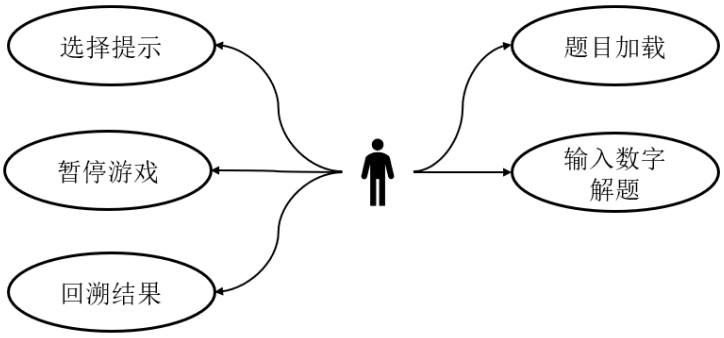
OOAD 逆向分析

一、 用例分析



本项目的主要功能

用例名称	题目加载
参与者	用户
前置条件	系统正常运行
后置条件	如果题目合规，则进入解题流程；否则，重新加载页面
事件流	基本流 1.用户进入系统界面； 2a. 用户通过系统提供难度进行； 2b. 用户自定义题目； 2b1.用户通过 sudokuwiki.org 题目链接导入题目； 2b2.系统校验题目的合规性； 3.系统可视化数独问题，进入解题界面。

用例名称	输入数字解题
参与者	用户
前置条件	系统处在解题状态，计时正在进行
后置条件	若输入合法，更新棋盘单元格内容，记录用户输入；若不合法，提示错误，保持棋盘当前状态
事件流	基本流 1. 用户在游戏解题过程中，鼠标点击棋盘上某一单元格使其获得焦点，或通过键盘导航到目标单元格。 2. 用户通过键盘输入数字 1~9。 3. 系统校验输入数字：检查该数字在所在行、列及九宫格是否重复，是否符合数独规则要求。

	4. 若校验通过，系统更新该单元格显示为输入数字，记录该格输入；若不通过，在界面

用例名称	回溯结果
参与者	用户
前置条件	系统处在解题状态，计时正在进行
后置条件	系统展示解题过程相关信息，用户可查看并从中学习，系统记录用户查看回溯结果的操作。
事件流	基本流 1. 用户点击“undo”按钮。 2. 系统根据游戏过程中的用户输入顺序，整理出解题过程的关键步骤、错误点或完整的解题思路复盘。 3. 系统以可视化或文本形式展示这些信息给用户，用户可浏览查看。

用例名称	选择提示
参与者	用户
前置条件	系统处在解题状态，**用户的提示次数大于 0**
后置条件	系统在棋盘上展示提示信息，用户可参考提示继续解题，系统记录剩余提示次数等信息。
事件流	基本流 1. 用户在游戏解题过程中，遇到困难，点击 “提示” 按钮。 2. 系统根据当前棋盘上用户填写的数字、未填数字及数独规则，计算潜在的解或给出下一步解题方向（如某单元格的可能候选数字、某区域的关键数字等）。 3. 系统以适当方式在棋盘上用**特殊颜色、显示数字**的方式突出关键单元格。

用例名称	暂停游戏
参与者	用户
前置条件	系统处在解题状态，计时正在进行
后置条件	系统停止计时，**游戏界面隐藏**，保存当前游戏进度，用户可进行相关后续操作。
事件流	基本流 1. 用户在游戏解题过程中，点击 “暂停” 按钮。 2. 系统隐藏游戏界面或使其进入不可操作状态，保存当前游戏状态，包括棋盘数字、

	用户输入等。

二、 领域模型

2.1 核心概念

Game：

(1)**游戏状态管理**:通过 gamePaused/gameWon/resumeGame 等控制游戏的状态，把控了各阶段组件的可操作状态；

(2)**关联中枢**：与数独棋盘 grid/userGrid、操作控制组件的 ActionBar、计时器 Timer 和键盘输入控制进行了集成，是一个系统协调单位。

Board：

(1)Cell：展示自身的数字，选中后带来外观变化，接受鼠标点击、键盘输入等交互事件来改变自身的值；

(2)Board：由 Cell 数组组成属性，记录了无效单元格 invalidCells、用户填写数字后的网格 userGrid、原始网格 grid 等信息。负责棋盘的渲染展示，协调单元格关系，基于用户操作更新自身状态。

Settings：

(3)记录游戏的默认设定，如显示时间 displayTimer、设置提示限制 hitsLimited、高亮显示相同数字 highlight、高亮显示矛盾数字 highlightConflicting、高亮显示同区域单元格 highlightCells 等游戏展示效果和逻辑配置选项。

(4)作为一个单例为游戏的各个模块提供设定依据，是游戏的“潜在状态”。

2.2 对象关系

(5)组合关系

Game：由 Board、Controls、Modal 等组件组成，Game 负责组件间的协调；

Controls：由 ActionBar、Timer、Actions(Redo/Undo/Hints)等组件组成，负责用户辅助功能的实现；

Board：主要由 Cell 数组组成，承载具体的游戏数据和交互逻辑。

(6)依赖关系

Cell：依赖于光标位置 cursor、游戏是否暂停 gamePaused 等状态来展示

自己的样式；

Controls：依赖于其下的组件 Actions、Timer、Keyboard，且依赖于 gamePaused、Settings 等存储状态。

2.3 领域规则与约束

(1)数独游戏的棋盘约束：每个单元格的填写范围限制子 1~9 中的一个，除非特殊的提示需要。每个行、列和每个九宫格区域不能有重复的数字。通过在填写校验、获取提示等可能更新数独矩阵的操作中进行合法性逻辑检验。

(2)游戏状态一致性约束：系统中，gamePaused、cursor、数独矩阵、禁用状态之间需要保持逻辑的一致性。最简单的，游戏属于暂停状态时，除了 Modal 给出的提示框组件外，其余组件应处在禁用、停止运行的状态；光标位置的改变、鼠标点击单元格操作应该与对应的单元格保持一致。

(3)设置规则的约束：Settings 中的各种选项应该被受影响的组件严格执行，比如设置了 hintsLimited,“Hint”组件的提示数量应该按要求对用户进行限制。

三、 设计模式分析

3.1 工厂模式

使用了 createGrid/createUserGrid/createCandidates 等工厂函数。项目使用工厂模式创建对象，并封装对象创建的具体逻辑。

项目通过工厂函数创建 grid、userGrid 和 candidates 等全局状态。通过工厂函数，对创建过程进行封装，避免直接使用 writable 的复杂性。

这种做法提供了更清晰的对象创建接口，便于扩展。

3.2 单例模式

项目对 candidates/grid/userGrid/modal 等位置是用 writable 状态实现单例模式。

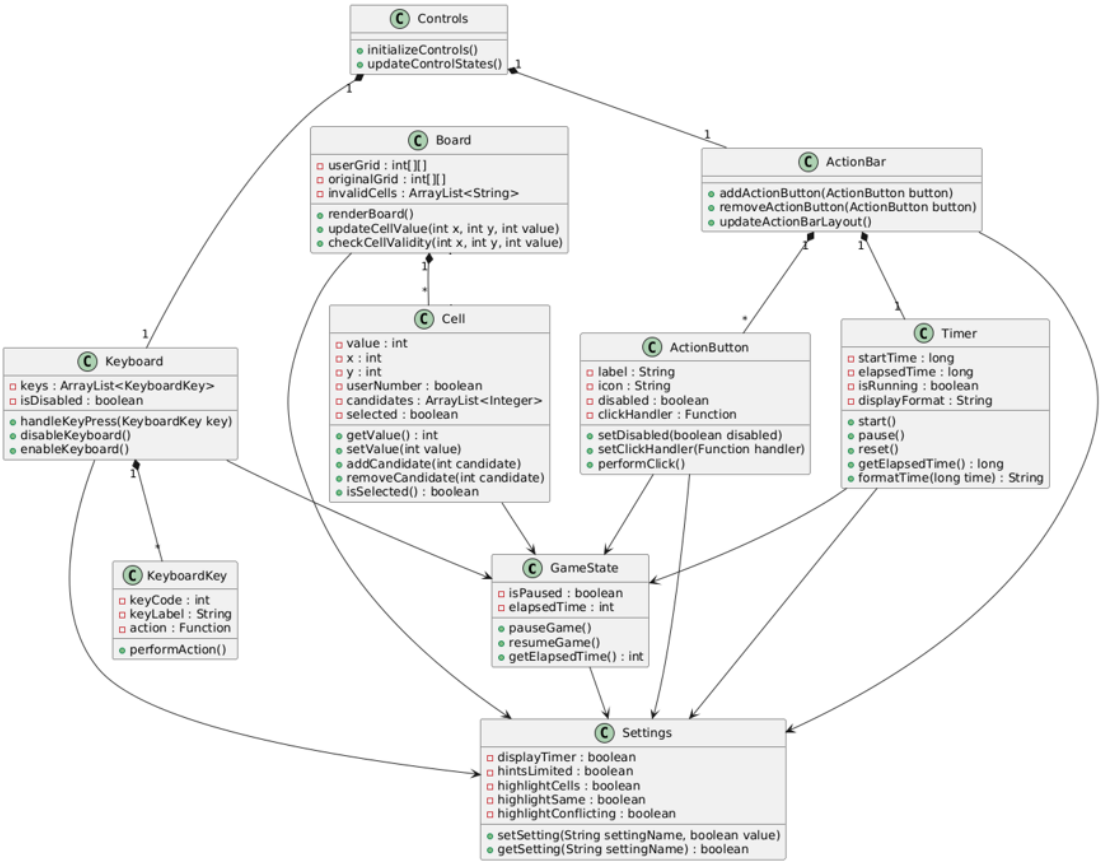
这类实例在状态管理上可以保证整个项目都只有一个实例，保持了系统状态的一致性。

3.3 观察者模式

实现了时间的订阅与通知机制，当观察对象的状态发生变化时，所有订阅者都会收到通知，从而做出对应的变化。

项目中使用 writable 和 derived 状态，在系统状态发生变化时，candidates/modal/userGrid 等会收到消息，自动触发 UI 渲染。

四、对象模型



五、改进分析

5.1 项目优点

简单直观：代码逻辑相对简洁，易于理解。每个 store 管理一个特定的领域概念状态，无过度复杂
的类继承关系。

灵活扩展：通过新增 store 和组件文件，可以快速添加新功能或修改现有逻辑。

数据驱动 UI：Svelte Store 与组件双向绑定，UI 自动随数据变化更新，降低耦合。

5.2 缺点

缺乏明显的面向对象抽象：没有类与接口，难以通过多态或继承来拓展功能。

对新需求的扩展可能要求在全局范围搜索替换逻辑。

业务逻辑分散：逻辑分布在多个 store、组件与函数文件中，没有统一的领域模型类，导致逻辑理解需要在多文件间跳转。

可测试性与可维护性欠佳：缺少清晰的分层架构和抽象接口，不易为各独立模块进行单元测试。

5.3 改进建议

引入更面向对象的抽象层：

1) 数独逻辑封装为独立类：

将数独的核心逻辑(求解、生成、验证)封装成独立类，提升代码的模块化与复用性。例如，SudokuPuzzle 类代表数独谜题，包含 generate()方法用于生成新谜题，solve()方法用于求解谜题，以及 validate() 方法用于验证当前谜题的合法性。

2)服务类的抽象

将候选数字、用户输入、提示、计时器等功能抽象为独立服务类，使用接口与实现分离，方便未来的扩展或替换:

GridService：管理数独网格的候选数字与用户输入。

HintService：提供提示逻辑，支持不同难度下的策略。

TimerService：负责计时功能，记录游戏时间。

使用依赖注入或模块边界清晰化

将游戏逻辑模块化，使用依赖注入来管理类之间的依赖关系。在 Game 类的构造函数中注入 GridService、TimerService 和 HintService 等依赖。

采用更丰富的设计模式

1) 策略模式用于提示逻辑

为不同难度级别的提示逻辑使用策略模式，根据游戏难度动态调整提示策略。定义一个 HintStrategy 接口，不同策略实现此接口。提供简单提示策略和高级提示策略，基于当前难度动态选择合适的策略。

2)抽象工厂或策略模式用于求解和生成算法

使用抽象工厂或策略模式，为数独求解和生成提供不同的算法实现，通过接口注入，使得算法可以灵活替换。

分层架构

1)UI、逻辑与数据存储层分离

将用户界面、业务逻辑和数据存储层明确分离，提升代码的可维护性。

① UI 层：负责用户交互，调用逻辑层的方法。

② 逻辑层：封装核心业务逻辑，提供服务接口供 UI 层调用。

③ 数据存储层（Store）：作为数据观察层，负责数据的管理与同步。

2)降低 Store 中的业务逻辑负担

将逻辑进一步封装在服务对象中，Store 只负责数据的存储和观察，逻辑操作交由服务对象处理，降低 Store 的复杂性。