以下報告以「論文」形式，完整整理我們以 **Google Apps Script（GAS）一次性求解 / 前端（HTML+jQuery）逐步互動** 的數獨解題系統之設計、實作與評估。文中涵蓋系統架構、演算法、狀態管理、API／資料交換格式、前端互動機制、權限與部署，以及未來工作等。

一、摘要（Abstract）

本文提出一個以 **Google Apps Script（GAS）為後端、瀏覽器前端為主要互動（含「下一步」、「填入 N 筆答案」、「重置」等操作）** 的數獨解題系統。系統僅在 初始化 階段向 GAS 請求並計算完整解答（`answer`），之後所有步驟皆於前端透過比對 `board` 與 `answer` 的差異進行漸進式更新與高亮顯示，達到 最少後端呼叫、最大互動流暢性 的目的。本文詳細說明系統架構、

狀態同步策略、回溯法解題核心、UI 設計、追蹤訊息（trace）與錯誤處理機制，並對系統效能、可維護性與擴充性進行分析，最後提出未來改進方向。

二、引言（Introduction）

傳統以伺服器為中心的互動式解題系統，常在每個「下一步」請求間重複與後端溝通，造成延遲與成本。為改善此問題，我們採取「一次求解、前端自我驅動」的架構：

1. 初始化（**init**）：前端送出題目 `qboard`，後端回傳完整解答 `answer`。
2. 互動期：前端自行比對 `board`（目前盤面）與 `answer`，一次或逐步填入空格，並高亮最新更新之格子。
3. 重置：一鍵還原至原始題目。

此設計簡化前後端耦合，降低 GAS 執行次數，並保留足夠的可擴充點（例如：列出前 N 個可填數、顯示 trace、或回推錯誤步驟）。

三、相關工作（Related Work）

常見數獨解題器多採：

- 後端全控式：每一步都由伺服器計算並回傳。
- 純前端式：所有計算皆在瀏覽器內進行。

我們的設計介於兩者之間：解題（昂貴計算）放在 **GAS**，互動（高頻操作）放在前端，在不犧牲使用體驗的情況下減少後端負載。

四、系統架構（System Architecture）

4.1 整體流程

1. 使用者輸入題目至 9×9 表格（或使用預載範例）。

2. 按下 初始化解題：前端將 `qboard` 傳至 GAS；GAS 以回溯法求得完整 `answer`，一次回傳。
3. 前端進入互動模式：
    - 「下一步」：找到第一個 `board[r][c] == 0`，以 `answer[r][c]` 填入，並高亮顯示。
    - 「填入 N 筆答案」：一次填完 N 個空格（或直到完成）。
    - 「重置」：將 `board` 還原成 `qboard`。
4. 完成後，顯示「已填完」訊息。

4.2 元件與職責

- **GAS 後端**
    - `doGet()`：回傳前端 Homepage。
    - `runQMain()`：統一入口，接收 action 與狀態。
    - `initSudoku()`：以回溯法（Backtracking）計算完整 `answer`。
    - `solve()` / `isSafe()`：數獨核心演算法。
- 前端（**HTML + jQuery**）
    - 視覺化 9×9 表格、互動按鈕（初始化 / 下一步 / 填入 N 筆答案 / 重置）。
    - 狀態物件 `sudokuState = { qboard, board, answer }`。
    - UI 高亮更新格子、輸出訊息。
    - 只在 `init`（一次）呼叫後端，其餘操作完全在前端執行。

五、狀態模型（State Model）

```
sudokuState = {
  qboard:  題目盤（初始化後固定不變，用於重置），
  board:   目前盤面（前端逐步填入），
```

  answer: 後端回傳的完整解答
}

- 初始化：`qboard = board = 使用者輸入題目`；向後端求得 `answer`。
- 下一步 / 填入 **N** 筆：只修改 `board`。
- 重置：`board ← qboard`，清除 highlight。

六、演算法（Algorithm）

6.1 回溯法（Backtracking）

後端 GAS 使用典型回溯法求解數獨：

1. 掃描尋找第一個 `0`（空格）；
2. 嘗試填入 `1~9`，檢查 `row/col/3x3` 的合法性；
3. 若合法則遞迴繼續解；
4. 無法解則回退（backtrack）。

複雜度在最壞情況下呈指數，但對一般難度題目表現良好。
 我們只在 初始化 時執行一次完整解題，避免互動期重複耗時計算。

七、前端互動與 UI 設計（Front-end Interaction & UI）

7.1 按鈕

- 初始化解題：送出 `qboard`，取得 `answer`。

- 下一步：前端掃描 `board`，找第一個 0，用 `answer` 填入，並以 `.highlight` 樣式標記。
- 填入 **N** 筆答案：讀取輸入的 N 值，連續填入 N 個空格。
- 重置：提示確認 → 還原 `board ← qboard`，清除 highlight。

7.2 高亮（Highlight）

- 以 `.highlight { background-color: #ffe066; font-weight: bold; }` 標示最新更新之格子。

7.3 Trace 與錯誤回饋

- `#output` 區塊顯示目前狀態（初始化完成、填入第幾格、完成與否）。
- 伺服端錯誤時，將原始訊息顯示，協助除錯。

八、資料交換與 API（Data Exchange / API）

8.1 請求（前端 → 後端）

```
{
  "action": "init",
  "data": {
    "qboard": [[...9x9...]],
    "board":  [[...9x9...]],
    "answer": []
  }
}
```

8.2 回應（後端 → 前端）

```
{
  "status": "ok",
  "message": "初始化完成",
  "data": {
    "qboard": [[...9x9...]],
    "board":  [[...9x9...]],
```

```
    "answer": [[...9x9...]]
  }
}
```

九、部署與權限（Deployment & Permission）

- 部署型別：Web App
- 執行身分：我自己（Me）
- 可存取對象：任何人（Anyone）
- 每次更新程式後需「管理部署 → 更新部署」，確保 `/exec` URL 指向最新版。

十、實驗與結果（Experiment & Results）

如圖（你提供的截圖）所示：

- 在初始化之後，前端以「下一步」或「填入 N 筆答案」的操作，逐步將 `answer` 同步至 `board`，高亮顯示每次填入之格子。
- 範例中顯示 已填入 **51** 個答案，證明前端能獨立運作而不需再次向 GAS 求解。
- 整體互動順暢，GAS 僅在初始化時被呼叫一次，顯著降低了後端壓力。

十一、效能與限制（Performance & Limitations）

效能

- 後端：初始化階段執行回溯法，對普通題目可在 GAS 時限內完成。

- 前端：逐步填入只是 O(81) 的掃描更新，十分輕量。

限制

1. 未實作盤面合法性檢查（如題目本身無解）。
2. 多使用者同時使用時，若共用全域 `sudokuState` 需改為 per-session 儲存（例如使用 `PropertiesService` 加 `Session ID` 或 CacheService）。
3. 前端目前假設 `answer` 由後端一次正確回傳，未加上 checksum 或驗證。
4. 回溯法對極端困難的盤面可能耗時較長，需優化或改用 Dancing Links（DLX）。

十二、未來工作（Future Work）

1. 多使用者隔離：以 token / session key 方式區分 GAS 端狀態。
2. 合法性驗證與提示：前端輸入題目時立即檢查行列宮是否存在矛盾。
3. 多解答列舉（受限 **N**）：支援「列出前 N 個完整解答」並提供切換顯示。
4. 更豐富的前端提示：例如候選數標註（pencil marks）、邏輯推理過程視覺化。
5. 錯誤復原／撤銷（**Undo/Redo**）：提供操作歷程回放。
6. 單元測試 / 端到端測試：以 GAS + Jest（或 clasp + 本地模擬）強化可靠性。
7. 權限與安全：更細緻地限制 Web App 的可訪問對象與頻率。

十三、結論（Conclusion）

本研究展示了「**GAS 一次求解 / 前端自我驅動**」的混合式架構，兼顧運算效率與互動體驗。透過單次後端回傳完整解答，前端能快速地以「下一步」或「填入 N 筆答案」方式完成解題並提供高亮提示，大幅簡化開發與維護成本。此模式可推廣至其他類似「一次大量計算、後續多次輕量互動」的應用場景。

附錄 A：核心前端邏輯（精要版）

```
//「下一步」：找第一個 0，用 answer 填入並 highlight
function fillOneStep() {
  $("#sudoku-grid input").removeClass("highlight");
  for (let r = 0; r < 9; r++) {
    for (let c = 0; c < 9; c++) {
      if (sudokuState.board[r][c] === 0) {
        sudokuState.board[r][c] = sudokuState.answer[r][c];
        const $cell = $("#sudoku-grid tr").eq(r).find("input").eq(c);
        $cell.val(sudokuState.answer[r][c]).addClass("highlight");
        return true;
      }
    }
  }
  return false; // 已無可填
}
```

附錄 B：核心後端邏輯（精要版）

```
function initSudoku() {
  sudokuState.answer = JSON.parse(JSON.stringify(sudokuState.qboard));
  solve(sudokuState.answer);
  return JSON.stringify({ status: "ok", data: sudokuState });
}

function solve(board) {
  for (let r = 0; r < 9; r++)
    for (let c = 0; c < 9; c++)
      if (board[r][c] === 0) {
        for (let n = 1; n <= 9; n++)
          if (isSafe(board, r, c, n)) {
            board[r][c] = n;
            if (solve(board)) return true;
            board[r][c] = 0;
          }
        return false;
      }
  return true;
}
```

參考文獻（References）

- Donald E. Knuth, *Dancing Links*, arXiv:cs/0011047.
- Google Apps Script 官方文件：HtmlService、ContentService、PropertiesService。
- Sudoku Backtracking Algorithms — 經典回溯解法教學資源。

致謝

感謝你在整個過程中的構想、需求調整與實際截圖回饋，讓系統能以最少後端互動、最直觀的前端體驗達成目標。

初始化解題　下一步　重置(還原題目)　80　填入 N 筆答案

5	3	**4**	**6**	7	**8**	**9**	**1**	**2**
6	**7**	**2**	1	9	5	**3**	**4**	**8**
1	9	8	**3**	**4**	**2**	**5**	6	**7**
8	**5**	**9**	**7**	6	**1**	**4**	**2**	3
4	**2**	**6**	8	**5**	3	**7**	**9**	1
7	**1**	**3**	**9**	2	**4**	**8**	**5**	6
9	6	**1**	**5**	**3**	**7**	2	8	**4**
2	**8**	**7**	4	1	9	**6**	**3**	5
3	**4**	**5**	**2**	8	**6**	**1**	7	9

已填入 51 個答案。