**Miner’s Coffee**

**A Powerful and Accessible**

**GPU Mining Software**

**TEST DOCUMENT**



Content

[Purpose 3](#_Toc71888200)

[Environment 3](#_Toc71888201)

[Testing Framework 3](#_Toc71888202)

[Test Implementation 5](#_Toc71888203)

[Test Initialization 5](#_Toc71888204)

[Test Utilities 5](#_Toc71888205)

[Unit Test 5](#_Toc71888206)

[Component/Subsystem Test 8](#_Toc71888207)

[Full System Test 8](#_Toc71888208)

[Nonfunctional Test 9](#_Toc71888209)

## Purpose

Testing is the process of executing a program to discover errors in the program. The purpose of testing is to find as many errors in the software as possible before it is put into production operation. Successful test can find the errors in the system and make the system run correctly.

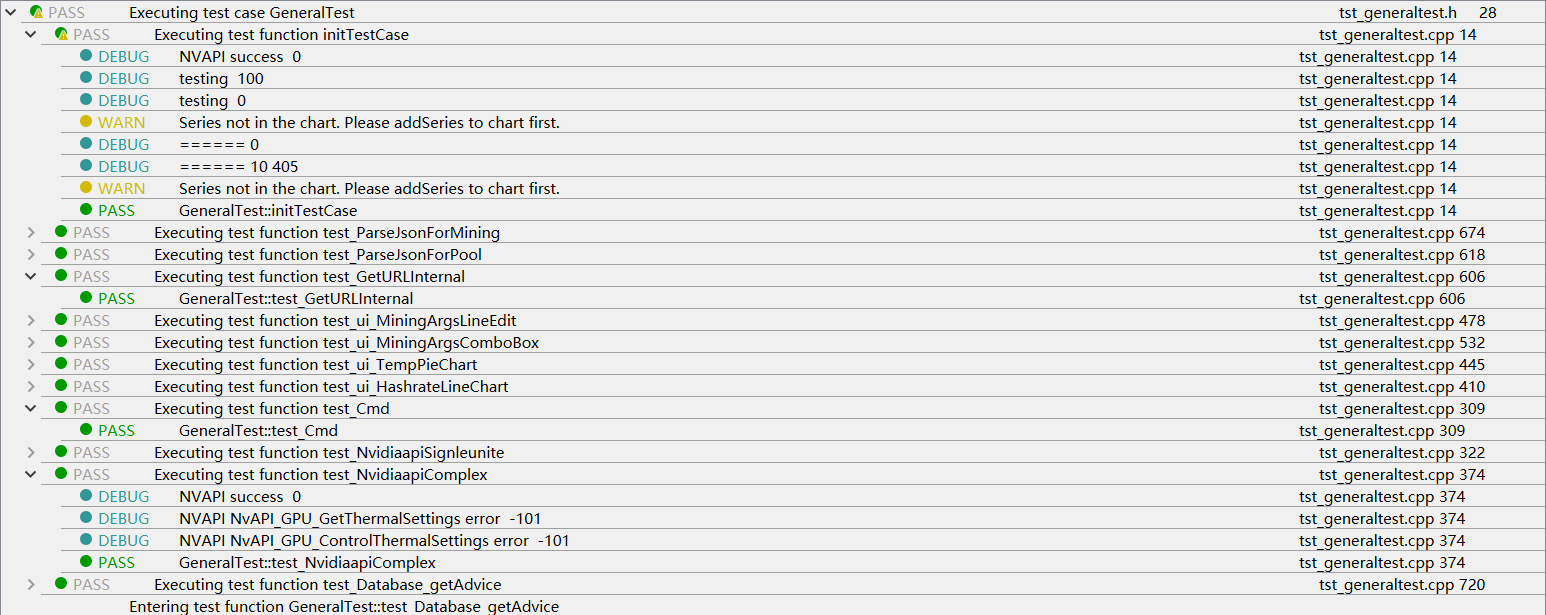
## Environment

The test project and the program project are two opposite Qt project. Rely on QTestLib, the software project needs to rely on this platform and the environment needs to be integrated.

## Testing Framework

QTestLib is a unit testing framework provided by Qt for programs or libraries written on Qt. QTestLib provides the basic functionality of the unit testing framework and provides extended functionality for GUI testing.

Use AutoTest plug-in to achieve visual effects.



**Figure 1**: Test Results

## Test Implementation

### Test Initialization

private:

MainWindow\* w;

Helper helper;

public:

**GeneralTest**();

~***GeneralTest***();

private:

void **initTestCase**();

Parameters: None.

Return: None.

Task: Initialize private variables MainWindow w.

void **cleanupTestCase**();

Parameters: None.

Return: None.

Task: Stop Mining-core and delete Main-window w.

### Test Utilities

private:

void **GetTest data**(QList<QString>& input, QList<QString>& result, const QString& in\_filename, const QString& res\_filename);

Parameters: Pointers to input and results’ data file data and filename.

Return: None.

Task: Input the parameter data and expected results in pairs to compare for tests

void **ShowDataError**(const QString& filename1, const QString& filename2);

Parameters: Pointers to input data filename and result data filename.

Return: None.

Task: Warn errors’ location when tests goes wrong.

### Unit Test

* 1. Command Line Module

void **test\_Cmd**():

Task: Check if we get right information of the disk memory size from Wincmd.

Test data: Directly passed

Test data size: 1

* 1. GPU Monitoring & Overclocking Module

void **test\_NvidiaapiSetTempLimit**();

Task: Check if program reset the gpu temprature limit correctly

Test data: Passed by function **test\_NvidiaapiSetTempLimit\_data**()

Test data size: 5

void **test\_NvidiaapiSetGPUoffset**();

Task: Check if program reset the gpu offset correctly

Test data: Passed by function **test\_NvidiaapiSetGPUoffset\_data**()

Test data size: 5

void **test\_NvidiaapiSetMemoffset**();

Task: Check if program reset the memory offset correctly

Test data: Passed by function **test\_NvidiaapiSetMemoffset\_data**()

Test data size: 5

void **test\_NvidiaapiGetTemp**();

Task: Check if program successfully access to the nvidia api and get gpuinfo.

Test data: Directly passed

Test data size: 1

void **test\_NvidiaapiComponent**();

Task: Check if program successfully access to the nvidia api and get the temperature.

Test data: Passed by function **test\_NvidiaapiComponent\_data**()

Test data size: 5

void **test\_NvidiaapiControlTest**():

Task: Check if program successfully control the gpuinfo

Test data: Directly passed

Test data size: 3

* 1. Network Module

void **test\_GetURLInternal**();

Task: Check if program successfully connect to sparkpool

Test data: Directly passed

Test data size: 1

* 1. Json Parsing Module

void **test\_ParseJsonForMining**();

Task: Check if program successfully access to the pool api and get mining information.

Test data: Passed by function **test\_ParseJsonForMining\_data**()

Test data size: 3

QString input\_filename = "test\_ParseJsonForMining\_input.txt";

QString result\_filename = "test\_ParseJsonForMining\_result.txt";

void **test\_ParseJsonForPool**();

Task: Check if program successfully access to the pool api and get pool information.

Test data: Passed by function **test\_ParseJsonForPool\_data**()

Test data size: 3

QString input\_filename = "test\_ParseJsonForPool\_input.txt";

QString result\_filename = "test\_ParseJsonForPool\_result.txt";

* 1. Database System

void **test\_Database\_getAdvice**();

Task: Check if program successfully connect to database and get the advice data

Test data: Directly passed

Test data size: 11

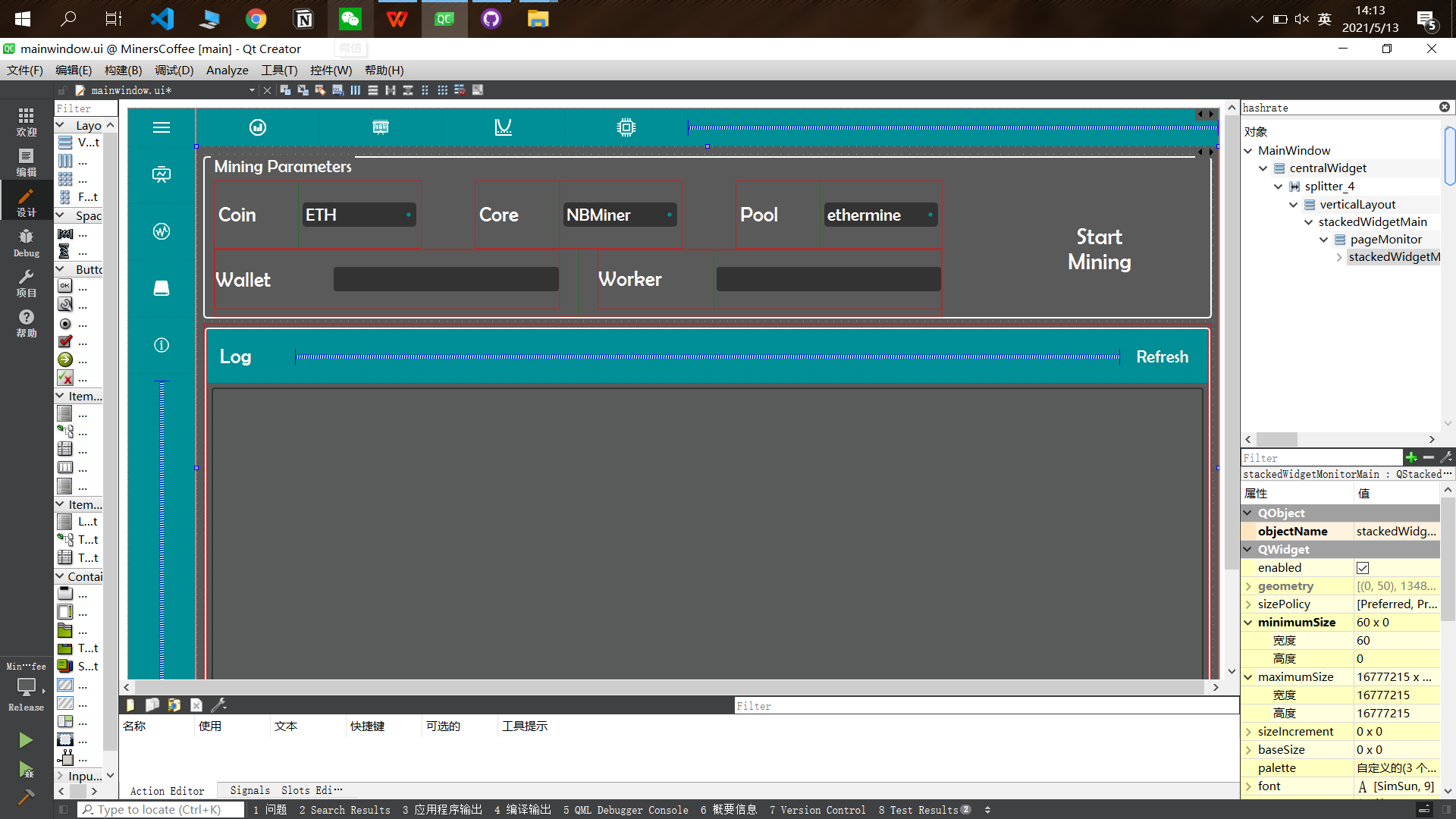
* 1. User Interface

For the test of GUI graphic operation, set the data as the event list for simulation test. Use, for example, built-in functions, passed through internal events, to simulate the events of the local window system.

void **test\_ui\_MiningArgsLineEdit**();

void **test\_ui\_MiningArgsLineEdit\_data**();

Test component:

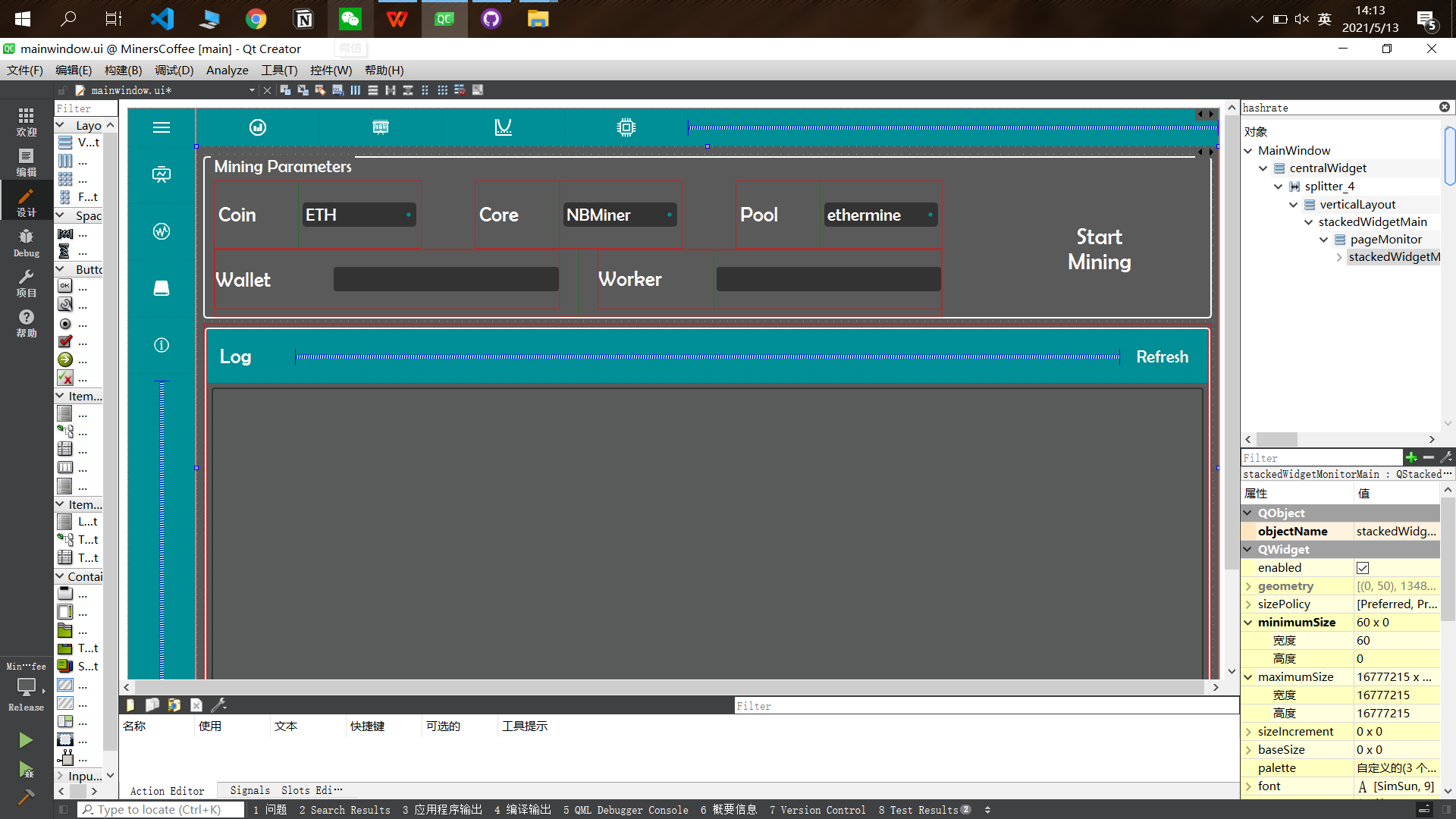


Test task:Check whether the wallet and worker filling and data acquisition are successful.

void **test\_ui\_MiningArgsComboBox**();

void **test\_ui\_MiningArgsComboBox\_data**();

Test component:



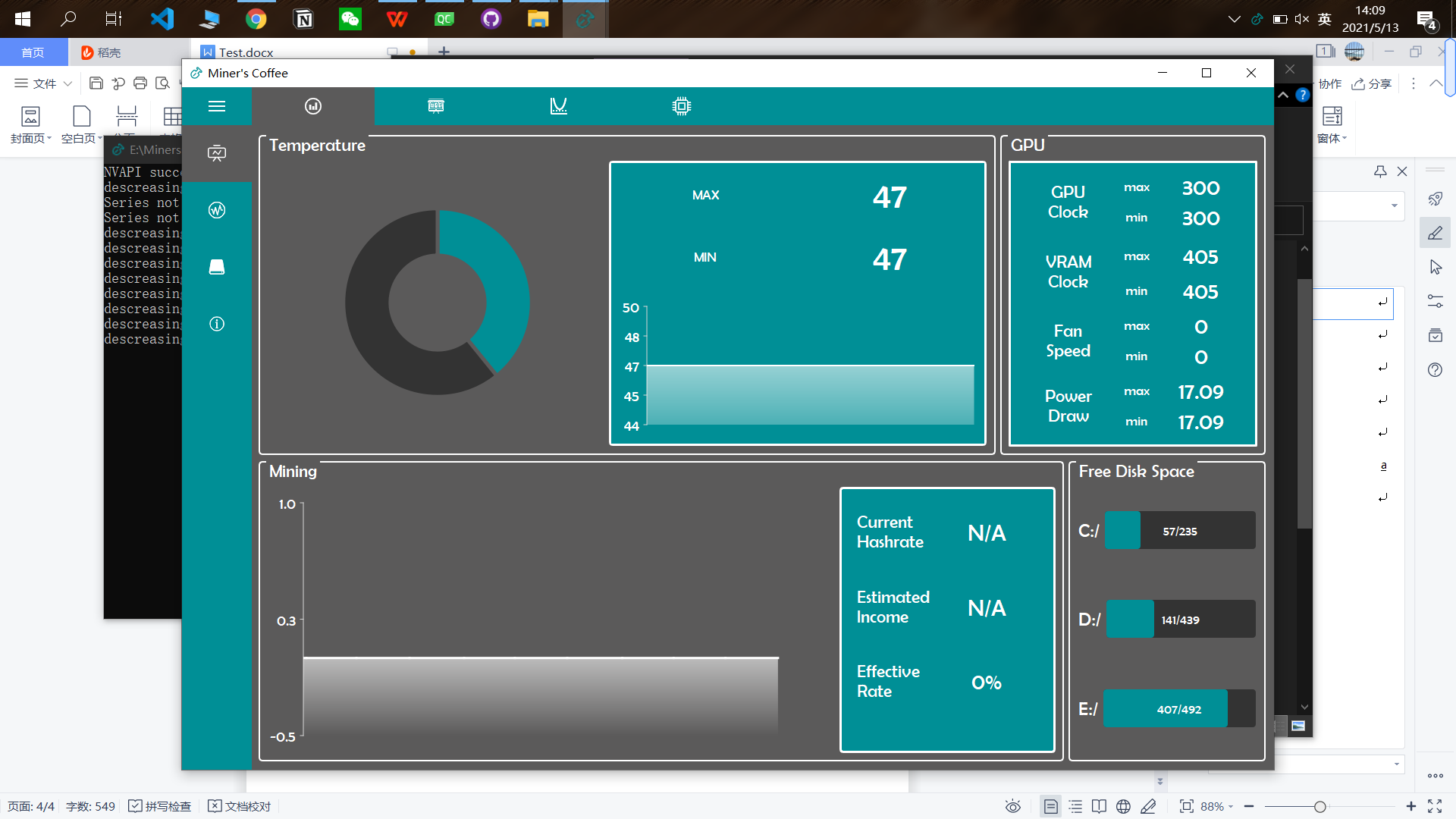
Test task:Check whether the drop-down box can be displayed when clicked, whether the

normal selection can be performed and the data after selection can be returned.

void **test\_ui\_TempPieChart**();

void **test\_ui\_TempPieChart\_data**();

Test component:

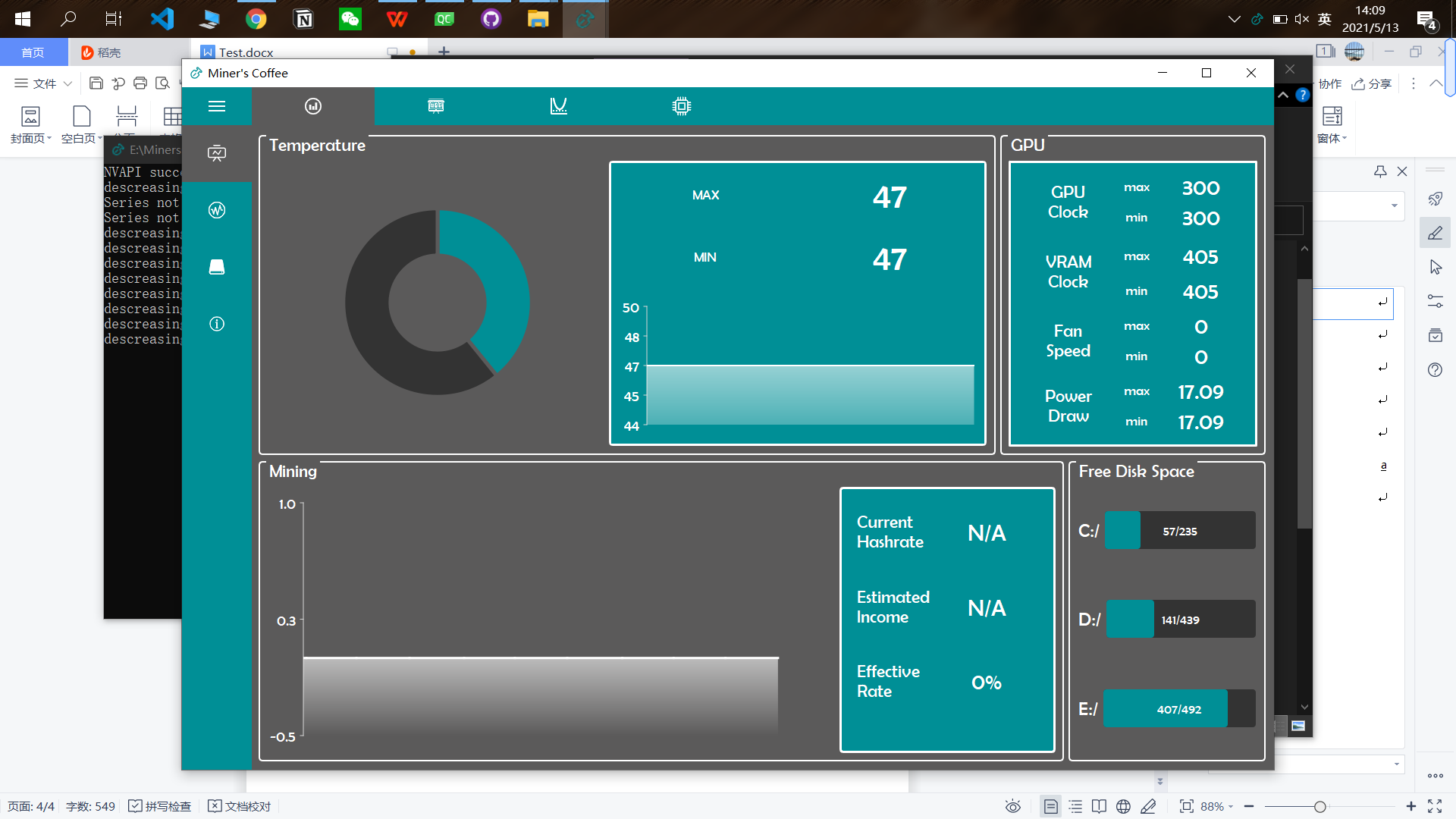


Test task: Check whether the pie-shaped color distribution can change with the change of the temperature value.

void **test\_ui\_HashrateLineChart**();

void **test\_ui\_HashrateLineChart\_data**();

Test component:



Test task: Check whether the line-shaped color distribution can change with the change of the Hashrate value.

### Component/Subsystem Test

void **test\_TempPieChart**();

Test Task: Check if program successfully get the temperature of the GPU and pass it to the right UI interface to display

Test data: Directly passed

Test data size: 1

void **test\_HashrateLineChart**();

Test Task: Check if program successfully get the Hashrate of the mining process and pass it to the right UI component to display

Test data: Directly passed

Test data size: 1

void **test\_MiningCore**();

Test Task: Check whether the user enters the correct wallet and username, and try to connect to the mining pool with the obtained data.

Test data: Directly passed

Test data size: 1

void **test\_MiningArgs**();

Test Task: Check whether the connection can be successfully connected and the mining data is obtained.

Test data: Directly passed

Test data size: 1

void **test\_ParsePoolInfo**();

Test Task: Check whether we successfully process the data and pass in the corresponding function module.

Test data: Directly passed

Test data size: 1

### Full System Test

void **test\_FullSystem**()

Task: View the order between threads for possible blocking situations, view the circulation of the regularization process and check the integrity of the system

### Nonfunctional Test

1. Real-time Performance

To make sure our program running smoothly at users’ computers, we conduct closed beta tests and public beta tests.

* + 1. Closed beta test:

In order to provide a reliable and excellent running performance, we mimic most of the possible situations when running our program in closed beta tests. Our tests are mainly focus on four aspects: number of running program, running mode, computer environments and mining mode. Each kind of test is running for at least three times to make sure result is reliable. Also, most of the running time of testing is more than one hour which is to make sure our program could run persistently and stably.

Test Result:

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Running Mode | Running  Environment | #opening programs1 | Mining Mode | Test Running Time | Program Open Time2 | Program close Time3 | React Time4 | Running Smoothness  (Bad, Normal,  Good) |
| Foreground | Light loaded:  CPU occupied <= 20%  GPU occupied <= 20% | 1 | Mining | 1.5H | <1s | <1s | <1s | Good |
| Not Mining | 1.5H | <1s | <1s | <1s | Good |
| 3 | Mining | 1H | <1s | ~2s | ~1.5s | Normal |
| Not Mining | 1H | <1s | ~2s | <1s | Good |
| Heavy loaded:  CPU occupied >= 20%  GPU occupied >= 20% | 1 | Mining | 1.5H | <1s | <1s | <1s | Good |
| Not Mining | 1.5H | <1s | <1s | <1s | Good |
| 3 | Mining | 1H | <1s | ~4s | ~1.5s | Normal |
| Not Mining | 1H | <1s | ~2s | <1s | Normal |
| Background | Light loaded:  CPU occupied <= 20%  GPU occupied <= 20% | 1 | Mining | 1.5H | <1s | <1s | Not appliable | Good |
| Not Mining | 1.5H | <1s | <1s | Good |
| 3 | Mining | 1H | <1s | ~1s | Good |
| Not Mining | 1H | <1s | ~1s | Good |
| Heavy loaded:  CPU occupied >= 20%  GPU occupied >= 20% | 1 | Mining | 1.5H | <1s | <1s | Good |
| Not Mining | 1.5H | <1s | <1s | Good |
| 3 | Mining | 1H | <1s | ~1s | Good |
| Not Mining | 1H | <1s | ~1s | Good |
| 1: the number of our program opened for testing  2: If multiple of our program is opened, the time means the average time for opening each program  3: If multiple of our program is opened, the time means the average time for closing each program  4: The average response time for our program to response including the AutoOC, Start Mining and Search History | | | | | | | | |

* + 1. Public Beta test:

We totally invited 4 users to use our program for 3 days, all of them give the feedback of good performance, good smoothness and good react time.

Feedback:

|  |  |  |  |
| --- | --- | --- | --- |
| Volunteer No. | Performance Smoothness  (Bad, Normal, Good) | React Time  (Bad, Normal, Good) | Running Smoothness  (Bad, Normal, Good) |
| 1 | Good | Good | Good |
| 2 | Good | Good | Good |
| 3 | Good | Good | Good |
| 4 | Good | Good | Good |

1. Usability for real usage:

We also test all of our program’s functions to make sure our functions work properly. We also conduct closed beta test and public beta test with four volunteers. Our test covers the core functions of our programs including mining, retrieving GPU history, displaying correct GPU information, displaying correct mining information. In the tests, all these functions work well.

Result:

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Public Beta Test | Volunteer No. | Mining Functions Correctly | Retrieving correct GPU history | Displaying correct GPU information | displaying correct mining information |
| 1 | Yes | Yes | Yes | Yes |
| 2 | Yes | Yes | Yes | Yes |
| 3 | Yes | Yes | Yes | Yes |
| 4 | Yes | Yes | Yes | Yes |
| Closed Beta test | | Yes | Yes | Yes | Yes |

1. Simplicity for normal user:

We include the following function or features in our program to make sure our program is user-friendly.

* + 1. Clear Design :
       1. Most of the buttons are marked with their corresponding functions to make sure users can always find their target functions quickly and know what they are doing.

图形用户界面

描述已自动生成

**Figure 2**: Clear Design in Overclocking Page

* + - 1. We separate main functions into several different pages and only provide necessary buttons to make sure our program looks [concise](javascript:;) and easy to use.

图形用户界面, 应用程序

描述已自动生成

**Figure 3**: Separate Functions into 8 Pages

* + 1. User manual and tutorials:

We provide clear and detailed user manuals and quick tutorial videos (the demo videos) to help users understand all functions of our program.

* + 1. Remember users’ input and settings:

To help users get rid of troubles of remembering last input settings we save some necessary user inputs in the MinersCoffee.ini automatically which will make sure all the settings will remain the most recent one when users open the program.

图形用户界面, 文本, 应用程序

描述已自动生成

**Figure 4**: Auto Saved User Inputs in MinersCoffee.ini

1. Interoperability to user:

We also include some features to make our program interact with users.

* + 1. Buttons showing status:

Buttons including the “Start Mining” and “AutoOC” will display their status by changing the text on the buttons after user click them.

图形用户界面

低可信度描述已自动生成 图形用户界面

中度可信度描述已自动生成

**Figure 5**: StartMining Button Before Clicked (left) and After Clicked (right)

* + 1. Auto Overclocking:

After user clicked AutoOC, the program will change the overclocking settings for the GPU automatically. Also, it will move the sliders of the settings to visualize the change of the overclocking settings. User can also interact with these sliders and adjust these settings.

图形用户界面

描述已自动生成

**Figure 6**: Settings Before AutoOC

图形用户界面

描述已自动生成

**Figure 7**: Settings Before After Clicking AutoOC

* + 1. Auto Fan Speed:

To prevent user from changing the fan speed after choosing the “Auto speed fan” mode, we hide the slider of the fan speed after user choose this mode. In this way, our program interact with users to show that fan speed is controlling automatically.

图形用户界面, 文本

描述已自动生成

**Figure 8**: Fan Speed Slider in Normal Mode

图形用户界面

描述已自动生成

**Figure 9**: Fan Speed Slider in “Auto speed fan” Mode

* + 1. Rapidly response to feedback from users:

During developing, we invited some users to use our programs and give feedback. We always respond quickly to the users’ needs and make modification.

图形用户界面, 应用程序

描述已自动生成

**Figure 10**: Quick Response to User Feedbacks