Debugging

Debug tools and troubleshooting

Debug Mode (F3)

Press **F3** or **~** (tilde) in-game to toggle debug overlay.

What Debug Mode Shows

1. **FPS Counter** - Should be ~60 FPS
2. **Instance Count** - Total number of active objects
3. **Collision Boxes** - Color-coded by object type
4. **View/Camera Position** - Current camera coordinates

Color-Coded Collision Boxes

Each object type gets a unique color:

- Red
- Green
- Blue
- Yellow
- Magenta
- Cyan

Uses:

- Verify collision box size
- Check for overlapping objects
- Debug placement issues
- Visualize object relationships

Console Logging

Using `show_debug_message()`

```
step(): void {
  show_debug_message.call(this, `Player pos: ${this.x}, ${this.y}`);
  show_debug_message.call(this, `Health: ${this.health}`);
  show_debug_message.call(this, `Speed: ${this.speed}`);
}
```

Output (Browser console, F12):

```
[obj_player] Player pos: 150, 200
[obj_player] Health: 95
[obj_player] Speed: 4
```

Conditional Logging

```
step(): void {
  // Only log when something interesting happens
  if (this.health < 20) {
    show_debug_message.call(this, `⚠️ Low health: ${this.health}`);
  }

  const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
  if (enemy) {
    show_debug_message.call(this, '💥 Hit enemy!');
  }
}
```

Using console.log()

Standard JavaScript logging also works:

```
step(): void {
  console.log('Player position:', this.x, this.y);
  console.log('Player object:', this);
}
```

Browser Developer Tools

Opening Dev Tools

- **Chrome/Edge**: F12 or Ctrl+Shift+I
- **Firefox**: F12 or Ctrl+Shift+K
- **Safari**: Cmd+Option+I (enable first in preferences)

Console Tab

View all log messages, errors, and warnings.

Filter by level:

- Info (console.log)
- Warnings (console.warn)
- Errors (console.error)

Sources Tab

Breakpoint Debugging:

1. Navigate to your TypeScript/JavaScript files
2. Click line number to set breakpoint
3. Game pauses when line is hit
4. Inspect variables, step through code

Network Tab

View loaded assets:

- Sprites (PNG files)
- Scripts (JS files)

- Metadata (JSON files)

Check for:

- 404 errors (missing files)
- Load times
- File sizes

Common Issues & Solutions

Sprites Don't Appear

Symptoms:

- Object exists (shows in debug mode)
- But no sprite visible

Checklist:

- ✅ `metadata.json` exists in sprite folder?
- ✅ PNG files named `frame_0.png`, `frame_1.png`, etc.?
- ✅ No gaps in frame numbering?
- ✅ Sprite folder name matches `sprite_index`?
- ✅ Check browser console (F12) for errors?
- ✅ Is `visible` property true?
- ✅ Is `image_alpha` greater than 0?

Solution:

```
create(): void {
  this.sprite_index = 'spr_player'; // Verify name
  this.visible = true;
  this.image_alpha = 1.0;
}
```

Collision Not Working

Symptoms:

- Objects pass through each other
- `place_meeting()` returns false

Checklist:

- ✅ Using `.call(this)` with collision functions?
- ✅ Object names match exactly (case-sensitive)?
- ✅ Objects registered in `src/main.ts`?
- ✅ Collision boxes overlap (enable F3 debug mode)?
- ✅ Both objects have sprites with bounding boxes?

Solution:

```
step(): void {
  // CORRECT: use .call(this)
  if (place_meeting.call(this, this.x, this.y, 'obj_wall')) {
    // Collision detected
  }

  // WRONG: missing .call(this)
  if (place_meeting(this.x, this.y, 'obj_wall')) {
    // Won't work!
  }
}
```

Object Not Appearing

Symptoms:

- Object exists in room JSON
- But never appears in game

Checklist:

- ✅ Object registered in `src/main.ts`?
- ✅ Room JSON has correct object name?
- ✅ Position is within room bounds?
- ✅ Did you rebuild after changes? (`npm run build`)

Solution:

```
// src/main.ts
import { obj_player } from '../objects/obj_player.js';
engine.registerObject(obj_player); // Must register!
```

Animation Not Playing

Symptoms:

- Sprite appears but doesn't animate
- Stuck on first frame

Checklist:

- ✅ `fps` in metadata.json greater than 0?
- ✅ Multiple frame files exist (frame_0, frame_1, etc.)?
- ✅ `image_speed` is not 0?
- ✅ Sprite has more than one frame?

Solution:

```
create(): void {
  this.sprite_index = 'spr_player';
  this.image_speed = 1.0; // Enable animation
}
```

Low FPS / Performance Issues

Symptoms:

- FPS counter shows < 60 FPS
- Game feels sluggish

Common causes:

1. **Too many instances** - Check instance count in debug mode
2. **Heavy draw() methods** - Minimize drawing operations
3. **Console spam** - Remove excessive `show_debug_message()` calls
4. **Large sprites** - Optimize image sizes

Solutions:

```
// Reduce instance count
if (instance_count('obj_particle') > 100) {
  // Stop creating more particles
}

// Optimize drawing
draw(): void {
  draw_self.call(this);
  // Avoid complex calculations here
}

// Batch operations
private frameCounter: number = 0;
step(): void {
  this.frameCounter++;
  // Only run expensive checks every 10 frames
  if (this.frameCounter % 10 === 0) {
    // Expensive operation here
  }
}
```

Input Not Working

Symptoms:

- Keyboard/mouse input ignored

Checklist:

- ✅ Using correct key constants? (`vk_space` , `vk_a` , etc.)
- ✅ Browser tab has focus?
- ✅ Input check is in `step()` method?
- ✅ Check for JavaScript errors in console?

Solution:

```
step(): void {
  // Check keyboard
  if (keyboard_check(vk_space)) {
    console.log('Space pressed');
  }

  // Check mouse
  if (mouse_check_button(mb_left)) {
    console.log('Left mouse pressed');
  }
}
```

Debugging Techniques

Visualizing Variables

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

  // Draw debug text above object
  draw_set_color('#FFFF00');
  draw_text(this.x, this.y - 20, `HP: ${this.health}`);
  draw_text(this.x, this.y - 35, `Speed: ${this.speed.toFixed(1)}`);
  draw_set_color('#FFFFFF');
}
```

Temporary Debug Objects

Create a debug object to display game state:

```
export class obj_debug extends GameObject {
  draw(): void {
    const playerCount = instance_count('obj_player');
    const enemyCount = instance_count('obj_enemy');
    const coinCount = instance_count('obj_coin');

    draw_set_color('#FFFFFF');
    draw_text(10, 10, `Players: ${playerCount}`);
    draw_text(10, 25, `Enemies: ${enemyCount}`);
    draw_text(10, 40, `Coins: ${coinCount}`);
  }
}
```

State Debugging

```
enum PlayerState {
  Idle,
  Running,
  Jumping,
  Falling
}

export class obj_player extends GameObject {
  private state: PlayerState = PlayerState.Idle;

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

    // Show current state
    const stateName = PlayerState[this.state];
    draw_text(this.x, this.y - 30, stateName);
  }
}
```

Path Visualization

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

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

  // Draw circle at target
  draw_circle(this.targetX, this.targetY, 5, true);
}
```

Performance Profiling

Measuring Execution Time

```
step(): void {
  const startTime = performance.now();

  // Your code here
  this.expensiveOperation();

  const endTime = performance.now();
  const duration = endTime - startTime;

  if (duration > 1) { // More than 1ms
    console.log(`⚠️ Slow operation: ${duration.toFixed(2)}ms`);
  }
}
```

Instance Counting

```
step(): void {
  const totalInstances = instance_count();
  if (totalInstances > 1000) {
    console.warn(`⚠️ Too many instances: ${totalInstances}`);
  }
}
```

Error Messages

Common TypeScript Errors

"Cannot read property 'x' of undefined":

```
// Problem: object might be null
const enemy = instance_find('obj_enemy', 0);
enemy.x = 100; // Crash if no enemy exists!

// Solution: Check first
const enemy = instance_find('obj_enemy', 0);
if (enemy) {
  enemy.x = 100;
}
```

"is not a function":

```
// Problem: Missing .call(this)
place_meeting(this.x, this.y, 'obj_wall'); // Not a function!

// Solution: Add .call(this)
place_meeting.call(this, this.x, this.y, 'obj_wall');
```

Best Practices

1. **Remove Debug Code**: Comment out or remove `show_debug_message()` calls before release
2. **Use Debug Flags**: Control debug output with a flag

```
private DEBUG = true;

step(): void {
  if (this.DEBUG) {
    show_debug_message.call(this, `Pos: ${this.x}, ${this.y}`);
  }
}
```

3. **Consistent Naming**: Use descriptive object names for easier debugging
4. **Small Iterations**: Test frequently, make small changes
5. **Browser Console**: Keep dev tools open while developing
6. **F3 Debug Mode**: Use it liberally to visualize collision boxes

---

Next Steps

- **04-gameobjects.md** - Understanding object structure
- **08-input.md** - Input handling
- **40-common-patterns.md** - Best practices

← Back to Index