数独乐乐报告

邱梓钿 常鹏 郭集河 王舒航

1 需求规格

1.1 用例分析

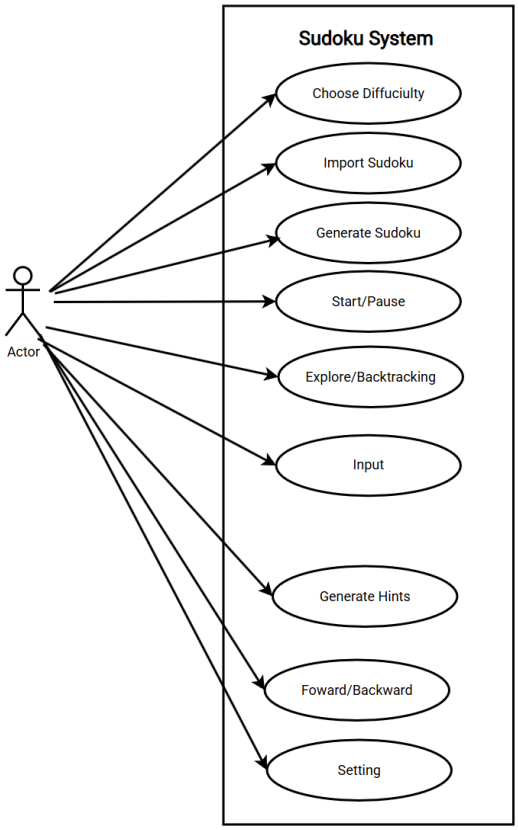

图 1 数独乐乐用例图

如图所示，在数独乐乐中，相比于原先的数独游戏系统，我们对了下一步提示的功能进行了完善，使用策略类与求解类代替了原有项目中第三方库的使用。同时，我们还增加了探索、回溯的功能，当所需网格的具有多个提示数字时，用户可以分别对每个分支进行探索，并且在探索完毕时一键回到分支点，提升了使用的便捷性。此外，我们还完善了系统的资源整合功能，可以方便地导入外部题目，使用外部的算法策略，增强了系统的可拓展性。

1.2 领域模型

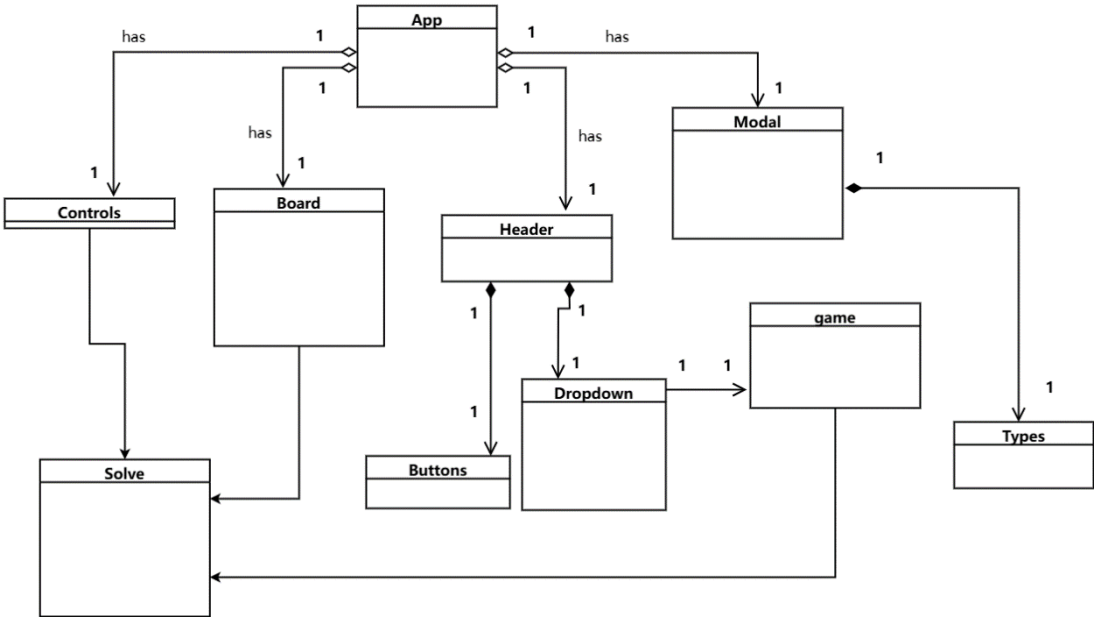

图 2 数独乐乐领域模型图

在数独乐乐中，除了保留传统数独游戏的核心功能外，我们对原有系统进行了多方面的扩展和优化。相比于原始的数独游戏系统，我们加入了更为丰富的功能模块，例如下一步提示、回溯探索、资源集成等。这些新功能不仅能够帮助用户在解题过程中提供更好的引导，还能增强系统的交互性和可操作性。为了解决复杂的数独问题，我们对原有模块内部的逻辑和方法进行了深入的补充和修改，以提高系统的效率和稳定性。

此外，为了实现高效的求解和动态交互，我们新增了求解模块，并将其与现有的各个模块进行了紧密的关联。通过这样的关联，各模块能够相互配合，共同推动游戏的进行。例如，求解模块能够与提示模块协作，及时提供下一步的建议；与回溯模块合作，有效地探索并解决错误路径问题；并且与资源集成模块联动，灵活调用各种资源进行优化计算。所有模块的协作使得系统不仅能实时响应用户操作，还能在复杂问题面前保持高效运算，提升了整体的游戏体验。

2 软件设计规格

2.1 系统技术架构

（1）数独推理的技术架构

业务逻辑层：通过 solveManagement 类与其他组件进行交互，输入是一个需要提示的数独，输出是经过一次迭代得到的可填入的 grid，以及其他 grid 的候选值列表，并且输出策略推理链。

数据访问层：通过继承基类 baseStrategy，得到下图包括 HiddenSingle 等策略类，在每个策略类中 applyStrategy()函数进行推理，策略类中共享一个候选值列表，从而发挥出各个策略的作用。将策略共有的函数作为基类函数，减少代码冗余。

（2）数独探索和回溯的技术架构

该部分采用结合状态管理与逻辑分离的 MVC 模式，其中：

模型层：通过 Svelte 的 store 和 SudokuDAG 实现数据状态管理与操作逻辑。

视图层：使用 Svelte 框架构建界面，实时响应数据变化并提供用户交互。

控制层：通过相关操作函数（如 addState、undo、redo 等）处理用户输入并更新模型层。

2. 2 对象模型

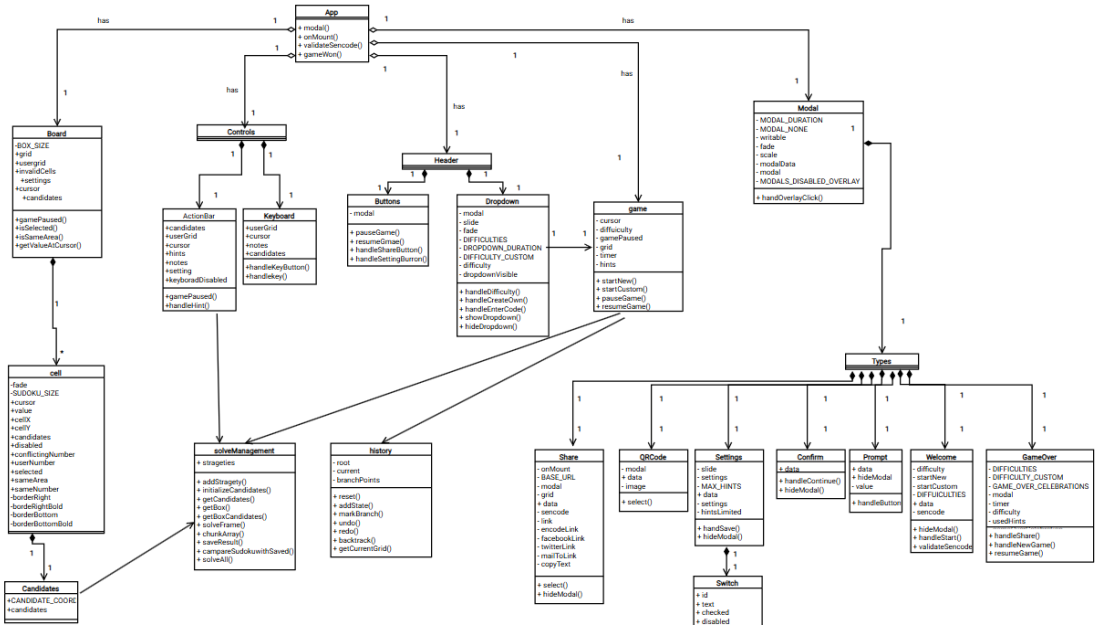

图 3 数独乐乐对象模型图

在数独乐乐中，为了实现数独的求解，我们构造了求解管理类（solveMangement）和策略基类（baseStragety）。求解管理类主要负责求解逻辑的实现，具有策略类调用的顺序以及正确性判断对策略类判断等功能，它对策略类具有组成关系，可以通过全局的策略列表来调用策略类，实现求解。当需要对求解方法进行扩展与资源集成时，用户可以通过继承策略基类来构建新的求解类，并将其加入策略列表中，以便求解管理类调用。同时，为了实现探索-回溯功能，我们对 game 逻辑进行了补充，通过关连求解管理类来实现状态的保存与查询。同时，为了实现候选值的可视化，我们将相关的 ActionBar 与 Candidates 模块与求解管理类进行关联。此外，我们还对题目导入等模块进行了补充，以便更好地完成资源集成。

2.3 设计说明

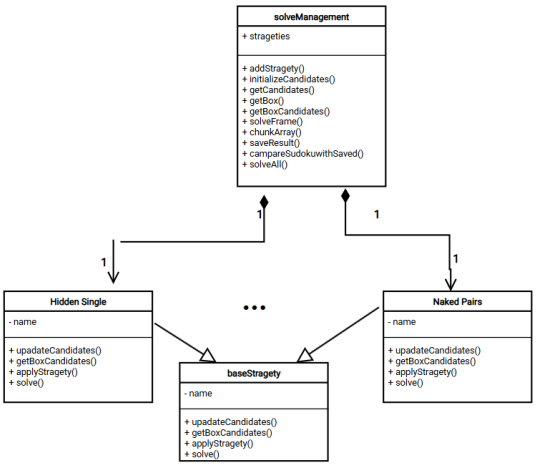

图 4 数独乐乐求解模块图

如上图所示，在数独乐乐的数独求解模块中，我们使用了“开放-封闭原则”进行程序

的设计。其中，“封闭”指的是数独解决策略的调用逻辑以及数独是否有解等评判标准封闭，而“开放”指的是使得开发者能够在不修改现有代码的情况下扩展算法策略。

在设计中，我们使用了“模板方法模式”，可以通过对基础类的继承来定义新的解决策略，并加入至策略列表中，以便求解管理类进行调用。其中，基类含有基础的迭代子和比较子，不同的策略类可以共享相同的“迭代子”和“比较子”来进行求解。同时，我还设计了“自动化的策略正确性验证方法”，在策略进行修改后，可以将修改后的求解结果与已经保存的，正确的结果进行对比，来验证策略的正确性，而不是每次都需要人工验证，这一设计也大大增加了策略扩展的便利性。

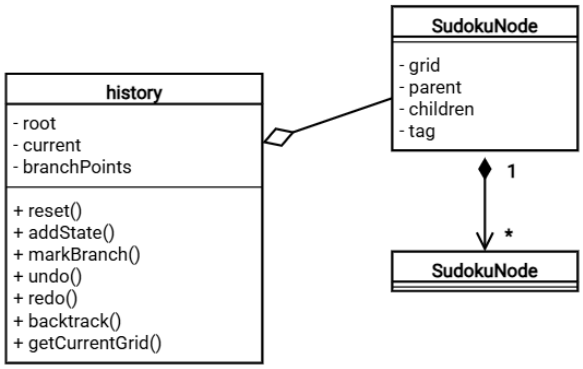

图 5 数独乐乐历史记录模块图

在撤销、重做和回溯功能的设计中，我们遵循了面向对象的设计原则和设计模式，实现了灵活且可扩展的数独历史记录管理。SudokuNode 类作为备忘录模式的核心，负责保存数独棋盘的历史状态，包括当前棋盘数据、子节点、父节点以及分支点的候选值标记，确保状态的独立性和可恢复性。createHistory 函数通过状态模式管理 canUndo 和 canRedo 的状态，通过观察者模式结合 Svelte 的 writable 变量实现了状态与 UI 的实时同步，动态调整用户界面的交互行为。历史记录的管理基于有向无环图（DAG）结构，root 节点表示初始状态，current 节点表示当前状态，branchPoints 栈标记所有分支点，支持撤销、重做和回溯操作。reset 方法初始化历史记录，addState 添加新状态，markBranch 标记分支点，undo 和 redo 分别用于状态回退和前进，backtrack 则回溯到上一个分支点并返回排除的候选值，getCurrentGrid 提供当前棋盘状态。撤销和重做通过节点的父子关系简单高效，分支标记与回溯机制灵活，可快速回退到特定分支。整体设计通过单一职责原则确保每个模块职责明确，通过开闭原则支持扩展，通过依赖倒置原则降低模块耦合，通过组合优于继承原则提升灵活性，同时结合备忘录模式、观察者模式和状态模式，实现了高效、可维护的历史记录管理功能。