RG Super Meter 2.1
User's Guide

Written by Edward F. Maurina III
Roaming Gamer, LLC.

Version 2.1
Last modified: Feb 11, 2014

Table of Contents

Basic Usage

To use the RG Super Meter, do the following:

1. Copy the folder rgmeter into your app's root directory (same directory as main.lua).
2. In main.lua or any other file where you intend to start the meter, require the file like this:

```
local rgmeter = require "rgmeter.rgmeter"
```

3. Immediately create a meter like this:

```
rgmeter.create()
```

Reading The Meter

The RG Super Meter has five components:

1. **FPS Meter** – A meter that graphs the FPS over time and calculates the running average for FPS over a specified number of samples (30 by default) with real-time output.
2. **Memory Meter** – A meter that graphs the total (main) memory usage over time and calculates the running average for memory usage over a specified number of samples (30 by default) with real-time output.
3. **Video Memory Meter** – A meter that graphs the total video memory usage over time and calculates the running average for video memory usage over a specified number of samples (30 by default) with real-time output.
4. **INFO Tabs** – A series of tabs/pages that provide tons of information about your device and your app. Please see the INFO meter section below for more details.
5. **DBG Tabs** – An ever expanding set of controls that allow you to control Corona settings in real-time and to implement other debug activities like screen grabs (with e-mail capabilities).

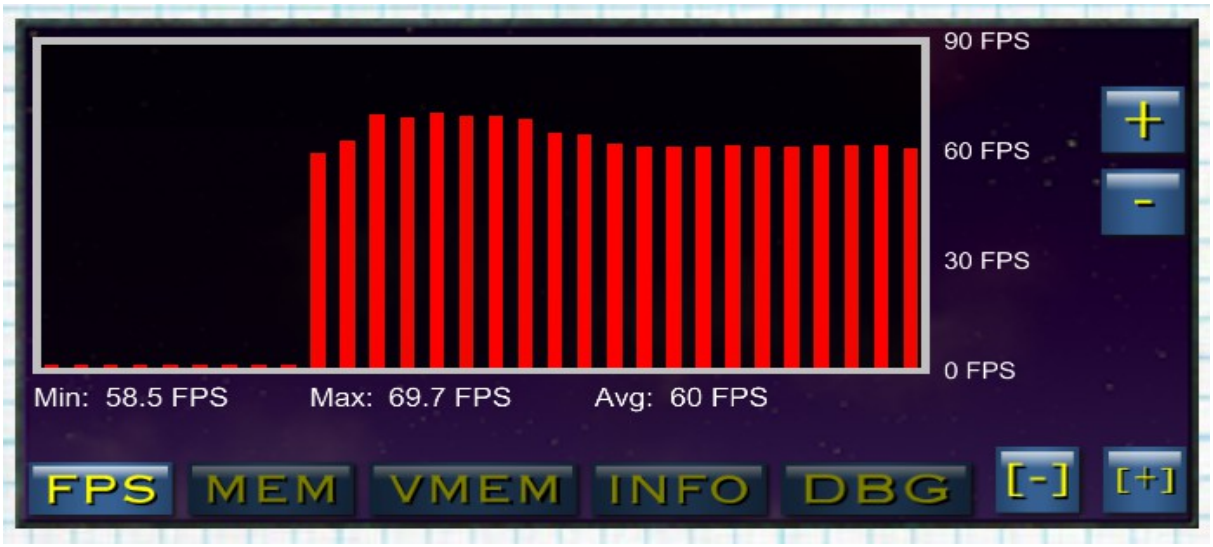
Navigating the Meter

The meter has six buttons at the bottom. Touching any of the first five buttons, 'FPS', 'MEM', 'VMEM', 'INFO', or 'DBG', will switch to that meter. Touching the '[-]' button will minimize the meter. Touching the '[+]' will scaled the meter to full- screen width and center it. Tapping the same button again will scale back to the original meter size and move back to its original position.

Additionally, the meter can be dragged around the screen by touching anywhere on the meter (except the buttons) and dragging.

It will always stay on top of all app content (if not placed in a specific group) and so may on occasion obscure the screen. Simply drag it to get it out of the way.

Note: The meter has a safety feature whereby it will not allow itself to be complete dragged off-screen.



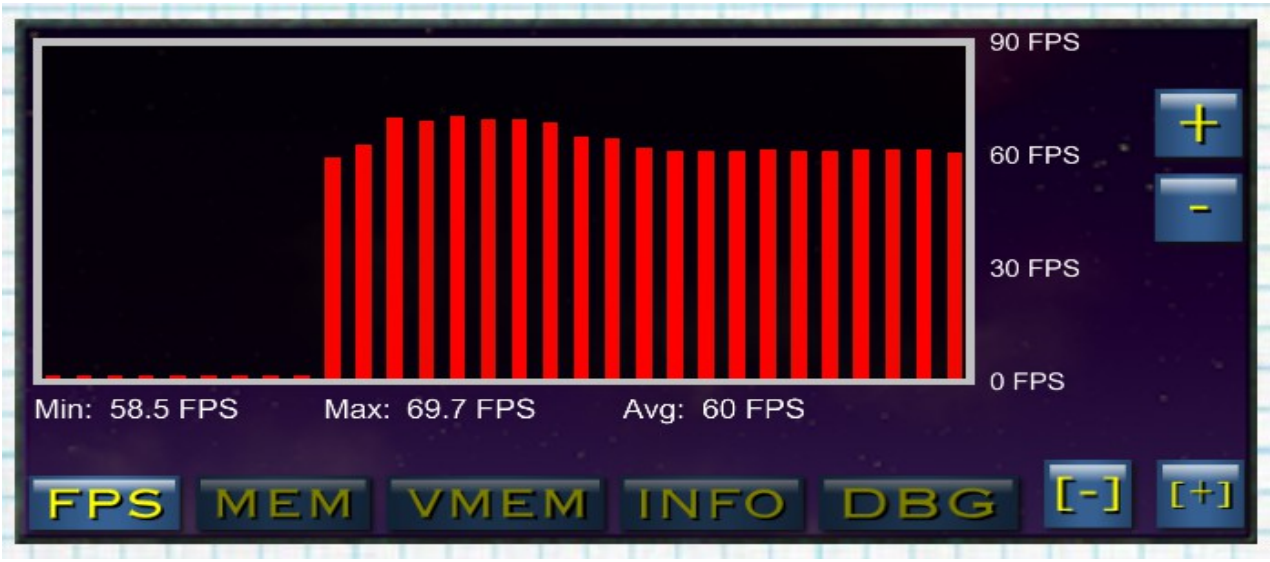
When the meter is minimized it will present itself as a small icon. This icon can be dragged around to get it out of the way, or tapped to re-open the meter.

Both the meter and the icon remember their last positions and will return to them when the meter is opened/closed.

The FPS Meter

The FPS Meter has the following major parts:

- **Vertical Graph** – The meter detects the target FPS (either 30 or 60) and provides a graph with space for up to 1.5X the target FPS. This allows for over-shoot.
- **Horizontal Ticks** – You will notice that this meter shows a series of RED ticks that advance right-to-left, where the most recent tick is on the right. These represent historic values.
    - Note: If a tick turns PINK, it means that it has over-shot graph limit.
- **Min, Max, and Average Labels** – Below the graph you will find the minimum, maximum, and average FPS values.
    - ***Tip: You can adjust the size of the averaging window from 1 to N frames. See references below.***
- **'+'/'-' Buttons** – The '+'/'-' buttons on the right side of the meter can be used to increase or decrease the meter scale in real-time.

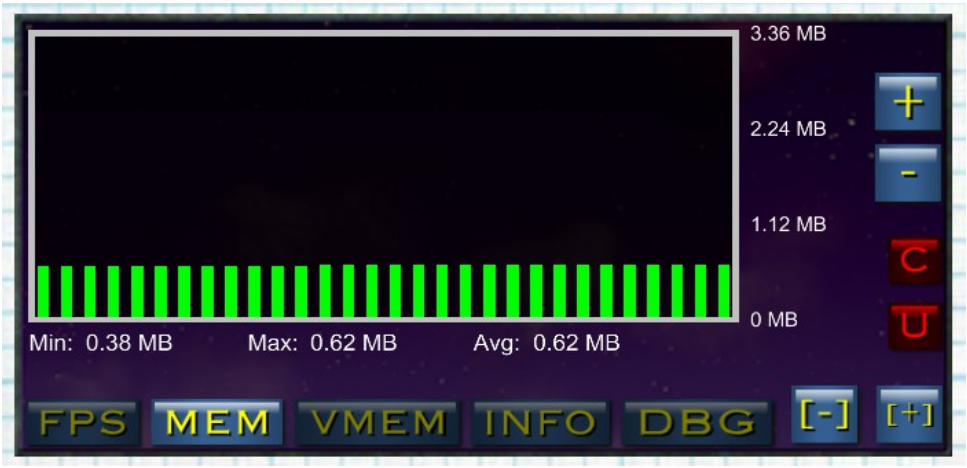


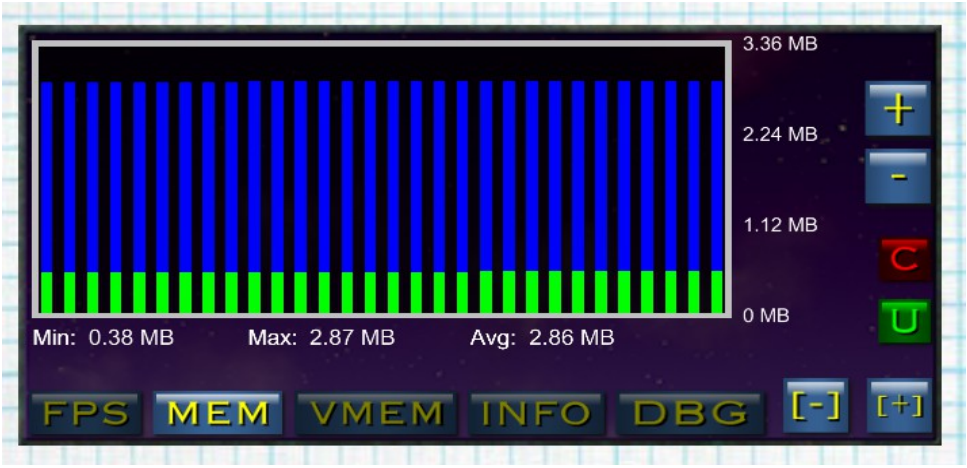
The Memory Meter

This meter is similar to the FPS meter, but records CPU memory usage in MB. As with the FPS meter, the range can be adjusted dynamically by pressing the '+'/'-' buttons on the right.

This meter has seen a major update since version 1.3. Now, you can do the following:

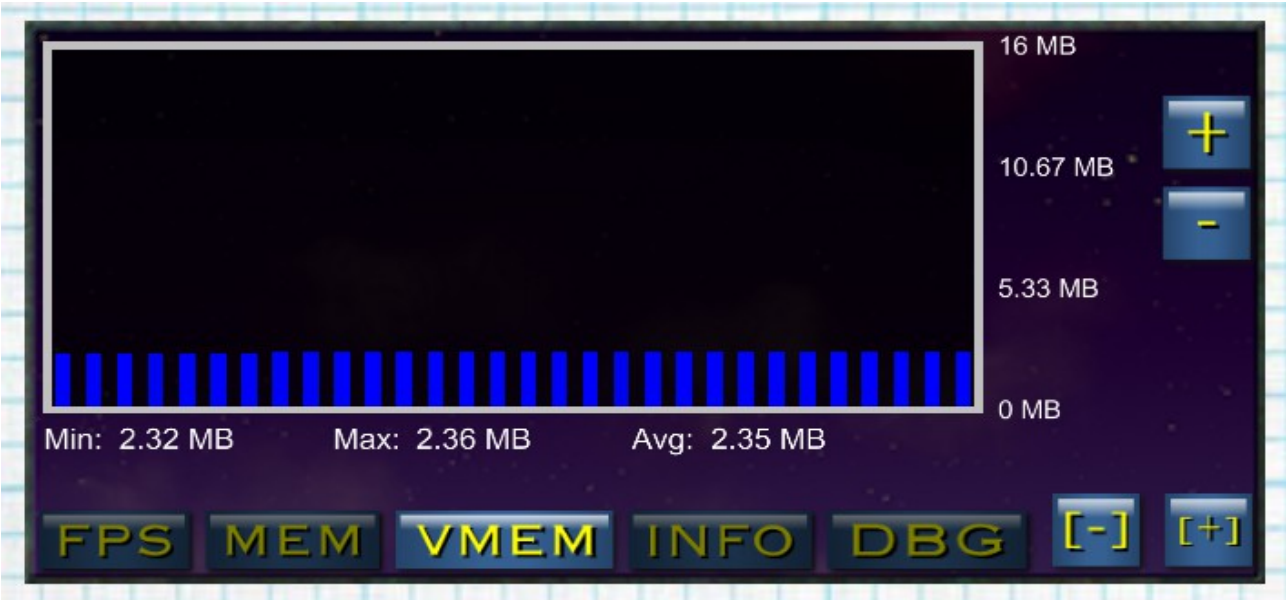
- Enable/disable memory collection in real-time by pressing the 'C' button on the right side of the screen. When the button is red, collection is off. When it is green, collection is done prior to every reading in order to give you the most accurate memory reading possible.
- Display unified memory usage. Today, many devices utilize unified memories, where the CPU and Video chips share a single memory source. By pressing the 'U' button on the right side of the screen you can show unified memory usage to get a better idea of total memory usage. When the 'U' is red, the meter is showing only CPU memory usage. When it is green, CPU and Video memory usage are combined.





The Video Memory Meter

This meter has the same layout as the prior memory meter, and only records the video memory usage.



The INFO Meter

This meter has seen a massive overhaul since version 1.3. The meter now has sub-pages for all of the following data types:

- **General Info (GEN) –** Basic information about the app and build environment, including:
  - App Name, Environment, Corona Build, Target, UI Language, and Locale data.
- **Architecture Info (ARC) –** Information about the device architecture, including:
  - Architecture, Model, Platform, Max Texture Size, Max Texture Units (not currently supported by Corona), (supports) High Precision Shaders, (active) Image Suffix
- **Resolutions (RES) –** Resolution data including:
  - Design resolution, (actual) Display resolution, Excess Resolution, Scaling Settings, Device Size (in pixels), Screen Origin.
- **OS & Timer (OST) –** OS supplied data and the current system timer (i.e. how long the app has been running).
- **OpenGL Info (OGI) –** Open GL settings including:
  - (hardware) Vendor, Renderer, Version, Shading Language Version
- **OpenGL Extensions –** An entire list of the OpenGL extensions supported on this device. May be multiple pages.
- **Android Settings (AND) –** If the device is running Android, two pages of Android data are provided.
- **IOS Settings (IOS) –** If the device is running iOS, one page of iOS settings is provided.
- **Loaded Packages (PKG) –** A full listing of all packages that were loaded (required) before the meter was created. (Tip: Create the meter after executing all require() calls to get a correct listing.)
- **Directories (DIR) –** Provides the status/existence of all possible directory types.
- **Fonts (FNT) –** A full alphabetic listing of all fonts supported on the device.
- **Media (MED) –** Lists the availability of the camera, photo library, and saved photo library.
- **System Events (SYE) –** Two pages listing the availability and current event inputs for:
  - Accelerometer, gyroscope, heading (compass), multi-touch, and (device) orientation, and location (GPS).
- **Suffix Test (SFX) –** A single image demonstrating that the listed suffix (from General Info) is in fact working.

The Debug Meter/Tools

This meter currently only has two tools under it, but I am open to suggestions, so please feel free to email them to me (roaminggamer <AT> gmail <DOT> com; use the word 'RGMETER' in the title).

The current tools are:

Physics Control (PHY) Tool

This tool lets you dynamically start, stop, and pause physics. As well as toggle between “normal” rendering and “hybrid” rendering.

Capture (CAP) Tool

This tool is meant to be used by yourself, teammates, and testers. It allow you to take a screen shot or to capture a part of the screen and then to optionally send the image to someone via e-mail.

To use this tool, do the following:

1. Include rgmeter and start it as per prior instructions.
2. Open the Debug Tools Page and hit the '+' button till you come to the “Capture” page.
3. Click either 'Full' or 'Bounded'.

“Full” Grabs

If you clicked 'Full', the meter will be hidden, a full screen grab will be taken, the meter will re-appear, and the image will be shown on the meter. Additionally, two new buttons will be shown:

1. **e-mail** – Click this button to email the image to one or more recipients.
2. **Delete** – Click this button to delete the screen copy of the image.

Regardless of your actions with the above two buttons, the image will also be stored in the device's photo library for future recall.

“Bounded” Grabs

If you clicked 'Bounded', the meter will be hidden. You now need to click and drag on the screen to create a bounding rectangle around the part of the screen you want to keep. When you lift you finger, the bounded screen grab will occur and follow the same sequence as a “full” grab.

RG Super Meter Syntax References

rgmeter.create()

Purpose

Creates a meter.

Syntax

```
rgmeter.create( [ x [, y [, width [, refUL [, startMinimized ]]]]] )
```

- **x / y** – <x,y> position of meter.
  - Defaults to < centerX, centerY >
- **width** – Width of meter.
  - Defaults to 320 pixels.
- **refUL** – Reference the upper-left. i.e. Anchor to the upper-left corner of the meter. By default, the meter is placed and anchored in the center.
- **startMinimized** – If set to true, the meter will shrink to the minimized state when started.

Examples

```
-- Create new meter at centerX, centerY with size 320 x 155
rgmeter.create()
```

```
-- Create meter at position < 300, 300 > with size  500 x 242 (height auto-
calculated)
rgmeter.create( 300, 300, 500 )
```

rgmeter.setAverageWindow()

Purpose

Sets the average window width. By default the average FPS/Main Mem/Video Mem are calculated over the prior 30 samples. You can increase or decrease the number of samples with this function.

Note: This must be called before creating the meter.

Syntax

```
rgmeter.setAverageWindow( samples )
```

- **samples** – Number of samples to calculate average over.

Example

```
rgmeter.setAverageWindow( 10 ) -- Calculate avearges over last 10 samples
```

rgmeter.setUpdatePeriod()

Purpose

Changes the number of frames between graph updates. While the average of an individual metric is tracked every frame, you may slow down or speed up the updating of the graphs. By default, the Graphs are updated every 11 frames.

Note: This must be called before creating the meter.

Syntax

```
rgmeter.setUpdatePeriod( period )
```

- **period** – Number of frames between graph updates.

Example

```
rgmeter.setAverageWindow( 1 ) -- Update graphs ever frame
```

rgmeter.setMaxMEM()

Purpose

By default, the meter assumes you have are expecting a maximum main memory usage of 2MB (2048 KB). However, this may not match your actual usage. You may use this function to further refine the scaling of the main memory meter.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.setMaxMEM( size )
```

- **size –** Maximum expected main memory usage in Kilo-Bytes.

Example

```
rgmeter.setMaxMEM( 128 ) -- Set meter to 128KB
```

rgmeter.setMaxVMEM()

Purpose

By default, the meter assumes you have are expecting a maximum main video memory usage of system.getInfo(“maxTextureSize”). However, this is often far more than you will end up using and the graph will be far out of scale. You may use this function to further refine the scaling of the video memory meter.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.setMaxVMEM( size )
```

- **size –** Maximum expected video memory usage in Kilo-Bytes.

Example

```
rgmeter.setMaxVMEM ( 16 ) -- Set meter to 16KB of video usage.
```

rgmeter.enableCollection()

Purpose

By default, the RG Super Meter does not force garbage collection every time it samples current main memory usage. It does this because doing so may affect your FPS measurements.

However, if you are trying to check for a memory leak, it is better to enable this feature. i.e. If you force garbage collection before every memory reading you will get a more stable and clear picture of memory usage. However, be warned that this may impact your frame to frame FPS rate.

Note: This must be called before creating the meter.

Syntax

```
rgmeter.enableCollection( enable )
```

- **enable** – If set to 'true', the meter will force garbage collection before reading the current memory usage.

Example

```
rgmeter.enableCollection ( true ) -- Collect garbage before reading usage.
```

rgmeter.setFontScale()

Purpose

On most systems and devices, the default font sizes (for the interfaces) will be fine. However, if you find that they are too small or too large, you can use this function to adjust them.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.setFontScale( scale )
```

- **scale** – Scale multiplier used for all fonts.

Example

```
rgmeter.setFontScale ( 0.8 ) -- Render all text at 80% of the default sizes.
```

rgmeter.minimizeInfo()

Purpose

Often times, you will only need the FPS/MEM/VID and most basic of Info meters. This function turns off all of these info meters in one fell swoop: OST, OGI, AND, IOS, PKG, DIR, FNT, MED, POP, SYE, SFX.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.minimizeInfo(  )
```

Example

```
rgmeter.minimizeInfo()
```

rgmeter.enableInfo()

Purpose

This function can be used to enable or disable specific info meters.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.enableInfo( name, enable )
```

- **name** – Three letter string for the meter to enable/disable (see meter titles).
- **enable** – 'true' to enable, 'false' to disable.

Example

```
-- Hide most info meters, but re-enable the System Events and SFX tabs.
rgmeter.minimizeInfo()
rgmeter.enableInfo( "SYE", true  )
rgmeter.enableInfo( "SFX", true  )
```

rgmeter.enableDbg()

Purpose

This function can be used to enable or disable specific debug meters.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.enableDbg( name, enable )
```

- **name** – Three letter string for the meter to enable/disable (see meter titles).
- **enable** – 'true' to enable, 'false' to disable.

Example

```
rgmeter.enableInfo( "PHY", false  ) -- Disable the physics debug meter.
```

rgmeter.setAppName()

rgmeter.setDefaultEmail()

Purpose

These functions allow you to set an app name and a default e-mail for use with debug Capture meter.

Note: This must be called before creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.setAppname( name )
rgmeter.setAppname( url )
```

- **name** – Name for app.
- **url** – Email address.

Example

```
rgmeter.setAppName( "My Cool Game" )
rgmeter.setAppName( "debug@yourcompany.com" )
```

rgmeter.setNumTicks()

Purpose

You can adjust the number of ticks used in the graphs from 10 to 60.

Note: This must be called <u>before</u> creating the meter.

Note2: All meters measure up to 1.5 types the expected value to allow for over-runs.

Syntax

```
rgmeter.setNumTicks( ticks )
```

- **ticks** – Number of ticks to display on graph.

Example

```
rgmeter.setNumTicks( 60 ) -- Use the maximum number of ticks.
```

Debugging Tips

Debugging with this meter is mostly straightforward. However, here are a few tips to help you out:

- Be sure to adjust the memory meters' vertical scales to match your expected (or observed) usage. Having the scales appropriately sized will make it easier to debug issues and to see variances.
- When looking for memory leaks, be sure to enable forced garbage collection like this:

```
rgmeter.enableCollection ( true ) -- Collect garbage before reading usage.
```

- When examining frame rates, do not enable forced garbage collection. (By default this is disabled.)
- Use an appropriate update rate and averaging window to your purpose.
- Be aware that sampling every frame can (slightly) affect your frame-rate.
- When debugging main memory usage, be aware that for cyclic operations, it is normal for memory usage to climb gradually and then to peak.
- When looking for memory leaks, you are primarily looking for a continuous (non-abating) ramp where more and more memory is in use and never freed. Eventually the graph will turn on pink.
- When you are debugging a frame-rate issue, where every other (or so) frame is about half or less than the expected rate, be sure to examine the main memory graph too (with forced collection off). If you see an extreme saw-tooth graph, you may be able to resolve the problem with a garbage collection setting like this like this:

```
-- Wait for memory in-use to quadruple (4x) before collecting.
collectgarbage( "setpause", 400 )
```

Examples With This Guide

This guide (and tool) comes with a number of examples which you can run and play with to see the meter in use.

Updates

After buying this tool, you will receive free life-time updates. So, don't throw away e-mails from Sellfy, GumRoad, and/or roaminggamer@gmail.com or you will miss out.

Templates & Docs

Lastly, you can find more Templates, guides, and tools here:

http://roaminggamer.com/makegames/.