Sudoku 代码质量分析

在进行 Sudoku 的代码质量分析时，我们并没有采用使用方法较为简单的 CppCheck 工具，因为 CppCheck 工具目前无法支持 installer 的下载，只能支持源代码编译使用，并且 CppCheck 工具多为静态分析，具有一定的局限性。因此我们选择更为流行、专业的工具 SonarCloud 与 SonarQube，也是目前业界使用的最为广泛的代码质量分析工具。

SonarQube 是一种开源的代码质量管理平台，旨在帮助开发团队发现和修复代码中的潜在问题。它提供了静态代码分析、代码覆盖率、代码复杂度、重复代码检测等功能。通过在代码构建过程中集成 SonarQube，开发团队可以及早发现并解决代码中的问题，以提高代码质量和可维护性。

SonarQube 支持多种编程语言，包括 Java、C＃、C / C ++、JavaScript、TypeScript 等。它提供了一个易于使用的 Web 界面，显示代码的分析结果和问题报告。你可以查看各种指标和图表，以了解项目的整体代码质量情况，并针对性地进行改进。

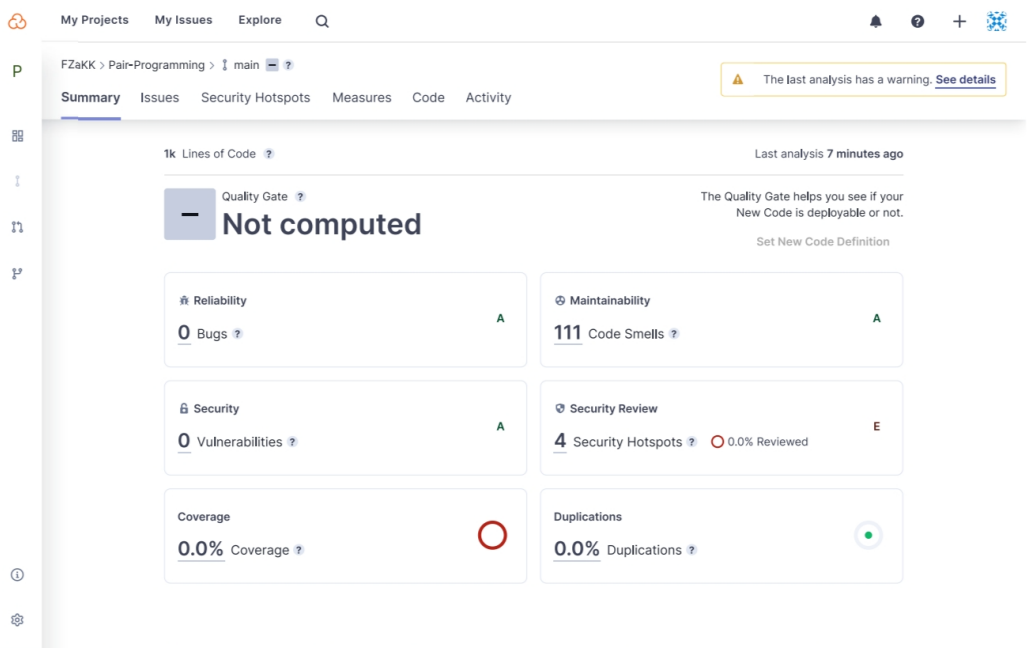
SonarCloud 是 SonarQube 的云托管版本，提供了与 SonarQube 相似的功能，但作为一项云服务，无需自行安装和管理。SonarCloud 适用于个人开发者、小型团队或不希望自行维护 SonarQube 服务器的组织。它允许你将代码仓库与 SonarCloud 集成，并自动进行代码质量分析。你可以在 SonarCloud 上查看代码质量报告和分析结果，以及与团队成员共享和讨论问题。

所以我们根据官网中的指导手册，来进行代码质量分析，使用 SonarCloud 将我们的 Github 代码仓库进行连接，并且在本地配置对应的 Java 和数据库环境，使用 Manually 的扫描方式即可。因为最终得到的报告为 Web 网页形式，无法支持导出 PDF 的形式，我们这里仅仅给出扫描结果的概述摘要。

在扫描的过程之中，我们需要注意的是，我们应当在 Windows 操作系统中集成好 make 命令以及 g++编译器，因为 g++编译器与 Visual Studio 中微软的默认编译器环境是有所不同的，所以我们需要下载好 MinGw 工具，并且配置好环境变量，获得 make 与 g++命令。接着，我们需要通过 make 命令进行 make clean all 的指令替换，才能够成功进行 SonarCloud 的扫描，同时我们需要注意将官网上的命令中的\符号，切换为 cmd 命令中能使用的^符号，去适配 cmd 命令。

至此，我们就能够成功运行 SonarCloud 中的 Manually 扫描方式，来进行代码质量分析并且获得质量分析报告了，通过代码质量分析报告，我们来消除 Bug 和 Warning 等各种问题。

我们得到的代码质量分析报告的概述如下：



根据上述的代码质量分析的报告中，可以看出，我们的程序已经没有任何 Bug 和 Vulnerabilities 了，我们已经成功消除了所有的 Bug 和警告，并且代码之中并没有什么重复的部分，基本上都是原创内容。对于 security hotpots 部分，我们经过详细研究发现是所有的 rand 部分随机数种子的问题，这里我们采用根据时间变化的随机数种子来使用类似于 srand 和 shuffle 等随机函数。我们也根据 Smell 部分中的一些代码规范问题，进行了修改与优化。

Sudoku 代码覆盖率测试

在代码覆盖率测试之中，我们采用的是 OpenCppCoverage 工具。该工具的使用方法为，使用命令：

OpenCppCoverage.exe --sources 源代码路径 -- 可执行文件路径

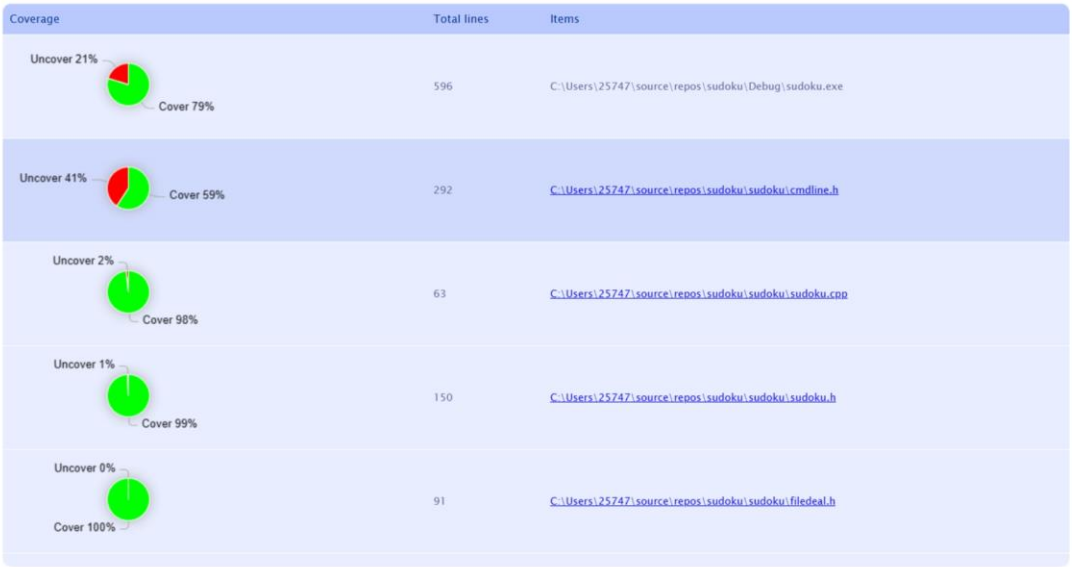
即可以成功获得测试率报告，并且还会给出具体的代码覆盖展示，样例如下：

```cpp
#pragma once
#include <iostream>
#include <fstream>
#include <vector>
#include <random>
#include <string>
#include <cstdlib>
#include <ctime>
#include <algorithm>
#include <Windows.h>
using namespace std;


const int N = 9;


int randEx(){
    LARGE_INTEGER seed;
    QueryPerformanceFrequency(&seed);
    QueryPerformanceCounter(&seed);
    srand(unsigned int(seed.QuadPart));

    return rand();
}


class sudoku{
public:
    bool active = false;

    sudoku() {
        active = false;
    }

    sudoku(vector<vector<char>>& board){
        // printBoard(board);
        active = dfs(board, 0);
    }

    void printBoard(vector<vector<char>>& board){
        cout << " |---|---|---|---|---|---|---|---|---| " << endl;
        for (int i = 0; i < int(board.size()); i++){
            for (int j = 0; j < int(board.size()); j++){
                cout << " | " << board[i][j];
```

OpenCppCoverage 工具会将每个 cpp 文件中哪句代码被覆盖了，哪句代码没被覆盖进行详细的展示，非常清晰。需要注意的是，该工具进行一次测试时，仅能进行一次命令运行，所以我们需要将测试样例的内容写入到 main 函数之中，然后根据新的测试代码，重新生成可执行文件再使用 OpenCppCoverage 工具代码覆盖率测试。

至此，我们就能够成功运行 OpenCppCoverage 工具，来进行代码代码覆盖率测试并且获得详细的代码覆盖率测试报告了，通过代码覆盖率测试报告，我们来检查程序的稳健性。

我们得到的代码覆盖率测试的摘要如下图所示：



从代码覆盖率测试的摘要中我们能够看出对于整体测试样例的覆盖率而言，达到了不错的 79%，其中 sudoku.h、filedeal.h 两个头文件和 sudoku.cpp 源文件达到了 98%以上的代

码覆盖率，在我们的实现中我们对程序所有可能出现的合规与不合规的输入情况都进行了考虑，设计构造了总计 25 个测试样例来对所有可能出现的分支来进行覆盖测试。对于我们自己设计实现的程序流程与功能部分，达到了非常好的接近 100%的代码测试覆盖率。最后对于 cmdline.h 来说，cmdline 工具源自 Github 开源的命令行参数解析工具，该工具功能较为强大，很多功能在我们的程序中并不需要使用，我们使用该工具时使用了该工具的命令行参数合法性、范围判断以及语法解析检查功能，所以覆盖率仅有 59%。

总的来说，我们的测试样例仍旧覆盖了大部分功能代码，整体的覆盖率达到了 79%以上。