Design Document (Group 9 / Blotto Beats)

1. Purpose:

The system we are designing is a client-server model, with larger emphasis on the client for doing the majority of the work. Storage of the songs will be client based, with the server getting involved if the user chooses to upload their generated song. The client will be doing the work of generation and storing variables.

2. Non-Functional requirements:

As a user, I would like tracks to be created in a trivial amount of time. This is because the client serves no purpose to the user during the song generation time, so the time should be trivial at most.

As a user, I would like the application to be stable over long periods of time. This is because the application is designed to be able to run continuously in the background over larger periods of time, and stability is necessary for the client to perform this basic function.

As a user, I would like there to be sufficient variance within the random tracks. This is because music serves the purpose of being interesting both from a compositional and listener standpoint when there is variation between songs.

As a developer, I want the server to be lightweight on resources. This is because we will have reasonably limited server resources, and there will be large quantities of data to be handled.

As a developer, I would like the addition of new generator parameters to have trivial modification time. This is because song variance and interest is increased with the number of potential parameters, but not all parameters are possible to insert at one single point in time. For example, different genres have drastically different compositional structures that will be added at separate times.

As a user, I would like the server to be able to handle the input of thousands of track data sets. This is because the program will be able to generate massive quantities of different tracks, and the server should be able to handle the voting information on as many of these tracks as possible to provide the info to the user.

As a developer, I would like to be able to incorporate the generator algorithm into other projects easily. This is because the usage of music is very common in a wide variety of programs, and this procedural generator could be used in other projects for music composition.

3. Design Outline:

We're choosing the client-server model. We will be implementing a fat client, which will do the majority of the work, including letting the user input variables, generating a song for the user, saving the variables with the song. The client will store the user's personal

voting records also. The server will store songs that the chooses to upload, as well as storing the voting information for tracks, and the variables for the track that they are connected to. The voting system will just be a score attached to each song.

b. Describe the interactions between individual system components (2 points).
The generator will generate the song when the play button is pressed in the client. The song will play when the client gets a message from the generate function that the song has been generated. The song will be locally downloaded. When the user presses the upvote or downvote button, the client will send a message to the server, if the song is in the database, then it will update the votes, if not, then the song will be added to the database.

4. Design Issues:

Vote Storage: One issue we had was how to store the upvotes and downvotes. We could store them on the server or on the client. We could store the upvotes and downvotes separately, or we could just conglomerate it into one score. Finally, we could store the data on who has already voted on the client, or the server.

We chose to store upvotes and downvotes on the server as that was where the songs that are shared get stored. We stored it as one score in order to make handling the data on the server easier. Finally, we stored the data on who has voted for a song on the clients, as it is simpler and it saves space on the server.

Song Indexing: We had several options for storing and indexing the songs on the server. To sort them, we could leave them unsorted, sort them by genre, sort them by one or more of the parameters, or some combination of that. We could index the songs by seeds, we could assign them a unique ID, or we could create some kind of hash.

We chose to sort the songs by genre, as that was a simple way to divide up the songs in a logical manner. We chose to index the songs by the seed, as there would be very few collisions between songs sharing the same seed. Any songs that do will be further indexed by their parameters.

5. Design Details:

Our system will consist mostly of three classes. We will have the client, which will be the main window that the user will be using to interact with the system; the generator, which will be where we will be generating the music; and the server, which is where we will be storing the upvote/downvote score for all of the songs. Most of the user interaction will be based in the client. If they press a button to, say, open the settings window, then the client will be able to handle that within itself. However, whenever the client presses the play button (or autoplay is turned on and the previous song has ended) then the client will interact with the generator to get

back a song to play. The client will also interact with the server if they press the upvote or downvote buttons, in that, the server will check to see if the currently playing song is in the database and if it is, then the server will update the score and if not then the server will add the song to the database.