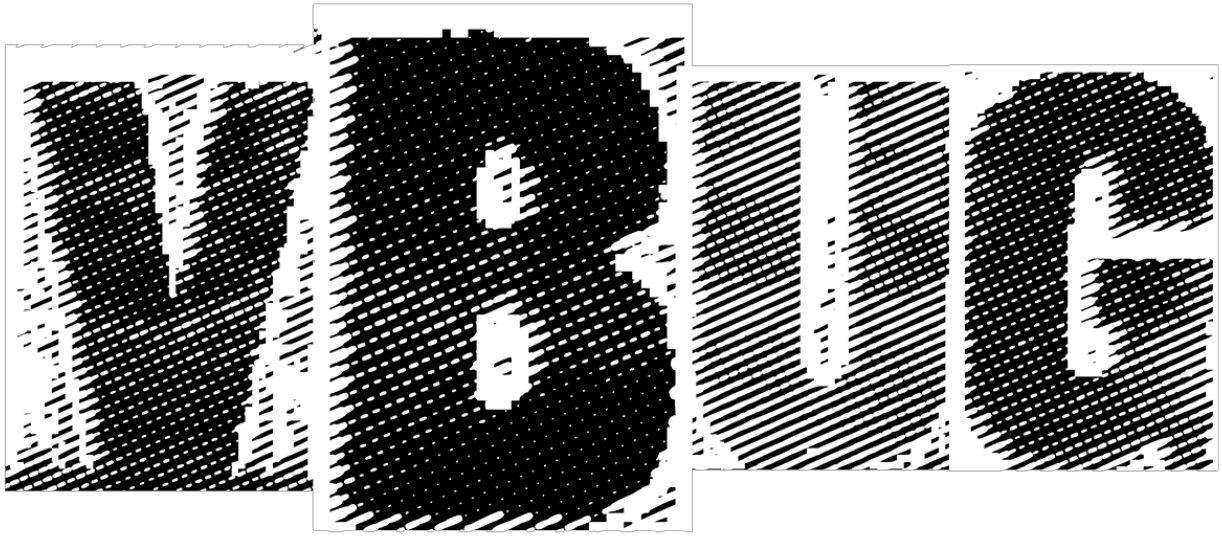


Table of content

1. Preface

1.1. About vBug

My name is Michiel Frankfort and I'm a senior gameplay-programmer and designer at *Little Chicken Game Company* in Amsterdam - The Netherlands. During my many-years experience with Unity3D, I learned a great deal about the inner workings of the Unity3D game-engine.

While I was working on one of Little Chickens biggest projects, I learned that debugging and especially QA-testing can yield test results that are not always very useful or simply lack viable information. Thats why I started vBug, a tool that should give clear insight in player-behaviour but also helps the developer reproduce and fix problems.

Thanks to my lovely wife, who supported me every step of the way, I was able to find the time and focus to create this project. Without her, I could not have pulled it of.

I also would like to thank my colleagues for their feedback and enthusiasm, especially Eamon Woortman whom soon will tribute to this great project by implementing 'Stream over WIFI' functionality.

Thanks Unity3D for creating such an awesome engine! I loved every moment working with it.

2. How to setup your project and scenes?

2.1. Start recording

In order to make vBug record your game-play sessions, you can either start it from code or use the 'vBug Initializer' prefab. The prefab contains all settings you need to run vBug and it will start the recorder once its awakes.
Simply add it to the scene and hit play.

If you prefer to record only certain segments of the game, use the following code to start and stop recording manually:

- vBug.StartRecording();
- vBug.StopRecording();

Every time the recorder starts, a new session is created. You will find more info about the way sessions are structured and saved at 'vBug Repository window' page.

2.2. Project & Scene setup

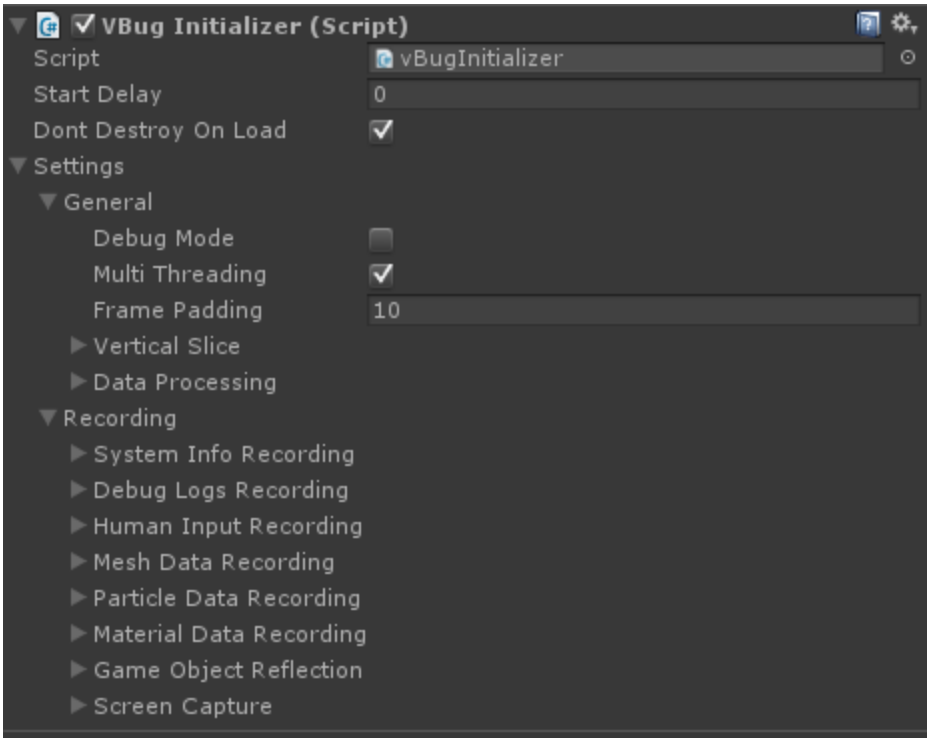
Mobile:

When you are preparing your project for mobile testing, make sure the first time DEBUG-Mode is enabled, because vBug will prune all of its own messages if disabled. You can enable this by setting 'debugMode' to 'true' over at the 'General' vBug-settings.

On Android, its important to set the 'Write Access' to 'External (SDCard)' over at the Android player settings.

General:

By default, all settings are tweaked to make a project the size of 'Angry Bots' run on mobile (S3, Nexus 5, iPhone 4s, iPad 3rd gen). Therefore this page can be SKIPPED if you don't want to tweak or alter any settings.



The general settings can be changed and accessed from code using:

- vBug.settings.general;

The recording settings can be changed and accessed from code using:

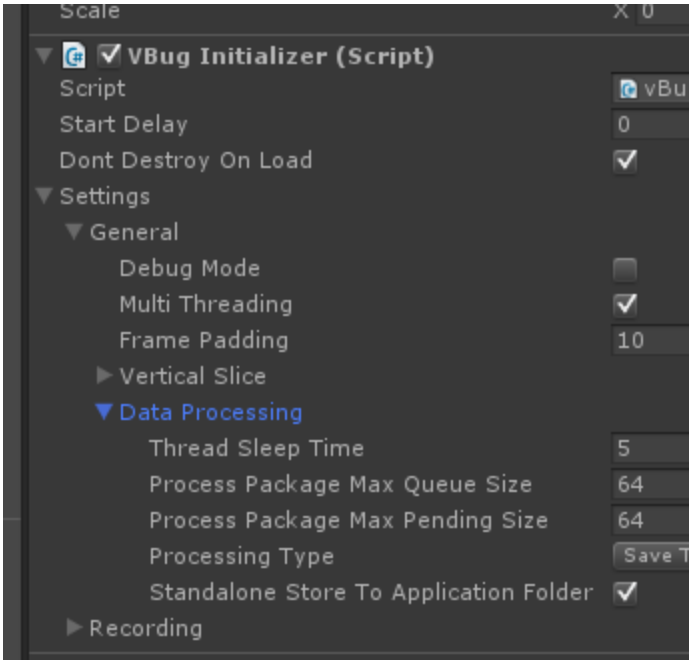
- vBug.settings.recording;

If you are looking for in-depth info about settings, please read the 'vBug Recording-settings guide', but here are some general tips per Recorder-type.

Storage:

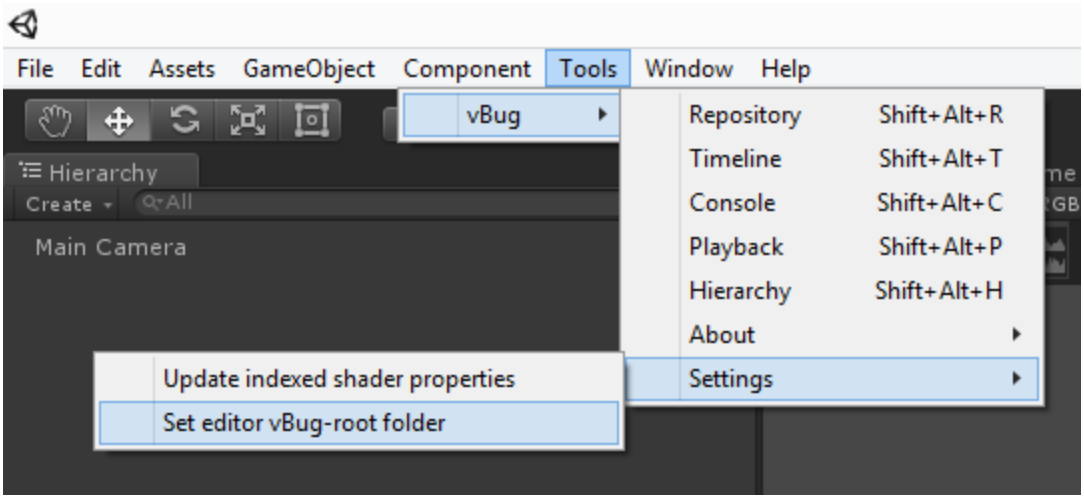
Windows, Mac & Linux standalone:

By default, all the vBug data will be stored to the same folder the standalone-application is running. If you prefer the always-safe 'persistentDataPath' instead, disable the 'StandaloneStoreToApplicationFolder' boolean over at the General/DataProcessing settings:



Windows & Mac editor:

You can change to location where vBug will store its data by setting the root-folder from the settings menu. By default the 'persistentDataPath' is used, just like any other platform.



System-info

This recorder is enabled by default and is well-tweaked out of the box (skip for quick setup) System info is divided into two parts: Frames-per-second info & Memory-footprint info.By default the 'System-info' recorder is enabled and its wise to keep it that way to make most use of the 'vBug Timeline Window'.

Capturing memory info is optional and disabled by default. You can turn it on by enabling 'autoEnableProfiler', but this is an Unity-PRO feature only! Enabling this setting will cost steep additional performance.

Debug-logs

This recorder is enabled by default and is well-tweaked out of the box (skip for quick setup) All debug-logs are captured, including exceptions, errors and warnings. If you want to exclude one of your Debug.Log calls from the capture-process, please use vBug.Log instead.

Multi-camera capture (Screen-capture settings)

This recorder is enabled by default and is well-tweaked out of the box (skip for quick setup)

If you want to change the output image-size, you can alter 'maxScreenCaptureSize'. Please note that doubling the resolution takes 4X the performance! The default 'maxScreenCaptureSize', 'captureInterval' and other settings where tweaked to run on mobile with acceptable performance spikes/costs.

There are two ways of capturing the screen. There used to be more options, but after testing all of them intensively, it turned out these two are most useful for different reasons.

- jitRender (just-in-time render)

  Re-renders the entire image again directly to the right format. Does not require Graphics.Blit and allows you to re-render everything at lower quality.
  Pro's
  - Its very fast at low render-resolutions
  - It allows multiple active camera's to be rendered at the same time

  Con's
  - PRO-licence only.
  - It does not include Unity's OnGUI renders.

- endOfFrame

  Reads all the pixels directly to a texture once the End-of-frame is reached.
  Pro's
  - Includes OnGUI renders!
  - No PRO-licence required

  Con's
  - Slow!
  - Does not allow multiple camera's

Device-input (HumanInput-capture settings)

This recorder is enabled by default and is well-tweaked out of the box (skip for quick setup) In order to capture any device-input, all you have to do is make sure the 'HumanInput' recorder is enabled. Its lightweight and does not take up a lot of space, therefore its best to keep it enabled.

There is a catch however. If you want the mouse & touch input recording to work, make sure Screen-capture is enabled as well, since it needs an screen-shot as a base to work with.

The following data-providers are currently supported:

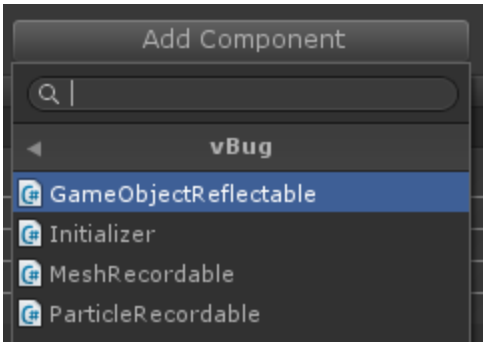
- Mouse
- Touch
- Keyboard

Hierarchy & Components (GameObjectReflection settings)

This is one of the most complex and powerful tools vBug has to offer. It enabled the developer to review the entire scene-hierarchy frame-by-frame including exposed fields from components attached to GameObjects.

Its relatively easy to setup. If you want to record the hierarchy only, just set 'enabled' to 'true' over at the 'GameObjectReflectionSettings'.

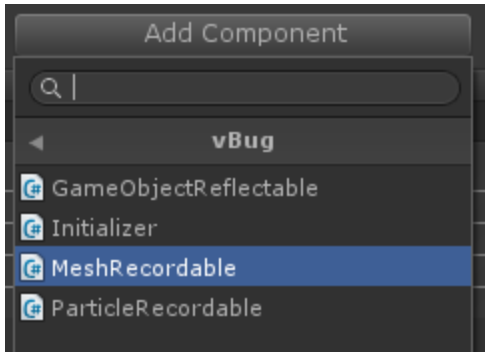
If you want to capture component information as well, all you have to do is add the 'vBugGameObjectReflectable' to an GameObject in the scene and all its components will be reflected recursively.



For more in-depth setting options, please read the 'GameObjectReflectionSettings' chapter over at the 'vBug Recording-settings guide' page.

Mesh-data (advanced setup)

Capturing mesh-data is easy to setup and use, but it does have its limitations. You can simply capture a mesh by attaching 'vBugMeshRecordable' component to its game-object.



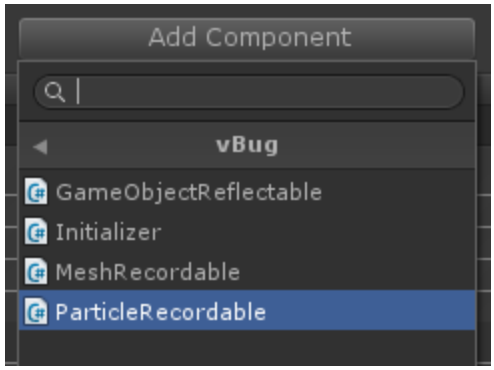
For starters, there are 2 ways of capturing 3 types of animated meshes:

- smartMesh (lightweight and greatly compressed)
  - Animated MeshFilters (position, rotation, scale)
  - Animated SkinnedMeshRenderers (bones driven)
- fullMesh (very heavy and huge memory footprint)
  - Fully deformed / altered mesh-data (procedural generated / distorted)

All three of them are supported, but you need the right setting for the right type. If done properly, lots of meshed can be captured, but keep it to a minimum to prevent clutter and extensive memory usage.

Particle-data (advanced setup)

Just like capturing mesh-data, all you have to do is attach 'vBugParticleRecordable' component to the game-object containing all particle systems/emitters.



Both 'Shuriken' (ParticleSystem) and 'Legacy' (ParticleEmitter) types are supported.

3. Running your game

3.1. Performance

Quality:

Capturing the screen, system-info, tons of mesh-data & particles, the complete hierarchy etc is not an easy job. Therefore I have tried to make most processes as lightweight and multi-threaded as possible.

In order to make it all work, we have to reduce quality of different aspects during capturing to make it work within reasonable boundaries. This means that not everything is captured as-is, but rather a smart-representation of the actual thing.

Screen-capture:

The screen-capture is one of the most costly, jet useful things to capture. That's why by default the quality has been reduced to 'acceptable' in most cases, in order to make it run properly on most modern devices.

If you want to change settings and experiment with it, please read the 'ScreenCaptureSettings' chapter at the 'Recording specific settings' page.

Mesh-capture:

During runtime, all normals and tangents are skipped and during playback all normals and tangents are calculated instead. This saves a lot of performance and disk-space during capturing, but requires extra performance during playback.

SkinnedMeshRenderers can be captured with a set amount of bone-influences. This greatly saves capture performance, but might generate some artefacts during playback. This setting can be altered if you like for better looking results.

Hierarchy & Component reflection:

It just didn't make sense to capture static objects, so by default these are skipped, saving performance and disk-space.

Quantity:

Capturing lots of objects (mesh-data, particle-data, component reflection, etc) is very costly. Especially when it comes down to storage-space and internal memory usage. Its always best to setup a scene in such a way that you will only record what's needed.

Besides the performance cost during capturing, it also influences the performance during playback as well. If the UnityEditor needs to visualize tons of captured meshes, particles or component, it will clearly slow down editor performance.

3.2. Tips and Tricks

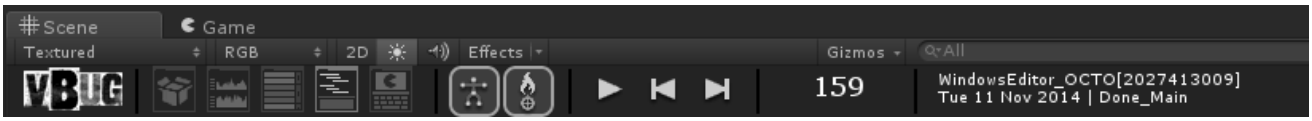
- Do not capture every frame:
  By default most recorders are set to an target-interval of 3, meaning it will capture every 3 frames. This will create less accurate results, but saves a lot of performance and disk-space.

- Less is more!
  Capturing everything just doesn't make sense, so choose your targets wisely. This will save capture performance, disk-space, internal memory usage and overall playback performance.

- Disable what you don't need:
  If you are not interested in looking at the hierarchy for example, just disable this before capturing.

4. How to playback the recorded data?

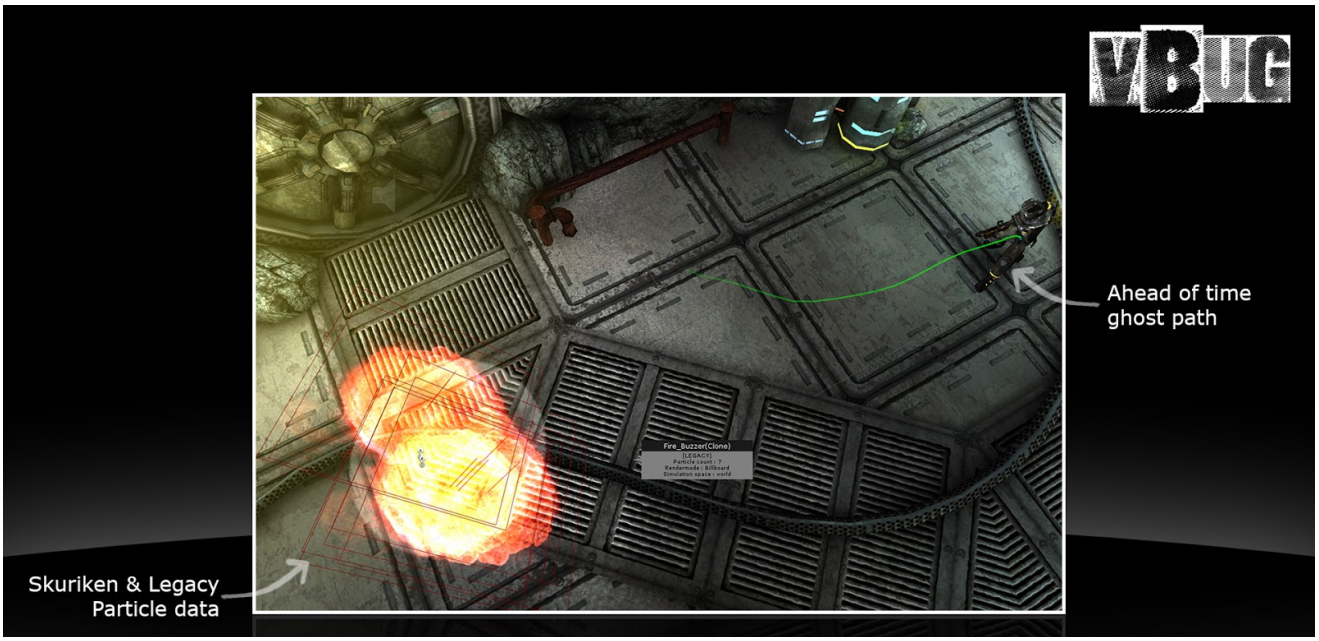
4.1. vBug Sceneview navigation toolbar



This toolbar only appears when a vBug window is opened and will be mounted to the most prominent scene-view available, making it context-sensitive.
The first five icons are hotlinks to the vBug windows. From left to right: Repository, Timeline, Console, Hierarchy & Playback.

The next two icons are toggles:

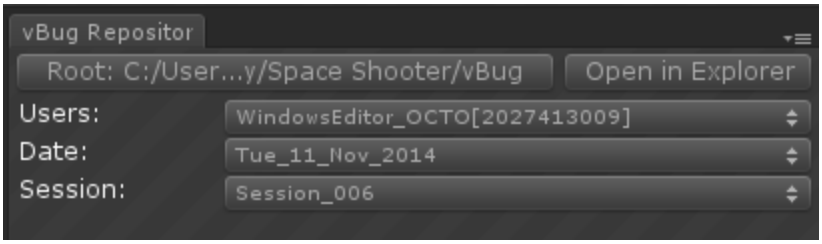
- Mesh-data triple toggle: First one shows everything: mesh-data & ghost path. The second one only shows the mesh-data and the third one disables everything.
- Particle-data triple toggle: First one shows the particles fully rendered with a mouse-hover wire-frame overlay. The second one only shows the wire-frames and the third one disables everything.



Next to the toggles we find a 'Play/Stop' toggle plus 'previous frame' and 'next frame' buttons. This allows for easy playback and accurate frame stepping.

Next to the playback buttons, the current frame-number is shown. This corresponds to the frame-indicator in the 'vbug Timeline window' and updates when scrubbing the timeline or double-clicking logs in the 'vBug Console window'.

4.2. vBug Repository window



Once you are done recording a gameplay-session, this is the place to start. For starters, you need to select the root-folder (by default called 'vBug').

Once a correct root-folder is selected, the 'user-folder', and 'date-folder' will automatically be set to the most recent available and it wall automatically start loading the most recent session found.

Unity Editor:

If you're capturing data while running the game in editor-mode, pressing the 'Open vBug savedata rootfolder' button will automatically take you to the root-folder last used.

Mobile:

Obtaining save-data from mobile devices is a bit more tricky. That's why we are currently working on a new feature: Streaming over WIFI! But for the time being, you need to obtain the data manually.

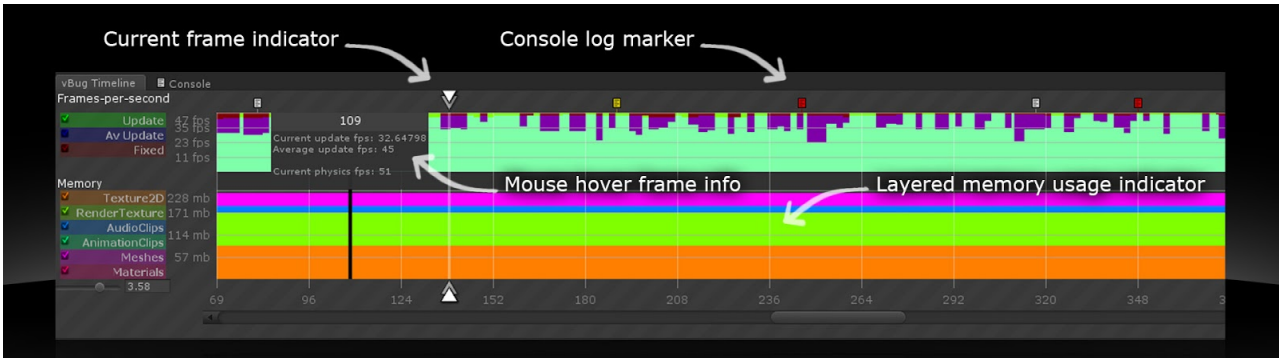
- Android: Just hookup your phone or tablet to a pc or mac (make sure the USB-drives will detect and mount the devices), and navigate to "[your device]/Android/data/[com.yourcompname.yourappname]/files" and copy the entire vBug-root folder to MAC or PC.

  Before capturing, please remember to set the 'Write Access' to 'External (SDCard)' over at the Android player settings.

- IOS:
  Install a tool on your PC or MAC that allow you to access files remotely and copy-paste the vBug-root folder from the 'Documents' to your PC or MAC.

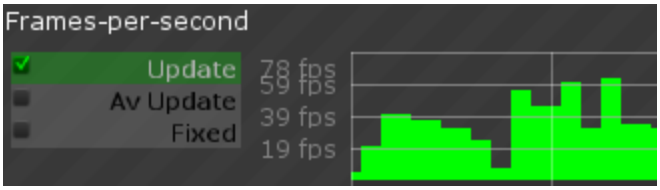
  Personally I prefer 'IFunBox', a free application for MAC & PC:
  http://www.i-funbox.com/

4.3. vBug Timeline window



This window allows you to navigate over time, and displays the following items captured by the *'SystemInfo' recorder:*

- Frames-per-second graph

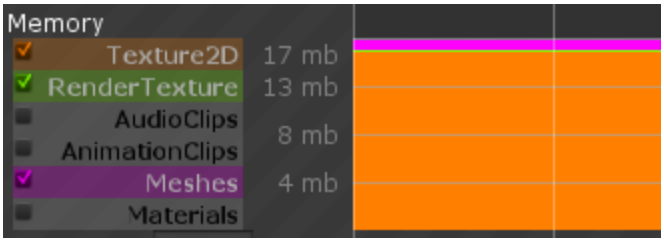


You can hover over the three individual toggles on the left to highlight each one, or just toggle the overlay by tapping it.

There are three overlays available:

- 'Update' - Current frame-rate.
- 'AV Update' - Average frame-rate over the past second.
- 'Fixed' - The Physics-update frame-rate, usually very steady, but might fluctuate during heavy operations.
- You can hover over the the graph to get additional FPS-info per frame.

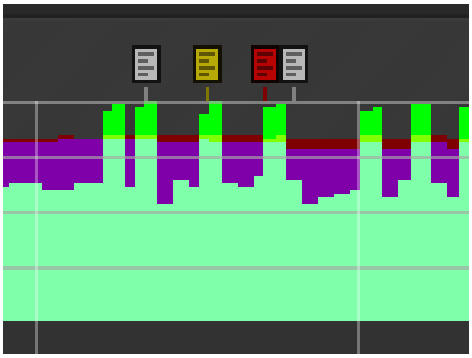
- Memory footprint graph



You can hover over the six individual toggles on the left to highlight each one, or just toggle the overlay by tapping it. Unlike the FPS-graph, these values stack up to to a total amount of memory consumed.

These different colors allows for better comparison, but you can always disable those whom are invalid for your purposes.
You can hover over the the graph to get additional Memory-info per frame.

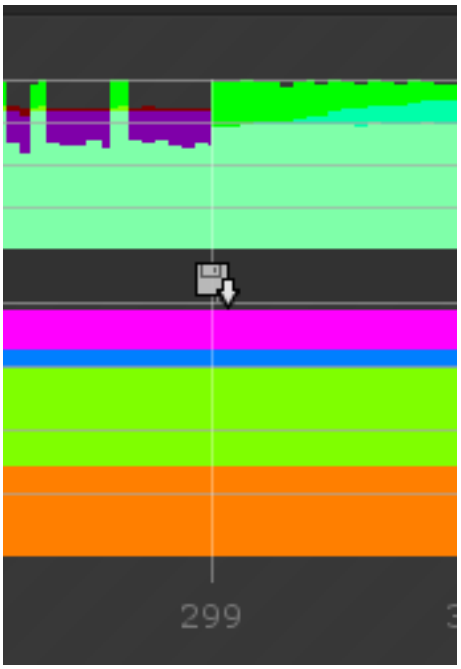
Console-logs indicators:



On the top of the Timeline you will find small 'Console'-icons, indicating a Log was fired that frame. There are three possible icons:

- White: A normal Debug.Log()
- Yellow: A Debug.LogWarning()
- Red: A Debug.LogError() or exception.

Load-scene indicators:



In the middle of the Timeline (to prevent it from cluttering) the frame a new scene was loaded, a small icon appears.

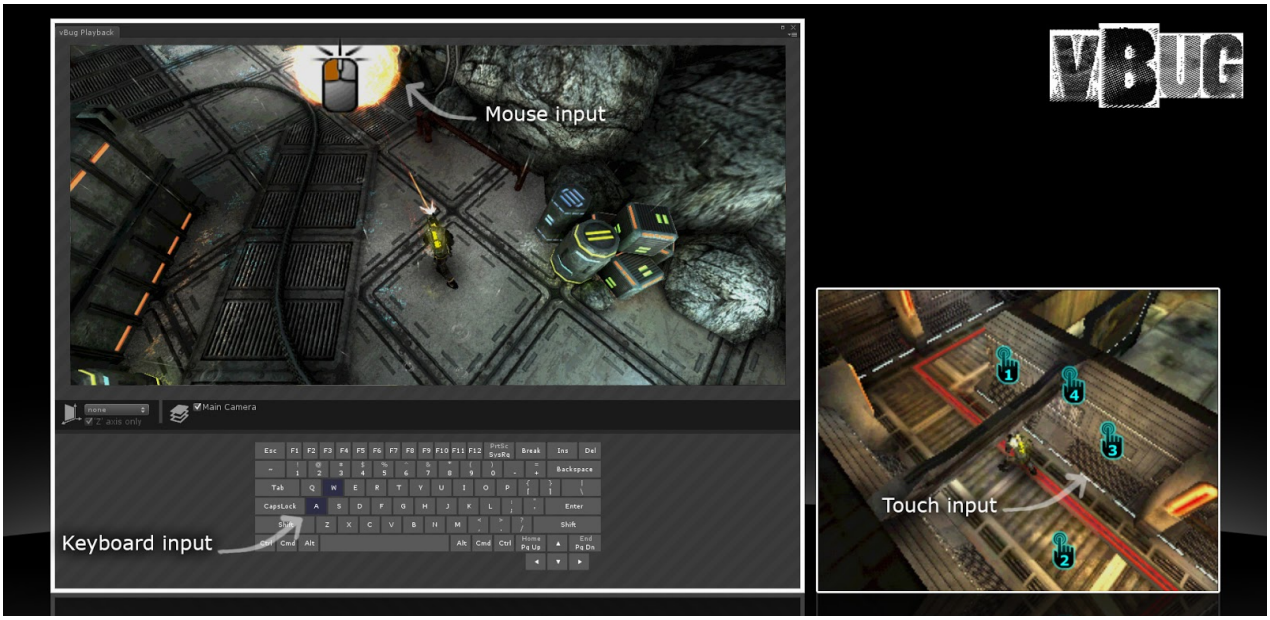
4.4. vBug Console window

This window will display all logs fired during your gameplay-session. It usually takes a while to load all the *.vBugSlices in the background, therefore please be patient.

If you click on a log, beneath you will find the stack trace of this log. In case you are running on mobile, make sure 'Development Build' & 'Script Debugging' are enabled.

If you double-click a log, the 'vBug Timeline window' will automatically jump to this frame, and all other windows will update accordingly.

4.5. vBug Playback window



This window is primarily focused around the output of the 'ScreenCapture' recorder, combined with the 'HumanInput' recorder. All of these combined makes vBug super useful and you will never wonder 'what was the player doing?' ever again.

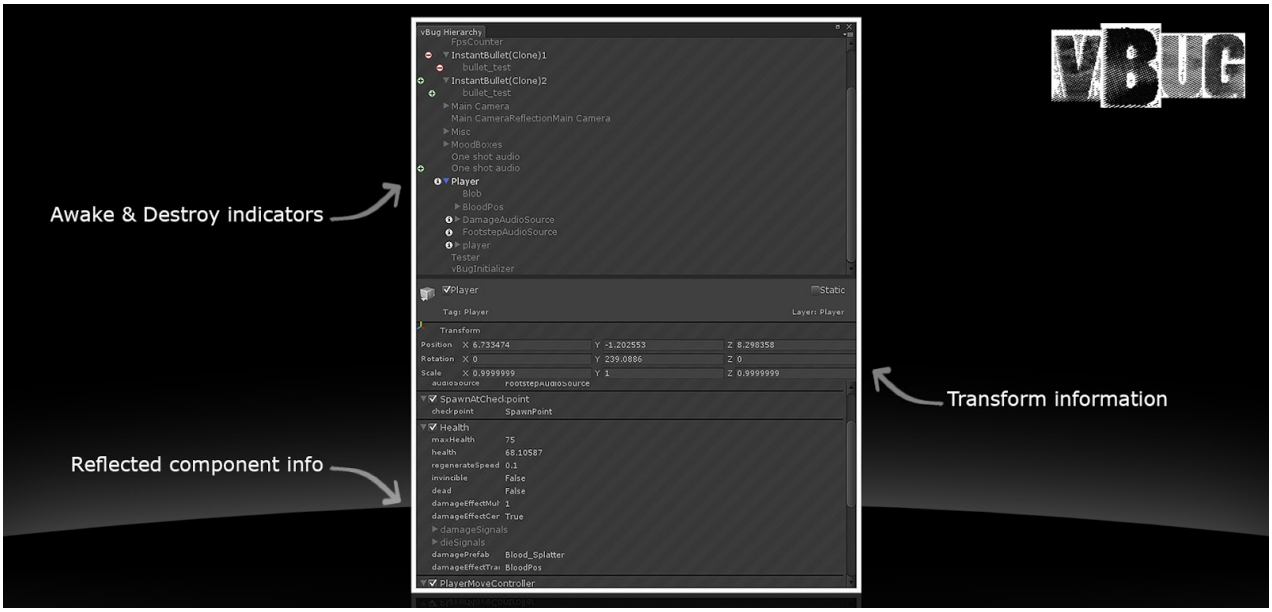
The window is split in two parts:
At the upper-part of the window, you will find the (closest available) frame corresponding with the current frame of the timeline (displayed in the 'vBug Sceneview navigation toolbar'). The captured frame might be a composition of multiple active camera's. Each individual camera render-layer can be toggled.

Besides toggling layers on and off, you can also review the output in 3D-mode. Its comes in handy when for example you created a camera that's linked to the 'accelerometer' to compensate mobile-device rotations.

On top of all the camera-layers, the Human-Input is displayed. Currently only mouse & touch are supported. You will be able to tell exactly where the player was tapping at any given moment, explaining why things went wrong or why he got stuck.

On the bottom part of the window, you will find a virtual-keyboard. Each button that's pressed will be highlighted, revealing the players behaviour any step of the way.

4.6. vBug Hierarchy window



This window is split in two parts: The scene-hierarchy and the 'inspector'-style component information.

Scene-hierarchy:
This upper-part of the window is meant to feel familiar and navigation is pretty much equal to Unity's own 'Hierarchy' window. I did make some alterations though:

Next to each item in the hierarchy, sometimes you will find an additional icon:

- A 'green plus' icon represents an object that got created that frame
- A 'red minus' icon represents an object that no longer lives the next frame, therefore this is its last frame 'alive'.
- A 'gray info' icon represents an object that has reflected component information available. Select this object to review its component info at the bottom part of this window.

Component-inspector:
This bottom-part of the window is meant 'look and feel' like Unity's own 'Inspector' window.

You can select an object in the 'Hierarchy' part of this window with the a little information-icon next to it. If there are none available, please read the 'Project & Scene setup' -> 'Hierarchy & Component' chapter.

Once selected, the 'Inspector' part of the screen will update. If you scrub the timeline or pressed the 'play' button in the 'vBug navigation toolbar', the selected object will stay focused for as long as its alive.

The biggest difference between Unity's own 'Inspector' and this one is the fact that its a reflection of something that 'happened' and not 'is'. To make clear what this means, let me give you an example:

If one of your components has a public field like this: 'public Texture2D myTexture' and this has a reference to an actual texture, vBug will be unable to capture it.

I am currently investigating if I can save references and 'find' the original reference during playback so it can be visualized, but for now its just to much data to capture every frame.

5. Limitations & Troubleshooting

5.1. Platform limitations

Flash & Webplayer

Due to the limitations of System.IO operations, vBug is not allowed to write any data to the disk. We are currently investigating ways to stream the data to a server, but for the time being these platforms are not supported.

Unity-free vs Unity-pro licence

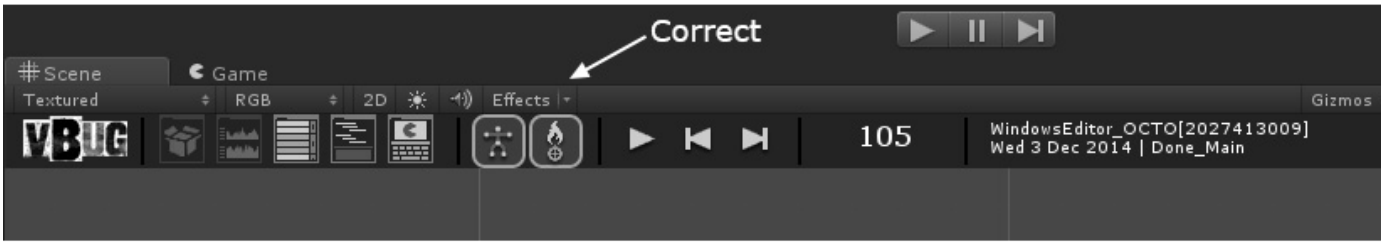
Unity free does not allow camera's to use render-textures. Therefore during screen capturing vBug cannot use the 'jit-render' (Just In Time) capture method and will fallback to "end-of-frame" capturing. Unfortunately this will disable the multiple camera support and cost additional performance.

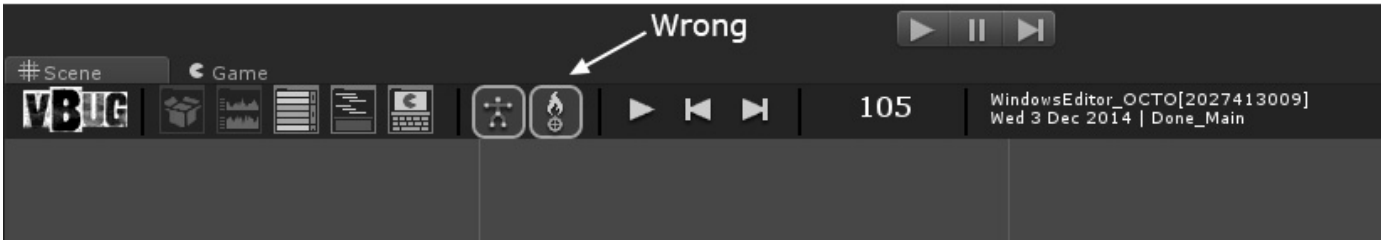
5.2. Bugs and issues

Sceneview navigation-bar Direct-X issue

What's it about?

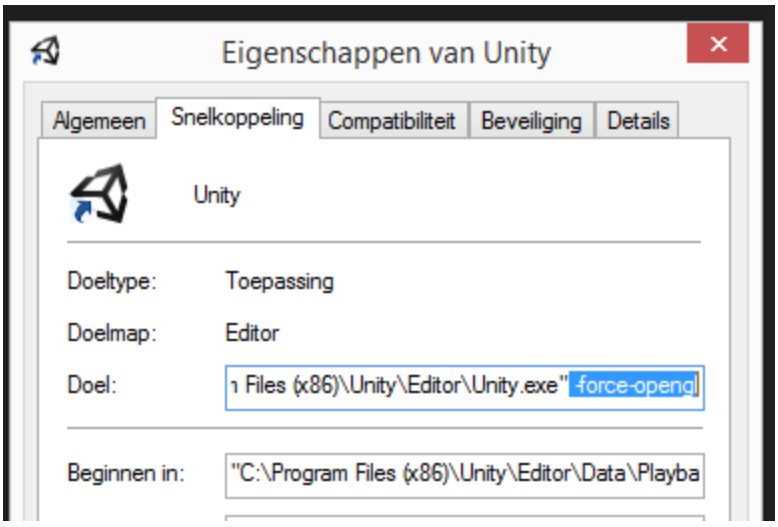
The vBug navigation-toolbar is rendered on top of the sceneview, but for some reason this gets a little y-offset in perspective camera mode running Direct-X editor only. It disappears when switching between perspective and orthographic mode in your sceneview.





What's the solutions?

I reported this bug months ago, together with some other folk that experienced the same problem. For now, the only known workaround is to force Unity to use OpenGL by adding *"-force-opengl"* to the command line of your Unity-shortcut. (screenshot is in dutch)



6. vBug.settings guide

6.1. General settings

General

bool debugMode = false
Enable 'debugMode' to receive logs and errors by vBug itself.

bool multiThreading = true
Enable/Disable multi-threading during recording. This speeds-up data-encoding and capturing.

int framePadding = 10
Each frame is saved as 'vBugSlice' file, and framePadding ensures the amount of digits are constant (ex: 000000001.vBugSlice)

VerticalSliceSettings verticalSlice = new VerticalSliceSettings()
Settings regarding frame-processing

DataProcessingSettings dataProcessing = new DataProcessingSettings()
Settings regarding data-processing

DataProcessingSettings

int threadSleepTime = 5
Sleep-time per worker thread between processing-jobs

int processPackageMaxQueueSize = 64
Max pending data-packages ready to be processed

int processPackageMaxPendingSize = 64
Max amount of packages being processed in parallel

VerticalSliceSettings

int maxPendingQueueSize = 15
Number of unprocessed slices in the queue

int maxPendingLifetimeMS = 5000
Maximum lifetime per slice to be processed

int maxPendingSleep = 5
Incase the max queue is reached, the main thread will be 'slowed down' allowing the processing-threads to catch-up. If its set to false, the unprocessed slices will be pruned

int sleepTimeMS = 30
Maximum mainthread sleep-time

6.2. Shared settings between recorders

bool enabled = true
Enable / Disable recording

CaptureIntervalSettings

float maxFrameRate = -1
The max-framerate cap with which the current recorder is running

int targetInterval = -1
The interval the recorder captures frames. In case this still exceeds the max-framerate, it will be capped to that value

int frameOffset = 0
The frame-offset with which a recorder will start.
By setting each recorder to a slightly different offset combined with a larger targetInterval will help distribute the workload among multiple frames

6.3. Recorder specific settings

SystemInfoSettings

bool autoEnableProfiler = false
Set this settings to 'true' if you also want the memory-usage to captured. This is an Unity-PRO feature only. Enabling this setting will cost additional performance and to be fair, Unity's own Deep-profiler is absolutely

int memoryUpdateInterval = 30
This 'sub-capture-interval' will spread the memory capture even further (on top of the 'captureInterval' settings). This is needed because scanning all the memory takes-up a lot of the performance (much like Unity's own deep-profiling)

bool gcForceFullCollection = false
In order to get a correct estimate of internal memory usage, this boolean should be enabled. However, if set to 'true', performance will be killed!

bool recordTexture2Ds = true
bool recordRenderTextures = true
bool recordMaterials = true
bool recordMeshes = true
bool recordAnimations = true
bool recordAudioClips = true
Turning off individual types will save a lot of performance each time memory is measured. This is especially useful when 'memoryUpdateInterval' is set to a low value and you experience performance dorps.

DebugLogGrabSettings

Only *'enabled'* and *'captureInterval'* are available settings.

HumanInputSettings

string[] activeProviders = new string[]
"MouseInputProvider",
"KeyboardInputProvider",
"TouchInputProvider",
"AxisInputProvider"
A list of the enabled data-providers to be included. Please disable by commenting the lines.

MouseSettings

float moveDistanceThreshold = 5f
float holdDurationThreshold = 0.25f
The amount of pixels a mouse needs to move before it will appear as 'moving' in the 'vBug PlayBackWindow'

AxisSettings

bool useGetAxisRaw = false
If set to false it will use the normal 'GetAxis'. If set tot true, it will use GetAxisRaw instead/

float detectMotionThreshold = 0.5f
The minimal threshold movement per second in order to appear as 'moving'

MeshDataSettings

int forceKeyFrameInterval = 120
During playback, vBug reconstructs meshes by searching for the closest available keyframe-data, much like modern video-data-compressions.
Each keyframe contains a full meshcapture, ensuring relevant meshdata to work with during playback

bool scanRecursiveUp = true
Only objects with a 'vBugMeshRecordable' component will be captured. This flag will also capture all of its children. If you only want to capture the object containing the 'vBugMeshRecordable' component and not his children, then set this boolean to 'false'

int maxBlendWeights = 1
When performing a full mesh capture, we can drastically gain performance by limiting the amount of bone-influences (much like Unity's own quality setting)

string[] activeProviders = new string[]
"MeshFilterProvider",
"SkinnedMeshProvider"
A list of the enabled data-providers to be included. Please disable by commenting the lines.

ParticleDataSettings

string[] activeProviders = new string[]
"ShurikenParticleDataProvider",
"LegacyParticleDataProvider"
A list of the possible data-providers to be included. Please disable by commenting the lines.

MaterialCaptureSettings

Only *'enabled'* and *'captureInterval'* are available settings.

GameObjectReflectionSettings

bool allowStaticGameObjectScanning = false
This will also scan all static game-objects. Since these will not change, its set to 'false' by default because they are not that interesting to capture anyways.

bool allowCustomClassPropertyScanning = false [EXPERIMENTAL]
The recursive-component reflection will (just like Unity) only capture exposed (or flagged as [SerializeField]) fields. This options allows you to capture properties as well, calling the 'getter' which might trigger some logic...

int objectMemberScanDepth = 3
When scanning nested member collections or objects recursively, it needs to end at some point, otherwise capturing will be too costly and burn too much storage-space.

int collecionMemberMaxSize = 25

When scanning collections, we want to make sure its memory-footprint and performance-cost are kept sane. Therefore we limit the content using this value. Please increase this value if you depend on collections to be properly reflected/captured.

ScreenCaptureSettings

int maxScreenCaptureSize = 256

The maximum x/y dimension the captured texture will be stored (and rendered when using jitRender). The lower the value, the sooner storage/rendering will be finished, drastically improving performance. This also greatly influences the required space to store a single .vBugSlice file

ScreenCropMethod cropMethod = ScreenCropMethod.none [EXPERIMENTAL]

This will force the used Texture2D's & RenderTextures to be initialized at POT values only. In some situations/devices this might save a lot of performance

ScreenCaptureQuality quality = ScreenCaptureQuality.lzfRGBA16

Storage quality & compression settings

lzf: Fast and simlpe type of compression - save a LOT of space and thanks to multithreading not a big deal performance-wise

24 bits: 8 bits per color (256)
32 bits: 8 bits per color (256), including alpha
16 bits: 4 bits per color (16)
8 bits: 8 bits for one color (grayscale)

ScreenCaptureMethod captureMethod = ScreenCaptureMethod.jitRender

The main capture method:

- endOfFrame: If you DONT have a pro-licence, or you want to capture the Unity.OnGUI drawings, this is the way to go.
- jitRender: Just-In-Time render allows for great performance if 'maxScreenSize' is kept small. This will render each Camera again, right before the capture occures.

RenderTextureFormat rtFormat = RenderTextureFormat.ARGB32 / RGB565

By default set to 'ARGB32', but 'RGB565' results in better performance, especially on Android-devices. However, this will reduce the quality and it will cause some errors on certain OIS-devices

int rtDepth = 16

The render-texture depth setting: 0, 16 or 24 bits

bool jitRenderLowQuality = false [EXPERIMENTAL]

This will change the render-settings (like AA etc) to its absolute minimum right before it jitRenders a scene, and resotres it afterwards.