Collision

Collision detection and response

Overview

Origami Engine uses **AABB** (Axis-Aligned Bounding Box) collision detection. Each sprite has a bounding box defined in its `metadata.json` file.

Collision Functions

`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

Example:

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

Important: Must be called with `.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
- `y` (number) - Y position
- `objectType` (string) - Object type

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

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);
  }
}
```

Use case: When you need to interact with the colliding object.

`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

Example:

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

`collision_rectangle()`

Checks rectangle collision.

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

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!');
  }
}
```

`collision_point()`

Checks point collision.

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

Arguments:

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

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

Example:

```
// Check if mouse is over an enemy
const enemy = collision_point(mouse_x, mouse_y, 'obj_enemy');
if (enemy && mouse_check_button_pressed(mb_left)) {
  instance_destroy.call(enemy);
}
```

`instance_position()`

Gets instance at exact point.

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

Arguments:

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

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

Example:

```
// Similar to collision_point but for specific position
const wall = instance_position(100, 200, 'obj_wall');
```

Bounding Boxes

Default Bounding Box

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

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

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

Custom Bounding Box

Define a smaller collision area:

```
{
  "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 on all sides.

Visualizing Bounding Boxes

Press **F3** in-game to see collision boxes with color coding!

Common Collision Patterns

Wall Collision (Platformer)

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

  // 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;
    }
  }

  // Vertical collision with pixel-perfect stopping
  if (place_meeting.call(this, this.x, this.y + this.vspeed, 'obj_wall')) {
    // Move pixel by pixel until touching wall
    while (!place_meeting.call(this, this.x, this.y + Math.sign(this.vspeed),
      this.y += Math.sign(this.vspeed);
    }
    this.vspeed = 0;
  } else {
    this.y += this.vspeed;
  }
}
```

Collectibles

```
step(): void {
  const player = instance_place.call(this, this.x, this.y, 'obj_player');
  if (player) {
    // Add to score
    (window as any).score += 10;
    // Destroy this coin
    instance_destroy.call(this);
  }
}
```

Damage on Touch

```
step(): void {
  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; // 2 seconds
  }

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

Destroy on Collision

```
// Bullet hitting enemy
step(): void {
  const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
  if (enemy) {
    instance_destroy.call(enemy);
    instance_destroy.call(this);
  }
}
```

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;
    }
  }
}
```

Advanced Collision

Multiple Object Types

Check collision with any of several types:

```
step(): void {
  const hitWall = place_meeting.call(this, this.x + this.hspeed, this.y, 'obj_w
  const hitBlock = place_meeting.call(this, this.x + this.hspeed, this.y, 'obj_

  if (hitWall || hitBlock) {
    this.hspeed = 0;
  }
}
```

Collision Radius

Check circular collision area:

```
step(): void {
  const player = instance_find('obj_player', 0);
  if (player) {
    const dist = point_distance(this.x, this.y, player.x, player.y);
    if (dist < 100) {
      // Player within 100 pixels
      this.state = 'chase';
    }
  }
}
```

Trigger Zones

Create invisible trigger areas:

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

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

  step(): void {
    if (!this.activated) {
      const player = instance_place.call(this, this.x, this.y, 'obj_player');
      if (player) {
        this.activated = true;
        // Trigger event
        await instance_create(200, 100, 'obj_enemy');
      }
    }
  }
}
```

Collision Optimization

Reduce Checks

Don't check collision every frame if not needed:

```
private checkTimer: number = 0;

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

  // Only check every 10 frames
  if (this.checkTimer % 10 === 0) {
    const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
    // ...
  }
}
```

Distance Checks First

Check distance before expensive collision:

```
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
  const collision = place_meeting.call(this, this.x, this.y, 'obj_player');
  if (collision) {
    // Do something
  }
}
```

Common Issues

Collision Not Working

Problem: Objects pass through each other

Checklist:

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

Solution:

```
// CORRECT
if (place_meeting.call(this, this.x, this.y, 'obj_wall')) { }

// WRONG (missing .call(this))
if (place_meeting(this.x, this.y, 'obj_wall')) { }
```

Stuck in Walls

Problem: Player gets stuck inside walls

Cause: Moving too fast (speed > wall width)

Solution: Pixel-perfect collision or reduce speed

```
// Pixel-perfect stopping
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), 'ob
    this.y += Math.sign(this.vspeed);
  }
  this.vspeed = 0;
}
```

Jittery Collision

Problem: Object vibrates when touching walls

Cause: Alternating between collision states

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

Next Steps

- **04-gameobjects.md** - GameObject properties
- **05-sprites.md** - Setting up bounding boxes
- **40-common-patterns.md** - More collision patterns
- **23-api-collision.md** - Complete collision API

← Back to Index