A brief guide on reamber_base

Eve-ning

23/2/2019

Part I
Common Concepts

In this document, I will be referencing a few terms/abbreviations to save time. I will assume that you understand most of the following terms.

Table 1: Input Abbreviations

Abbr	**Expanded**	**Eg.**
TP	Timing Point	150,200,4,2,0,20,1,0
TP_{SV}	Timing Point (Slider Velocity)	150,-200,4,2,0,20,0,0
TP_{BPM}	Timing Point (Beats Per Minute)	150,200,4,2,0,20,1,0
HO	Hit Object	448,192,150,1,0,0:0:0:0:
HO_{NN}	Hit Object (Normal Note)	448,192,150,1,0,0:0:0:0:
HO_{LN}	Hit Object (Long Note)	448,192,150,128,0,200:0:0:0:0:
EHO	Editor Hit Object	00:00:150 (150|3,200|3) -

Table 2: Timing Point Deconstructed

	Example	
TP_{SV}	100,-200,4,2,1,20,0,1	
TP_{BPM}	100,200,4,2,1,20,1,1	
Label	**Code**	**Remarks**
100	Offset	Unit in ms
-200	Code (TP_{SV})	Actual Value: $-100/Code$
200	Code (TP_{BPM})	Actual Value: $60000/Code$
4	Metronome	
2	Sample Set	*Reference: Sample Sets*
1	Sample Set Index	
20	Volume	Set local volume for note
0	Type	$0\|1 := SV\|BPM$
1	Kiai	$0\|1 := Off\|On$

Table 3: Hit Object Deconstructed

Example		
HO_{NN}	448,192,150,1,0,1:2:3:40:	
HO_{LN}	448,192,150,128,0,200:1:2:3:40:	
Label	**Code**	**Remarks**
448	Column Code/X-Axis	Column: $round((code * keys - 256)/512)$
192	Y-Axis	No Effect
150	Offset	Unit in ms
1	Type	$1\|5\|128 := NN\|NN\|LN$
0	Hitsound	*Reference: Sample Sets*
200	Long Note End	$(OPTIONAL) Only for HO_{LN}$
1	Sample Set	*Reference: Sample Sets*
2	Addition Set	*Reference: Sample Sets*
3	Custom Set Index	Set hitsound file index to reference
40	Volume	Set local volume for note

Table 4: Sample Sets

Value	**Sample Set**
0	AUTO
1	NORMAL
2	SOFT
3	DRUM

Part II
Coding Guidelines

1 Aim

The main purpose of this guideline is to make sure that any contributor will follow the same focus

2 Guidelines

1. Keep everything stupid simple
   (a) File names should be as short as possible
   (b) Function names should be too
   (c) Function inputs should always be constant, functions should always return
   (d) More heavy weight functions, less fragmented ones. i.e. overload.
2. Reuse functions as much as possible
3. Function names should make as much sense as possible, if not possible, attach a visual
4. Use folders to group similar functions if needed, however, consider collapsing them into a large function first.

Part III
Objects

3 Singular

The simplest block is the Hit Object and Timing Point, these are derived from a osu_object class. This makes up the basic building block for both types.

4 Multiple

The secondary block would be Hit Object List, and Timing Point List. However, it's important to note that these are not directly derived from hit_object or timing_point, these are derived from a template of osu_object_v.

Most of the details are not covered as they are more-so self explanatory.

Part IV
Functions

This is the part that becomes confusing, however, it's important to define every single nook and cranny of this

5 Stutter

Let's say we have a vector of offsets, we need to ask a few questions before we can generate a Stutter that works.

1. Should we entrust the task of assigning the offsets to the user?

   We should! Split the work so that it's more specific

2. Can we determine if it's valid?

   We can apply a precursory check towards the input to see if the both the value and size is valid

3. Can we omit certain traits if needed?

   We also need to perform precursory checks on if some parameters have overlapped, such as $Location_abs$ and $Location_rel$

4. Lastly, how many functions are needed?

This problem felt like it needed a function that allows varying amounts of arguments like Python's named arguments.

We can either create a function that takes in a struct that holds certain traits, or we can create a class/struct that handles Stutter as a whole. We will go with the latter as the former would be more effective if there are other functions using the traits.

```
struct AMBER_BASE Stutter
{
        double m_initial_val
        double m_threshold_val
        double m_average_val

        double m_threshold_rel
        double m_threshold_abs

}
```