Design

In this documentation, the design of the programming language is presented. We will delve into the features and commands of BitBeats, explaining their required arguments. We will also outline the envisioned structure of a BitBeats program, and finally, provide example programs.

Commands

Below are the key commands of BitBeats along with their arguments and a respective example code line:

`start`

Starts the beat playback.

`stop`

Stops the beat playback.

`tempo`

Sets the tempo for beat playback in beats per minute (BPM).

Argument	Description
`bpm`	Tempo in beats per minute.

Table 1: Arguments for `tempo`

Example code for `tempo`:

```
tempo 120
```

`master_vol`

Sets the overall volume of beat playback.

Argument	Description
`vol`	Volume from 0 to 1.

Table 2: Arguments for `master_vol`

Example code for `master_vol`:

```
master_vol 0.7
```

Argument	Description
`vol`	Volume of channel playback from 0 to 1.
`binary_pattern`	Binary pattern representing the temporal playback of the sound.
`filter`	Filter type. Here, `hp` is a high-pass filter, `lp` is a low-pass filter, and `0` is no filter selection.
`cut_off`	Cutoff frequency of the filter in Hz.
`A D S R`	Attack, Decay, Sustain, and Release in milliseconds.

Table 3: Arguments for `play noise`

`play noise`

Configures sound generation and temporal playback of the noise channel.

Example code for `play noise`:

```
play noise 0.4 10101011 hp 3000 10 50 3 50
```

Explanation: `binary_pattern`

The `binary_pattern` is a crucial argument in BitBeats. It represents the temporal pattern of sound output and is depicted by a string of zeros and ones. Each digit in this pattern represents a specific time when the sound is either active (1) or inactive (0). Since we have defined a 4/4 time signature for the beats, up to 8 numbers are included in this pattern. This means each of these numbers represents an eighth note in the bar. If fewer than 8 numbers (states) are provided, they are filled with zeros, so no note is played at the end of the bar. If more than 8 numbers are provided, an error occurs.

Explanation: `ADSR`

The ADSR envelope allows control of various parameters for sound shaping [Kvi89]. The envelope is also shown in Figure 1. The 4 parameters in our case have the following meanings:

1. Attack: The time it takes for the sound to rise from silence to the set volume.
2. Decay: The time it takes for the sound to decrease from maximum volume to the sustain level.
3. Sustain: The time the sound holds at 50% of the set volume.
4. Release: The time it takes for the sound to fall from the sustain volume to silence after the sound source has ended.

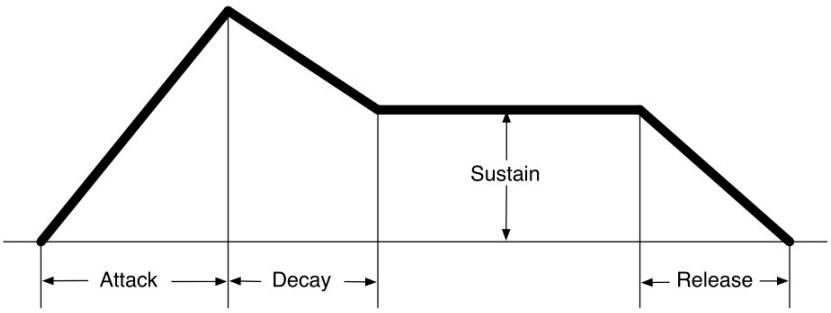

Figure 1: Elements of the ADSR envelope [Kvi89, S.171]

`play channel`

Configures sound generation and temporal playback of one or more waveform channels. When entering multiple channels, separate them with a comma.

Argument	**Description**
`channel`	The channel on which the note should be played. Here, `square1` is the first square wave channel, `square2` is the second square wave channel, and `triangle` is the triangle wave channel.
`vol`	Volume of channel playback from 0 to 1.
`binary_pattern`	Binary pattern representing the temporal playback of the sound.
`note`	Pitch in English notation from c1 to c8, as indicated in Table 10.
`length`	Length of the played note in eighth notes from 1 to 4.
`duty_cycle`	Duty cycle of the square wave signal from 0 to 1. When using the triangle wave channel, set this to 0.

Table 4: Arguments for `play channel`

Example code for `play channel`:

```
play channel square1 0.8 10101011 c4 2 0.5
```

0.0.1 Explanation: `duty_cycle`

The `duty_cycle` is a parameter that affects the sound of the square wave channels. It represents the ratio between the time the wave is high and the time it is low in a single period. A duty cycle of 0.5 results in a square wave with a

50% duty cycle, and this produces a balanced sound. Other values can be used to modify the sound characteristics, as shown in Figure 2.

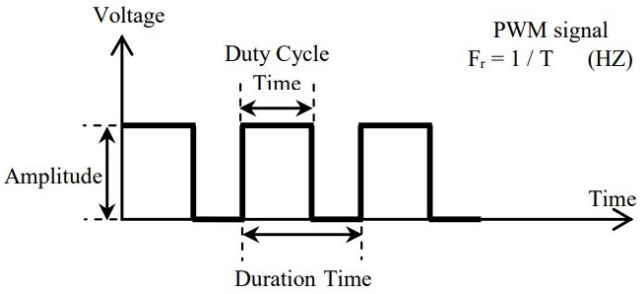

Figure 2: Effect of duty cycle on square wave sound [MEH13, S.3]

`set_effect`

Sets an arpeggiating effect for a specific channel.

Argument	Description
`channel`	The channel for which the effect should be set, either `square1`, `square2`, or `triangle`.
`cycle_steps`	Number of subdivisions within an 8th note.
`intervals`	The intervals to be played upwards from the root note in sequence. Interval abbreviations are shown in Table 11.

Table 5: Arguments for `set_effect`

Example code for `set_effect`:

```
set_effect square1 3 P1M3P4
```

`stop_effect`

Stops the effect for a specific channel.

Argument	Description
`channel`	The channel for which the effect should be stopped, either `square1`, `square2`, or `triangle`.

Table 6: Arguments for `stop_effect`

Example code for `stop_effect`:

```
stop_effect square1
```

pause_channels

Pauses one or more channels.

Argument	Description
`channels`	The channels to be paused, either `square1`, `square2`, or `triangle`.

Table 7: Arguments for `pause_channels`

Example code for `pause_channels`:

```
pause_channels noise,square1
```

run_script

Executes a saved BitBeats script in the form of a text file.

Argument	Description
`script`	Text file name where the script is saved. The file must be in the same folder as the program.

Table 8: Arguments for `run_script`

Example code for `run_script`:

```
run_script script.txt
```

wait

Designed for use in a BitBeats script. This command plays the configured channels for a specific number of bars in a BitBeats script.

Argument	Description
`bars_nb`	The number of bars for which the configured channels in a BitBeats script should be played.

Table 9: Arguments for `wait`

Example code for `wait`:

```
wait 2
```

Variables

BitBeats also allows the storage of configured channels as variables to improve code readability and reusability.

`sq1` is initialized as an example with the command `square1 10001101 0.7 d3 0.2 1`:

```
sq1 = square1 0.7 10001101 d3 1 0.2
```

`snare` is initialized with the command `noise 11101110 1 hp 4000 15 5 100 150`:

```
snare = noise 1 11101110 hp 4000 15 5 100 150
```

The order of the arguments corresponds to the commands of `play_channel` and `play_noise`. To reproduce the previously defined variables `sq1` and `snare`, the `play` command can be used:

```
play sq1, snare
```

Example Code

While commands can be used individually in BitBeats to create and modify beats live, the `run_script` command also allows the playback of prepared scripts, ensuring better temporal precision. Here are possible examples of scripts for using the `run_script` command:

```
tempo 120
master_vol 0.2
#
sq1_d = square1 0.7 10001101 d3 1 0.2
sq2_d = square2 0.7 10001101 f3 1 0.8
tri_d = triangle 0.7 10001101 a3 1 0
#
sq1_v = square1 0.7 10001101 c#3 1 0
sq2_v = square2 0.7 10001101 e3 1 0
tri_v = triangle 0.7 10001101 g3 1 0
#
sq2_m = square2 0.7 10101010 d3 2 0
#
snare = noise 0.7 11101110 hp 4000 15 5 100 150
kick = noise 0.7 11101110 lp 500 15 5 50 400
#
play snare
start
wait 1
play sq1_d,sq2_d,tri_d
wait 2
```

```
play sq1_v,sq2_v,tri_v
wait 2
play sq1_d,sq2_d,tri_d
wait 2
play sq2_m,kick
set_effect square2 3 m3P4
wait 2
stop
```

This is a typical structure of a BitBeats script. At the beginning, basic parameters such as tempo and overall volume are set with the `tempo` and `master_vol` commands.

Then, various channels are configured and saved as variables. In this script, three different channels are first defined, harmonically tuned, forming a C major triad: `sq1_c`, `sq2_c`, and `tri_c`. Each channel is configured with specific parameters such as binary patterns, volume, pitch, and other properties. Then, the three channels are saved with different parameters.

The noise channel is configured with `snare` and `bass` twice to increase the variety of tones in the beat.

After configuring the channels, playback is done with the `play` command, which plays the previously defined channels. With `start`, beat playback is initiated, and with `wait 2`, it waits for 2 bars, allowing the set channels to play for 2 bars.

Effects, like the arpeggiating effect on `square2`, are set, and after two more bars, beat playback is stopped with `stop`. Overall, this beat has a length of 9 bars.

The use of comments (introduced by #) is to document the code and explain the different parts of the script.

```
tempo 100
master_vol 0.2
#
sq1_c = square1 0.7 10000000 c2 1 0.5
sq2_a = square2 0.7 01000000 a1 1 0.5
tri_bb = triangle 0.7 00100000 bb2 1 0
#
sq1_f = square1 0.7 10000000 f1 1 0.5
sq2_s = square2 0.7 01000000 d1 1 0.5
tri_es = triangle 0.7 00100000 eb1 1 0
#
snare = noise 1 10001000 hp 4000 15 5 100 150
kick = noise 1 11101110 lp 500 15 5 50 400
#
play sq1_c, sq2_a, tri_bb
set_effect square1 2 P1P8
```

```
set_effect square2 2 P1P8
set_effect triangle 2 P1P8
start
wait 2
play sq1_f, sq2_s, tri_es
wait 2
play snare, sq1_c, sq2_a, tri_bb
wait 2
play sq1_f, sq2_s, tri_es
wait 2
stop
```

This BitBeats script, inspired by the Underworld Theme from Super Mario Bros., demonstrates the ability to reproduce well-known melodies in BitBeats and use creative freedom through effects. In BitBeats, you can't create free melodies within a bar, but by cleverly using the arpeggio effect, you can achieve a certain flexibility in note selection.

English Notation	**German Notation**	**Frequency in Hz**
C8	c”’’	4186.01
C7	c””	2217.46
C6	c”’	1046.50
C5	c”	523.25
C4	c’	261.63
C3	c	130.81
C2	C	61.74
C1	C,	27.50

Table 10: Examples of note values in different octaves

Abbreviation	Interval	Semitones
P1	Perfect Unison	0
m2	Minor Second	1
M2	Major Second	2
m3	Minor Third	3
M3	Major Third	4
P4	Perfect Fourth	5
A4	Augmented Fourth	6
d5	Diminished Fifth	6
P5	Perfect Fifth	7
m6	Minor Sixth	8
M6	Major Sixth	9
m7	Minor Seventh	10
M7	Major Seventh	11
P8	Perfect Octave	12

Table 11: Abbreviations of intervals with spelled-out intervals and semitones

References

[Kvi89] T. Kvifte, *Instruments and the Electronic Age: Toward a Terminology for a Unified Description of Playing Technique.* Solum Forlag, 01 1989, vol. 35.

[MEH13] M. Mohamed, A. Elmahalawy, and H. Harb, "Developing The Pulse Width Modulation Tool (PWMT) for Two Timer Mechanism Technique in Microcontrollers," 12 2013.