CLI Templates

Template system and customization

Overview

Origami Engine provides project templates for quick setup. Templates are stored as git branches and fetched during project creation.

Built-in Templates

Fresh Template

Name: `fresh`

Description: Minimal starter project with empty structure.

Includes:

- Empty `objects/` folder
- Empty `sprites/` folder
- Single `rooms/rm_main.json` room
- Basic `src/main.ts` entry point
- Default `game.json` config

Use Case: Starting from scratch

Create Project:

```
ori create my-game --template fresh
```

Platformer Template

Name: `platformer`

Description: Complete platformer game with player, enemies, and level.

Includes:

- Player object with WASD movement and jumping
- Wall and platform objects
- Enemy with AI
- Coin collectibles
- Multiple rooms (menu, level1, level2)
- Sprite assets
- Full game loop

Use Case: Learning or building upon existing game

Create Project:

```
ori create my-game --template platformer
```

Template Structure

Template Branch

Templates are stored as git branches in the engine repository:

```
template/fresh       → Fresh template
template/platformer  → Platformer template
```

Template Contents

Each template branch contains a complete project structure:

```
template/platformer/
├── game.json               # Game config
├── package.json            # Dependencies
├── tsconfig.json           # TypeScript config
├── index.html              # Entry HTML
├── src/
│   └── main.ts             # Main entry point
├── objects/
│   ├── obj_player.ts       # Player object
│   ├── obj_wall.ts         # Wall object
│   └── obj_enemy.ts        # Enemy object
├── sprites/
│   ├── spr_player/
│   │   ├── 0.png
│   │   └── metadata.json
│   ├── spr_wall/
│   │   └── 0.png
│   └── spr_enemy/
│       ├── 0.png
│       └── metadata.json
├── rooms/
│   ├── rm_menu.json
│   └── rm_level1.json
└── README.md               # Template-specific docs
```

Using Templates

Interactive Selection

```
ori create
```

Prompt:

```
? Select a template:
  ❯ fresh - Minimal starter project
    platformer - Complete platformer game
```

Direct Template Selection

```
ori create my-game --template platformer
```

List Available Templates

```
ori templates list
```

Output:

```
Available templates:

  fresh
  Minimal starter project with empty structure

  platformer
  Complete platformer game with player, enemies, and level
```

Creating Custom Templates

Step 1: Create Template Branch

Create a new git branch for your template:

```
cd origami-engine
git checkout -b template/my-template
```

Step 2: Set Up Project Structure

Create a complete project structure:

```
# Create folders
mkdir -p objects sprites rooms src

# Create game config
cat > game.json << 'EOF'
{
  "name": "My Template",
  "version": "1.0.0",
  "author": "Your Name",
  "description": "Custom template description",
  "entryRoom": "rm_main",
  "engineVersion": "0.1.0"
}
EOF

# Create TypeScript config
cat > tsconfig.json << 'EOF'
{
  "compilerOptions": {
    "target": "ES2022",
    "module": "ES2022",
    "moduleResolution": "node",
    "lib": ["ES2022", "DOM"],
    "strict": true,
    "esModuleInterop": true,
    "skipLibCheck": true,
    "outDir": "./dist",
    "rootDir": "./src"
  },
  "include": ["src/**/*", "objects/**/*"],
  "exclude": ["node_modules", "dist"]
}
EOF

# Create entry point
cat > src/main.ts << 'EOF'
import { Runtime } from 'origami-runtime';

const runtime = new Runtime();
runtime.start();
```

```
EOF

# Create README
cat > README.md << 'EOF'
# My Custom Template

Description of your template.

## Features
- Feature 1
- Feature 2

## Quick Start
See main documentation.
EOF
```

Step 3: Add Template Objects

Create sample objects:

objects/obj_example.ts:

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

export class obj_example extends GameObject {
  create(): void {
    this.x = 100;
    this.y = 100;
  }

  step(): void {
    // Template logic
  }

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

Step 4: Add Sprites

Create sprite folders with assets:

```
mkdir -p sprites/spr_example
# Add sprite images (0.png, 1.png, etc.)

cat > sprites/spr_example/metadata.json << 'EOF'
{
  "origin": { "x": 16, "y": 16 },
  "fps": 10
}
EOF
```

Step 5: Create Rooms

rooms/rm_main.json:

```
{
  "name": "rm_main",
  "width": 640,
  "height": 360,
  "backgroundColor": "#2d2d2d",
  "instances": [
    {
      "type": "obj_example",
      "x": 100,
      "y": 100
    }
  ]
}
```

Step 6: Commit Template

```
git add .
git commit -m "Add my-template"
git push origin template/my-template
```

Step 7: Register Template

Edit `.origami/config.json` on main branch:

```
{
  "templateBranches": {
    "fresh": "template/fresh",
    "platformer": "template/platformer",
    "my-template": "template/my-template"
  }
}
```

Step 8: Test Template

```
git checkout main
ori create test-game --template my-template
```

Template Best Practices

Include README

Each template should have a README explaining:

- What the template includes
- How to use it
- Key features
- Customization tips

Provide Examples

Include well-commented code:

```
export class obj_player extends GameObject {
  // Movement speed
  private readonly SPEED = 4;
  // Jump force
  private readonly JUMP_FORCE = 10;

  step(): void {
    // WASD movement
    if (keyboard_check(vk_d)) {
      this.x += this.SPEED;
    }
    if (keyboard_check(vk_a)) {
      this.x -= this.SPEED;
    }

    // Jump on space
    if (keyboard_check_pressed(vk_space) && this.onGround) {
      this.vspeed = -this.JUMP_FORCE;
    }
  }
}
```

Use Descriptive Names

Good:

- `obj_player` - Player character
- `obj_enemy_walker` - Walking enemy
- `spr_player_idle` - Player idle sprite

Bad:

- `obj_1` - Unclear purpose
- `test` - Not descriptive

- `thing` - Vague

Include Assets

Provide placeholder sprites if needed:

```
sprites/
├── spr_placeholder_16x16/
│   └── 0.png           # 16x16 colored square
├── spr_placeholder_32x32/
│   └── 0.png           # 32x32 colored square
```

Test Thoroughly

Before publishing:

1. Create project from template
2. Run `ori dev`
3. Test all features
4. Fix any errors
5. Update documentation

Template Metadata

Template Info File

File: `.template.json` (in template branch)

```
{
  "name": "My Custom Template",
  "description": "A template for building XYZ games",
  "author": "Your Name",
  "version": "1.0.0",
  "tags": ["rpg", "topdown", "multiplayer"],
  "features": [
    "Player movement",
    "Inventory system",
    "Quest system"
  ],
  "dependencies": {
    "origami-runtime": "^0.1.0"
  },
  "screenshots": [
    "screenshots/1.png",
    "screenshots/2.png"
  ]
}
```

Display Template Info

```
ori templates info platformer
```

Output:

```
Template: platformer
Version: 1.0.0
Author: Origami Engine Team

Description:
  Complete platformer game with player, enemies, and level.

Features:
  - WASD + Space controls
  - Collision detection
  - Enemy AI
  - Score system
  - Multiple rooms

Dependencies:
  - origami-runtime ^0.1.0
```

Template Updates

Updating a Template

1. Checkout template branch:

   ```
   git checkout template/platformer
   ```

2. Make changes:

   ```
   # Edit files
   nano objects/obj_player.ts
   ```

3. Commit and push:

   ```
   git add .
   git commit -m "Update player movement"
   git push origin template/platformer
   ```

4. Update version in `.template.json`:

```
{
  "version": "1.1.0"
}
```

User Updates

Users get template updates by:

1. Creating new project from updated template
2. Manually copying files
3. Using `ori update` (for engine changes)

Note: Template updates don't auto-apply to existing projects

Template Marketplace

Share Your Template

1. Fork Origami Engine repo
2. Create template branch
3. Submit pull request
4. Include:
    - Template code
    - Screenshots
    - Documentation
    - `.template.json` metadata

Community Templates

Browse community templates:

```
ori templates browse
```

Output:

```
Community Templates:

  rpg-starter (by user123)
  ⭐ 45  📥 120
  Basic RPG with inventory and quests

  topdown-shooter (by user456)
  ⭐ 32  📥 89
  Top-down shooter with weapons system
```

Install Community Template

```
ori templates install rpg-starter
ori create my-rpg --template rpg-starter
```

Template Variables

Variable Substitution

Templates support variable substitution during project creation.

In template files:

```
{
  "name": "{{PROJECT_NAME}}",
  "author": "{{AUTHOR}}",
  "description": "{{DESCRIPTION}}"
}
```

CLI prompts for values:

```
ori create
? Project name: my-game
? Author: John Doe
? Description: A fun game
```

Result (`game.json`):

```
{
  "name": "my-game",
  "author": "John Doe",
  "description": "A fun game"
}
```

Available Variables

- `{{PROJECT_NAME}}` - Project name
- `{{AUTHOR}}` - Author name
- `{{DESCRIPTION}}` - Project description
- `{{DATE}}` - Current date (YYYY-MM-DD)
- `{{YEAR}}` - Current year
- `{{ENGINE_VERSION}}` - Engine version

Advanced Templates

Multi-File Generation

Template: `objects/obj_{{NAME}}.ts`

Create:

```
ori create my-game --template platformer
```

Variables during creation:

```
? Additional objects to create: player, enemy, coin
```

Result: Creates `obj_player.ts`, `obj_enemy.ts`, `obj_coin.ts`

Conditional Includes

File: `.template.json`

```
{
  "conditionalFiles": {
    "multiplayer": [
      "objects/obj_network_manager.ts",
      "src/network.ts"
    ],
    "audio": [
      "sounds/",
      "objects/obj_audio_manager.ts"
    ]
  }
}
```

Prompt:

```
ori create
? Include multiplayer support? Yes
? Include audio system? No
```

Result: Only includes multiplayer files

Template Debugging

Validate Template

```
ori templates validate template/my-template
```

Output:

```
✅ Template structure valid
✅ All required files present
✅ game.json valid
✅ tsconfig.json valid
❌ Missing sprites/spr_player/0.png
⚠️  No README.md found
```

Test Template Locally

```
# Create test project
ori create test --template my-template

# Run and verify
cd test
ori dev
```

Next Steps

- **30-cli-commands.md** - CLI commands
- **31-cli-configuration.md** - Configuration
- **02-installation.md** - Installation guide

← Back to Index