

Miner's Coffee

**A Powerful and Accessible
GPU Mining Software**

TEST DOCUMENT



Content

Purpose.....	3
Environment.....	3
Testing Framework.....	3
Test Implementation.....	5
Test Initialization.....	5
Test Utilities.....	5
Unit Test.....	5
Component/Subsystem Test.....	8
Full System Test.....	8
Nonfunctional Test (incl. User Experience).....	9

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.

✓	PASS	Executing test case GeneralTest	tst_generaltest.h 28
✓	PASS	Executing test function initTestCase	tst_generaltest.cpp 14
	DEBUG	NVAPI success 0	tst_generaltest.cpp 14
	DEBUG	testing 100	tst_generaltest.cpp 14
	DEBUG	testing 0	tst_generaltest.cpp 14
	WARN	Series not in the chart. Please addSeries to chart first.	tst_generaltest.cpp 14
	DEBUG	===== 0	tst_generaltest.cpp 14
	DEBUG	===== 10 405	tst_generaltest.cpp 14
	WARN	Series not in the chart. Please addSeries to chart first.	tst_generaltest.cpp 14
	PASS	GeneralTest::initTestCase	tst_generaltest.cpp 14
>	PASS	Executing test function test_ParseJsonForMining	tst_generaltest.cpp 674
>	PASS	Executing test function test_ParseJsonForPool	tst_generaltest.cpp 618
✓	PASS	Executing test function test_GetURLInternal	tst_generaltest.cpp 606
	PASS	GeneralTest::test_GetURLInternal	tst_generaltest.cpp 606
>	PASS	Executing test function test_ui_MiningArgsLineEdit	tst_generaltest.cpp 478
>	PASS	Executing test function test_ui_MiningArgsComboBox	tst_generaltest.cpp 532
>	PASS	Executing test function test_ui_TempPieChart	tst_generaltest.cpp 445
>	PASS	Executing test function test_ui_HashrateLineChart	tst_generaltest.cpp 410
✓	PASS	Executing test function test_Cmd	tst_generaltest.cpp 309
	PASS	GeneralTest::test_Cmd	tst_generaltest.cpp 309
>	PASS	Executing test function test_NvidiaapiSignleunite	tst_generaltest.cpp 322
✓	PASS	Executing test function test_NvidiaapiComplex	tst_generaltest.cpp 374
	DEBUG	NVAPI success 0	tst_generaltest.cpp 374
	DEBUG	NVAPI NvAPI_GPU_GetThermalSettings error -101	tst_generaltest.cpp 374
	DEBUG	NVAPI NvAPI_GPU_ControlThermalSettings error -101	tst_generaltest.cpp 374
	PASS	GeneralTest::test_NvidiaapiComplex	tst_generaltest.cpp 374
>	PASS	Executing test function test_Database_getAdvice	tst_generaltest.cpp 720
		Entering test function GeneralTest::test_Database_getAdvice	

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
```

(2) 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

(3) Network Module

void test_GetURLInternal();

Task: Check if program successfully connect to sparkpool

Test data: Directly passed

Test data size: 1

(4) 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";

(5) 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

(6) 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:

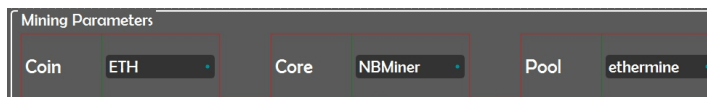


Test task: Check whether the wallet and worker filling and data acquisition are successful.

```
void test_ui_MiningArgsComboBox();
```

```
void test_ui_MiningArgsComboBox_data();
```

Test component:

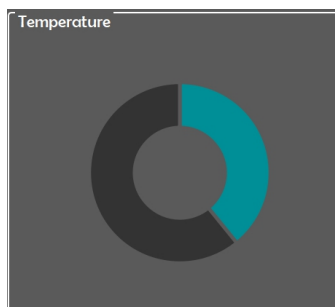


Test task: Check whether the drop-down box can be displayed when clicked, whether the normal selection can be performed and the data after selection can be returned.

```
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 (incl. User Experience)

a) Real-time Performance

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

i. 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 programs ¹	Mining Mode	Test Running Time	Program Open Time ²	Program close Time ³	React Time ⁴	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	1	Mining	1.5H	<1s	<1s	<1s	Good
			Not Mining	1.5H	<1s	<1s	<1s	Good

	occupied >= 20% GPU occupied >= 20%	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 applicable	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

ii. 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:

Volunt eer 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

b) 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

c) Simplicity for normal user:

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

i. 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.

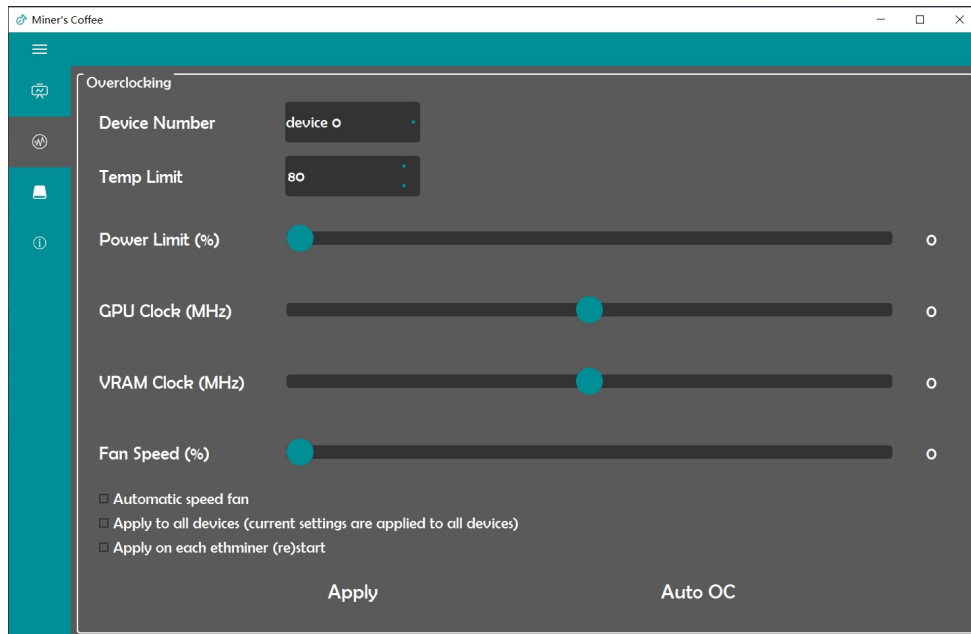


Figure 2: Clear Design in Overclocking Page

2. We separate main functions into several different pages and only provide necessary buttons to make sure our program looks concise and easy to use.

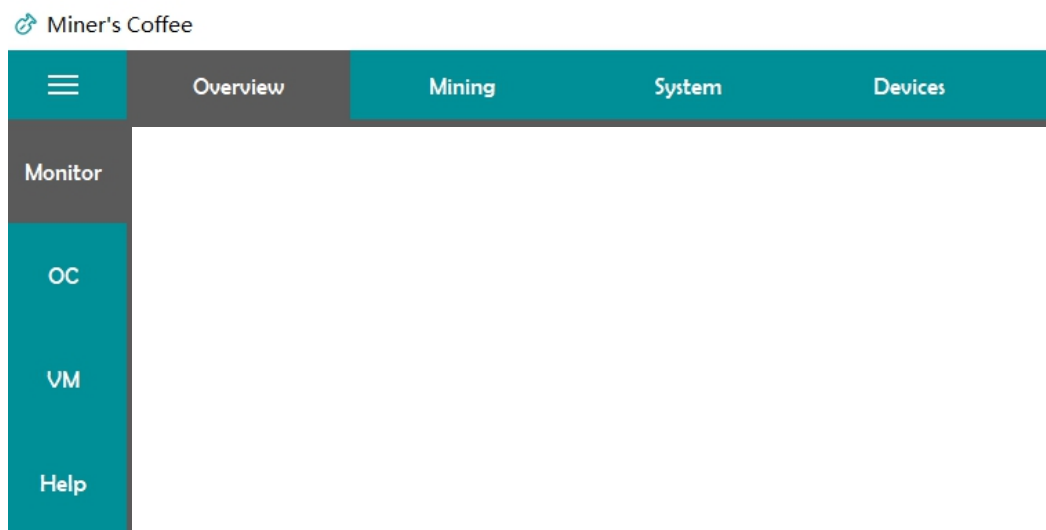


Figure 3: Separate Functions into 8 Pages

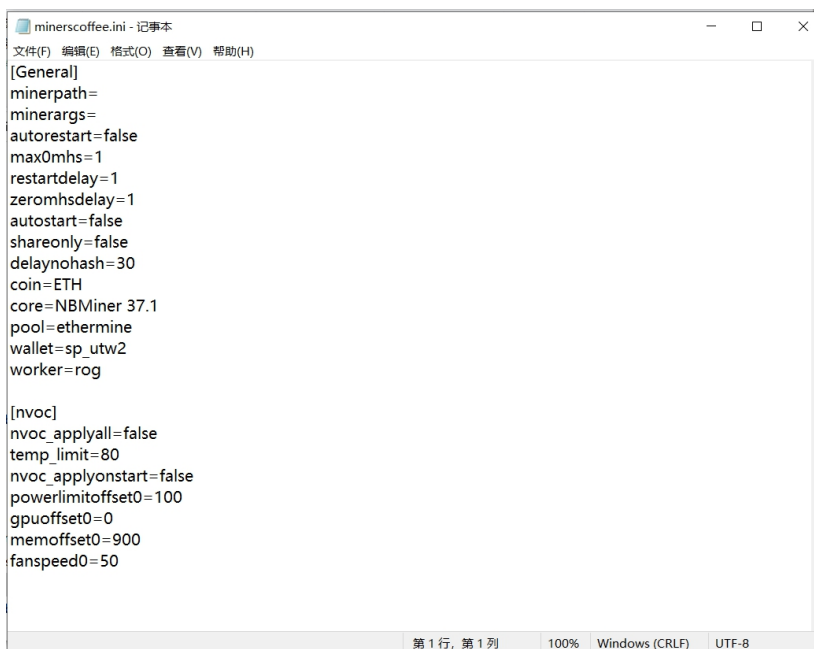
- ii. User manual and tutorials:

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

of our program.

iii. 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.



```
minerscoffee.ini - 记事本
文件(F)  编辑(E)  格式(O)  查看(V)  帮助(H)

[General]
minerpath=
minerargs=
autorestart=false
max0mhs=1
restartdelay=1
zeromhsdelay=1
autostart=false
shareonly=false
delaynohash=30
coin=ETH
core=NBMiner 37.1
pool=ethermine
wallet=sp_utw2
worker=rog

[nvoc]
nvoc_applyall=false
temp_limit=80
nvoc_applyonstart=false
powerlimitoffset0=100
gpuoffset0=0
memoffset0=900
fanspeed0=50

第 1 行, 第 1 列    100%    Windows (CRLF)    UTF-8
```

Figure 4: Auto Saved User Inputs in MinersCoffee.ini

d) Interoperability to user:

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

i. 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.



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

ii. 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.



Figure 6: Settings Before AutoOC

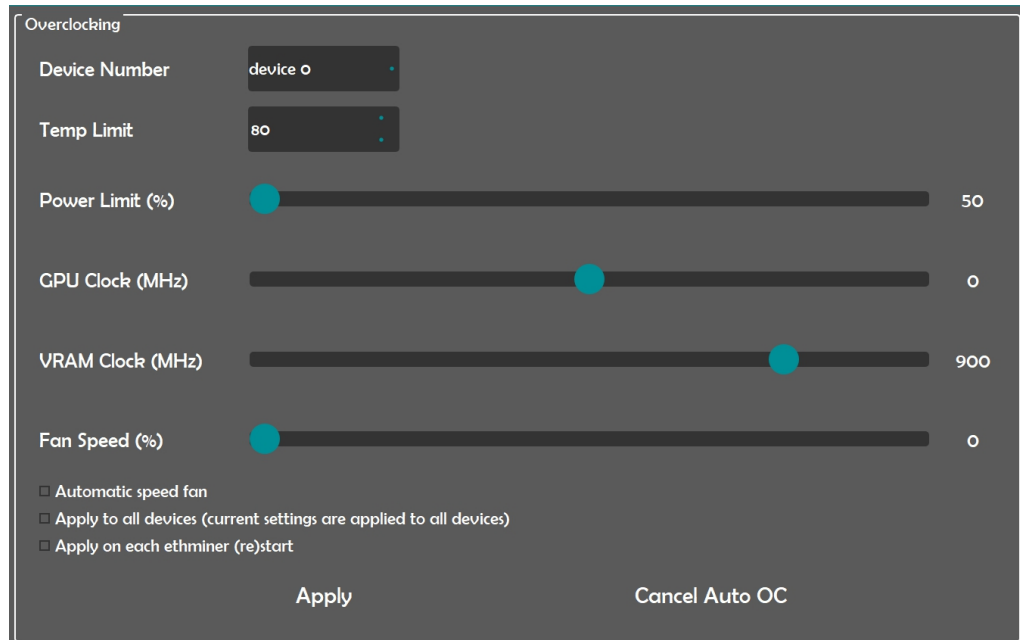


Figure 7: Settings Before After Clicking AutoOC

iii. 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.

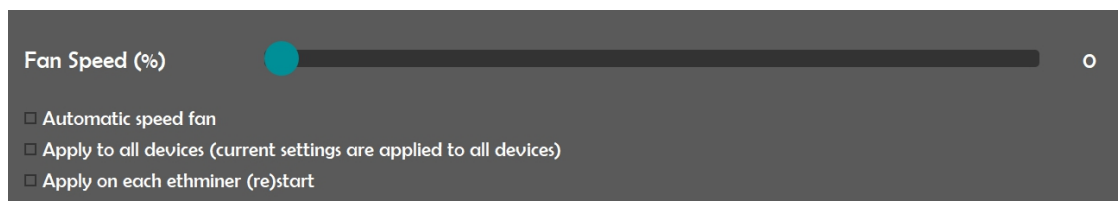


Figure 8: Fan Speed Slider in Normal Mode

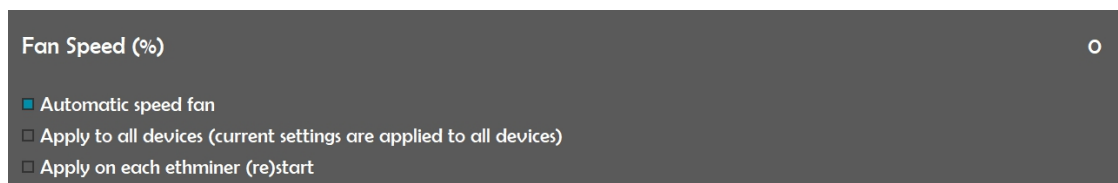


Figure 9: Fan Speed Slider in “Auto speed fan” Mode

iv. 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.

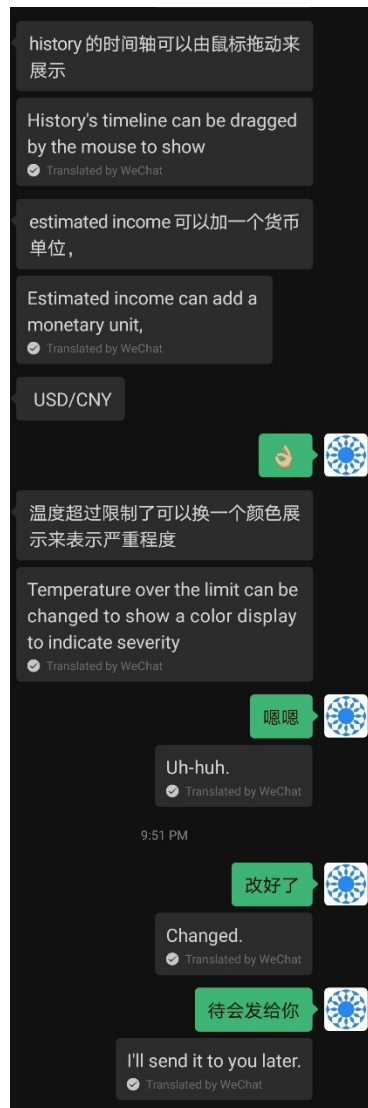


Figure 10: Quick Response to User Feedbacks