Origami Runtime - Complete Documentation

Package: `origami-runtime` **Version**: 0.1.0 **Type**: Game Engine Runtime Library

Table of Contents

Introduction

The Origami Runtime is a TypeScript game engine inspired by GameMaker Studio 1.4.9999. It provides a familiar event-driven programming model for creating 2D games that run in web browsers.

Key Features

- **Event-Based Programming** - GameObject classes with create, step, draw events
- **Automatic Systems** - Built-in sprite animation, motion, and collision
- **GMS-Compatible API** - Familiar functions like `instance_create`, `place_meeting`, `draw_sprite`
- **TypeScript Support** - Full type safety with strict mode
- **Canvas Rendering** - Efficient 2D rendering with depth sorting
- **Debug Tools** - Built-in FPS counter, collision visualization

System Requirements

- Node.js 18.0.0 or higher
- TypeScript 5.3.3 or higher
- Modern web browser with Canvas 2D support

Getting Started

Installation

```
npm install origami-runtime
```

Basic Usage

```typescript
import { GameEngine, GameObject } from 'origami-runtime';

// Define a game object
export class obj_player extends GameObject {
  create(): void {
    this.sprite_index = 'spr_player';
    this.x = 100;
    this.y = 100;
  }

  step(): void {
    if (keyboard_check(vk_right)) {
      this.x += 4;
    }
  }

  draw(): void {
    draw_self.call(this);
  }
}

// Initialize the engine
const config = {
  canvas: document.getElementById('game-canvas') as HTMLCanvasElement,
  width: 640,
  height: 480,
  backgroundColor: '#000000',
  startRoom: 'room_start'
};

const engine = new GameEngine(config);
await engine.start();
```

GameObject Reference

Description

The `GameObject` class is the base class for all game objects. All custom objects must extend this class.

Built-in Properties

Position Properties

Property	Type	Description
`x`	number	Current X position in room
`y`	number	Current Y position in room
`xprevious`	number	X position from previous frame
`yprevious`	number	Y position from previous frame
`xstart`	number	Starting X position (set in room)
`ystart`	number	Starting Y position (set in room)

Motion Properties

Property	Type	Description
`speed`	number	Movement speed (pixels per frame)
`direction`	number	Movement direction (0-360 degrees, GMS-style)
`hspeed`	number	Horizontal speed component
`vspeed`	number	Vertical speed component

Note: The engine automatically updates `hspeed` and `vspeed` from `speed` and `direction` each frame, and then updates `x` and `y` from `hspeed` and `vspeed`.

Sprite Properties

Property	Type	Description
`sprite_index`	string | null	Current sprite name
`image_index`	number	Current animation frame (auto-increments)
`image_speed`	number	Animation speed (frames per game frame)
`image_alpha`	number	Transparency (0.0 to 1.0)
`image_angle`	number	Rotation angle in degrees
`image_xscale`	number	Horizontal scale (1.0 = normal)
`image_yscale`	number	Vertical scale (1.0 = normal)

Other Properties

Property	Type	Description
`visible`	boolean	Whether instance is drawn
`depth`	number	Draw order (higher values draw behind)
`order`	number	Step execution order (lower executes first)
`persistent`	boolean	Survives room transitions

Event Methods

All event methods are optional. Override them to define object behavior.

create()

Syntax: `create(): void`

Called once when the instance is created.

Example:

```
create(): void {
  this.sprite_index = 'spr_player';
  this.speed = 4;
  this.health = 100;
}
```

---

step()

Syntax: `step(): void`

Called every frame (60 times per second by default).

Example:

```
step(): void {
  if (keyboard_check(vk_space)) {
    this.vspeed = -10;
  }
  this.vspeed += 0.5; // Gravity
}
```

---

draw()

Syntax: `draw(): void`

Called every frame for rendering. If not defined, automatically calls `draw_self()`.

Example:

```
draw(): void {
  draw_self.call(this);
  draw_text(this.x, this.y - 20, `HP: ${this.health}`);
}
```

---

gameStart()

Syntax: `gameStart(): void`

Called once when the game begins.

---

gameEnd()

Syntax: `gameEnd(): void`

Called when the game ends.

---

roomStart()

Syntax: `roomStart(): void`

Called when entering a room (including the first room).

---

roomEnd()

Syntax: `roomEnd(): void`

Called when leaving a room.

Game Structure

Sprites

Sprites are organized in folders with individual frame images and metadata.

Folder Structure:

```
sprites/
└── spr_player/
    ├── metadata.json
    ├── frame_0.png
    ├── frame_1.png
    └── ...
```

metadata.json:

```
{
  "origin": { "x": 16, "y": 16 },
  "fps": 10,
  "bbox": {
    "left": 4,
    "top": 4,
    "right": 28,
    "bottom": 28
  }
}
```

Properties:

- `origin` - Pivot point for positioning and rotation
- `fps` - Animation framerate
- `bbox` (optional) - Custom collision bounding box

Rooms

Rooms are defined in JSON format.

Example room_level1.json:

```json
{
  "name": "room_level1",
  "width": 1024,
  "height": 768,
  "speed": 60,
  "backgroundColor": "#87CEEB",
  "instances": [
    {
      "object": "obj_player",
      "x": 100,
      "y": 200
    },
    {
      "object": "obj_wall",
      "x": 0,
      "y": 400
    }
  ],
  "views": [{
    "enabled": true,
    "x": 0,
    "y": 0,
    "width": 640,
    "height": 480,
    "portX": 0,
    "portY": 0,
    "portWidth": 640,
    "portHeight": 480,
    "object": "obj_player",
    "hborder": 200,
    "vborder": 150
  }],
  "backgrounds": []
}
```

Functions Reference

Instance Functions

instance_create

Syntax: `instance_create(x, y, objectType)`

Creates a new instance of an object.

Arguments:

- `x` (number) - X position
- `y` (number) - Y position
- `objectType` (string) - Object class name

Returns: `Promise<GameObject>` - The created instance

Description: Creates a new instance of the specified object type at the given position. The instance's `create()` event is called immediately.

Example:

```
// Create a bullet at player position
const bullet = await instance_create(this.x, this.y, 'obj_bullet');
bullet.direction = this.aim_direction;
bullet.speed = 8;
```

---

instance_destroy

Syntax: `instance_destroy.call(this)`

Destroys the calling instance.

Arguments: None

Returns: `void`

Description: Removes the instance from the game. The instance will be deleted at the end of the current frame.

Example:

```
// Destroy when health reaches zero
if (this.health <= 0) {
  instance_destroy.call(this);
}
```

Note: Must be called with `.call(this)` to specify which instance to destroy.

---

instance_exists

Syntax: `instance_exists(objectType)`

Checks if any instance of a type exists.

Arguments:

- `objectType` (string) - Object class name

Returns: `boolean` - True if at least one instance exists

Example:

```
if (instance_exists('obj_player')) {
  console.log('Player is alive');
}
```

---

instance_number

Syntax: `instance_number(objectType)`

Counts instances of a type.

Arguments:

- `objectType` (string) - Object class name

Returns: `number` - Count of instances

Example:

```
const enemyCount = instance_number('obj_enemy');
if (enemyCount === 0) {
  // All enemies defeated
  room_goto('room_victory');
}
```

---

instance_find

Syntax: `instance_find(objectType, n)`

Gets the nth instance of a type.

Arguments:

- `objectType` (string) - Object class name
- `n` (number) - Index (0-based)

Returns: `GameObject | null` - The instance or null if not found

Example:

```
// Get the first player instance
const player = instance_find('obj_player', 0);
if (player) {
  const dist = point_distance(this.x, this.y, player.x, player.y);
}
```

Note: Instance order is not guaranteed. Use for iteration, not specific instances.

Collision Functions

place_meeting

Syntax: `place_meeting.call(this, x, y, objectType)`

Checks if instance would collide at a position.

Arguments:

- `x` (number) - X position to check
- `y` (number) - Y position to check
- `objectType` (string) - Object type to check collision with

Returns: `boolean` - True if collision would occur

Description: Tests if the calling instance's bounding box would intersect with any instance of the specified type if moved to the given position. Uses AABB (axis-aligned bounding box) collision.

Example:

```
// Check wall collision before moving
if (!place_meeting.call(this, this.x + this.hspeed, this.y, 'obj_wall')) {
  this.x += this.hspeed;
} else {
  this.hspeed = 0; // Stop at wall
}
```

Note: Must be called with `.call(this)` to specify which instance is checking.

---

place_free

Syntax: `place_free.call(this, x, y)`

Checks if position is free of all solid objects.

Arguments:

- `x` (number) - X position
- `y` (number) - Y position

Returns: `boolean` - True if position is free

Example:

```
// Apply gravity if in air
if (place_free.call(this, this.x, this.y + 1)) {
  this.vspeed += 0.5;
}
```

---

instance_place

Syntax: `instance_place.call(this, x, y, objectType)`

Gets instance at position (if any).

Arguments:

- `x` (number) - X position
- `y` (number) - Y position
- `objectType` (string) - Object type

Returns: `GameObject | null` - The colliding instance or null

Description: Similar to `place_meeting`, but returns the actual instance instead of just a boolean.

Example:

```
// Collect coins
const coin = instance_place.call(this, this.x, this.y, 'obj_coin');
if (coin) {
  this.score += 10;
  instance_destroy.call(coin);
}
```

---

instance_position

Syntax: `instance_position(x, y, objectType)`

Gets instance at exact point.

Arguments:

- `x` (number) - X coordinate
- `y` (number) - Y coordinate
- `objectType` (string) - Object type

Returns: `GameObject | null` - Instance at that point or null

---

collision_rectangle

Syntax: `collision_rectangle(x1, y1, x2, y2, objectType)`

Checks rectangle collision.

Arguments:

- `x1, y1` (number) - Top-left corner
- `x2, y2` (number) - Bottom-right corner
- `objectType` (string) - Object type

Returns: `GameObject | null` - First colliding instance or null

Example:

```
// Check area for enemies
const enemy = collision_rectangle(0, 0, 100, 100, 'obj_enemy');
```

---

collision_point

Syntax: `collision_point(x, y, objectType)`

Checks point collision.

Arguments:

- `x, y` (number) - Point coordinates
- `objectType` (string) - Object type

Returns: `GameObject | null` - Instance at point or null

Drawing Functions

All drawing functions must be called within the `draw()` event.

draw_self

Syntax: `draw_self.call(this)`

Draws the instance's sprite.

Arguments: None

Returns: `void`

Description: Draws the instance's current sprite with all transformations (position, scale, rotation, alpha).

Example:

```
draw(): void {
  draw_self.call(this);
}
```

---

draw_sprite

Syntax: `draw_sprite(sprite, subimg, x, y)`

Draws a sprite at a position.

Arguments:

- `sprite` (string) - Sprite name
- `subimg` (number) - Frame index
- `x, y` (number) - Position

Returns: `void`

Example:

```
draw_sprite('spr_bullet', 0, this.x, this.y);
```

---

draw_text

Syntax: `draw_text(x, y, text)`

Draws text.

Arguments:

- `x, y` (number) - Position

- `text` (string) - Text to draw

Returns: `void`

Example:

```
draw_text(10, 10, `Score: ${this.score}`);
```

---

draw_rectangle

Syntax: `draw_rectangle(x1, y1, x2, y2, outline)`

Draws a rectangle.

Arguments:

- `x1, y1` (number) - Top-left corner
- `x2, y2` (number) - Bottom-right corner
- `outline` (boolean) - Draw outline only?

Returns: `void`

Example:

```
draw_set_color('#FF0000');
draw_rectangle(10, 10, 100, 50, false); // Filled red rectangle
```

---

draw_circle

Syntax: `draw_circle(x, y, radius, outline)`

Draws a circle.

Arguments:

- `x, y` (number) - Center position
- `radius` (number) - Circle radius
- `outline` (boolean) - Draw outline only?

Returns: `void`

Example:

```
draw_circle(this.x, this.y, 20, true);
```

---

draw_set_color

Syntax: `draw_set_color(color)`

Sets drawing color.

Arguments:

- `color` (string) - Hex color code (e.g., "#FF0000")

Returns: `void`

Example:

```
draw_set_color('#00FF00'); // Green
draw_rectangle(0, 0, 50, 50, false);
```

---

draw_set_alpha

Syntax: `draw_set_alpha(alpha)`

Sets drawing transparency.

Arguments:

- `alpha` (number) - Alpha value (0.0 to 1.0)

Returns: `void`

Example:

```
draw_set_alpha(0.5); // 50% transparent
draw_self.call(this);
draw_set_alpha(1.0); // Reset to opaque
```

Motion Functions

lengthdir_x

Syntax: `lengthdir_x(length, direction)`

Gets X component of a vector.

Arguments:

- `length` (number) - Vector length
- `direction` (number) - Angle in degrees (GMS-style: 0=right, 90=up)

Returns: `number` - X component

Example:

```
// Move in direction of mouse
const dir = point_direction(this.x, this.y, mouse_x, mouse_y);
this.hspeed = lengthdir_x(5, dir);
this.vspeed = lengthdir_y(5, dir);
```

---

lengthdir_y

Syntax: `lengthdir_y(length, direction)`

Gets Y component of a vector.

Arguments:

- `length` (number) - Vector length
- `direction` (number) - Angle in degrees

Returns: `number` - Y component

---

point_direction

Syntax: `point_direction(x1, y1, x2, y2)`

Gets angle between two points.

Arguments:

- `x1, y1` (number) - First point
- `x2, y2` (number) - Second point

Returns: `number` - Angle in degrees (0-360)

Example:

```
// Aim at player
const player = instance_find('obj_player', 0);
if (player) {
  this.direction = point_direction(this.x, this.y, player.x, player.y);
}
```

---

point_distance

Syntax: `point_distance(x1, y1, x2, y2)`

Gets distance between two points.

Arguments:

- `x1, y1` (number) - First point
- `x2, y2` (number) - Second point

Returns: `number` - Distance in pixels

Example:

```
const dist = point_distance(this.x, this.y, target.x, target.y);
if (dist < 100) {
  // Within attack range
}
```

---

move_towards_point

Syntax: `move_towards_point.call(this, x, y, speed)`

Moves instance towards a point.

Arguments:

- `x, y` (number) - Target point
- `speed` (number) - Movement speed

Returns: `void`

Description: Sets the instance's `speed` and `direction` to move towards the target point.

Example:

```
// Chase player
const player = instance_find('obj_player', 0);
if (player) {
  move_towards_point.call(this, player.x, player.y, 3);
}
```

Math Functions

Use native JavaScript `Math` functions for mathematical operations:

- `Math.abs(x)` - Absolute value
- `Math.floor(x)` - Round down
- `Math.ceil(x)` - Round up
- `Math.round(x)` - Round to nearest integer
- `Math.min(a, b)` - Minimum value
- `Math.max(a, b)` - Maximum value
- `Math.sin(rad)` / `Math.cos(rad)` - Trigonometry (use radians)
- `Math.sqrt(x)` - Square root
- `Math.pow(base, exp)` - Power

Note: Trigonometry functions use radians. Convert degrees: `radians = degrees * Math.PI / 180`

Input Functions

keyboard_check

Syntax: `keyboard_check(key)`

Checks if key is held down.

Arguments:

- `key` (number) - Virtual key constant

Returns: `boolean` - True if key is down

Example:

```
if (keyboard_check(vk_right)) {
  this.x += 4;
}
if (keyboard_check(vk_left)) {
  this.x -= 4;
}
```

---

keyboard_check_pressed

Syntax: `keyboard_check_pressed(key)`

Checks if key was just pressed this frame.

Arguments:

- `key` (number) - Virtual key constant

Returns: `boolean` - True if just pressed

Example:

```
if (keyboard_check_pressed(vk_space)) {
  // Jump only on initial press
  this.vspeed = -10;
}
```

---

keyboard_check_released

Syntax: `keyboard_check_released(key)`

Checks if key was just released this frame.

Arguments:

- `key` (number) - Virtual key constant

Returns: `boolean` - True if just released

---

mouse_check_button

Syntax: `mouse_check_button(button)`

Checks if mouse button is held.

Arguments:

- `button` (number) - Button constant (mb_left, mb_right, mb_middle)

Returns: `boolean` - True if button is down

Example:

```
if (mouse_check_button(mb_left)) {
  // Fire weapon while holding
  this.fire();
}
```

---

mouse_check_button_pressed

Syntax: `mouse_check_button_pressed(button)`

Checks if mouse button was just pressed.

Arguments:

- `button` (number) - Button constant

Returns: `boolean` - True if just pressed

---

mouse_check_button_released

Syntax: `mouse_check_button_released(button)`

Checks if mouse button was just released.

Arguments:

- `button` (number) - Button constant

Returns: `boolean` - True if just released

Room Functions

room_goto

Syntax: `room_goto(roomName)`

Transitions to another room.

Arguments:

- `roomName` (string) - Name of room to go to

Returns: `Promise<void>`

Description: Ends the current room and loads the specified room. All instances' `roomEnd()` events are called, then the new room loads and all instances' `roomStart()` events are called.

Example:

```
// Go to next level
await room_goto('room_level2');
```

Note: Non-persistent instances are destroyed during room transitions.

Game Functions

game_end

Syntax: `game_end()`

Stops the game.

Arguments: None

Returns: `void`

Example:

```
if (this.lives <= 0) {
  game_end();
}
```

---

game_restart

Syntax: `game_restart()`

Restarts the game from the beginning.

Arguments: None

Returns: `Promise<void>`

Example:

```
if (keyboard_check_pressed(vk_r)) {
  game_restart();
}
```

Storage Functions

game_save

Syntax: `game_save(slot)`

Saves game data to localStorage.

Arguments:

- `slot` (string | number) - Save slot identifier

Returns: `boolean` - True if successful

Description: Saves the current game state to browser localStorage. You must implement custom serialization for your game data.

Example:

```
const saveData = {
  level: this.currentLevel,
  score: this.score,
  health: this.health
};
localStorage.setItem('saveData', JSON.stringify(saveData));
if (game_save('slot1')) {
  show_debug_message.call(this, 'Game saved!');
}
```

---

game_load

Syntax: `game_load(slot)`

Loads game data from localStorage.

Arguments:

- `slot` (string | number) - Save slot identifier

Returns: `boolean` - True if successful

---

game_save_exists

Syntax: `game_save_exists(slot)`

Checks if a save exists.

Arguments:

- `slot` (string | number) - Save slot identifier

Returns: `boolean` - True if save exists

---

game_save_delete

Syntax: `game_save_delete(slot)`

Deletes a save.

Arguments:

- `slot` (string | number) - Save slot identifier

Returns: `boolean` - True if successful

Random Functions

random

Syntax: `random(n)`

Returns random float between 0 and n.

Arguments:

- `n` (number) - Maximum value (exclusive)

Returns: `number` - Random value

Example:

```
const speed = random(5); // 0.0 to 5.0
```

---

irandom

Syntax: `irandom(n)`

Returns random integer between 0 and n (inclusive).

Arguments:

- `n` (number) - Maximum value (inclusive)

Returns: `number` - Random integer

Example:

```
const roll = irandom(5); // 0, 1, 2, 3, 4, or 5
```

---

random_range

Syntax: `random_range(min, max)`

Returns random float between min and max.

Arguments:

- `min` (number) - Minimum value
- `max` (number) - Maximum value

Returns: `number` - Random value

Example:

```
const enemySpeed = random_range(2, 5);
```

Debug Functions

show_debug_message

Syntax: `show_debug_message.call(this, message)`

Logs message to console with object name.

Arguments:

- `message` (string) - Message to log

Returns: `void`

Description: Outputs a message to the browser console prefixed with the object's class name.

Example:

```
show_debug_message.call(this, 'Player jumped!');
// Output: [obj_player] Player jumped!
```

Constants Reference

Keyboard Constants

Arrow Keys

- `vk_left` = 37
- `vk_right` = 39
- `vk_up` = 38
- `vk_down` = 40

Common Keys

- `vk_space` = 32
- `vk_enter` = 13
- `vk_escape` = 27
- `vk_shift` = 16
- `vk_control` = 17
- `vk_alt` = 18
- `vk_backspace` = 8
- `vk_tab` = 9

Letter Keys (A-Z)

- `vk_a` through `vk_z` = 65-90

Function Keys

- `vk_f1` through `vk_f12` = 112-123

Numpad Keys

- `vk_numpad0` through `vk_numpad9` = 96-105
- `vk_multiply` = 106
- `vk_add` = 107
- `vk_subtract` = 109
- `vk_decimal` = 110
- `vk_divide` = 111

Other Keys

- `vk_home` = 36
- `vk_end` = 35
- `vk_pageup` = 33
- `vk_pagedown` = 34
- `vk_delete` = 46
- `vk_insert` = 45

Mouse Constants

- `mb_left` = 0 - Left mouse button
- `mb_right` = 2 - Right mouse button
- `mb_middle` = 1 - Middle mouse button

Global Variables

Room Variables

- `room_width` (number) - Current room width in pixels
- `room_height` (number) - Current room height in pixels
- `room_speed` (number) - Current room FPS (usually 60)

Mouse Variables

- `mouse_x` (number) - Mouse X position in room coordinates
- `mouse_y` (number) - Mouse Y position in room coordinates

View Variables

- `view_xview` (number) - View X position in room
- `view_yview` (number) - View Y position in room
- `view_wview` (number) - View width
- `view_hview` (number) - View height
- `view_xport` (number) - View X position on screen
- `view_yport` (number) - View Y position on screen
- `view_wport` (number) - View width on screen
- `view_hport` (number) - View height on screen

Debug Variables

- `debug_mode` (boolean) - Whether debug mode is enabled (toggle with F3)

Advanced Topics

Execution Order

1. **Step Event Order**: Instances execute step events based on their `order` property (lower values first)
2. **Motion System**: Automatic conversion of `speed`/`direction` → `hspeed`/`vspeed` → `x`/`y`
3. **Animation System**: Automatic `image_index` advancement based on `image_speed`
4. **Draw Order**: Instances drawn by `depth` (higher values behind)

Collision System

- Uses AABB (axis-aligned bounding box) collision
- Bounding box calculated from sprite dimensions
- Custom bbox can be set in sprite metadata
- No automatic "solid" property - all collision is manual

Performance Tips

1. **Minimize draw calls**: Combine multiple `draw_*` calls when possible

2. **Use depth wisely**: Group objects by depth for efficient rendering
3. **Limit collision checks**: Only check collision when necessary
4. **Cache instance lookups**: Store results of `instance_find` if used multiple times

---

Sources:

- GameMaker Studio 1.4.9999 Documentation
- GameMaker Docs Repository
- GameMaker Manual