Drawing

Rendering sprites, shapes, and text

Overview

All drawing must happen in the `draw()` event of GameObjects. The engine renders objects in order by their `depth` property (higher depth = behind).

Basic Drawing

`draw_self()`

Draws the instance's current sprite.

Syntax: `draw_self.call(this)`

Example:

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

This is the most common drawing function. It automatically:

- Draws the sprite at the instance's position
- Applies rotation (`image_angle`)
- Applies scaling (`image_xscale` , `image_yscale`)
- Applies transparency (`image_alpha`)
- Uses current animation frame (`image_index`)

Sprite Drawing

`draw_sprite()`

Draws a specific sprite at a position.

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

Arguments:

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

Example:

```
draw(): void {
  // Draw health icons
  for (let i = 0; i < this.lives; i++) {
    draw_sprite('spr_heart', 0, 10 + (i * 32), 10);
  }
}
```

Use cases:

- Drawing UI elements
- Drawing sprites different from instance's sprite_index
- Drawing multiple copies of a sprite

Text Drawing

`draw_text()`

Draws text at a position.

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

Arguments:

- `x, y` (number) - Position

- `text` (string) - Text to draw

Example:

```
draw(): void {
  // Score display
  draw_text(10, 10, `Score: ${this.score}`);

  // Debug info
  draw_text(this.x, this.y - 20, `HP: ${this.health}`);

  // Instructions
  draw_text(320, 240, 'Press SPACE to start');
}
```

Note: Text position is top-left corner. Font is system default unless customized via canvas context.

Shape Drawing

`draw_rectangle()`

Draws a rectangle.

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

Arguments:

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

Example:

```
draw(): void {
  // Health bar background (red)
  draw_set_color('#FF0000');
  draw_rectangle(this.x - 25, this.y - 35, this.x + 25, this.y - 30, false);

  // Health bar foreground (green)
  const healthPercent = this.health / this.maxHealth;
  draw_set_color('#00FF00');
  draw_rectangle(
    this.x - 25,
    this.y - 35,
    this.x - 25 + (50 * healthPercent),
    this.y - 30,
    false
  );

  draw_set_color('#FFFFFF'); // Reset
}
```

`draw_circle()`

Draws a circle.

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

Arguments:

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

Example:

```
draw(): void {
  // Detection radius visualization
  draw_set_color('#FFFF00');
  draw_set_alpha(0.3);
  draw_circle(this.x, this.y, 100, false); // Filled yellow circle
  draw_set_alpha(1.0);
  draw_set_color('#FFFFFF');

  draw_self.call(this);
}
```

`draw_line()`

Draws a line between two points.

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

Arguments:

- `x1, y1` (number) - Start point
- `x2, y2` (number) - End point

Example:

```
draw(): void {
  // Draw line to target
  draw_set_color('#FF0000');
  draw_line(this.x, this.y, this.targetX, this.targetY);
  draw_set_color('#FFFFFF');

  draw_self.call(this);
}
```

Color and Style

`draw_set_color()`

Sets the drawing color.

Syntax: `draw_set_color(color)`

Arguments:

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

Example:

```
draw(): void {
  draw_set_color('#FF0000'); // Red
  draw_rectangle(10, 10, 50, 50, false);

  draw_set_color('#00FF00'); // Green
  draw_rectangle(60, 10, 100, 50, false);

  draw_set_color('#0000FF'); // Blue
  draw_rectangle(110, 10, 150, 50, false);

  draw_set_color('#FFFFFF'); // Reset to white
}
```

Common colors:

- `'#FFFFFF'` - White
- `'#000000'` - Black
- `'#FF0000'` - Red
- `'#00FF00'` - Green
- `'#0000FF'` - Blue
- `'#FFFF00'` - Yellow
- `'#FF00FF'` - Magenta
- `'#00FFFF'` - Cyan

`draw_set_alpha()`

Sets drawing transparency.

Syntax: `draw_set_alpha(alpha)`

Arguments:

- `alpha` (number) - Alpha value (0.0 = transparent, 1.0 = opaque)

Example:

```
draw(): void {
  // Semi-transparent overlay
  draw_set_alpha(0.5);
  draw_set_color('#000000');
  draw_rectangle(0, 0, room_width, room_height, false);
  draw_set_alpha(1.0); // Reset
  draw_set_color('#FFFFFF');

  // Fading text
  draw_set_alpha(this.fadeAmount);
  draw_text(100, 100, 'Fading...');
  draw_set_alpha(1.0);
}
```

Drawing Order

Depth Property

Objects with **higher depth** draw **behind** objects with lower depth.

```
create(): void {
  this.depth = 0;   // Default (middle)
  // depth = -10;    // Draw in front
  // depth = 100;    // Draw behind
}
```

Common depth values:

- `-100` - HUD/UI (always on top)
- `-10` - Player
- `0` - Default (enemies, items)
- `10` - Walls, platforms
- `100` - Background decorations

Example:

```
// obj_player
create(): void {
  this.depth = -10; // Draw in front of enemies
}

// obj_enemy
create(): void {
  this.depth = 0; // Default
}

// obj_hud
create(): void {
  this.depth = -100; // Always on top
}
```

Common Drawing Patterns

Health Bar

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

  // Background
  draw_set_color('#000000');
  draw_rectangle(this.x - 26, this.y - 36, this.x + 26, this.y - 29, false);

  // Red (missing health)
  draw_set_color('#FF0000');
  draw_rectangle(this.x - 25, this.y - 35, this.x + 25, this.y - 30, false);

  // Green (current health)
  const barWidth = 50;
  const healthPercent = this.health / this.maxHealth;
  draw_set_color('#00FF00');
  draw_rectangle(
    this.x - 25,
    this.y - 35,
    this.x - 25 + (barWidth * healthPercent),
    this.y - 30,
    false
  );

  draw_set_color('#FFFFFF');
}
```

Score Display

```
export class obj_hud extends GameObject {
  create(): void {
    this.depth = -100; // Always on top
  }

  draw(): void {
    draw_set_color('#FFFF00');
    draw_text(10, 10, `Score: ${(window as any).score || 0}`);
    draw_text(10, 30, `Lives: ${(window as any).lives || 3}`);
    draw_set_color('#FFFFFF');
  }
}
```

Timer Display

```
draw(): void {
  const seconds = Math.floor(this.timer / 60);
  const minutes = Math.floor(seconds / 60);
  const displaySeconds = seconds % 60;

  const timeString = `${minutes}:${displaySeconds.toString().padStart(2, '0')}

  draw_set_color('#FFFFFF');
  draw_text(10, 10, `Time: ${timeString}`);
}
```

Progress Bar

```
draw(): void {
  const barX = 100;
  const barY = 400;
  const barWidth = 200;
  const barHeight = 20;
  const progress = this.loadProgress; // 0.0 to 1.0

  // Background
  draw_set_color('#333333');
  draw_rectangle(barX, barY, barX + barWidth, barY + barHeight, false);

  // Progress
  draw_set_color('#00FF00');
  draw_rectangle(
    barX,
    barY,
    barX + (barWidth * progress),
    barY + barHeight,
    false
  );

  // Border
  draw_set_color('#FFFFFF');
  draw_rectangle(barX, barY, barX + barWidth, barY + barHeight, true);
}
```

Direction Indicator

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

  // Draw line showing direction
  const lineLength = 40;
  const endX = this.x + lengthdir_x(lineLength, this.direction);
  const endY = this.y + lengthdir_y(lineLength, this.direction);

  draw_set_color('#FF0000');
  draw_line(this.x, this.y, endX, endY);
  draw_set_color('#FFFFFF');
}
```

Flash Effect

```
private flashTimer: number = 0;

step(): void {
  if (this.flashTimer > 0) {
    this.flashTimer--;
  }
}

draw(): void {
  // Flash white when hit
  if (this.flashTimer > 0 && this.flashTimer % 4 < 2) {
    draw_set_color('#FFFFFF');
    draw_set_alpha(0.5);
    draw_rectangle(
      this.x - 16,
      this.y - 16,
      this.x + 16,
      this.y + 16,
      false
    );
    draw_set_alpha(1.0);
  }

  draw_self.call(this);
}

// Trigger flash
takeDamage(): void {
  this.health -= 10;
  this.flashTimer = 20;
}
```

GUI Layer

For UI elements that don't scroll with the camera:

```
export class obj_hud extends GameObject {
  create(): void {
    this.depth = -1000; // Very high priority
    this.persistent = true; // Survive room changes
  }

  draw(): void {
    // These coordinates are relative to the viewport, not the room
    draw_set_color('#FFFFFF');
    draw_text(10, 10, `Score: ${(window as any).score}`);
    draw_text(10, 30, `Health: ${(window as any).playerHealth}`);
  }
}
```

Performance Tips

Avoid Heavy Drawing in step()

```
// BAD: Drawing in step (doesn't display, wastes CPU)
step(): void {
  draw_text(10, 10, 'Text'); // ❌ Won't display!
}

// GOOD: Drawing in draw()
draw(): void {
  draw_text(10, 10, 'Text'); // ✅ Displays correctly
}
```

Cache Complex Calculations

```
private cachedHealthBar: number = 0;

step(): void {
  // Calculate once per frame
  this.cachedHealthBar = (this.health / this.maxHealth) * 50;
}

draw(): void {
  // Use cached value
  draw_rectangle(
    this.x - 25,
    this.y - 35,
    this.x - 25 + this.cachedHealthBar,
    this.y - 30,
    false
  );
}
```

Draw Only When Visible

```
draw(): void {
  // Check if on screen
  if (this.x < view_xview[0] - 50 || this.x > view_xview[0] + view_wview[0] + 
    return; // Off screen, skip drawing
  }
  if (this.y < view_yview[0] - 50 || this.y > view_yview[0] + view_hview[0] + 
    return;
  }

  draw_self.call(this);
}
```

Common Issues

Drawing Not Appearing

Checklist:

- ✅ Drawing code is in `draw()` event (not `step()`)
- ✅ Object has correct `depth` (not behind everything)
- ✅ Colors and alpha are set correctly
- ✅ Object is within view/camera bounds
- ✅ Object's `visible` property is true

Colors Don't Reset

Problem: Everything draws in the wrong color

Cause: Forgot to reset color after changing it

Solution: Always reset to white

```
draw(): void {
  draw_set_color('#FF0000');
  draw_rectangle(10, 10, 50, 50, false);
  draw_set_color('#FFFFFF'); // Always reset!
}
```

Text Position Wrong

Problem: Text appears in wrong place relative to sprite

Cause: Text position is absolute, not relative to sprite origin

Solution: Calculate relative to sprite position

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

  // Text above sprite (accounting for origin)
  const textX = this.x;
  const textY = this.y - 30; // 30 pixels above center
  draw_text(textX, textY, 'Name');
}
```

Next Steps

- **05-sprites.md** - Understanding sprites
- **04-gameobjects.md** - Draw event
- **22-api-drawing.md** - Complete drawing API

← Back to Index