



南开大学
Nankai University

南 开 大 学

网 络 空 间 安 全 学 院

软件工程

结对编程作业

刘扬 2011839 袁铭泽 2012777

年级：2020 级

专业：信息安全专业

指导教师：徐思涵

2023 年 6 月 30 日

目录

一、 任务说明	1
(一) 总体目标	1
(二) 任务要求	1
二、 用户手册	2
(一) 引言	2
1. 编写目的	2
2. 项目背景	2
3. 术语和缩略词	2
(二) 软件概述	2
1. 软件目标	2
2. 软件功能	3
3. 软件性能	3
(三) 运行环境	3
(四) 运行说明	3
1. 存储形式	3
2. 存取方式	3
3. 项目结构	4
4. 运行方式	4
5. 操作命令	5
(五) 用户操作举例	5
三、 质量分析	12
(一) 实验环境和工具	12
(二) 分析步骤	12
1. vs 自带代码分析工具	12
2. clang-tidy 代码分析工具	14
四、 覆盖率测试分析	18
(一) 测试用例表	18
(二) 使用工具进行测试	18
1. 工具说明: OpenCPPCoverage	18
2. 覆盖率测试过程	18
3. 修改后覆盖率再测试	20
(三) 覆盖率说明	20
五、 源码链接	22

一、 任务说明

(一) 总体目标

实现一个能够生成数独游戏并求解数独问题的控制台程序，具体包括：

1. 生成不重复的数独终局至文件
2. 读取文件内的数独文件，求解并将结果输出至文件

(二) 任务要求

1. 采用 c++ 语言实现
2. 可以使用 .Net Framework
3. 运行环境为 64-bit Windows 10
4. 代码经过质量分析并且消除警告
5. 写出至少 10 个测试用例来覆盖主要功能，统计覆盖率
6. 使用 GitHub 来管理源代码和测试用例，根据正常进度及时提交 commit
7. 生成数独：shudu.exe -c 100
8. 求解数独：shudu.exe -s path_of_file
9. 空格用 \$ 表示

参数名字	参数意义	范围限制	示例
-c	需要的数独终盘数量	1-1,000,000	'shudu.exe -c 100' 生成 100 个数独终盘
-s	读取数独游戏文件的路径并给出解	绝对路径或相对路径	'shudu.exe -s game.txt' 从 game.txt 读取文件，并给出解，把解保存到 shudu.txt 中
-n	需要的游戏数量	1-10,000	'shudu.exe -n 20' 生成 20 个
-m	生成游戏的难度	1-3	'shudu.exe -n 1000 -m 1' 表示生成最简单的 1000 个游戏，m 必须与 n 同时使用，否则报错
-r	生成游戏中挖空的数量范围	20-55	'shudu.exe -n 20 -r 2055' 表示生成 20 个挖空数在 20-55 之间的数独游戏，只有 r 和 n 一起使用才可以，否则报错
-u	生成游戏的解唯一		'shudu.exe -n 20 -u' 表示生成 20 个解唯一的数独游戏，-u 和 -n 一起使用时才不报错

表 1: 参数列表

二、 用户手册

(一) 引言

本用户手册旨在指导用户使用数独游戏生成和求解软件。本软件是一个基于 C++ 开发的控制台程序，旨在提供一种方便、灵活和高效的方式来生成不重复的数独终局、求解数独游戏，并提供其他相关功能。本节将介绍编写目的、项目背景、术语和缩略词的概述。

1. 编写目的

本用户手册的编写目的是为用户提供关于数独软件的详细说明和操作指南，以帮助用户正确安装、配置和使用该软件。通过阅读本手册，用户将了解软件的功能、使用方法和参数选项，从而能够方便地使用本数独软件。

2. 项目背景

数独游戏是一种受欢迎的逻辑推理游戏，玩家需要在九宫格中填写数字，使每一行、每一列和每一个小九宫格内的数字都不重复。为了方便玩家进行数独游戏，我们开发了数独软件，用于生成具有不同难度级别的数独终局，并提供求解数独游戏的功能。

3. 术语和缩略词

为了更好地理解本用户手册和数独软件的功能，以下是一些常用术语和缩略词的定义：

- 数独游戏：九宫格中的逻辑推理游戏，玩家需要填写数字以满足一定规则。
- 数独终局：完整的数独游戏，所有的空格都已经填满，符合数独规则。
- 求解：通过逻辑推理方法填写数独游戏的空格，使其符合数独规则。
- 参数：用于配置数独软件行为的选项和数值。
- 命令行：通过键入命令和参数来与计算机交互的文本界面。
- 绝对路径：文件或目录在文件系统中的完整路径，包括驱动器或根目录。
- 相对路径：文件或目录相对于当前工作目录的路径，不包括驱动器或根目录。

以上术语和缩略词将在本用户手册中频繁使用，确保您理解这些术语和缩略词的含义有助于更好地理解数独软件的功能和操作。

(二) 软件概述

1. 软件目标

数独软件旨在为用户提供生成和求解数独游戏的功能。其主要目标是使用户能够方便地生成具有不同难度级别的数独终局，并通过求解算法自动解决数独游戏。通过使用该软件，用户可以享受数独游戏的乐趣，提高逻辑推理能力，并挑战不同难度的数独问题。

2. 软件功能

数独软件具有以下主要功能：

- 生成数独终局：根据用户指定的数量和难度级别，生成不重复的数独终局，可以灵活地控制生成的数独游戏的难度。
- 求解数独游戏：用户可以通过指定数独游戏文件的路径，由数独软件自动求解数独游戏，并将解保存到指定的文件中。
- 参数配置：用户可以通过命令行参数来配置生成和求解数独游戏的行为，包括生成数量、难度级别、挖空数量范围等选项。
- 解唯一检测：用户可以选择生成解唯一的数独游戏，确保每个生成的游戏都只有唯一解。

3. 软件性能

数独软件在性能方面具有以下特点：

- 高效的生成和求解算法：数独软件使用高效的算法来生成不重复的数独终局和求解数独游戏，以提高处理速度和准确性。
- 可扩展性：数独软件可以处理大量的数独游戏生成和求解任务，支持生成数独终局和求解数独游戏的批量操作。
- 灵活参数配置：用户可以通过配置参数来调整生成和求解数独游戏的行为，以满足不同需求和难度要求。
- 准确性和稳定性：数独软件经过质量分析和测试，确保生成的数独终局和求解的结果准确无误，并具有良好的稳定性。

(三) 运行环境

为了正常运行数独软件，您的系统需要满足以下要求：

- 操作系统：64-bit Windows 10
- 处理器：Intel Core i3 或以上
- 内存：4GB 或以上
- 磁盘空间：100MB 或以上

(四) 运行说明

1. 存储形式

本软件是一个由 C++ 语言编写的数独游戏的生成和求解的控制台程序，以可执行文件 `sudoku.exe` 的形式存储，大小约为 200KB。

2. 存取方式

用户可以从 github 上获取本项目的全部文件。可以直接获取已经编译好的 `sudoku.exe` 可执行文件，也可以下载源代码文件，并使用 C++ 编译器生成可执行文件。

github 链接：<https://github.com/runxii/SE-sudoku>

3. 项目结构

github 上的项目结构如下，用户可以根据自己的需要获取相应的文件：

```

1  --README.md
2  --img          // README中的图像文件目录
3  --example
4  --game.txt     // 用于对-s进行测试的数独游戏文件
5  --incomplete.txt // 用于对-s进行测试的不完整的数独游戏文件
6  --solveSheet.txt // 使用-n生成游戏时对应的终局文件，每次使用-n前都要删除
7  --generate.txt  // 使用-n生成的数独游戏文件，每次使用-n生成游戏前都要删除
                        该文件
8  --sudoku.txt   // 使用-s生成的解决数独得到的答案，每次使用-s前都要删除
9  --src          // 源代码目录
10 --main.cpp     // 入口文件
11 --generate.h   // 生成数独游戏文件
12 --solver.h     // 解决数独问题文件
13 --test         // 覆盖率测试用例目录

```

4. 运行方式

用户可以使用命令行窗口来运行数独软件，双击打开 sudoku.exe 可执行文件运行，在命令行窗口输入 sudoku.exe 加上相应的参数来实现对应的功能。或者进入 sudoku.exe 对应的文件夹路径下，打开 cmd 窗口，输入 sudoku.exe 加上相应的参数来运行本软件。

注意，在每次运行前应删除 solveSheet.txt, gernerate.txt, sudoku.txt, 在每次运行时重新生成这三个文件。如果运行前不删除这三个文件，本次运行的结果会跟随在之前运行的结果之后，可能会造成某些错误。而 game.txt 是固定要解密的数独游戏，不必在运行前删除。

此外，如果更改了数独软件的位置，应确保相应地修改对应文件的位置。如，sudoku.exe 可执行文件应与用于测试-s 参数的 game.txt 文件处于相同路径下，才能保证运行-s 参数时，可以获取到 game.txt 的内容并进行求解。

当然，如果您想读取一个自己给定的文件内的数独问题，而不是我们给出的 game.txt 中的数独问题，可以在可执行文件 sudoku.exe 的同级文件夹下准备一个包含您想求解的数独问题的文本文件，并将其命名为 game.txt。数独问题应以 9*9 矩阵的形式呈现，每个数字之间用空格隔开，空格用 \$ 表示。每个数独矩阵之间用 “——i——” 隔开，其中 i 为第几个数独游戏。

一个可能的示例如下：

```

-----1-----
$ 2 $ 6 $ $ 1 $ 3
9 1 7 $ $ $ 6 $ 2
3 $ 6 5 2 $ 9 $ 7
$ 6 $ $ 3 8 $ $ $
$ $ $ $ $ 6 4 3 $
$ 3 1 $ $ 2 5 6 $
$ 8 $ $ $ 7 3 $ $
2 $ $ 8 1 4 $ 9 6
1 7 9 3 $ 5 8 $ $

```

图 1: 示例数独形式

5. 操作命令

操作命令的格式为 `sudoku.exe [参数选项] [参数值]`。参数选项以“-”开头，如-c, -s, 表示对应不同的功能。多个选项中间以空格进行分割。参数值是跟在选项后的数字或字符串，提供给参数选项具体的值。

所有的参数选项：

- -c: 生成给定数量的数独终局；
- -s: 读取数独游戏文件并给出解答；
- -n: 生成给定数量的数独游戏；
- -m: 生成指定游戏难度的数独游戏；
- -r: 生成指定挖空数量范围的数独游戏；
- -u: 生成解唯一的数独游戏。

对应的参数值：

- -c: 需要一个参数，表示需要生成的数独终局数量，范围是 1-1,000,000；
- -s: 需要一个参数，表示需要解的数独棋盘文件路径，默认为 `sudoku.exe` 的同级文件夹下的路径；
- -n: 需要一个参数，表示需要的游戏数量，范围是 1-10,000；
- -m: 需要一个参数，表示难度等级，范围是 1-3；
- -r: 需要一个参数，表示挖空数量范围，格式为下限上限，范围是 20-55；
- -u: 不需要参数。

操作命令示例：

- `sudoku.exe -c 100` 表示生成 100 个数独终局，并在控制台屏幕上显示；
- `sudoku.exe -s game.txt` 表示从 `game.txt` 读取若个数独游戏，给出其解答；
- `sudoku.exe -n 20` 表示生成 20 个数独游戏；
- `sudoku.exe -n 1000 -m 1` 表示生成 1000 个最简单的数独游戏；
- `sudoku.exe -n 20 -r 20 55` 表示生成 20 个挖空数在 20 到 55 之间的数独游戏；
- `sudoku.exe -n 20 -u` 表示生成 20 个解唯一的数独游戏。

(五) 用户操作举例

(1) `sudoku.exe -c 20`

表示生成 20 个数独终局，并在控制台屏幕上显示。

```
D:\C++\sudoku\Debug>sudoku.exe -c 20
```

```
Sudoku board 1:
```

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

```
Sudoku board 2:
```

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

(a)

```
Sudoku board 19:
```

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

```
Sudoku board 20:
```

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

```
D:\C++\sudoku\Debug>
```

(b)

图 2: 生成 20 个数独终局

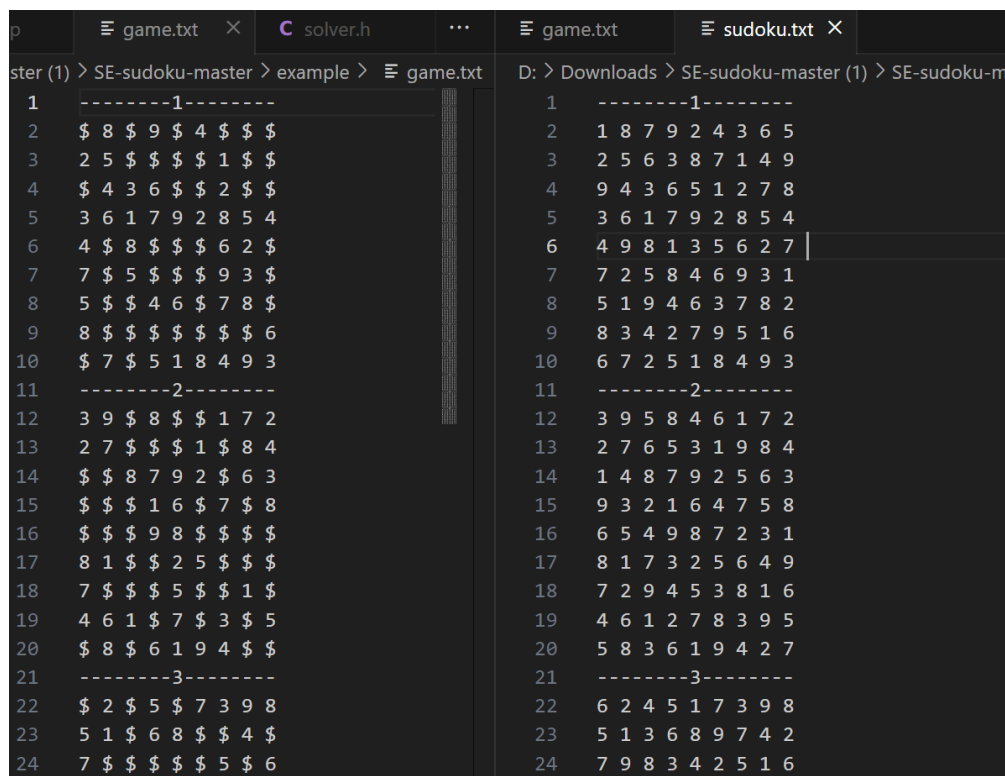
(2) sudoku.exe -s game.txt

表示从 game.txt 读取若个数独游戏，给出其解答，保存至 sudoku.txt 中。

```
D:\C++\sudoku\Debug>sudoku.exe -s game.txt
文件里一共有20个数独游戏待解决
解决已完成，答案保存到sudoku.txt中，请查看。

D:\C++\sudoku\Debug>_
```

(a)



game.txt	sudoku.txt
1 -----1-----	1 -----1-----
2 \$ 8 \$ 9 \$ 4 \$ \$ \$	2 1 8 7 9 2 4 3 6 5
3 2 5 \$ \$ \$ 1 \$ \$	3 2 5 6 3 8 7 1 4 9
4 \$ 4 3 6 \$ \$ 2 \$ \$	4 9 4 3 6 5 1 2 7 8
5 3 6 1 7 9 2 8 5 4	5 3 6 1 7 9 2 8 5 4
6 4 \$ 8 \$ \$ \$ 6 2 \$	6 4 9 8 1 3 5 6 2 7
7 7 \$ 5 \$ \$ \$ 9 3 \$	7 7 2 5 8 4 6 9 3 1
8 5 \$ \$ 4 6 \$ 7 8 \$	8 5 1 9 4 6 3 7 8 2
9 8 \$ \$ \$ \$ \$ \$ 6	9 8 3 4 2 7 9 5 1 6
10 \$ 7 \$ 5 1 8 4 9 3	10 6 7 2 5 1 8 4 9 3
11 -----2-----	11 -----2-----
12 3 9 \$ 8 \$ \$ 1 7 2	12 3 9 5 8 4 6 1 7 2
13 2 7 \$ \$ \$ 1 \$ 8 4	13 2 7 6 5 3 1 9 8 4
14 \$ \$ 8 7 9 2 \$ 6 3	14 1 4 8 7 9 2 5 6 3
15 \$ \$ \$ 1 6 \$ 7 \$ 8	15 9 3 2 1 6 4 7 5 8
16 \$ \$ \$ 9 8 \$ \$ \$ \$	16 6 5 4 9 8 7 2 3 1
17 8 1 \$ \$ 2 5 \$ \$ \$	17 8 1 7 3 2 5 6 4 9
18 7 \$ \$ \$ 5 \$ \$ 1 \$	18 7 2 9 4 5 3 8 1 6
19 4 6 1 \$ 7 \$ 3 \$ 5	19 4 6 1 2 7 8 3 9 5
20 \$ 8 \$ 6 1 9 4 \$ \$	20 5 8 3 6 1 9 4 2 7
21 -----3-----	21 -----3-----
22 \$ 2 \$ 5 \$ 7 3 9 8	22 6 2 4 5 1 7 3 9 8
23 5 1 \$ 6 8 \$ \$ 4 \$	23 5 1 3 6 8 9 7 4 2
24 7 \$ \$ \$ \$ 5 \$ 6	24 7 9 8 3 4 2 5 1 6

(b)

图 3: 从 game.txt 中读取数独游戏并求解

上图分别为控制台的显示和 game.txt 与 sudoku.txt 中的内容对比。

可以看到，控制台显示了 game.txt 中待解决的数独游戏个数，并在求解之后显示解决完成，答案保存在 sudoku.txt 中，供用户查看。

而打开 game.txt 和 sudoku.txt 进行对比后可以看到,sudoku.txt 中已经给出了对应 game.txt 中数独游戏的正确的解。

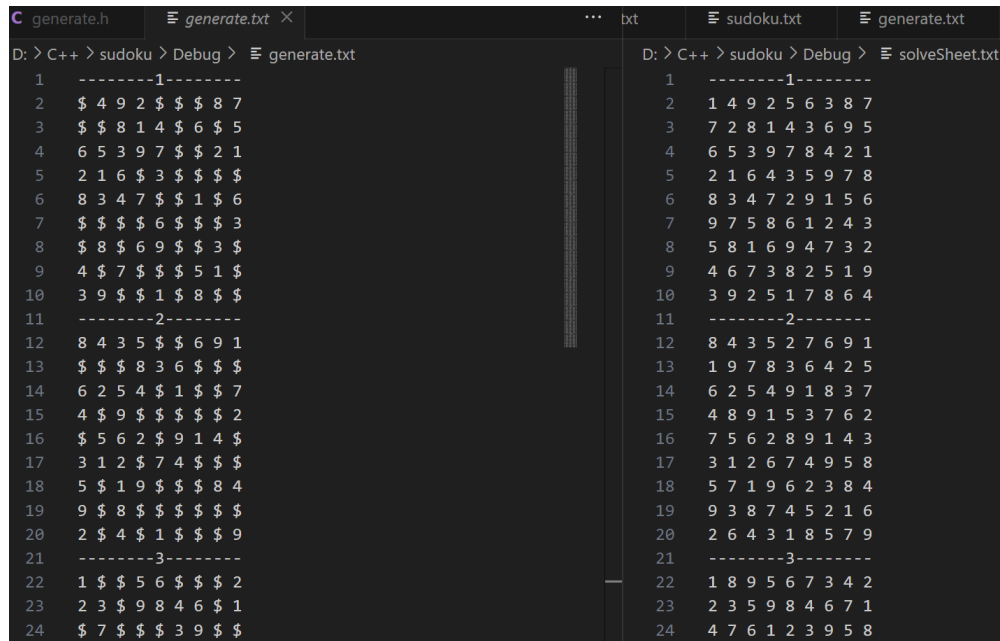
(3) sudoku.exe -n 20

表示生成 20 个数独游戏。

```
D:\C++\sudoku\Debug>sudoku.exe -n 20
20个默认数独游戏已成功生成，并保存generate.txt中
对应的终局已保存到solveSheet.txt中

D:\C++\sudoku\Debug>_
```

(a)



generate.txt	solveSheet.txt
1 -----1-----	1 -----1-----
2 \$ 4 9 2 \$ \$ \$ 8 7	2 1 4 9 2 5 6 3 8 7
3 \$ \$ 8 1 4 \$ 6 \$ 5	3 7 2 8 1 4 3 6 9 5
4 6 5 3 9 7 \$ \$ 2 1	4 6 5 3 9 7 8 4 2 1
5 2 1 6 \$ 3 \$ \$ \$ \$	5 2 1 6 4 3 5 9 7 8
6 8 3 4 7 \$ \$ 1 \$ 6	6 8 3 4 7 2 9 1 5 6
7 \$ \$ \$ \$ 6 \$ \$ \$ 3	7 9 7 5 8 6 1 2 4 3
8 \$ 8 \$ 6 9 \$ \$ 3 \$	8 5 8 1 6 9 4 7 3 2
9 4 \$ 7 \$ \$ \$ 5 1 \$	9 4 6 7 3 8 2 5 1 9
10 3 9 \$ \$ 1 \$ 8 \$ \$	10 3 9 2 5 1 7 8 6 4
11 -----2-----	11 -----2-----
12 8 4 3 5 \$ \$ 6 9 1	12 8 4 3 5 2 7 6 9 1
13 \$ \$ \$ 8 3 6 \$ \$ \$	13 1 9 7 8 3 6 4 2 5
14 6 2 5 4 \$ 1 \$ \$ 7	14 6 2 5 4 9 1 8 3 7
15 4 \$ 9 \$ \$ \$ \$ \$ 2	15 4 8 9 1 5 3 7 6 2
16 \$ 5 6 2 \$ 9 1 4 \$	16 7 5 6 2 8 9 1 4 3
17 3 1 2 \$ 7 4 \$ \$ \$	17 3 1 2 6 7 4 9 5 8
18 5 \$ 1 9 \$ \$ \$ 8 4	18 5 7 1 9 6 2 3 8 4
19 9 \$ 8 \$ \$ \$ \$ \$ \$	19 9 3 8 7 4 5 2 1 6
20 2 \$ 4 \$ 1 \$ \$ \$ 9	20 2 6 4 3 1 8 5 7 9
21 -----3-----	21 -----3-----
22 1 \$ \$ 5 6 \$ \$ \$ 2	22 1 8 9 5 6 7 3 4 2
23 2 3 \$ 9 8 4 6 \$ 1	23 2 3 5 9 8 4 6 7 1
24 \$ 7 \$ \$ \$ 3 9 \$ \$	24 4 7 6 1 2 3 9 5 8

(b)

图 4: 生成 20 个数独游戏

上图分别为控制台的显示和 generate.txt 与 solveSheet.txt 中的内容对比。

可以看到，控制台窗口显示，20 个默认数独游戏已成功生成，并保存在了 generate.txt 文件中，而对应数独游戏的终局保存在了 solveSheet.txt 中。

而打开 generate.txt 和 solveSheet.txt 进行对比后可以看到，solveSheet.txt 中已经给出了对应 generate.txt 中数独游戏的正确的解。

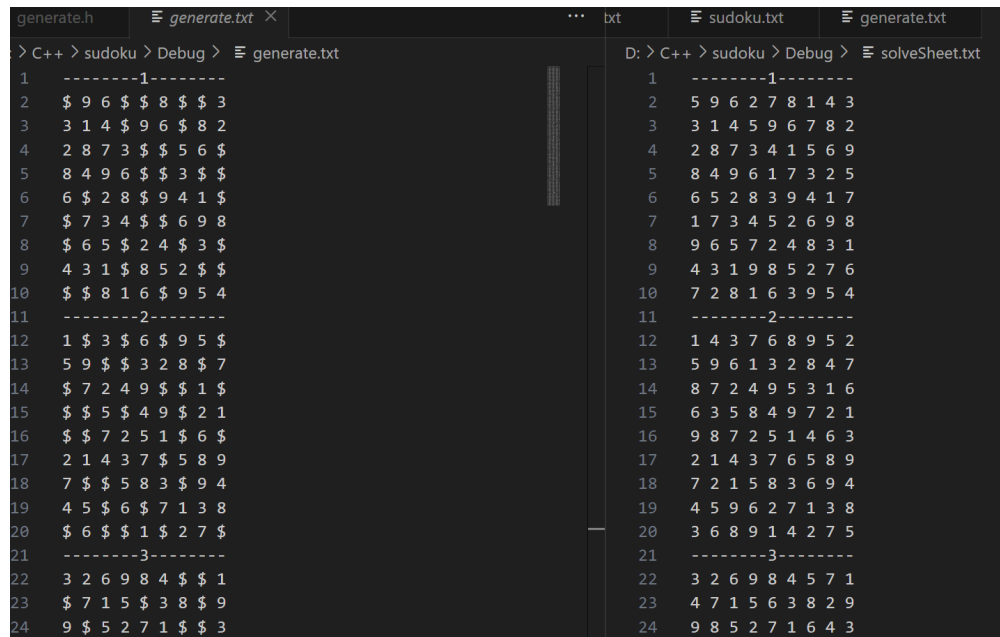
(4) `sudoku.exe -n 10 -m 1`

表示生成 10 个最简单的数独游戏。

```
D:\C++\sudoku\Debug>sudoku.exe -n 10 -m 1
10个难度为1的数独游戏已成功生成，并保存到generate.txt
对应终局保存到solveSheet.txt中

D:\C++\sudoku\Debug>
```

(a)



```
generate.h  generate.txt  ...  txt  sudoku.txt  generate.txt
D: > C++ > sudoku > Debug > generate.txt
1  -----1-----
2  $ 9 6 $ $ 8 $ $ 3
3  3 1 4 $ 9 6 $ 8 2
4  2 8 7 3 $ $ 5 6 $
5  8 4 9 6 $ $ 3 $ $
6  6 $ 2 8 $ 9 4 1 $
7  $ 7 3 4 $ $ 6 9 8
8  $ 6 5 $ 2 4 $ 3 $
9  4 3 1 $ 8 5 2 $ $
10 $ $ 8 1 6 $ 9 5 4
11 -----2-----
12 1 $ 3 $ 6 $ 9 5 $
13 5 9 $ $ 3 2 8 $ 7
14 $ 7 2 4 9 $ $ 1 $
15 $ $ 5 $ 4 9 $ 2 1
16 $ $ 7 2 5 1 $ 6 $
17 2 1 4 3 7 $ 5 8 9
18 7 $ $ 5 8 3 $ 9 4
19 4 5 $ 6 $ 7 1 3 8
20 $ 6 $ $ 1 $ 2 7 $
21 -----3-----
22 3 2 6 9 8 4 $ $ 1
23 $ 7 1 5 $ 3 8 $ 9
24 9 $ 5 2 7 1 $ $ 3

D: > C++ > sudoku > Debug > solveSheet.txt
1  -----1-----
2  5 9 6 2 7 8 1 4 3
3  3 1 4 5 9 6 7 8 2
4  2 8 7 3 4 1 5 6 9
5  8 4 9 6 1 7 3 2 5
6  6 5 2 8 3 9 4 1 7
7  1 7 3 4 5 2 6 9 8
8  9 6 5 7 2 4 8 3 1
9  4 3 1 9 8 5 2 7 6
10 7 2 8 1 6 3 9 5 4
11 -----2-----
12 1 4 3 7 6 8 9 5 2
13 5 9 6 1 3 2 8 4 7
14 8 7 2 4 9 5 3 1 6
15 6 3 5 8 4 9 7 2 1
16 9 8 7 2 5 1 4 6 3
17 2 1 4 3 7 6 5 8 9
18 7 2 1 5 8 3 6 9 4
19 4 5 9 6 2 7 1 3 8
20 3 6 8 9 1 4 2 7 5
21 -----3-----
22 3 2 6 9 8 4 5 7 1
23 4 7 1 5 6 3 8 2 9
24 9 8 5 2 7 1 6 4 3
```

(b)

图 5: 生成 10 个最简单的数独游戏

上图分别为控制台的显示和 `generate.txt` 与 `solveSheet.txt` 中的内容对比。

可以看到，控制台窗口显示，10 个难度为 1 的数独游戏已成功生成，并保存到 `generate.txt` 中，而对应数独游戏的终局保存在了 `solveSheet.txt` 中。

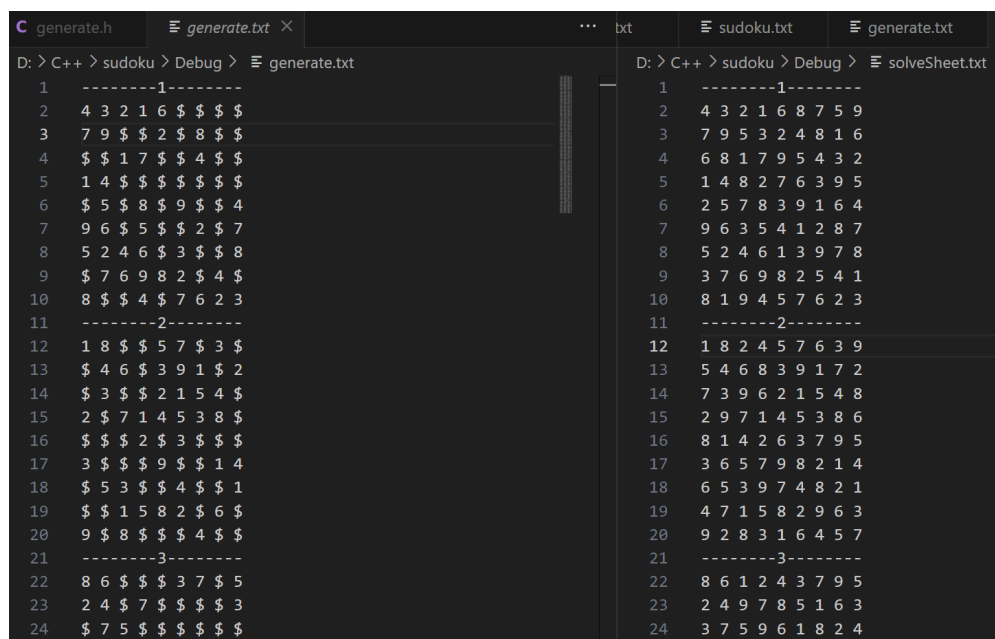
而打开 `generate.txt` 和 `solveSheet.txt` 进行对比后可以看到，`solveSheet.txt` 中已经给出了对应 `generate.txt` 中数独游戏的正确的解。

(5) `sudoku.exe -n 10 -r 20 55`

表示生成 10 个挖空数在 20-55 之间的数独游戏。

```
D:\C++\sudoku\Debug>sudoku.exe -n 10 -r 20 55
10个指定挖空的数独游戏已成功生成，并保存generate.txt中
对应的终局已保存到solveSheet.txt中
D:\C++\sudoku\Debug>
```

(a)



generate.txt	solveSheet.txt
1 -----1-----	1 -----1-----
2 4 3 2 1 6 \$ \$ \$ \$	2 4 3 2 1 6 8 7 5 9
3 7 9 \$ \$ 2 \$ 8 \$ \$	3 7 9 5 3 2 4 8 1 6
4 \$ \$ 1 7 \$ \$ 4 \$ \$	4 6 8 1 7 9 5 4 3 2
5 1 4 \$ \$ \$ \$ \$ \$	5 1 4 8 2 7 6 3 9 5
6 \$ 5 \$ 8 \$ 9 \$ \$ 4	6 2 5 7 8 3 9 1 6 4
7 9 6 \$ 5 \$ \$ 2 \$ 7	7 9 6 3 5 4 1 2 8 7
8 5 2 4 6 \$ 3 \$ \$ 8	8 5 2 4 6 1 3 9 7 8
9 \$ 7 6 9 8 2 \$ 4 \$	9 3 7 6 9 8 2 5 4 1
10 8 \$ \$ 4 \$ 7 6 2 3	10 8 1 9 4 5 7 6 2 3
11 -----2-----	11 -----2-----
12 1 8 \$ \$ 5 7 \$ 3 \$	12 1 8 2 4 5 7 6 3 9
13 \$ 4 6 \$ 3 9 1 \$ 2	13 5 4 6 8 3 9 1 7 2
14 \$ 3 \$ \$ 2 1 5 4 \$	14 7 3 9 6 2 1 5 4 8
15 2 \$ 7 1 4 5 3 8 \$	15 2 9 7 1 4 5 3 8 6
16 \$ \$ \$ 2 \$ 3 \$ \$ \$	16 8 1 4 2 6 3 7 9 5
17 3 \$ \$ \$ 9 \$ \$ 1 4	17 3 6 5 7 9 8 2 1 4
18 \$ 5 3 \$ \$ 4 \$ \$ 1	18 6 5 3 9 7 4 8 2 1
19 \$ \$ 1 5 8 2 \$ 6 \$	19 4 7 1 5 8 2 9 6 3
20 9 \$ 8 \$ \$ \$ 4 \$ \$	20 9 2 8 3 1 6 4 5 7
21 -----3-----	21 -----3-----
22 8 6 \$ \$ \$ 3 7 \$ 5	22 8 6 1 2 4 3 7 9 5
23 2 4 \$ 7 \$ \$ \$ \$ 3	23 2 4 9 7 8 5 1 6 3
24 \$ 7 5 \$ \$ \$ \$ \$ \$	24 3 7 5 9 6 1 8 2 4

(b)

图 6: 生成 10 个挖空数在 20-55 之间的数独游戏

上图分别为控制台的显示和 `generate.txt` 与 `solveSheet.txt` 中的内容对比。

可以看到，控制台窗口显示，10 个指定挖空的数独游戏已成功生成，并保存 `generate.txt` 中，而对应数独游戏的终局保存在了 `solveSheet.txt` 中。

而打开 `generate.txt` 和 `solveSheet.txt` 进行对比后可以看到，`solveSheet.txt` 中已经给出了对应 `generate.txt` 中数独游戏的正确的解。

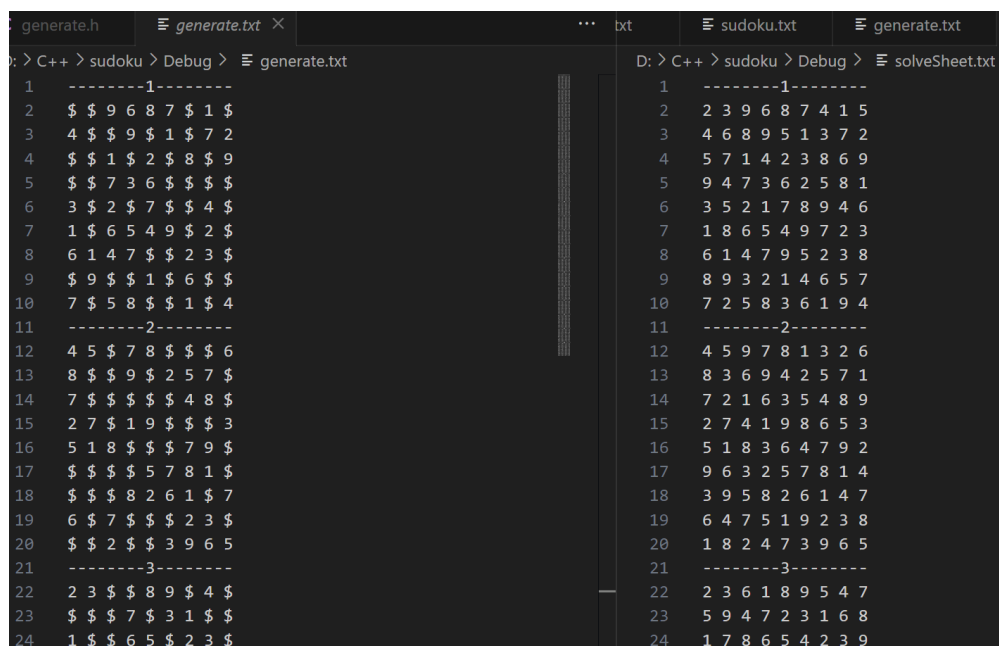
(6) sudoku.exe -n 20 -u

表示生成 20 个解唯一的数独游戏。

```
D:\C++\sudoku\Debug>sudoku.exe -n 20 -u
20个有唯一解的数独游戏已成功生成，并保存generate.txt中
对应的终局已保存到solveSheet.txt中

D:\C++\sudoku\Debug>
```

(a)



generate.txt	solveSheet.txt
1 -----1-----	1 -----1-----
2 \$ \$ 9 6 8 7 \$ 1 \$	2 2 3 9 6 8 7 4 1 5
3 4 \$ \$ 9 \$ 1 \$ 7 2	3 4 6 8 9 5 1 3 7 2
4 \$ \$ 1 \$ 2 \$ 8 \$ 9	4 5 7 1 4 2 3 8 6 9
5 \$ \$ 7 3 6 \$ \$ \$ \$	5 9 4 7 3 6 2 5 8 1
6 3 \$ 2 \$ 7 \$ \$ 4 \$	6 3 5 2 1 7 8 9 4 6
7 1 \$ 6 5 4 9 \$ 2 \$	7 1 8 6 5 4 9 7 2 3
8 6 1 4 7 \$ \$ 2 3 \$	8 6 1 4 7 9 5 2 3 8
9 \$ 9 \$ \$ 1 \$ 6 \$ \$	9 8 9 3 2 1 4 6 5 7
10 7 \$ 5 8 \$ \$ 1 \$ 4	10 7 2 5 8 3 6 1 9 4
11 -----2-----	11 -----2-----
12 4 5 \$ 7 8 \$ \$ \$ 6	12 4 5 9 7 8 1 3 2 6
13 8 \$ \$ 9 \$ 2 5 7 \$	13 8 3 6 9 4 2 5 7 1
14 7 \$ \$ \$ \$ \$ 4 8 \$	14 7 2 1 6 3 5 4 8 9
15 2 7 \$ 1 9 \$ \$ 3	15 2 7 4 1 9 8 6 5 3
16 5 1 8 \$ \$ \$ 7 9 \$	16 5 1 8 3 6 4 7 9 2
17 \$ \$ \$ \$ 5 7 8 1 \$	17 9 6 3 2 5 7 8 1 4
18 \$ \$ \$ 8 2 6 1 \$ 7	18 3 9 5 8 2 6 1 4 7
19 6 \$ 7 \$ \$ \$ 2 3 \$	19 6 4 7 5 1 9 2 3 8
20 \$ \$ 2 \$ \$ 3 9 6 5	20 1 8 2 4 7 3 9 6 5
21 -----3-----	21 -----3-----
22 2 3 \$ \$ 8 9 \$ 4 \$	22 2 3 6 1 8 9 5 4 7
23 \$ \$ \$ 7 \$ 3 1 \$ \$	23 5 9 4 7 2 3 1 6 8
24 1 \$ \$ 6 5 \$ 2 3 \$	24 1 7 8 6 5 4 2 3 9

(b)

图 7: 生成 10 个挖空数在 20-55 之间的数独游戏

上图分别为控制台的显示和 generate.txt 与 solveSheet.txt 中的内容对比。

可以看到，控制台窗口显示，20 个有唯一解的数独游戏已成功生成，并保存 generate.txt 中，而对应数独游戏的终局保存在了 solveSheet.txt 中。

而打开 generate.txt 和 solveSheet.txt 进行对比后可以看到，solveSheet.txt 中已经给出了对应 generate.txt 中数独游戏的正确的解。

三、 质量分析

(一) 实验环境和工具

本软件基于 Visual Studio 进行开发，使用的 Visual Studio 版本为 Microsoft Visual Studio 2019。在质量分析过程中，我们使用 Visual Studio 自带的代码分析工具以及 clang-tidy 工具分别进行质量分析，并修改代码，从而满足消除所有警告的要求。

在 Visual Studio 中，如要开启 clang-tidy 代码检查，需要下载 clang 支持。点击 vs 菜单栏-> 工具-> 获取工具和功能，在打开的 Visual Studio Installer 窗口中选择“使用 C++ 的桌面开发”，在右侧工具集中找到“适用于 Windows 的 C++ Clang 工具”并勾选，点击修改按钮确认。接下来等待下载和安装完成。安装完成以后，打开项目，右键项目选择“属性”，在 Code Analysis 中找到 Clang-Tidy 并启用。

(二) 分析步骤

1. vs 自带代码分析工具

在 Visual Studio 中点击“生成”按钮，再点击“对 sudoku 进行代码分析”，在错误列表和输出窗口查看代码分析结果，如下所示：

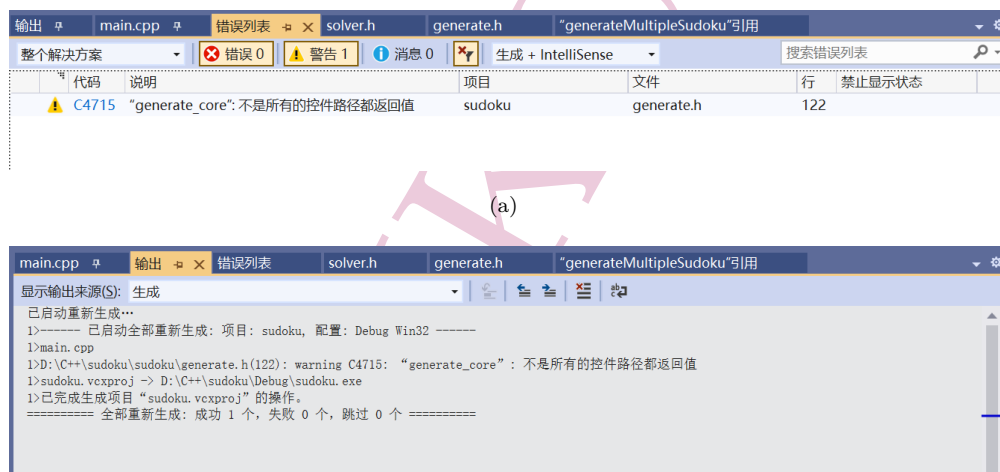


图 8: 代码分析结果 1

可以看到提示有一个警告，为 warning C4715: “generate_core”: 不是所有的控件路径都返回值。

查看 generate_core 源代码如下：

```
1 bool generate_core(vector<vector<int>>& num, int row, int col)
2 {
3
4     vector<int> number;
5     for (int i = 1; i <= 9; i++)
6         number.emplace_back(i);
7     while (!number.empty())
8     {
```

```

9      int randindex = rand() % number.size();
10     int randnum = number[randindex];
11     number.erase(number.begin() + randindex);
12     num[row][col] = randnum;
13     if (IsRightPlace(num, row, col) == false)
14         continue;
15     if (row == SIZE - 1 && col == SIZE - 1)
16     {
17         return true;
18     }
19     int nextrow, nextcol;
20     if (col == SIZE - 1)
21     {
22         nextrow = row + 1;
23         nextcol = 0;
24     }
25     else
26     {
27         nextrow = row;
28         nextcol = col + 1;
29     }
30     bool next = generate_core(num, nextrow, nextcol);
31     if (next)
32         return true;
33 }
34 if (number.empty())
35 {
36     num[row][col] = -5;
37     return false;
38 }
39 }

```

经过分析, 问题出在代码最后, if (number.empty()) 为真时, 返回 false, 但 if 条件语句判断为假时, 没有返回值, 因此会出现不是所有的控件路径都返回值的警告。事实上, 能够跳出 while 循环, 并执行 if 判断语句, 只有 number.empty() 的情况, 因此, 不用再判断 number.empty() 是否为真。将代码更改为如下:

```

1      bool generate_core(vector<vector<int>>& num, int row, int col)
2      {
3
4          vector<int> number;
5          for (int i = 1; i <= 9; i++)
6              number.emplace_back(i);
7          while (!number.empty())
8          {
9              int randindex = rand() % number.size();
10             int randnum = number[randindex];
11             number.erase(number.begin() + randindex);
12             num[row][col] = randnum;

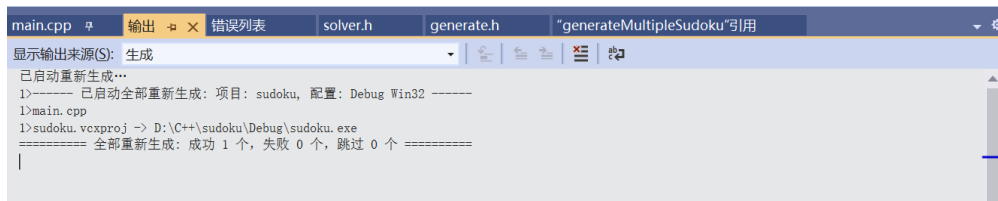
```

```
13     if (IsRightPlace(num, row, col) == false)
14         continue;
15     if (row == SIZE - 1 && col == SIZE - 1)
16     {
17         return true;
18     }
19     int nextrow, nextcol;
20     if (col == SIZE - 1)
21     {
22         nextrow = row + 1;
23         nextcol = 0;
24     }
25     else
26     {
27         nextrow = row;
28         nextcol = col + 1;
29     }
30     bool next = generate_core(num, nextrow, nextcol);
31     if (next)
32         return true;
33 }
34 num[row][col] = -5;
35 return false;
36 }
```

重新运行代码分析，查看结果：



(a)



(b)

图 9: 最终结果 1

可以看到，在 Visual Studio 自带的代码分析工具中已消除了所有警告。

2. clang-tidy 代码分析工具

运行 clang-tidy 进行代码分析，结果如下：


```

PS A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku> cl /analyze main.cpp
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.29.30141 版
版权所有 (C) Microsoft Corporation. 保留所有权利。

main.cpp
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\ostream(284): warning C4530: 使用了 C++ 异常处理程序, 但未启用展开语义。请指定 /EHsc
C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\ostream(269): note: 在编译 类 模板 成员函数“std::basic_ostream<char,std::char_traits<char>> &std::basic_ostream<char,std::char_traits<char>>::operator <<(int)”时
A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(31): note: 查看对正在编译的函数模板实例化“std::basic_ostream<char,std::char_traits<char>> &std::basic_ostream<char,std::char_traits<char>>::operator <<(int)”的引用
A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(28): note: 查看对正在编译的 类 模板 实例化“std::basic_ostream<char,std::char_traits<char>>”的引用
A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp(28) : warning C6246: “eraseCount”的局部声明遮蔽了外部作用域中具有相同名称的声明。有关其他信息, 请参见此前位于“19”行(“a:\_uni\junior\second term\software-engineering\se-labs\se-lab5-6.30\sudoku\sudoku\main.cpp”中)的声明。: Lines: 19
A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp(39) : warning C6246: “eraseCount”的局部声明遮蔽了外部作用域中具有相同名称的声明。有关其他信息, 请参见此前位于“19”行(“a:\_uni\junior\second term\software-engineering\se-labs\se-lab5-6.30\sudoku\sudoku\main.cpp”中)的声明。: Lines: 19
A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(123) : warning C4715: “generate_core”: 不是所有的控件路径都返回值
Microsoft (R) Incremental Linker Version 14.29.30141.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:main.exe
main.obj

```

图 10: 代码分析结果 2

图片中的具体的分析结果如下:

- 1 C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\ostream(284): warning C4530: 使用了 C++ 异常处理程序, 但未启用展开语义。请指定 /EHsc
- 2 C:\Program Files (x86)\Microsoft Visual Studio\2019\Community\VC\Tools\MSVC\14.29.30133\include\ostream(269): note: 在编译 类 模板 成员函数“std::basic_ostream<char,std::char_traits<char>> &std::basic_ostream<char,std::char_traits<char>>::operator <<(int)”时
- 3 A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(31): note: 查看对正在编译的函数模板实例化“std::basic_ostream<char,std::char_traits<char>> &std::basic_ostream<char,std::char_traits<char>>::operator <<(int)”的引用
- 4 A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(28): note: 查看对正在编译的类模板实例化“std::basic_ostream<char,std::char_traits<char>>”的引用
- 5 A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp(28) : warning C6246: “eraseCount”的局部声明遮蔽了外部作用域中具有相同名称的声明。有关其他信息, 请参见此前位于“19”行(“a:_uni\junior\second term\software-engineering\se-labs\se-lab5-6.30\sudoku\sudoku\main.cpp”中)的声明。: Lines: 19
- 6 A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp(39) : warning C6246: “eraseCount”的局部声明遮蔽了外部作用域中具有相同名称的声明。有关其他信息, 请参见此前位于“19”行(“a:_uni\junior\second term\software-engineering\se-labs\se-lab5-6.30\sudoku\sudoku\main.cpp”中)的声明。: Lines: 19
- 7 A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate.h(123) : warning C4715: “generate_core”: 不是所有的控件路径都返回值

可以看到, 最后一个警告如 vs 自带的代码分析器所提示的警告一样, 这个警告已经消除掉, 我们只需关注其余警告。clang-tidy 里提示, main.cpp 中的第 28 和 39 行关于“eraseCount”的局部声明遮蔽了外部作用域中具有相同名称的声明。

查看源代码如下:

```

1  int eraseCount = ERASE;
2  if (argc < 4) {
3      //generateMultipleSudoku(sudokuCount, eraseCount);
4      cout << sudokuCount << "个默认数独游戏已成功生成, 并保存generate.
      txt中\n对应的终局已保存到solveSheet.txt中" << endl;
5  }
6  else {
7      string next_option = argv[3];
8      if (next_option == "-u") {
9          // 需要去除的数字数量, 范围在36到44之间 (保证有解且唯一解)
10         int eraseCount = 0;
11         eraseCount = rand() % 45 + 36;
12         cout << sudokuCount << "个有唯一解的数独游戏已成功生成, 并保
            存generate.txt中\n对应的终局已保存到solveSheet.txt中" <<
            endl;
13     }
14     else {
15         string next_argument = argv[4];
16         if (next_option == "-r")
17         {
18             // 按照挖空格子的数量范围生成数独游戏
19             int minErase = stoi(next_argument);
20             int maxErase = stoi(argv[5]);
21             int eraseCount = rand() % (maxErase - minErase + 1) +
                minErase;
22             cout << sudokuCount << "个指定挖空的数独游戏已成功生成,
                并保存generate.txt中\n对应的终局已保存到solveSheet.
                txt中" << endl;
23         }
24         .....

```

可以看到, 代码中重复声明了三次 eraseCount, 因此, 将后两次的 int 删掉, 直接引用变量而不声明。修改后的代码如下:

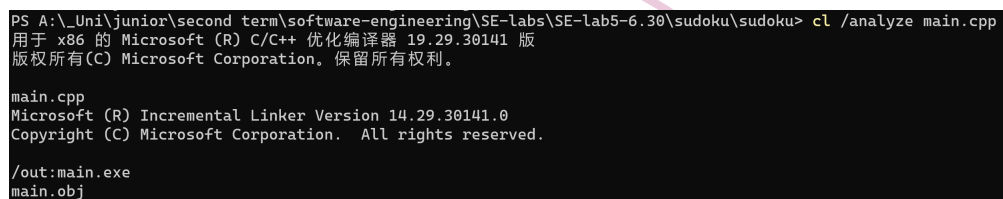
```

1  int eraseCount = ERASE;
2  if (argc < 4) {
3      //generateMultipleSudoku(sudokuCount, eraseCount);
4      cout << sudokuCount << "个默认数独游戏已成功生成, 并保存generate.
      txt中\n对应的终局已保存到solveSheet.txt中" << endl;
5  }
6  else {
7      string next_option = argv[3];
8      if (next_option == "-u") {
9          // 需要去除的数字数量, 范围在36到44之间 (保证有解且唯一解)
10         eraseCount = 0;
11         eraseCount = rand() % 45 + 36;
12         cout << sudokuCount << "个有唯一解的数独游戏已成功生成, 并保
            存generate.txt中\n对应的终局已保存到solveSheet.txt中" <<

```

```
endl;
13     }
14     else {
15         string next_argument = argv[4];
16         if (next_option == "-r")
17         {
18             // 按照挖空格子的数量范围生成数独游戏
19             int minErase = stoi(next_argument);
20             int maxErase = stoi(argv[5]);
21             eraseCount = rand() % (maxErase - minErase + 1) +
                minErase;
22             cout << sudokuCount << "个指定挖空的数独游戏已成功生成,
                并保存generate.txt中\n对应的终局已保存到solveSheet.
                txt中" << endl;
23         }
24         .....
```

修改后重新运行 clang-tidy, 结果如下:



```
PS A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku> cl /analyze main.cpp
用于 x86 的 Microsoft (R) C/C++ 优化编译器 19.29.30141 版
版权所有(C) Microsoft Corporation。保留所有权利。

main.cpp
Microsoft (R) Incremental Linker Version 14.29.30141.0
Copyright (C) Microsoft Corporation. All rights reserved.

/out:main.exe
main.obj
```

图 11: 最终结果 2

可以看到, 经过修改, clang-tidy 中也成功消除所有警告。

四、覆盖率测试分析

(一) 测试用例表

以下是综合本数独软件用户手册考虑给出的测试用例表：

命令行	解释说明	命令行	解释说明
-c 0	报错：参数超出范围	-c 1000001	报错：参数超出范围
-c 100	打印一百个数独终局		
-n	报错：参数缺失	-n 20	生成 20 个默认的数独游戏
-n 100 -m 3	生成 100 个难度为 3 的数独游戏	-n 100 -m 5	报错：参数超出范围
-n 100 -r 30	报错：参数缺失	-n 100 -r 30 40	生成 100 个挖空在 30 40 之间的数独游戏
-n 100 -r 15 60	报错：参数超出范围		
-n 20 -u	生成 20 个解唯一的数独游戏	-n 20 -u 8	报错：参数多余
-s	报错：参数缺失	-s "incomplete.txt"	报错：数独文件不完整
-s "game.tx"	报错：无法读取文件	-s "game.txt"	成功解决文件中的数独问题并存储答案

表 2: 代码覆盖率测试用例表

(二) 使用工具进行测试

1. 工具说明：OpenCPPCoverage

OpenCPPCoverage 是一款用于代码覆盖率测试的工具，它可以帮助开发者检测代码的执行情况，找出未被测试的代码段，提高代码质量和可靠性。OpenCPPCoverage 支持多种编程语言，如 C++，C#，Java 等，也支持多种测试框架，如 Google Test，NUnit，JUnit 等。OpenCPPCoverage 可以生成 HTML 格式的 report，方便查看和分析代码覆盖率的结果。

2. 覆盖率测试过程

这款工具也可以作为 Visual Studio 的插件使用。安装、使用及覆盖率测试过程如下：

1. 在 Visual Studio 中安装该工具、并编译了项目之后，就可以直接在解决方案窗口中选择"Run OpenCPPCoverage"来运行这款工具分析代码覆盖率。
2. 在设定中输入运行程序的参数，并点击运行。可以看到以下结果：

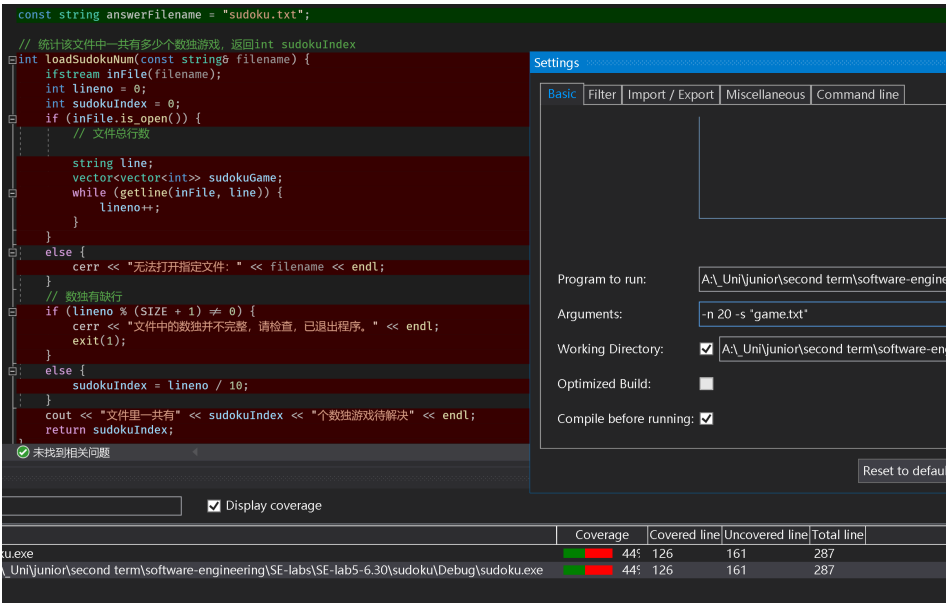


图 12: 对参数“-s game.txt” 进行覆盖率测试

- 如图12所示，红色标注的代码是违背使用到的，绿色标注的代码是被使用到的，本项目初次测试的代码覆盖率较低。
 - 原因一：测试用例未覆盖所有分支语句。
 - 原因二：存在部分冗余代码。
3. 紧接着根据表2中的测试用例，在 OpenCPPCoverage 的设置中填入参数（如图13）。每个参数都需要修改 Export Path（输出类型为 Binary），并注意对输出的.cov 文件命名应当清晰明了，一目了然文件对应的测试场景。

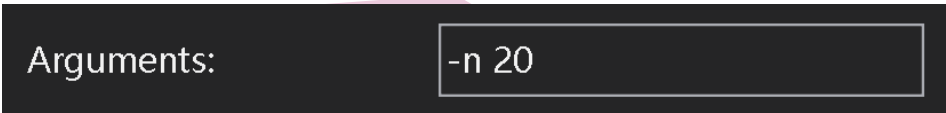


图 13: OpenCPPCoverage 设置的参数栏

4. 将测试用例表（表2）中的所有用例都按照上一步的方法输出为.cov 之后，在插件设置的 Input Coverage 栏内加入所有.cov 文件（如图14所示），表示对所有分支语句进行测试并输出为一个报告。



图 14: 在 OpenCPPCoverage 中添加测试用例覆盖文件

5. 点击运行覆盖率测试，可以得到如图15所示的结果。

sudoku.exe		82%	306	67	373
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\Debug\sudoku.exe		82%	306	67	373
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\solver.h		66%	93	48	141
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp		86%	90	15	105
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate		97%	123	4	127

图 15: 初次覆盖率测试结果截图

- 该图中得到的总覆盖率为 93%，说明仍有部分分支未被覆盖或者仍有代码冗余。
- 进入 main.cpp，发现覆盖率为 85%，而 main.cpp 是分支语句主要所在文件。故而推测仍有用例未被覆盖。

3. 修改后覆盖率再测试

根据初步覆盖率测试得到的结果进行**加入更多测试用例**和**修改冗余代码**两个方面的改进。修改后再次进行覆盖率得到以下结果：

	Coverage	Covered line	Uncovered line	Total line
sudoku.exe	93%	266	21	287
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\Debug\sudoku.exe	93%	266	21	287
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\main.cpp	85%	53	9	62
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\solver.h	92%	94	8	102
A:_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30\sudoku\sudoku\generate	97%	119	4	123

图 16: 修改过后进行覆盖率测试的截图

(三) 覆盖率说明

本程序中存在六个参数，对于它们的分支讨论由图17展现的思维导图所示，分别有以下情况：

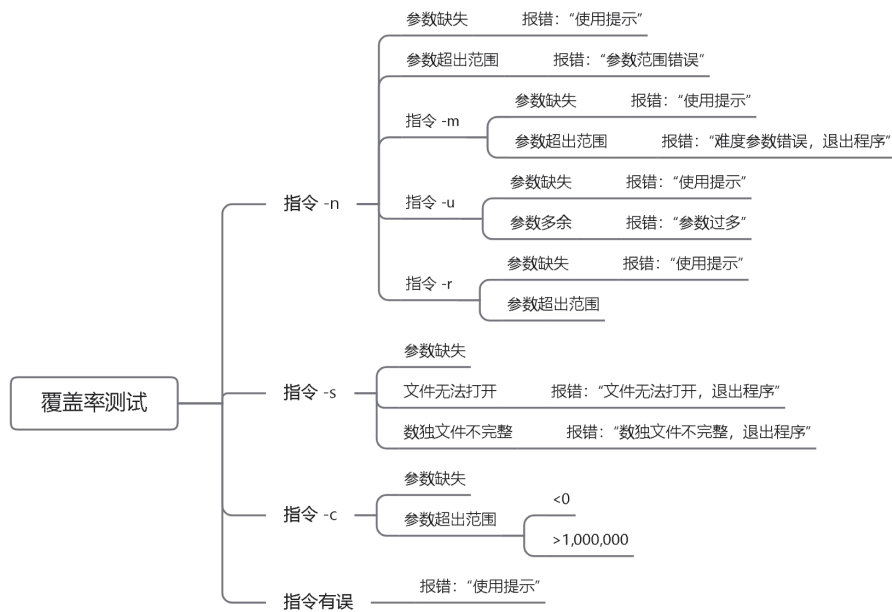


图 17: 展现程序分支逻辑的思维导图

- **参数不足**: 总参数小于两个或类型不正确, 如 `1234`, `-n`, `-s`

```

\sudoku\Debug> ./sudoku.exe -n 20 -r op op
ERROR: -r的参数无效
PS A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30
\sudoku\Debug> ./sudoku.exe -n 20 -r 43 29
ERROR: -r的参数范围错误, 请从小到大输入20~55之间的两个整数
  
```

- **不存在该参数**: 参数大于两个且类型正确, 但设定没有该参数, 如 `-a 20`

```

\sudoku\Debug> ./sudoku.exe -a 20
ERROR: 无效参数 -a
  
```

- **-c**

- **-c <number_of_game>**: 成功输出指定个数的数独终局
- **-c 参数范围错误**: -c 后的参数小于 0 或大于 1,000,000

```

\sudoku\Debug> ./sudoku.exe -c 0
ERROR: -c <number_of_games> 参数范围为0~1,000,000
  
```

- **-n**

- **-n <number_of_game>**: 成功输出指定个数的默认数独游戏
- **-n 参数范围错误**: -n 后的参数小于 0 或大于 100,000

```

\sudoku\Debug> ./sudoku.exe -n 100000
ERROR: -n之后的参数范围错误, 请输入一个在0~10,000之间的整数
  
```

- **-n 参数类型错误**: -n 后的参数不是整型, 如 `-n -u`

```

\sudoku\Debug> ./sudoku.exe -n -u
ERROR: -n之后的参数类型错误, 请输入一个在0~1,000,000的整数
  
```

- -u

- -n 100 -u: 成功输出指定个数的唯一解数独游戏
- -u 参数过多: -u 后的参数过多, 本来不需要参数如 -n 100 -u 8

```
\sudoku\Debug> ./sudoku.exe -n 20 -u 3
ERROR: -u的参数过多
```

- -m

- -n 100 -m 1: 成功输出指定难度等级的数独游戏
- -m 参数范围错误: -m 后的参数应当介于 1 和 3 之间

```
\sudoku\Debug> ./sudoku.exe -n 20 -m 6
无效的的难度等级参数: 6
```

图 18: 错误的难度等级

- -r

- -n 20 -r <min> <max>: 成功输出指定挖空数的数独游戏
- -r 参数范围错误: -r 后的参数应该由小到大介于 20 55
- -n 参数类型错误: -r 后的参数不是整型, 如 -n 100 -r op op

```
\sudoku\Debug> ./sudoku.exe -n 20 -r op op
ERROR: -r的参数无效
PS A:\_Uni\junior\second term\software-engineering\SE-labs\SE-lab5-6.30
\sudoku\Debug> ./sudoku.exe -n 20 -r 43 29
ERROR: -r的参数范围错误, 请从小到大输入20~55之间的两个整数
```

- -s

- -s <path_of_file>: 成功解出指定位置的数独, 并存储答案
- -s 参数无法解析: -s 后的参数代表的文件无法打开

```
\sudoku\Debug> ./sudoku.exe -s "game.t"
无法打开指定文件: game.t
```

- -s 打开的文件不完整: 打开的文件中, 数独不完整, 无法解, 如数独棋盘缺行

```
\sudoku\Debug> ./sudoku.exe -s "A:\se\incomplete.txt"
ERROR: 文件中的数独并不完整, 请检查
```

五、 源码链接

请访问以下链接: <https://github.com/runxii/SE-sudoku> 查看项目源码。