**2. Requirements Analysis** – **Data and Operations**

* **Motifs**

A motif is the smallest musical element. It is a short pattern of rhythmic and/or tonal elements.

**Operations**

* + - select\_random\_motif(): Chooses and returns a random motif from the motif table.
    - create\_motif(): Allows developers (maybe users) to add a motif to the database.
    - select\_motif(style): Returns a motif that matches the style argument.
    - generate\_random\_motif(): Creates a random motif and adds it to the database.
* **Notes**

This is a collection of all possible musical notes within the range of the standard, 88 key piano. This includes all accidentals.

**Operations**

* add\_to\_motif(note): Adds the argument note to the set of notes in an existing motif.
* remove\_from\_motif(note): Removes the specified note from an existing motif.
* get\_notes\_for\_scale(scale): Returns a list of all notes in the specified scale.
* up\_octave(note): Returns the given note, raised one octave.
* down\_octive(note): Returns the given note, lowered one octave.
* stepwise\_note(note): Returns a note for a stepwise sequence.
  + **Tempo**

This is a collection of all available tempos, along with ranges for the classical names (Allegro, etc.). We will allow 40 bpm to 240 bpm.

**Operations**

* + increase\_tempo(bpm): Increments the current tempo by bpm.
  + decrease\_temp(bpm): Decrements the current tempo by bpm.
  + set\_tempo(bpm): Sets the tempo to bpm.
* **Instruments**

Collection of all instruments we’ll allow, along with their range, and any associated MIDI information.

**Operations**

* add\_instrument(inst): Adds the specified instrument to a composition
* remove\_intrument(inst): Removes the specified instrument to a composition.
* chorus(instr): Creates a chorus effect for the specified instrument.
* solo(instr): Creates a solo instrument sound.
  + - * **Rhythms**

Holds rhythmic patterns that can be used for motifs or styles. These will typically be just a couple measures long or shorter. Durations will be specified by w (whole note), h (half), q (quarter), e (eighth), s (sixteenth).

**Operations**

* get\_random\_rhythm(measures, time\_sig): Returns a random rhythm that is of length *measures* and fits the specified time signature.
* add\_rhythm(rhythm): Adds specified rhythm to the database.
  + - * **Scales**

Holds the notes and steps associated with the common scales in western music. Notes and step values (whole or half) need to be stored.

**Operations**

* + - * create\_scale(scale): Adds user generated scale to the database.
      * generate\_motif(scale): Generates a pseudo random motif that fits the scale.

**Progressions**

Stores standard classical chord progressions for major and minor modes. Progressions will be represented as a finite state machine: a predominant moves to a dominant, a dominant moves to a tonic or third, etc.

**Operations**

move(state): returns a list of available chords to move to from the current chord (*state).*

* + - * **Chords**

Represents the possible combinations of notes to form the most common chords: major and minor triads, seventh chords, etc.

**Operations**

* + - * augment(chord): Returns an augmented version of the specified chord.
      * diminish(chord): Returns a diminished version of the specified chord.
      * harmonization(chord): Returns possible melody notes that harmonize the chord.
      * **Styles**

Representations of different styles of music. This includes typical rhythmic patterns, tempos, scales, chords, instruments, etc. For example, if a user wants a MIDI file in a “blues” style, this would mean guitar drums and base, 12-bar blues organization, I-IV-V progressions, and blues scales.

**Operations**

* + - * add\_style(style): Allows for adding user generated styles to the database.
      * **Time Signatures**

Stores all available time signatures.

**Operations**

* + - * double\_time(time\_sig): Changes the current time signature to double time.
      * half\_time(time\_sig): Changes the current time signature to half time.

.