Game Optimization Toolkit User Guide

UWA GOT v 1.2.0

Youhu Technologies was founded on the purpose of exploring innovation of technical consulting in game industry. The first product, UWA, is kind of SaaS for developers to analyze and optimize potential performance issues. UWA can help developers to find performance bottlenecks in projects at runtime. Meanwhile, UWA provides solutions on how to solve problems and how to improve fluency App runs. So far, there are more than 5000 evaluation reports in the past 16 months in UWA website.

In order to make the performance analysis more convenient, we develop the **Game Optimization Toolkit** (referred to as: **UWA GOT**). Developers can test Apps' performance using real mobile devices and read performance report on local host with UWA GOT. The highlight of UWA GOT is that users can set up a local server in Unity editor to receive and analyze performance data for optimization. Here are the key features as follow:

- **Easy to integrate**

5 steps without any code modification.

- **Customized Testing**

It is free to control start/end point, testing mode and the specific process. Do not root your mobile devices necessarily.

- **Efficient Feedback**

After each testing, it takes only a few minutes to transfer data to the local LAN server and process the data before viewing the report.

- **In-depth Data Collection**

In-depth data are collected, including details of various resources, top time-consuming functions, specific stack trace and object type in Mono heap etc.

- **Local Analysis Server**

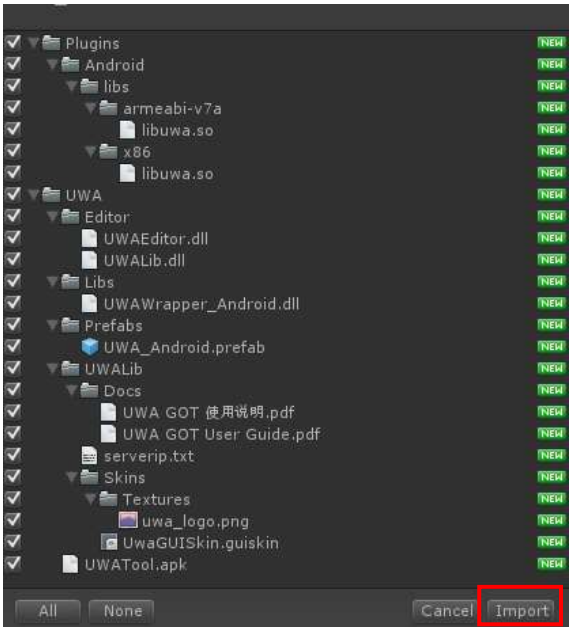
All the test data are analyzed and established within the local server, which can be set up in Unity Editor directly.

- **Professional Statistics**

Base on big data of game performance, we organize the analysis in a way which is easier to identify the performance bottleneck objectively.

GOT SDK Integration

1. Drag the package file into the Unity project. Click “Import” button, when the window displays as follow.



2. Drag the prefab file “UWA Android.prefab” into the hierarchy window to add this Gameobject to the first scene, as shown in the following figure.

3. Run your project in the editor. If the “UWA Mode Select” UI appears on the upper right corner of the Game view window without any error message, it means UWA plugin has been integrated successfully.

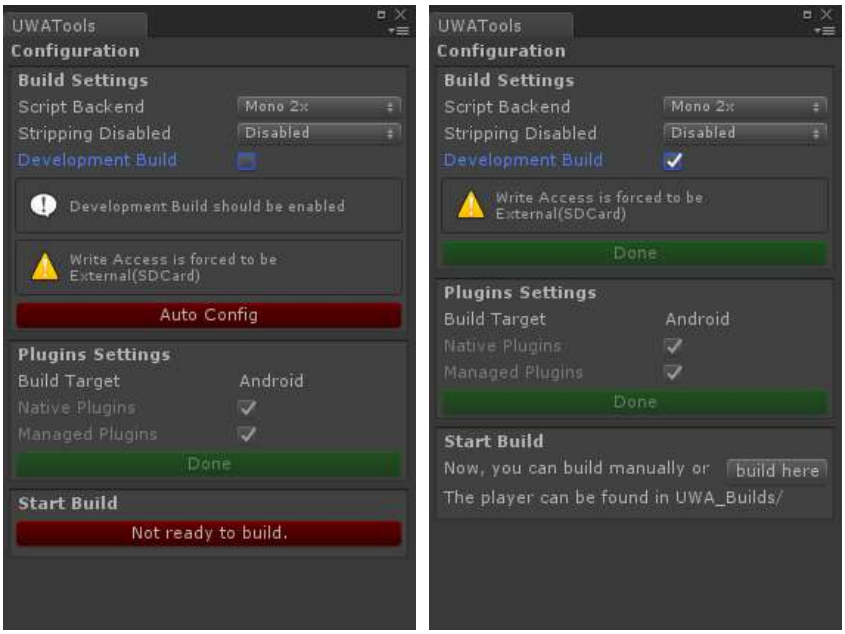
4. Click the menu “Tools->UWA->SDK Integration”, open UWA toolbar.

5. Apply either of the following method to setup the configuration:
    - Set “Stripping Disabled” to “Disabled”, and meanwhile enable “Development Build”

- One-click “Auto Config”

In addition, please make sure the “Native Plugins” and the “Managed Plugins” are enabled, also the “Write Access” is on “External(SDCard)” status.

The button will turn green and show “Done” (as shown in the figure at the bottom right) when the configuration is correct.



6. Click “build here” to release your games. The Apk file will be saved under the directory “UWA Builds/Android”. You can also release your games manually through “Build Settings->Build” alternatively.

Notice:

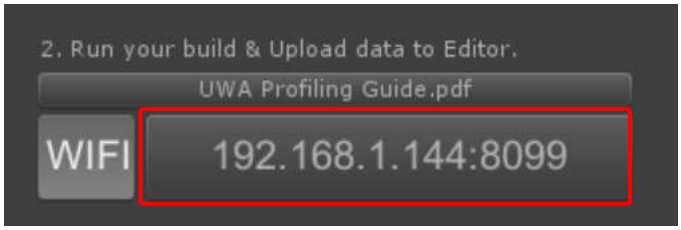
1. This tool only supports **Android** platform at this moment. The root permission is not required.
2. Make sure that “**Stripping Level**” is on “**Disabled**” before publishing.
3. It is advisable to click “**build here**” to publish your games for the **Development Build**. Remember to add BuildOptions.Development when using BuildPlayer to publish your game.
4. Make sure that the “**Write Access**” is on **External(SDCard)**.
5. This tool does not support projects built with **il2cpp** so far.
6. Screenshots tracking is only supported on **Android 5.0** or above.

UWA Local Server

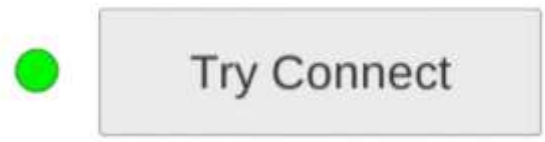
1. When the target app is built, then install it on the Android device.
2. Open the app, a GUI panel can be found on the upper right corner. Click “Local Mode” and then you can select one of the modes for your purpose. If you want the test to be started immediately after launching the app, you can press “Direct Mode” (the button left will become green). And then the app will quit automatically after you select one of the modes, and the test will be started immediately after the next launch.

3. The “Stop” panel is draggable, tap the “stop” when you want to stop this test. UWA SDK will store the data on the device.

4. Click the menu “Tools->UWA->Local Server” in the editor menu bar to open the UWA Local Server. Click the “WIFI” button to turn it on, and you will see the IP address as follow with the port fixed at 8099.

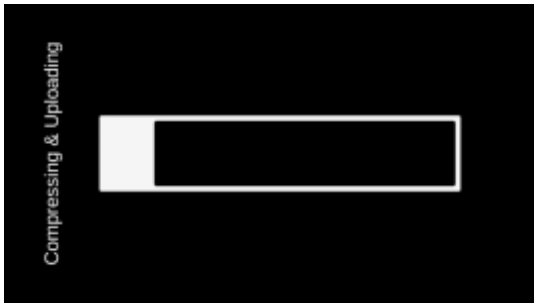


5. Install the UWATool.apk on your device, it can be found at UWA/UWATool.apk in Unity project.
   This app is used to upload the data from device to Local Server. After installation, open UWATool and input the IP address of UWA local server. Then tap “Try Connect” button. The circle on the left will be green if the Local Server can be reached, otherwise, please make sure the device and Unity Editor are in the same LAN.



6. Click “Upload Data”, all the test data will be listed. Choose one of them and click “submit”,

you will see the uploading window. Click “delete”, the data will be deleted and removed from the list. When progress becomes 100% and the uploading window is closed, the uploading is done.



Notice：

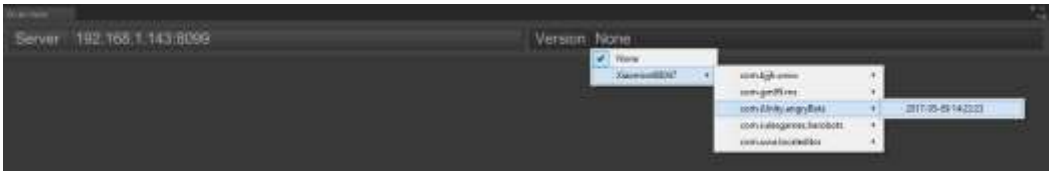
1. “Deep (Online)” refers to our online performance analysis service. If you want to perform a more detailed performance analysis, please also submit your App to the UWA website (www.uwa4d.com) for evaluation.
2. The local test file (APK file) is consistent with the online one, so you do not need to release it again. After uploading, we will provide detailed performance analysis report within 48 hours. Here is a demo of online performance analysis report.
3. We are sorry that the online report is in Chinese at this moment, English version is coming soon!

Performance Overview

Select the “Overview” mode, you will find the overall performance of your application through the whole runtime.

1. CPU Usage

(1) Select the test data which ones you want under “Version” drop-down list. Data are grouped by device/application/test date.



(2) The corresponding data will then be analyzed and displayed in the panel, which consists of several areas:

a) CPU Cost Chart

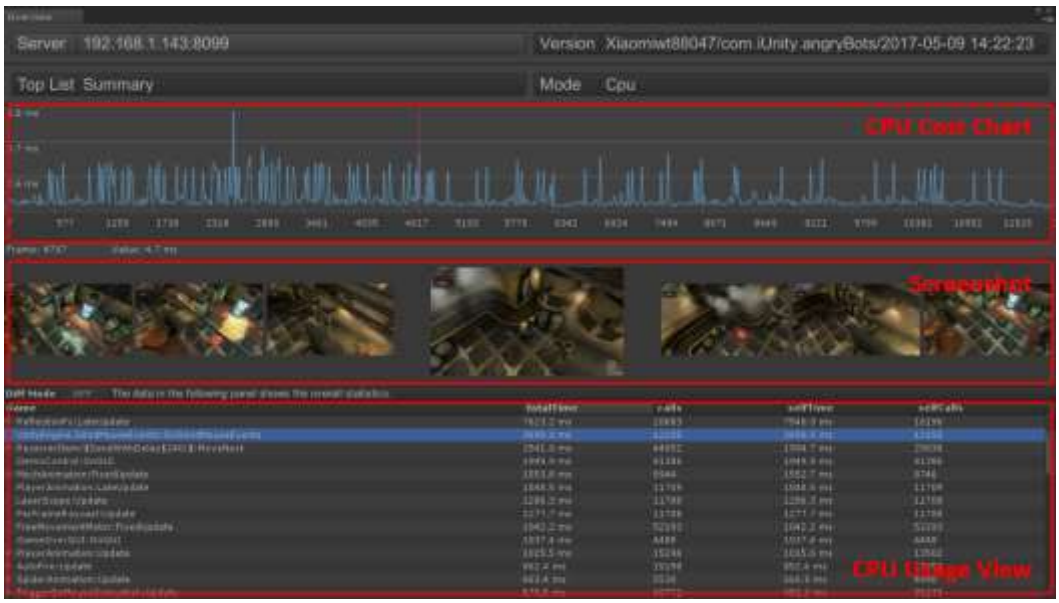
You may select either one of function and find the CPU cost chart over the runtime.

b) Screenshot

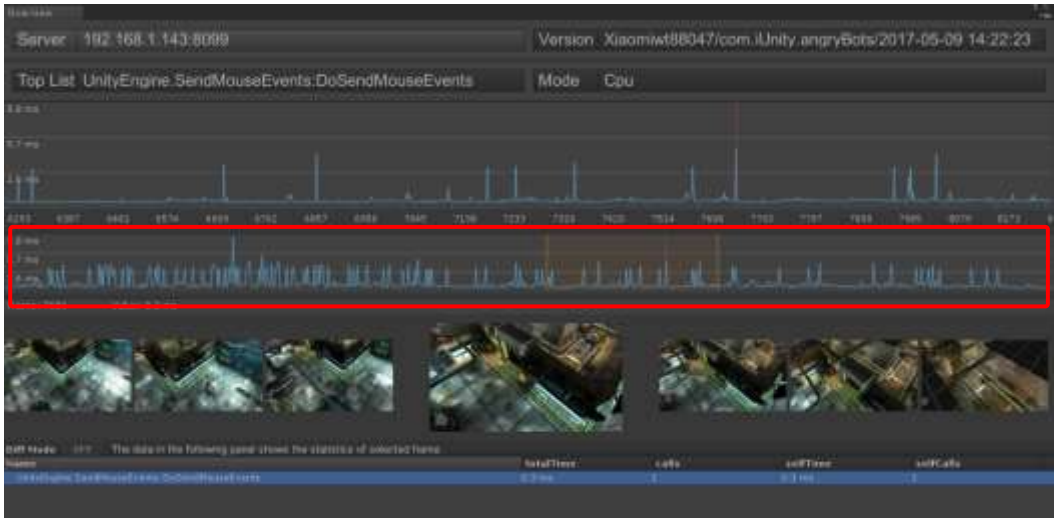
Selecting one frame, the corresponding screenshot will be displayed here.

c) CPU Usage View

This area displays the CPU usage of the top 10 or 20 time-consuming functions in a hierarchical way. You could profile any pieces of code with UWA API, please see appendix 1 for details.



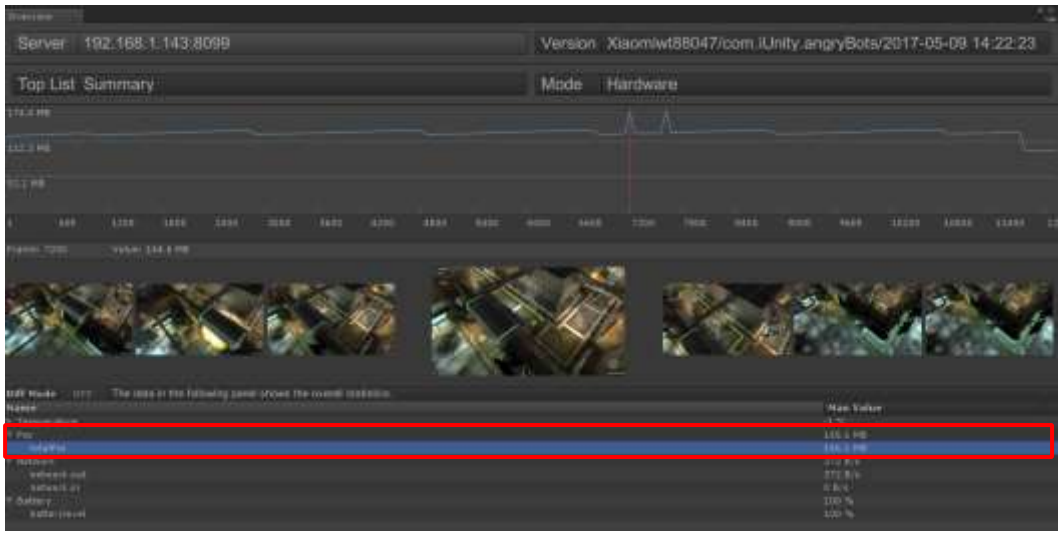
(3) In this panel, you can switch to “Summary” mode by selecting “Top List->Summary” to review the CPU usage of all functions (user code) over the whole time.
Alternatively you can select one specific function to see performance during runtime. You can also adjust the slider to change ROI to see more details in the chart.



2. Hardware information

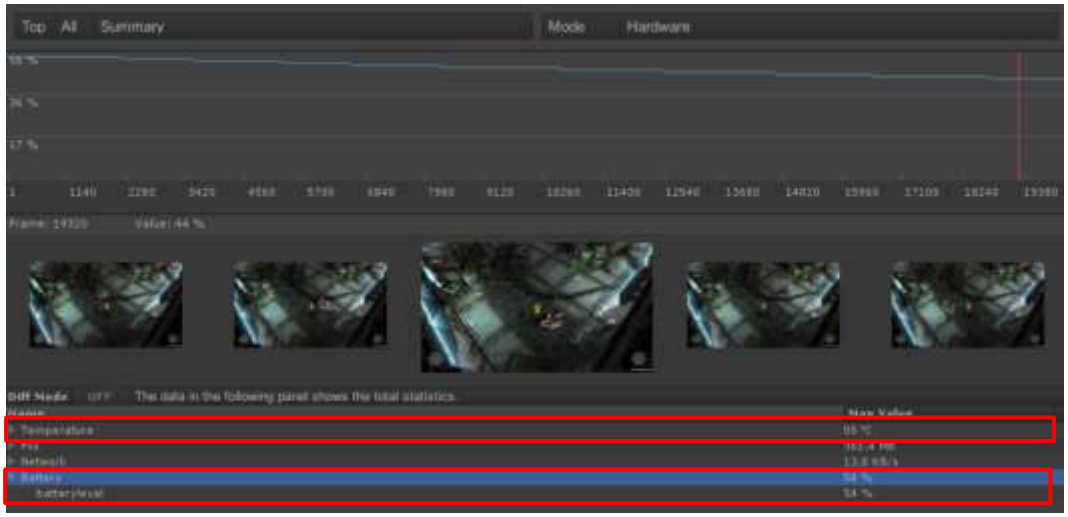
Select “Mode-> Hardware” to switch the mode to “Hardware” where you can inspect the following hardware information of your application during this period.

(1) Device memory.
The PSS information over the whole runtime can be displayed on Android devices ONLY.



(2) Network traffic statistics.
Network traffic flow of the devices can be provided. The download and upload data flow are represented as Network.in and Network.out respectively.

(3) Battery and temperature information.

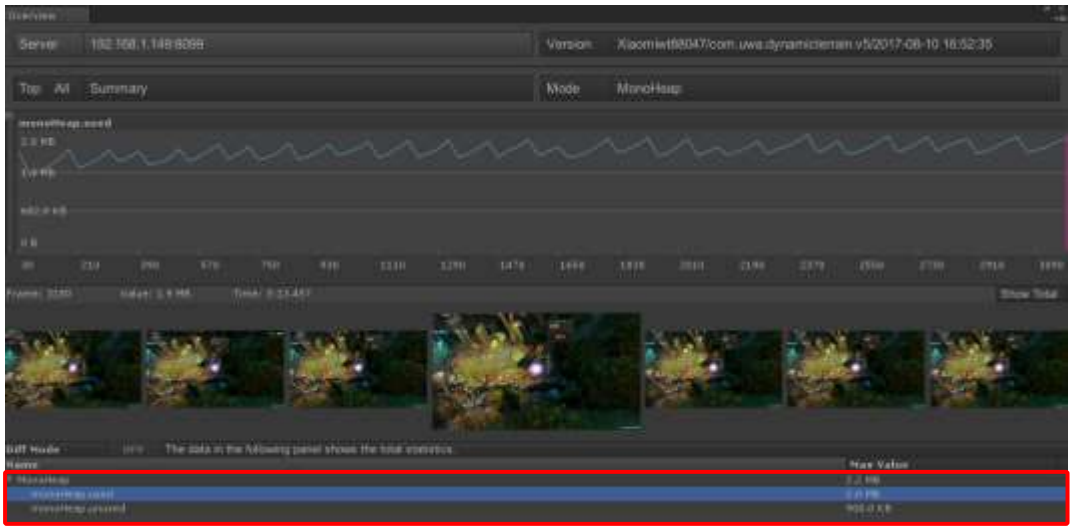


3. FPS information

Select “Mode-> FPS” to switch the mode to “FPS” where you can inspect the following FPS information of your application during this period.

4. Mono Heap information

Select “Mode-> MonoHeap” to switch the mode to “MonoHeap” where you can inspect the following size of mono heap of your application during this period, including the used and unused parts.

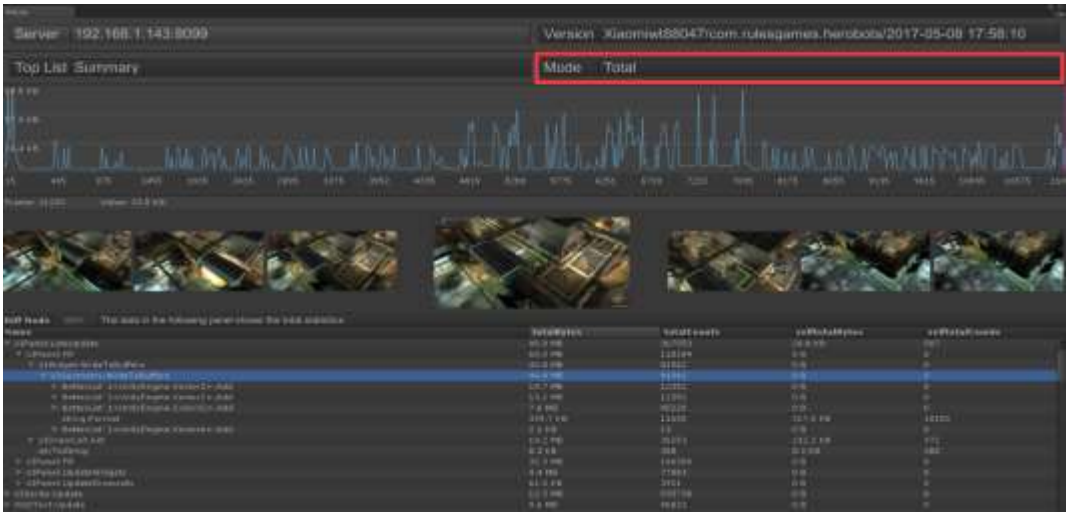


Mono Memory Analysis

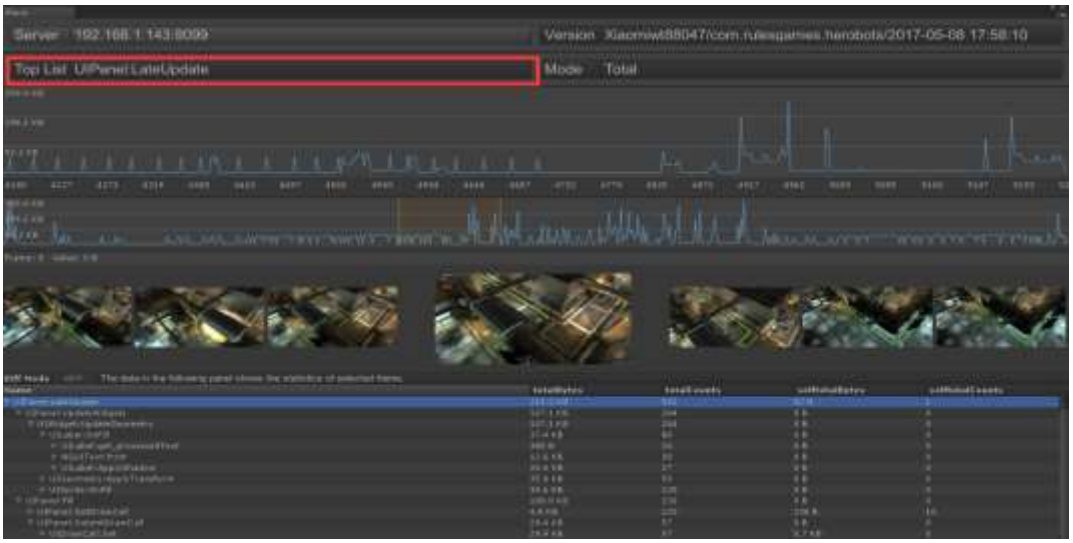
Click the “Mono” button, you may see the Mono heap memory allocation statistics at runtime.

1. Allocated Heap Memory

(1) Select “Total” in the “Mode” drop down menu, you may see the heap memory allocation of each function at runtime.



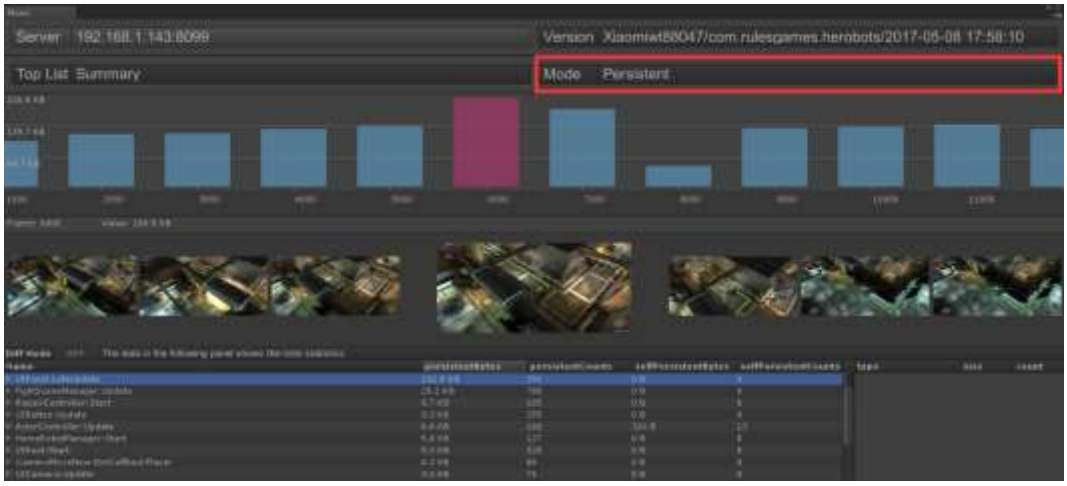
(2) Select the specific function name in the “Top List” drop down menu, you may see its heap memory allocation at runtime. In addition, you can select a single frame in the chart, the heap memory allocation in the frame shows.



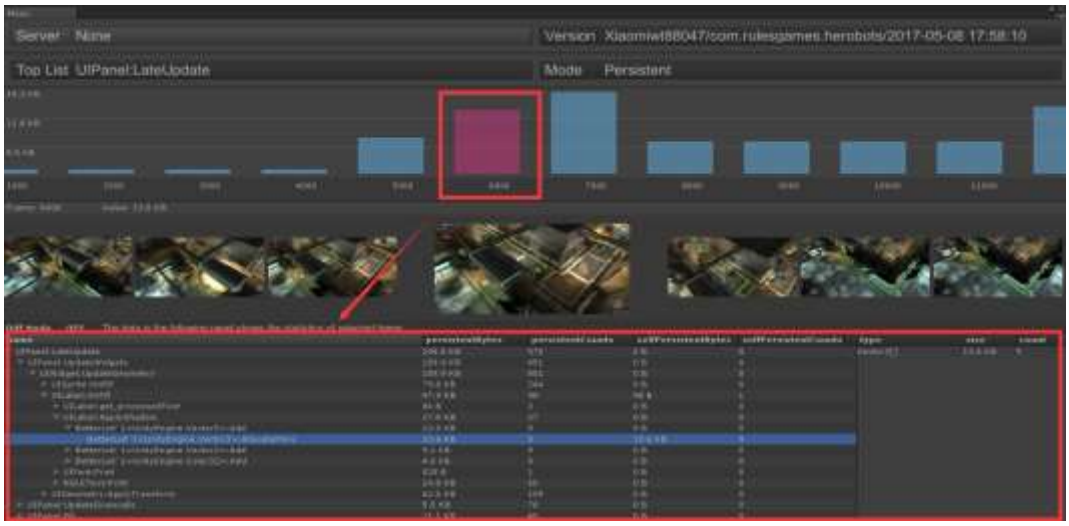
2. Memory Leak

(1) Select “Persistent” in the “Mode” drop down menu, you may see the memory usage of each

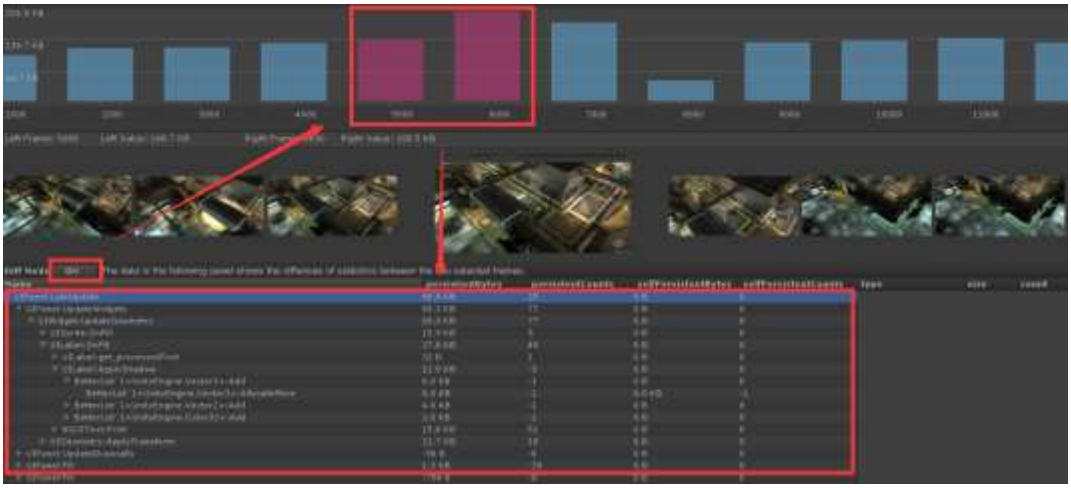
function at runtime. UWA captures the Mono heap memory usage of each function every 1K frame by default. The statistics are shown as histogram.



(2) Select the specific function name in “Top List”the drop down menu, you may see the heap memory allocation of the specific function. In addition, when “selfPersistentCounts” is not 0, you can click the function name and see variable type which hold heap memory in the function.



(3) In “Persistent” mode, you can also check the difference between two heap memory statistics by switching “Diff Mode” to “ON”. The difference between the two histograms are shown after you select them in the chart.



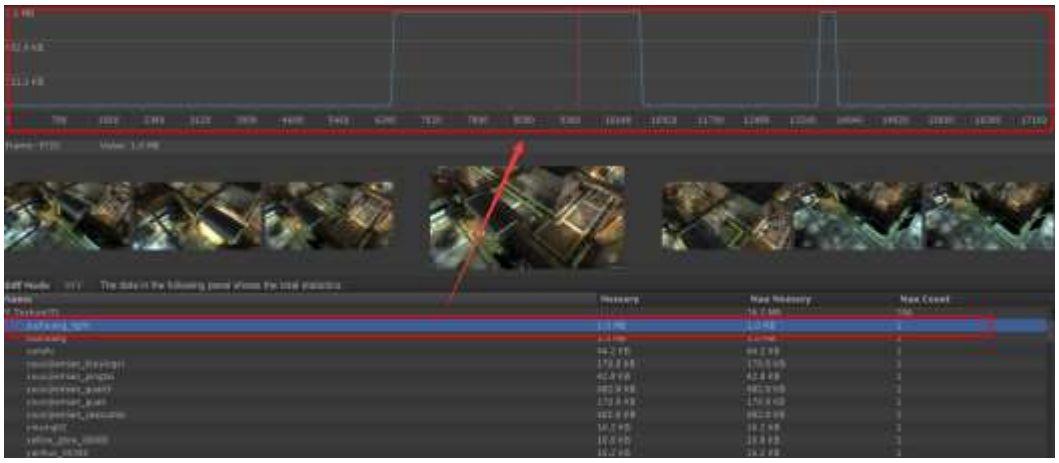
Runtime Asset Tracking

Click the “Assets” button, you may inspect the assets information of your application over the whole runtime.

1. Overview

(1) For each asset type, you can track the total memory usage of all assets over time.

(2) Click any asset to track its lifecycle during testing.

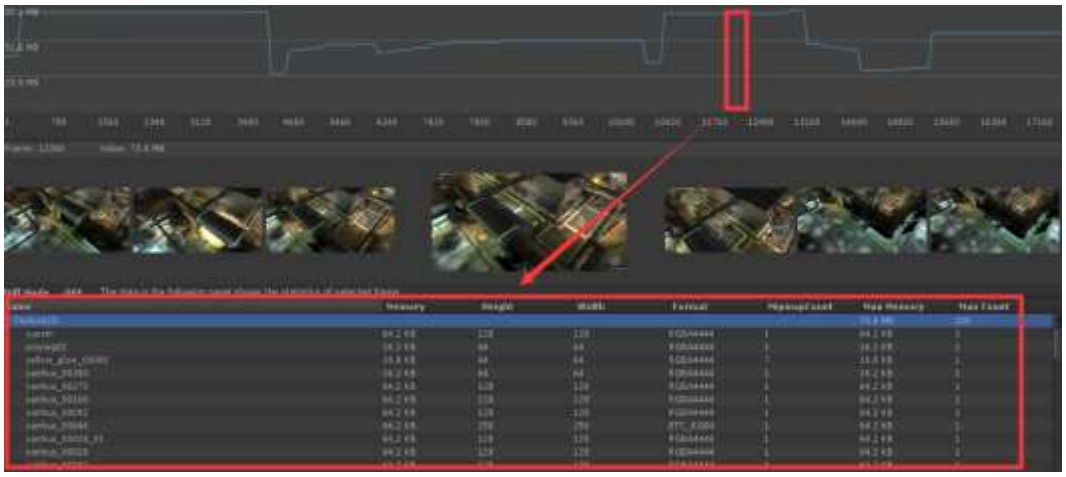


2. Assets usage per frame

(1) Select an asset type in the “TopList” dropdown menu.

(2) Detailed asset information will be shown while you click one frame in the memory usage

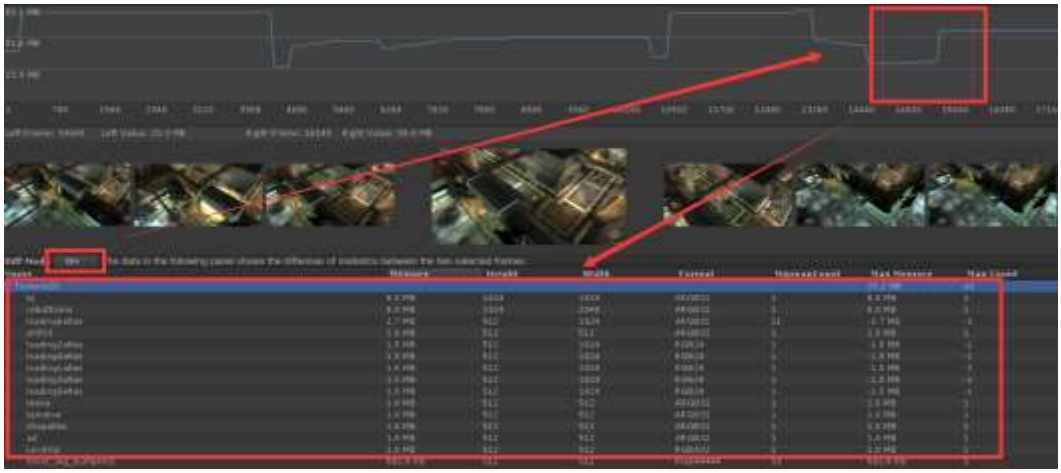
chart.



3. Memory Leak Analysis

You can check whether there is asset leak with comparing the assets difference between any two frames.

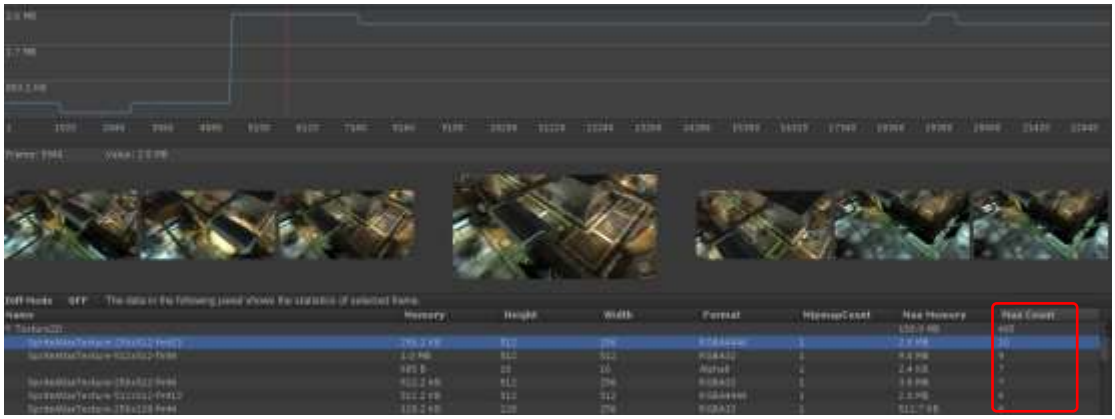
(1) To make a comparison, **switch "Diff Mode" to ON**, then select any two frames, the difference will be shown in the below list.



In the above example, we compared the "Texture" assets between frame #16140 & # 14640. The positive value of an asset in "Max Memory" column indicates that the qty of that asset is increased in the frame #16140. And the negative "Max Memory" value means the qty is decreased in the frame #16140.

4. Asset Redundancy Analysis

The assets are probably redundant during the runtime. We advise to check the asset list that whether the "Max Count" value of the asset is larger than 1. "Max Count" value is the maximum quantity of the asset used in one frame. If the count is more than 1, there is the high possibility of asset redundancy.



Note: It's possible that two different assets have the same name, same memory size and other properties. So we advise to double check whether it is reasonable while "Max Count" value is more than 1.

Appendix 1: Introduction of UWA API

UWAEngine.StaticInit
public static void StaticInit();

This api can be used to initialize the UWA SDK, instead of draging the UWA_Android.prefab into your scene.

UWAEngine.PushSample/PopSample
public static void PushSample(string sampleName);
public static void PopSample();

This api can be used to profile custom pieces of code, which makes it simple to find the bottleneck in the script.

sampleName is used as a custom label. UWAEngine will record the time cost between the *PushSample* and *PopSample*, and you will see the statistics with the given label in the Local Server. Here is a simple usage example.

```
UWAEngine.PushSample("MyCode");
// some code …
UWAEngine.PopSample();
```

The statistics with custom labels is displayed as follows.

Name	percent	selfPercent	totalTime	calls	selfTime	selfCalls
Perf:Update	100.00 %	0.04 %	90.5 ms	3	0.0 ms	1
A	54.40 %	0.03 %	49.2 ms	11	0.0 ms	1
B	54.37 %	0.27 %	49.2 ms	110	0.3 ms	10
C	54.09 %	2.98 %	49.0 ms	1100	2.7 ms	100
D	51.11 %	30.92 %	46.3 ms	11000	28.0 ms	1000
E	20.19 %	20.19 %	18.3 ms	10000	18.3 ms	10000

Please make sure that the *PushSample* and *PopSample* are called in pairs. Any *return* or *yield return* between them will make the result confusing.
On the other hand, please be aware that, do not call *PushSample* and *PopSample* too many times in one single frame. Based on the preliminary statistics on some low level devices, 10000 calls makes about 50ms overhead.

UWAEngine.Start
public static void Start(Mode mode)

This api can be used to start the test with the given mode, instead of pressing the button in GUI panel.

UWAEngine.Stop
public static void Stop()

This api can be used to stop the test, instead of pressing the button in GUI panel.

PS:

1. Now, UWA API are all compiled with [Conditional("ENABLE_PROFILER")]. So that the UWA API cost nothing in **Non-Development** Build.
2. **UWAEngine.Start/Stop** can be called only once in a run of the game.