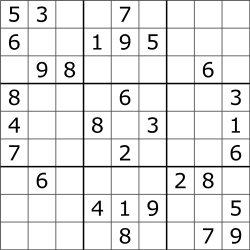
**CHỦ ĐỀ: GAME SUDOKU VÀ THIẾT KẾ API SUDOKU**

**1, Giới thiệu về game Sudoku**

Sudoku (数独 (số độc) ban đầu có tên gọi là Number Place) là một trò chơi câu đố sắp xếp chữ số dựa trên logic theo tổ hợp. Mục tiêu của trò chơi là điền các chữ số vào một lưới (grid) 9×9 sao cho mỗi cột, mỗi hàng, và mỗi phần trong số chín lưới con 3×3 cấu tạo nên lưới chính (cũng gọi là "hộp" (box), "khối" (block), hoặc "khu vực" (region)) đều chứa tất cả các chữ số từ 1 tới 9. Cơ cấu điều chỉnh câu đố cung cấp một lưới đã được hoàn thành một phần, mỗi câu đố được thiết lập tốt có một giải pháp hoàn thành duy nhất.



*Game sudoku*

Dưới đây là đáp án cho bảng sudoku trên:



*Lời giải*

**2, Thiết kế API cho trò chơi**

Để phục vụ cho việc viết 1 game sudoku, ta cần phải giải quyết 1 số vấn đề chính sau:

* Sinh 1 bảng sudoku ngẫu nhiên với n ô bị xóa
* Giải 1 sudoku với n ô bị xóa
* Kiểm tra một bảng sudoku có đúng hay không

Trước tiên, để biểu diễn 1 bảng sudoku bằng ngôn ngữ lập trình, ta có nhiều cách biểu diễn (Mảng 2 chiều, danh sách liên kết, …). Chắc chắn mỗi cách biểu diễn đều có những ưu điểm và nhược điểm trong các trường hợp cụ thể. Ở bài viết này, tôi chọn biểu diễn bằng một bảng 2 chiều, lý do chọn cách biểu diễn này là vì nó đơn giản và dễ sử dụng, quy mô ứng dụng bé (first project)

Như vậy, ta có thể hình dung công việc phải làm như sau. Ta sẽ thiết kế 1 lớp SudokuSolver để cung cấp các API này cho chương trình:

* Int[][] getRandomGrid (int numberOfEmptyCells): Trả về 1 sudoku với numberOfEmptyCells rỗng
* Int[][] getSolution(int[][] grid): Trả về 1 sudoku hoàn chỉnh
* Boolean check (int[][] grid): kiểm tra grid có hợp lệ hay không

2.1. Sinh 1 sudoku ngẫu nhiên

Công việc sinh 1 sudoku ngẫu nhiên tương đối đơn giản, ta chỉ cần quay lui từ ô (0, 0) đến ô (8, 8), thời gian sinh ra 1 sudoku không đáng kể. Tuy nhiên, cần lưu ý là để sudoku được ngẫu nhiên ta phải random lại bộ giá trị cho mỗi ô đang xét. Tức là nếu ta chỉ random một lần đầu tiên, thứ tự các ô sau hoàn toàn có thể đoán ra được dựa vào quy luật sinh. Việc random mỗi lần chọn giá trị cho mỗi ô đảm bảo không có cách nào đoán ra quy luật sinh. Dưới đây là code tham khảo cho thuật toán trên:

**void** generate(**int** pos) {  
 **if** (pos == 81) {  
 **genFlag** = **false**;  
 } **else if** (**genFlag**) {  
 **int** row = pos / 9;  
 **int** col = pos - row \* 9;  
 **int** box = (row / 3) \* 3 + col / 3;  
 randomValuesArray(5);  
 **for** (**int** val : **values**) {  
 **if** (**genFlag** && **rowSet**[row][val] == **false** && **colSet**[col][val] == **false** && **boxSet**[box][val] == **false**) {  
 **grid**[row][col] = val;  
 **rowSet**[row][val] = **true**;  
 **colSet**[col][val] = **true**;  
 **boxSet**[box][val] = **true**;  
 generate(pos + 1);  
 **rowSet**[row][val] = **false**;  
 **colSet**[col][val] = **false**;  
 **boxSet**[box][val] = **false**;  
 }  
 }  
 }  
 }

2.2. Tạo ra một sudoku với một số ô rỗng

Công việc này tưởng đơn giản nhưng lại không hề dễ dàng. Ý tưởng đầu tiên ta nghĩ ra là random một số vị trí rồi xóa đi là xong. Tuy nhiên, nếu ta xóa đi ngẫu nhiên một số ô sẽ không thể đảm bảo rằng bảng sudoku này sẽ có duy nhất một đáp án.

Để thỏa mãn được tính chất “duy nhất”, ta có thể vừa sinh vừa quay lui giải sudoku để đếm số đáp án. Tuy nhiên cách tiếp cận này rất tốn tài nguyên, chỉ đáp ứng được với số lượng ô rỗng ít. Để giải quyết triệt để bài toán này một cách trực tiếp là điều cực kì khó. Tuy nhiên, ta còn cách giải quyết vấn đề này bằng cách tính toán hết mọi thứ, sau đó lưu kết quả vào cơ sở dữ liệu rồi lấy ra mỗi lần chạy. Đây là cách mà các game sudoku được đầu tư trên store làm, tuy nhiên ta phải thiết kế nhiều thứ hơn mất tốn công sức hơn.

Xét trên khía cạnh khác, ta có thể thấy rằng việc có nhiều Solution không thực sự ảnh hưởng nhiều đến trò chơi, có chăng chỉ làm độ khó tăng lên đòi hỏi người chơi phải “xoắn não” hơn, góp phần thú vị hơn cho trò chơi. Chỉ cần người chơi đưa ra 1 solution hợp lệ, solution đó được chấp nhận. Vì vậy, ta sẽ theo hướng giải quyết này, tức là sẽ random xóa một số ô ngâu nhiên, vừa đơn giản vừa đỡ tốn công sức

2.3. Giải một sudoku

Việc giải một sudoku tương tự như việc sinh một sudoku, ta sẽ quay lui tương tự theo cách sinh. Tuy nhiên ta có thể không cần đến phương thức giải sudoku khi ta là người sinh sudoku, tức là ta sẽ không thực sự xóa bất kì ô nào đi mà sẽ thực hiện đánh dấu ô đó là ô bị xóa và backup lại nếu cần. Trong API này, tôi sẽ sử dụng bit thứ 10 của ô bất kì là cờ, tức nếu bit này bằng 1 thì ô này bị xóa, ngược lại là ô nguyên vẹn. Cụ thể có thể xem ở phần cài đặt hoàn chỉnh

2.4. Kết luận

Trên đây là ba bài toán chủ yếu mà một game sudoku cần giải quyết, mỗi bài toán đều đã có cách giải quyết. Dựa trên thiết kế này tôi sẽ viết API hỗ trợ xây dựng một game sudoku, để xem cài đặt chi tiết có thể vào file: [SudokuSolver.java](https://github.com/tuananhcnt55vmu/Sudoku-Android/blob/master/app/src/main/java/sudoku/hfad/com/gridviewexample/SudokuSolver.java) (có cập nhật theo project)

**3. Cài đặt API**

**package** android.hfad.com.sudoku;  
  
**import** java.util.Random;  
  
*/\*\*  
 \* Created by tuana on 10-03-2018.  
 \*/***public class** SudokuSolver {  
 **private int**[] **values** = {1, 2, 3, 4, 5, 6, 7, 8, 9};  
 **private** Random **rnd** = **new** Random();  
 **private int**[][] **grid**;  
 **private boolean**[][] **rowSet**, **colSet**, **boxSet**;  
 **private boolean genFlag**;  
  
 **public** SudokuSolver() {  
 **grid** = **new int**[9][9];  
 **rowSet** = **new boolean**[9][10];  
 **colSet** = **new boolean**[9][10];  
 **boxSet** = **new boolean**[9][10];  
 randomValuesArray(10);  
 }  
  
 **void** clearGrid() {  
 **for** (**int** r = 0; r < 9; ++r) {  
 **for** (**int** c = 0; c < 9; ++c) {  
 **grid**[r][c] = 0;  
 }  
 }  
 **for** (**int** i = 0; i < 9; ++i) {  
 **for** (**int** v = 1; v <= 9; ++v) {  
 **rowSet**[i][v] = **false**;  
 **colSet**[i][v] = **false**;  
 **boxSet**[i][v] = **false**;  
 }  
 }  
 }  
  
 **void** randomValuesArray(**int** numberOfTurns) {  
 **for** (**int** turn = 0; turn < numberOfTurns; ++turn) {  
 **int** randomIndex = 1 + **rnd**.nextInt(8);  
  
 **int** tmp = **values**[0];  
 **values**[0] = **values**[randomIndex];  
 **values**[randomIndex] = tmp;  
 }  
 }  
  
 **void** generate(**int** pos) {  
 **if** (pos == 81) {  
 **genFlag** = **false**;  
 } **else if** (**genFlag**) {  
 **int** row = pos / 9;  
 **int** col = pos - row \* 9;  
 **int** box = (row / 3) \* 3 + col / 3;  
 randomValuesArray(5);  
 **for** (**int** val : **values**) {  
 **if** (**genFlag** && **rowSet**[row][val] == **false** && **colSet**[col][val] == **false** && **boxSet**[box][val] == **false**) {  
 **grid**[row][col] = val;  
 **rowSet**[row][val] = **true**;  
 **colSet**[col][val] = **true**;  
 **boxSet**[box][val] = **true**;  
 generate(pos + 1);  
 **rowSet**[row][val] = **false**;  
 **colSet**[col][val] = **false**;  
 **boxSet**[box][val] = **false**;  
 }  
 }  
 }  
 }  
  
 **int**[][] getRandomGrid(**int** numberOfEmptyCells) {  
 clearGrid();  
 **genFlag** = **true**;  
 generate(0);  
 */\* erase cells \*/* **for** (**int** i = 0; i < numberOfEmptyCells; ) {  
 **int** row = **rnd**.nextInt(9);  
 **int** col = **rnd**.nextInt(9);  
 **if** (**grid**[row][col] <= 9) {  
 */\* the 10-th bit mark the cell is empty\*/* **grid**[row][col] |= 1024;  
 ++i;  
 }  
 }  
 **return grid**;  
 }  
  
  
 **public boolean** checkAcceptedGrid(**int**[][] grid) {  
 **boolean**[][] rowSet = **new boolean**[9][10];  
 **boolean**[][] colSet = **new boolean**[9][10];  
 **boolean**[][] boxSet = **new boolean**[9][10];  
  
 **for** (**int** row = 0; row < 9; ++row) {  
 **for** (**int** col = 0; col < 9; ++col) {  
 **int** value = grid[row][col];  
 **if** (value == 0) **return false**;  
 **int** box = (row / 3) \* 3 + col / 3;  
 **if** (rowSet[row][value] || colSet[col][value] || boxSet[box][value]) **return false**;  
 rowSet[row][value] = **true**;  
 colSet[col][value] = **true**;  
 boxSet[box][value] = **true**;  
 }  
 }  
  
 **return true**;  
 }  
  
}