Contributing

How to contribute to the Origami Engine

Welcome Contributors!

Origami Engine is an open-source project, and contributions are welcome! Whether you're fixing bugs, adding features, improving documentation, or creating examples, your help is appreciated.

Getting Started

Prerequisites

Before contributing, ensure you have:

- **Node.js** 18.0.0 or higher
- **pnpm** 8.0.0 or higher
- **Git**
- Basic knowledge of TypeScript
- Familiarity with GameMaker Studio 1.4 (helpful but not required)

Setting Up Development Environment

1. **Fork the repository** on GitHub
2. **Clone your fork**:

```
git clone https://github.com/YOUR-USERNAME/origami-engine
cd origami-engine
```

3. **Install dependencies**:

```
pnpm install
```

4. **Build all packages**:

```
pnpm build
```

5. **Test the platformer example**:

```
cd platformer
# Open index.html in browser
```

6. **Create a feature branch**:

```
git checkout -b feature/my-new-feature
```

Project Structure

```
Origami-Engine/
├── packages/
│   ├── runtime/          # Core game engine
│   │   ├── src/          # TypeScript source
│   │   ├── dist/         # Compiled output
│   │   └── DOCUMENTATION.md
│   └── cli/              # CLI tool
│       ├── src/
│       ├── dist/
│       └── DOCUMENTATION.md
├── platformer/           # Example game
├── docs/                 # Documentation
│   ├── md/              # Markdown sources
│   └── pdf/             # Generated PDFs
├── scripts/             # Build and utility scripts
└── README.md
```

Key Directories

- **packages/runtime/src/** - Engine core (GameObject, collision, rendering, etc.)
- **packages/cli/src/** - CLI commands (create, update, build, etc.)
- **docs/md/** - Documentation markdown files
- **platformer/** - Working example game for testing

Types of Contributions

🐛 Bug Fixes

Found a bug? Great!

1. **Check existing issues** - Search to see if it's already reported
2. **Create an issue** - Describe the bug with:
    - Steps to reproduce
    - Expected vs actual behavior
    - Browser/OS information
    - Code samples if applicable
3. **Submit a PR** - Reference the issue number

Example:

```
Fix collision detection with rotated sprites (#123)

- Updated bounding box calculation to account for image_angle
- Added test cases for rotated collisions
- Fixes #123
```

✨ New Features

Want to add a feature?

1. **Open a discussion first** - Create an issue with the `enhancement` label
2. **Get feedback** - Make sure the feature aligns with project goals
3. **Implement** - Follow coding standards
4. **Document** - Update relevant documentation

5. **Submit PR** - Include examples of the new feature

Guidelines:

- Keep features aligned with GameMaker Studio 1.4 philosophy
- Maintain backward compatibility when possible
- Add TypeScript types for all new APIs
- Include JSDoc comments

📚 Documentation

Documentation improvements are always welcome!

Areas to contribute:

- Fix typos or unclear explanations
- Add more examples
- Create tutorials
- Improve API reference
- Add screenshots/diagrams

Documentation standards:

- Files in `docs/md/` use numbered prefixes (01-99)
- Keep files under 300 lines
- One topic per file
- Include code examples
- Cross-reference related docs

🎮 Example Games

Create example games to showcase features:

Requirements:

- Working game demonstrating a concept
- Clean, commented code
- README explaining the example
- Appropriate license

Examples needed:

- Top-down shooter

- Puzzle game
- Tower defense
- Endless runner

🎨 Templates

New project templates for `ori create`:

Requirements:

- Complete game structure
- Documented code
- sprites/ with all assets
- rooms/ with example levels
- README with instructions

Coding Standards

TypeScript

```
// ✅ GOOD
export class GameObject {
  private health: number = 100;

  public takeDamage(amount: number): void {
    this.health -= amount;
  }
}

// ❌ BAD
export class GameObject {
  health = 100; // Should be private

  takeDamage(amount) { // Missing types
    this.health -= amount;
  }
}
```

Rules:

- Use **strict mode** TypeScript
- Explicit return types on all functions
- Use `private` / `public` / `protected` modifiers
- No `any` types (use `unknown` if necessary)
- JSDoc comments for public APIs

Naming Conventions

```
// Classes: PascalCase
class GameObject { }
class SpriteManager { }

// Functions: camelCase
function instanceCreate() { }
function drawSprite() { }

// Constants: UPPER_SNAKE_CASE
const MAX_INSTANCES = 1000;
const DEFAULT_FPS = 60;

// Private members: camelCase with underscore prefix
private _cachedSprite: Sprite | null = null;
```

File Organization

```
packages/runtime/src/
├── core/
│   ├── GameObject.ts
│   ├── GameEngine.ts
│   └── types.ts
├── systems/
│   ├── CollisionSystem.ts
│   ├── RenderSystem.ts
│   └── InputSystem.ts
├── functions/
│   ├── instance.ts
│   ├── collision.ts
│   └── drawing.ts
└── index.ts
```

Testing

Manual Testing

Before submitting a PR:

1. **Test the platformer example**:

   ```
   cd platformer
   # Open index.html
   # Play through the level
   # Press F3 to check FPS
   ```

2. **Create a test game**:

```
ori create test-game
cd ../test-game
npm install
npm run build
# Test your changes
```

3. **Test in multiple browsers**:
    - Chrome/Edge
    - Firefox
    - Safari (if on macOS)

Automated Tests

Currently, the project uses manual testing. Contributions to add automated tests are welcome!

Future test areas:

- Unit tests for core functions
- Integration tests for game loop
- Visual regression tests for rendering

Pull Request Process

Before Submitting

- ✅ Code follows style guidelines
- ✅ All builds complete successfully (`pnpm build`)
- ✅ Tested manually in platformer example
- ✅ Documentation updated (if applicable)
- ✅ Commit messages are clear and descriptive

PR Template

```
## Description
Brief description of changes

## Type of Change
- [ ] Bug fix
- [ ] New feature
- [ ] Documentation
- [ ] Example/template

## Related Issues
Fixes #123

## Testing
Describe how you tested the changes

## Screenshots (if applicable)
Add screenshots for visual changes
```

Review Process

1. **Automated checks** - Builds must pass
2. **Code review** - Maintainer reviews code quality
3. **Testing** - Maintainer tests functionality
4. **Merge** - PR merged to main branch

Response time: We aim to respond within 1 week.

Git Commit Guidelines

Commit Message Format

```
<type>: <short summary>

<optional detailed description>

<optional footer>
```

Types

- `feat:` New feature
- `fix:` Bug fix
- `docs:` Documentation changes
- `refactor:` Code refactoring
- `perf:` Performance improvements
- `test:` Adding tests
- `chore:` Build/tooling changes

Examples

```
# Good commits
feat: add audio system with sound and music support
fix: collision detection with rotated sprites
docs: add performance optimization guide
refactor: simplify sprite loading logic

# Bad commits
update stuff
fixes
changes
wip
```

Building the Project

Full Build

```
# Build all packages
pnpm build

# Build runtime only
cd packages/runtime
pnpm build

# Build CLI only
cd packages/cli
pnpm build
```

Development Workflow

```
# Watch mode (auto-rebuild on changes)
cd packages/runtime
pnpm dev

# In another terminal
cd platformer
# Refresh browser after changes
```

Clean Build

```
# Remove all build artifacts
pnpm clean

# Rebuild everything
pnpm install
pnpm build
```

Documentation Contributions

Writing Documentation

Documentation lives in `docs/md/`:

```
# Create new doc file
docs/md/25-api-motion.md

# Follow numbering scheme:
# 01-19: Getting started
# 20-29: API reference
# 30-39: CLI tools
# 40-49: Advanced topics
# 90-99: Contributing
```

Documentation Style

- **Headings**: Use sentence case
- **Code blocks**: Always specify language (```typescript)
- **Examples**: Include complete, working examples
- **Links**: Use relative links to other docs
- **Formatting**: Keep lines under 120 characters

Generating PDFs

```
# Generate PDFs from markdown
cd scripts
node generate-pdfs.js
```

Community Guidelines

Code of Conduct

- Be respectful and inclusive

- Provide constructive feedback
- Focus on what's best for the project
- Respect maintainer decisions

Getting Help

- **Questions**: Open a GitHub Discussion
- **Bugs**: Create an Issue
- **Chat**: (Discord/Slack link if available)

Recognition

Contributors are recognized in:

- GitHub contributors list
- Release notes for significant contributions
- Documentation credits (for major doc contributions)

License

By contributing, you agree that your contributions will be licensed under the MIT License.

Resources

- **GitHub Repository**
- **Issue Tracker**
- **Discussions**
- **91-architecture.md** - Engine design details

Thank You!

Every contribution, no matter how small, helps make Origami Engine better. Thank you for being part of the community!

← Back to Index