Collision API

Complete collision functions reference

Overview

Origami Engine uses **AABB** (Axis-Aligned Bounding Box) collision detection. All collision functions check rectangular bounding boxes defined in sprite metadata.

Instance Collision

`place_meeting()`

Checks if instance would collide at a position.

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

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 at the given position.

Example:

```
step(): void {
  // Check 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
  }

  // Check ground
  this.onGround = place_meeting.call(this, this.x, this.y + 1, 'obj_wall');

  // Jump if on ground
  if (keyboard_check_pressed(vk_space) && this.onGround) {
    this.vspeed = -10;
  }
}
```

Important: Must use `.call(this)` to specify which instance is checking.

`instance_place()`

Gets the instance at a position (if any).

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

Arguments:

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

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

Description: Same as `place_meeting()` but returns the actual instance instead of just a boolean. Useful when you need to interact with the colliding object.

Example:

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

  // Take damage from enemy
  const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
  if (enemy && !this.invincible) {
    this.health -= 10;
    this.invincible = true;
    this.invincibilityTimer = 120;
  }

  // Push objects
  const box = instance_place.call(this, this.x + this.hspeed, this.y, 'obj_box
  if (box) {
    box.x += this.hspeed;
  }
}
```

`place_free()`

Checks if position is free of all solid objects.

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

Arguments:

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

Returns: `boolean` - True if position is free

Description: Checks if the position is free of solid objects. In Origami Engine, this checks against all instances (there's no explicit "solid" property).

Example:

```
step(): void {
  // Apply gravity if in air
  if (place_free.call(this, this.x, this.y + 1)) {
    this.vspeed += this.GRAVITY;
  } else {
    this.vspeed = 0;
    this.onGround = true;
  }

  // Random movement in free space
  const newX = this.x + random_range(-10, 10);
  const newY = this.y + random_range(-10, 10);
  if (place_free.call(this, newX, newY)) {
    this.x = newX;
    this.y = newY;
  }
}
```

Area Collision

`collision_rectangle()`

Checks collision within a rectangular area.

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

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

Description: Checks if any instance of the specified type is within the rectangular area.

Example:

```
step(): void {
  // Check area for enemies
  const enemy = collision_rectangle(
    this.x - 50,
    this.y - 50,
    this.x + 50,
    this.y + 50,
    'obj_enemy'
  );
  if (enemy) {
    show_debug_message.call(this, 'Enemy nearby!');
    this.alertState = true;
  }

  // Sweep attack
  if (keyboard_check_pressed(vk_space)) {
    const target = collision_rectangle(
      this.x - 30,
      this.y - 30,
      this.x + 30,
      this.y + 30,
      'obj_enemy'
    );
    if (target) {
      instance_destroy.call(target);
    }
  }

  // Area effect
  const victims = [];
  let index = 0;
  while (true) {
    const victim = collision_rectangle(
      this.x - 100,
      this.y - 100,
      this.x + 100,
      this.y + 100,
      'obj_enemy'
    );
    if (!victim || victims.includes(victim)) break;
    victims.push(victim);
    // Damage all enemies in area
    victim.health -= 50;
  }
}
```

`collision_point()`

Checks collision at a single point.

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

Arguments:

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

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

Description: Checks if any instance of the specified type contains the given point within its bounding box.

Example:

```
step(): void {
  // Check mouse hover
  const hovered = collision_point(mouse_x, mouse_y, 'obj_button');
  if (hovered) {
    hovered.highlighted = true;

    if (mouse_check_button_pressed(mb_left)) {
      hovered.onClick();
    }
  }

  // Laser pointer
  const hit = collision_point(this.laserX, this.laserY, 'obj_enemy');
  if (hit) {
    hit.health -= 1;
  }
}
```

`instance_position()`

Gets instance at exact position.

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

Arguments:

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

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

Description: Similar to `collision_point()`, checks for instances at a specific position.

Example:

```
step(): void {
  // Grid-based collision
  const gridX = Math.floor(mouse_x / 32) * 32;
  const gridY = Math.floor(mouse_y / 32) * 32;
  const tile = instance_position(gridX, gridY, 'obj_tile');

  if (mouse_check_button_pressed(mb_left) && !tile) {
    await instance_create(gridX, gridY, 'obj_tile');
  }
}
```

Collision Patterns

Platformer Collision (Pixel-Perfect)

```
step(): void {
  const moveSpeed = 4;
  const GRAVITY = 0.5;

  // Horizontal movement
  if (keyboard_check(vk_d)) {
    if (!place_meeting.call(this, this.x + moveSpeed, this.y, 'obj_wall')) {
      this.x += moveSpeed;
    }
  }
  if (keyboard_check(vk_a)) {
    if (!place_meeting.call(this, this.x - moveSpeed, this.y, 'obj_wall')) {
      this.x -= moveSpeed;
    }
  }

  // Apply gravity
  this.vspeed += GRAVITY;

  // Vertical collision (pixel-perfect)
  if (place_meeting.call(this, this.x, this.y + this.vspeed, 'obj_wall')) {
    // Move pixel by pixel until we hit the wall
    while (!place_meeting.call(this, this.x, this.y + Math.sign(this.vspeed),
      this.y += Math.sign(this.vspeed);
    }
    this.vspeed = 0;
    this.onGround = true;
  } else {
    this.y += this.vspeed;
    this.onGround = false;
  }
}
```

One-Way Platforms

```
step(): void {
  // Only collide if falling and not holding down
  if (this.vspeed >= 0 && !keyboard_check(vk_s)) {
    if (place_meeting.call(this, this.x, this.y + this.vspeed, 'obj_platform'))
      // Land on platform
      while (!place_meeting.call(this, this.x, this.y + 1, 'obj_platform')) {
        this.y++;
      }
      this.vspeed = 0;
      this.onGround = true;
    }
  }
}
```

Sliding Collision

```
step(): void {
  // Try diagonal movement
  const newX = this.x + this.hspeed;
  const newY = this.y + this.vspeed;

  if (!place_meeting.call(this, newX, newY, 'obj_wall')) {
    // Free diagonal movement
    this.x = newX;
    this.y = newY;
  } else {
    // Try horizontal only
    if (!place_meeting.call(this, newX, this.y, 'obj_wall')) {
      this.x = newX;
      this.vspeed = 0;
    }
    // Try vertical only
    else if (!place_meeting.call(this, this.x, newY, 'obj_wall')) {
      this.y = newY;
      this.hspeed = 0;
    }
    // Fully blocked
    else {
      this.hspeed = 0;
      this.vspeed = 0;
    }
  }
}
```

Push Objects

```
step(): void {
  // Try to push boxes
  const box = instance_place.call(this, this.x + this.hspeed, this.y, 'obj_box
  if (box) {
    // Check if box can move
    if (!place_meeting.call(box, box.x + this.hspeed, box.y, 'obj_wall')) {
      box.x += this.hspeed;
      this.x += this.hspeed; // Move player too
    } else {
      this.hspeed = 0; // Box is blocked, stop player
    }
  } else if (!place_meeting.call(this, this.x + this.hspeed, this.y, 'obj_wall
    this.x += this.hspeed;
  }
}
```

Damage on Touch

```
step(): void {
  if (!this.invincible) {
    const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
    if (enemy) {
      this.health -= 10;
      this.invincible = true;
      this.invincibilityTimer = 120; // 2 seconds at 60 FPS

      // Knockback
      const angle = point_direction(enemy.x, enemy.y, this.x, this.y);
      this.hspeed = lengthdir_x(8, angle);
      this.vspeed = lengthdir_y(8, angle);
    }
  }

  // Update invincibility
  if (this.invincibilityTimer > 0) {
    this.invincibilityTimer--;
  } else {
    this.invincible = false;
  }
}
```

Trigger Zones

```
export class obj_trigger extends GameObject {
  private activated: boolean = false;

  create(): void {
    this.visible = false; // Invisible trigger
  }

  step(): void {
    if (!this.activated) {
      const player = instance_place.call(this, this.x, this.y, 'obj_player');
      if (player) {
        this.activated = true;
        this.onTrigger();
      }
    }
  }

  private onTrigger(): void {
    // Spawn enemies
    await instance_create(200, 100, 'obj_enemy');
    await instance_create(300, 100, 'obj_enemy');

    // Show message
    show_debug_message.call(this, 'Ambush!');
  }
}
```

Bounding Boxes

Default Bounding Box

If no `bbox` is specified in `metadata.json`, the entire sprite is used:

```json
{
  "origin": { "x": 16, "y": 16 },
  "fps": 10
}
```

For a 32x32 sprite: bbox is automatically (0, 0, 32, 32)

Custom Bounding Box

Define a smaller collision area:

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

This creates a 24x24 collision box with 4-pixel insets.

When to use:

- Tighter collision for characters
- Ignore transparent edges
- Better gameplay feel

Visualizing Bounding Boxes

Press **F3** in-game to see:

- Collision boxes (color-coded by object)
- FPS counter
- Instance count

Colors: Each object type gets a unique color for easy identification.

Optimization

Distance Check First

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (!player) return;

  // Cheap distance check first
  const dist = point_distance(this.x, this.y, player.x, player.y);
  if (dist > 200) return; // Too far, skip collision

  // Expensive collision check
  if (place_meeting.call(this, this.x, this.y, 'obj_player')) {
    // Do something
  }
}
```

Batch Collision Checks

```
private checkTimer: number = 0;

step(): void {
  this.checkTimer++;

  // Only check every 10 frames for non-critical collisions
  if (this.checkTimer % 10 === 0) {
    const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
    if (enemy) {
      // Alert state
    }
  }

  // Critical collisions every frame
  if (place_meeting.call(this, this.x, this.y + this.vspeed, 'obj_wall')) {
    this.vspeed = 0;
  }
}
```

Limit Collision Objects

```
// Instead of checking all enemies
const allEnemies = instance_number('obj_enemy');

// Only check nearby enemies
const nearbyEnemies = [];
for (let i = 0; i < allEnemies; i++) {
  const enemy = instance_find('obj_enemy', i);
  if (enemy) {
    const dist = point_distance(this.x, this.y, enemy.x, enemy.y);
    if (dist < 100) {
      nearbyEnemies.push(enemy);
    }
  }
}

// Check collision only with nearby
for (const enemy of nearbyEnemies) {
  if (instance_place.call(this, this.x, this.y, enemy)) {
    // Handle collision
  }
}
```

Common Issues

Stuck in Walls

Problem: Player gets stuck inside walls

Cause: Moving too fast (speed > wall width)

Solution: Use pixel-perfect collision or reduce speed

```
// Pixel-perfect
if (place_meeting.call(this, this.x, this.y + this.vspeed, 'obj_wall')) {
  while (!place_meeting.call(this, this.x, this.y + Math.sign(this.vspeed), 'ol
    this.y += Math.sign(this.vspeed);
  }
  this.vspeed = 0;
}
```

Collision Not Working

Checklist:

- ✅ Using `.call(this)` with collision functions?
- ✅ Object names match exactly (case-sensitive)?
- ✅ Both objects have sprites with bounding boxes?
- ✅ Enable F3 to visualize collision boxes

Jittery Movement

Problem: Object vibrates when touching walls

Cause: Alternating between collision and no collision states

Solution: Use `Math.sign()` and careful speed management

Next Steps

- **06-collision.md** - Collision guide
- **05-sprites.md** - Setting up bounding boxes
- **40-common-patterns.md** - More patterns

← Back to Index