Origami CLI - Complete Documentation

Package: `origami-cli` **Version**: 0.1.0 **Type**: Command-Line Interface Tool

Table of Contents

Introduction

The Origami CLI (`ori`) is a command-line tool for creating, developing, and building games with the Origami Engine. It provides scaffolding, development server, and build functionality inspired by modern JavaScript tooling while maintaining the simplicity of GameMaker workflows.

Features

- **Project Scaffolding** - Create new games from templates with one command
- **Development Server** - Local server with instant browser preview
- **Build System** - Production-ready build output
- **Template System** - Pre-configured platformer example included
- **TypeScript Support** - Full type checking and compilation

Installation

Global Installation (Recommended)

```
npm install -g origami-cli
```

After global installation, the `ori` command is available system-wide.

Local Installation

```
npm install origami-cli
```

Use with npx:

```
npx ori <command>
```

From Source (Development)

```
# Clone repository
git clone <repository-url>
cd Origami-Engine

# Install dependencies
pnpm install

# Build packages
pnpm build

# Use CLI
node packages/cli/dist/index.js <command>
```

Commands Reference

ori create

Syntax: `ori create <project-name> [options]`

Creates a new Origami Engine game project from a template.

Arguments

Argument	Type	Required	Description
`project-name`	string	Yes	Name of the project directory to create

Options

Currently no additional options. Future versions may support:

- `--template <name>` - Choose different templates
- `--typescript` / `--javascript` - Language choice
- `--skip-install` - Don't run npm install

Description

Creates a new game project directory with the following:

- Complete project structure
- Platformer example game
- All necessary sprites and assets
- TypeScript configuration
- Package.json with scripts
- Git ignore file

The template includes:

- **Player object** with WASD movement and jumping
- **Wall object** for platforms
- **Collectible object** with respawn behavior
- **Enemy object** with patrol AI
- **Working room** with all objects placed

- **All sprites** (PNG files with metadata)

Examples

```
# Create a new game called "my-platformer"
ori create my-platformer

# Navigate into the project
cd my-platformer

# Install dependencies
npm install

# Start development
npm start
```

Output

The command provides checklist-style output:

```
Creating Origami Engine project: my-platformer

✓ Created project directory
✓ Copied template files
✓ Generated package.json
✓ Created game.json configuration
✓ Initialized project structure

Project created successfully!

Next steps:
  cd my-platformer
  npm install
  npm start
```

Project Contents

After creation, your project will have:

```
my-platformer/
├── objects/
│   ├── obj_player.ts        # Player controller
│   ├── obj_wall.ts          # Solid platforms
│   ├── obj_collectible.ts   # Collectible items
│   └── obj_enemy.ts         # Enemy AI
├── sprites/
│   ├── spr_player/
│   │   ├── metadata.json
│   │   └── frame_0.png
│   ├── spr_wall/
│   ├── spr_collectible/
│   └── spr_enemy/
├── rooms/
│   └── room_level1.json     # First level
├── src/
│   └── main.ts              # Game entry point
├── game.json                # Game configuration
├── index.html               # HTML entry
├── package.json
├── tsconfig.json
└── .gitignore
```

---

ori dev

Syntax: `ori dev [options]`

Starts a local development server for testing your game.

Arguments

None

Options

Currently no additional options. Future versions may support:

- `--port <number>` - Custom port number
- `--open` - Automatically open browser

- `--watch` - Auto-rebuild on file changes

Description

Launches a simple HTTP server that serves your game. The server:

- Serves static files from your project directory
- Hosts on `localhost:3000` by default
- Provides CORS headers for asset loading
- Logs requests to console

Note: This is a simple static file server. You must manually refresh the browser to see changes. Hot module reloading is not supported in the MVP.

Examples

```
# Start dev server
ori dev

# Server output:
# Origami Dev Server
# ==================
#
# Server running at: http://localhost:3000
# Press Ctrl+C to stop
```

Workflow

1. Run `ori dev` in your project directory
2. Open `http://localhost:3000` in your browser
3. Make changes to TypeScript files
4. Refresh browser to see changes

Port Configuration

The port can be configured in `game.json`:

```
{
  "devServer": {
    "port": 3000
  }
}
```

---

ori build

Syntax: `ori build [options]`

Builds the game for production deployment.

Arguments

None

Options

Currently no additional options. Future versions may support:

- `--output <dir>` - Custom output directory
- `--minify` - Minify output
- `--sourcemap` - Generate source maps

Description

Creates a production-ready build of your game. The build process:

- Compiles TypeScript to JavaScript
- Copies all assets (sprites, rooms, etc.)
- Generates optimized output
- Creates deployment-ready files

Note: In the current MVP, this performs a simple file copy. Future versions will include bundling and minification.

Examples

```
# Build for production
ori build

# Output:
# Building Origami Engine project...
#
# ✓ Compiled TypeScript
# ✓ Copied assets
# ✓ Generated output
#
# Build complete! Output in: ./build/
```

Output Structure

```
build/
├── assets/
│   ├── sprites/
│   └── rooms/
├── js/
│   └── game.js
├── index.html
└── game.json
```

Deployment

After building, deploy the `build/` directory to any static file host:

- **GitHub Pages**: Push `build/` to gh-pages branch
- **Netlify**: Drag and drop `build/` folder
- **Vercel**: Deploy from repository
- **itch.io**: Zip `build/` and upload as HTML5 game

---

ori help

Syntax: `ori help` or `ori --help` or `ori -h`

Displays help information for the CLI tool.

Arguments

None

Description

Shows available commands and usage information.

Output

```
Origami Engine CLI - Create GMS-style games with TypeScript

Usage:
  ori <command> [options]

Commands:
  create <project-name>   Create a new game project
  dev                     Start development server
  build                   Build game for production
  help                    Show this help message

Examples:
  ori create my-platformer
  ori dev
  ori build

For more information, visit: https://github.com/yourusername/origami-engine
```

Project Structure

Standard Project Layout

```
my-game/
├── objects/              # Game object TypeScript classes
│   ├── obj_player.ts
│   └── obj_enemy.ts
│
├── sprites/              # Sprite assets and metadata
│   ├── spr_player/
│   │   ├── metadata.json
│   │   ├── frame_0.png
│   │   └── frame_1.png
│   └── spr_enemy/
│       ├── metadata.json
│       └── frame_0.png
│
├── rooms/                # Room definition JSON files
│   ├── room_level1.json
│   └── room_level2.json
│
├── src/                  # Source code
│   └── main.ts          # Game entry point
│
├── game.json            # Game configuration
├── index.html           # HTML entry point
├── package.json         # npm configuration
├── tsconfig.json        # TypeScript configuration
└── .gitignore           # Git ignore rules
```

File Descriptions

game.json

Central configuration file for your game.

Structure:

```
{
  "title": "My Platformer",
  "width": 640,
  "height": 480,
  "backgroundColor": "#000000",
  "startRoom": "room_level1",
  "devServer": {
    "port": 3000
  }
}
```

Properties:

- `title` - Game window title
- `width` - Game canvas width
- `height` - Game canvas height
- `backgroundColor` - Default background color (hex)
- `startRoom` - Name of first room to load
- `devServer.port` - Development server port

---

package.json

npm package configuration with scripts.

Generated Structure:

```
{
  "name": "my-platformer",
  "version": "1.0.0",
  "type": "module",
  "scripts": {
    "start": "ori dev",
    "dev": "ori dev",
    "build": "ori build"
  },
  "dependencies": {
    "origami-runtime": "^0.1.0"
  },
  "devDependencies": {
    "origami-cli": "^0.1.0",
    "typescript": "^5.3.3"
  }
}
```

Scripts:

- `npm start` or `npm run dev` - Start development server
- `npm run build` - Build for production

---

tsconfig.json

TypeScript compiler configuration.

Generated Structure:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "ES2020",
    "lib": ["ES2020", "DOM"],
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "outDir": "./dist",
    "rootDir": "./src"
  },
  "include": ["src/**/*", "objects/**/*"],
  "exclude": ["node_modules", "dist", "build"]
}
```

Key Settings:

- `strict: true` - Enables all strict type checking
- `target: ES2020` - Modern JavaScript output
- `module: ES2020` - ES modules for browser

---

index.html

HTML entry point for the game.

Generated Structure:

```
<!DOCTYPE html>
<html lang="en">
<head>
  <meta charset="UTF-8">
  <meta name="viewport" content="width=device-width, initial-scale=1.0">
  <title>My Platformer</title>
  <style>
    body {
      margin: 0;
      padding: 0;
      background: #000;
      display: flex;
      justify-content: center;
      align-items: center;
      min-height: 100vh;
    }
    canvas {
      border: 1px solid #333;
      image-rendering: pixelated;
      image-rendering: crisp-edges;
    }
  </style>
</head>
<body>
  <canvas id="game-canvas"></canvas>
  <script type="module" src="./src/main.js"></script>
</body>
</html>
```

---

src/main.ts

Game initialization and entry point.

Generated Structure:

```
import { GameEngine } from 'origami-runtime';
import { obj_player } from '../objects/obj_player.js';
import { obj_wall } from '../objects/obj_wall.js';
import { obj_collectible } from '../objects/obj_collectible.js';
import { obj_enemy } from '../objects/obj_enemy.js';

// Register game objects
const objectTypes = {
  obj_player,
  obj_wall,
  obj_collectible,
  obj_enemy
};

// Load game configuration
const response = await fetch('./game.json');
const gameConfig = await response.json();

// Initialize engine
const canvas = document.getElementById('game-canvas') as HTMLCanvasElement;
const engine = new GameEngine({
  canvas,
  width: gameConfig.width,
  height: gameConfig.height,
  backgroundColor: gameConfig.backgroundColor,
  startRoom: gameConfig.startRoom
});

// Register objects with engine
for (const [name, ObjectClass] of Object.entries(objectTypes)) {
  engine.registerObject(name, ObjectClass);
}

// Start the game
await engine.start();
```

Configuration

Game Configuration (game.json)

Complete list of supported configuration options:

```
{
  "title": "My Game",
  "width": 640,
  "height": 480,
  "backgroundColor": "#000000",
  "startRoom": "room_level1",

  "devServer": {
    "port": 3000,
    "host": "localhost"
  },

  "build": {
    "outputDir": "build",
    "minify": false,
    "sourceMaps": true
  }
}
```

TypeScript Configuration

Recommended settings for Origami Engine projects:

```
{
  "compilerOptions": {
    "target": "ES2020",
    "module": "ES2020",
    "lib": ["ES2020", "DOM"],
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "moduleResolution": "node"
  }
}
```

Important: Always use `strict: true` to catch type errors early.

Template System

Default Template: Platformer

The default template includes a complete working platformer game demonstrating all engine features.

Included Objects

obj_player.ts - Player Controller

```
- WASD movement
- Space to jump
- Gravity physics
- Wall collision
- Ground detection
```

obj_wall.ts - Solid Platform

```
- Static obstacle
- Used for collision
```

obj_collectible.ts - Collectible Item

```
- Collision detection
- Respawn after delay
- Score tracking
```

obj_enemy.ts - Enemy AI

```
- Back-and-forth patrol
- Wall detection
- Edge detection
```

Template Modification

Future versions will support custom templates:

```
# Use custom template (future)
ori create my-game --template rpg
ori create my-game --template shooter
```

Workflow Guide

Creating Your First Game

Step 1: Create Project

```
ori create my-first-game
cd my-first-game
npm install
```

Step 2: Test Template

```
npm start
```

Open `http://localhost:3000` and play the platformer example.

Step 3: Modify Player

Edit `objects/obj_player.ts`:

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

  // Change movement speed
  const moveSpeed = 6; // Was 4
  if (keyboard_check(vk_d)) {
    this.hspeed = moveSpeed;
  }
}
```

Refresh browser to see changes.

Step 4: Add New Object

Create `objects/obj_powerup.ts`:

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

export class obj_powerup extends GameObject {
  create(): void {
    this.sprite_index = 'spr_powerup';
  }

  step(): void {
    const player = instance_place.call(this, this.x, this.y, 'obj_player');
    if (player) {
      // Give player power-up
      instance_destroy.call(this);
    }
  }
}
```

Step 5: Add Sprite

1. Create folder: `sprites/spr_powerup/`
2. Add `frame_0.png` (your sprite image)

3. Create `metadata.json`:

```
{
  "origin": { "x": 8, "y": 8 },
  "fps": 0
}
```

Step 6: Register 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 here
};
```

Step 7: Add to Room

Edit `rooms/room_level1.json`:

```
{
  "instances": [
    { "object": "obj_player", "x": 100, "y": 100 },
    { "object": "obj_powerup", "x": 300, "y": 200 }
  ]
}
```

Step 8: Build for Production

```
npm run build
```

Deploy the `build/` folder to your web host.

Development Tips

Hot Reloading (Manual)

Since auto-reload isn't available, use this workflow:

1. Keep browser and editor side-by-side
2. Edit code
3. Alt+Tab to browser
4. Press F5 to refresh

Debug Mode

Press F3 in-game to toggle debug overlay showing:

- FPS counter
- Instance count
- Collision boxes (color-coded)
- View position

Console Logging

Use `show_debug_message` for logging:

```
show_debug_message.call(this, `Player pos: ${this.x}, ${this.y}`);
```

Output format: `[obj_player] Player pos: 100, 200`

Common Tasks

Adding a New Room

1. Create `rooms/room_level2.json`:

```
{
  "name": "room_level2",
  "width": 1024,
  "height": 768,
  "speed": 60,
  "backgroundColor": "#87CEEB",
  "instances": [],
  "views": [{
    "enabled": true,
    "x": 0,
    "y": 0,
    "width": 640,
    "height": 480,
    "object": "obj_player",
    "hborder": 200,
    "vborder": 150
  }]
}
```

2. Transition in code:

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

Changing Game Resolution

Edit `game.json`:

```
{
  "width": 1280,
  "height": 720
}
```

The canvas automatically scales to fit the browser window.

Publishing to itch.io

1. Build your game:

```
npm run build
```

2. Create a zip file of the `build/` folder
3. Upload to itch.io:
    - Select "HTML5" project type
    - Upload zip file
    - Set "This file will be played in the browser"
    - Set viewport dimensions to match `game.json`

Troubleshooting

Command Not Found: ori

Problem: `ori: command not found`

Solution:

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

# Or use with npx
npx origami-cli create my-game
```

TypeScript Errors

Problem: Type errors when running project

Solution:

1. Ensure `strict: true` in tsconfig.json
2. Declare all custom properties in your classes

3. Use proper types for function parameters

```
// Bad
health = 100;

// Good
private health: number = 100;
```

Assets Not Loading

Problem: Sprites don't appear, console shows 404 errors

Solution:

1. Verify sprite folder structure:

```
sprites/spr_name/
  ├── metadata.json
  └── frame_0.png
```

2. Check metadata.json format
3. Ensure PNG files are named `frame_0.png`, `frame_1.png`, etc.
4. Verify sprite name matches in object code

Dev Server Won't Start

Problem: `Error: listen EADDRINUSE: address already in use`

Solution:

1. Port 3000 is already in use
2. Change port in `game.json`:

```
{
  "devServer": {
    "port": 3001
  }
}
```

3. Or stop the other process using port 3000

Build Fails

Problem: Build command fails with errors

Solution:

1. Run `npm install` to ensure dependencies are installed
2. Delete `node_modules` and `package-lock.json`, then reinstall
3. Check TypeScript compilation: `npx tsc --noEmit`
4. Fix any TypeScript errors before building

Advanced Usage

Custom Build Scripts

Add custom scripts to `package.json`:

```
{
  "scripts": {
    "start": "ori dev",
    "build": "ori build",
    "clean": "rm -rf dist build",
    "deploy": "npm run build && netlify deploy --prod --dir=build"
  }
}
```

Environment-Specific Configuration

Create multiple config files:

- `game.json` - Development config
- `game.production.json` - Production config

Use build scripts to swap configs.

Integration with Game Engines

The Origami CLI output is standard HTML5, so it can be embedded in:

- Electron apps (desktop games)
- Cordova/Capacitor (mobile apps)
- Custom web frameworks

Future Features

Planned features for future versions:

- **Asset Pipeline** - Image optimization, sprite atlas generation
- **Code Bundling** - Webpack/Rollup integration for smaller builds
- **Live Reload** - Auto-refresh on file changes
- **Multiple Templates** - RPG, shooter, puzzle templates
- **Plugin System** - Custom CLI plugins
- **Cloud Deploy** - One-command deployment to hosting services
- **Project Migration** - Import from GameMaker Studio projects

---

Version: 0.1.0 MVP **License**: MIT **Repository**: https://github.com/yourusername/origami-engine

---

Sources:

- GameMaker Studio 1.4.9999 Documentation
- GameMaker Docs Repository

- GameMaker Manual