MEMORY USAGE

View online: https://www.construct.net/en/make-games/manuals/construct-3/tips-and-guides/memory-usage

Managing the amount of memory used is important to ensure a wide range of devices can run your project. If a project uses more memory than is available on a particular device, it may crash due to running out of memory. Reducing memory use can also in some cases improve performance.

Images

Usually object images (including sprite animations) are the most memory-consuming part of a project. For this reason Construct estimates the peak memory use from images and displays it in the *Project Statistics* dialog. (Right click the project name in the Project Bar and select Tools▶View project statistics to view the dialog.) You should check this from time-to-time while developing your project, but it should only be regarded as an estimate. The debugger can also show the image memory usage during runtime, but note that it can vary depending on what is happening in your game. Remember this measurement is only for images, so your project will need at least that much memory to run.

Some tips to ensure the image memory usage remains reasonable are:

1. **Crop all animation frames.** Transparent pixels still use up memory. If you have a 1000x1000 image that is all transparency and just a 200x200 size piece of artwork in the middle, it will use as much memory as a 1000x1000 image - about 4 MB, whereas if cropped a 200x200 image would use less than 0.2 MB - a saving of over 95%
2. **Avoid unnecessarily high-resolution images.** If you import an image that is 2000x2000, but only display it sized 400x400 inside the game, then it uses as much memory as the source image (about 16 MB), whereas if the image was more appropriately sized it would only use about 0.6 MB of memory - also a saving of over 95%.
3. **Avoid excessively long animations.** All images in an animation need to be loaded in to memory. If an animation with 100 frames can be reasonably achieved with just 50 frames, it will use half as much memory.
4. **Stay under power-of-two sizes.** Construct packs Sprite images onto spritesheets, which for technical reasons are always a power-of-two size (e.g. 512, 1024, 2048, 4096...). This means that images just over a power-of-two size, like 530px, pack inefficiently: they don't fit neatly across a spritesheet and end up wasting space. However images just under a power-of-two size, like 500px, do pack efficiently as they can fit more images on a spritesheet. Note that Construct also adds some padding so also avoid exactly power-of-two sizes like 512px - you should try to keep images at least 4px under this size to account for padding. To illustrate this, consider that on a spritesheet sized 2048x2048, an image that is 530x530 wide can only be fit in 9 times (3 across and 3 down), but an image that is 500px wide can be fit in 16 times (4 across and 4 down) - in both cases the spritesheet is the same size and so uses the same amount of memory, but the

latter case is a much more efficient use of that memory; also consider that if both cases use 16 images, the former case will then need to spill another 7 images on to another spritesheet, which will use more memory and also pack inefficiently. (Note an exception is Tiled Backgrounds are not used on spritesheets, and so can be a power-of-two size.)

5. **Use composition for level design.** Some people are tempted to use lots of unique large image tiles to design levels as if they were one huge image. This will use a lot of memory and is not how most modern games are designed. Instead use a Tiled Background to make a repeating base layer (possibly using tile randomization to make the repetition less obvious), and then design lots of individual pieces of a level that can be independently placed and re-used at different sizes and angles across the level. For example rather than designing a huge single image with a whole forest drawn on to it, use a repeating background, design a small selection of unique trees, and then place these at different sizes, scales and angles across the level to build a forest. Effects like color tint and opacity can also be used to add more variety and break up any repetition. Then the entire level's image memory usage is only the amount for the background and each unique tree, even though a much larger level was designed.

The estimated peak memory use is based on only the single layout with the largest memory requirement, because only images for one layout are loaded at a time. In Construct this is called *layout-by-layout loading*.

Layout-by-layout loading

Construct only loads the images for the current layout. This avoids loading the entire project in memory which would be slow and consume a great deal of memory. When starting a layout, all images for the objects placed in the Layout View are pre-loaded. This includes all frames in all animations of any Sprite objects. (In other words, Sprites are either fully loaded in to memory, or not at all - they are never part-loaded.) When the layout ends, all images that are loaded but not used on the next layout are released from memory.

If an object is not placed in the Layout View, but an event sheet creates it at runtime, its images are not pre-loaded. Construct is forced to load the object images at the moment of creation, which can cause a momentary pause, or in extreme cases a constant stuttering (also known as "jank"). Construct's image memory usage estimate in the editor will not include such images as it doesn't know if they will be used, and so the memory usage of these kinds of dynamic loaded images will only be reflected in the debugger's runtime image memory measurement. A better approach is to place any objects that will be used by the layout in the Layout View. If they are not immediately needed then they can be destroyed in a *Start of layout* event. They will then not exist when the layout starts, but Construct will still have pre-loaded their images, ensuring that they can later be created at runtime without any jank.

Calculating image memory use

First of all it is important to note the image format has no effect on memory use. You can save on your project's download size by setting some images to a different format like PNG, JPEG or WebP. However this does nothing to memory use: compressed images cannot be directly rendered, so upon loading all images are decompressed in to a 32-bit ARGB bitmap format. This

means each pixel takes four bytes for the alpha, red, green and blue channels. (Note images are also stored compressed inside your project file - so the size of your project file has nothing to do with its memory use. You can easily have a small project file with a huge memory requirement.)

Consequently, the approximate memory use of an image in bytes is its number of pixels multiplied by 4. For example a 100x100 image would use 100 x 100 x 4 = 40000 bytes, or about 39 KB. A HD-sized image at 1920x1080 would take 1920 x 1080 x 4 = 8294400 bytes, or about 7.9 MB. Note that transparent pixels still count!

It is also important to note that the memory use is based on the source image - that is, as it appears in the image editor. If the object is stretched in the Layout View or at runtime, it does not use any more or less memory. It is just rendering the source image in memory (which is always at its original size) at a different size on to the screen.

The image memory usage is based on all the images (including all animation frames) of all objects on the current layout. Therefore using fewer, smaller images will always result in less memory usage.

Don't mis-use 'Downscaling quality'

Construct uses several optimisations, including spritesheeting, to save memory. Setting the *Downscaling* project property to *High quality* forces spritesheeting to pad out all sprites to power-of-two sizes, negating the memory saving of spritesheets. This can significantly add to the memory requirement of your project. *High quality* mode exists only to address two relatively minor rendering issues (edge fringing on sprites, or altered quality on the last animation frame). It should not be used unless one of these rendering issues has been specifically observed and the issue is resolved by choosing this option. Otherwise your project will be paying a very high price in memory usage for no reason.

Audio

Usually images take up the most of a project's memory use. However it's worth noting how audio is loaded in to memory.

It's important to categorise audio between the sound and music folders because they are loaded differently and therefore have different memory usage.

Sounds

Audio in the *Sounds* folder is fully decompressed in to memory. This allows sound effects to be played instantly without any latency from having to first load or decompress the audio, ensuring sound effects are heard at the appropriate time. Like with images, the compressed size helps reduce the download but does not reduce memory use: the sound will be decompressed in to PCM wave buffers.

By default the project property *Preload sounds* is enabled, meaning all sounds are downloaded and decompressed while the loading bar is showing. As a result *all* sounds in the project will be decompressed in to memory on startup. With a common playback configuration of 16-bit

samples at 44.1 KHz, one second of single-channel audio will use 88000 bytes (about 86 KB), and double that for stereo (about 172 KB). For one minute of audio, that is about 5 MB for single channel and 10 MB for stereo. This is the primary reason music tracks should not be categorised as "sound": if you have 15 minutes of stereo music, that will consume 150 MB of memory. On some platforms, decoding a full music track can also be quite slow, adding a lot to the startup time.

If *Preload sounds* is disabled, sounds are not loaded during startup and the project can start quicker. However the first time each sound is played may be delayed since it must first download and decode it in full. Using the *Preload* action in the Audio object can help mitigate this. Note Construct does not release sounds once loaded to ensure they can be played quickly again. To release the memory, you must use one of the *Unload* actions. This makes you responsible for managing audio memory in your project.

Audio in the *Sounds* folder should be short, latency-sensitive sound effects. Consider cutting down any very long duration sound effects - or, if playback latency is not important, consider categorising it as "Music" instead.

Music

Contrary to sound effects, music is streamed. Generally this means the audio engine will have a small playback buffer of a fixed length, and while the audio is playing it is loaded, decoded and played in small chunks that connect together seamlessly. This means the memory use is low regardless of the length of the track, and it can even start playing the audio before it has finished downloading. This is why music is not pre-loaded while the loading bar is showing - there's no need to wait for it to finish downloading before starting the project. However playback cannot always start immediately, since it may need to wait for the download to finish buffering, or for the first chunk to load and decode. In terms of memory use, the main consideration is simply to make sure long audio tracks is categorised as music and not sound.

Other resources

Many other types of resources will use a small amount of memory. However these are not usually significant relative to the memory usage of images and audio, so usually you can disregard them. Remember though that nothing in computing comes for free - for example every object that you create will use a small amount of memory, so you could also end up using lots of memory if you create tens of thousands of objects. This could even happen by accident, such as if you have an event sheet that creates objects but you forgot to add any logic to destroy them later, so it just endlessly creates more and more objects.

Anything that is pushed to an extreme will likely end up using a lot of memory, so try to make sure all aspects of your project remain in reasonable proportions, and it is unlikely you will have any problems with anything else.