

用户手册

数独系统

修改历史

版本	日期	说明	作者
v1.0	2023-06-23	实现数独终局的生成功能	齐传辉
v2.0	2023-06-24	数独程序的求解功能	周子跃
v3.0	2023-06-25	完成最终的数独程序	齐传辉、周子跃
v4.0	2023-06-26	代码质量测试+单元测试+用户手册的编写	周子跃、齐传辉

目录

- 1. 引言 3
 - 1.1. 编写目的 3
 - 1.2. 用户对象 3
 - 1.3. 环境要求 3
- 2. 软件配置 3
 - 2.1 项目总体 3
 - 2.2. 参数配置 4
- 3. 软件综述 5
 - 3.1. 系统简介 5
 - 3.2. 系统流程 5
- 4. 操作说明 5
 - 4.1. 功能一 5
 - 4.2. 功能二 5
 - 4.3. 功能三 5
 - 4.4. 功能四 6
 - 4.5. 功能五 6
- 5. 质量测试与单元测试 6

1. 引言

1.1. 编写目的

数独是由 9 个 3×3 的小棋盘所组成的 9×9 的大棋盘，小棋盘包含 1-9 这 9 个数字，大棋盘的每一行、每一列的 9 个数字均不相同，即每一行、每一列都包含 1-9 这 9 个数字。随着数独运动的广泛传播和深受喜爱，本项目旨在开发一个控制台应用来实现数独游戏的生成和求解。用户可以通过在控制台中输入不同的指令来调用程序中不同的函数以实现不同的功能，其中包括生成数独终局、生成数独游戏、解决数独问题等功能。本手册将详细介绍项目的背景、目的、范围和预期成果。

1.2. 用户对象

本项目面相的用户对象主要是对数独游戏感兴趣的人群，比如希望通过程序可以实现自动出题、解题功能的人群。同时，本项目还面相对 C++ 编程和控制台应用开发有兴趣的读者。

1.3. 环境要求

64-bit Windows 10 环境。

2. 软件配置

2.1 项目总体

在整个项目中我们主要编写了四个 `cpp` 文件，分别是 `main.cpp`、`zhongju.cpp`、`qiujie.cpp` 和 `youxi.cpp`。其中，`zhongju.cpp`、`youxi.cpp` 和 `qiujie.cpp` 是程序功能的核心部分，里面包含了终局生成、数独问题生成和数独求解等功能的算法。`main.cpp` 是程序的主函数，包含了程序的入口和对控制台指令的解析等功能。同时，我们还通过 `main.cpp` 这一文件和解析后的结果来调用其余两个文件中编写的函数以实现功能。上述所有代码文件可以在 Visual Studio 2019 中进行编译和运行，在生成可执行文件后可以在 Windows 10 系统下运行。

2. 2. 参数配置

参数名字	参数意义	范围限制	用法示例
-c	需要生成的数独终盘数量	1-1000000	示例：shudu.exe -c 20[表示生成 20 个数独终局到 shudu.txt 中]
-s	需要求解的数独棋盘文件路径	绝对或相对路径	示例：shudu.exe -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 jieguo.txt 中]
-n	需要生成的游戏数量	1-10000	示例：shudu.exe -n 1000[表示生成 1000 个数独游戏到 game.txt 中，同时将答案输出到 answer.txt 中]
-m	生成游戏的难度	1-3	示例：shudu.exe -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	游戏中挖空的数量范围	20-55	示例：shudu.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，只有 r 和 n 一起使用才认为参数无误，否则请报错]
-u	生成解唯一的数独游戏		示例：shudu.exe -n 20-u[表示生成 20 个解唯一的数独游戏，只有 u 和 n 一起使用才认为参数无误，否则请报错]

3. 软件综述

3.1. 系统简介

本数独系统是一个基于控制台的数独游戏生成和求解应用，使用 C++ 编写而成。用户可以通过在控制台中输入不同的指令来调用应用的不同功能，包括生成数独终局、生成数独游戏、解决数独问题等。

3.2. 系统流程

用户可以通过编译四个 `cpp` 源文件来生成可执行文件 `shudu.exe`，之后在控制台中执行此 `exe` 文件并且添加相应的 `-c/-n` 等字母和参数来实现具体的功能例如： `./shudu.exe -n 100`，之后可执行文件会调用相应的文件和函数来响应功能，最终在目录下输出对应条件下的结果。

4. 操作说明

4.1. 功能一

4.1.1. 功能描述

生成指定数量的数独终局。

4.1.2. 操作方法

示例： `shudu.exe -c 20`

4.2. 功能二

4.2.1. 功能描述

从给定的文件路径中读取数独游戏并给出其解答，将结果输出到 `answer.txt` 中。

4.2.2. 操作方法

示例： `shudu.exe -s game.txt`

4.3. 功能三

4.3.1. 功能描述

表示生成指定数量的简单数独游戏。

4.3.2. 操作方法

示例：shudu.exe-n 1000-m 1， 只有 m 和 n 一起使用才认为参数无误， 否则请报错。

4.4. 功能四

4.4.1. 功能描述

表示生成指定数量的挖空数在指定范围之间的数独游戏。

4.4.2. 操作方法

示例：shudu.exe-n 20-r 20~55， 只有 r 和 n 一起使用才认为参数无误， 否则请报错。

4.5. 功能五

4.5.1. 功能描述

生成指定个数的解唯一的数独游戏。

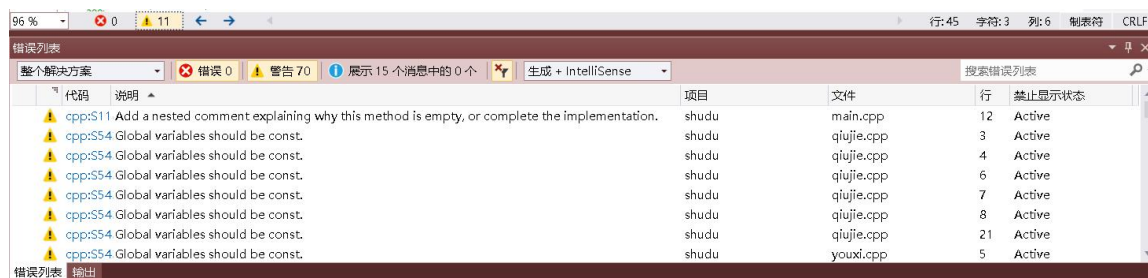
4.5.2. 操作方法

示例：shudu.exe-n 20-u， 只有 u 和 n 一起使用才认为参数无误， 否则请报错。

5. 质量测试与单元测试

5.1. 质量测试

质量测试采用 visual studio 2019 上的 SonarLint 来执行， 测试结果如下：



图一：质量测试

查看质量检测反馈出的警告信息可以看到，警告大多是以下几种（只列举其中部分）：

Add a nested comment explaining why this method is empty, or complete the implementation.

Global variables should be const.

implicit conversion loses integer precision: 'int' to 'char'

Use "std::array" or "std::vector" instead of a C-style array.

可以看到警告大部分都是由于参数的类型不正确。而前面的警告则大多出自应用的运行过程中对命令行输入的指令和参数进行信息的提取和判别，并且在 `main.cpp` 中有许多分支与运算会引发警告而无法避免。在 `youxi.cpp` 中，定义很多函数和变量都需要进行全局定义以及引用和判断，在代码编写上很难避免出现警告。

5.2. 单元测试

通过 vs2019 中的本机单元测试项目也就是 `unittest` 对项目进行单元测试，分别对具体用例中参数的五种情况（`-c,-n,-m,-r,-u`）进行测试：

```
TEST_METHOD(TestMethod_c1)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-c";
    argv[2] = "0";
    Assert::AreEqual(0, main(argc, argv));
}

- 测试状态(C++、C# 和 Visual Basic)
TEST_METHOD(TestMethod_c2)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-c";
    argv[2] = "100";
    Assert::AreEqual(1, main(argc, argv));
}

TEST_METHOD(TestMethod_c3)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-c";
    argv[2] = "1000001";
    Assert::AreEqual(0, main(argc, argv));
}
```

图二：单元测试（1）

```
TEST_METHOD(TestMethod_s1)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-s";
    argv[2] = "game.txt";
    Assert::AreEqual(1, main(argc, argv));
}

TEST_METHOD(TestMethod_s2)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-s";
    argv[2] = "none.txt";
    Assert::AreEqual(1, main(argc, argv));
}
```

图三：单元测试（2）

```

TEST_METHOD(TestMethod_n1)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "1000";
    Assert::AreEqual(1, main(argc, argv));
}

```

图四：单元测试（3）

```

TEST_METHOD(TestMethod_nom)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-m";
    argv[2] = "3";
    Assert::AreEqual(0, main(argc, argv));
}

TEST_METHOD(TestMethod_nor)
{
    int argc = 3;
    char* argv[10];
    argv[1] = "-r";
    argv[2] = "20-50";
    Assert::AreEqual(0, main(argc, argv));
}

```

图五：单元测试（4）

```

TEST_METHOD(TestMethod_m1)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "0";
    argv[3] = "-m";
    argv[4] = "1";
    Assert::AreEqual(0, main(argc, argv));
}

```

```

TEST_METHOD(TestMethod_m2)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10001";
    argv[3] = "-m";
    argv[4] = "1";
    Assert::AreEqual(0, main(argc, argv));
}

```

```

TEST_METHOD(TestMethod_m3)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10";
    argv[3] = "-m";
    argv[4] = "0";
    Assert::AreEqual(0, main(argc, argv));
}

```

```

TEST_METHOD(TestMethod_r1)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10";
    argv[3] = "-r";
    argv[4] = "18-20";
    Assert::AreEqual(0, main(argc, argv));
}

```

```

TEST_METHOD(TestMethod_r2)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10";
    argv[3] = "-r";
    argv[4] = "55-60";
    Assert::AreEqual(0, main(argc, argv));
}

```

```

TEST_METHOD(TestMethod_r3)
{
    int argc = 5;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10";
    argv[3] = "-r";
    argv[4] = "30-50";
    Assert::AreEqual(1, main(argc, argv));
}

```

```

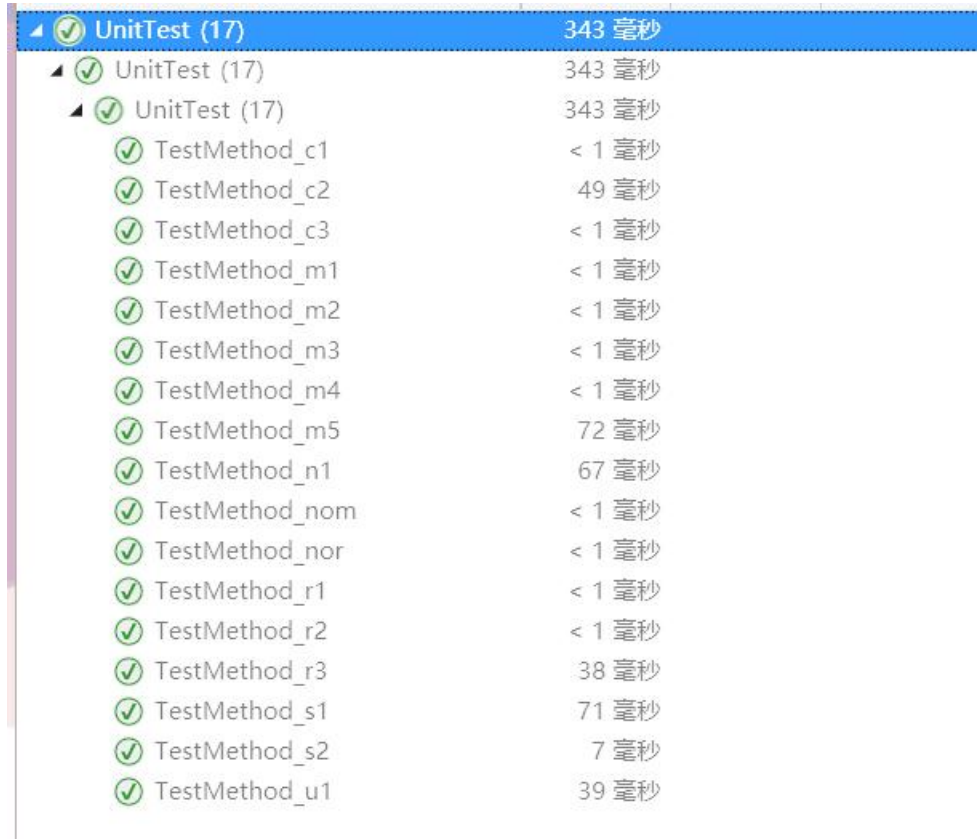
TEST_METHOD(TestMethod_u1)
{
    int argc = 4;
    char* argv[10];
    argv[1] = "-n";
    argv[2] = "10";
    argv[3] = "-u";
    Assert::AreEqual(1, main(argc, argv));
}

```


图六：单元测试（5）

图七：单元测试（6）

根据实验要求中的实例对每一种命令行的参数以不同数据进行测试，设置参数 `unittest`，当输入参数错误时 `unittest` 为 0，正确执行输出情况时为 1，在函数的最后返回 `unittest` 以完成测试，测试结果如下：



UnitTest (17)	343 毫秒
UnitTest (17)	343 毫秒
UnitTest (17)	343 毫秒
TestMethod_c1	< 1 毫秒
TestMethod_c2	49 毫秒
TestMethod_c3	< 1 毫秒
TestMethod_m1	< 1 毫秒
TestMethod_m2	< 1 毫秒
TestMethod_m3	< 1 毫秒
TestMethod_m4	< 1 毫秒
TestMethod_m5	72 毫秒
TestMethod_n1	67 毫秒
TestMethod_nom	< 1 毫秒
TestMethod_nor	< 1 毫秒
TestMethod_r1	< 1 毫秒
TestMethod_r2	< 1 毫秒
TestMethod_r3	38 毫秒
TestMethod_s1	71 毫秒
TestMethod_s2	7 毫秒
TestMethod_u1	39 毫秒

图八：单元测试结果

5.3. 代码仓库

Github 链接：<https://github.com/qichuanhui/SoftwareEngineering.git>