Program Requirements:

The program should be able to produce a musical phrase. A musical phrase is defined as one that encapsulates the antecedent-consequent model (question-answer) model, concluded by a cadence. In order to create musical phrases, the program should make use of grammar rules defined in the GTTM.

The program should promote further enhancements and refinements. Ultimately it should provide for the potential for larger pieces of music to be created using the musical grammar rules. It should also provide the potential for adding in more rules for consultation when composing a piece of music. For example, compositional rules might need to be added at a later date, and this should be accommodated for within the program.

The program should compose music in a way that reduces randomness. This will be achieved by making use of musical grammar rules which constrain how the program goes about composing a piece of music.

Thoughts on how to compose a piece of music using the GTTM grammar rules:

There are four important aspects of the GTTM: grouping structure, metrical structure, time-span reduction, and prolongational reduction.

Grouping structure is used to analyse a piece of music as being a series of groups, and these groups conjoin hierarchically into larger groups. The smallest groups might be a simple rhythmic motives which are grouped together sequentially and conjoined to form larger groups which might be considered to constitute a phrase. This is an important structural aspect of a piece.

Metrical structure is used to analyse the rhythm of the piece in terms of its meter - (where the beat falls). It's less clear to me how I can use this to create rules of composition, but it's important for time-span reduction and, by extension, prolongational reduction.

Time-spans are 'apprehended rhythmic units in terms of which pitch structure is heard'. A piece is exhaustively segmented into time-spans at every level. Time-span reduction works to analyse the most-important pitch event/events (in the case of a cadence, for example) within a time-span. Thus time-span reduction may be used to provide elaborations and embellishments on a simple structure. E.g. the cadential chords may be identified, and then the beginning of the phrase, and further notes may be 'filled in'.

Prolongational reduction analyses the piece in terms of tension and relaxation. Normative form is one where a piece of music begins by increasing in tension before culminating in a relaxing motion. Phrases are in normative form.

General Ideas:

One prospective approach would be to get the program, given a specific grammatical structure, to fill in the appropriate pitches from the structure. The program could then also work to generate its own grammars, which it would then fill in.

What would these input grammars look like? They would involve providing the structure of the piece in terms of prolongational reduction, time-span reduction, metrical structure and grouping

structure. For musical phrases, the prolongational reduction would be in normative form, indicating what pitches within the piece should be doing in terms of tension or relaxation. (How complex should these generated grammars be, and how much flexibility should the program have to modify/extend them?) Grouping structure could be use to guide the low level construction of bits of the music. Preference rules could be used in reverse to decide what pitches/accents/ etc would fit the grouping grammar. (e.g. groups within hierarchies would need to consider how certain choices of pitch would fit within this broader picture. Furthermore, considerations of the prolongational structure would need to be taken into account in choosing appropriate pitches.)

So we have two elements of the GTTM we want to focus on: the high level prolongational structure, and the low-level grouping structure. Inbetween these two components are the metrical grouping structure, the grammar of which will need to be constructed in accordance with the grouping structure, and we have the time-span reduction which is important for generating the prolongational structure, and is based on the grouping and metrical structures. Time-span segementation will also need to be implemented.

In the chain of construction, it might looks something like this: Prolongational structure is generated, which is then used to generate a corroborating time-span reduction structure. The time-span reduction structure is then used to generate a corroborating grouping structure and metrical structure, which is then filled in by the program.

Further details:

Music will need to be constructed using the tonal music idiom. This will involve constructing code pieces that are able to generate diatonically related notes given a specific key. This will further involve the need to produce code which is able to generate things such as cadences, harmonies, etc. Melody is generated through the GTTM.

Some way of converting the generated music into an audible piece of music will need to be constructed. I expect there will be Java compatible solutions to allow the production of audible versions of the generated music. At the very least the music should be constructed in a universally recognisable way (i.e. displayed as a conventional musical score).

Low level details:

How will the elements of the program be structured?

1. Pitches:

Pitches can be modelled as individual objects. Key attributes are: frequency, duration, pitch class. Further things that need to be modelled are accent, slurring, and broader considerations such as tempo, general volume, etc.

2. Phrases:

Pitches will need to be contained, ultimately, in groups which make up a phrase. Key attributes of the phrase: notes and their order, time-signature?, accents.

3. Groups:

The GTTM requires that music be analysed in terms of rhythmic groups, and these should probably be modelled at objects which contain notes and other groups. Key attributes: notes, groups, length,

level.

4. Grammar Generator:

The program needs to be able to generate its own grammar, which provides the skeleton structure for a piece of music. Grammars can be specific or flexible (e.g., they might give an entire or near entire grammatical structure, or just give the bare bones which needs to be completed by the music generator). Specific structures will probably be generated from analysing other pieces of music.

5. Music generator:

The program needs to be able to generate music given a grammar as an input

6. Extensibility:

The program should work such that new modules can be 'plugged in' which provide more sophisticated functionality. Ideally, the functioning of the program should not be set in stone increased complexity should be possible. For example, at this stage the program is only being constructed as a musical phrase generator, but the program should be designed with the idea in mind that larger scale pieces of music should be constructed. For instance, phrases may be constructed and then input into another module to create larger scale pieces of music (this implies that pieces of music should be modifiable and not set in stone once generated, or else it implies that broader grammars should be producible and that the existing modules that are used to construct a phrase can have the potential for filling in a larger scale grammar).

7. A clear grammar:

A clear grammar should be developed which can be input into a system without too much difficulty and obscurity. It should be heavily based upon the GTTM, and modifications, simplifications, and sophistications of the GTTM should be clearly marked out and justified.

8. Capacity for grammar to be expanded and refined:

The program will depend upon a general grammatical structure in order for it to function, but modules within the program which codify how the grammar functions will be able to modified, replaced, and updated without having to change any other aspect of the code. In this way, the broader program can be seen to 'consult' these modules, and only the interfaces of these modules should remain the same. This implies that a constructed grammar that is being filled in by the music generator will be consulting these modules in order to identify potential 'solutions' to the grammar 'problems'.

This further implies that the constructed grammars should be seen to be stable constructions, and once I have codified their form, they should in general not need to be altered. They simply provide a proposed musical structure in terms of grouping, time-spans, metricity, and prolongational structure, and are general in form. Only when it comes to filling in the grammars will the modifiable modules need to be consulted.

9. Potential for analysing the grammar of another piece of music and providing its generalised form.

The grammar modules described above should be able to be consulted such that they are used to analyse other pieces of music. The ability to analyse other pieces of music will not be implemented,

but should be forseen as a potential adaptation.

Important things to do now are:

Think about how a generalised grammar should be formed.
Think about how it could be 'filled in'.

Think about overall structure of the program in terms of modules.

Generalised Grammar:

Provide structure for musical piece (whether it be a phrase, two phrases, a whole piece, or etc) which is used in the generation of a piece of music. Piece is given a metrical structure and grouping structure which is dependent on the time-span reduction structure, as well as the prolongational structure.

Music generator should be able to analyse the given grammar and produce a piece of music that matches the grammatical structure. It will do this by consulting modules that implement preference and well-formedness rules of grammatical structure. It will also refer to a note selection module which provides potential pitches that are diatonically contained within a given key (might also provide modulating pitches, sets of pitches, etc).

Potential for musical generator module to reference other modules in 'composing' a piece. Composition rules, constraints, etc? The program structure should accommodate this potential. Perhaps begin by building up a set of interfaces.