# Rooms

**Creating and managing game levels**

## Overview

Rooms are the levels/scenes in your game. Each room is a JSON file that defines the layout, objects, camera settings, and visual properties.

## Room File Location

```
rooms/
├── room_start.json
├── room_level1.json
├── room_level2.json
└── room_boss.json
```

**Requirements**:

- Place room files in the `rooms/` folder
- Use `.json` extension
- File name should match the room name

## Basic Room Structure

```
{
  "name": "room_level1",
  "width": 1024,
  "height": 768,
  "speed": 60,
  "backgroundColor": "#87CEEB",
  "instances": [],
  "views": [],
  "backgrounds": []
}
```

---

## Properties

### `name`

The room's identifier (must match filename).

```
{
  "name": "room_level1"
}
```

### `width` and `height`

Room dimensions in pixels.

```
{
  "width": 1024,
  "height": 768
}
```

**Common sizes**:

- `640 x 480` - Small room

- `1024 x 768` - Medium room
- `2048 x 1536` - Large room
- `4096 x 3072` - Very large room

### `speed`

Room update rate (frames per second).

```
{
  "speed": 60
}
```

**Always use 60** for consistent gameplay across browsers.

### `backgroundColor`

Background color (hex or CSS color).

```
{
  "backgroundColor": "#87CEEB"
}
```

**Examples**:

- `"#87CEEB"` - Sky blue
- `"#000000"` - Black
- `"#FFFFFF"` - White
- `"transparent"` - Transparent (for layered rooms)

---

## Instances

The `instances` array defines which objects appear in the room and where.

## Basic Instance

```
{
  "instances": [
    {
      "object": "obj_player",
      "x": 100,
      "y": 100
    }
  ]
}
```

## Multiple Instances

```
{
  "instances": [
    {
      "object": "obj_player",
      "x": 100,
      "y": 100
    },
    {
      "object": "obj_wall",
      "x": 0,
      "y": 400
    },
    {
      "object": "obj_wall",
      "x": 32,
      "y": 400
    },
    {
      "object": "obj_enemy",
      "x": 300,
      "y": 350
    }
  ]
}
```

### Creating Level Layouts

**Tip**: Use a consistent grid (e.g., 32x32) for wall placement:

```
{
  "instances": [
    {"object": "obj_wall", "x": 0, "y": 400},
    {"object": "obj_wall", "x": 32, "y": 400},
    {"object": "obj_wall", "x": 64, "y": 400},
    {"object": "obj_wall", "x": 96, "y": 400}
  ]
}
```

---

## Views (Camera System)

The `views` array controls the camera. Most games use a single view.

### No Camera (Fixed Screen)

```
{
  "views": []
}
```

Room displays at full size, no scrolling.

## Basic Following Camera

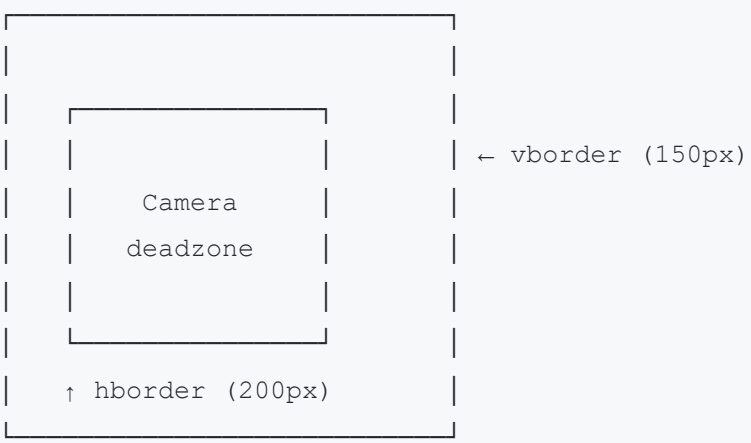
```
{
  "views": [{
    "enabled": true,
    "x": 0,
    "y": 0,
    "width": 640,
    "height": 480,
    "object": "obj_player",
    "hborder": 200,
    "vborder": 150
  }]
}
```

**Properties**:

- `enabled` - Whether this view is active
- `x`, `y` - Initial camera position
- `width`, `height` - Viewport size (visible area)
- `object` - Object to follow (e.g., player)
- `hborder` - Horizontal deadzone (pixels)
- `vborder` - Vertical deadzone (pixels)

## Deadzone Explanation

The camera only moves when the player gets within the border distance from the edge of the view.



Player can move freely within deadzone. Camera follows once player exits deadzone.

### Camera Configurations

**Tight Follow** (small borders):

```
{
  "hborder": 100,
  "vborder": 75
}
```

**Loose Follow** (large borders):

```
{
  "hborder": 250,
  "vborder": 200
}
```

**Center on Player** (borders = half viewport):

```
{
  "width": 640,
  "height": 480,
  "hborder": 320,
  "vborder": 240
}
```

---

## Backgrounds

The `backgrounds` array defines background layers.

## Simple Background

```
{
  "backgrounds": [
    {
      "sprite": "spr_background_sky",
      "x": 0,
      "y": 0,
      "htiled": true,
      "vtiled": false,
      "hspeed": 0,
      "vspeed": 0
    }
  ]
}
```

### Parallax Scrolling

```
{
  "backgrounds": [
    {
      "sprite": "spr_background_mountains",
      "x": 0,
      "y": 0,
      "htiled": true,
      "vtiled": false,
      "hspeed": -0.5,
      "vspeed": 0
    },
    {
      "sprite": "spr_background_clouds",
      "x": 0,
      "y": 0,
      "htiled": true,
      "vtiled": false,
      "hspeed": -1.0,
      "vspeed": 0
    }
  ]
}
```

Backgrounds with different speeds create depth illusion.

---

## Room Transitions

### Going to Another Room

```
// In any GameObject
async step(): Promise<void> {
  if (keyboard_check_pressed(vk_enter)) {
    await room_goto('room_level2');
  }
}
```

### Conditional Transitions

```
async step(): Promise<void> {
  // Collect all coins to proceed
  if (!instance_exists('obj_coin')) {
    await room_goto('room_level2');
  }
}
```

### Door/Portal Objects

```
export class obj_door extends GameObject {
  private targetRoom: string = 'room_level2';

  create(): void {
    this.sprite_index = 'spr_door';
  }

  async step(): Promise<void> {
    const player = instance_place.call(this, this.x, this.y, 'obj_player');
    if (player && keyboard_check_pressed(vk_up)) {
      await room_goto(this.targetRoom);
    }
  }
}
```

---

## Persistent Objects

Objects can persist between rooms:

```
export class obj_player extends GameObject {
  create(): void {
    this.persistent = true; // Don't destroy when changing rooms
    this.sprite_index = 'spr_player';
  }
}
```

**Use cases**:

- Player character
- Score/HUD objects
- Music controllers
- Game managers

**Note**: Persistent objects keep their position. Reset position in `roomStart()` if needed:

```
roomStart(): void {
  // Reset position at start of new room
  this.x = 100;
  this.y = 100;
}
```

---

# Room Events

### `roomStart()`

Called for all instances when room loads.

```
roomStart(): void {
  console.log('Level started!');
  this.health = 100; // Reset health
}
```

### `roomEnd()`

Called for all instances when leaving room.

```
roomEnd(): void {
  console.log('Leaving room...');
  // Save progress
  localStorage.setItem('level', 'room_level2');
}
```

---

## Complete Room Example

```
{
  "name": "room_level1",
  "width": 2048,
  "height": 768,
  "speed": 60,
  "backgroundColor": "#87CEEB",
  "instances": [
    {
      "object": "obj_player",
      "x": 100,
      "y": 100
    },
    {
      "object": "obj_wall",
      "x": 0,
      "y": 400
    },
    {
      "object": "obj_wall",
      "x": 32,
      "y": 400
    },
    {
      "object": "obj_enemy",
      "x": 300,
      "y": 350
    },
    {
      "object": "obj_coin",
      "x": 200,
      "y": 300
    },
    {
      "object": "obj_coin",
      "x": 250,
      "y": 300
    },
```

```
    {
      "object": "obj_door",
      "x": 900,
      "y": 350
    }
  ],
  "views": [{
    "enabled": true,
    "x": 0,
    "y": 0,
    "width": 640,
    "height": 480,
    "object": "obj_player",
    "hborder": 200,
    "vborder": 150
  }],
  "backgrounds": []
}
```

# Common Patterns

## Hub World with Multiple Exits

```
export class obj_door extends GameObject {
  private targetRoom: string;
  private doorNumber: number;

  create(): void {
    this.sprite_index = 'spr_door';
  }

  setTarget(room: string, num: number): void {
    this.targetRoom = room;
    this.doorNumber = num;
  }

  async step(): Promise<void> {
    const player = instance_place.call(this, this.x, this.y, 'obj_player');
    if (player && keyboard_check_pressed(vk_up)) {
      await room_goto(this.targetRoom);
    }
  }

  draw(): void {
    draw_self.call(this);
    draw_text(this.x, this.y - 40, `Level ${this.doorNumber}`);
  }
}
```

### Respawn Points

```
export class obj_player extends GameObject {
  private respawnX: number = 100;
  private respawnY: number = 100;

  roomStart(): void {
    // Spawn at respawn point
    this.x = this.respawnX;
    this.y = this.respawnY;
  }

  step(): void {
    // Check for checkpoint
    const checkpoint = instance_place.call(this, this.x, this.y, 'obj_checkpoi
    if (checkpoint) {
      this.respawnX = checkpoint.x;
      this.respawnY = checkpoint.y;
    }

    // Die and respawn
    if (this.y > room_height) {
      this.x = this.respawnX;
      this.y = this.respawnY;
    }
  }
}
```

---

## Tips & Best Practices

1. **Room Size**: Make rooms large enough for exploration but not so large that navigation is tedious
2. **Camera Settings**: Test different border values. Too tight = motion sickness, too loose = player off-screen
3. **Grid Layout**: Use a consistent grid (32x32 or 64x64) for wall placement
4. **Object Organization**: Group similar objects in the instances array for easier editing

5. **Starting Room**: Name your first room `room_start` for clarity

6. **Testing**: Create a simple test room for trying out new mechanics

---

## Next Steps

- **04-gameobjects.md** - Creating objects for rooms
- **06-collision.md** - Collision with room objects
- **40-common-patterns.md** - Level design patterns

---

← Back to Index