Getting Started with Origami Engine

Complete guide to creating your first game with Origami Engine

📋 Prerequisites

- Node.js 18.0.0 or higher
- Modern web browser (Chrome, Firefox, Edge, Safari)
- Code editor (VS Code recommended)
- Basic TypeScript/JavaScript knowledge

🚀 Installation

Option 1: Try the Example (Fastest)

If you have access to this repository:

```
# Clone or navigate to repository
cd Origami-Engine

# Install dependencies (requires pnpm)
npm install -g pnpm
pnpm install

# Build packages
pnpm build

# Run example game
cd platformer
pnpm install
pnpm start
```

Open http://localhost:3000 in your browser!

Option 2: Install CLI (After npm Publishing)

Once published to npm:

```
# Install CLI globally
npm install -g origami-cli

# Create a new project
ori create my-first-game

# Navigate and setup
cd my-first-game
npm install

# Start development
npm start
```

🎮 Your First Game

Step 1: Explore the Template

The template includes a working platformer. Try playing it first:

Controls:

- **W/A/S/D** - Move player
- **Space** - Jump
- **F3** - Toggle debug mode

What to Notice:

- Player physics (gravity, jumping)
- Wall collision
- Yellow collectibles
- Red enemy patrol

Step 2: Modify the Player

Open `objects/obj_player.ts` and make changes:

```
step(): void {
  // CHANGE: Make jump higher
  if (keyboard_check_pressed(vk_space) && this.onGround) {
    this.vspeed = -15; // Was -10
  }

  // CHANGE: Make player faster
  const moveSpeed = 6; // Was 4
  if (keyboard_check(vk_d)) {
    this.hspeed = moveSpeed;
  }
  if (keyboard_check(vk_a)) {
    this.hspeed = -moveSpeed;
  }

  // ... rest of code
}
```

Save the file and refresh your browser to see the changes!

Step 3: Add a New Object

Create a new file: `objects/obj_powerup.ts`

```
import { GameObject } from 'origami-runtime';

export class obj_powerup extends GameObject {
  private collected: boolean = false;

  create(): void {
    this.sprite_index = 'spr_collectible'; // Reuse collectible sprite
    this.image_xscale = 1.5; // Make it bigger
    this.image_yscale = 1.5;
  }

  step(): void {
    // Spin animation
    this.image_angle += 5;

    // Check for player collision
    const player = instance_place.call(this, this.x, this.y, 'obj_player');
    if (player && !this.collected) {
      this.collected = true;

      // Give player double jump power
      show_debug_message.call(this, 'Power-up collected!');

      // Destroy after collection
      instance_destroy.call(this);
    }
  }
}
```

Step 4: Register the New Object

Edit `src/main.ts`:

```
import { obj_powerup } from '../objects/obj_powerup.js';

const objectTypes = {
  obj_player,
  obj_wall,
  obj_collectible,
  obj_enemy,
  obj_powerup // Add this line
};
```

Step 5: Add to Room

Edit `rooms/room_level1.json`, add to the instances array:

```
{
  "object": "obj_powerup",
  "x": 400,
  "y": 200
}
```

Refresh browser - you should now see a spinning power-up!

🎨 Creating Custom Sprites

Step 1: Create Sprite Folder

```
mkdir -p sprites/spr_mysprite
```

Step 2: Add Image

Create or add `sprites/spr_mysprite/frame_0.png`:

- Use any image editor (GIMP, Photoshop, Aseprite, etc.)
- Size: any size (e.g., 32x32 pixels)
- Format: PNG with transparent background
- Name: `frame_0.png`

For animation, add more frames:

- `frame_1.png`
- `frame_2.png`
- etc.

Step 3: Create Metadata

Create `sprites/spr_mysprite/metadata.json`:

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

Properties:

- `origin` - Pivot point (usually center of sprite)
- `fps` - Animation speed (frames per second)

Step 4: Use in Object

```
create(): void {
  this.sprite_index = 'spr_mysprite';
  this.image_speed = 1; // Play animation at normal speed
}
```

🏠 Creating New Rooms

Step 1: Create Room File

Create `rooms/room_level2.json`:

```
{
  "name": "room_level2",
  "width": 1024,
  "height": 768,
  "speed": 60,
  "backgroundColor": "#336699",
  "instances": [
    {
      "object": "obj_player",
      "x": 100,
      "y": 100
    },
    {
      "object": "obj_wall",
      "x": 0,
      "y": 700
    }
  ],
  "views": [{
    "enabled": true,
    "x": 0,
    "y": 0,
    "width": 640,
    "height": 480,
    "object": "obj_player",
    "hborder": 200,
    "vborder": 150
  }]
}
```

Step 2: Transition to New Room

In any object, call:

```
await room_goto('room_level2');
```

Example - transition when collecting an item:

```
// In obj_goal
step(): void {
  const player = instance_place.call(this, this.x, this.y, 'obj_player');
  if (player) {
    await room_goto('room_level2');
  }
}
```

🎯 Common Patterns

Pattern: Health System

```
export class obj_player extends GameObject {
  private health: number = 100;
  private maxHealth: number = 100;

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

  step(): void {
    // Take damage from enemies
    const enemy = instance_place.call(this, this.x, this.y, 'obj_enemy');
    if (enemy) {
      this.health -= 1;
      if (this.health <= 0) {
        game_restart();
      }
    }
  }

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

    // Draw health bar
    const barWidth = 50;
    const barHeight = 5;
    const healthPercent = this.health / this.maxHealth;

    draw_set_color('#FF0000');
    draw_rectangle(
      this.x - 25, this.y - 35,
      this.x - 25 + (barWidth * healthPercent), this.y - 30,
      false
    );
    draw_set_color('#FFFFFF');
  }
}
```

Pattern: Score System

Create a global score variable in `src/main.ts`:

```
// Add before engine initialization
(window as any).score = 0;

// In any object:
step(): void {
  const coin = instance_place.call(this, this.x, this.y, 'obj_coin');
  if (coin) {
    (window as any).score += 10;
    instance_destroy.call(coin);
  }
}

draw(): void {
  draw_text(10, 10, `Score: ${(window as any).score}`);
}
```

Pattern: Simple AI

```
export class obj_enemy extends GameObject {
  private moveDirection: number = 1; // 1 = right, -1 = left

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

  step(): void {
    const speed = 2;

    // Move
    this.x += speed * this.moveDirection;

    // Check for walls
    if (place_meeting.call(this, this.x + speed, this.y, 'obj_wall')) {
      this.moveDirection *= -1; // Turn around
    }

    // Check for edges (don't fall off)
    if (!place_meeting.call(this, this.x, this.y + 32, 'obj_wall')) {
      this.moveDirection *= -1;
    }
  }
}
```

🐛 Debugging

Use Debug Mode (F3)

Press F3 in-game to see:

- FPS (should be ~60)
- Instance count
- Collision boxes (each object type has different color)
- Camera position

Use show_debug_message()

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

Check browser console (F12) for output:

```
[obj_player] Player position: 150, 200
[obj_player] Health: 95
```

Common Issues

Sprites don't appear:

- Check `metadata.json` exists in sprite folder
- Verify PNG files are named `frame_0.png`, `frame_1.png`, etc.
- Check browser console for loading errors

Collision not working:

- Use `place_meeting.call(this, ...)` with `.call(this)`
- Check object names match exactly (case-sensitive)
- Verify sprites have proper bounding boxes

Object not creating:

- Check object is registered in `src/main.ts`
- Verify room JSON has correct object name
- Check browser console for errors

📦 Building for Distribution

Step 1: Build Project

```
npm run build
```

This creates a `build/` folder with your compiled game.

Step 2: Test Build

Open `build/index.html` in a browser to test.

Step 3: Deploy

Option A: itch.io

1. Zip the `build/` folder
2. Upload to itch.io as HTML5 game
3. Set viewport to your game dimensions

Option B: GitHub Pages

1. Push `build/` to `gh-pages` branch
2. Enable GitHub Pages in repository settings
3. Access at `https://username.github.io/repo-name/`

Option C: Netlify/Vercel

1. Connect your repository
2. Set build command: `npm run build`
3. Set publish directory: `build`
4. Deploy!

📚 Next Steps

- **Runtime API Reference** - All functions and events
- **CLI Documentation** - CLI commands and configuration
- **Project Specification** - Engine architecture and design

💡 Tips

1. **Save Often** - Browser refresh loses runtime state
2. **Use TypeScript** - Strict mode catches errors early
3. **Check Console** - F12 shows errors and debug messages
4. **Test Incremental** - Make small changes and test frequently
5. **Study the Template** - The platformer example demonstrates all features

---

Happy game making! 🎮