软件工程化说明文档

本项目在开发与部署过程中，采用了多项现代软件工程化与自动化手段，极大提升了开发效率、协作体验和代码质量。具体实践如下：

1. 自动化构建（Rollup）

项目采用 Rollup 作为构建工具，自动完成源码的打包、压缩和优化。

- 配置文件：rollup.config.js
- 构建命令：

```
npm run build
```

- 支持 Svelte、PostCSS、TailwindCSS 等现代前端技术栈。
- 生产环境下自动压缩代码，提取关键 CSS，提升加载性能。

2. 持续集成与自动化部署（GitHub Actions）

项目集成了 GitHub Actions，实现自动化测试与部署流程。

- 配置文件：.github/workflows/deploy.yml
- 主要流程：
  1. 代码推送到 master 分支后自动触发。
  2. 自动安装依赖、构建项目、运行测试。
  3. 构建产物自动部署到 GitHub Pages。
- 关键步骤如下：

```
- name: Install ♻️
  run: npm install
- name: Build 🏭
  run: npm run build
- name: 运行测试
  run: npm test
- name: Deploy 🚀
  uses: JamesIves/github-pages-deploy-action@v4.6.1
  with:
    folder: dist
```

3. 依赖管理（npm）

项目使用 npm 进行依赖管理，所有依赖均声明在 package.json 文件中。

- 统一依赖版本，便于团队协作和环境一致性。
- 支持一键安装和升级依赖：

```
npm install
```

- 支持开发依赖与生产依赖分离，提升构建效率。

4. 模块化开发

项目采用模块化目录结构，便于功能解耦和多人协作。

- 主要源码位于 src 目录下，按功能划分为 components、libs、styles 等子目录。
- 业务逻辑、UI 组件、工具库等均独立封装，提升代码复用性和可维护性。
- 通过 alias（如 @sudoku/xxx）简化模块导入路径，见 rollup.config.js 的 alias 配置。

5. 单元测试（Jest）

为保障核心逻辑的正确性和稳定性，项目引入了单元测试。

- **测试框架**：采用 Jest 流行的测试框架，对关键模块进行测试。
- **测试内容**：重点测试数独生成、求解、验证等核心算法的准确性，以及关键 UI 组件的交互行为。
- **自动化执行**：在持续集成流程中，`npm test` 命令会自动执行所有测试用例。只有当所有测试通过后，部署流程才会继续，有效防止了有问题的代码被合并到主分支或部署到生产环境。

6. 团队协作流程

项目通过标准化的协作流程，确保多人开发时的高效与协同。

- **版本控制**：使用 Git 进行版本控制，所有开发工作在独立的特性分支（Feature Branch）上进行，避免直接修改 `main` 分支。
- **编码规范**：项目配置了 ESLint 和 Prettier，在代码提交前自动格式化代码、检查语法错误，确保整个项目的代码风格保持一致，提升了代码的可读性。

总结

通过上述软件工程化与自动化实践，项目实现了高效的开发、测试、构建与部署流程。从模块化开发、自动化构建到持续集成与部署，再到全面的单元测试和标准化的团队协作规范，这一系列措施共同保障了项目的代码质量、稳定性和团队的协作效率。