Joshua Choi - Language Guru
jc4881@columbia.edu

Alice Yuanjia Zhang - Tester
ayz2105@columbia.edu

Emily Li - Systems Architect
el2895@columbia.edu

Natalia Dorogi - Project Manager
ngd2111@columbia.edu

# Improv Programming Language Proposal

## Introduction

Improvisation, made simple, is founded upon arranging notes from the pentatonic scale of a song's key over its instrumental. We are building the Improv language to synthesize the music file of an improvised solo based on the user's inputs and specifications.

The user first sets a key, e.g. C major, and a tempo in BPM (beats per minute), e.g. 86 BPM. The declared key dictates the notes that the user has access to, namely those from the key's pentatonic scale, e.g. C major includes C, D, E, G, and A notes. The user is then able to design melodies with these notes and set to the specified BPM; the use of the pentatonic scale guarantees that the progression of notes harmonizes well over any instrumental with the corresponding key and BPM. Additionally, the user can choose to specify a particular style for the improvised solo, which slightly changes the bank of notes for the key. The default style includes solely the five notes in the pentatonic scale, but for example, if the user wants to create a blues-style solo, Improv inserts the "blues" note, e.g. D#/Eb for C major. Other specifications the user can set include varying note lengths, e.g. eighth, quarter, half notes, and different rhythm patterns, e.g. repeated note lengths.

**Features**

- Strong, static typing
- Static (lexical) scoping
- Automatic memory allocation; garbage collection

**Control Flow**

- Sequencing: statements separated by newlines
- Selection: if...else...end
- Iteration: for...end, while...end, repetition/duplication
- Functions: func...end

**Types**

| Data Type / Structure | Description | Values | Example |
|---|---|---|---|
| key | Primitive type; the key of the solo | CMAJ<br>CMIN<br>C#MAJ<br>C#MIN<br>DBMAJ<br>DBMIN<br>DMAJ<br>DMIN<br>... | key k = C#MAJ |
| bpm | Primitive type; the bpm of the solo | 40-218 | bpm b = 86<br>// defaults to 80 |
| style | Primitive type; the style of the solo | DEFAULT<br>BLUES<br>JAZZ | style s = DEFAULT |
| note | Primitive type; represents a note; | (0-6 wh/hf/qr/ei/sx)<br>*/* 0 : rest;* | note n1 = (1 wh)<br>-- |

| | | | |
|---|---|---|---|
| | includes note.tone and note.rhythm | *1-5 : pentatonic scale;*<br>*6 : optional style note */*<br>*/* wh: whole;*<br>*hf: half;*<br>*qr: quarter;*<br>*ei: eighth;*<br>*sx: sixteenth */* | note n2 = 1<br>*/* = (1 wh); defaults to rhythm wh */*<br>--<br>note n3 = hf<br>*/* = (1 hf); defaults to tone 1 */* |
| int | Primitive type | *integers* | -1, 0, 24 |
| float | Primitive type | *decimal numbers* | -0.5, 1.0, 3.25 |
| bool | Primitive type | true, false | |
| string | Primitive type | *a-zA-Z0-9_‘ ’* | ‘a’, ‘hello’, ‘31’ |
| arr | Data structure; represents an array of the same type; immutable | [] | a = [(1 wh), (2 hf), (1 wh)] |
| map | Data structure; represents a map; mutable | {} | song_map = {<br>‘twinkle’: twinkle1,<br>‘mary’: mary_lamb }<br><br>note_map = {<br>‘a1’: (1 wh),<br>‘a2’: (2 hf) } |

**Operators**

| Operator | Description | Types |
|---|---|---|
| = | Assignment | *all types* |
| $ | Concatenation | note; arr |
| ^x | Duplication, x times | note; arr |
| @ | Binding | note.tone to note.rhythm;<br>note.rhythm to note.tone |
| +, -, *, / | Mathematical operations | |
| ==, <, > | Equivalence, inequality | |

### Sample Code

```
// single-line comment

/*
multi-line comment
*/

func create_improv1(key k, bpm b, style s)
  // notes = {0=>rest, 1=>C, 2=>Eb, 3=>F, 4=>G, 5=>Bb, 6(BLUES ONLY)=>Gb}

  // Define note
  note a_note = (1 wh)

  // Create line of notes
  note[] line1 = [a_note, (2 hf), (5 hf), (3 ei), (1 ei), (4 wh)]
  // Create line of note tones with uniform rhythm
  note[] line2 = hf@[1, 5, 3, 4] // = [(1 hf), (5 hf), (3 hf), (4 hf)]
  // Create line of note rhythms of same tone
  note[] line3 = 3@[wh, hf, ei, sx] = [(3 wh), (3 hf), (3 ei), (3 sx)]

  // Duplicate notes
  note[] line4 = (2 hf)^2 // = [(2 hf), (2 hf)]
  // Duplicate note rhythms and bind to same tone
  note[] line5 = 1@[hf, ei]^2 // = [(1 hf), (1 ei), (1 hf), (1 ei)]

  /*
   * Bind note tones with previously defined note tones or rhythms
   * eg: line3.rhythm returns [wh, hf, ei, sx]
  */
  note[] line6 = [1, 2, 3]@line3.rhythm // = [(1 wh), (2 hf), (3 ei)]
  note[] line7 = [wh, hf]@a_note.tone // = [(1 wh), (1 hf)]

  // Concatenate note lines to create melody
  note[] melody = a_note $ line1 $ line2 $ line3 $ line4 $ line5 $ line6 $
line7
  return melody
end

func create_improv2()
  key k = EBMAJ
  bpm b = 86
  style s = BLUES

  line1 = [(1 wh), (3 hf)]
  // ...
  return line1
end
```

```
func create_improv3()
  // Default to key CMAJ, bpm 86, style DEFAULT

  line1 = [(5 ei), (5 sx)]
  // ...
  return line1
end

func main()
  map<string, note[]> song_map = {}

  improv1 = create_improv1(CMAJ, 86, DEFAULT)
  song_map.add('cool_solo_cmaj': improv1)

  improv1_in_cmin = create_improv1(CMIN, 86, DEFAULT)
  song_map.add('cool_solo_cmin': improv1_in_cmin)

  improv2 = create_improv2()
  song_map.add('bob_solo': improv2)

  improv3 = create_improv3()
  song_map.add('anna_solo': improv3)

  for name, song in song_map
    render_wav(song, name)
  end
end
```