Tululoo HTML5 Game Maker

Tululoo Documentation

This manual is created to provide simple language reference on Tululoo Game Maker 1.2.0.

Program by Zoltan Percsich.

Manual by YellowAfterlife.

Tululoo Game Maker is a complex game creation environment that allows you to create browser based games. The program generates a Javascript code that can be played in web browsers using the next generation web standard HTML5.

With Tululoo you can create games for your mobile devices as the latest ones already support the new HTML5 standards. Tululoo comes with a user friendy resource manager and script editor, where you can import your images, backgrounds, animated sprites, sounds. You can easily create the objects (actors like player, enemies, collectable items, etc.) and you can also add your own functions and scripts to the game objects.

The built-in script editor has some great features like syntax highlighting or code completion so writing your own code is easy and fun. Tululoo offers several built-in game functions as well, but it is dynamic, so you can use unlimited custom functions, variables, structures etc. Beginners can browse several pre-made game kits, which can be modified easily.

Reference

Tululoo uses JavaScript as programming language, extending it with additional functions, which are meant to simplify game & application development.

For now, it does not contain basic syntax documentation, thus it would be useful to follow a few tutorials on JavaScript before actively programming in Tululoo.

If you have used either GML, Lite-C, or ActionScript, learning should be slightly easier.

Computations

The following routines exist for computing values: choose(arg1, arg2, ..., argN) - randomly returns one of providen arguments.

degtorad(degree) - converts degrees to radians.

radtodeg(radian) - converts radians to degrees.

abs(x) - returns absolute value of x.

sign(x) - returns sign of x (1, 0, or -1).

min(arg1, arg2, ..., argN) - returns the smallest of providen arguments (shortcut to Math.min).

max(arg1, arg2, ..., argN) - returns the largest of providen arguments (shortcut to Math.max).

random(value) - returns floating point integer between 0 (inclusive) and value (exclusive).

irandom(value) - returns random integer between 1 (inclusive) and value (inclusive).

point_direction(x1, y1, x2, y2) - returns direction between two points.

point_distance(x1, y1, x2, y2) - returns distance between two points.

lengthdir_x(length, direction) - returns the horizontal x-component of the vector determined by the indicated length and direction.

lengthdir_y(length, direction) - returns the horizontal x-component of the vector determined by the indicated length and direction.

You may also use the following functions from standard JS Math object (or by including 'Math functions' extension): Math.abs(x) - returns absolute value of x.

Math.min(arg1, arg2, ..., argN) - returns the smallest of providen arguments.

Math.max(arg1, arg2, ..., argN) - returns the largest of providen arguments.

Math.random() - returns a random number between 0 and 1.

Math.round(x) - returns x, rounded to the nearest integer.

Math.floor(x) - returns x, rounded downwards to the nearest integer.

Math.ceil(x) - returns x, rounded downwards to the nearest integer.

Math.sqrt(x) - returns the square root of x.

Math.pow(x, y) - returns x to the power of y.

Math.exp(x) - returns E to the power of x.

Math.log(x) - returns natural logarithm (base E) of x.

Math.sin(x) - returns the sine of x (x is in radians).

Math.cos(x) - returns the cosine of x (x is in radians).

Math.tan(x) - returns the tangent of x (x is in radians).

Math.acos(x) - returns the arccosine of x, in radians.

Math.asin(x) - returns the arcsine of x, in radians.

Math.atan(x) - returns the arctangent of x, in radians.

Math.atan2(y, x) - returns the arctangent of the quotient of its arguments.

User Interaction

The following routines exist for tracking keyboard input: keyboard_check(key) - returns if specified key is down

keyboard_check_pressed(key) - returns if specified key was pressed

keyboard_check_released(key) - returns if specified key was released

The following functions and variables exist for tracking mouse (and pointer) input: mouse_x - indicates current mouse X position in stage.

mouse_y - indicates current mouse Y position in stage.

mouse_check() - returns if left mouse button is down.

mouse_check_pressed() - returns if left mouse button was pressed.

mouse_check_released() - returns if left mouse button was released.

Since version 1.2.0, you may also access the following variables (as alternatives to functions above): key_down[key] - indicates if specified key is down

key_pressed[key] - indicates if specified key was pressed

key_released[key] - indicates if specified key was released

mouse_down - indicates if left mouse button is down.

mouse_pressed - indicates if left mouse button was pressed.

mouse_released - indicates if left mouse button was released.

The following routines exist for misc. operations over user input: hide_mouse() - turns off mouse cursor visibility.

show_mouse() - tuns on mouse cursor visibility.

pause_game(key) - suspends game execution until specified key is pressed. To mention, game will be paused after completition of currently executing code block, so do not use it as delay function in the middle of the code.

User Interaction (mobile devices)

As you may know, HTML5 games can be played on mobile devices (if browser of ones supports HTML5). Such games can be also created with Tululoo.

When played on mobile device, mouse position & state is updated automatically - cursor position is calculated as a average between all 'touch' points.

To create 'onscreen' keys, the following routine exists:

vkey_add(x, y, width, height, index) - adds a virtual key with specified parameters. Parameter index indicates key index to be simulated when virtual key is pressed (i.e. vk_enter).

Function returns a virtual key object, which has the following parameters: left - virtual keys min X

top - virtual keys min Y

right - virtual keys max X

bottom - virtual keys max Y

key - key index, linked to virtual key

active - indicates if virtual key is active, thus can be pressed

Instances

The following routines exist for manipulating instances: instance_create(x, y, object) - creates a new instance of specified object at coordinates.

instance_number(object) - returns number of specified object type in current room.

instance_list(object) - returns array with all objects of specified type in current room.

instance_destroy() - destroys the calling instance.

move_towards_point(x, y, speed) - moves instance towards specified point by one step.

place_meeting(x, y, object) - returns object of given type (or null) that collides with calling instance at given position.

Objects have the following properties, which can be accessed via (instance).property.

The following properties affect object's position and motion: x - x position

y - y position

xstart - starting x position

ystart - starting y position

xprevious - previous x position

yprevious - previous y position

direction - movement direction.

speed - movement speed.

The following properties affect object's appearance and animation: depth - object depth (affects drawing order)

visible - indicates if object is visible (thus if its Draw event will be executed)

sprite_index - indicates sprite of object. Sprite affects drawing and collision behaviours.

image_index - current frame number.

image_speed - animation speed, in frames per step.

image_single - setting this to value, other than -1, will stop animation and display specified frame.

image_angle - rotation value of graphic.

image_alpha - transparency of graphic (0..1).

The following special properties are adviced to be only readen.

Modifying these may cause mis-behaviours in compiled program, so be careful. id - contains reference to instance.

other - is modified at collision events, indicates colliding instance.

object_index - contains 'type' of instance, which can be used in creation and collision-checking functions.

parent - contains 'parent type' of instance. May be used to look-up inheritance.

Drawing

The following routines exist for drawing objects on screen: draw_line(x1, y1, x2, y2) - draws a line between specified coordinates

draw_rectangle(x1, y1, x2, y2) - draws a rectangle between specified coordinates (top left - bottom right)

draw_circle(x, y, radius, outline) - draws a circle at specified coordinates with chosen radius. Parameter 'outline' indicates if circle should be filled (0) or not (1).

draw_sprite(sprite, frame, x, y) - draws chosen frame of specified sprite at given coordinates.

draw_sprite_ext(sprite, frame, x, y, xscale, yscale, direction, alpha) - draws specified sprite with given parameters of scaling, rotation, and alpha at given coordinates.

The following functions affect settings, used by above functions: draw_set_color(r, g, b) - changes color for subsequent drawing operations (0..255)

draw_set_alpha(alpha) - changes transparency for subsequent drawing operations (0..1)

draw_set_linewidth(width) - changes line width for subsequent line drawing operations.

Text

Text can be drawn with following routines: draw_text(x, y, text) - draws specified text at given coordinates.

As the other drawing functions, it is affected by color and alpha settings.

Text-specific settings can be changed with the following routines: draw_set_font(font) - changes font for subsequent text drawing operations.

draw_set_halign(halign) - changes horizontal font align for subsequent drawing operations.

draw_set_valign(valign) - changes vertical font align for subsequent drawing operations.

Fonts can be defined in the IDE, and have the following accessible properties: bold - indicates if the font is weighted (bold)

italic - indicates if the font is emphased (italic)

size - indicates font size

family - indicates font family ("Arial", "Sans", etc.)

Sprites

Sprite objects can be referenced via their name, drawn, and assigned as sprite_index to instances.

They have the following fields: collision_left - indicates min. x of collision bounds.

collision_top - indicates min. y of collision bounds.

collision_right - indicates max. x of collision bounds.

collision_bottom - indicates max. y of collision bounds.

collision_shape - indicates collision box type (either "Box" or "Circle")

height - indicates sprite height

width - indicates sprite width

xoffset - indicates sprite x offset (origin)

yoffset - indicates sprite y offset (origin)

Audio

The following functions exist for dealing with sound\music playback: sound_play(sound) - plays specified sound.

sound_loop(sound) - loop specified sound.

sound_stop(sound) - stops specified sound.

sound_stop_all(sound) - stops all sounds.

sound_volume(sound, volume) - changes volume for specified sound.

Scenes

The following properties provide basic information about current room: room_current - indicates current room index.

room_width - indicates current room width.

room_height - indicates current room width.

room_speed - indicates target framerate of current room.

fps - indicates current framerate. Notable, as of version 1.2.0RC, this may display a lower framerate, than it actually is, due to few issues in time delta capturing. So do not make a conclusion that game is laggy if fps is few (<15%) frames lower than room_speed.

The following properties affect room viewport: room_viewport_x - indicates x position of viewport in current room.

room_viewport_y - indicates y position of viewport in current room.

room_viewport_width - indicates width of viewport in current room.

room_viewport_height - indicates height of viewport in current room.

room_viewport_object - indicates object index to be followed by viewport in current room.

room_viewport_hborder - indicates horizontal 'border', when viewport is following object in current room.

room_viewport_vborder - indicates vertical 'border', when viewport is following object in current room.

The following properties affect way of drawing rooms background: room_background - indicates background image of current room.

room_background_color_red - indicates background color red component.

room_background_color_green - indicates background color green component.

room_background_color_blue - indicates background color blue component.

room_background_tile_stretch - indicates if background image should be stretched over entire room area.

room_background_tile_x - indicates if background should be tiled horizontally.

room_background_tile_y - indicates if background should be tiled vertically.

The following routines exist for changing current room: room_goto(room) - changes room to specified one.

room_restart() - restarts the current room.

room_goto_next() - changes room to next one.

room_goto_previous() - changes room to previous one.

room_goto_first() - changes room to first one (as specified in editor).

room_goto_last() - changes room to last one (as specified in editor).

Tiles

Since version 1.2.0, Tululoo supports tiles to be added both from room and in realtime.

Tiles are small (normally static) images, which represent parts of specific background.

These can be used for creating complex scenery without slowing the game down or making a separate object for each part of background.

The following routines are created to deal with tiles:

tile_add(background, left, top, width, height, x, y, depth) - adds a new tile and returns Tile object (see below).

tile_delete(tile) - removes specified Tile object from scene.

tile_find(x, y, width, height, depth) - returns an array, containing all tiles that intersect specified region.

Tile object has the following properties: source - background, from which tile image is taken.

x - x position in stage.

y - y position in stage.

left - x position of part to display (in background).

top - x position of part to display (in background).

width - width of part to display (in background).

height - height of part to display (in background).

width2 - 'on-screen' width of tile. Having this different from width property will stretch the tile.

height2 - 'on-screen' height of tile. Having this different from height property will stretch the tile.

Local storage

Local web storage can be used to store information between game\program sessions.

The following routines exist for reading information: load_web_data(key) - loads and returns a variable from local storage

load_web_integer(key) - loads and returns a integer from local storage

load_web_float(key) - loads and returns a floating-point value from local storage

load_web_string(key) - loads and returns a string from local storage

The following routines exist for writing information to local storage: save_web_data(key, value) - saves a variable into local storage.

save_web_integer(key, value) - saves an integer into local storage.

save_web_float(key, value) - saves floating-point value into local storage.

save_web_string(key, value) - saves string into local storage.

The following routines exist for removing existing information from local storage:
delete_web_data(key) - removes variable from local storage.

delete_web_integer(key) - removes integer from local storage.

delete_web_float(key) - removes floating-point value from local storage.

delete_web_string(key) - removes string from local storage.

The following routines exist for performing misc. operations with local storage:
web_data_number() - returns total number of fields saved in local storage.

clear_web_data() - deletes all information from local storage.

'System' functions and variables

Apart from functions and variables that can be found in above topics, Tululoo contains a framework of its own, which as well has its own variables and functions. These might not be needed for 'everyday' use, and are documented here primarily for extensions.

Setters & Getters: var_override(object, property, getfunction, setfunction) - 'overrides' specified variable for object, by adding a setter and getter function. These can be both references, or inline functions. Example usage:

var_override(this, 'rdirection',

function() { return tu_d2r * this.direction; },

function(value) { this.direction = value * tu_r2d; });

All subsequent references to object.rdirection variable will return its current direction, converted to radians.

Changing rdirection would change object's current direction accordingly.

Setters & Getters can be used to simplify variable synchronization in\between objects a lot.

Math: tu_2pi - contains value of 2 Math.PI (full circle in radians).

tu_r2d - multiply a radian angle by this constant to convert it to degrees.

tu_d2r - multiply a degree angle by this constant to convert it to radians.

tu_elapsed - indicates time (in ms) spent from previous game cycle.

Resources: tu_idle - 'dummy' (empty) function, that can be used as default value for function variables.

tu_sprites - array, containing all game sprite resources.

tu_backgrounds - array, containing all game background resources.

tu_fonts - array, containing all game font resources.

tu_scenes - array, containing all game scene resources.

tu_audios - array, containing all game sound/music resources.

Instances: tu_depth, tu_depthi - arrays, containing list of instances sorted by depth value. To get instances on certain depth, use tu_depth[tu_depthi.indexOf(depth)]

tu_types - array, containing lists of instances by their object inheritance. Use tu_types[object_index] to get all instances that are (or are child of) instances of specified type.

Drawing: tu_canvas - contains a pointer to HTML5 canvas, which Tululoo uses for drawing.

tu_context - contains a pointer to 'context' of tu_canvas.

Sound & Music: tu_wav_supported - indicates if WAV sounds are supported by browser.

tu_ogg_supported - indicates if OGG sounds are supported by browser.

tu_mp3_supported - indicates if MP3 sounds are supported by browser.