用户手册

1 引言和概述

1.1 产品或服务介绍

生成数独游戏并求解数独问题的控制台程序是一个用于自动生成数独游戏并提供解决方案的工具。数独是一种逻辑谜题，由 9 x 9 的网格组成，玩家需要填入数字 1 到 9，使得每行、每列和每个 3 x 3 的子网格中的数字都不重复。该控制台程序能够自动生成具有唯一解的数独终局，并提供求解数独问题的功能。

1.2 手册目的和使用对象

本手册旨在向用户提供有关生成数独游戏并求解数独问题的控制台程序的详细信息和操作指南。它适用于计算机科学爱好者、编程学习者以及对数独游戏感兴趣的用户。本手册将帮助您了解如何正确安装、配置和使用该控制台程序，并获得最佳的用户体验。

本项目的github仓库链接 (https://github.com/erssss/2023SE-Sudoku)

1.3 背景信息

数独游戏作为一种受欢迎的智力游戏，在全球范围内拥有广泛的玩家群体。然而，手动创建数独终局并求解数独问题需要一定的时间和技巧。因此，生成数独游戏并求解数独问题的控制台程序应运而生，它能够自动创建数独终局，并提供一种快速有效的方式来解决数独问题。
该控制台程序基于 C++语言实现，支持在 64-bit Windows 10 操作系统下运行。它遵循高质量编码标准，并经过了代码质量分析，以确保稳定性和可靠性。
通过本手册，您将了解如何正确安装和配置该控制台程序，生成数独终局，并使用程序提供的功能来解决数独问题。

2 安装和设置

2.1 系统要求和依赖项

在使用生成数独游戏并求解数独问题的控制台程序之前，请确保您的系统满足以下要求，并已安装所需的依赖项。
系统要求：

- 操作系统：64 位 Windows 10/11。
- 内存：建议至少 8 GB RAM，以确保足够的内存空间来运行程序。
- 存储空间：控制台程序的安装和生成的数独文件可能占用一定的磁盘空间，建议预留足够的存储空间。

2.2 安装步骤

在本地文件夹中，拉取代码并进行编译：

```
git clone https://github.com/erssss/2023SE-Sudoku.git
cd 2023SE-Sudoku
make
```

参照入门指南中的基本操作步骤使用命令进行操作即可。

3 快速入门指南

3.1 基本操作步骤

本程序所使用的命令行参数如下：

参数名字	参数意义	范围限制	用法示例	意义
-c	需要的数独终盘数量	1-10000000	sudoku -c 20	表示生成 20 个数独终盘
-s	需要解的数独棋盘文件路径	绝对或相对路径	sudoku -s game. Txt	表示从 game. Txt 读取若干个数独游戏，并给出其解答，生成到本地文件中
-n	需要的游戏数量	1-10000	sudoku -n 1000	表示生成 1000 个数独游戏

参数名字	参数意义	范围限制	用法示例	意义
-m	生成游戏的难度	1-3	sudoku -n 1000 -m 1	表示生成 1000 个简单数独游戏，只有 m 和 n 一起使用才认为参数无误，否则请报错
-r	生成游戏中挖空的数量范围	20-55	sudoku -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，只有 r 和 n 一起使用才认为参数无误，否则请报错
-u	生成游戏的解唯一		sudoku -n 20 -u	表示生成 20 个解唯一的数独游戏，只有和 n 一起使用才认为参数无误，否则请报错
-f	进行数独游戏		sudoku -f	用户自行求解数独

4 功能说明

4.1 主要功能

- ⓘ 该项目主要包括三个模块的功能：

1. Solver：主要用于**求解数独**。根据给定的数独谜题进行求解。
2. Generator：用于**生成数独终局/谜题**。根据用户具体的数量、难度等要求生成相应的数独。
3. Fitter：用于**用户自行解数独**的功能。通过按键在数独盘上移动填写数字，并检测数独是否正确填写。

4.2 基础功能

4.2.1 生成数独终盘

- 说明使用命令 `sudoku.exe -c 20` 可生成 20 个数独终盘
- 操作示例

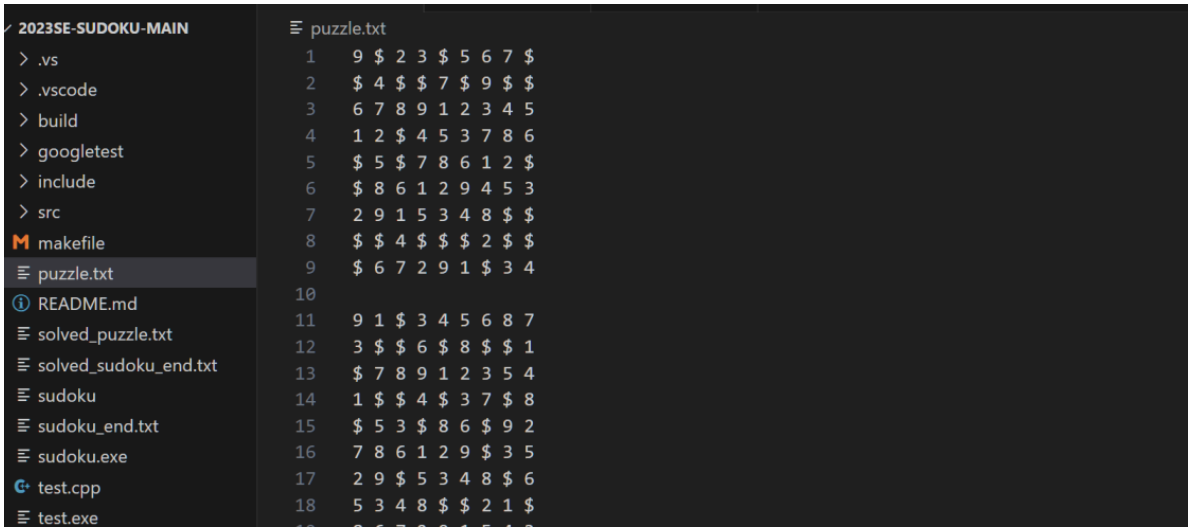
```
D:\MyFiles\software engineering\2023SE-Sudoku-main>sudoku -c 20
[成功]数独终局生成成功
```

4.2.2 生成数独游戏

- 说明使用命令 `sudoku.exe -n [count]` 可生成指定数量的数独游戏，可以配合下面的参数实现指定要求的数独游戏的生成：-m 可以指定游戏的难度，而-r 可以指定挖空的数量，-u 可以指定生成的数独游戏具有唯一解。
- 操作示例

```
D:\MyFiles\software engineering\2023SE-Sudoku-main>sudoku -n 10 -u
[成功]数独谜题生成成功
```

在 puzzle. Txt 中生成了未解开的数独游戏：

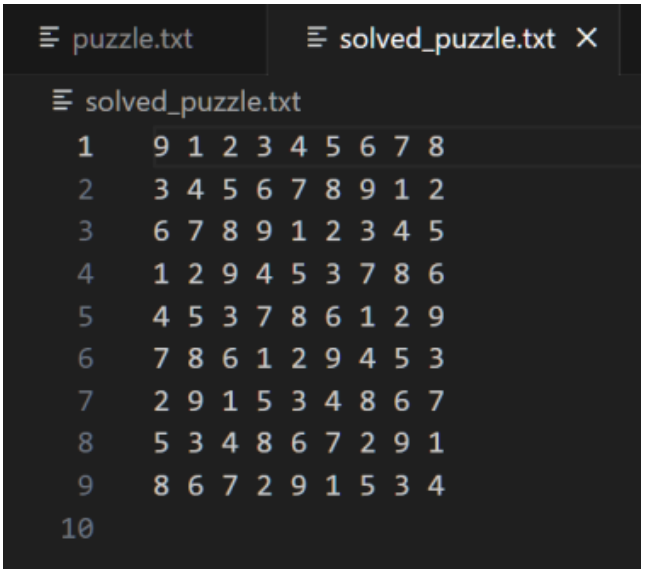


4.2.3 求解数独游戏

- 说明使用命令 `sudoku.exe -s [filepath]` 可将文件中的数独谜题求解
- 操作示例

```
D:\MyFiles\software engineering\2023SE-Sudoku-main>sudoku -s puzzle.txt
input_path = puzzle.txt
9 1 2 3 4 5 6 7 8
3 4 5 6 7 8 9 1 2
6 7 8 9 1 2 3 4 5
1 2 9 4 5 3 7 8 6
4 5 3 7 8 6 1 2 9
7 8 6 1 2 9 4 5 3
2 9 1 5 3 4 8 6 7
5 3 4 8 6 7 2 9 1
8 6 7 2 9 1 5 3 4
cnt = 162
[成功]数独求解成功
```

同时，结果也会写入到 solved_puzzle. Txt 中：

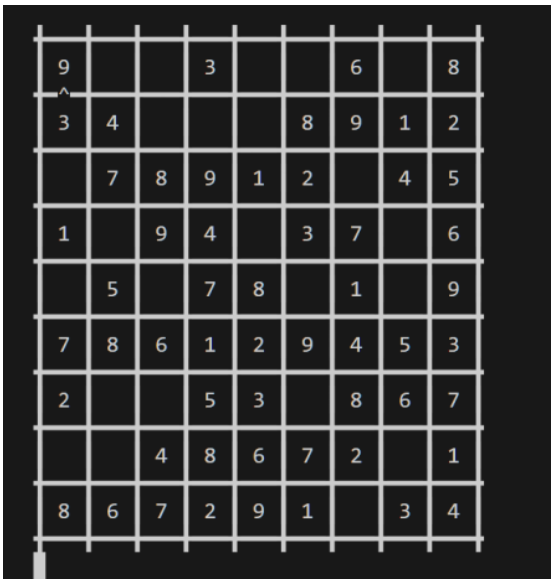


4.3 进阶功能：在线解数独

使用命令：

```
sudoku -f
```

即可打开数独问题文件里的数独谜题进行在线解题，如图所示：

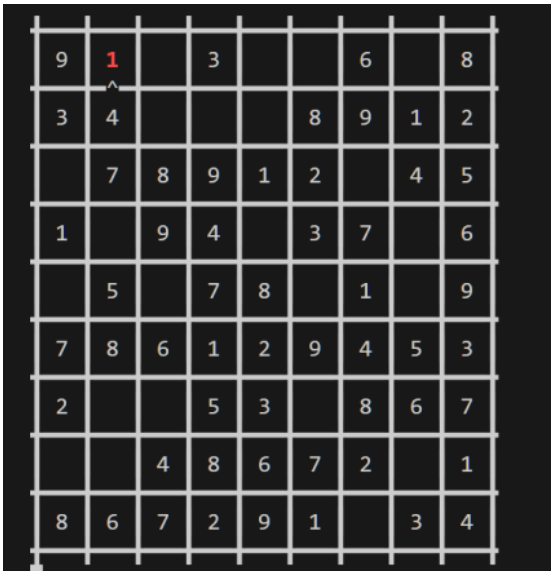


4.3.1 操作说明

使用的键功能如下：

Esc（退出）	W（↑）	U（撤回）
A（←）	S（↓）	D（→）

填入数字如下：



5 质量分析

编写脚本如下，对 `src` 及 `include` 文件夹下所有源代码进行代码的质量检测与分析，并消除所有警告：

```
echo ********** START TEST ********** >> %OUTPUT_FILE%

set CPPLINT= < cpplint.exe 路径 >
set INCLUDE_DIR="G:/repo/2023SE-sudoku/include"
set SOURCE_DIR="G:/repo/2023SE-sudoku/src"
set OUTPUT_FILE=lint_result.txt

for /R %INCLUDE_DIR% %%G in (*.cpp, *.h) do (
    echo --------- Checking: %%G --------- >> %OUTPUT_FILE%
    %CPPLINT% %%G >> %OUTPUT_FILE%
)
for /R %SOURCE_DIR% %%G in (*.cpp, *.h) do (
    echo --------- Checking: %%G --------- >> %OUTPUT_FILE%
    %CPPLINT% %%G >> %OUTPUT_FILE%
)
echo ********** END TEST ********** >> %OUTPUT_FILE%

type %OUTPUT_FILE%

```

测试结果如下，可以看到已经消除所有警报：

```
********** START TEST **********
--------- Checking: G:\repo\2023SE-sudoku\include\controller.h ---------
Done processing G:\repo\2023SE-sudoku\include\controller.h
--------- Checking: G:\repo\2023SE-sudoku\include\fitter.h ---------
Done processing G:\repo\2023SE-sudoku\include\fitter.h
--------- Checking: G:\repo\2023SE-sudoku\include\generator.h ---------
Done processing G:\repo\2023SE-sudoku\include\generator.h
--------- Checking: G:\repo\2023SE-sudoku\include\solver.h ---------
Done processing G:\repo\2023SE-sudoku\include\solver.h
--------- Checking: G:\repo\2023SE-sudoku\src\controller.cpp ---------
Done processing G:\repo\2023SE-sudoku\src\controller.cpp
--------- Checking: G:\repo\2023SE-sudoku\src\fitter.cpp ---------
Done processing G:\repo\2023SE-sudoku\src\fitter.cpp
--------- Checking: G:\repo\2023SE-sudoku\src\generator.cpp ---------
Done processing G:\repo\2023SE-sudoku\src\generator.cpp
--------- Checking: G:\repo\2023SE-sudoku\src\main.cpp ---------
Done processing G:\repo\2023SE-sudoku\src\main.cpp
--------- Checking: G:\repo\2023SE-sudoku\src\solver.cpp ---------
Done processing G:\repo\2023SE-sudoku\src\solver.cpp
********** END TEST **********
```

根据分析结果，所有文件都通过了 cpplint 的检查，没有发现任何问题。这意味着代码在格式和风格方面符合 cpplint 的要求，没有触发任何警告或错误。

6 测试样例

所使用的测试样例如下表：

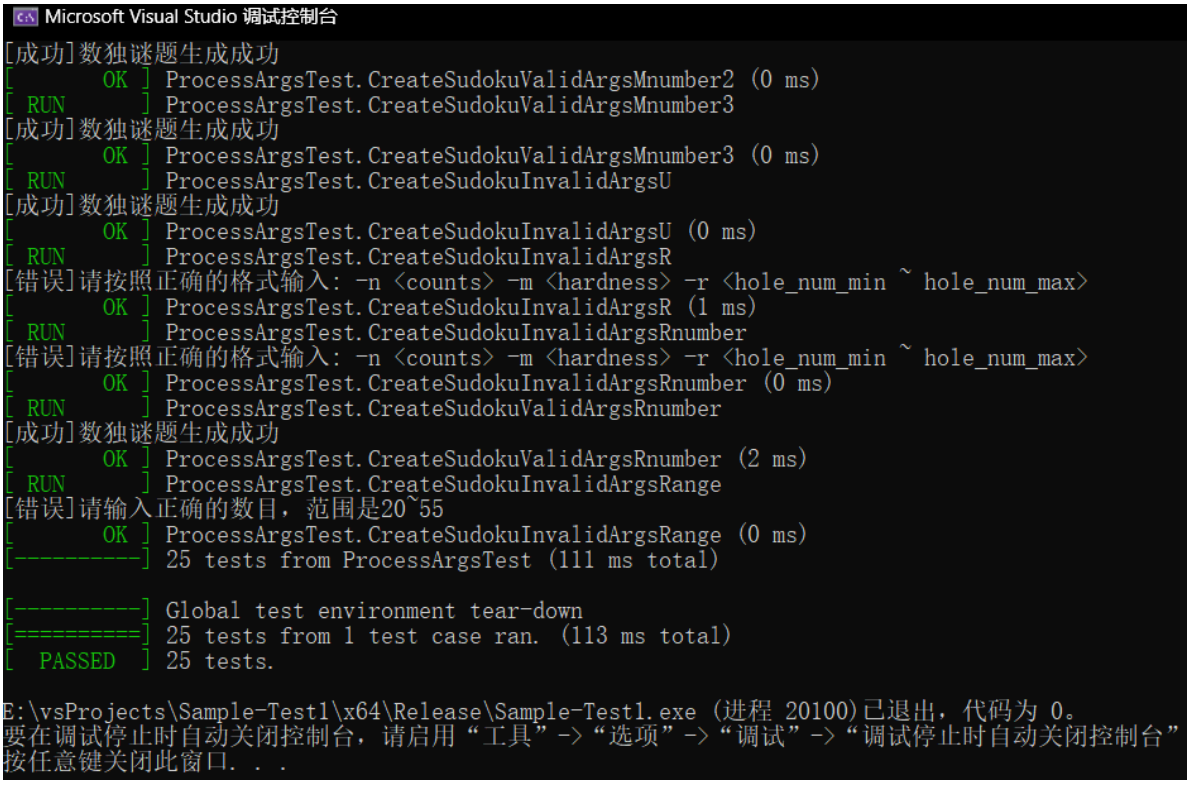
测试编号	测试用例	描述	输入参数	预期结果
1	SolveSudokuWithPathValidTest	使用有效路径解决数独	program_name, -s, G:/repo/2023 SE-sudoku/puzzle. Txt	ASSERT_TRUE

测试编号	测试用例	描述	输入参数	预期结果
2	SolveSudokuWithPathInvalidTest	使用无效路径解决数独	program_name, -s, invalid_path. Txt	ASSERT_FALSE
3	SolveSudokuWithCountsValidTest	使用有效路径和计数解决数独	program_name, -s, G:/repo/2023 SE-sudoku/puzzle. Txt, -n, 1	ASSERT_TRUE
4	SolveSudokuWithCountsInvalidTest_neg	使用有效路径和负数计数解决数独	program_name, -s, G:/repo/2023 SE-sudoku/puzzle. Txt, -n, -10	ASSERT_FALSE
5	SolveSudokuWithCountsInvalidTest_zero	使用有效路径和零计数解决数独	program_name, -s, G:/repo/2023 SE-sudoku/puzzle. Txt, -n, 0	ASSERT_FALSE
6	SolveSudokuInvalidArgsTest	缺少路径解决数独	program_name, -s	ASSERT_FALSE
7	CreateSudokuWithCountsValidTest	使用有效计数创建数独	program_name, -c, 10	ASSERT_TRUE
8	CreateSudokuInvalidArgsTest	缺少计数创建数独	program_name, -c	ASSERT_FALSE
9	PrintHelpTest	打印帮助信息	program_name, -h	ASSERT_TRUE
10	OtherInvalidArgsTest	使用其他无效参数	program_name, -x	ASSERT_FALSE
11	CountsBoundaryTest	使用边界计数创建数独	program_name, -c, 1000000	ASSERT_TRUE
12	CountsInvalidBoundaryTest	使用无效边界计数创建数独	program_name, -c, 0	ASSERT_FALSE
13	CreateSudokuInvalidArgsN	缺少 n 参数创建数独	program_name, -n	ASSERT_FALSE
14	CreateSudokuValidArgsN	使用有效 n 参数创建数独	program_name, -n, 5	ASSERT_TRUE

测试编号	测试用例	描述	输入参数	预期结果
15	CreateSudokuInvalidArgsNCounts	使用无效 n 参数和计数创建数独	program_name, -n, 0	ASSERT_FALSE
16	CreateSudokuInvalidArgsM	缺少 m 参数创建数独	program_name, -m	ASSERT_FALSE
17	CreateSudokuInvalidArgsMnumber	使用无效 m 参数创建数独	program_name, -n, 2, -m, 5	ASSERT_FALSE
18	CreateSudokuValidArgsMnumber 1	使用有效 m 参数 1 创建数独	program_name, -n, 2, -m, 1	ASSERT_TRUE
19	CreateSudokuValidArgsMnumber 2	使用有效 m 参数 2 创建数独	program_name, -n, 2, -m, 2	ASSERT_TRUE
20	CreateSudokuValidArgsMnumber 3	使用有效 m 参数 3 创建数独	program_name, -n, 2, -m, 3	ASSERT_TRUE
21	CreateSudokuInvalidArgsU	使用无效 u 参数创建数独	program_name, -n, 2, -u	ASSERT_TRUE
22	CreateSudokuInvalidArgsR	缺少 r 参数创建数独	program_name, -n, 2, -r	ASSERT_FALSE
23	CreateSudokuInvalidArgsRnumber	使用无效 r 参数创建数独	program_name, -n, 2, -r, 20-50	ASSERT_FALSE
24	CreateSudokuValidArgsRnumber	使用有效 r 参数创建数独	program_name, -n, 2, -r, 20~50	ASSERT_TRUE
25	CreateSudokuInvalidArgsRange	使用无效范围参数创建数独	program_name, -n, 2, -r, 5~80	ASSERT_FALSE

如图所示，使用 `Google Test + openCppCoverage` 进行测试，除了拓展中需要用户额外响应的 fitter 外，可以看到 25 个测试样例均通过，代码覆盖率达到 100%。

控制台截图如下：



测试资源管理器如下：

覆盖率报告如下：