Mod Version: `1.1.4`

Other Languages

Japanese (日本語)

Thanks to kudan for the japanese translation!

Lua Scripting Mod User Manual (EN)

Table Of Contents

Introduction

This mod provides the ability to create and run Lua scripts in game without any compilation process with multiplayer support allowing you to create much more complex machines. Scripts can only control local player machine; they do not affect the level or other players machines in any way.

Big thanks to the authors of UniLua.

Useful Links

- Lua 5.2 Reference Manual.
- Unity – Manual: Order of execution for event functions.
- UnityEngine.GUI.
- Unity – Scripting API: Vector4.
- Unity – Scripting API: Rect.
- Unity – Scripting API: Input.
- Unity – Scripting API: Physics.

What exactly does mod do?

- When you load a machine or create a new one mod will automatically load it's LuaRoot files if it has any. Otherwise, it will create a new LuaRoot inside current machine.
- LuaRoot is a directory where all Lua files and folders are stored. You can access this directoruy by pressing `Open LuaRoot Folder` in mod menu.
- When you run the simulation mod will run the script `main.lua` and if everything started without critical errors mod will save LuaRoot into current machine.
- When you save your machine, LuaRoot will also be included into a save file. You also can save LuaRoot manually by pressing `Save LuaRoot manually`.
- Machine script contains all base unity callbacks. There are short descriptions for all of them inside the default script file or down below.

Quick Summary of the Mod

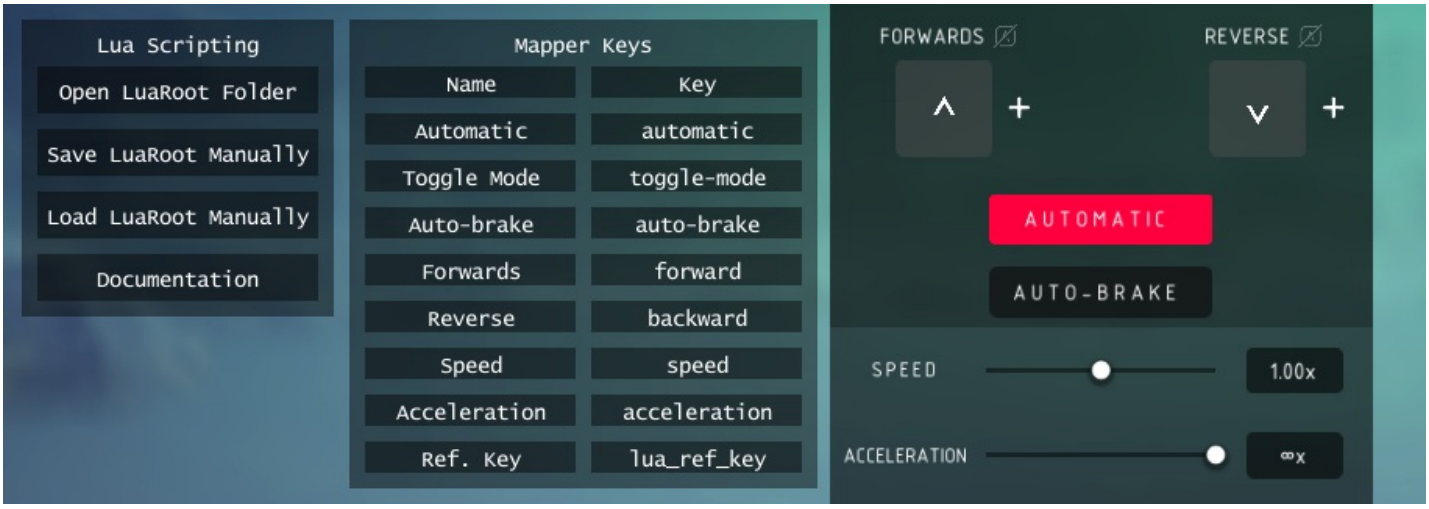
- There are 5 main callbacks: `play`, `update`, `late_update`, `fixed_update` and `on_gui`. Each has its own purpose. See Unity Scripting API about them.
  - `play` called on simulation start. Mostly useless, but can be used for adding chat listeners.
  - `update` called each frame update. Used to get player input.
  - `late_update` called after frame update.
  - `fixed_update` called fixed times per second. Used for physics and so. Put all your cool code here.
  - `on_gui` called only to draw GUI on screen.
- Key emulation is provided by key emulator, it can be started, stopped or just clicked:

```
local some_key_emulator = machine.new_key_emulator('c')
some_key_emulator.start()
some_key_emulator.stop()
some_key_emulator.click()
```

- Slider values can be changed by creating a new `reference controller`. Reference controller controls all blocks with same reference key.

1. Assign reference key to the block. In this case we are using wheels. So, don't forget to enable automatic mode as we are changing wheels speed, not forcing them to spin. Let's call them `rotor`.

2. Find out mapper type key name of the slider you are changing. In our case it is called `speed`.



3. Create a new `reference controller` and set the speed.

```
local rotor = machine.get_refs_control('rotor')
rotor.set_slider('speed', 0.5)
```

- Adjusting steering angles can be done by creating a new `reference controller` (or using existing) as shown above and setting desired angle.

```
local hinge = machine.get_refs_control('hinge')
hinge.set_steering(45)
```

- Block position, rotation, velocity and other information can be obtained by creating a new `block info`.

```
local machine_info = machine.get_machine_info()
local starting_block = machine_info.get_block_info(0)
local position = starting_block.position()
```

It is also possible to get information about other players blocks and machines.

```
for i = 0, players.count() - 1 do
    local player = players.get(i)
    if not player.is_local_player() and player.is_simulating() then
        local enemy_position = players.get(i).get_machine_info().get_block_info(0).position()
        print('enemy at height ' .. enemy_position.y)
    end
end
```

- Raycasting may be done using `physics`.

```
local raycast_start = vector.add(starting_block.position(), vector.multiply(starting_block.up(), -0.5))
local raycast_direction = vector.multiply(starting_block.up(), -1)
local raycast_hit = physics.raycast(raycast_start, raycast_direction)
```

- Near colliders can be found by using sphere overlapping using `physics`.

```
local colliders = physics.overlap_sphere(starting_block.position(), 5)
for i in pairs(colliders) do
    print(colliders[i].is_block)
end
```

- Debug lines can be drawn using `lines` library.

```
local line = lines.new_line_renderer()
line.set_points(vector.new(0, 0, 0), vector.new(10, 10, 10))
```

- Chat messages can be handled by creating a new `chat listener`. Don`t forget that you must add listener only after declaring callback function.

```
local function on_chat(sender, text)
    print(sender .. ' just said ' .. text)
end

chat.add_listener(on_chat)
```

Write your own chat messages with rich text without any nickname prefix.

```
chat.set_visible(true)

chat.write_local('<color=\"red\">Only you can see this!</color>')
chat.write_team('<color=\"red\">Only your team can see this!</color>')
chat.write_global('<color=\"green\">Hello, everyone!</color>')
```

- I`ll be very happy to add new suggested features!

Lua Tables

There is no such thing in this Lua interpreter used in mod as object-oriented programming, but there are tables! Tables allows to store values or even functions, so they are used as objects. However, remember: there are no references, only new tables every time (reference support WIP).

Rectangle

Created by rectangle library using `rect.new(...)`

field	type
x	`int`
y	`int`
width	`int`
height	`int`

Vector

Created by vector library using `vector.new(...)`

field	type
x	`number`
y	`number`
z	`number`
w	`number`

Key Emulator

Created by machine library using `machine.new_key_emulator(...)`.

field	type
start	*function* (no args; `void`)
stop	*function* (no args; `void`)
click	*function* (no args; `void`)
active	*function* (no args; returns `boolean`)

Refs Controller

Created by machine library using `machine.new_refs_control(...)`.

field	type
set_slider	*function* (args: `string` mapper_key, `number` value; `void`)
set_steering	*function* (args: `number` angle; `void`)

Block Info

Created by machine info using `machine_info.get_block_info(...)`.

field	type
position	*function* (no args; returns `vector`)
forward	*function* (no args; returns `vector`)
right	*function* (no args; returns `vector`)
up	*function* (no args; returns `vector`)
rotation	*function* (no args; returns `vector`)
being_vacuumed	*function* (no args; returns `boolean`)
id	*function* (no args; returns `int`)
build_index	*function* (no args; returns `int`)
health	*function* (no args; returns `number`)
burning	*function* (no args; returns `boolean`)
flipped	*function* (no args; returns `boolean`)

field	type
frozen	*function* (no args; returns `boolean`)
in_wind	*function* (no args; returns `boolean`)
destroyed	*function* (no args; returns `boolean`)
zero_g	*function* (no args; returns `boolean`)
original_mass	*function* (no args; returns `number`)
scale	*function* (no args; returns `vector`)
velocity	*function* (no args; returns `vector`)
angular_velocity	*function* (no args; returns `vector`)

Machine Info

Created by machine using `machine.get_machine_info(...)`.

field	type
get_block_info *(local machine only)*	*function* (args: `string` ref_key, `int` index_of_all *(optional)*; returns `block info`)
get_block_info *(both local and another player`s machine)*	*function* (args: `int` build_index; returns `block info`)
block_count	*function* (no args; returns `int`)
cluster_count	*function* (no args; returns `int`)
center	*function* (no args; returns `vector`)
mass	*function* (no args; returns `number`)
middle	*function* (no args; returns `vector`)
name	*function* (no args; returns `string`)
player_id	*function* (no args; returns `int`)
player	*function* (no args; returns `player info`)
position	*function* (no args; returns `vector`)
rotation	*function* (no args; returns `vector`)
velocity	*function* (no args; returns `vector`)
angular_velocity	*function* (no args; returns `vector`)
size	*function* (no args; returns `vector`)
unbreakable	*function* (no args; returns `boolean`)
infinite_ammo	*function* (no args; returns `boolean`)
is_dragging_blocks	*function* (no args; returns `boolean`)
team	*function* (no args; returns `int`)

field	type
is_simulating	*function* (no args; returns `boolean`)

Raycast Hit

Created by physics using `physics.raycast(...)`.

field	type
distance	`number`
point	`vector`
normal	`vector`
is_block	`boolean`
get_block_info	*function* (no args; returns `block info`)

Collider

Created by physics using `physics.overlap_sphere(...)`.

field	type
is_block	`boolean`
get_block_info	*function* (no args; returns `block info`)

Line Renderer (WIP)

Created by lines library using `lines.new_line_renderer()`.

field	type
set_points	*function* (`vector` start, `vector` end; `void`)
set_width	*function* (`number` start_size, `number` end_size; `void`)
set_color	*function* (`vector` color; `void`)

Player Info

Created by players library using `players.get(...)` and other.

field	type
in_local_sim	*function* (no args; returns `boolean`)
is_host	*function* (no args; `boolean`)
is_local_player	*function* (no args; returns `boolean`)
is_spectator	*function* (no args; returns `boolean`)
is_simulating	*function* (no args; returns `boolean`)
name	*function* (no args; returns `string`)
id	*function* (no args; returns `int`)

field	type
team	*function* (no args; returns `id`)
get_machine_info	*function* (no args; returns `machine info`)

Mod libraries

Warning! Every time when a function returns table it creates a **new** table, not returns a reference to a table. It means that you shouldn't create big tables such as `block info` in `fixed_update` loop. Instead, create `block info` at the top of script or in the `play` callback. However, it isn't necessery, just be careful.

Rectangle

`rect`

Used by GUI library.

function	arguments	return values
new	`int` x *(optional)*, `int` y *(optional)*, `int` width *(optional)*, `int` height *(optional)*	`rectangle`

Graphical user interface

`gui`

GUI library based on `UnityEngine.GUI` class except for some functions where arguments were changed. See Unity Scripting API about GUI in Useful Links.

function	arguments	return values
world_to_screen_point	`vector` world_position	`vector`
label	`rectangle` position, `string` text	
button	`rectangle` position, `string` text	`boolean`
toggle	`rectangle` position, `boolean` value, `string` text	`boolean`
begin_group	`rectangle` position	
begin_scroll_view	`rectangle` position, `vector` scroll_position, `rectangle` view_rect	
box	`rectangle` position, `string` text	
bring_window_to_front	`int` window_id	
bring_window_to_back	`int` window_id	
drag_window		
end_group		
end_scroll_view		
focus_control	`string` name	
focus_window	`string` name	
get_name_of_focused_control		`string`
horizontal_scrollbar	`rectangle` position, `number` value, `number` size, `number` left_value, `number` right_value	`number`
horizontal_slider	`rectangle` position, `number` value, `number` left_value, `number` right_value	`number`

function	arguments	return values
modal_window *(arguments were changed)*	`int` id, `rectangle` client_rect, `string` text, *function* (args: `int` window_id; `void`)	`rectangle`
password_field	`rectangle` position, `string` password, `char` mask	`string`
repeat_button	`rectangle` position, `string` text	`boolean`
scroll_to	`rectangle` position	
selection_grid	`rectangle` position, `int` selected, `string` array texts, `int` x_count	`int`
set_next_control_name	`string` name	
text_area	`rectangle` position, `string` text	`string`
text_field	`rectangle` position, `string` text	`string`
unfocus_window		
vertical_scrollbar	`rectangle` position, `number` value, `number` size, `number` top_value, `number` buttom_value	`number`
vertical_slider	`rectangle` position, `number` value, `number` top_value, `number` buttom_value	`number`
window *(arguments are changed)*	`int` window_id, `rectangle` client_rect, `string` title, *function* (args: `int` window_id; `void`)	`rectangle`

Vector

`vector`

Basic Vector4 library based on `UnityEngine.Vector4`. See Unity Scripting API about Vector4 in Useful Links.

function	arguments	return values
new	`number` w *(optional)*, `number` y *(optional)*, `number` z *(optional)*, `number` w *(optional)*	`vector`
distance	`vector` a, `vector` b	`number`
dot	`vector` a, `vector` b	`number`
lerp	`vector` a, `vector` b, `number` t	`vector`
lerp_unclamped	`vector` a, `vector` b, `number` t	`vector`
magnitude	`vector` a	`number`
max	`vector` lhs, `vector` rhs	`vector`
min	`vector` lhs, `vector` rhs	`vector`
move_towards	`vector` current, `vector` target, `number` max_distance_delta	`vector`
normalize	`vector` a	`vector`
project	`vector` a, `vector` b	`vector`
scale	`vector` a, `vector` b	`vector`
add	`vector` a, `vector` b	`vector`
subtract	`vector` a, `vector` b	`vector`
negative	`vector` a	`vector`

function	arguments	return values
multiply	`vector` a, `number` b	`vector`
equals	`vector` a, `vector` b	`boolean`
look_rotation	`vector` a	`vector`
angle	`vector` from, `vector` to	`number`
clamp_magnitude	`vector` a, `number` max_length	`vector`
cross	`vector` a, `vector` b	`vector`
project_on_plane	`vector` point, `vector` normal	`vector`
reflect	`vector` in_direction, `vector` in_normal	`vector`

Machine

`machine`

Machine library for key emulation, steering and sliders controlling, machine info.

function	arguments	return values
new_key_emulator	`string` key_code	`key emulator`)
get_refs_control	`string` ref_key	`refs controller`)
get_machine_info		`machine info`)

Input

`input`

Library for handling direct input from keyboard, mouse, joysticks and other. See Unity Scripting API about Input in Useful Links.

function	arguments	return values
mouse_screen_position		`vector`
mouse_raycast_hit_point		`vector`
get_axis	`string` axis	`number`
get_axis_raw	`string` axis	`number`
get_key	`string` key_code	`boolean`
get_key_down	`string` key_code	`boolean`
get_mouse_button	`int` mouse_button	`boolean`
get_mouse_button_down	`int` mouse_button	`boolean`
get_mouse_button_up	`int` mouse_button	`boolean`
any_key		`boolean`
any_key_down		`boolean`

Cursor

`cursor`

Library for controlling mouse cursor.

function	arguments	return values
set_state	`boolean` state	

Physics

`physics`

Library for obtaining information using physics. See Unity Scripting API about Physics in Useful Links.

function	arguments	return values
raycast	`vector` origin, `vector` direction	`raycast hit`
overlap_sphere	`vector` origin, `number` radius	`collider` array
gravity		`vector`

Players

`players`

Library for getting information about multiplayer session.

function	arguments	return values
count		`int`
get	`int` player_index	`player info`
by_id	`int` network_id	`player info`
get_all		`player info` array

Lines

`lines`

Library for drawing 3D debug lines (only for local client).

function	arguments	return values
new_line_renderer		`line renderer`

Screen

`screen`

Library for getting screen information.

function	arguments	return values
width		`number`
height		`number`
fullscreen		`boolean`
dpi		`number`

Chat

`chat`

Library for handling and writing chat messages.

function	arguments	return values
add_listener	*function* (args: `string` sender, `string` text; `void`)	
set_visible	`boolean` state	
write_local	`string` text	
write_team	`string` text	
write_global	`string` text	
clear		

Thanks for reading!