OOAD 数独乐乐：逆向分析

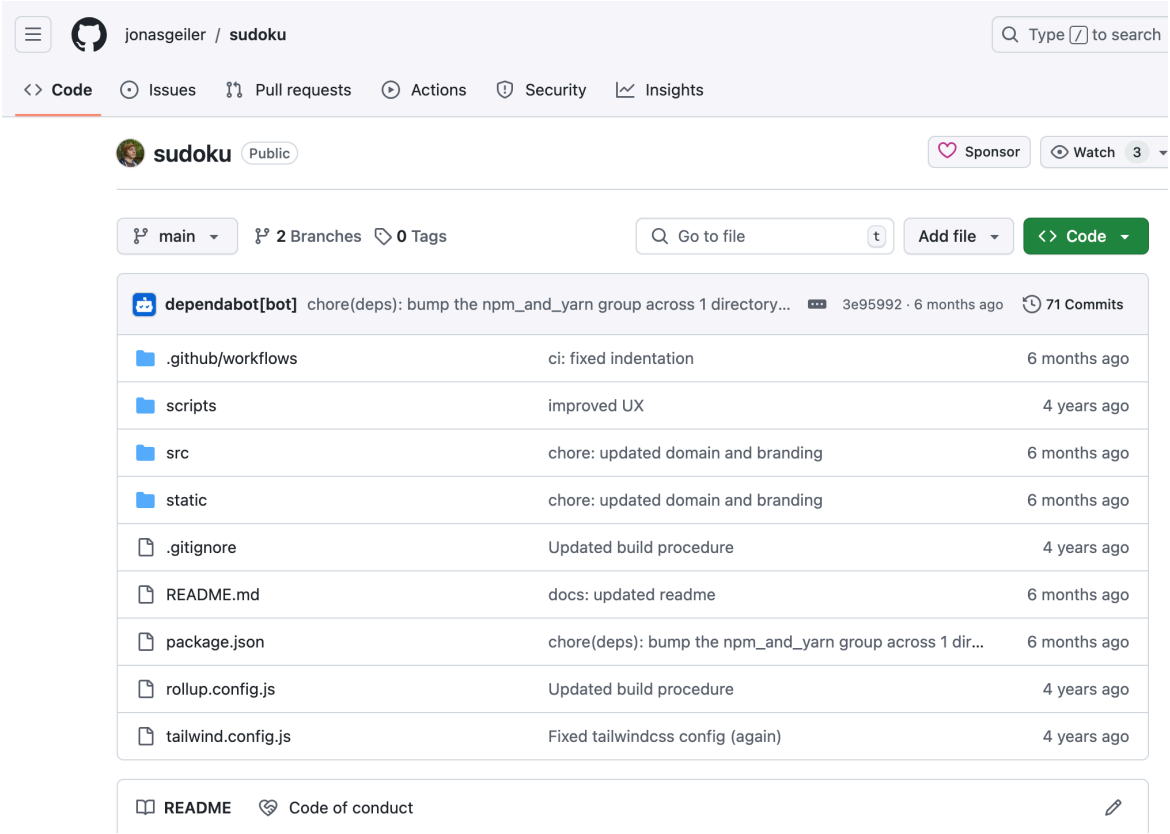
张嘉康*

2026 年 1 月 14 日

1 题目要求

逆向分析现有数独项目的面向对象技术，然后按新业务愿景，改进升级为数独乐乐应用；分析、设计出面向对象技术方案，并落地代码实现。

现有数独项目：https://github.com/jonasgeiler/sudoku



逆向工程，对现有项目的 OOA、OOD、OOP 进行分析：

1. 讨论其设计思想、设计原则和使用的设计模式，给出其愿景、用例分析、领域模型、技术架构与对象模型；

2. 结合课程，评价现有 OOD 架构与设计的优劣，给出改进建议。

*学号：25214604

2 分析与结论

围绕 GitHub 上的 jonasgeiler/sudoku 仓库，可以看到作者从一开始就把这个项目定位成” 用 Svelte 和 TailwindCSS 写成的一款漂亮、小巧的数独小游戏”。这意味着项目的首要愿景并不是做一个功能极其复杂的数独求解器，而是为终端用户提供一个打开网页即可游玩的、交互流畅且界面简洁的数独体验。基于这一愿景，整个系统的面向对象分析、设计与实现都自然而然地围绕” 玩家如何在浏览器里完成一盘数独” 这个核心需求展开。

从面向对象分析（OOA）的角度出发，首先需要明确系统的主要参与者和关键用例。唯一的外部参与者是” 玩家”，他打开网页后希望能够立即开始新游戏，在不同难度下获得系统自动生成的题目，并在一个 9x9 棋盘上用鼠标和键盘填写数字、擦除数字，希望系统在其输入错误或产生冲突时给予即时反馈，并在填满棋盘且满足数独约束时给出完成提示。与此同时，项目的 README 明确注明” 撤销/重做” 和” 创建自定义数独题目” 仍然是规划中但尚未完成的功能，这也限定了当前版本的功能边界。在这样的理解基础上，可以得到图 1 所示的用例模型，其中用例之间的 include 与 extend 关系刻画了” 开始新游戏” 与” 选择难度”” 判断是否完成”” 冲突高亮” 等行为之间的依赖与扩展关系，而” 撤销/重做”” 自定义题目” 则以规划中用例的形式出现在图中，指向未来的业务愿景。

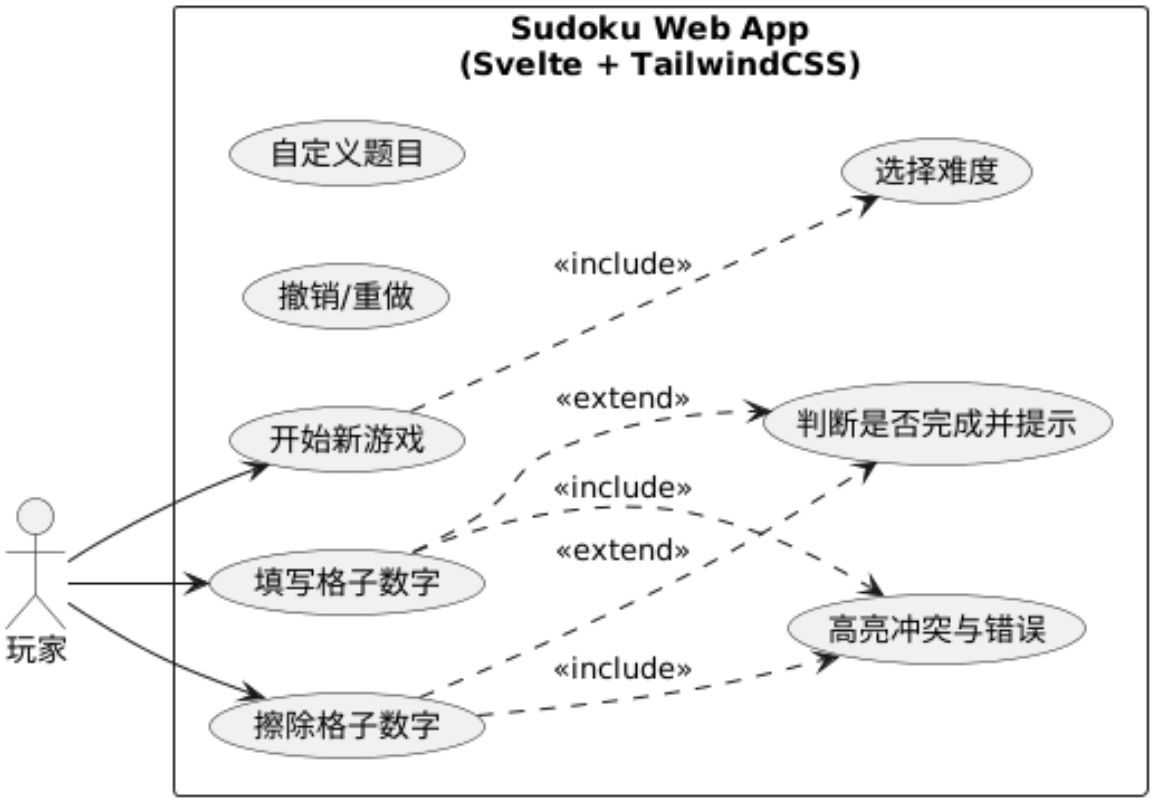


图 1: 现有项目的用例图与未完成功能愿景

在明确了用例之后，可以进一步抽取问题域中的核心概念，构建领域模型和对象模型。数独本身的规则非常清晰：一个 9x9 的棋盘、81 个格子、每个格子要么是系统预置的” 题目给定值”，要么是玩家输入的” 当前值”；任何一行、任何一列以及任何一个 3x3 宫格内，数字 1–9 都不能重复。基于这些不变约束，可以自然地抽象出” 棋盘”（Board）、” 格子”（Cell）、

” 位置”（Position）等实体，以及承载” 某一局题目” 的 Puzzle 对象。同时，游戏运行过程中还需要有一个” 游戏控制器”，负责管理当前棋盘状态、当前题目、选中的格子以及游戏是否已经完成，这一职责在对象模型中由 SudokuGame 承担；而判定某一步输入是否合法、当前棋盘是否已经完成、哪些格子存在冲突等逻辑，则集中由一个纯逻辑模块 SudokuEngine 提供。图 2 展示了基于仓库技术栈和现有功能推断出的核心领域对象模型。

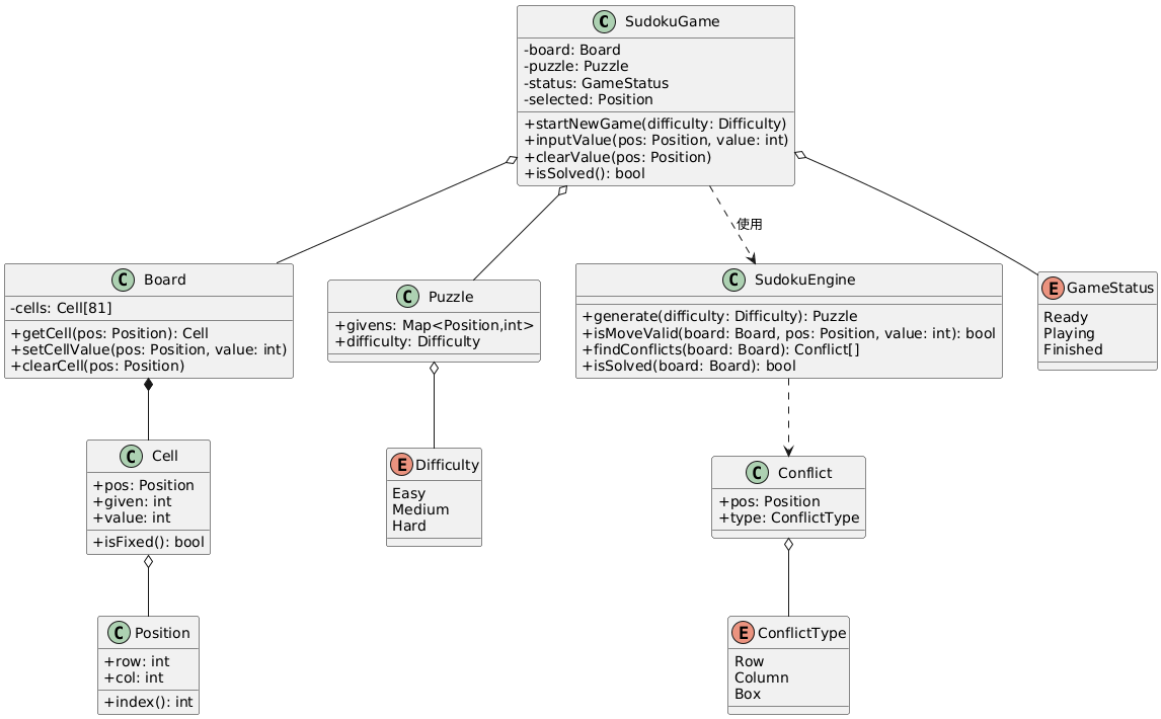


图 2: 现有数独项目的核心领域对象模型

这个模型体现了现有项目在设计上的几个关键思想。其一，Board 与 Cell 的关系采用组合（composition），棋盘对外只提供” 按位置访问和修改格子” 的接口，内部如何存储这 81 个格子则完全封装起来，这样 UI 层不需要知道内部是数组还是映射，从而降低了耦合。其二，Cell 同时保存” 题目给定值” 和” 玩家当前值”，并通过 isFixed 方法体现” 是否允许修改” 这一业务规则，相当于在对象内部维护了数独的一个不变量。其三，SudokuEngine 将规则校验与题目生成集中封装，SudokuGame 并不关心规则细节，而是通过调用引擎接口来完成” 判断某一步是否合法”” 计算冲突集合”” 判断是否完成” 等操作，从而实现了业务控制与规则算法的分离。这样一来，如果将来需要替换为更复杂的求解或生成算法，只需要替换 SudokuEngine 的实现即可。

在领域模型之上，现有项目采用的是一个前端单页应用的技术架构，Svelte 负责组件化界面构建，TailwindCSS 负责快速构建响应式样式，数独核心逻辑则通过 JavaScript 模块暴露给 Svelte 使用。这一架构可以抽象为三个层次的分离：UI 组件层负责渲染界面并捕获用户交互、状态管理层通过 Svelte Store 维护游戏状态、领域逻辑层实现纯粹的数独规则与算法。在这样的分层架构中，UI 层的各个 Svelte 组件将状态渲染成 DOM，并把用户交互事件转换成对状态层的调用；状态层往往通过一个全局的 GameStore 维护当前的棋盘、当前题目、冲突集合、选中格子等游戏状态；领域逻辑层的 SudokuGame 和 SudokuEngine 则实现具体的业务规则与算法。

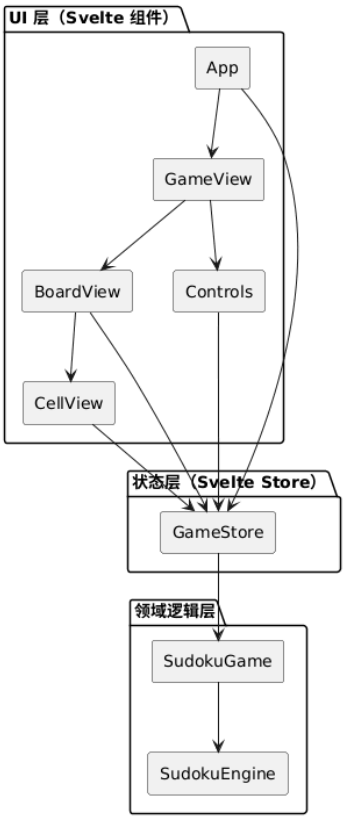


图 3: 现有项目的技术架构与组件分层

Svelte 的 Store 本身就是一个轻量级的观察者模式实现，UI 组件通过订阅 Store 的变化来自动更新界面，而 Store 内部的状态变更往往是通过调用 SudokuGame 的方法完成的。因此，当玩家在界面上对某个格子进行一次输入时，背后实际上触发了一条从 UI 组件到 Store，再到领域对象，最后又回到 UI 的完整消息流。图 4 就刻画了这样一个典型的顺序：玩家在界面上点击格子输入数字，组件通过 Store 分发意图，SudokuGame 更新棋盘并委托引擎校验，随后新的状态通过 Store 的响应式机制回流到各个订阅的组件。

从面向对象实现（OOP）和设计原则的角度评价，现有架构在整体方向上是合理的。首先，它实现了模型与界面的明确分离，数独规则和棋盘状态被封装在普通的 JavaScript 模块中，而 Svelte 组件更多承担” 视图” 和” 输入收集” 的职责，这与课程中倡导的 MVC/MVVM 思想是一致的。其次，通过 Svelte Store 的机制，项目天然获得了一种数据驱动的观察者模式实现，UI 无需手动管理刷新，只需订阅状态变化即可自动重绘，从而减轻了命令式更新界面的负担。此外，将规则与生成逻辑集中在 SudokuEngine 中，从单一职责原则和开放封闭原则的视角看也是合理的，未来如果要支持不同风格的题目生成算法，可以在不改动 UI 的前提下替换或扩展引擎实现。

与此同时，这一设计也暴露出一些随着功能扩展而逐渐显露的问题。README 中标出的” 撤销/重做” 和” 创建自定义题目” 两项未完成功能，本质上都指向领域层建模上的空缺。在现有模型中，SudokuGame 主要围绕” 当前棋盘” 和” 当前题目” 组织状态，如果没有显式的” 操作” 对象和” 操作历史” 结构，那么每一次玩家输入都只是一次简单的状态覆盖，这使得事后无法追溯一系列操作，更难以做到既可靠又易于维护的撤销/重做。同样地，当前的

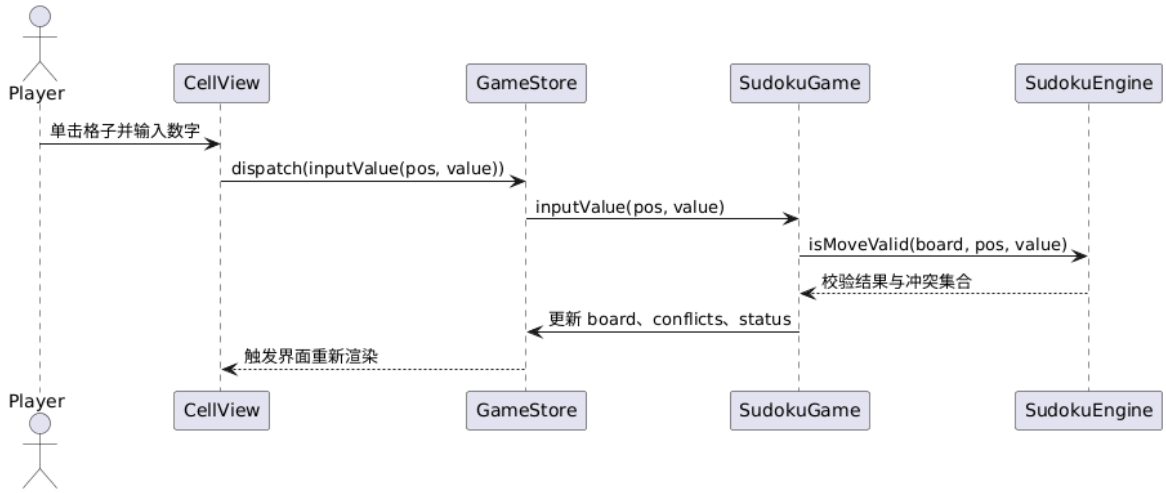


图 4: 玩家填写数字时的典型交互顺序

Puzzle 只被简单地看作” 系统生成的一局题目”，没有承载作者、自建来源、唯一解校验等更丰富的信息，也就难以自然扩展到支持用户创建、保存并分享自定义题目的新业务。

基于课程中关于命令模式、状态模式以及领域建模的内容，可以围绕” 数独乐乐” 这一升级愿景，对现有对象模型提出具体的改进建议。撤销/重做功能的核心在于将用户的每一次操作建模为一个 Move 对象，该对象不仅记录了操作的位置和数值变化，还包含执行（apply）和撤销（revert）的方法，这正是命令模式的典型应用。通过引入 MoveHistory 来管理已执行和已撤销的操作栈，就能以极小的代码复杂度实现多步撤销与重做，而且这种设计完全在领域层完成，UI 层只需要调用 undo/redo 方法即可，无需关心历史管理的细节。

对于自定义题目功能，改进的关键在于扩展现有的 Puzzle 模型，使其不仅包含题目的给定数字，还能承载作者信息、创建时间、难度评估、唯一解验证等元数据。可以设计一个 CustomPuzzle 类继承自基础 Puzzle，并提供 validate 方法来校验题目的合法性。这样既能复用现有的题目结构和规则引擎，又能为未来的题库管理、题目分享、难度评分等功能预留扩展点。

此外，引入显式的游戏会话概念也是一个重要的改进方向。将当前的棋盘状态、题目信息、操作历史、计时器等聚合到一个 Session 对象中，使得” 开始新游戏”” 暂停/继续”” 保存进度”” 恢复游戏” 等操作都能在清晰的对象边界内完成。这种设计不仅让代码结构更加清晰，也为将来支持多个游戏会话并行（例如同时进行多局游戏）提供了可能。

在状态管理方面，现有项目可能缺少一个明确的状态机来管理游戏生命周期。通过引入 GameStatus 枚举（如 Idle、Playing、Paused、Completed）并严格控制状态转换规则，可以避免因状态组合爆炸而导致的逻辑混乱。例如，只有在 Playing 状态下才允许输入数字，只有在 Completed 状态下才显示胜利动画，这样的约束可以在领域层通过状态机模式优雅地实现，而不是在 UI 层通过大量的条件判断来控制。

综合来看，现有的 jonasgeiler/sudoku 项目在 OOA 阶段对” 轻量、好看、可玩” 的愿景把握得比较准确，在 OOD 阶段通过 Svelte + Store + 纯逻辑模块的分层实现了界面、状态与规则的初步解耦，在 OOP 实现层利用函数式风格与轻量对象的结合避免了过度设计。它的主要不足在于对” 操作” 和” 会话” 这两个领域概念的建模较为薄弱，导致撤销/重做、自建

题目等高级功能难以自然落地，同时在状态管理上如果缺乏显式的游戏状态机描述，当功能逐渐增多时容易出现状态组合爆炸。通过引入命令模式来支持操作历史管理、扩展题目模型以支持自定义功能、引入会话概念来聚合游戏状态、采用状态机模式来管理生命周期，可以在保持当前项目简洁性的前提下，将其演进为一个更具扩展性和可维护性的”数独乐乐”应用，从而满足课程对”逆向分析 + 面向对象重设计”的综合要求。