Rhythm Timer Documentation

This extension provides the underlying framework to create a rhythm game. The system handles timing the user's input within a sequence with varying ranks. The system is time-based not FPS based and attempts to compensate for lag when detecting input.

This extension handles the logic of timing and input and does not handle graphical elements. However, there is a separate optional extension called Rhythm Timer Gui Backend that can be paired with this one to provide some extra features to help handle the visual aspect of the game if you do not wish to design it yourself.

Scripts:

rt_system_setstrict(strict)

Description: *Toggles whether "playful presses" are permitted between notes.*
Argument0: *(bool) If true, playful presses are not permitted and will count as a miss*
Returns: *N/A*

rt_system_getstrict()

Description: *Returns whether the system is using strict timing or not.*
Returns: *(bool) If true, strict timing is enabled*

rt_system_hasrank()

Description: *Returns if the user has achieved a rank for a timed input that has not been processed yet.*
Returns: *(bool) If true, there is data in the queue to be processed*

rt_system_getrank()

Description: *Returns the index of the next rank in the system queue then removes the value from the queue. A built-in rank **rt_miss** can also be returned if the player completely missed the note.*
Returns: *(real) The id for the rank, or **rt_norank** if none exists*

rt_system_getremainder()

Description: *Returns the amount of time (in ms) until the next timer is reached. Throws an error if no sequence is playing.*
Returns: *(real) Time left until next timer. Positive values means that there is time left. Negative values mean the user is late, but still has a chance to trigger the note.*

rt_system_settriggerfunction([STRING] function)

Description: *Sets a custom function to be executed when a trigger is activated. This function is **optional** and can be used when more advanced effects are desired.*

Argument0: *(string) The name of the function as a string. The function will be passed 4 arguments when executed:*

Argument0: Rank id
Argument1: Sequence id
Argument2: Trigger id
Argument3: Timer time value

Returns: *(real) The id of the function passed through, or noone if error*

rt_system_setpaused(paused)

Description: *Toggles whether the current playing sequence should be paused or not. If paused, all input will be ignored, and the timer will not progress until the system is unpaused.*

Argument0: *(bool) If true, the system will be paused*

Returns: *(bool) Whether or not the pause was successful*

rt_system_getpaused()

Description: *Returns if the system is currently paused or not. If no sequences are active, this function will always return false.*

Returns: *(bool) If true, the system is paused*

rt_sequence_create()

Description: *Creates a new sequence.*

Returns: *(real) id of the new sequence.*

rt_sequence_destroy(sequence)

Description: *Destroys the specified sequence.*

Argument0: *(real) id of the sequence to destroy.*

Returns: *N/A*

rt_sequence_start(sequence)

Description: *Starts a sequence and activates the timing system. If a sequence is already activated, this function will be ignored.*

Argument0: *(real) id of the sequence to trigger.*

Returns: *(bool) Whether or not the sequence was started successfully.*

rt_sequence_playing()

Description: *Returns the id of the sequence currently playing.*

Returns: *(real) id of the sequence currently playing, or noone if none.*

rt_sequence_isplaying(sequence)

Description: *Returns whether or not a specific sequence is playing.*

Argument0: *(real) id of the sequence to check.*

Returns: *(bool) whether or not the sequence is playing*

rt_sequence_stop([OPTIONAL] sequence)

Description: *Stops all or one sequence.*

Argument0: *(real) [OPTIONAL] If this argument is provided, the sequence will only be stopped if its id matches this argument. If no argument is provided the sequence will be stopped no matter what.*

Returns: *N/A*

rt_trigger_create([CONST / STRING] script, [OPTIONAL] arg1)

Description: *Creates a new trigger to detect user input.*

Argument0: *(real / string) If a real is passed, it will expect one of the following constants:*

- *rt_keyboard: Looks for keyboard input*
- *rt_mouse: Looks for mouse input*

Argument1 will then be required where you specify the keyboard key or mouse button expected for the trigger.

If a string is passed, it will expect a function name. This function will be called to analyse whether a trigger was activated. The function can take either 0 or 1 arguments, where ***Argument1*** *will be passed through if specified. The function itself should register succesfull with* ***rt_return_true()*** *or unsuccessful with* ***rt_return_false()****.*

Argument1: *See above*

Returns: *(real) id of the new trigger, or rt_notrigger if error*

rt_timer_add(sequence, trigger, time)

Description: *Adds a new timer to the sequence that is attached to the specified trigger. Timers are essentially the game's "notes" that the player is looking for.*

Argument0: *(real) id of the sequence to use*

Argument1: *(real) id of the trigger to attach*

Argument2: *(real) time (in ms) to place the trigger in the sequence*

Returns: *(bool) whether the timer was successfully created*

rt_rank_create(time)

Description: *Creates a new "rank" that will be used to measure how close the*

player's timing was in regard to the timer.

Argument0: *(real) max time (in ms) to activate this rank*

Returns: *(real) id of the new rank*

rt_return_true()

Description: *Used in custom trigger scripts to specify that input was received.*

Returns: *N/A*

rt_return_false()

Description: *Used in custom trigger scripts to specify that input was not received.*

Returns: *N/A*

rt_return_get()

Description: *Used in custom trigger scripts to grab the current state of return.*

Returns: *(bool) whether the script is currently marked as successful*

rt_bpmtms(bpm)

Description: *Converts the specified BPM to the number of milliseconds in a single beat. As the system works in milliseconds, this is useful for placing timers in the correct position, given you know the song's BPM.*

Argument0: *(real) the BPM to convert*

Returns: *(real) time (in ms) it takes to cover a single beat*

Constants:

rt_notrigger = -1 -> Returned when retrieving a non-existing trigger
rt_keyboard = 0 -> Specifies to check for keyboard input
rt_mouse = 1 -> Specifies to check for mouse button input
rt_norank = -2 -> Returned when retrieving a non-existing rank
rt_miss = -1 -> A pre-defined rank that is returned if the player missed a note

Objects:

There are no objects provided that require modification. However there is one object with the name ***__rt_ohidden_controller*** that is responsible for stepping through sequences. If you use deactivation in your games, make sure that this object stays active.

Getting Started:

If you are completely lost as to how this extension can be used, here is some extra information as to how things work.

A sequence can be regarded as a "song." All the timing info for gameplay will be stored in a sequence, so you should only need to create one per song per difficulty. A sequence should be started at the same time as the music so that it matches up correctly. Sequences are updated based on the system clock, so you don't have to worry about lag throwing off the inputs or causing the player to miss targets.

A trigger is an object that essentially specifies a type of user input. If you want to have "arrow" buttons that require the user to press a specific arrow key, you would create a trigger for each in this fashion: ***_trigger = rt_trigger_create(rt_keyboard, vk_left);***
If you wanted something more complex, such as having to press an arrow key and the space key you could write your own script. Let's say your script is called ***input_space_scr***. You could instead call ***_trigger = rt_trigger_create("input_space_scr", vk_left);*** Inside the script may look something like this:

```
if (keyboard_check(vk_space) && keyboard_check_pressed(argument0))
    rt_return_true();
else
    rt_return_false();
return undefined;
```

A timer puts all the elements together. You need to specify which sequence it belongs to, when in the sequence to place it, and the trigger that activates it. If you created a timer looking like ***rt_timer_add(g_sequence_id, _trigger, 1000)*** Then it may expect the space and left arrow to be pressed 1 second into the song.

Lastly, you will want to be able to detect how the user has been doing for graphical purposes as well as scoring. There are two ways to do this.

You can use ***rt_system_settriggerfunction*** to specify a function that will be called every time a trigger is activated or missed. All the required data will be passed into the function, which you can see in the function description above.

The other method is to pull the ranks off of the system stack. You can use a combination of ***rt_system_hasrank()*** and ***rt_system_getrank()*** to grab the ranks as the user earns them. Note, this returns the rank id that was provided upon the rank's creation. You will have to use a switch or if statement to determine if it is a good or bad rank.

This extension provides no graphical elements nor a backend for them. The entire graphical system will have to be implemented yourself. If that seems too difficult a task, there is a separate extension called Rhythm Timer Gui Backend that can be paired with this one to provide a graphical backend. The Gui Backend is geared towards the Hatsune Miku Diva series style of gameplay, however, and still requires that you provide the sprites. However it makes setting up the graphics extremely simple.

Contact:

If you have any questions, bug reports, or feature requests please feel free to use the Binskified Entertainment contact form on the Yoyo Games marketplace.