# 用例分析

1. 用例名称：生成棋盘

参与者：玩家

简要描述：玩家选择难度等级，初始化棋盘。

前置条件：游戏已启动。

基本流程：

* 玩家点开菜单栏。
* 系统提供难度选择选项(Very easy, Easy, Medium, Hard)。
* 玩家选择一个难度级别。
* 系统根据所选难度生成一个新的数独棋盘。

替代流程：如果系统无法生成符合要求的棋盘（如资源问题），则提示玩家稍后再试或选择其他难度。

后置条件：成功创建了一个新的数独棋盘供玩家解决。

特殊要求：盘应保证有唯一解，并且难度与玩家的选择相匹配。

1. 用例名称：资源集成

参与者：玩家

简要描述：玩家通过输入URL来导入自定义的数独棋盘资源。

前置条件：玩家有一个有效的URL指向所需的数独资源文件。

基本流程：

* 玩家点开菜单栏选择’Enter Code’。
* 系统弹出供玩家输入URL的窗口。
* 玩家输入URL并确认输入。
* 系统根据URL下载并解析棋盘资源。

替代流程：如果URL无效或资源格式不正确，系统会提示玩家重新输入或取消操作。

后置条件：自定义棋盘成功加载，玩家可以选择开始玩这个棋盘。

特殊要求：无

1. 用例名称：输入数字

参与者：玩家

简要描述：玩家可以在数独网格中输入数字以尝试解决问题。

前置条件：

* 数独谜题尚未完成。
* 玩家已选择了要填充的空格。

基本流程：

* 玩家点击或选择数独网格中的一个空格。
* 玩家在可输入的数字中选择一个填入。
* 系统验证输入是否符合数独规则（即行、列和3x3子网格内没有重复数字）。
* 如果输入有效，则更新网格；如果无效，则提示玩家重新输入。
* 如果玩家点击'删除'按钮，则清空该单元格的数字。

替代流程：如果输入不符合规则，系统会显示一条消息指出问题，并允许玩家再次尝试。

后置条件：数独网格上的数字被正确更新，或者玩家得到关于错误输入的反馈。

特殊要求：输入应即时验证，以保证游戏流畅性。

1. 用例名称：撤销 (undo)

参与者：玩家

简要描述：允许玩家撤回最近的一次或多次数字输入。

前置条件：至少有一次有效的数字输入记录。

基本流程：

* 玩家选择’撤销’功能。
* 系统恢复到前一次状态，移除最后输入的数字。
* 系统更新界面上的显示。

替代流程：如果没有可撤销的动作，撤销功能无法点击。

后置条件：数独网格上的最新输入被撤销，玩家可以继续编辑。

特殊要求：支持多级撤销；撤销动作应当即时反映在界面上。

1. 用例名称：重做 (redo)

参与者：玩家

简要描述：在玩家使用撤销功能后，允许再次应用已被撤销的操作。

前置条件：已经进行了至少一次撤销操作。

基本流程：

* 玩家选择’重做功能’。
* 系统重新应用最后一次被撤销的数字输入。
* 系统更新界面上的显示。

替代流程：如果没有可重做的动作，重做功能无法点击

后置条件：最近被撤销的数字输入被重新应用于数独网格。

特殊要求：支持与撤销功能相同的多级重做能力。

1. 用例名称：下一步提示 (1级)

参与者：玩家

简要描述：为玩家提示出仅剩一个候选值的单元格，并解释原因。

前置条件：

* 数独谜题尚未完成。

基本流程：

* 玩家选择“下一步提示(1级)”功能
* 系统显示全部的仅剩一个候选值的单元格
* 玩家点击其中一个仅剩一个候选值的单元格，系统提示线索，并说明推理答案所用的方法/策略。

替代流程：点击该功能后，如果所有空格均无法做到仅剩一个候选值，应该报告异常。

后置条件：系统显示所有仅剩一个候选值的单元格的候选值，并且显示提示线索，并说明推理答案所用的方法/策略。

特殊要求：提示应该即时显示，以保证游戏流畅性。

1. 用例名称：下一步提示 (2级)

参与者：玩家

简要描述：为玩家提示出仅剩两个候选值的单元格，并解释原因。

前置条件：

* 数独谜题尚未完成。

基本流程：

* 玩家选择“下一步提示(2级)”功能
* 系统显示全部的仅剩两个候选值的单元格
* 玩家点击其中一个仅剩两个候选值的单元格，系统提示线索，并说明推理答案所用的方法/策略。

替代流程：点击该功能后，如果所有空格均无法做到仅剩两个候选值，应该报告异常。

后置条件：系统显示所有仅剩两个候选值的单元格的候选值，并且显示提示线索，并说明推理答案所用的方法/策略。

特殊要求：提示应该即时显示，以保证游戏流畅性。

1. 用例名称：下一步提示 (3级)

参与者：玩家

简要描述：为玩家提示出所有候选值有所减少的单元格，并解释原因。

前置条件：

* 数独谜题尚未完成。

基本流程：

* 玩家选择“下一步提示(3级)”功能
* 系统显示全部的候选值减少的空白单元格。
* 玩家点击其中一个候选值减少的单元格，系统提示线索，并说明推理答案所用的方法/策略。

替代流程：点击该功能后，如果所有空格均无法做到减少候选值，应该报告异常。

后置条件：系统显示所有候选值有所减少的单元格的候选值，并且显示提示线索，并说明推理答案所用的方法/策略。

特殊要求：提示应该即时显示，以保证游戏流畅性。

1. 用例名称：探索回溯

参与者：玩家

简要描述：在面临分支点时，玩家可以选择一个数字进行探索，当探索过程遇见错误时，玩家可以一键回溯至分支点。

前置条件：

* 数独谜题尚未完成。
* 玩家已选择了要探索的空格，并填入自定义的候选值作为笔记，且候选值的数量超过1个。

基本流程：

* 玩家选择了要探索的空格，并且笔记中的候选值数量超过1个。
* 若一个单元空白格的候选者值的数量超过一个，系统存储该空白格的位置以及所有未选择的候选值，玩家选择其中一个候选值填入空白格，进行探索。
* 若选择后存在仅剩一个候选值的空白格，则将这些单元格的唯一候选值填写
* 如果后面某一个空白格的唯一的候选值出现不符合规则的现象，就一键回溯到一开始玩家选择的空白格，并删除该空格填入的数字，系统将该候选值从候选值笔记中删除并更新笔记。
* 玩家重复这个过程。

替代流程：如果探索过程中出现新的分支点，应该以新的分支点来执行探索回溯工作。

（另特别注意，如果探索过程中又遇见了另一个分支点，如何解决这个需求，希望做这一部分的同学能补充。）

如果点击的空白格的候选值笔记的数量为1，则直接填写唯一的候选值，探索结束。

后置条件：系统显示回溯后的上一次出现分支的空格，并显示该空格的最新的候选值供玩家探索。

特殊要求：回溯的过程要即时，以保证游戏流畅性。

1. 用例名称：候选值笔记

参与者：玩家

简要描述：玩家可以选择单元格，并填入自定义的候选值作为笔记，以辅助解题过程。

前置条件：

* 数独谜题尚未完成。
* 玩家开启‘笔记’功能。

基本流程：

* 玩家选择一个单元格。
* 玩家通过选择数字1-9来填入候选值。
* 系统将所选的候选值添加到选定单元格的笔记区域。
* 如果玩家再次选择同一个候选值，则系统将其从笔记中移除。
* 如果玩家点击'删除'按钮，则清空该单元格的候选值。

替代流程：如果玩家尝试在一个已填有确定值的单元格上添加笔记，系统应提示玩家先清除确定值。 后置条件：所选单元格的笔记被正确更新。 特殊要求：候选值应该即时更新，以保证游戏流畅性。

1. 用例名称：分享

参与者：玩家

简要描述：玩家可以通过数独识别码、链接、二维码等方式进行分享，或也可以直接点击所提供的按钮，通过推特、脸书、邮件进行分享。

前置条件：

* 游戏中存在可分享的内容，例如当前谜题、进度等。
* 系统支持多种分享方式，并已集成必要的API接口。
* 如果选择通过社交媒体分享，玩家需要登录相应的账户。

基本流程：

* 玩家选择‘分享’选项。
* 系统展示分享方法的选择界面（如识别码、链接、二维码、社交媒体按钮）。
* 玩家选择分享方式。
* 根据所选方式，系统生成相应格式的分享内容（如URL、二维码图像）。
* 如果玩家选择了社交媒体分享，系统大爱预填的分享窗口。
* 玩家确认并发送分享内容。

替代流程：

* 如果系统无法生成有效的分享内容或连接到外部服务师表，提示错误。
* 如果玩家取消分享操作，返回游戏主界面而不执行分享。

后置条件：分享内容成功传递给接收方，玩家可以选择继续游戏或其他活动。

特殊要求： 无。

1. 用例名称：高亮提醒

参与者：玩家

简要描述：玩家可通过设置设定，当玩家选择了一个单元格后，是否高亮同一行/列/九宫格的单元格，是否高亮与该单元格有相同数字的单元格，是否高亮与该单元格数字相冲突的单元格。

前置条件：玩家正在游戏中，游戏界面上存在可交互的单元格。

基本流程：

* 玩家选择'设置'。
* 设置菜单栏提供不同高亮提醒类型（同行/列/九宫格、相同数字、冲突数字）的开关选择。
* 玩家选择一个单元格。
* 系统根据玩家的选择应用高亮效果。
* 当玩家取消选择或移动到其他单元格时，高亮效果随之更新。

替代流程：如果玩家选择了一个确定值单元格，系统只显示数字相同的单元格。

后置条件：高亮提醒及时反馈与更新。

特殊要求： 高亮颜色应当足够醒目但不会干扰游戏体验。

1. 用例名称：计时器显示

参与者：玩家

简要描述：玩家可通过设置设定在游玩过程中是否显示计时器。

前置条件：游戏具备计时功能。

基本流程：

* 玩家选择'设置'。
* 设置菜单提供计时器显示开关选择。
* 计时器的显示随玩家选择更新。

替代流程：如果玩家选择不显示计时器，系统仍然保持计时以便于统计最终成绩。

后置条件：计时器显示设置被保存，后续游戏会话界面根据选择进行计时器的显示。

特殊要求： 计时器应当精确到秒，并能够在暂停游戏时正确暂停和恢复。

1. 用例名称：自定义提示次数限制

参与者：玩家

简要描述：玩家可通过设置设定是否进行提示次数限制，如果选择设置，还可以自定义具体次数。

前置条件：游戏提供提示功能，并允许玩家在设置中调整提示相关的参数。

基本流程：

* 玩家选择'设置'。
* 设置菜单提供提示次数限制的开关选择。
* 如果开启，玩家还需输入具体的提示次数（默认为5）。
* 系统保存设置并在玩家请求提示时检查剩余次数。
* 每次玩家使用提示，系统减少可用次数并更新显示。

替代流程：

* 如果玩家试图超出设定的提示次数，系统拒绝提供额外提示，并通知玩家剩余次数不足。
* 如果玩家关闭了提示次数限制，系统不限制提示的使用。

后置条件：提示次数限制设置被保存，并进行更新。

特殊要求： 提供清晰的提示次数显示，让玩家随时了解剩余次数。

# 领域模型

图示

描述已自动生成

**具体实体分析**

**1. SudokuGame (数独游戏)**

* **职责**：游戏的核心逻辑和全局状态管理
* **属性**：
  + difficulty：可用的所有游戏难度选项
  + currentDifficulty：当前选择的游戏难度。
  + gamePaused：游戏是否处于暂停状态。
  + gameWon：游戏是否已经获胜。
* **关系**：
  + 包含棋盘（SudokuBoard）和设置（Settings）。
  + 支持与提示（Hint）、用户输入（UserInput）、模态框（Modal）的交互。

**2. SudokuBoard (数独棋盘)**

* **职责**：管理数独棋盘的布局和状态。
* **属性**：
  + cells：二维数组，存储棋盘中的所有单元格（SudokuCell）。
  + shareCode：用于分享当前棋局的编码数据。
* **关系**：
  + 包含单元格（SudokuCell），是游戏的主要组件。
  + 当前的谜题可以供分享功能（Share）

**3. SudokuCell (数独单元格)**

* **职责**：表示棋盘上的一个单元格。
* **属性**：
  + cellX、cellY：单元格的坐标。
  + value：单元格的当前值（0 表示未填充）。
  + candidates：候选数字列表。
  + state：单元格的状态（如已填充、冲突，候选值状态，高亮状态等）。
* **关系**：
  + 被棋盘（SudokuBoard）包含。

**4. Settings (设置)**

* **职责**：存储用户的游戏偏好设置。
* **属性**：
  + showTimer：是否显示计时器。
  + highlightConflicts：是否高亮冲突。
  + highLightNeighbour：是否高亮单元格。
  + limitHints：是否限制提示次数上限。
  + hintLevel：提示等级。
  + hintNum：每局可使用的提示次数。
* **关系**：
  + 被 SudokuGame使用，使用调整游戏的基础设置。
  + 直接管理计时器（Timer）
  + 被Modal弹出

**5. Timer (计时器)**

* **职责**：记录游戏的时间。
* **属性**：
  + elapsedTime：已用时间。
  + running：计时器是否正在运行。
* **关系**：
  + 由 Settings 管理，用于游戏的时间控制。

**6. Hint (提示)**

* **职责**：提供游戏提示功能。
* **属性**：
  + - hintMessage：包含提示信息的字符串数组。
    - hintResult：提示的结果（如某些单元格的值或候选）
    - success：提示是否成功
* **关系**：
  + - 由 SudokuGame 提供，辅助玩家完成游戏。

**7. Share (分享)**

* **职责**：负责当前棋局的分享功能。
* **属性**：
  + shareData：分享数据（可能是棋局编码）。
  + type：分享的方式或类型（如二维码，棋局编码，脸书等）。
* **关系**：
  + 与 SudokuBoard 交互，用于保存或传播棋局。

**8. Modal（弹窗显示）**

* **职责**：提供游戏信息的弹窗显示（如设置，游戏结束信息等）。
* **属性**：
  + displayMessage：展示信息的数据内容。
  + displayType：展示信息的类型（如提示框、警告框、游戏结束等）。
* **关系**：
  + 对SudokuGame进行响应，弹窗显示不同的信息。
  + 特别的，可以弹出设置菜单

**9. UserInput (用户输入)**

* **职责**：处理用户输入。
* **属性**：
  + selectedCellX、selectedCellY：用户当前选择的单元格。
  + input：用户输入的值。
  + candidateMode：是否启用候选模式。
* **关系**：
  + 与 SudokuGame 交互，捕获和处理用户操作。

**10. State\_Reversion (状态回溯)**

* **职责**：实现游戏的状态回溯（如撤销、重做）。
* **属性**：
  + canUndo、canRedo、canBackTrack：是否支持对应功能。
  + isBranches：保存的不同历史状态是否是分支路径。
  + preStates：历史状态的记录
* **关系**：
  + 与 SudokuGame 集成，用于状态管理。

**11. Custom\_Resource (自定义资源)**

* **职责**：提供外部棋盘资源
* **属性**：
  + resourceURL：资源的 URL。
* **关系**：
  + 与 SudokuBoard 交互，加载自定义资源。

# 系统技术架构

这个项目是一个前端主导、轻量级的基于 **Svelte** 框架构建的数独游戏应用。

**总体架构模式**是一个分层架构。

**前端：主逻辑处理和用户交互**

* 使用 Svelte 框架进行 UI 和状态管理开发。
* 游戏逻辑嵌入前端代码中，具备组件化、模块化特点。

**核心模块**：

* 数独逻辑的生成、验证和求解。
* 状态管理与事件驱动的交互。
* 配置与可视化功能扩展。

**架构分层描述**

**(1) 应用层（Application Layer）**

负责主要的数独游戏逻辑与状态管理。包括：

* 游戏状态的维护：
  + 通过 Svelte 的 store（如 writable, derived）实现全局状态管理。
* 数独逻辑：
  + 提供棋盘生成、验证、求解的核心逻辑算法。
* 用户事件处理：
  + 响应用户的点击、输入等操作。

**(2) 组件层（Component Layer）**

用户界面的呈现和交互功能封装。主要组件包括：

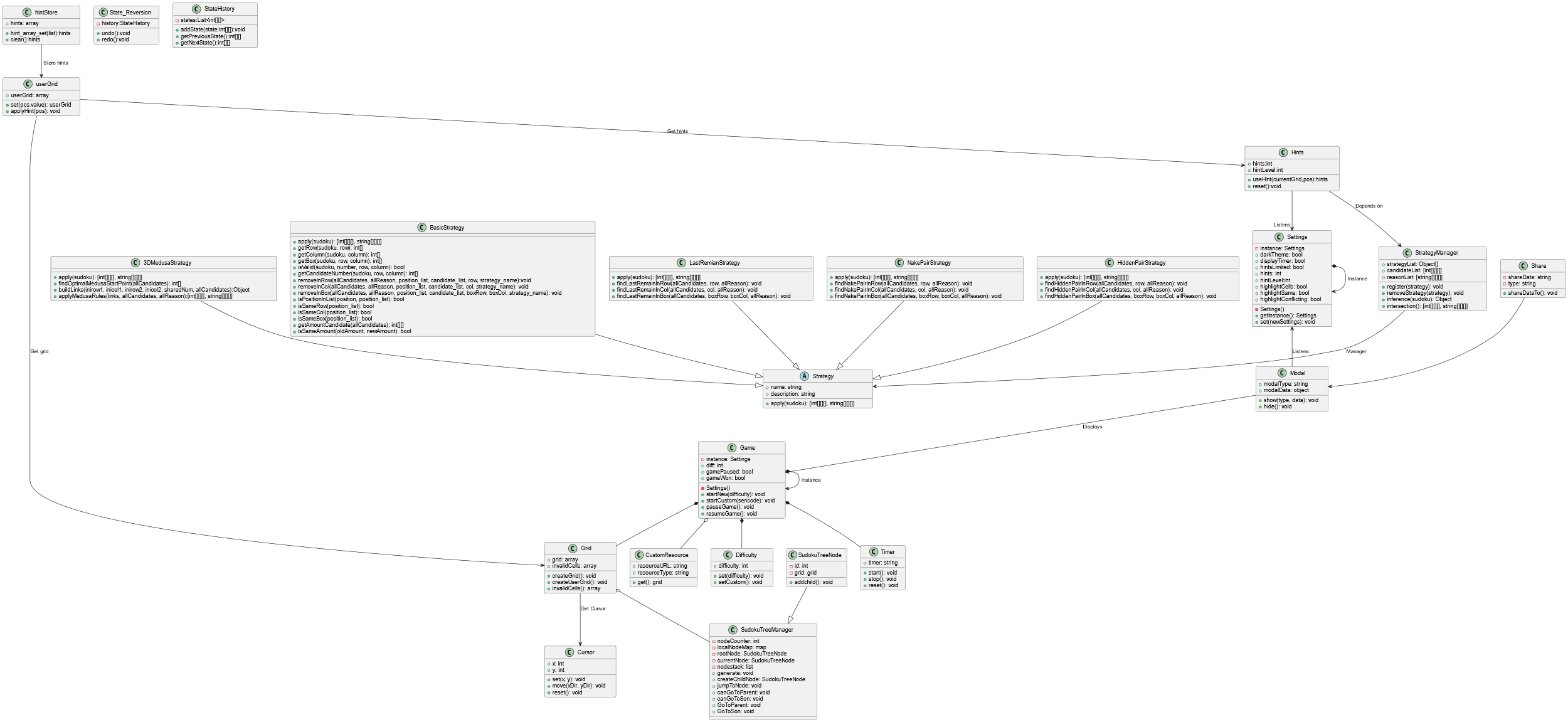
* **游戏界面组件**：渲染棋盘并实现交互功能。
* **设置组件**：控制显示选项（如时间、候选值高亮等）。
* **分享组件**：提供棋盘分享功能，生成或解析分享码。
* **提示显示组件**：提供棋盘提示来源信息提示
* **导入组件**：提供数独wiki上内容的导入功能。

**(3) 工具层（Utility Layer）**

实现数独功能相关的独立工具函数。包括：

* 数独算法工具：
  + 棋盘的生成、验证、求解逻辑封装。
* 分享工具：
  + 编码和解码棋盘为分享码。
* 时间管理工具：
  + 提供计时器功能。
* 状态回溯工具：
  + 提供状态回溯功能

# 对象模型



**对象模型作图代码**

@startuml  
class Game {  
 -instance: Settings  
 +diff: int  
 +gamePaused: bool  
 +gameWon: bool  
 -Settings()  
 +startNew(difficulty): void  
 +startCustom(sencode): void  
 +pauseGame(): void  
 +resumeGame(): void  
}  
  
class Grid {  
 +grid: array  
 +invalidCells: array  
 +createGrid(): void  
 +createUserGrid(): void  
 +invalidCells(): array  
}  
  
class userGrid{  
 +userGrid: array  
 +set(pos,value): userGrid  
 +applyHint(pos): void  
}  
  
class hintStore{  
 +hints: array  
 +hint\_array\_set(list):hints  
 +clear():hints  
}  
  
class Hints{  
 +hints:int  
 +hintLevel:int  
 +useHint(currentGrid,pos):hints  
 +reset():void  
}  
  
class Difficulty {  
 +difficulty: int  
 +set(difficulty): void  
 +setCustom(): void  
}  
  
class Timer {  
 +timer: string  
 +start(): void  
 +stop(): void  
 +reset(): void  
}  
  
class CustomResource {  
 +resourceURL: string  
 +resourceType: string  
 +get(): grid  
}  
  
class Cursor {  
 +x: int  
 +y: int  
 +set(x, y): void  
 +move(xDir, yDir): void  
 +reset(): void  
}  
  
class Modal {  
 +modalType: string  
 +modalData: object  
 +show(type, data): void  
 +hide(): void  
}  
  
class Settings {  
 -instance: Settings  
 +darkTheme: bool  
 +displayTimer: bool  
 +hintsLimited: bool  
 +hints: int  
 +hintLevel:int  
 +highlightCells: bool  
 +highlightSame: bool  
 +highlightConflicting: bool  
 -Settings()  
 +getInstance(): Settings  
 +set(newSettings): void  
}  
  
class Share {  
 -shareData: string  
 -type: string  
 +shareDataTo(): void  
}  
  
class State\_Reversion{  
 -history:StateHistory  
 +undo():void  
 +redo():void  
}  
  
class StateHistory{  
 -states:List<int[][]>  
 +addState(state:int[][]):void  
 +getPreviousState():int[][]  
 +getNextState():int[][]  
}  
  
class SudokuTreeNode {  
 - id: int  
 - grid: grid  
 + addchild(): void  
}  
  
class SudokuTreeManager {  
 - nodeCounter: int  
 - localNodeMap: map  
 - rootNode: SudokuTreeNode  
 - currentNode: SudokuTreeNode  
 - nodestack: list  
 + generate: void  
 + createChildNode: SudokuTreeNode  
 + jumpToNode: void  
 + canGoToParent: void  
 + canGoToSon: void  
 + GoToParent: void  
 + GoToSon: void  
}  
  
  
abstract class Strategy {  
 + name: string  
 + description: string  
 + apply(sudoku): [int[][][], string[][][]]  
}  
  
  
class 3DMedusaStrategy {  
 + apply(sudoku): [int[][][], string[][][]]  
 + findOptimalMedusaStartPoint(allCandidates): int[]  
 + buildLinks(inirow1, inicol1, inirow2, inicol2, sharedNum, allCandidates):Object  
 + applyMedusaRules(links, allCandidates, allReason):[int[][][], string[][][]]  
}  
  
  
class BasicStrategy {  
 + apply(sudoku): [int[][][], string[][][]]  
 + getRow(sudoku, row): int[]  
 + getColumn(sudoku, column): int[]  
 + getBox(sudoku, row, column): int[]  
 + isValid(sudoku, number, row, column): bool  
 + getCandidateNumber(sudoku, row, column): int[]  
 + removeInRow(allCandidates, allReason, position\_list, candidate\_list, row, strategy\_name):void  
 + removeInCol(allCandidates, allReason, position\_list, candidate\_list, col, strategy\_name): void  
 + removeInBox(allCandidates, allReason, position\_list, candidate\_list, boxRow, boxCol, strategy\_name): void  
 + isPositionInList(position, position\_list): bool  
 + isSameRow(position\_list): bool  
 + isSameCol(position\_list): bool  
 + isSameBox(position\_list): bool  
 + getAmountCandidate(allCandidates): int[][]  
 + isSameAmount(oldAmount, newAmount): bool  
}  
  
  
class LastRemianStrategy {  
 + apply(sudoku): [int[][][], string[][][]]  
 + findLastRemainInRow(allCandidates, row, allReason): void  
 + findLastRemainInCol(allCandidates, col, allReason): void  
 + findLastRemainInBox(allCandidates, boxRow, boxCol, allReason): void  
}  
  
class NakePairStrategy {  
 + apply(sudoku): [int[][][], string[][][]]  
 + findNakePairInRow(allCandidates, row, allReason): void  
 + findNakePairInCol(allCandidates, col, allReason): void  
 + findNakePairInBox(allCandidates, boxRow, boxCol, allReason): void  
}  
  
  
class HiddenPairStrategy {  
 + apply(sudoku): [int[][][], string[][][]]  
 + findHiddenPairInRow(allCandidates, row, allReason): void  
 + findHiddenPairInCol(allCandidates, col, allReason): void  
 + findHiddenPairInBox(allCandidates, boxRow, boxCol, allReason): void  
}  
  
  
class StrategyManager {  
 + strategyList: Object[]  
 + candidateList: [int[][][]]  
 + reasonList: [string[][][]]  
 + register(strategy): void  
 + removeStrategy(strategy): void  
 + inference(sudoku): Object  
 + intersection(): [int[][][], string[][][]]  
}  
  
  
  
  
  
Game \*-- Grid  
Game \*-- Difficulty  
Game \*-- Timer  
Game o-- CustomResource  
Grid --> Cursor : "Get Cursor"  
Grid o-- SudokuTreeManager  
Hints --> "Listens" Settings  
Settings <-- Modal:Listens  
Share --> Modal  
Hints --> StrategyManager : "Depends on"  
Modal --> Game:Displays  
Settings \*--> Settings: Instance  
Game \*--> Game:Instance  
SudokuTreeNode --|> SudokuTreeManager  
userGrid --> Grid: "Get grid"  
userGrid --> Hints: "Get hints"  
hintStore --> userGrid: " Store hints"  
StrategyManager --> Strategy: "Manager"  
3DMedusaStrategy --|> Strategy  
BasicStrategy --|> Strategy  
LastRemianStrategy --|> Strategy  
NakePairStrategy --|> Strategy  
HiddenPairStrategy --|> Strategy  
@enduml

# 设计模式和设计原则

**设计原则说明**

从 UML 类图分析，该项目的设计遵循了多个**软件设计原则**和**设计模式**，如下：

**设计原则**

1. **单一职责原则（Single Responsibility Principle, SRP）**

每个类只负责一项职责。如 Game 负责游戏的逻辑控制，Grid 负责生成和操作网格，Timer 管理时间，Settings 管理设置等。

通过职责分离，简化了类的复杂性，便于维护和扩展。

1. **开放封闭原则（Open-Closed Principle, OCP）**

类可以通过扩展来增强功能，而不需要修改已有的代码。如 SolveMethod 是抽象类，其具体实现通过子类 SolveMethod\_3DMedusa 和 SolveMethod\_SKLoops 来扩展功能。

新的解法可以通过继承 SolveMethod 来实现，不会影响现有代码。

1. **里氏替换原则（Liskov Substitution Principle, LSP）**

子类可以替代父类使用。如 SolveMethod 的子类可以在任何需要 SolveMethod 的地方使用，保证了代码的兼容性。

1. **接口隔离原则（Interface Segregation Principle, ISP）**

类图中各个类职责明确，没有强迫实现无关的方法。如 Modal 专注于显示和隐藏弹窗，不涉及其它功能。

1. **依赖倒置原则（Dependency Inversion Principle, DIP）**

高层模块（如 Game）不直接依赖低层模块，而是通过抽象接口进行依赖。如 SolveMethod 是一个抽象类，具体的实现通过其子类实现。

**设计模式**

1. **单例模式（Singleton Pattern）**

用于 Settings 和 Game的实例管理。如 Settings 的 getInstance() 方法确保了只有一个 Settings 实例，符合单例模式的特性。

1. **工厂模式（Factory Pattern）**

隐含在 CustomResource 中，通过 resourceURL 和 resourceType提供不同类型的网格资源。该模式可以用于动态生成或加载资源，适配不同的需求。

1. **观察者模式（Observer Pattern）**

用于 Hints 和 Settings 的交互，以及 Modal 和 Game的显示逻辑。如 Hints 监听 Settings 的状态变化，当设置变更时，Hints 会自动调整。

1. **策略模式（Strategy Pattern）**

体现在 Strategy 的设计中，通过不同子类实现不同解法策略（如 3DMedusaStrategy 和 NakePairStrategy）。可以动态选择解法策略，增强灵活性。

1. **备忘录模式（Memento Pattern）**

体现在 State\_Reversion 和 StateHistory的实现中。StateHistory 保存网格状态的历史记录，State\_Reversion 提供 undo 和 redo 的功能，帮助恢复之前的状态。

1. **依赖注入模式（Dependency Injection Pattern）**

用于依赖的动态传递。如 Hints 和 Strategy和StrategyManager 的依赖通过外部注入，而不是硬编码，提升了模块的解耦性。

1. **组合模式（Composite Pattern）**

体现在 Grid 和 userGrid的关系中。userGrid 是 Grid 的一个特化，能够继承 Grid 的结构和方法。

1. **命令模式（Command Pattern）**

隐含在 Game 的方法中，如 pauseGame 和 resumeGame。这些命令方法可以被封装为可执行命令，支持灵活调用。