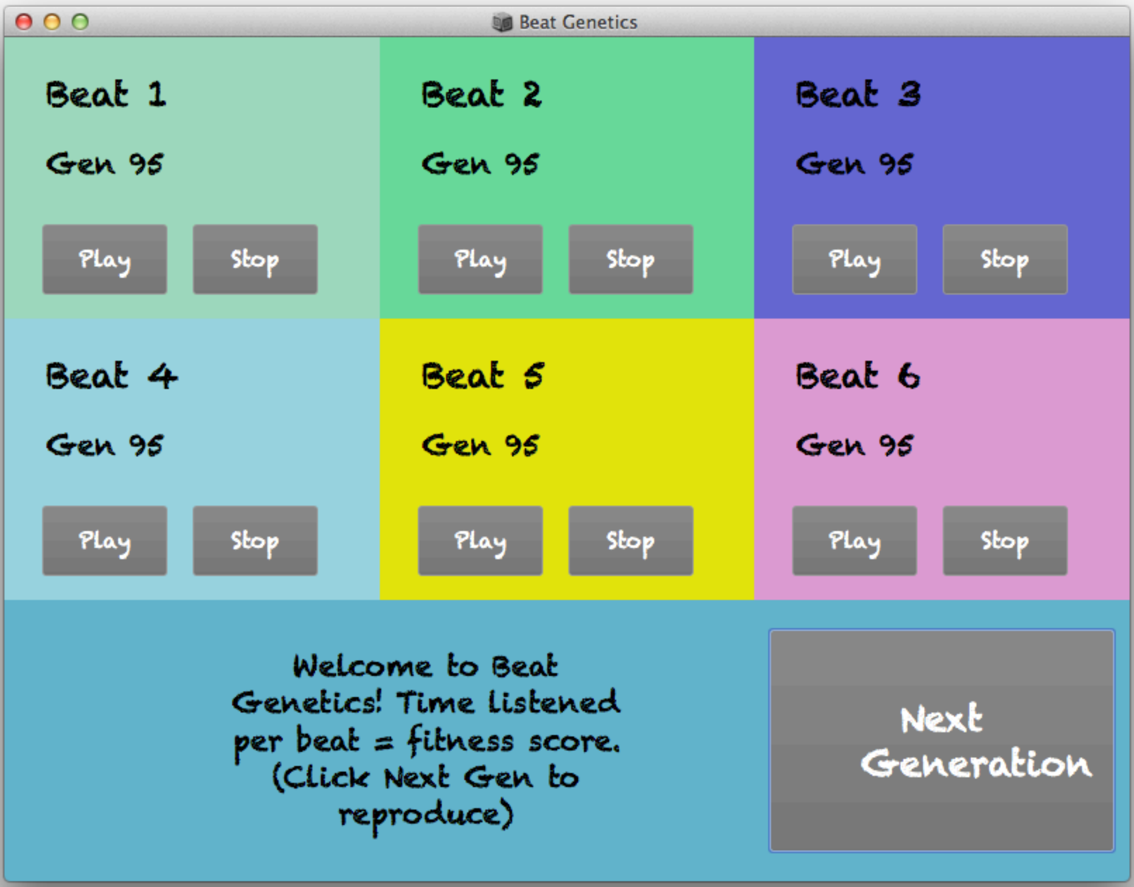
Beat Genetics is a genetic orientated system that generates and evolves drum beat sequences based upon the subjective aesthetic evaluation of the user. To realise this project, an interactive genetic algorithm was implemented whose fitness function was determined by the amount of time a user spends listening to a particular beat. The longer a beat is listened to, the greater fitness score it receives. This work was inspired by Karl Sims' '*Galapagos* [1997] where the fitness of an image is tied to the length of time that viewers look at the image[1]. "This is known as interactive selection, a genetic algorithm with fitness values assigned by users" [Shiffman, 2012].

The main goals of the project were to create a system that would tailor generated beats into patterns that the user could use for musical compositions. The hope was that after an amount of generations, beats would emerge that were in line with the user's subjective taste. Since each generation is dependent on the beats most listened too, the output of the system is in the hands of the user. If they are looking for fast beats then they need to spend more time listening to the faster instances. This subjective analysis applies for any kind of beat they choose to seek at a particular time.

Since the output is based upon a person's subjective opinion, the system can mutate and optimize specifically for that one user and act completely different for the next. It allows subjectivity to live and breathe and therefore it became a very flexible application that people could use to generate any type or style of beat they pleased, allowing anyone's unique aesthetic taste.

[1] Daniel Shiffman on Karl Sims' Galapagos, and a deeper explanation of Interactive Selection [http://natureofcode.com/book/chapter-9-the-evolution-of-code/, last accessed 10/05/14].

## 2.0 Illustrations



Welcome to Beat Genetics! Time listened per beat = fitness score. (Click Next Gen to reproduce)

Fig 2.1 Beat Genetics Graphical User Interface

Figure 2.1 illustrates the Beat Genetics GUI. It is divided into six sections, each for every beat in the population. The beat number is listed and so is the generation number underneath, which increments each time the user clicks the next generation button.

To make sure fitness scores didn’t interfere with one another, I made sure that only beat could be played at a time. In addition the next generation can only be executed once all beats have been stopped to make sure when the user next listens to the beats, they hear the newly bred children rather than their parents.

The Genotypes of Beat Genetics consisted of two arrays; the first contained 1s and 0s which corresponded to the phenotype of a hit or not hit. The second contained floating point values, whose phenotype related to the length duration of a beat.

## 3.0 System Overview

As discussed Beat Genetics uses an interactive genetic algorithm to generate beats that are tailored to the user's subjective evaluation. The overall program functions as follows:

1. Generate a random population of beats for all instruments: (Kick, snare, closed hihat, ride cymbal, tom, clap, percussion one, percussion two)

    a. A single drum beat is created using two Pseq arrays that will be imbedded into a Pbind. The first array dictates the level (whether a drum beat is played or not) and the second array assigns the duration of each hit (measured in note lengths).

    b. The array sizes are randomly selected from a range of 1-32. The level array values have a 50% chance of being a 1 or 0 (on or off) and the duration array is populated with values 4/sizeOfArray. Meaning if size = 32 then duration values will be 4/32 = 0.125, which all sum to 4, the time signature that all beats conform to.

    c. Each instrument consists of the amount of beats dictated by the population size. E.g. if the population size is 6 then each instrument will have 6 different beats.

2. The graphical user interface is divided into six sections (the population size) to signify each current generation beat. The user then clicks play or stop upon each beat in turn and the amount of time spent listening to each beat is tracked by the program using the TempoClock. Time listened per beat is stored as a fitness score, the longer the user listens, the higher the fitness score is given.

3. Once the user has listened through each beat and fitness scores have been acquired. The 'Next Generation' button is pressed to create the new set of children beats.

    a. The fitness scores acquired are normalised to give probability values for each beat as a potential parent in the mating pool. Those with higher probabilities will be more likely to become parent candidates in the pool.
    b. Next, two parents are chosen at random from the mating pool to be used as parents for the child beat.
    c. The size of the child level and duration arrays have a 50% chance of being either ParentA's or ParentB's size.
    d. Crossover genes for each level and duration value have a 50% chance from either parent.
    e. Mutation has a 25% chance for each level gene becoming 0 or 1. And 25% mutation chance for each duration gene becoming 4/random(1..32).
    f. Since we choose between array sizes and mutate the duration values the sum of the durations may add up to less than 4 or more than 4, which is not ideal. Therefore a quantization method was used to either contract or expand the arrays to make sure the sum of all durations equals four and the level array size is equal to the duration size.

4. These steps are repeated until we have the complete next generation. Fitness scores are reset and the beat offspring are ready to be listened to and evaluated by the user for another iteration.

## 4.0 Evaluation & Limitations

Given the nature of the algorithm, the system can flow in many different ways. Since the fitness function is based upon a user's subjective aesthetic value, the beats can tailor to faster hits (smaller durations), longer hits, minimal beats, dense beats or a deeper variety of patterns.

Within the crossover function when breeding new beat children, the size of each level and duration array is randomly chosen between ParentA or ParentB. I initially programmed the system to always choose the shortest Parent size to avoid creating a quantization method. Implementing this though meant with every generation the beats would become smaller and smaller and eventually reduce to no sound at all. Alternatively I could have always chosen the Parent with the largest array size but this would of had the opposite effect by creating longer and longer length beats. Therefore the quantization function was the best approach to tackle such a problem, because it lets us randomly select the array size each time, meaning the beat length doesn't converge to either very small or very large but constantly varies its length throughout the family beat tree.

A large limitation of the system arises from its multi-layered functionality. Each beat consists of eight different instruments all with different rhythms. If the user enjoys the kick pattern in a specific beat but not the snare, there is no opportunity to just select the kick to be continued in future generations without also regenerating the unfavored snare too. So it performs the genetic algorithm as a blanket that covers the entirety of a single beat, rather than being specific for each instrument. To solve this problem the GUI could allow for different screens to represent different instruments within a beat, however this would mean the user would have to listen through each beat for each instrument = 6*8 = 48, too time consuming.

Another downside of the program is that after a while, it can potentially become tiresome to have to listen to each beat in turn. It ends up with the user wanting the computer to take care of the fitness function itself after the novelty has worn off. Perhaps an option to trigger a computational based fitness function that would let users fluctuate in and out of subjective evaluation could be a solution. Furthermore the fitness function can fail if the user starts listening to the beats they dislike more, perhaps because they are hoping for a change or are quickly jumping past the good ones, meaning after a while nothing but unfavored beats are generated.

After a number of generations if the user has preferred a specific style of beat the system will end up converging to the similar sounding style and will become hard for the user to break free from. To overcome this, mutation was increased from a 1% chance to a 25% chance but it could be increased further still to help avoid such convergence.

## 5.0 Conclusions

This was my first attempt at implementing a genetic algorithm within a programming project and I believe that I have achieved this to a good standard. Being able to mutate and cross-breed arrays of varying sizes proved difficult but the quantization method I created to solve this problem was innovative and helped achieve the goal of creating a genetically orientated beat generator. It fulfills the aim of being flexible per user, generating different style beats at the appropriate times.

## 5.1 Future Projects

As discussed within the limitations, having a multi-layered system that plays numerous rhythms at once can create problems for the user especially when they like one instrument but not another. Therefore future developments would see an interactive genetic algorithm that can handle this subjective distinction performing well on a multiple levels without being time inefficient.

Other areas to explore based upon the implementation created for this project could be to use a fitness function according to music information retrieval. For example we could send the beats created to the EchoNest API[2] to measure the 'danceability' of each beat. The beats with higher danceability scores will become more likely to be chosen as parents in the mating pool and the system's target would be to create the most 'danciest' beat. Similar genetic algorithms could be used with targets for audio analysis such as create an audio file with a certain amount of pitch, or onset level etc. or even go as far as measuring each instrument stem from a pop song and assigning targets from measurements taken, try and create an optimized solution for creating a hit pop song.

[2] EchoNest API [http://the.echonest.com/, last accessed 24/04/14]

## 6.0 Self Assessment

### 6.1 Writing (9/10)

Sentences are clear and concise. Each section is easy to follow and understand, with deeper technical language used within the system overview that dives into specifics of the SuperCollider code. Certain words such as 'system' could be used less and replaced with alternatives.

### 6.2 Discursive Quality (7/10)

My project doesn't really stir up or question a debate upon the creative validity of the system, but rather critically evaluates the system created giving good positive and negative viewpoints with improvement suggestions and argues the implications of implementing certain methods into the system. It outlines valid reasons for solving problems that arose when programming and provides suitable areas of improvement

### 6.3 Presentation (4/5)

Considering the project was more focused upon the implementation of an auditory system, the visuals were very simple using a straightforward GUI. Therefore this report just includes the one figure of the final system. Sections are clearly outlined, citations are correctly typed up, but could do with some more references.

### 6.4 Range of Research (4/10)

Its clear that this project bases its main functionality on Karl Sim's *Galapagos,* and for this reason most of my time was spent trying to emulate his system but for an auditory application. Therefore references are few and not too much literature was researched, except from following the teachings of Daniel Shiffman.

### 6.5 System Comprehensibility (5/5)

The system uses well structured classes, that are easily accessible by users. All code follows strict commenting guidelines and details the arguments for each function explicitly. Code is abstracted into classes and functions for easy de-bugging and straightforward implementation. Program is flexible and generalised for updating efficiently.

### 6.6 System Quality and Sophistication (18/20)

Although the GUI makes the system very easy to use and simple to understand, the sophistication of implementing the algorithms, allowing for user samples, creating an interactive user dependent fitness score system and more highlight the dense functionality coded into the system. It is a testament to the program that the user only has to deal with the highest abstraction layer of the system, so it can just run out of the box and in such a simple way, but the skill to program such a stable system meant highest quality coding on every level of abstraction from low to high, in order for the system to exist.

## 6.7 Quality of Output (18/20)

The system that was implemented itself is of a high quality, with deep architectural programming. The flexibility the program allows by tailoring its genetic algorithm per user's subjective evaluation is very widespread and doesn't create big restrictions. Its aesthetic output could be improved as sometimes the beats sound dissonant with each other, however it allows for user's own samples to change this and also it is the innate nature of the algorithm to start in a unfavored place and reach a favored sound. The intellectual value from creating such a system holds strong, with a very robust code set that handles amounts of synths/buffers and pattern executions very well so as not to confuse the user or the system, and doesn't break considering such a large code base.

## 6.8 Course Awareness (9/10)

This project extends the lab teachings upon Shiffman's tutorials of genetic algorithms. Where the labs and Shiffman tailor the tutorials to processing and using visuals, I chose to try and extend the learning within SuperCollider for sound. It was my first time implementing a genetic algorithm and I found it difficult at first to understand. After probing the tutorials multiple times I finally managed to get a working model within SuperCollider and think I achieved it to a high standard using a creative fitness function using the TempoClock.

## 7.0 References

Daniel Shiffman (2012). *Nature of Code*. Online: Magic Book Project. Chapter 9.12.

Karl Sims. (1997). *Galapagos.* Available: http://www.karlsims.com/galapagos/. Last accessed 24th April 2014.

The Echo Nest. (-). *The Echo Nest.* Available: http://the.echonest.com/. Last accessed 24th April 2014.