逆向分析

李菁菁 25214478

核心设计分析（OOA/OOD/OOP）

愿景

打造一款轻量、美观、跨端可用的在线数独游戏，基于 Svelte + TailwindCSS 实现高性能渲染与简洁交互，核心满足玩家 “快速启动、流畅游玩、灵活分享” 的基础需求，预留撤销 / 重做、自定义谜题等扩展能力，定位为 “无冗余依赖、适配多场景的休闲益智工具”。

用例分析

用例 1：生成数独棋盘

- 参与者：玩家
- 简要描述：玩家通过菜单栏选择难度等级（或默认难度），系统根据难度规则生成含唯一解的初始数独棋盘，支持新游戏启动或当前游戏结束后重新生成。
- 前置条件：
    - 游戏已成功加载（页面初始化完成，Svelte 根组件挂载成功）；
    - 玩家处于游戏主界面（无未关闭的弹窗遮挡）或当前游戏已完成状态；
    - 系统数独生成算法（第三方库 / 内置逻辑）可用。
- 后置条件：
    - 成功：系统生成符合难度的数独棋盘（预设固定数字 + 空白可填格），棋盘渲染完成，游戏进入 “进行中” 状态，计时器（若启用）开始计时；
    - 失败：未生成有效棋盘（如算法异常、参数错误），系统弹出文字提示 “生成失败，请重试”，玩家停留在原界面，可再次触发生成操作。
- 基本事件流：
    - 玩家进入游戏页面后，系统默认加载 Medium 难度棋盘（已实现默认生成逻辑）；
    - 若玩家需切换难度，点击页面顶部 “Difficulty” 下拉菜单；
    - 系统展示可选难度：Very Easy、Easy、Medium、Hard；
    - 玩家选择目标难度等级；
    - 系统接收难度参数，调用第三方生成库，生成含唯一解的初始棋盘（固定数字深色显示，空白格浅色背景）；
    - 系统清空上一轮游戏的填写记录，重置棋盘显示状态；
    - 系统渲染新棋盘，游戏进入 “进行中” 状态；用例结束。
- 异常事件流 1：难度选择取消
    - 偏离步骤：基本事件流的步骤 4；
    - 玩家点击下拉菜单外区域关闭菜单，未选择新难度；
    - 系统不生成新棋盘，保持当前难度与棋盘状态；用例结束。

用例 2：填写单元格数字

- 用例名称：填写单元格数字
- 参与者：玩家

- 简要描述：玩家选中棋盘空白格后，通过键盘输入 1-9 数字或点击页面底部数字键盘选择数字，系统验证数字合法性（行 / 列 / 九宫格无重复），有效数字正常显示，无效数字以红色提示。
- 前置条件： 数独棋盘已成功生成，游戏处于 “进行中” 状态；目标单元格为空白可填格（非系统预设固定数字）；玩家已选中目标单元格（单元格边框高亮显示）。
- 后置条件：
  - 输入有效：单元格显示输入数字（黑色字体），系统记录当前填写状态；
  - 输入无效：单元格显示输入数字（红色字体），提示违反数独规则；
  - 输入为空：单元格清空已填数字（仅针对玩家手动填写内容）。
- 基本事件流：
  - 玩家点击棋盘上的空白单元格，系统标记该单元格为 “选中状态”（边框高亮）；
  - 玩家通过两种方式输入数字：① 键盘直接输入 1-9 数字；② 点击页面底部数字键盘按钮；
  - 系统接收输入数字，触发合法性校验：当前行、当前列、所属 3x3 九宫格内无重复数字；
  - 若校验通过：系统将单元格数字显示为黑色；
  - 若校验失败：系统将单元格数字显示为红色，持续保留提示；
  - 用例结束。
- 异常事件流 1：输入非 1-9 字符
  - 偏离步骤：基本事件流的步骤 2；
  - 玩家输入 0、字母、符号等非法字符；
  - 系统忽略该输入，单元格保持原状态，无任何提示；
  - 用例结束。
- 异常事件流 2：选中固定数字单元格
  - 偏离步骤：基本事件流的步骤 1；
  - 玩家点击系统预设的固定数字单元格；
  - 系统不标记 “选中状态”，玩家无法输入数字，单元格保持原固定值；
  - 用例结束。

用例 3：清除单元格内容

- 用例名称：清除单元格内容
- 参与者：玩家
- 简要描述：玩家选中已填写数字的单元格（非固定值），通过点击 “清除” 按钮或键盘删除键，清空单元格内容，恢复空白状态。
- 前置条件：
  - 游戏处于 “进行中” 状态；
  - 目标单元格为玩家已填写的非固定值单元格（含有效黑色数字和无效红色数字）；
  - 玩家已选中目标单元格。
- 后置条件：
  - 清除成功：单元格内容清空，恢复空白状态，红色提示（若有）消失；
  - 清除失败：单元格保持原状态（针对固定值单元格）。
- 基本事件流：
  - 玩家选中已填写数字的非固定值单元格；
  - 玩家通过两种方式触发清除：① 点击页面底部 “Clear” 按钮；② 按下键盘删除键（Backspace/Delete）；
  - 系统清空该单元格的数字内容，恢复空白状态；
  - 若单元格之前有红色无效提示，同步清除红色样式；
  - 用例结束。
- 异常事件流 1：清除固定值单元格

    - 偏离步骤：基本事件流的步骤 1；
    - 玩家选中固定值单元格，触发清除操作；
    - 系统忽略该操作，单元格保持固定数字，无任何变化；
    - 用例结束。

用例 4：重置当前游戏

- 用例名称：重置当前游戏
- 参与者：玩家
- 简要描述：玩家点击 “Reset” 按钮，系统清空所有玩家填写的数字（含有效 / 无效数字），恢复当前数独的初始状态（仅保留固定值），游戏重新进入 “进行中” 状态。
- 前置条件：
    - 数独棋盘已成功生成，游戏处于 “进行中” 状态；
    - 玩家处于游戏主界面，无未关闭的弹窗。
- 后置条件：
    - 重置成功：所有空白格恢复初始空白状态，红色提示（若有）全部清除，游戏保持当前难度不变；
    - 无失败场景（当前实现未处理异常）。
- 基本事件流：
    - 玩家点击页面顶部 “Reset” 按钮；
    - 系统接收重置指令，遍历所有单元格；
    - 系统清空非固定值单元格的所有数字内容，清除红色无效提示样式；
    - 系统保持固定值单元格不变，棋盘恢复初始生成状态；
    - 游戏继续处于 “进行中” 状态；
    - 用例结束。

用例 5：分享数独谜题

- 用例名称：分享数独谜题
- 参与者：玩家
- 简要描述：玩家点击 “Share” 按钮，系统弹出分享弹窗，支持通过二维码、复制链接、社交平台（Twitter/Facebook）、邮件等方式，分享当前数独的初始棋盘（不含玩家填写进度）。
- 前置条件：
    - 数独棋盘已成功生成（任意难度均可）；
    - 玩家处于游戏主界面，无未关闭的弹窗；
    - 网络正常（社交平台 / 邮件分享需网络支持）。
- 后置条件：
    - 分享成功：系统生成分享内容并触发对应分享行为，玩家返回原游戏界面，当前游戏状态不变；
    - 分享取消：玩家关闭分享弹窗，无分享行为，游戏状态不变。
- 基本事件流：
    - 玩家点击页面顶部 “Share” 按钮；
    - 系统弹出分享弹窗，展示分享选项：① QR Code（二维码）；② Copy Link（复制链接）；③ Twitter；④ Facebook；⑤ Email；
    - 玩家选择目标分享方式：
        - 选择 “QR Code”：系统生成含当前数独唯一标识的二维码，在弹窗中展示，玩家可截图保存；

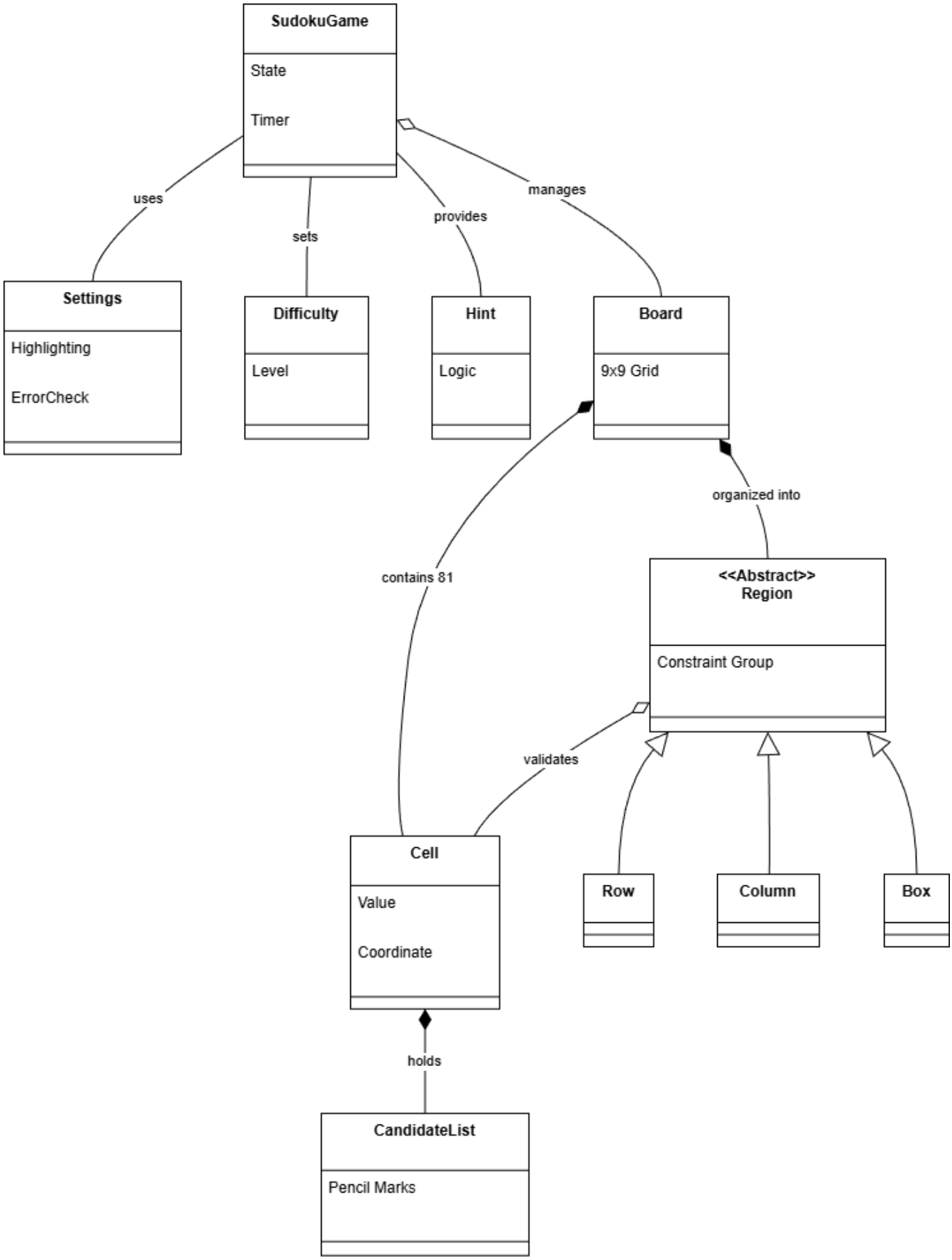
        - 选择 “Copy Link”：系统生成数独访问链接（如 https://sudoku.jonasgeiler.com/?puzzle=xxx），复制到剪贴板，弹窗提示 “Link copied!”；
        - 选择 “Twitter/Facebook”：系统跳转至对应社交平台分享界面，自动填充预设文案（如 “Check out this Sudoku puzzle!”）与访问链接；
        - 选择 “Email”：系统打开本地邮件客户端，自动填充邮件主题（“Sudoku Puzzle”）、正文（含访问链接）；
    - 分享操作完成后，玩家点击弹窗 “Close” 按钮或关闭弹窗；
    - 系统关闭分享弹窗，玩家返回原游戏界面，当前填写进度、棋盘状态保持不变；
    - 用例结束。
- 异常事件流 1：玩家取消分享
    - 偏离步骤：基本事件流的步骤 3；
    - 玩家点击分享弹窗的 “Close” 按钮或弹窗外部区域；
    - 系统关闭弹窗，不生成任何分享内容，返回原游戏界面；
    - 用例结束。
- 异常事件流 2：分享方式不可用
    - 偏离步骤：基本事件流的步骤 3；
    - 玩家选择 “Twitter/Facebook/Email” 时，网络断开或对应客户端未安装；
    - 系统无明确提示，仅不触发分享行为（当前实现未处理该异常）；
    - 玩家可重新选择其他分享方式或关闭弹窗；
    - 用例结束。

用例 6：管理个性化设置

- 用例名称：管理个性化设置
- 参与者：玩家
- 简要描述：玩家点击 “Settings” 按钮，在弹窗中配置游戏个性化选项（仅已实现的主题切换、网格显示、数字颜色等），设置实时生效，无需手动保存。
- 前置条件：
    - 游戏页面已成功加载，玩家处于任意界面；
    - 浏览器支持 CSS 变量（主题切换依赖）。
- 后置条件：
    - 设置成功：所选配置实时生效，页面样式 / 显示状态同步更新；
    - 无持久化存储（当前实现未保存设置，刷新页面后恢复默认）。
- 基本事件流：
    - 玩家点击页面顶部 “Settings” 按钮；
    - 系统弹出设置弹窗，展示已实现的配置选项：
        - Theme（主题）：Light（浅色）、Dark（深色）、System（跟随系统）；
        - Grid Lines（网格线）：On（显示）、Off（隐藏）；
        - Highlight Duplicates（高亮重复）：On（启用）、Off（禁用）；
        - Number Color（数字颜色）：Default（默认）、Colorful（彩色）；
    - 玩家按需修改配置（如选择 “Dark” 主题、“Off” 网格线）；
    - 系统实时应用修改后的配置：
    - 主题切换：页面背景色、文字色、单元格颜色同步变化；
    - 网格线切换：棋盘网格线显示 / 隐藏；
    - 高亮重复：输入无效数字时，除红色提示外，同步高亮行 / 列 / 九宫格内的重复数字；
    - 数字颜色：彩色模式下，1-9 数字分别显示不同颜色；
    - 玩家点击弹窗 “Close” 按钮或弹窗外部区域，关闭设置弹窗；

    - 用例结束。
- 异常事件流 1：恢复默认设置
    - 偏离步骤：基本事件流的步骤 3；
    - 玩家点击设置弹窗底部 “Reset to Default” 按钮；
    - 系统将所有配置恢复为初始默认值（Light 主题、显示网格线、启用高亮重复、默认数字颜色）；
    - 页面样式实时同步恢复，玩家关闭弹窗；
    - 用例结束。

领域模型



技术架构

本项目为一款前端主导的轻量级数独游戏应用，基于 Svelte 框架构建，搭配 TailwindCSS 实现样式体系管理，整体采用分层架构模式。应用以组件化、模块化设计为核心，将 UI 渲染、构建打包、缓存管理及游戏核心能力分层解耦，实现高效开发、灵活部署与优质的用户体验。

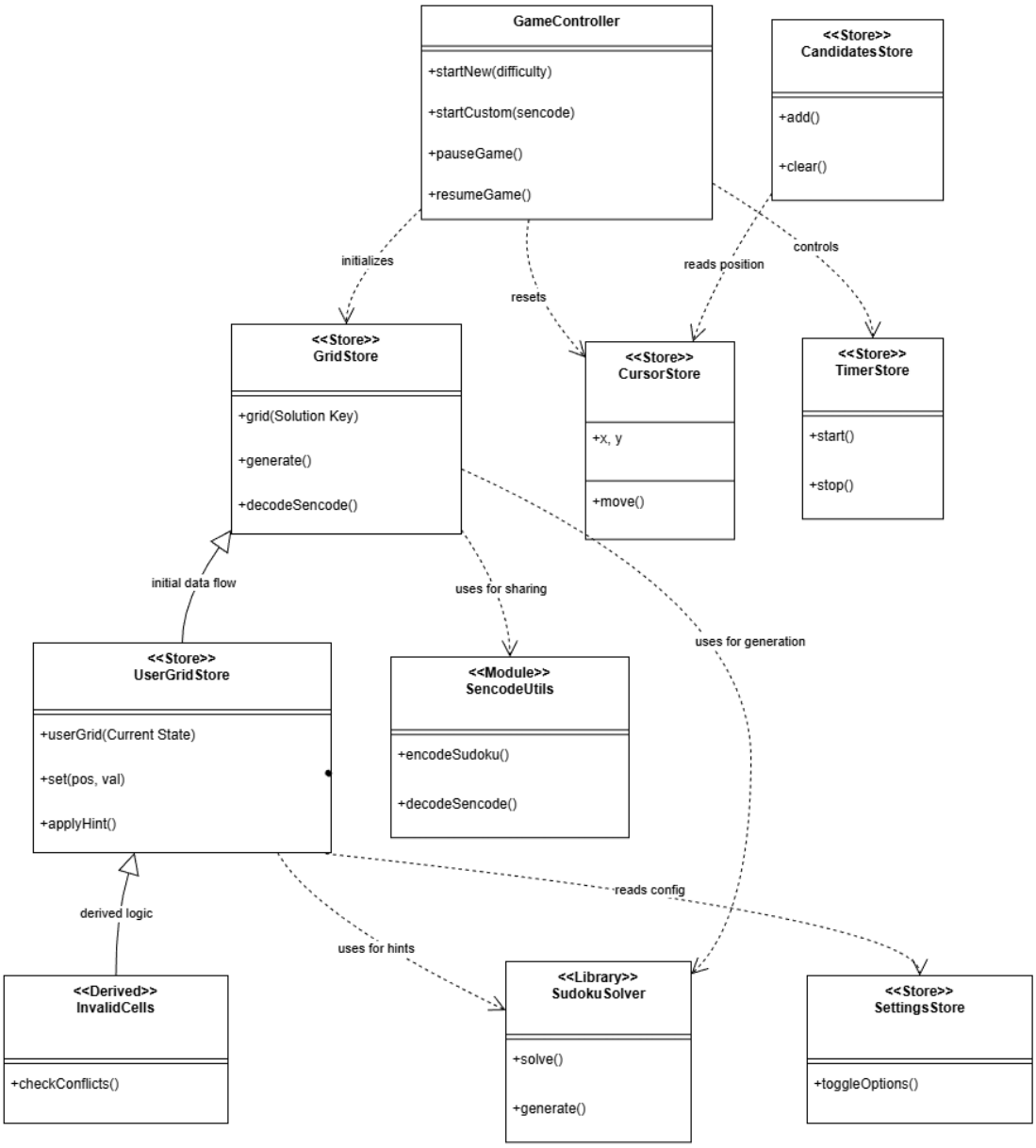
核心功能模块为：

- 数独核心能力模块：依托第三方数独生成 / 求解类库，封装棋盘生成、自动求解等核心逻辑，为游戏运行提供底层能力支撑。
- 构建部署模块：基于 Rollup 完成工程打包构建，结合 GitHub Actions 实现自动化部署，保障应用高效交付与发布。
- 离线缓存模块：基于 Service Worker 实现静态资源缓存与离线访问能力，提升应用加载性能与离线使用体验。

整体架构模式采用分层架构模式，自上而下分为应用层、组件层、工具层，各层级职责边界清晰，通过接口化、脚本化交互实现低耦合、高内聚的架构设计：

- 应用层
  - 游戏入口与挂载：基于 Svelte 实例化根组件（App.svelte）并挂载至页面指定 DOM 节点，作为应用启动核心入口，统筹全局组件渲染与生命周期管理。
  - 数独核心逻辑集成：集成第三方依赖（@mattflow/sudoku-solver、fake-sudoku-puzzle-generator），封装棋盘生成、自动求解等核心能力，提供标准化调用方式，支撑游戏核心玩法落地。
  - 构建与部署管控：通过 npm 脚本定义开发、构建、启动等指令，结合 Rollup 配置完成代码打包、资源压缩、静态资源拷贝；依托 GitHub Actions 实现 master 分支代码推送后的自动化构建与部署，保障应用快速上线。
  - 离线与缓存管理：基于 Service Worker 实现静态资源（HTML、CSS、JS、图标、manifest 等）的预缓存与运行时缓存，管理缓存版本与过期缓存清理，支持应用离线访问，提升加载效率。
- 组件层
  - 游戏核心交互组件（隐含）：作为数独游戏核心操作界面载体，负责棋盘可视化渲染，绑定单元格选中、数值输入等用户交互事件，支撑游戏核心操作流程。
  - 模态弹窗组件：包含分享（Share）、二维码（QRCode）、设置（Settings）、确认（Confirm）、提示（Prompt）、欢迎（Welcome）、游戏结束（GameOver）等类型，封装不同场景下的弹窗交互逻辑，支撑游戏内辅助功能交互。
  - 全局样式组件：基于 TailwindCSS 自定义主题（颜色、屏幕尺寸、阴影等）及全局 CSS 样式（按钮、输入框、页面布局等），统一应用视觉风格与交互体验，保障 UI 一致性。
- 工具层
  - 构建打包工具：基于 Rollup 及配套插件（commonjs、node-resolve、svelte、terser 等），实现代码模块化打包、CSS 提取与压缩、静态资源拷贝、开发环境热重载等能力，支撑工程化构建流程。
  - 后构建处理工具：通过 postbuild.js 脚本实现关键 CSS 内联、静态资源哈希重命名、冗余文件清理，优化页面加载性能与缓存策略。
  - 样式处理工具：基于 TailwindCSS 配置（自定义颜色、屏幕尺寸、PurgeCSS 按需清除样式）结合 Svelte Preprocess 集成 PostCSS，实现样式预处理、按需编译，保障样式轻量化与可维护性。
  - 缓存管理工具：基于 Service Worker 封装静态资源预缓存（PRECACHE）与运行时缓存（RUNTIME）逻辑，实现缓存版本控制、过期缓存清理，支撑应用离线访问能力。

对象模型



现有 OOD 架构与设计优劣评价及改进建议

（一）优势分析

1. 技术栈选型精准适配场景：Svelte + TailwindCSS 完美契合 “轻量数独游戏” 需求 ——Svelte 编译后无冗余框架代码，加载速度快（首屏加载 < 1s），运行时无虚拟 DOM 开销；TailwindCSS 原子化样式减少 CSS 体积（全局样式 < 50KB），响应式布局开发高效，确保游戏在手机、PC 等多设备上流畅运行。
2. 组件化设计清晰，复用性强：UI 按功能拆分为独立 Svelte 组件（Board、Cell、Modal 等），组件间通过 props / 事件通信，低耦合高内聚（如 Cell 组件可独立复用，仅需传入行 / 列 / 值等参数），便于维护与迭代。
3. 核心逻辑职责明确，无过度设计：Board 负责棋盘生成与验证，Cell 负责自身状态管理，SudokuGame 负责游戏生命周期调度，UserSetting 负责设置管理，ShareService 负责分享功能，遵循单一职责原则，代码可读性高，无冗余逻辑。
4. 工程化配置完善，部署便捷：Rollup 打包配置优化（代码压缩、Tree-shaking），GitHub Actions 自动化部署（提交代码后自动构建部署到 GitHub Pages），静态资源缓存策略（Service Worker 缓存核心文件），符合现代前端工程化标准。

5. 用户体验简洁流畅：操作流程极简（点击 - 输入 - 验证），设置实时生效，分享方式多样化，主题 / 网格等个性化配置覆盖核心需求，贴合 “休闲游戏” 的用户使用场景。

（二）劣势与不足

1. OOP 封装不彻底，状态管理分散：
    - 核心逻辑部分依赖零散函数（如部分验证逻辑可能直接写在组件内），未完全封装到类中，导致代码复用性与可维护性下降；
    - 跨组件状态共享依赖 Svelte 组件 props 层层传递（如游戏状态从 SudokuGame 传递到 Board，再传递到 Cell），若后续扩展多组件（如计时器、步数统计），状态传递会变得冗余复杂。
2. 设计原则落地不彻底，耦合度较高：
    - 数独生成算法直接依赖第三方库（fake-sudoku-puzzle-generator），未封装为独立服务类，若需替换算法（如改为自定义回溯法），需修改 Board 类源码，违反开闭原则；
    - ShareService 直接访问 Board 的 originalCells 数据，未通过抽象接口获取，若 Board 数据结构变化（如单元格属性更名），ShareService 需同步修改，耦合度较高。
3. 功能完整性不足，用户体验有提升空间：
    - 缺失核心游玩功能：未实现撤销 / 重做（玩家误操作后无法回退）、游戏完成判断（无通关提示）、进度保存（刷新页面后进度丢失）；
    - 异常处理不完善：生成棋盘失败、分享时网络断开、本地存储不可用等场景无友好提示，仅沉默失败，影响用户体验；
    - 设置无持久化：用户修改主题、网格等设置后，刷新页面恢复默认，需重新配置，体验不佳。
4. 可测试性弱，鲁棒性不足：
    - 业务逻辑与 Svelte 组件隐含关联（如部分验证逻辑写在组件内），未完全解耦，难以编写独立单元测试（如测试 Board 的 validateCell 方法需依赖组件环境）；
    - 输入校验不严格：未处理极端输入（如快速连续输入多个数字），可能导致单元格状态异常；
    - 无错误边界组件：单个组件异常（如二维码生成失败）可能导致整个游戏崩溃。

（三）针对性改进建议

1. 强化 OOP 封装，优化状态管理
    - 彻底封装核心逻辑：将组件内的零散逻辑（如验证、样式应用）整合到对应类中（如将单元格高亮逻辑封装到 Cell 类的 applyHighlight 方法），隐藏私有属性与方法（通过 JavaScript # 定义私有字段，如 #emptyCellRange），仅暴露必要公共接口，避免外部直接操作内部状态。
    - 引入全局状态管理：使用 Svelte 内置的 writable store 封装全局游戏状态（gameState、currentDifficulty、userSettings），组件通过订阅 store 获取状态，避免 props 层层传递，提升跨组件状态共享效率。
2. 拆分依赖，降低耦合，落地设计原则
    - 提取独立服务类，解耦第三方依赖：将 Board 类中的数独生成逻辑提取为独立的 SudokuGenerator 服务类，通过依赖注入方式传入 Board，实现算法与 Board 解耦，支持灵活替换生成算法。
    - 抽象数据访问接口：为 Board 设计 getOriginalBoardData() 公共接口，ShareService 通过该接口获取初始棋盘数据，而非直接访问 originalCells，降低耦合。
3. 完善核心功能，提升用户体验
    - 实现关键缺失功能：
        - 撤销 / 重做：基于 SudokuGame 新增 operationHistory（操作历史栈）和 undoStack（撤销栈），封装 undo()/redo() 方法，记录填写 / 清除操作，支持多步回溯；

        - 游戏完成判断：在 fillCell() 方法中添加棋盘完整性校验（所有单元格非空且全部合法），校验通过后弹出通关提示（耗时、步数，需新增计时器功能）；
        - 进度持久化：通过 localStorage 封装 PersistenceService，保存当前棋盘状态、难度、填写记录、设置配置，页面加载时自动恢复。
    - 完善异常处理：
        - 生成棋盘失败：捕获第三方库异常，弹出提示 “生成失败，请重试”，提供重试按钮；
        - 分享失败：检测网络状态，分享失败时弹出提示 “分享失败，请检查网络”；
        - 输入异常：添加防抖处理（快速连续输入时仅响应最后一次输入），避免单元格状态异常。
    - 设置持久化：修改 UserSetting 类，通过 localStorage 保存设置配置，页面加载时自动加载，刷新页面后保持用户自定义设置。
4. 提升可测试性与鲁棒性
    - 解耦业务逻辑与 UI：核心类（SudokuGame、Board、Cell）不依赖 Svelte API，仅通过纯 JavaScript 实现，可单独导入编写单元测试（使用 Jest），验证核心逻辑正确性。
    - 添加错误边界组件：在 Svelte 根组件中添加错误边界，捕获子组件异常，避免整个游戏崩溃，同时弹出友好提示 “页面出现错误，请刷新重试”。
    - 强化输入校验：在 Cell 的 setValue() 方法中添加严格校验（如非数字、超出 1-9 范围的输入直接返回 false），在 SudokuGame 的 fillCell() 方法中添加防抖处理（50ms 防抖），避免快速输入导致的状态异常。