SpringRTS Mapping guide for World Machine

- By Beherith (mysterme@gmail.com)

If you have questions/comments, use the comments feature of gdocs.
This is a detailed guide and checklist for making half-decent looking maps for SpringRTS. ☭

Prologue:

THERE IS NO INGAME MAP EDITOR FOR BAR. THIS IS A DIFFICULT BUT REWARDING JOURNEY.

Watch some videos:

SpringRTS map creation with World Machine Macros (by Beherith)

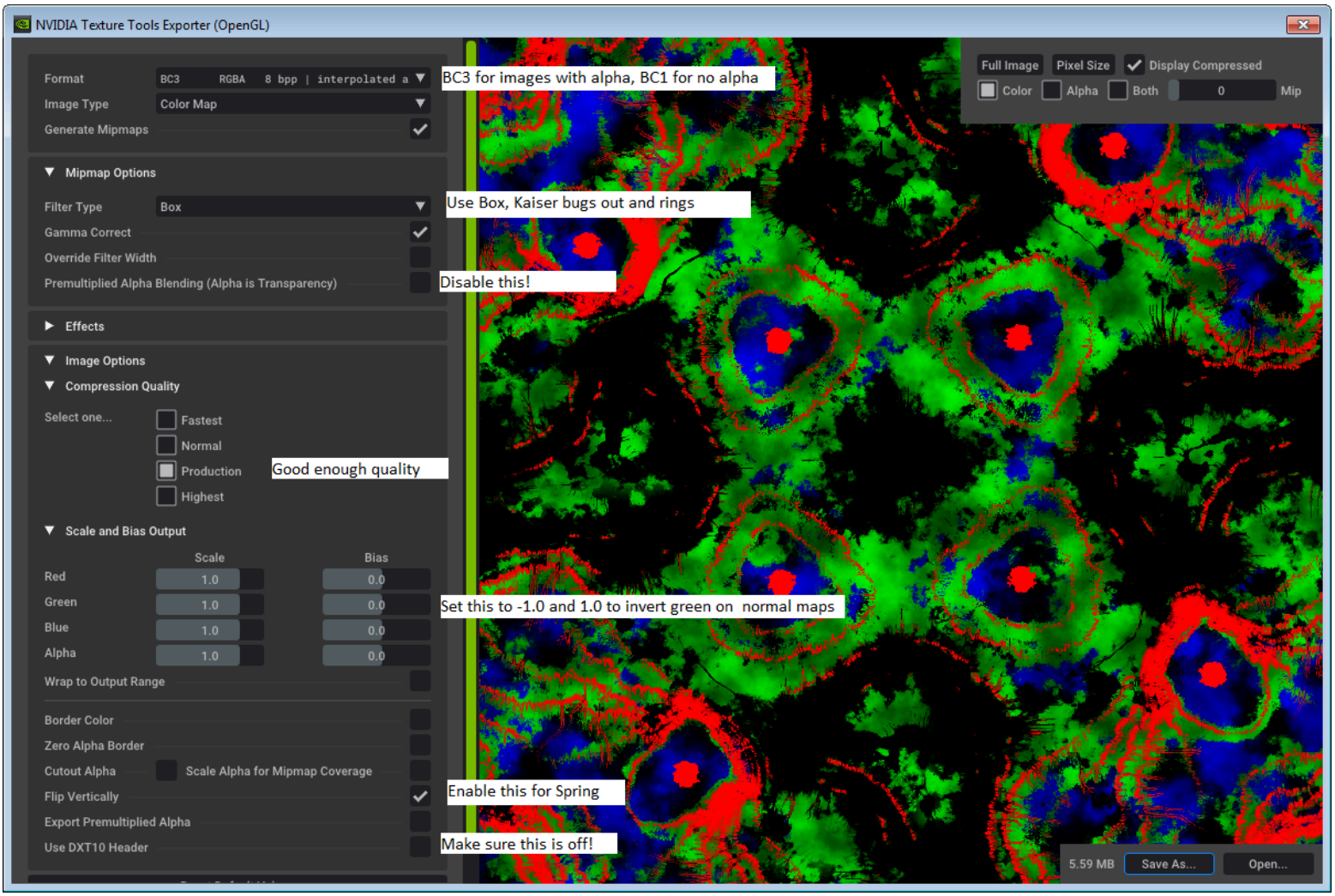
You can skip straight to the World Machine Usage section in this doc if you are familiar with the basics.

Recommended

Tools

1. Windows Texture Viewer
    a. https://github.com/Beherith/springrts_smf_compiler/blob/master/tools/win/Windows_Texture_Viewer_v089b.rar
    b. for viewing .DDS files
    c. if WTV can't display it, Spring can't either!
    d. Use left-right keys for viewing MIP levels
    e. Use ARGB keys to view individual channels
2. Photoshop or GIMP (optional)
    a. For miscellaneous fixes, especially for inverting the Green channels of normal maps
    b. Good for resizing
3. NVidia Texture Tools Exporter (see the image below)
    a. https://developer.nvidia.com/nvidia-texture-tools-exporter or https://github.com/Beherith/springrts_smf_compiler/tree/master/tools/win
    b. Best .dds generator
    c. Arbitrary sizes (16K+) possible, not just power-of-two!
    d. Fast enough
    e. Good preview.
    f. Has option to flip DDS vertically (needed almost everywhere)
    g. PyMapConv includes EZ mode .PNG to .DDS converters for mapping.
4. World Machine 3.x or World Creator for actually making the map. World Creator is- not detailed in this document, only World Machine 3+, but the general procedure is similar.
5. Pymapconv
    a. https://github.com/Beherith/springrts_smf_compiler
    b. The only map compiler/decompiler written, endorsed and supported by me.
6. Coffee, preferably plentiful and strong, and some willpower.

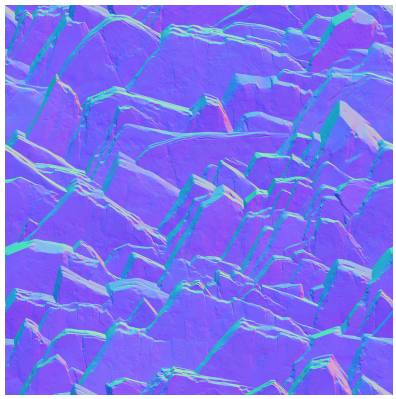
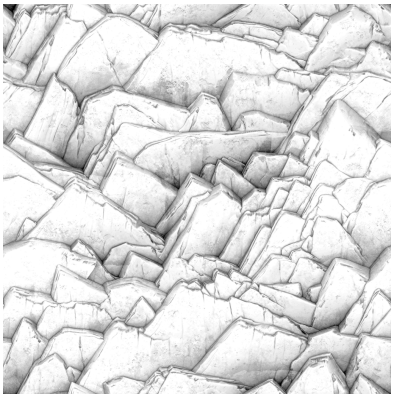
NVidia Texture Tools Exporter



Generating DNTS Maps from PBR materials

To create a good-looking DNTS splat texture from a PBR material, you need the following things:

Question: usually you just have a texture you have seen somewhere, how to get the normal and occlusion ones?
Answer: Normal maps are the most important, if you can't get that, you might as well just look for a different one, sorry.

 Normal Texture	Diffuse Texture	Self-ambient occlusion
		

If it is too clean, use Shadermap 3 pro to generate one from the diffuse texture and mix with the clean one in photoshop.	We will center this around grey and squash the range of the diffuse so that it doesn't overpower the normals with flat shading	Not strictly needed, but very useful 

splatDetailNormalTex

Make sure the directions of the channels are correct. Remember the **'red is right'** mnemonic, and the **'green is up**' mnemonics. To fix this, you may have to **invert and or swap the red and green** channels.

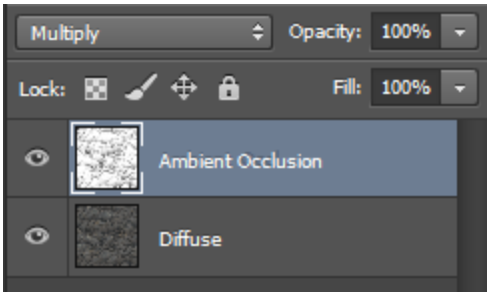
The format of the DNTS **splat map should be .TGA** (this matters because the origins of .DDS and .TGA are different, thus resulting in flipped green channels of the normal textures). The example shows the normals of a raised cone, sphere, rounded rectangle and the word 'UP', with a lowered 'DOWN' text.

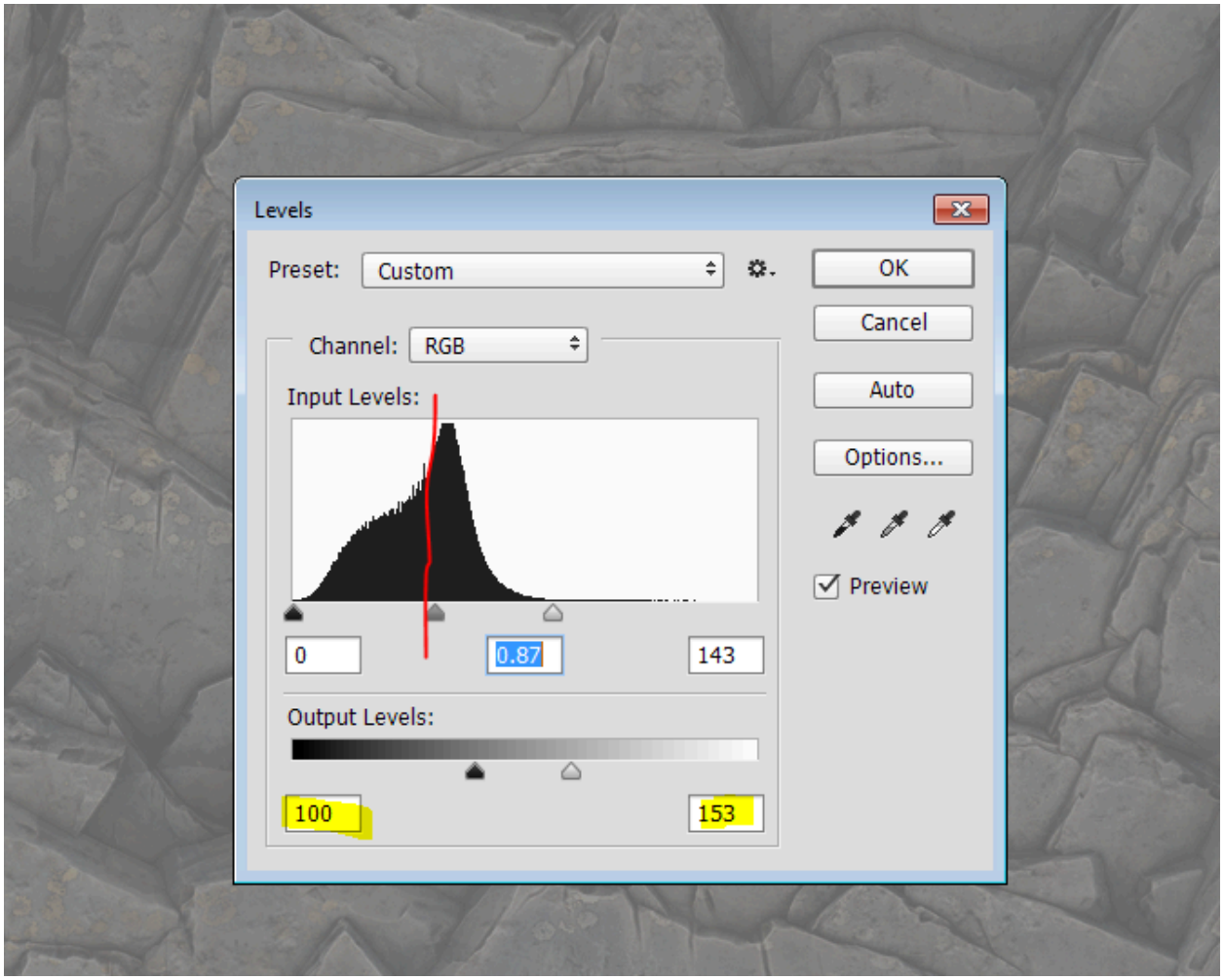
Do **NOT use .DDS formats for splatDetailNormalTex**, as the mipmapping results in too high compression, and ugly results. Use .TGA for best results. 1k sized splatDetailNormalTex is perfectly sufficient.

Making the DNTS

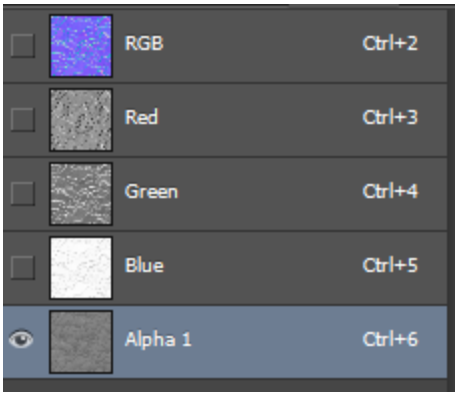
Combine the diffuse texture and the self ambient occlusion textures with multiply. Merge the layers.

Adjust the level to be centered around 128 and with a small spread, between 100 and 150, to make sure that the diffuse doesn't overpower the normal texture.





Copy this squashed diffuse texture to the alpha channel of the normal texture, and save as a .TGA file for export.



Put the texture in the ./maps folder, and reference it from mapinfo.lua:

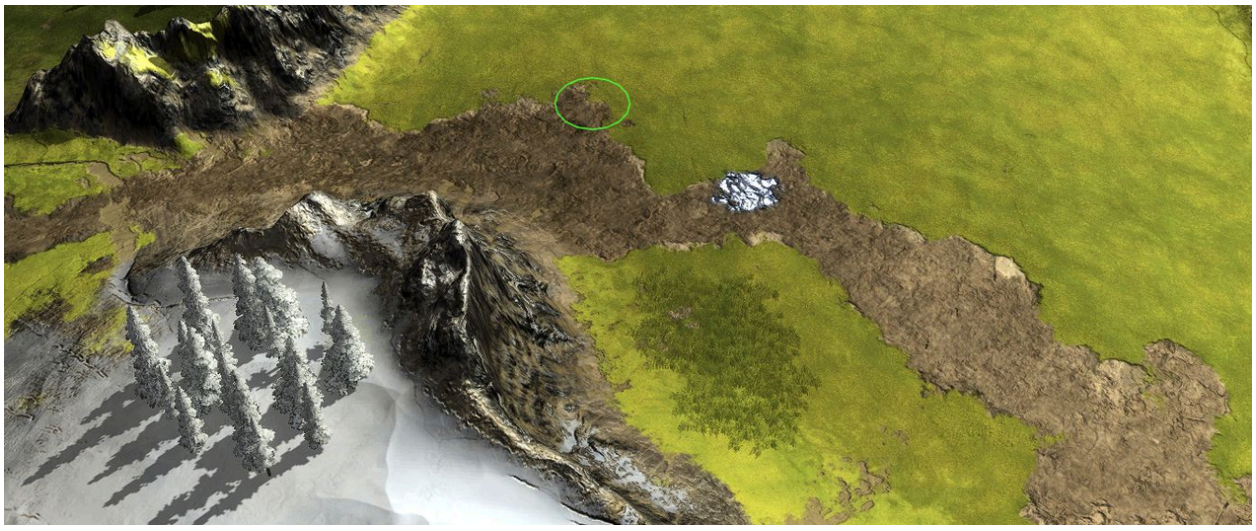
```
	resources = {
	specularTex = "spec_large.tga",
	splatDetailTex = "splattex.tga",
		splatDistrTex = "splatmap.tga", --sand, rock, pebbles, cracks
	--skyReflectModTex = "rrsky.dds",
	splatDetailNormalDiffuseAlpha = 1, -- specifies whether the diffuse (alpha channel) of splat textures get used
	splatDetailNormalTex1 = "Ground_IcySnowPack_1k_n.tga";
	splatDetailNormalTex2 = "rock_guiAngledRock_1k_dnts.tga";
```

```
        splatDetailNormalTex3 = "snow_NORM.tga";
        splatDetailNormalTex4 = "crystal_245_highpass_dnts.tga";
        --splatDetailNormalTex4 = "bump_map_example.dds";
        --detailNormalTex = "twinlakesmodlab_Normal.dds",
        --lightEmissionTex = "",
        },
        splats = {
        texScales = {0.002, 0.004, 0.0052, 0.005},
        texMults   = {0.72, 0.8, 0.65, 0.8}, --fresh snow, cliffs, packed snow,
metalspots
        },
```

Regular Map Texture (Main Diffuse Texture):

Refrain from adding too much ‘noisy’ small details to the map texture in general (like grass, sand dunes), as these will be done with the DNTS splats, much better than the default map texture could ever do.

It is also not recommended to do heavy alpha transitions between materials, they look very fake. Sharp transitions between materials looks best for BAR.This example shows the pixel-sharp transition between grass, rock and snow.



Format:

8bit RGB. Should only ever contain an Alpha channel if you actually wish to use transparency via VoidGround tags.

specularTex

This **texture must be specified for DNTS to be enabled**. At the very least specify a fully black texture.

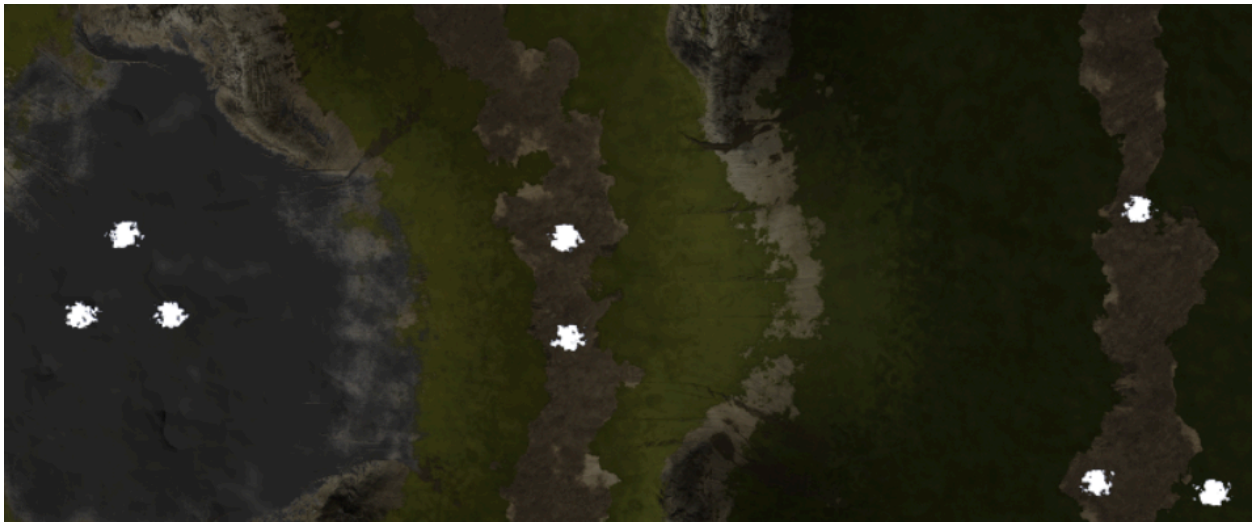
Alpha of 255 means the surface is highly polished and reflects only in a specific direction.
Alpha of 0 means the surface is very ‘matte’ and reflects in all directions. The specular exponent engine side maps alpha [0-255] to [0,16.0], linearly.

This has the unfortunate side effect, that the speculartex is straight ‘emitted’ off of alpha 0 (low exponent) surfaces, so the speculartex must be premultiplied with alpha :D

A good specularTex contains the albedo (unlit color) of the diffuse texture (ideally picture example comparing both) in the RGB channels, and contains some material-dependent alpha channel of shinyness. The RGB should be premultiplied with the alpha channel, avoiding the overexposure of matte surfaces. Best to have picture examples of what it all looks like and how to do the process in photoshop

High values of specularTex alpha is a good way of conveying ‘wetness/polishedness’, with low values meaning ‘dryness/matteness’.

Specular texture example. Note how most areas are very dark, as they will get added to the regular light. Also note that the metal patches are almost full white, but they won't be too bright because their high exponent will prevent overbrightness:



Specular alpha example, high values are high exponent (shiny), moderate values are less polished. Try not to use exponents lower than 16/255, because they will just look like diffuse due to reflecting light in every direction.

Format:

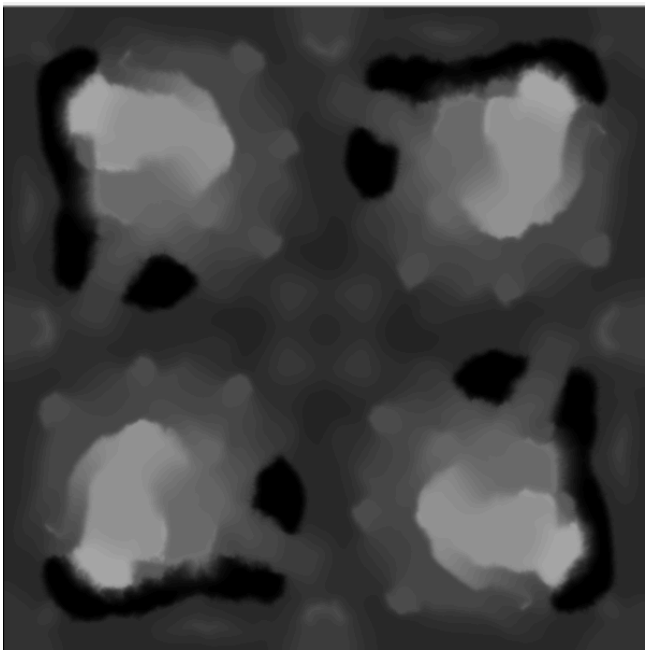
8 bit RGBA. Specularmap is recommended to be about 2k resolution. DDS compression is good, BC3 (DXT5) format, which is good for smooth alpha channels. A 1:1 pixel ratio with diffuse texture can be used, but this becomes prohibitively expensive with large diffuse textures, due to the size of BC3 compression being twice the size of BC1. Flip image vertically for .DDS compression.

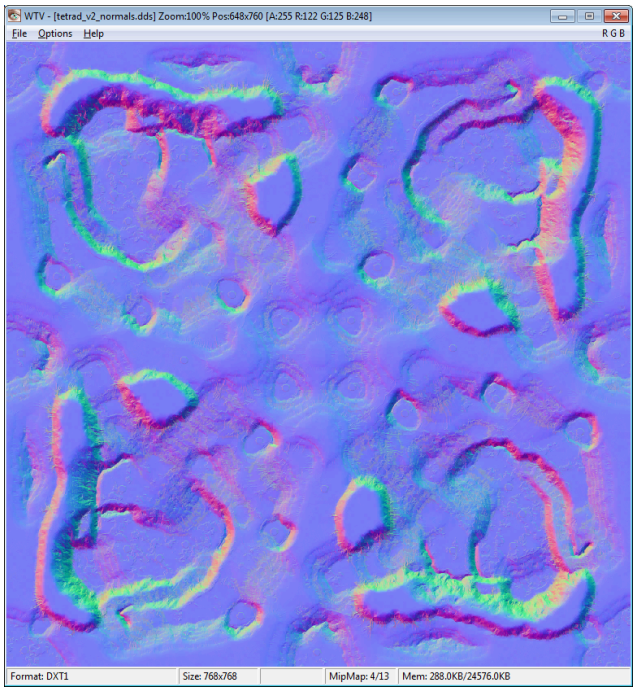
detailNormalTex

This is one of the keys to an excellent looking map. For best results, this should be identically sized to the Diffuse texture.

Example

Say you have the following heightmap:





Note that the corresponding **detailNormalTex is flipped**, and **note the orientation of the green channel** (green is UP)

Format

Recommended format is BC1 (DXT1) no alpha. Remember to: Orient the Y (Green) channel correctly, and to flip the image vertically when creating the .DDS compressed version.

detailTex

Needed for legacy rendering. I recommend sourcing the requisite detailtexblurred.bmp from one of my maps.

splatDetailTex

A fallback rendering mode, never used and no longer recommended, but must be specified for DNTS to work.

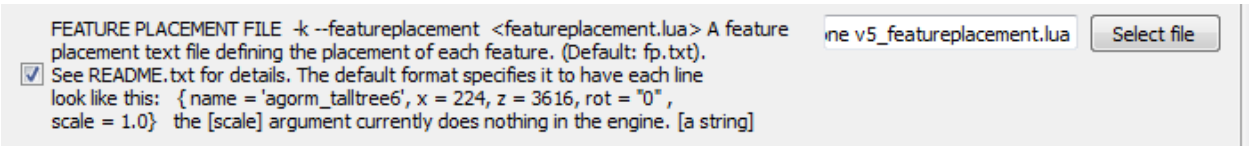
splatDetailNormalDiffuseAlpha

This boolean mapinfo.lua variable allows you to enable/disable the diffuse color packed into the alpha channels of the splatDetailTextures.

Feature Placement

Geo Vents

Geovents should be placed within PyMapConv, as it will perform the drawing of the geovent image as well. Use the relevant part of pymapconv to do this, and pass your lua file to it:



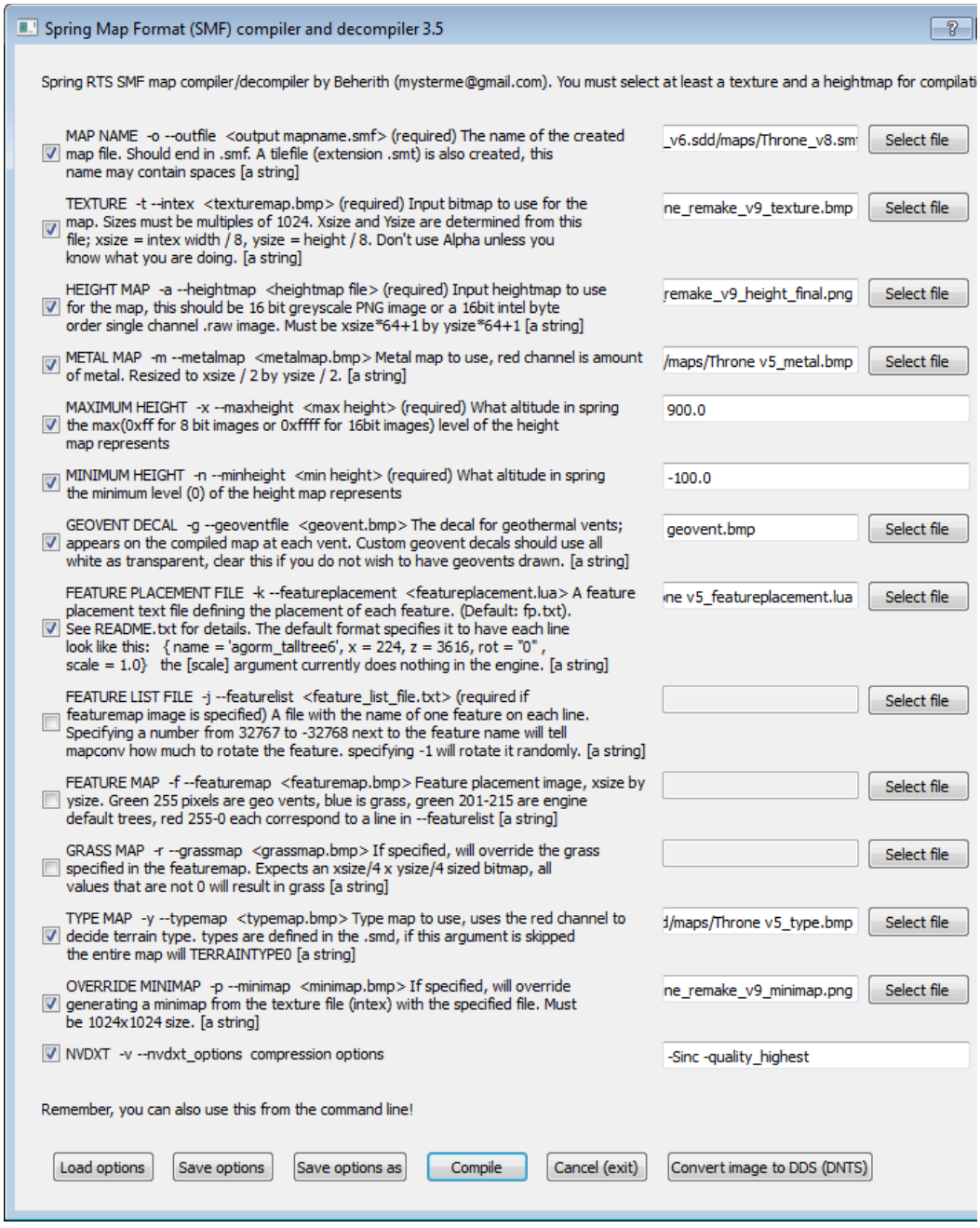
{ name = 'GeoVent', x = 5428, z = 1492, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 9332, z = 2668, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 2428, z = 3508, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 10268, z = 6524, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 2124, z = 7396, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 8492, z = 9940, rot = "0" ,scale = 1.000000 },
{ name = 'GeoVent', x = 4812, z = 10324, rot = "0" ,scale = 1.000000 },

Other Features

See the relevant info in the SpringBoard part, as that is the recommended method of placing features for a new map. You can also use the featureplacer method.

Compilation

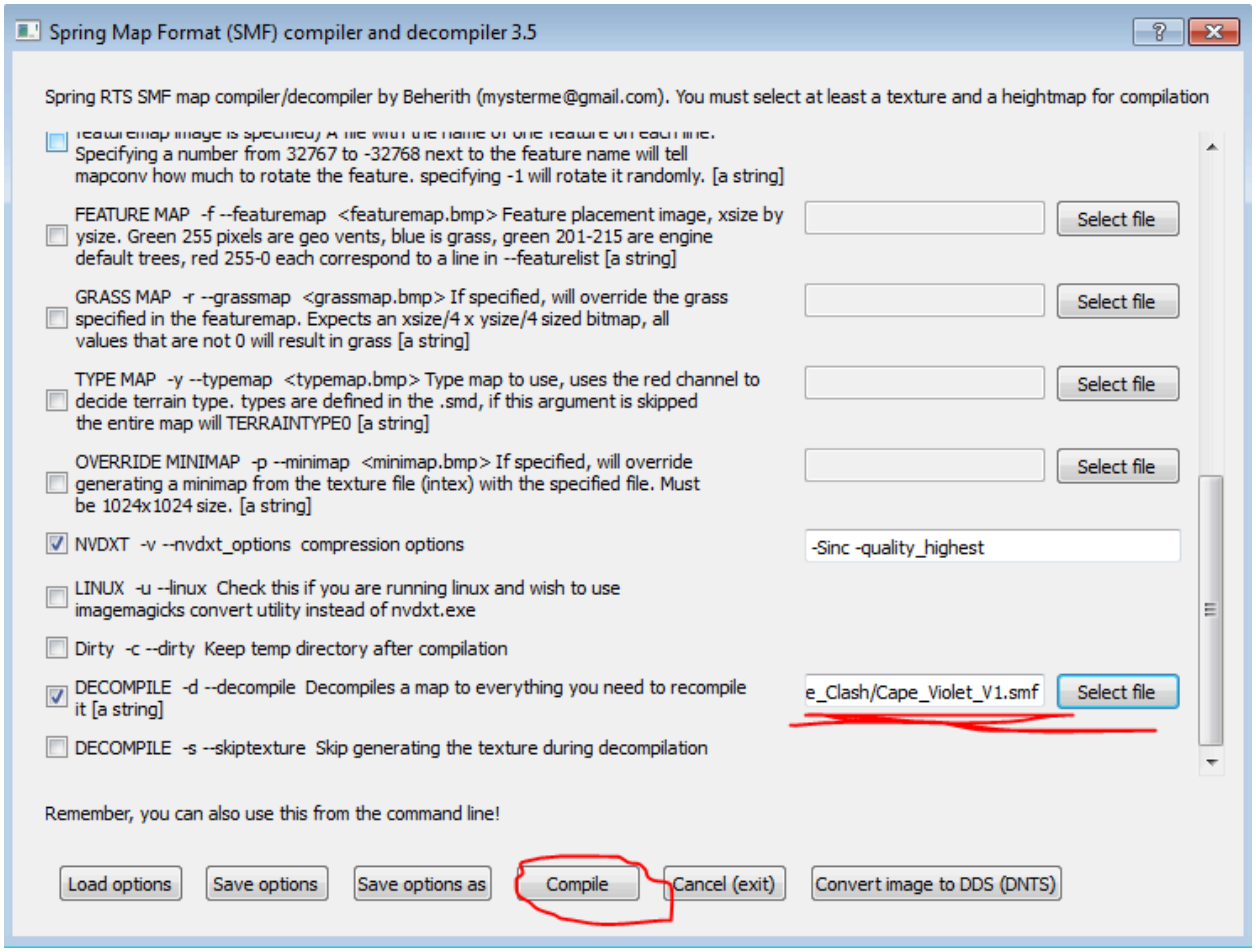
You need at least a main texture and a heightmap to compile a map.
Remember that you can save and load compilation settings using the buttons on the bottom.



Z

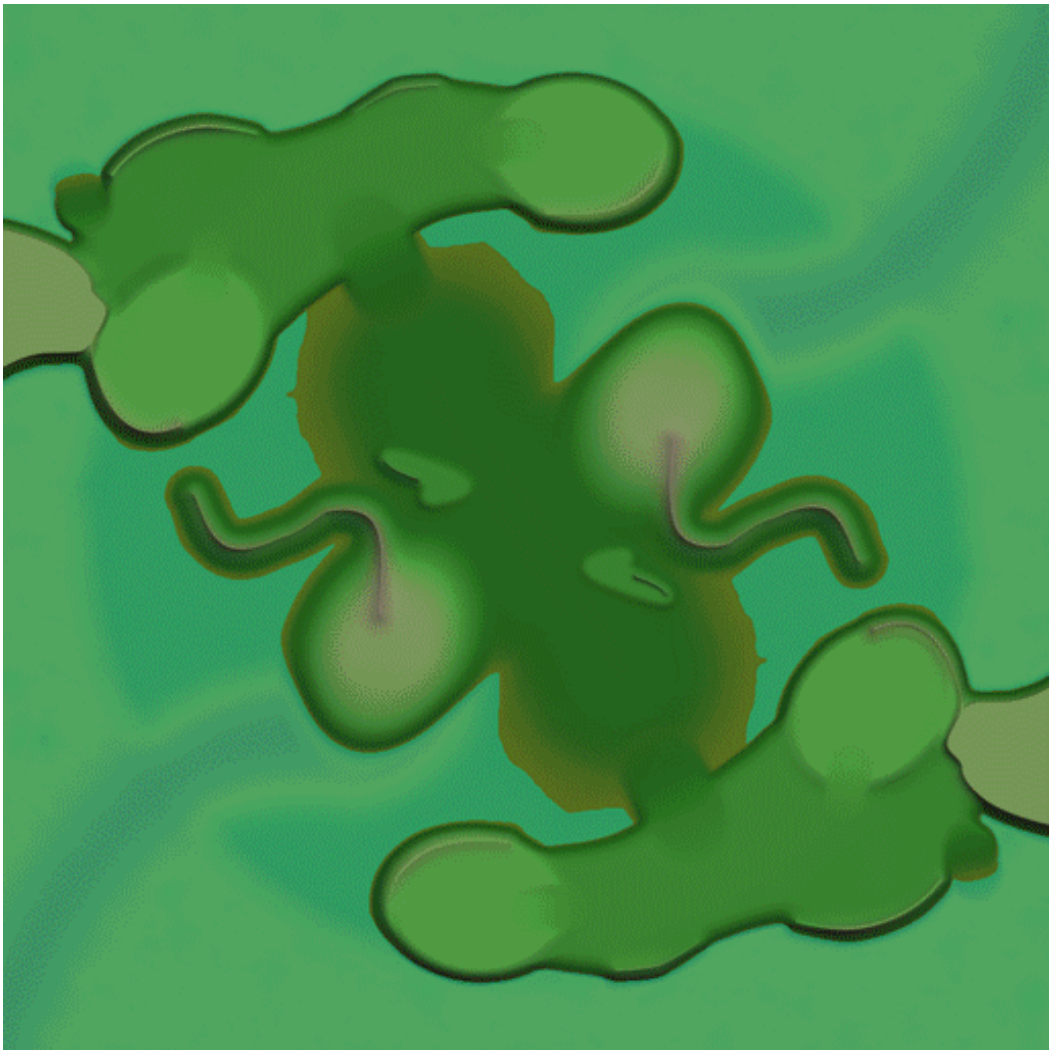
Decompilation

In order to change or update an existing map, you need to un-7zip it (open the .sdz or .sd7 file with 7zip) to a folder somewhere. Then start pymapconv, scroll down to the DECOMPILE option, select the .SMF file from the unzipped /maps/ folder, and click compile.



Mapinfo.lua

For now, refer to
And also, note the rather well documented mapinfo.lua files included in my maps.
Lighting



Map creation is an iterative process. Thanks to Angelwings for the gif of his progress.

World Machine Usage

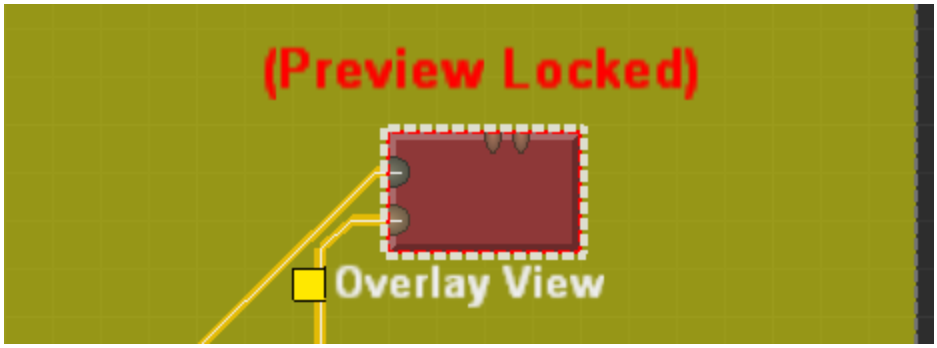
This tutorial is designed for World Machine build 3026 and up. I will abbreviate **World Machine as WM.** All text marked **bold** appears exactly as the WM user interface. Please take the time to familiarize yourself a bit with how it works, read up or watch some tutorials. The WM manual is itself an excellent and short introduction as to how it works. WM is a procedural tool, so all devices (nodes in the device graph) operate on the entire map at once, with a few exceptions, and we will use masks to select areas to operate on. Sample WM file available here: https://github.com/Beherith/springrts_smf_compiler (Crescent Bay Clean V3.tmd)

In the WM menu, go to **Views -> Open new top-level window**, and move that window to the right, while keeping the **Device View** on the left as this will allow you to see a rapid preview of your work. In this new window, select **Layout View** using the tabs at the top.

Selecting any device on the left **Device View**, will immediately start to render a preview on the right **Layout View.**

Very Important UI shortcuts

Selecting a device and hitting the **(F) key** will **Lock Preview** onto that device. This means that all views will show the output of that device, even if you select any other device. This allows you to view the effect of changing things in a previous device while showing the output of the locked device. For example, you can see how changing the sand mask areas will affect your final output directly, by locking onto the final output.



Set up map size

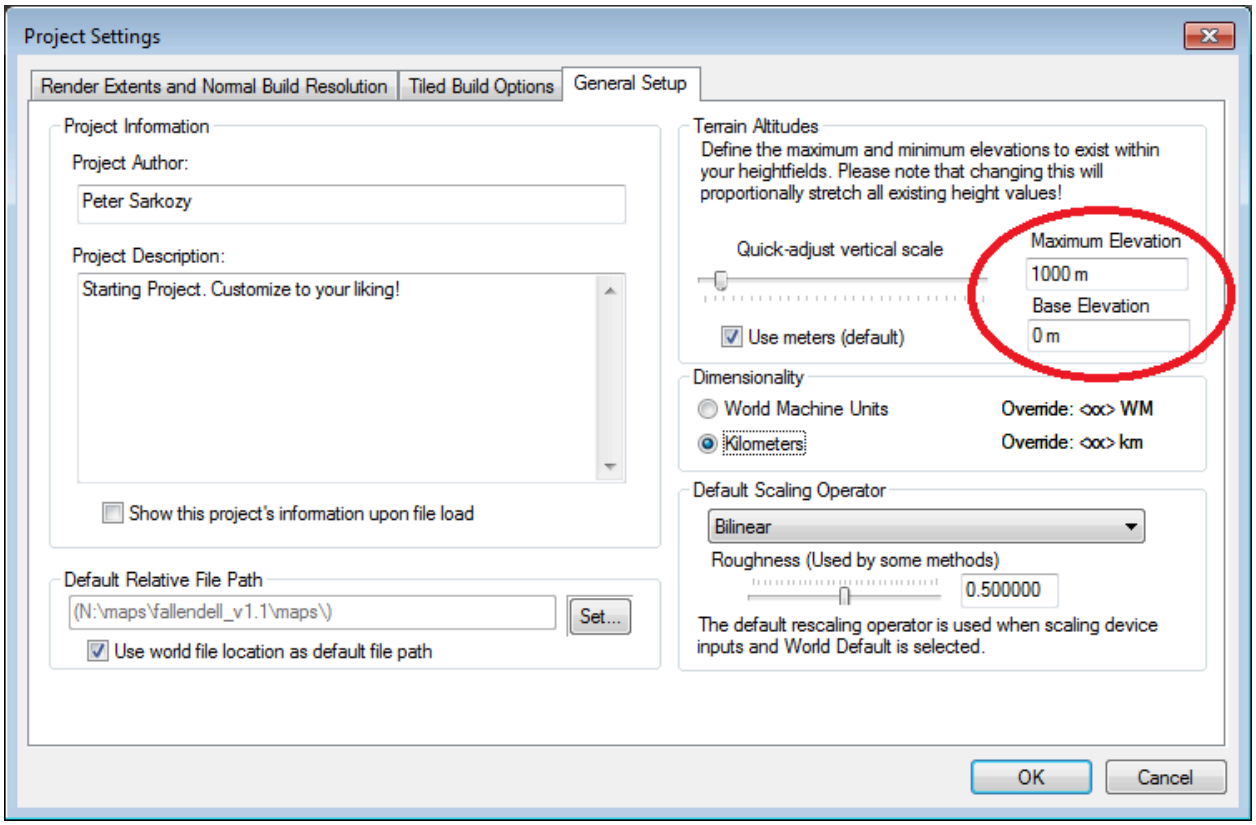
Scaling

We will , that 1 meter in World Machine will be approximately equal to 1 elmo (1 elmo is 1 ‘unit’ in Spring, which you can see in game in the tooltip). In World Machine, we will set up the world **render extents** to X by Y kilometers, where 1 kilometer is exactly 1024 elmos. This conveniently approximates the internal representation that Spring uses. You must use whole kilometers (multiples of 1024 elmos) for maps.

The first step in all mapping, is deciding on the size of the map that you wish to make. In general, Spring maps use an internal size calculation, where your average 16x16 sized map will be exactly 8192 x 8192 elmos in size, 8K by 8K for short, and in WM, we will set this as 8 kilometers by 8 kilometers.

Avoid making a map that is larger than 32x32 spring, 16K x 16K size, as the engine (nor gameplay) doesnt like it much.

In the WM menu, **World Commands -> Project World Parameters**, we are setting up the size of our map. In the above example, we are making a 12x10 Spring map, which will equate to 6km by 5km. Our final output resolution will thus be 6144 x 5120 (e.g. 6*1024 x 5 * 1024). You can change the **width** and the **height** of the map, but also specify the world coordinates. The resolution specified here in the normal build resolution can be used for medium-resolution previews of your map, but will be very limited by the amount of RAM you have.



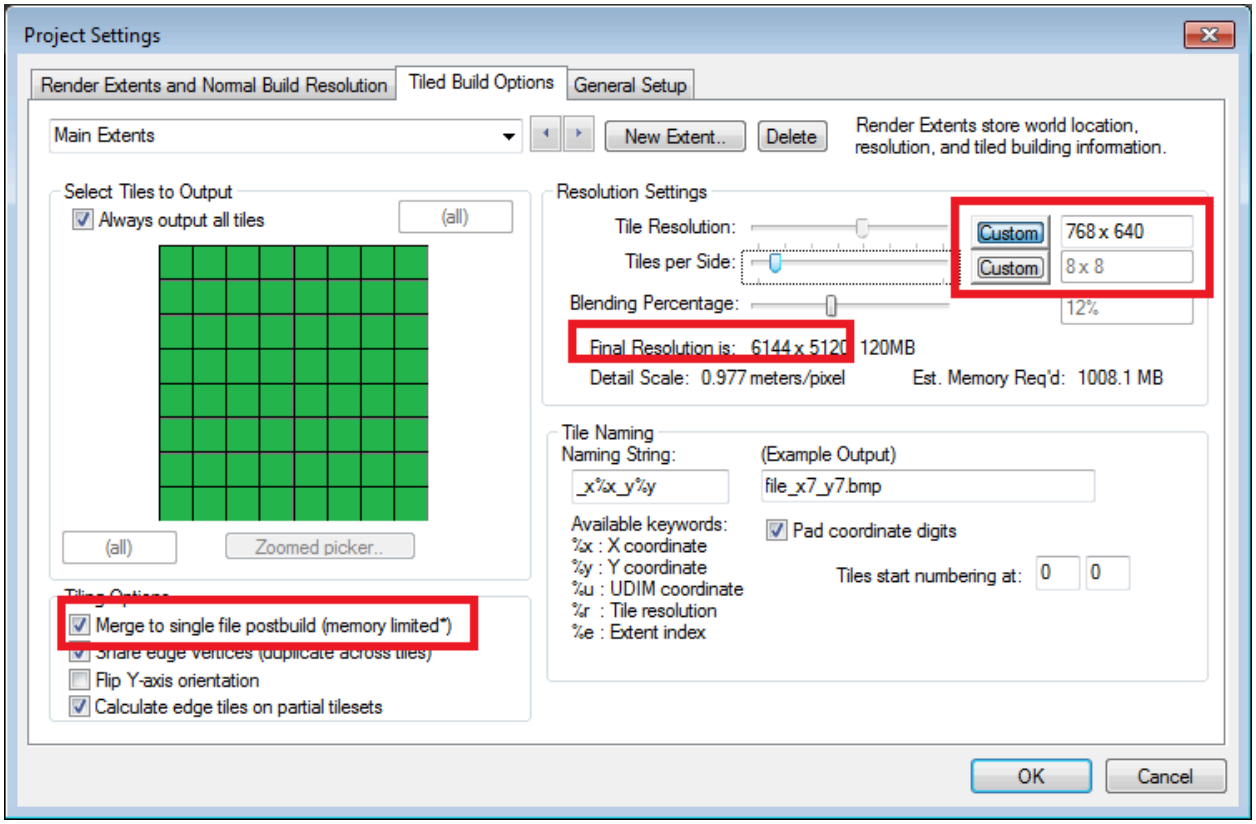
In the **General Setup** tab, we can adjust the **Terrain Altitudes** to more closely resemble what we might encounter ingame. I recommend setting the **Maximum Elevation** to 1000m, and the **Base Elevation** to 0m. This will give us an elevation range of 1km to work with, which will be plenty, and remember that 1 meter in WM will be 1 elmo in Spring. The ingame tooltip in the bottom left will show positions and altitude in elmos.

Altitude Scaling

What we set up in the previous section is just a general guideline, and will limit our working height range to 1000 elmos (1000m in WM). Obviously this is more than what the usual map uses, and if you want to use a smaller height range, I recommend using **Clamp devices**, Keeping the 1000m total height will keep things tidy. This may mean that your final heightmap won't use the full black-white range, but this is not an issue.

Setting up tiled builds

Tiled builds are essential for making anything but the smallest maps without running out of RAM. In order to make sure that our output resolutions are correct, set **Tiles per Side to 8x8,** and set the **Tile Resolution** with the slider or the **Custom** button to ensure that the **Final Resolution** is what you desire. Make sure that **Merge to single file postbuild** is checked.



Render Extents

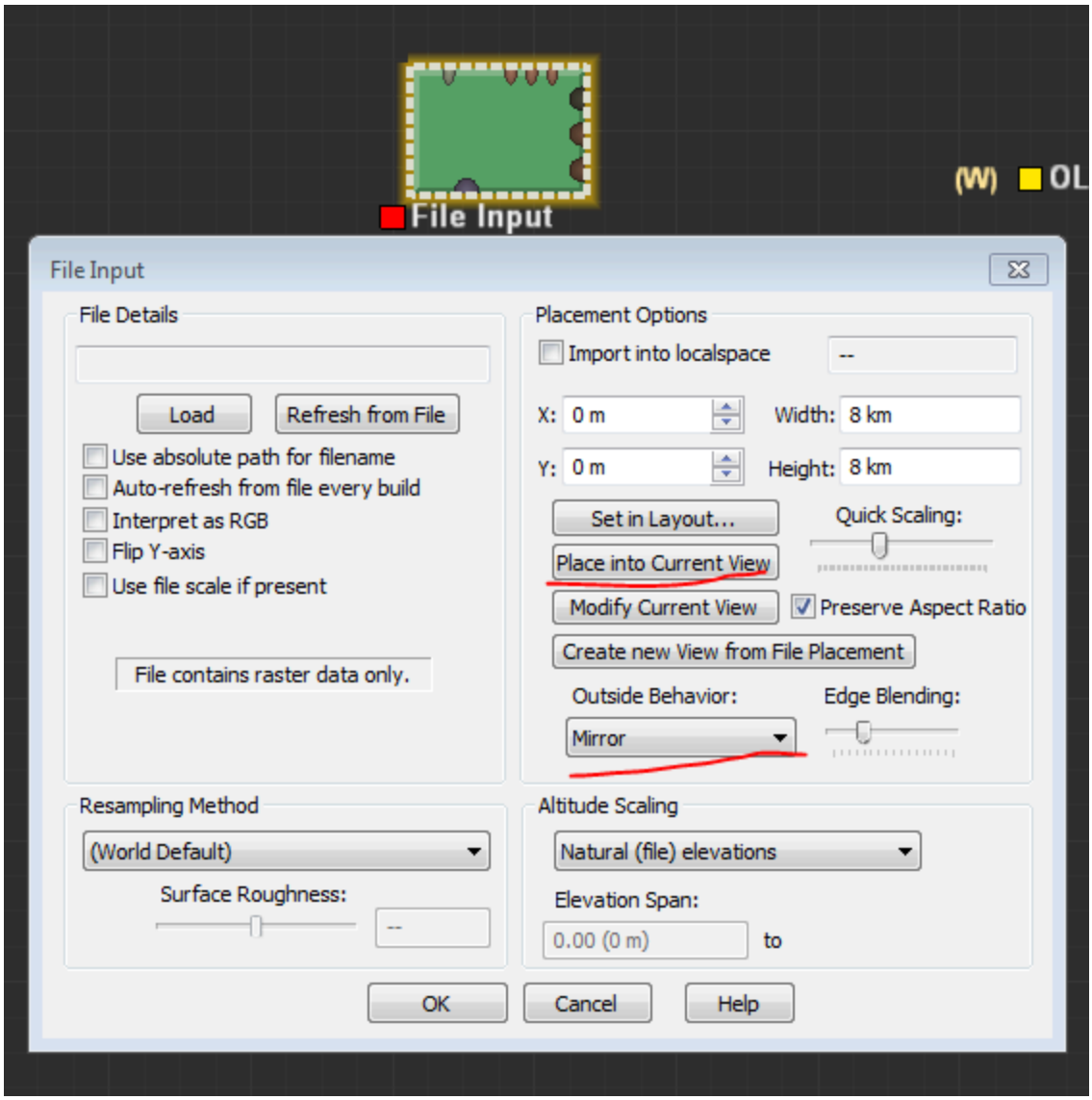
You can set up additional **Render Extents**, if you need full resolution previews of small areas of the map, but they are not required.

Creating Your Heightmap

Import existing heightmap

Start off by importing an existing heightmap by adding **File Input** device, from the **Generator** tab group.

Double click the device, and you are greeted with the file input dialog:

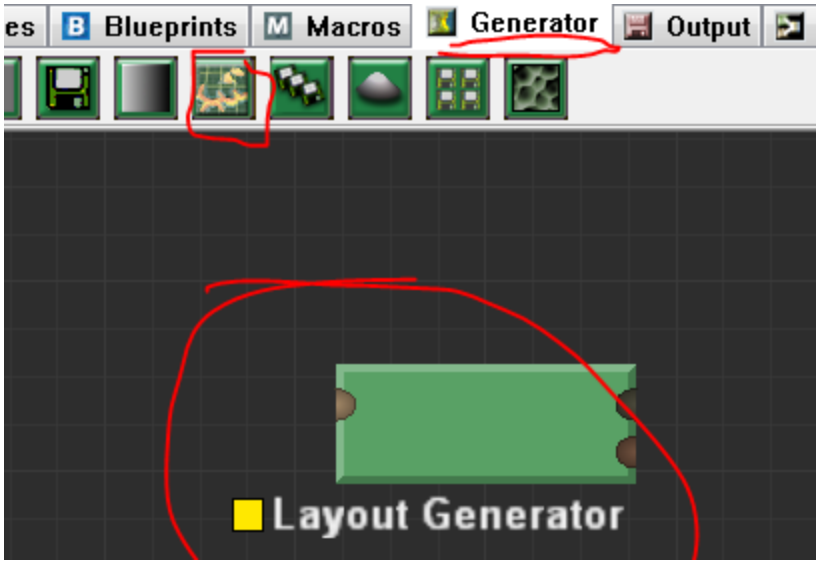


Click on **Place into Current view** to center the imported heightmap in your world's render extents. Also click on **Outside Behaviour : Mirror** if you want to avoid artifacts on the edges.

To take a look at what your heightmap looks like, select it in the **Device view**, and it should start to render in the **Layout View.** Excellent!

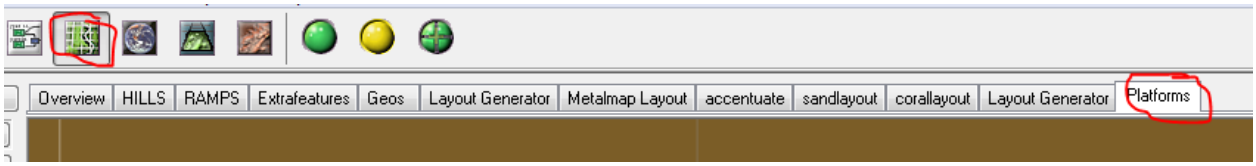
Layout Mode: Create or modify an existing heightmap

Start off by placing a **Layout Device** in the **Generators** into your **Device view:**

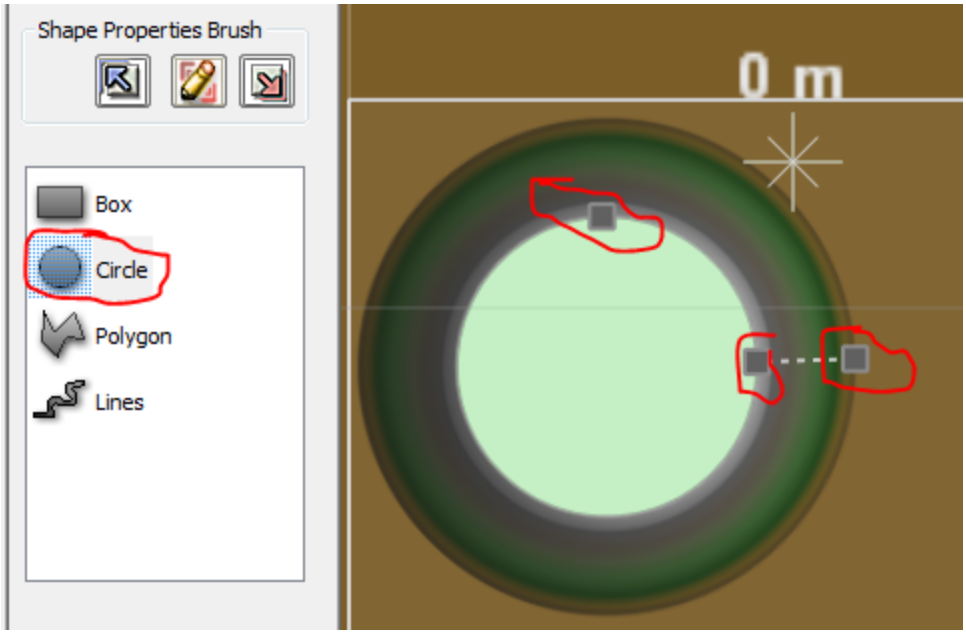


Select this new **Layout Generator**, right click it, and **rename** it to something clever like Platforms**.** Keep it selected.

In the **Layout view**, select this Platforms tab from the list of tabs. You will now be placing flat platforms onto this.

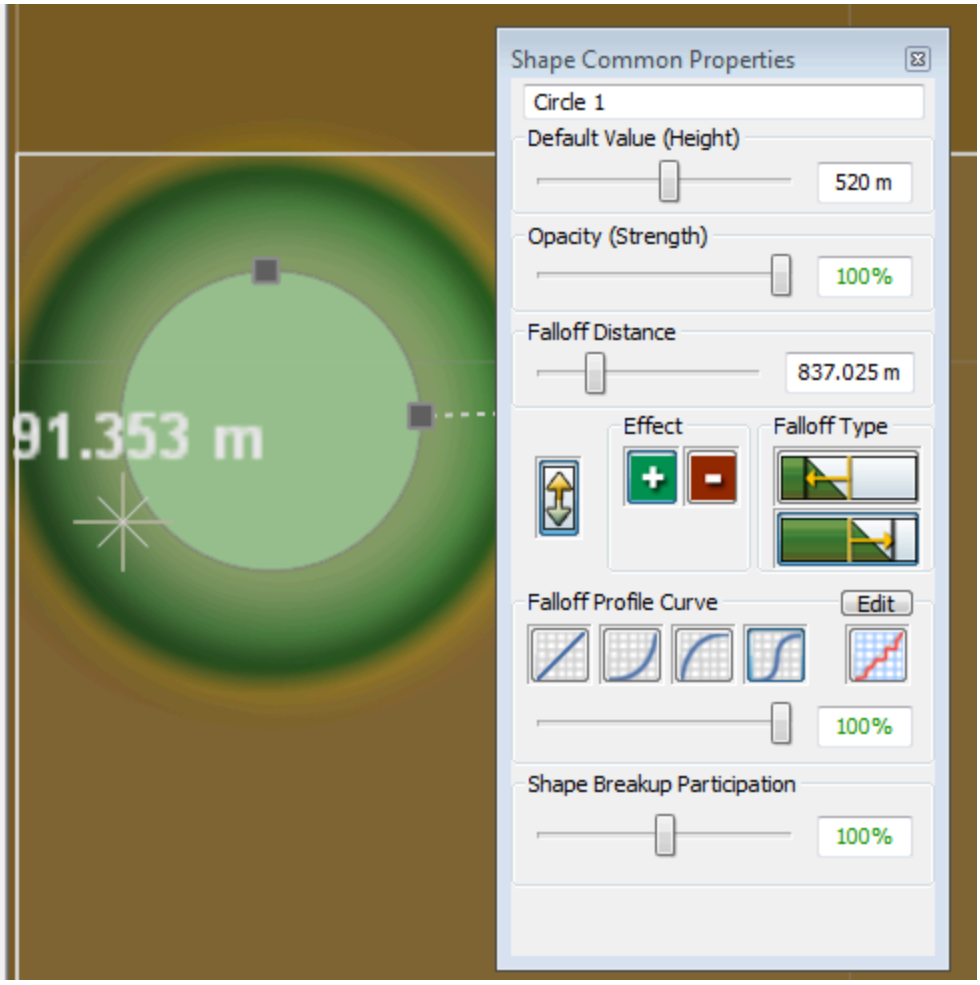


Place your first hill from the shapes into the highlighted area of your layout:



You can change the shape of the circle with these handles.

To change the slope type, falloff distance, absolute height, or any other parameter of this shape, double click the circle to bring up the **Shape Properties** dialog:

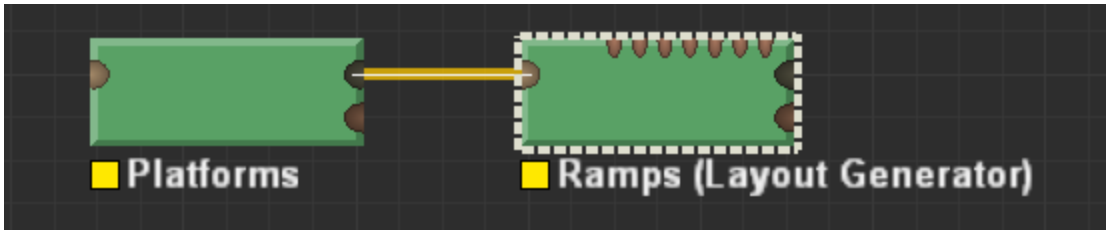


Play with these to get a feel for what they do, but lets make the **Falloff Distance** of this circle very small, so we can later add a ramp to this platform.

Also play around with polygons, boxes, bezier polygons, and lines!

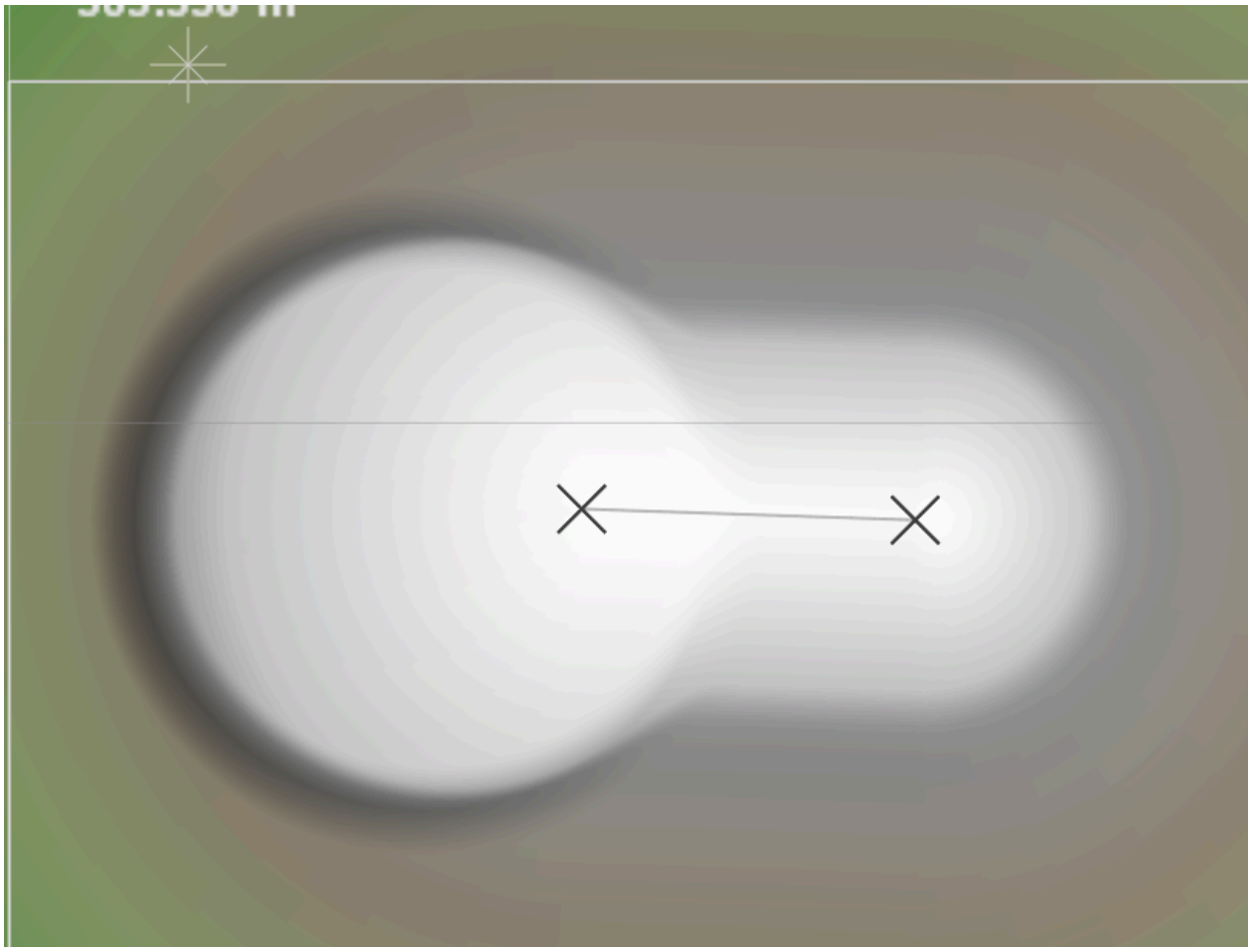
Making Ramps

Making good, pathable ramps in WM is easy, but does involve a bit of setup, but we only have to do it once, and then all our ramps will be perfect!
Add a new **Layout Generator** to your device graph, and name it Ramps, and connect it to your Platforms **Layout Generator:**

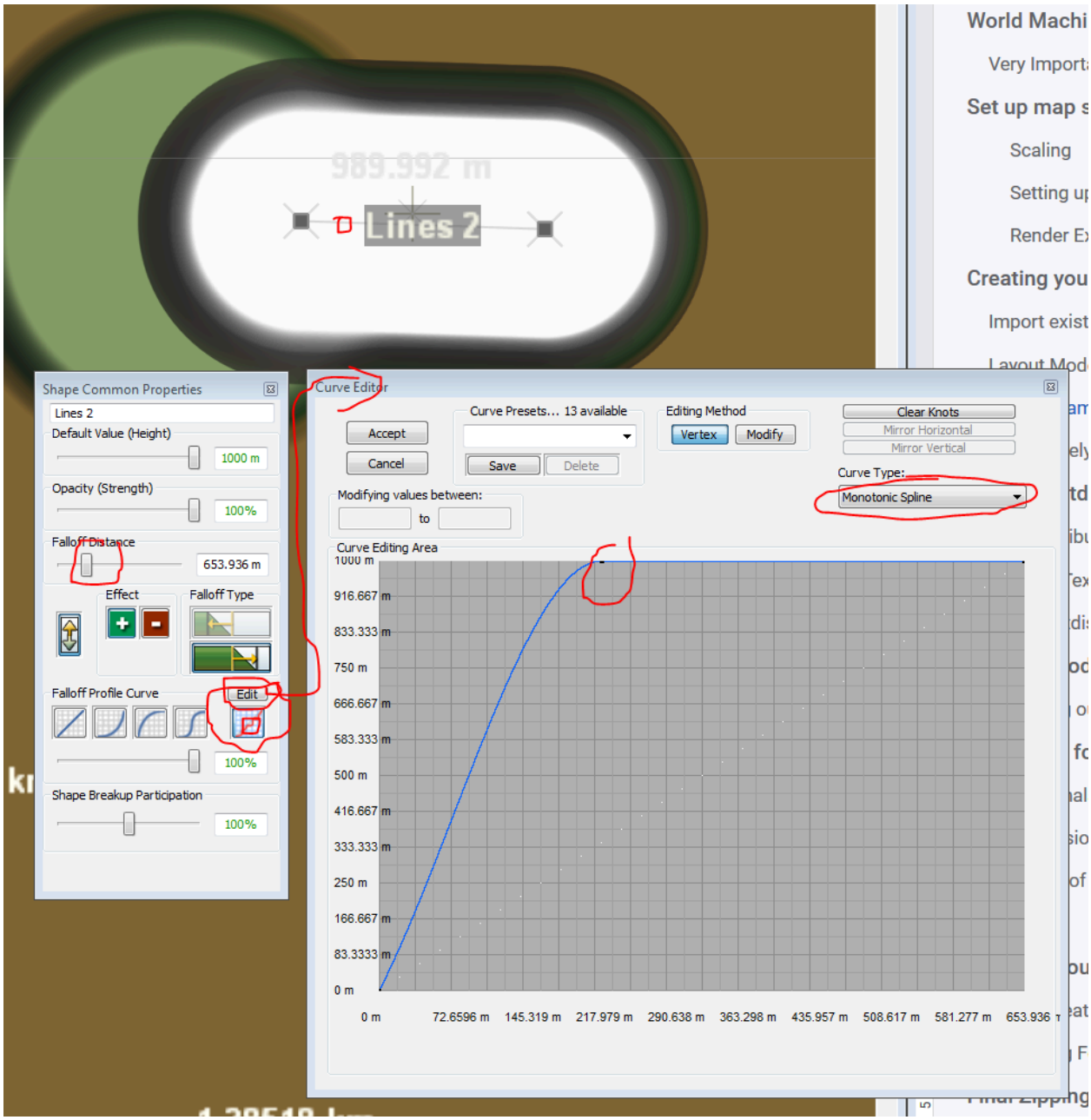


Go into **Layout View,** and select the Ramps tab, and zoom in on the circle you placed:

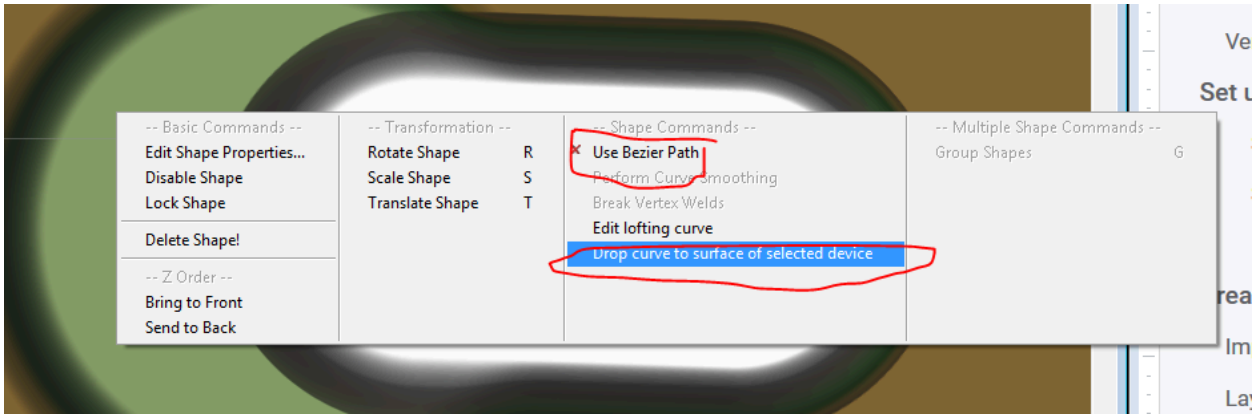
Good, now add a line consisting of just a start and end point of where you want your ramp to start and end:



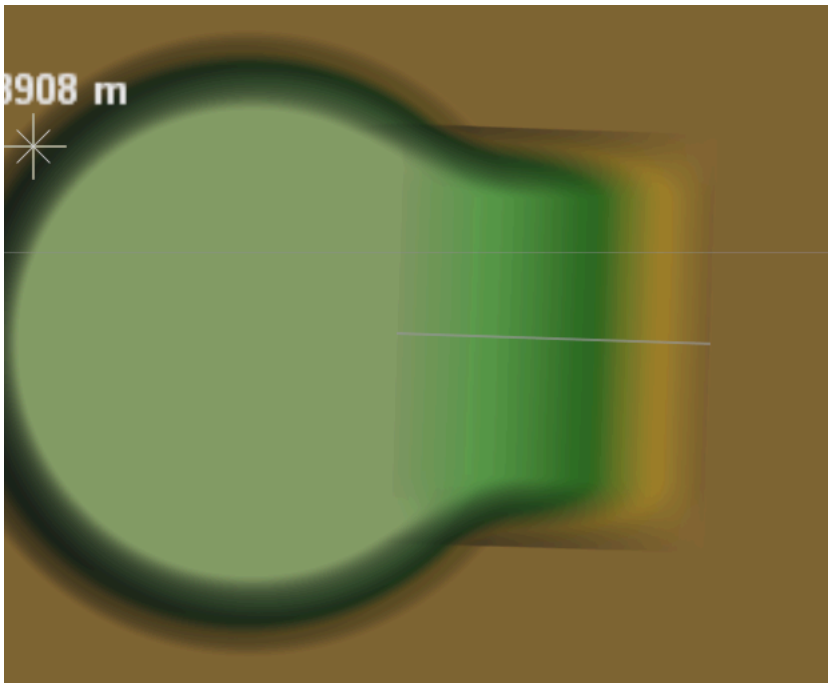
Ouch, that looks bad, but lets double click the line to bring up the **Shape Properties** dialog, and set the **Falloff distance** to something small. Set the **Falloff Profile Curve** to the custom button, and click the **Edit** button on the **falloff profile curve.** Now you can Set the falloff of this curve to a ramp-like profile by clicking inside the curve area and selecting **Monotonic Spline** from the **Curve Type.**



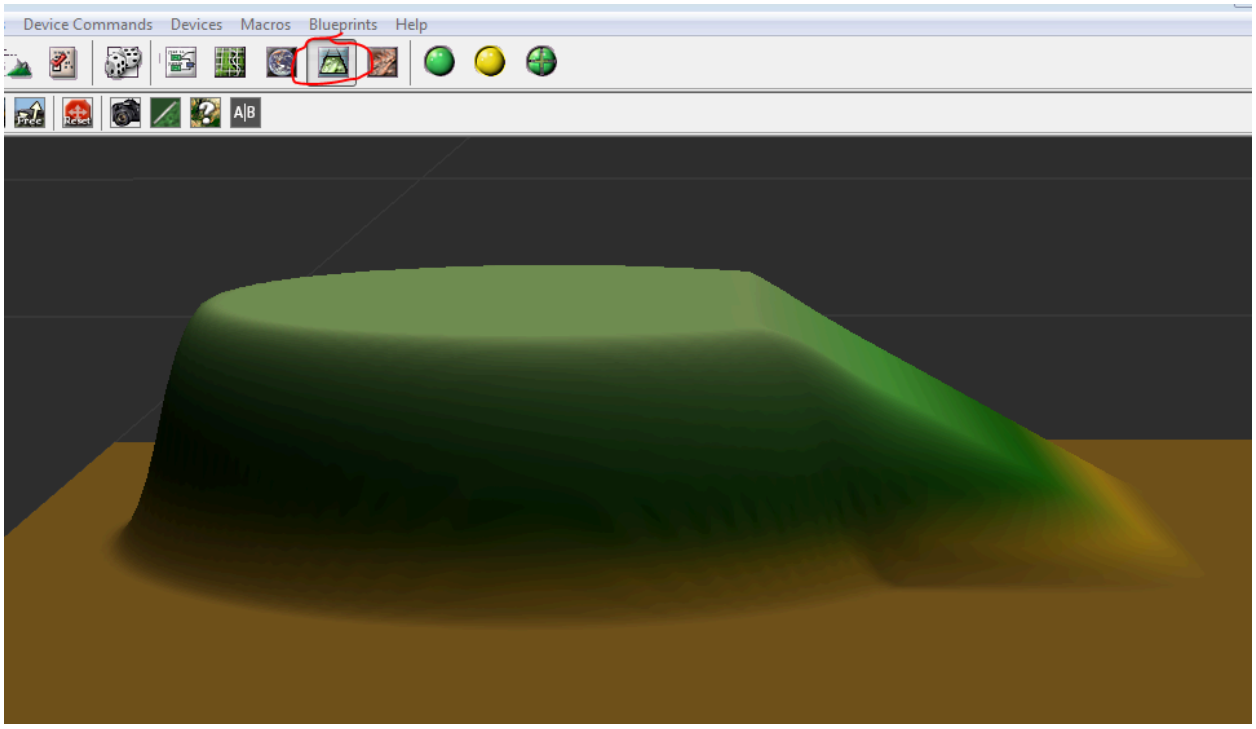
Our last step is to right click on our Line once more, and click **Drop curve to surface of selected device (note that you must have this Layout Generator selected in Device view for this to work as intended, see the extremely important chapter below to find out why).** You can also play with **Use Bezier Path.**



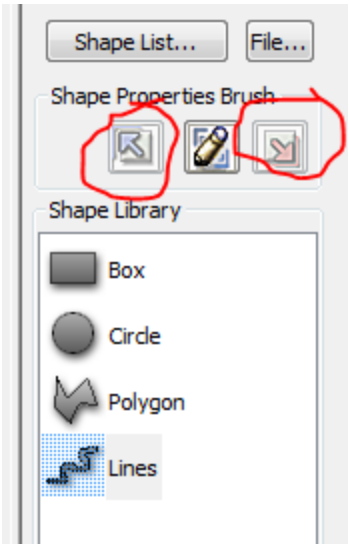
The result, a PERFECT linear ramp!



Check it out in 3D view too:



You can always use the **Shape Properties brush** to copy-paste this ramp style to any Line:



Extremely important Layout Mode info:

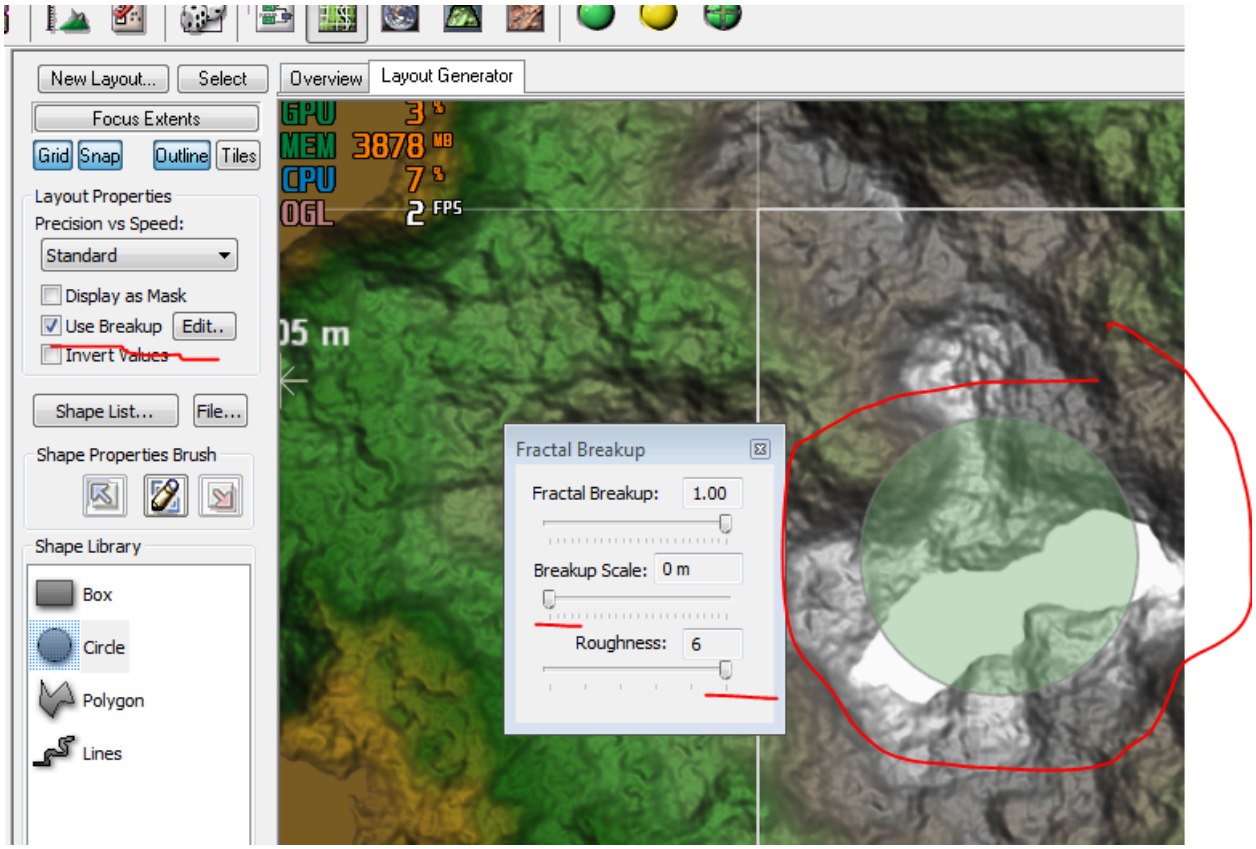
What you see in layout mode is the composite of two things:

1. The output of whatever **device** you have selected in **Device view**.
2. The vector graphics (layout) view of whichever **tab** you have selected in **Layout View.**

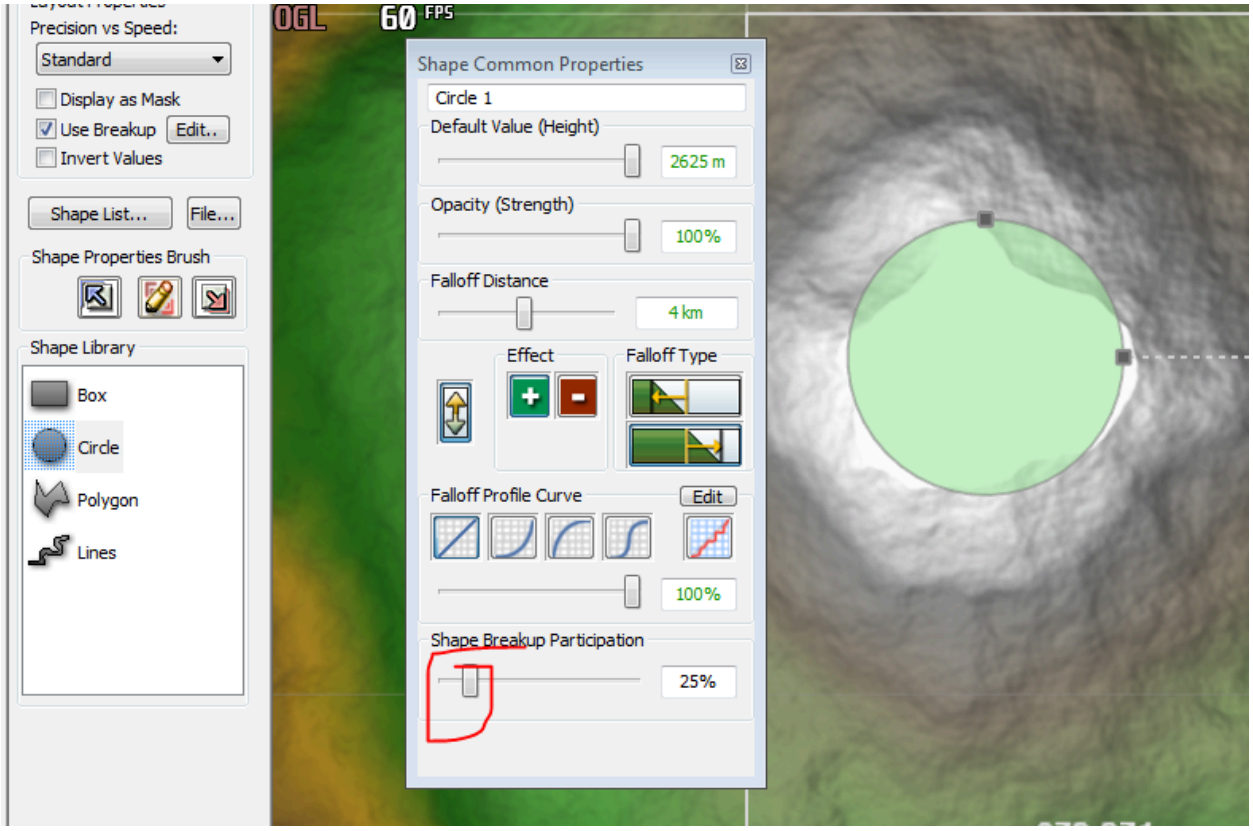
When you place your cursor into a position in the Layout view, it will show you the exact height at that position.

Breaking up Layout Mode unnatural shapes

Often you'll get unnaturally regular shapes while working with layout generators. Thankfully WM offers easy tools to apply fractal breakup to a layout generator:



You can then control the amount of breakup each shape receives with the slider at the bottom of the shape properties brush seen when you edit the properties of a shape.

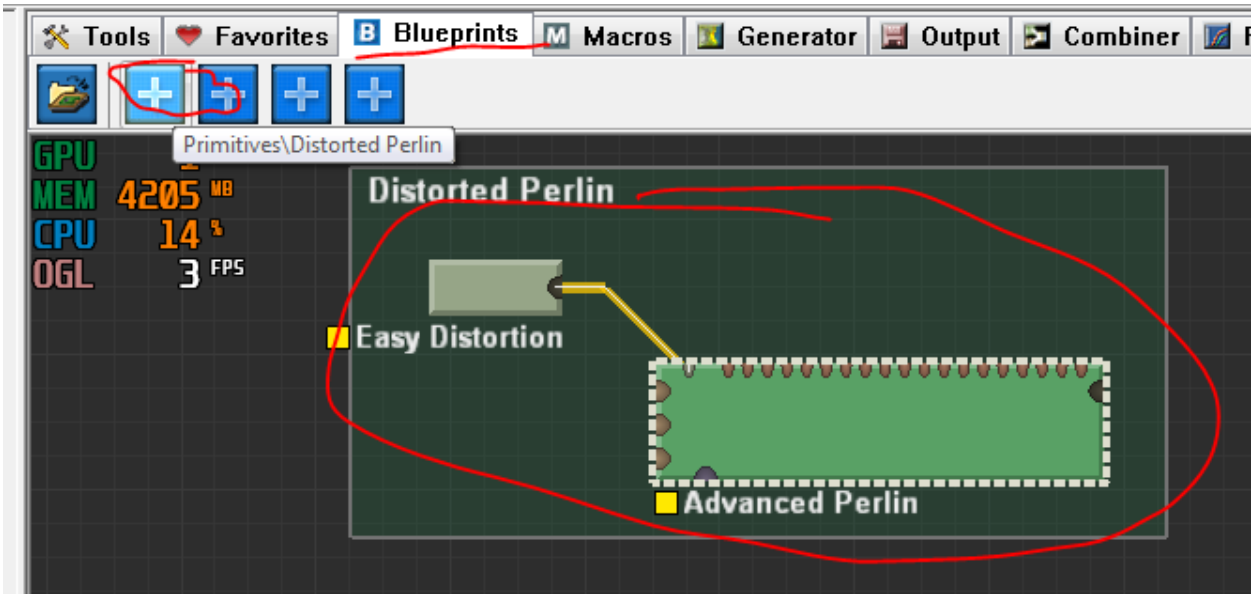


Adding detail

At this point you can use the full arsenal of WM to apply any amount of distortion, perlin noise, erosion or thermal weathering to your map, the more different kinds you use, the better.

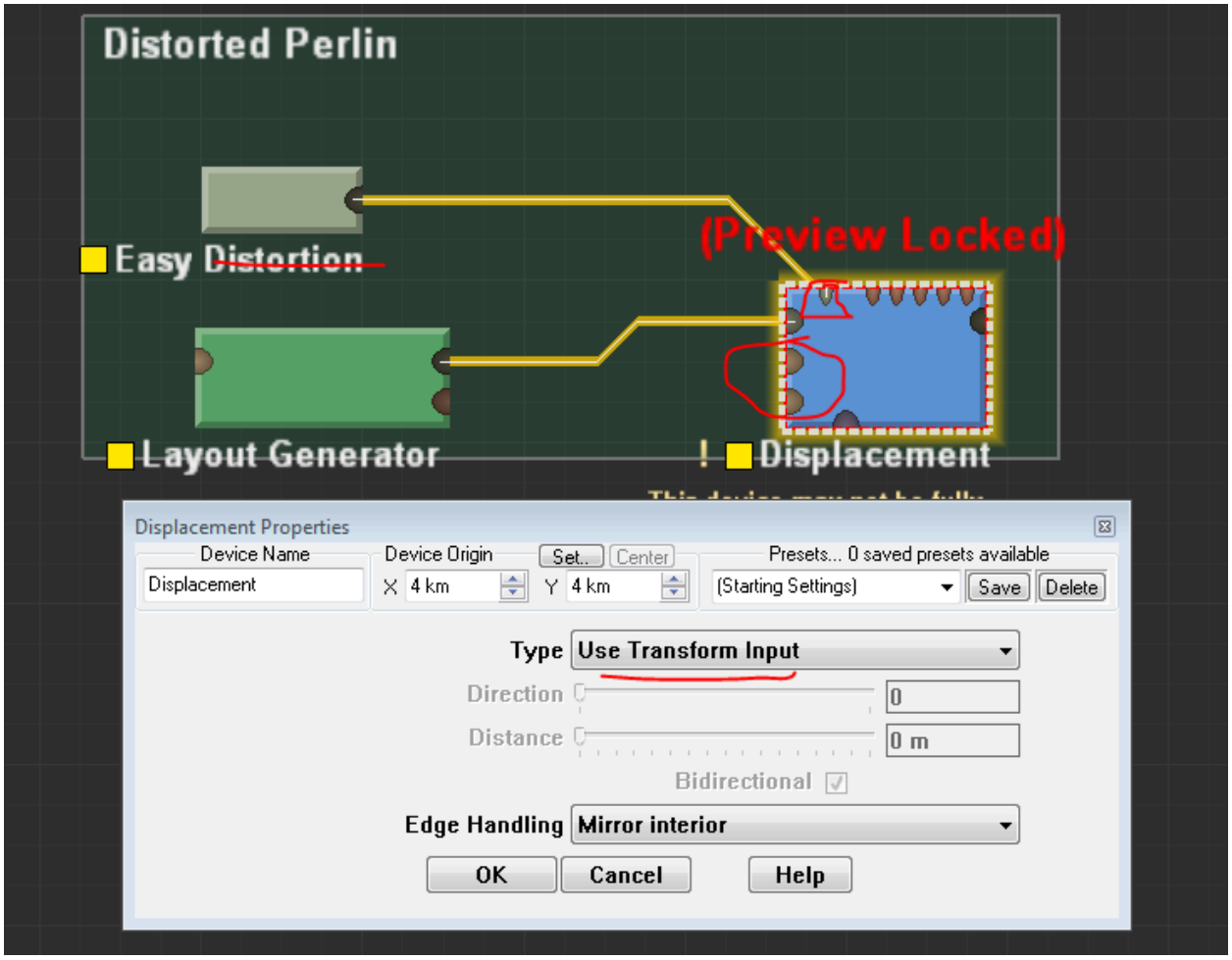
Distorted perlin noise generators

These should be some of your best friends when you want nice, varied and natural looking perlin noise. Combine these to your heart's content with your terrain for nice effects.



Displacement Devices

These can be used to great effect when breaking up steep cliff edges in a million possible ways:

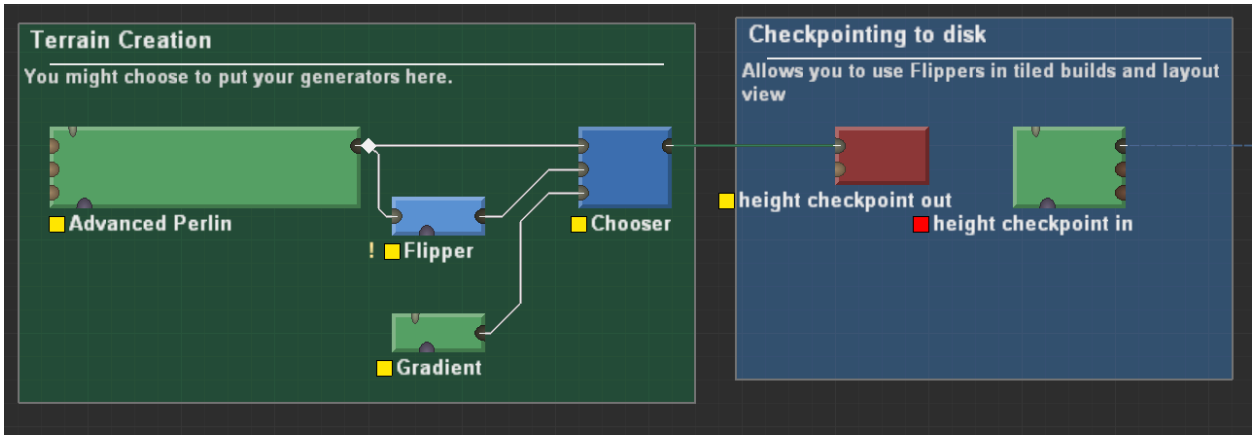


Verifying passability for kbots and vehicles

Use the Spring Passability Map macro for this. Red marks impassable, blue is kbot only, yellow is hovers/amphib tanks and green is passable by all.

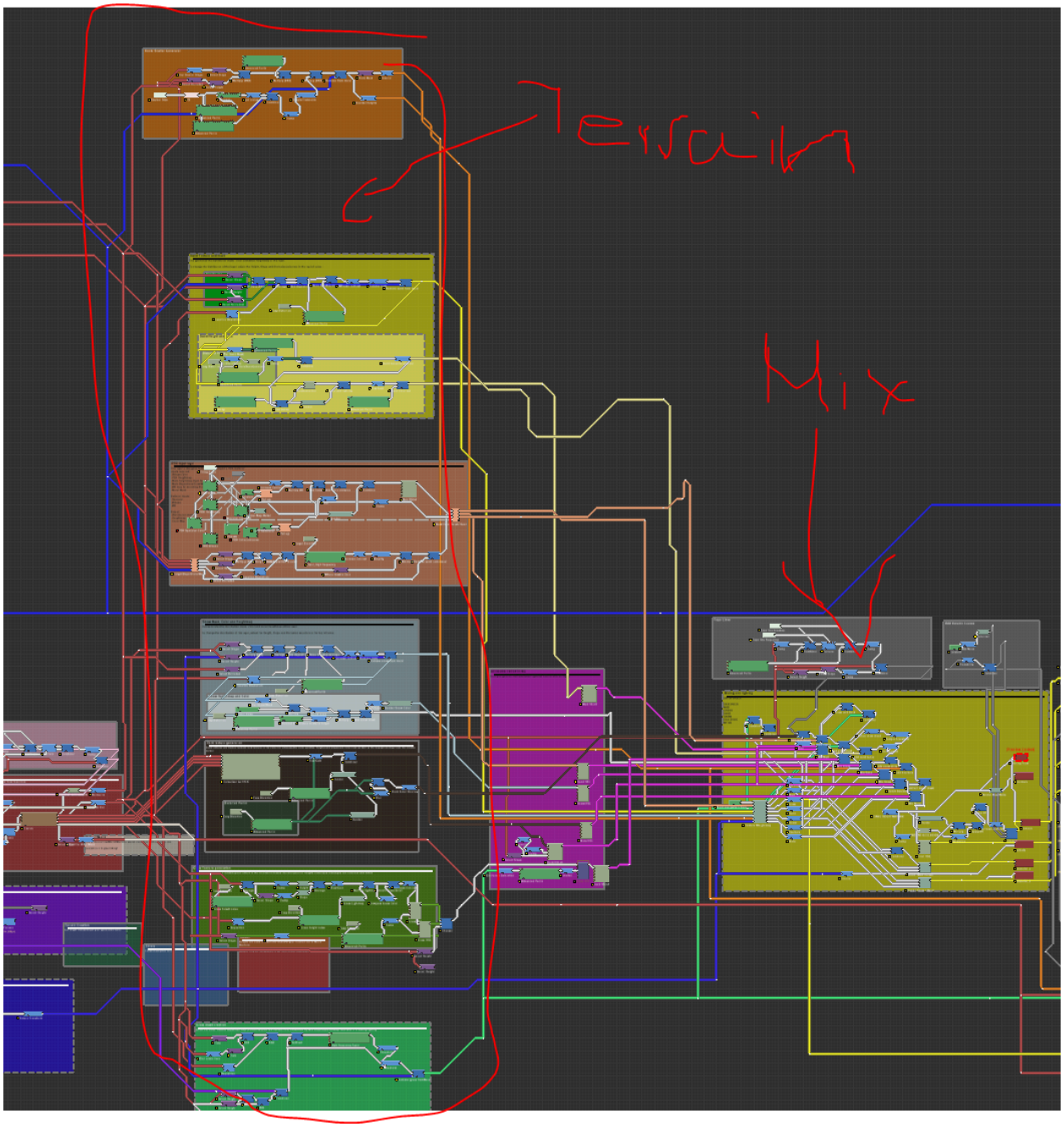
Making a symmetrical map

Some devices in WM are so context dependent, like the flipper device, that they won't work well in layout mode. Often the solution to using these devices is building them in one go (no tiling!) and checkpointing them to disk. Making a symmetrical map involves taking your original heightmap, flipping it, then combining it with the original.



Texturing

Welcome to the meat of the tutorial, where we will investigate the individual different kinds of terrain layers we can adjust on the map.
The worlds (.tmd files) you see here are included in all of my map releases, which are available on http://springfiles.springrts.com/ Or you can use Crescent Bay Clean v3 from this repo: https://github.com/Beherith/springrts_smf_compiler



You can quickly see, that there is a large stack of groups of devices, each with a different color. These mostly correspond to individual terrain type layers, like rock, sand, mud, grass, snow and metal.

These groups share the following inputs:

- Heightmap
- Deposition mask (from the Erosion device, to show where ground deposits from erosion)
- Occlusion mask - this indicates how much an area is occluded from global light

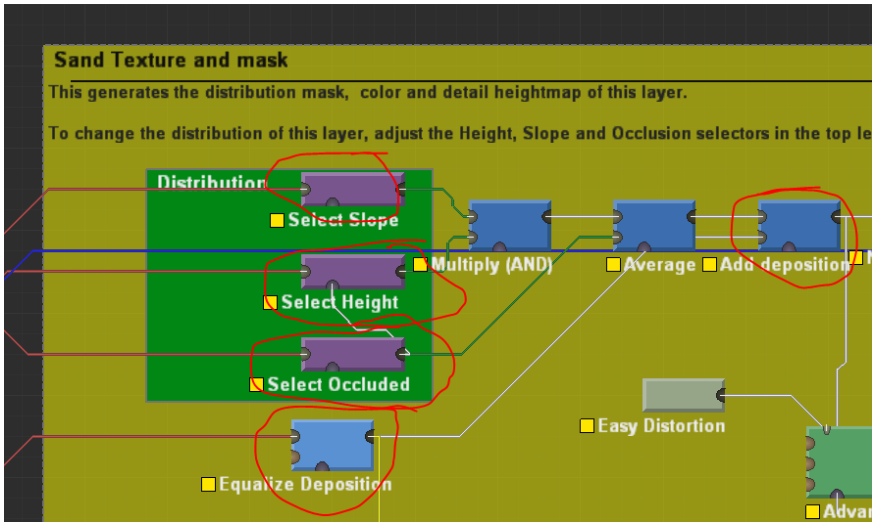
- Metal Mask - this shows where we will have metal spots, this one isn't strictly needed, but useful in general

And generates 3 outputs:

- A colored texture for that terrain layer
- A distribution for that terrain layer
- A detail (very small variations) heightmap layer that will be used to generate lighting and normals

Layer Distribution

Most layers can be adjusted, as to where they can occur on the map, at the start of each terrain layer box:

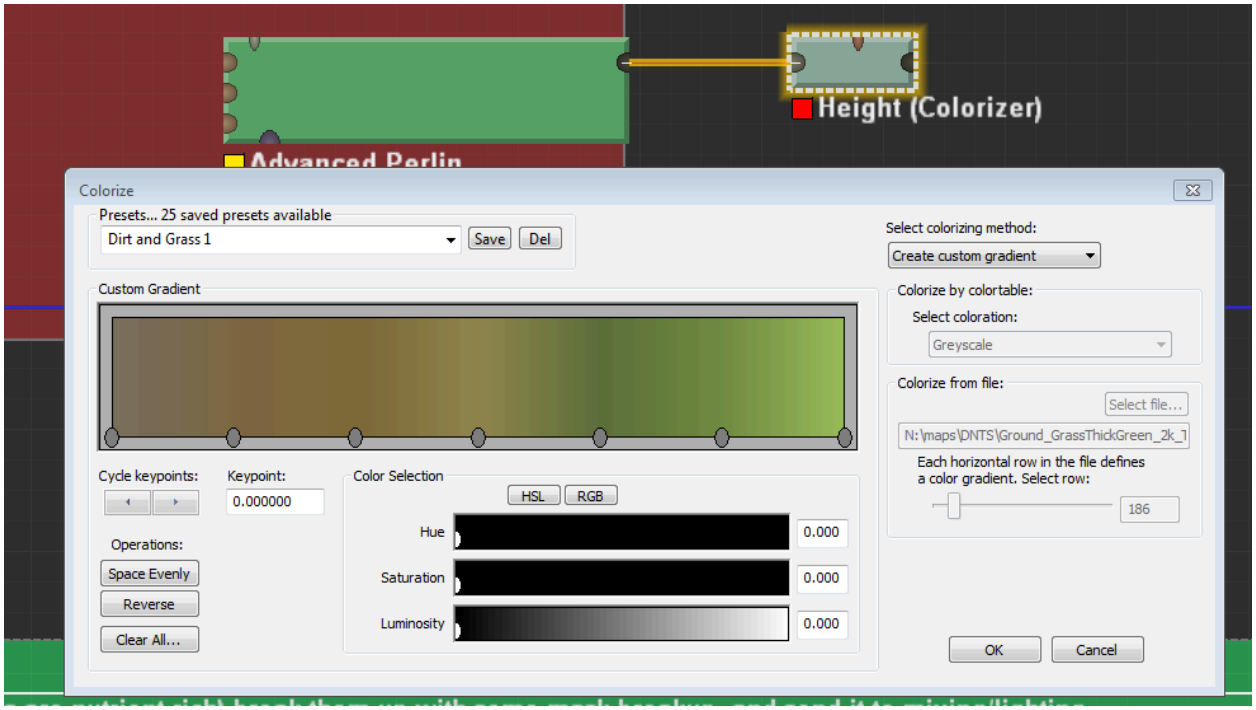


You can change the slopes, heights, occlusion, and deposition factors of them, note that they are usually designed to be AND masks. So that a part of the map must satisfy all 3 requirements of being the selected slope, selected height, and selected occlusion, and can optionally be modulated with deposition.

These selections are then multiplied together, and often broken up with a bit of high-frequency noise, to make them a bit more natural. Note that we like to use 'sharp' transitions between terrain types, as smooth alpha transitions usually look bad. Grass doesn't gradually 'blend' into the side of a mountain, there's a sharp point at where it ends.

Layer colorization

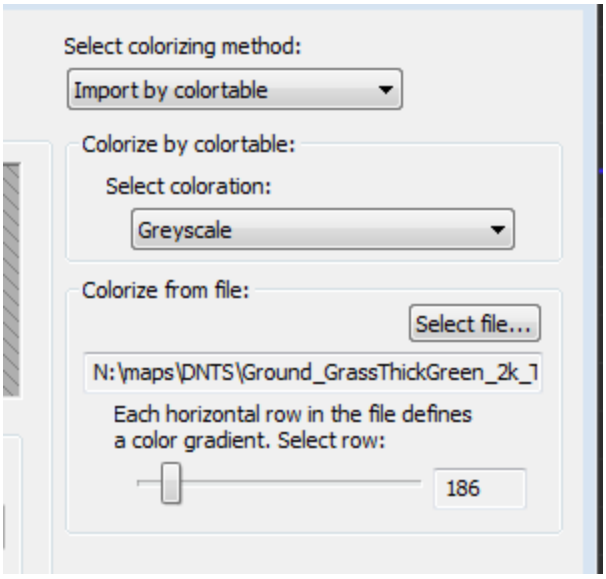
Most color textures for layers (with the notable exception of some straight up PBR textures) are generated by feeding noise (or derivatives of noise) into **Colorizer devices.** Colorizers take an input heightmap, and use it as a lookup table into a gradient of colors. Low values of height will take from the left of the gradient, high values from the right.



You can define your own color tables here, and even better, if you suck at choosing more than 1 color, you can use any random image you find online, and use any one line of it as a color table.

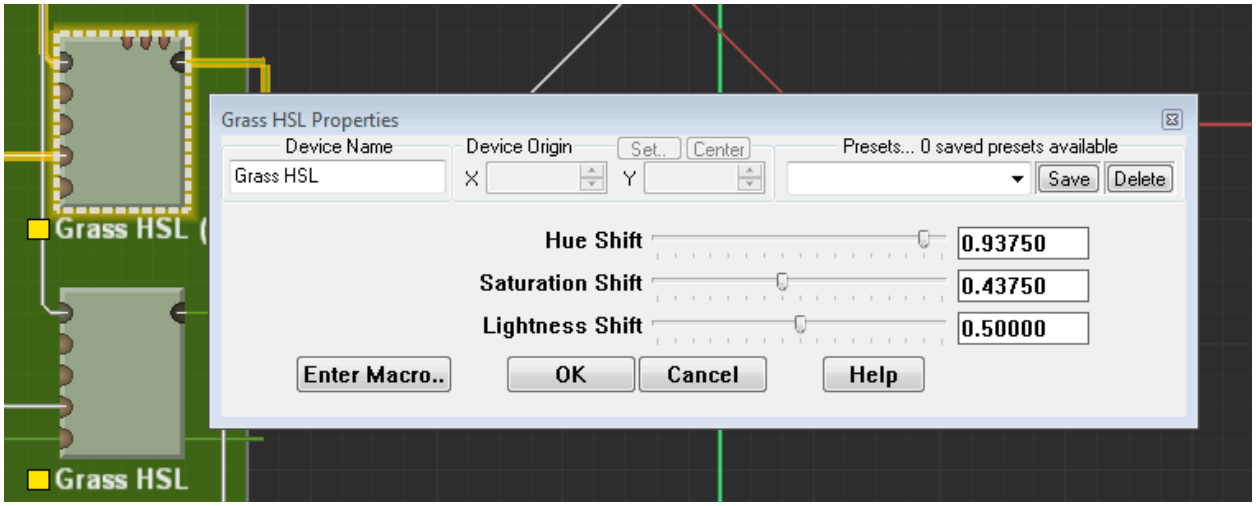
These colorizers are then mixed together, and optionally some distorted perlin noise is rendered from a global light perspective, and multiplicatively mixed onto them.

You can also just use a file input from any random tileable texture, and use that instead of color.



Hue-saturation-lightness control (HSL)

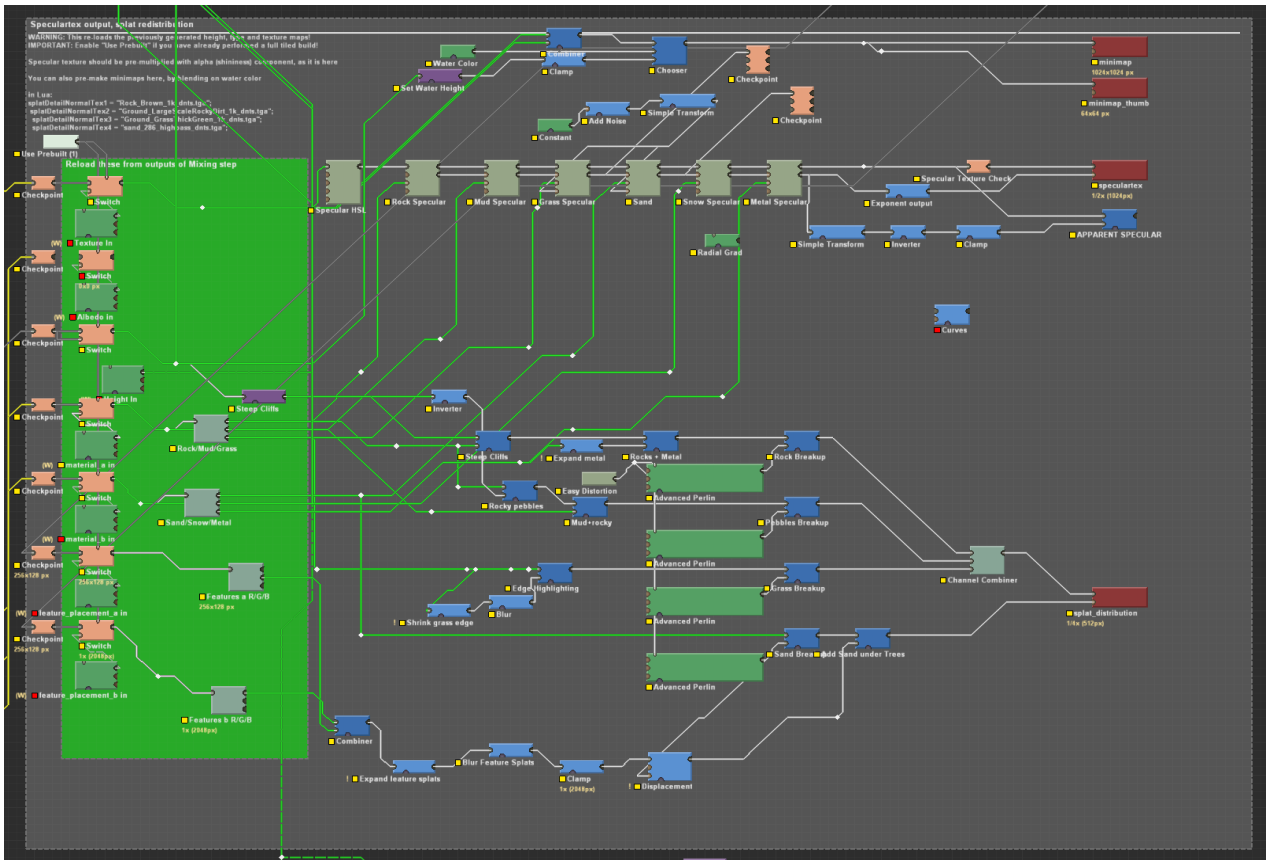
You will often find a lot of HSL macros littered everywhere in the path of color layers. This allows you to fine tune the appearance of any layer's color.



You can even mix the results of multiple of these depending on height or slope or anything.

DNTS Splatdistribution and Speculartex Generation

Splatdistribution defines where the individual splatdetailnormals will be placed. There are 4 channels available, the RGBA of the splatdistributiontex.
SpecularTex defines the amount (and color) of light reflected in a specular way in the RGB channel, and it defines the shinyness of the surface in the A channel.



The speculartex and splatdistributiontex share a lot of common inputs, as they are generated from the heightmap, the texture map, the materials map, and optionally, the feature distribution map (to allow you to put splats under trees, for example).

Splatdistribution

You may know that Spring only supports 4 different splat detail normals, and we are using 7+ different layers of materials, so we will have to share these 4 splat detail normals between the 7+ layers.

Splat distributions are calculated from the material_a and material_b outputs, which essentially just indicate in masks where each layer is distributed. We take these 7+ layers, and mix them down into 4 channels, using some of the following ideas:

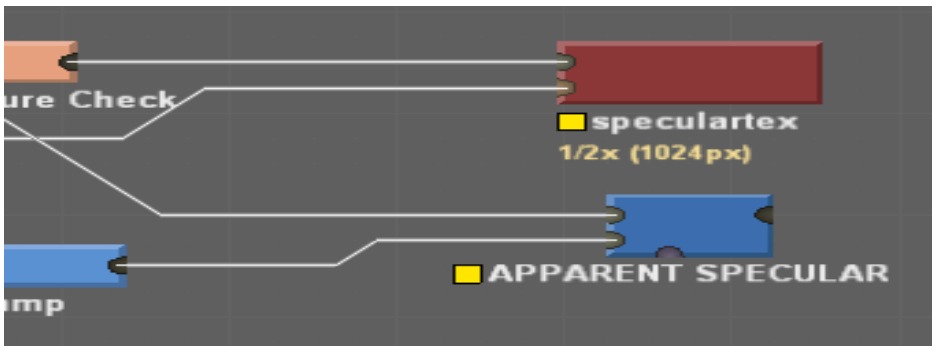
1. Snowy areas are smooth, so we don't really need a splat detail normal here
2. Metal patches usually share a splat detail normal with steep cliffs.
3. It is recommended to break up (multiply with noise) the DNTS channels, as this hides the repeating (tiling) nature of the splat detail normals.

SpecularTex

OK I'll be the first to admit this is a monster, but read the general knowledge about it and we'll learn how to work with it:
The specular texture defines what color specular (also known as 'shiny') reflections individual areas of the map will have.

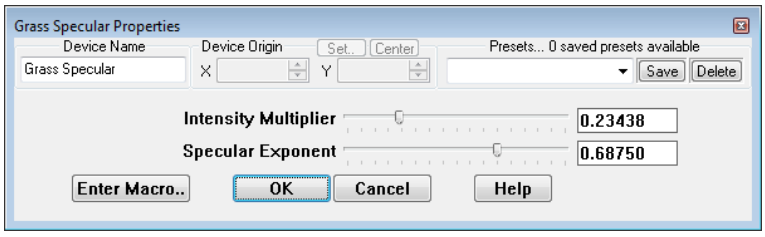
- The specular color is additive, in a sense that all the color in the specular texture will be added to the color of the terrain, if the sun is shining at it at just the right angle.
- This amount of specular reflection is stored in the RGB channel of the specular texture.
- In general, we will use the regular diffuse texture color as the specular texture color, but since its additive, we will decrease its brightness greatly.
- The alpha channel stores the specular power, or shininess, or polishedness of the surfaces, with full black corresponding to a specular power of 0 (very rough), and full white being 16 (very polished).
- A specular power of 16 is still considered as not very shiny, but emulates wet surfaces well.
- Due to the way Spring's specular is calculated, terrain with low specular power will show a lot of additional specular color, and shinier surfaces will emit less light on specular highlights, and you will need to compensate for this.
- Very shiny areas should have a brighter specular color, while very rough (low power) areas should have very dark specular color textures.

When you want to specify the specular of a layer, think long and hard about how shiny that layer should be, and how much light that layer should emit.



Each layer has two of its own sliders:

1. **Intensity multiplier**: how much light should be emitted from this layer
2. Specular Exponent: how shiny this layer should be.

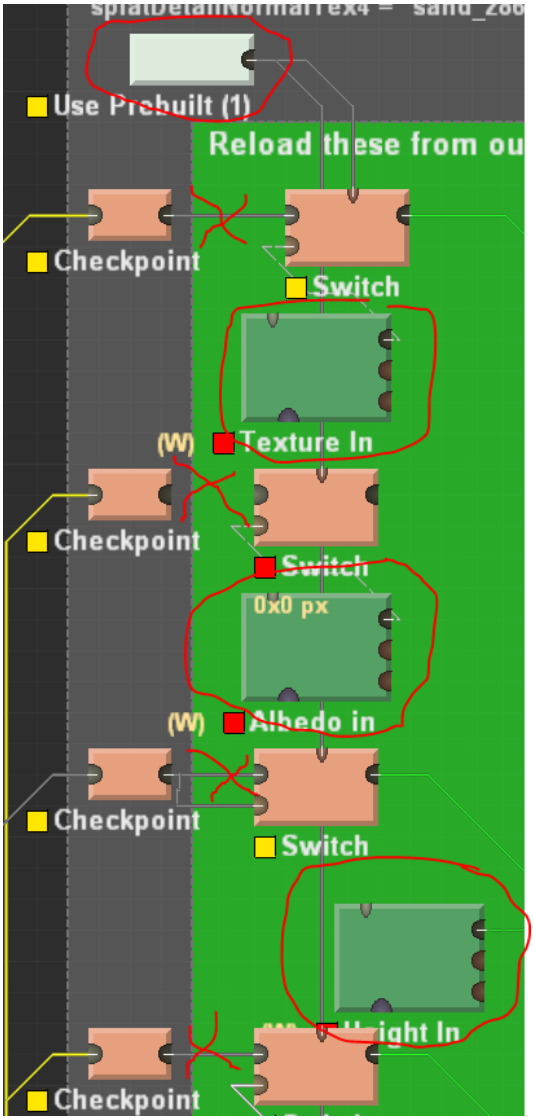


Note that if you want a layer to be not shiny, make sure the intensity multiplier is set very very low, with a low specular exponent.
If you want it metal-like, then crank up the both.
The shinyness of each DNTS splat detail normal is controlled by the specularтex.

Fast splatdistribution, specularmap, and minimap regeneration

This is very advanced magic, feel free to ignore. WM allows us to take the previously built height, texture, material, feature distribution and albedo maps and rebuild splatdistribution, specularmap and minimap in a few minutes, given enough RAM. For this to work, you must do the following things:

1. Set the Use Prebuilt control to 1.
2. Remove the wires connecting the **Checkpoints** to all **Switches** (there's like 7 of them)
3. Re-load the inputs of the switches (e.g. **Texture In)** by opening them and selecting the previously built images
    a. Use "**Place into current view**" for correct scaling
4. In **World Commands** -> **Project World Parameters** :
    a. set the **Normal Build Resolution** to the exact resolution of your main texture (yes this is huge)
    b. Check **Final Build,** unless you want to run out of RAM immediately.



This rerouting is needed so that WM knows to only use the prebuilt files, and not rebuild everything.
Now you can adjust the parameters of the splatdistribution, specular texture and minimap. This will also make tweaking things quite fast, so you can view your changes in layout mode rapidly.

When you are ready to re-export one of the outputs from this section:

1. SAVE YOUR WORLD
2. Select the output device you wish to build,
3. Click the yellow **Build to Current Device** button in the top bar. Ignore the RAM warning.
4. Once built, double click the device and click **Write Output to Disk!**

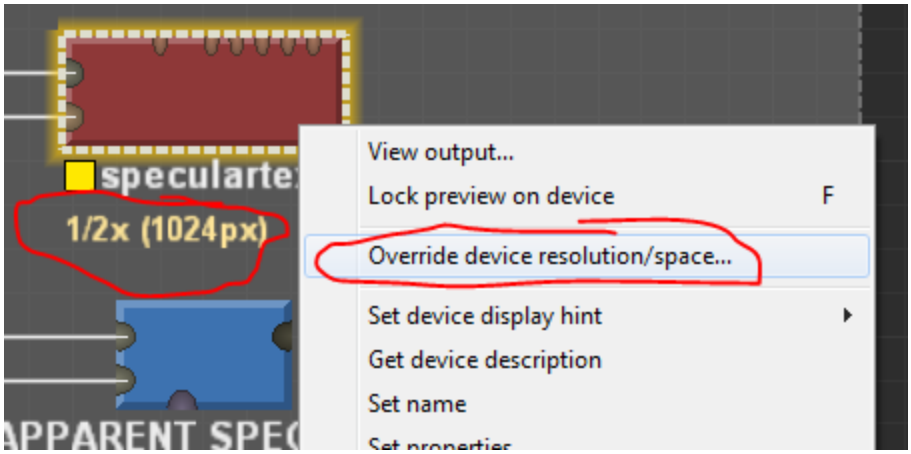
Outputs produced by WM

Scaling outputs

You can change the resolution of each output, specified as a fraction of the resolution of your main render. Specifying an exact resolution does not work with tiled builds, as each tile will have that resolution, which isn't something you really want

Some World Machine layouts have a known bug with Tiled Builds. If the output of any image file has scrambled tiles, you will need to remove the custom output resolution scale adjustment set on the file output nodes, build at full resolution, then resize later to the required resolution

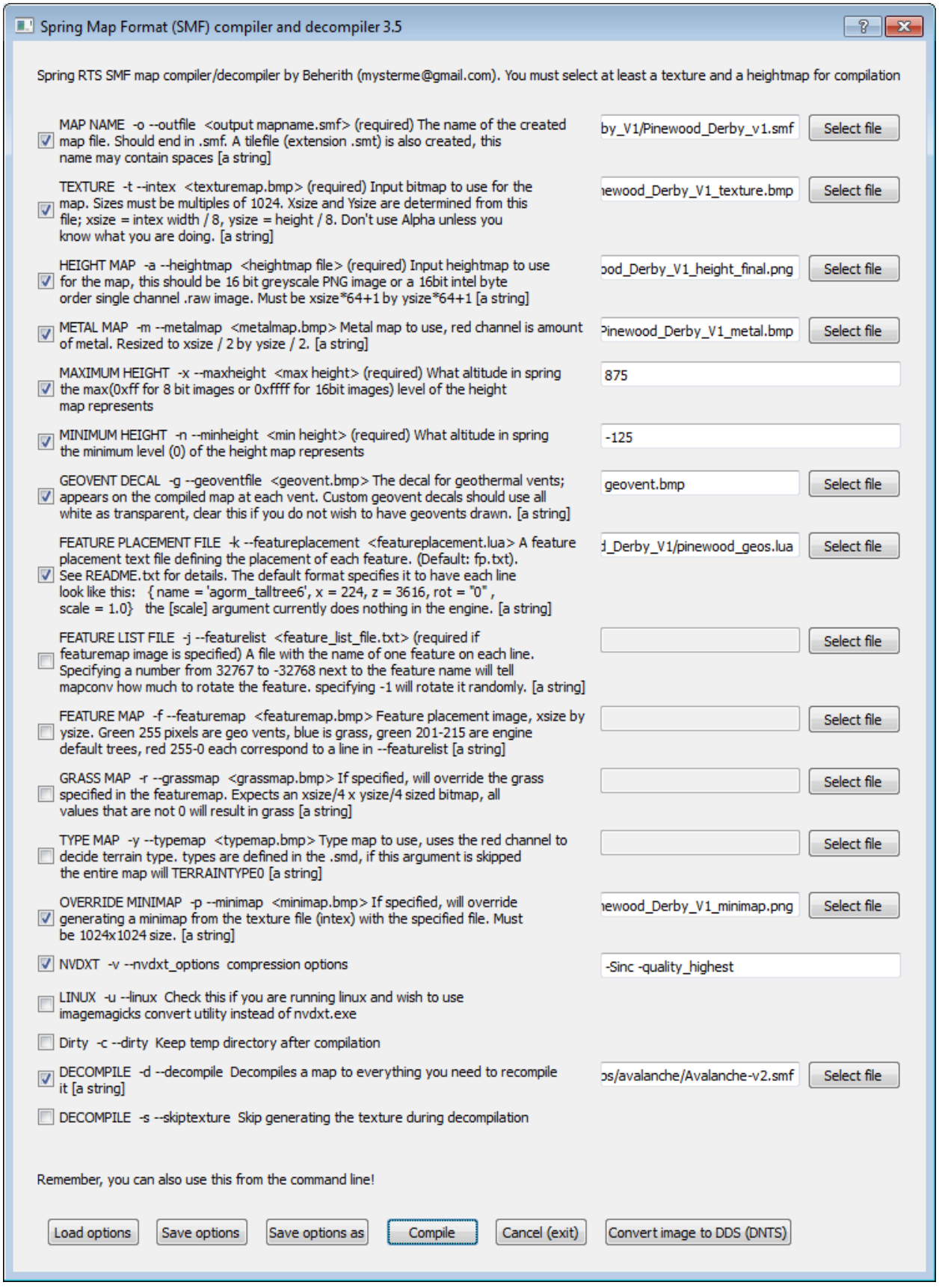
By right clicking on an output device, you can override the output. If you want a lower than full resolution specular, or splat distribution texture for example, you can render it at ½, ¼ or even ⅛ resolution



Compilation with PyMapConv

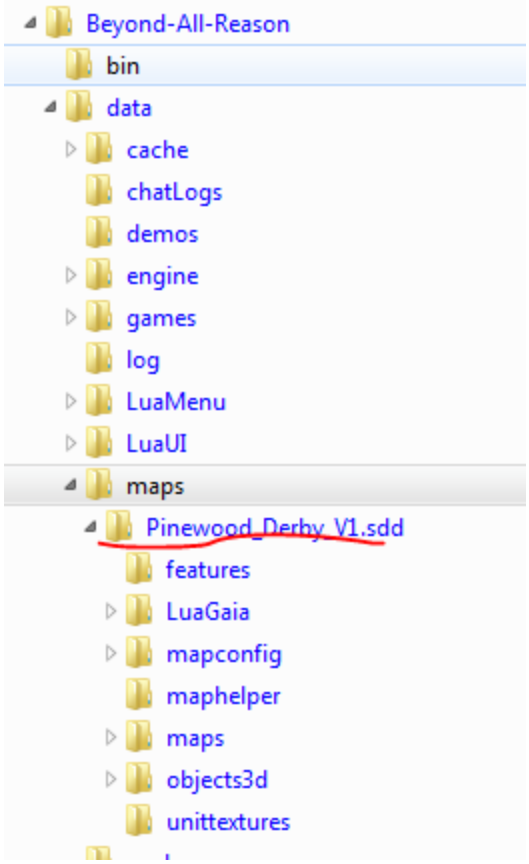
Get: [GitHub - Beherith/springrts_smf_compiler: This tool allows the compilation and decompilation of maps to springrts's binary smf map format.] and unzip the whole thing somewhere safe.

Pymapconv documents itself:


Spring Map Format (SMF) compiler and decompiler 3.5
Spring RTS SMF map compiler/decompiler by Beherith (mysterme@gmail.com). You must select at least a texture and a heightmap for compilation
MAP NAME -o --outfile <output mapname.smf> (required) The name of the created map file. Should end in .smf. A tilefile (extension .smt) is also created, this name may contain spaces [a string]
by_V1/Pinewood_Derby_v1.smf
Select file
TEXTURE -t --intex <texturemap.bmp> (required) Input bitmap to use for the map. Sizes must be multiples of 1024. Xsize and Ysize are determined from this file; xsize = intex width / 8, ysize = height / 8. Don't use Alpha unless you know what you are doing. [a string]
ıewood_Derby_V1_texture.bmp
Select file
HEIGHT MAP -a --heightmap <heightmap file> (required) Input heightmap to use for the map, this should be 16 bit greyscale PNG image or a 16bit intel byte order single channel .raw image. Must be xsize*64+1 by ysize*64+1 [a string]
ɔod_Derby_V1_height_final.png
Select file
METAL MAP -m --metalmap <metalmap.bmp> Metal map to use, red channel is amount of metal. Resized to xsize / 2 by ysize / 2. [a string]
Pinewood_Derby_V1_metal.bmp
Select file
MAXIMUM HEIGHT -x --maxheight <max height> (required) What altitude in spring the max(0xff for 8 bit images or 0xffff for 16bit images) level of the height map represents
875
MINIMUM HEIGHT -n --minheight <min height> (required) What altitude in spring the minimum level (0) of the height map represents
-125
GEOVENT DECAL -g --geoventfile <geovent.bmp> The decal for geothermal vents; appears on the compiled map at each vent. Custom geovent decals should use all white as transparent, clear this if you do not wish to have geovents drawn. [a string]
geovent.bmp
Select file
FEATURE PLACEMENT FILE -k --featureplacement <featureplacement.lua> A feature placement text file defining the placement of each feature. (Default: fp.txt). See README.txt for details. The default format specifies it to have each line look like this: {name = 'agorm_talltree6', x = 224, z = 3616, rot = "0", scale = 1.0} the [scale] argument currently does nothing in the engine. [a string]
ɟ_Derby_V1/pinewood_geos.lua
Select file
FEATURE LIST FILE -j --featurelist <feature_list_file.txt> (required if featuremap image is specified) A file with the name of one feature on each line. Specifying a number from 32767 to -32768 next to the feature name will tell mapconv how much to rotate the feature. specifying -1 will rotate it randomly. [a string]
Select file
FEATURE MAP -f --featuremap <featuremap.bmp> Feature placement image, xsize by ysize. Green 255 pixels are geo vents, blue is grass, green 201-215 are engine default trees, red 255-0 each correspond to a line in --featurelist [a string]
Select file
GRASS MAP -r --grassmap <grassmap.bmp> If specified, will override the grass specified in the featuremap. Expects an xsize/4 x ysize/4 sized bitmap, all values that are not 0 will result in grass [a string]
Select file
TYPE MAP -y --typemap <typemap.bmp> Type map to use, uses the red channel to decide terrain type. types are defined in the .smd, if this argument is skipped the entire map will TERRAINTYPE0 [a string]
Select file
OVERRIDE MINIMAP -p --minimap <minimap.bmp> If specified, will override generating a minimap from the texture file (intex) with the specified file. Must be 1024x1024 size. [a string]
ıewood_Derby_V1_minimap.png
Select file
NVDXT -v --nvdxt_options compression options
-Sinc -quality_highest
LINUX -u --linux Check this if you are running linux and wish to use imagemagicks convert utility instead of nvdxt.exe
Dirty -c --dirty Keep temp directory after compilation
DECOMPILE -d --decompile Decompiles a map to everything you need to recompile it [a string]
ɔs/avalanche/Avalanche-v2.smf
Select file
DECOMPILE -s --skiptexture Skip generating the texture during decompilation
Remember, you can also use this from the command line!
Load options
Save options
Save options as
Compile
Cancel (exit)
Convert image to DDS (DNTS)

Notes:
You need at **least a height and a texture map**. You probably also need a metal map.
Make sure maxheight - minheight = 1000. This is because this was our working range in world machine. If you want spring water level too at height 150 in WM, then set:
Maxheight: -150
Maxheight: 850
But dont worry too much, you can change map heights in mapinfo.lua.

Grab one of my previous maps, unzip into your spring data folder (usually /data/maps/mymapname.sdd) (.sd7 and .sdz files can be unzipped with 7zip). So that the dir structure looks like this:



Place the .smf and .smt files made by pymapconv into data/maps/yourmapname.sdd/
Edit mapinfo.lua located in the root.

Packaging for Testing and Distribution

For just testing the map locally, you do not have to zip it up if all the contents are in the /data/**maps/yourmapname.sdd/** directory. You can quickly iterate changes this way.

MapNormals, Specular and Splatdistribution resolutions

The current best practice, and the way the world is set up, is to have identical resolution texture and mapnormals, and optionally identical resolution specular texture. To save space, and to allow them to load at 2K+ sizes, they must be converted to .DDS.

So if the texture is 8K, mapnormals should absolutely be 8K in size. Mapnormals is RGB.

Specular can be smaller, to save space, and ideally is an integer fraction of texture, so 8K, 4K, 2K, 1K can be used. Specular is RGBA.

Splatdistribution can usually be smaller, to save some space, so 4K, 2K, or even 1K can be used if required. Splatdistribution is RGBA.

Compression to .DDS with NVTT_Export

The latest NVTT Export tools allow for the fast, efficient compression of extremely large textures. I have prepared a set of .bat files to ease their use.

https://github.com/Beherith/springrts_smf_compiler/blob/master/NVTT_DragAndDropConvertToDDSTools.7z

Unzip this to where you keep your PyMapConv.exe, the archive is included in the full PyMapConv github download.

To compress any 8 bit png or bmp image to DDS, you have to **drag-and-drop onto the included .bat files**. You can and drop multiple files at the same time, and they will all be converted. The output .dds file is placed next to the original files by the .bat.

Drag Mapnormals onto NVTT_DragAndDropOnThis_Convert_RGB_To_DXT1C.bat

Drag Specular and Splatdistribution onto NVTT_DragAndDropOnThis_Convert_RGBA_To_DXT5.bat

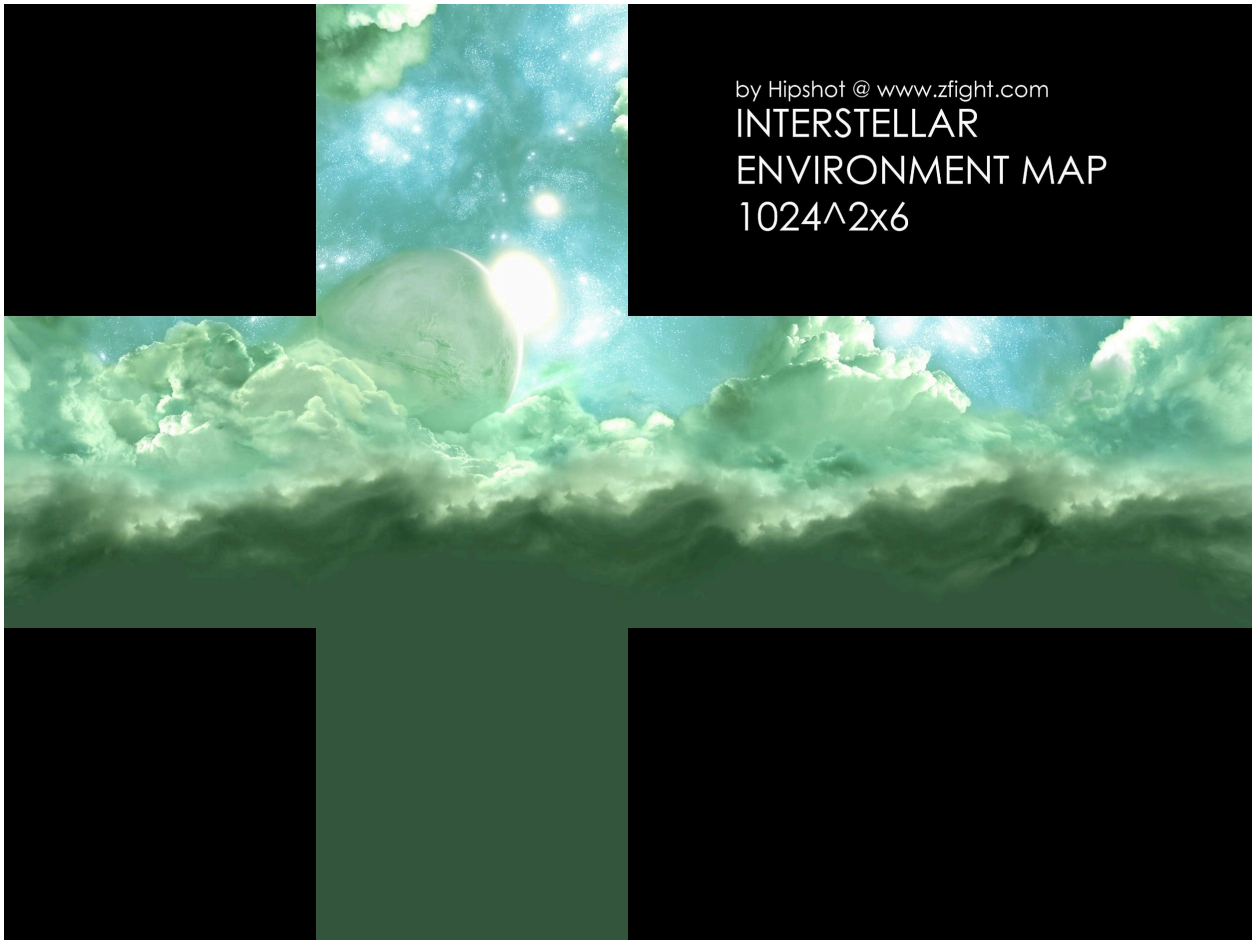
Due to where the origin (0,0) point of dds images are, the results will be correctly flipped top to bottom.
Place the resulting .dds files into the /maps folder next to .smf, and link them in the mapinfo.lua file.

Inclusion of map source files

If you base your map on a WM macro made by me, please **include the map source in the archive, as required by its CC BY SA NC license**. Place the .tmd WM file and any input images you used during generation into /maps/sources. These will compress well so don't worry about file size.

Skyboxes

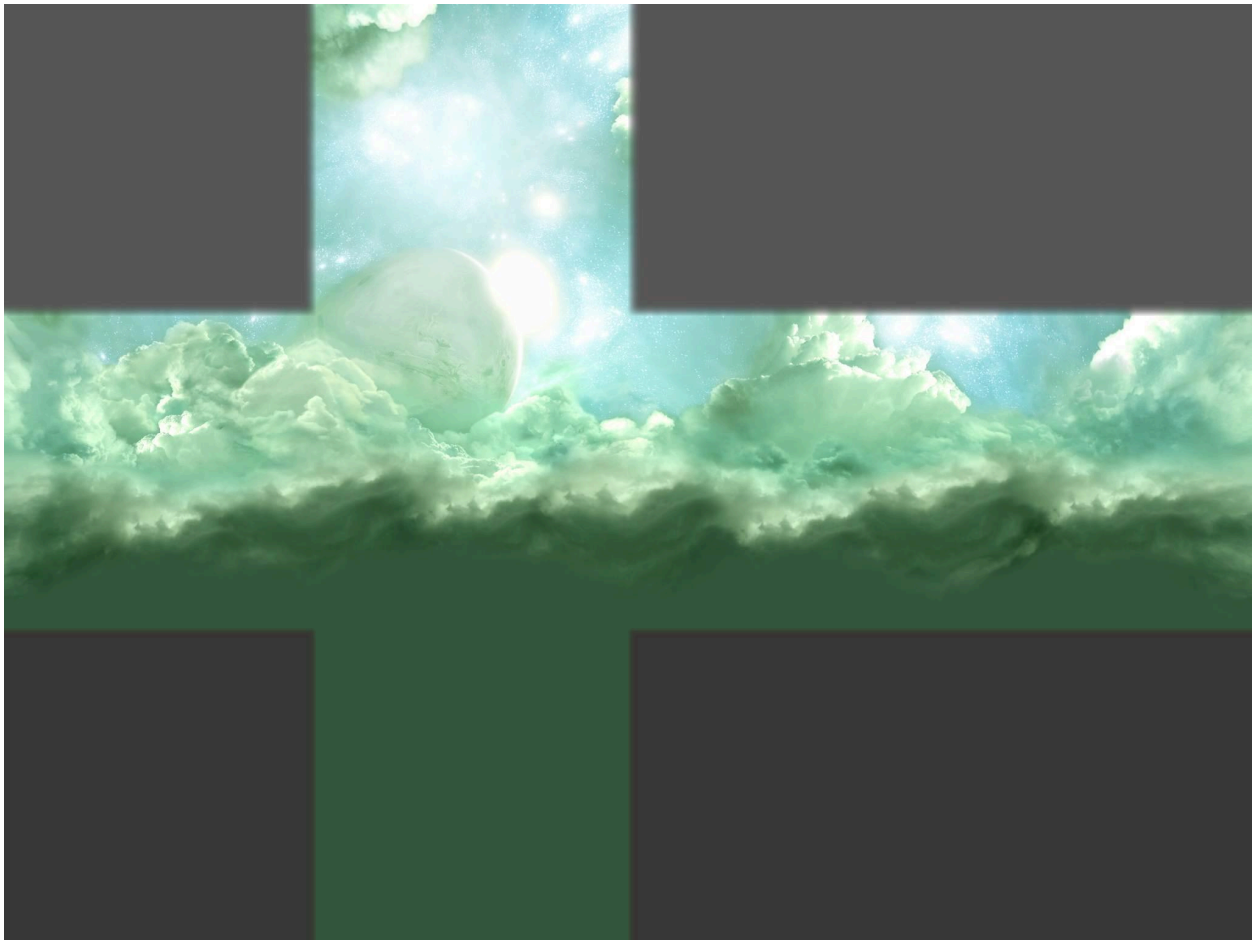
Skyboxes are easy to do, and add a great touch to any map. The required format is a .dds cubemap, looking like this skybox cross image:



The recommended skybox resolution is 2k per side of the cube, so the skybox cross source image should be 8k by 6k.

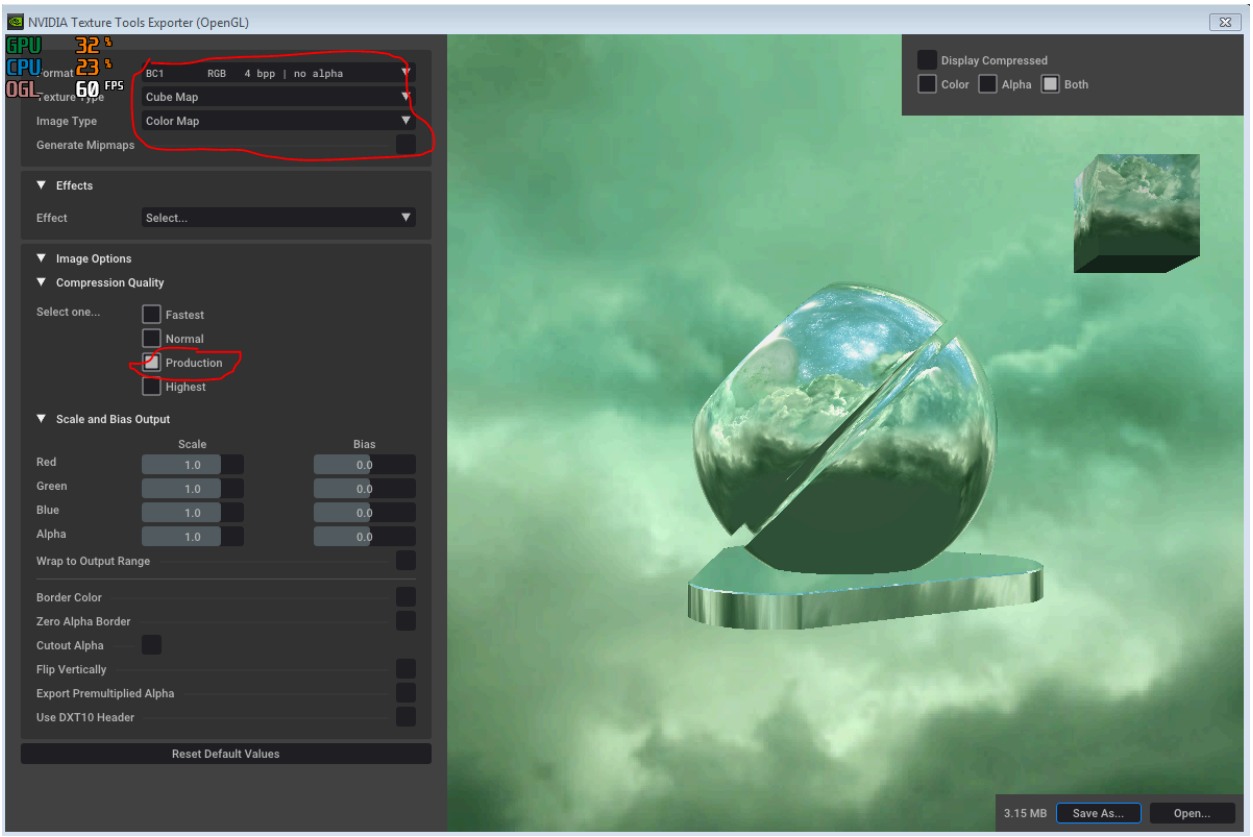
Due to how PBR materials on units reflect the skyboxes, make the top of the cube much brighter, and the sides brighter too, as the top wont realistically ever be seen, except on mirror

shiny surfaces.



You can also create a skybox from any image of the sky with a custom world machine macro in a single click, contact me for details.

Drag your cross image onto NVTT_Export.exe, and set it up as marked on the screenshot.



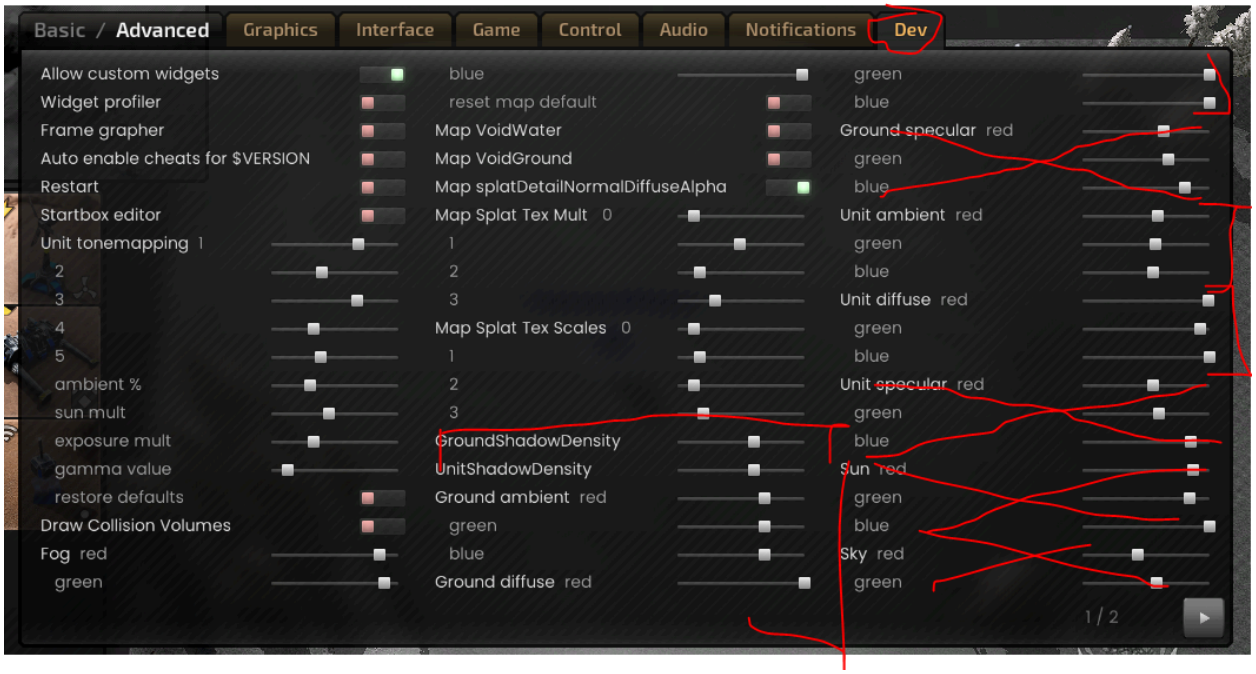
Click save as, and include it in your /maps folder.

Lighting in BAR

There are a bunch of settings in mapinfo.lua pertaining to the lighting of the map. The most important ones are:

1. GroundAmbientColor, the color of the shadowed or non sun-lit areas. Recommended 0.5, 0.5, 0.5
2. GroundDiffuseColor, the color of sun lit terrain: recommended 0.9, 0.9, 0.9
3. Groundshadowdensity, how much shadowed terrain gets darkened, default 0.75
4. UnitAmbientColor, the color units in shadow, Recommended 0.5, 0.5, 0.5
5. UnitDiffuseColor, the color of fully sun lit units, recommended 0.9, 0.9, 0.9
6. unitshadowdensity,how much shadowed units get darkened, default 0.75
7. sunDir, where the sun shines from, an x, y, z vector that will get normalized, with X pointing toward player, Z pointing right, and Y pointing up

Load the map up in BAR, and hit Settings (F10), enable advanced, and the dev tab by entering the /devmode command into the chat prompt. Tweak the following sliders, and type the values into mapinfo.lua once satisfied:



Finishing touches in SpringBoard

Download and install SpringBoard https://github.com/Spring-SpringBoard/SpringBoard-Core
Create a directory for maps at
C:\users\myusername\AppData\local\Programs\Springboard\data\springboard\maps
Place your maps .sdd folder into SpringBoard\data\springboard\maps\mymap.sdd

Adding Features in SpringBoard

Copy over all three folders (features, Objects3D, Unittextures) from
https://github.com/beyond-all-reason/support/tree/master/MapFeatures
into your mymap.sdd folder, to a path like this:
C:\Users\myusername\AppData\Local\Programs\SpringBoard\data\maps\Throne_v7.sdd

Run SpringBoard, and select New Project on the right, and select your map from the dropdown menu:

Save your project, and put dbg_feature_dump.lua into: [THIS DOESNT WORK YET]
\SpringBoard\data\springboard\projects\rifted_1.sdd\LuaUI\Widgets

On the features tab on the right, click whichever feature you want, then left click on the map to ‘paint’ the feature you desire, right-click to remove. Adjust your brush size and spread to the way you want your features to look.


Objects
Map
Env
Misc
Units
Features
Properties
Collision
83
Brush
Add
Type:
Other
Wreck:
Units
Terrain:
Ground
Search:
Fir Tree
Team:
Team 0 (ID: 0)
Size:
89
Spread:
6
Noise:
1
Min rot x:
0.00
Max rot x:
0.00
Min rot y:
-180.00
Max rot y:
180.00
Min rot z:
0.00
Max rot z:
0.00

Feature visibility in springboard:

/featurefadedistance 32000
/featuredrawdistance 32000

Alternate method of converting SpringBoards Model.lua

Find the model.lua file in the saved springboard project (this contains all features, geos too!)
Place this python script next to it:
https://github.com/Beherith/springrts_smf_compiler/blob/master/src/springboard_model_lua_to_set_lua_feature_dumper.py
Run the python script.
Then copy the features from **featureplacement_set.lua** into /**mapconfig/featureplacer/set.lua**, into the **objectlist** table.

Water

Use only /water 4 (bumpmapped water), all other waters look pretty bad. You can tune the settings for water in springboard, then copy over the settings into mapinfo.lua. Your easiest bet however, is to copy over water settings from one of my newer maps: **Mediterraneum_v1**. Pay close attention to fresnelmin and fresnelmax, these control the transparency of water surface at various angles.
https://springrts.com/wiki/Mapdev:mapinfo.lua#water

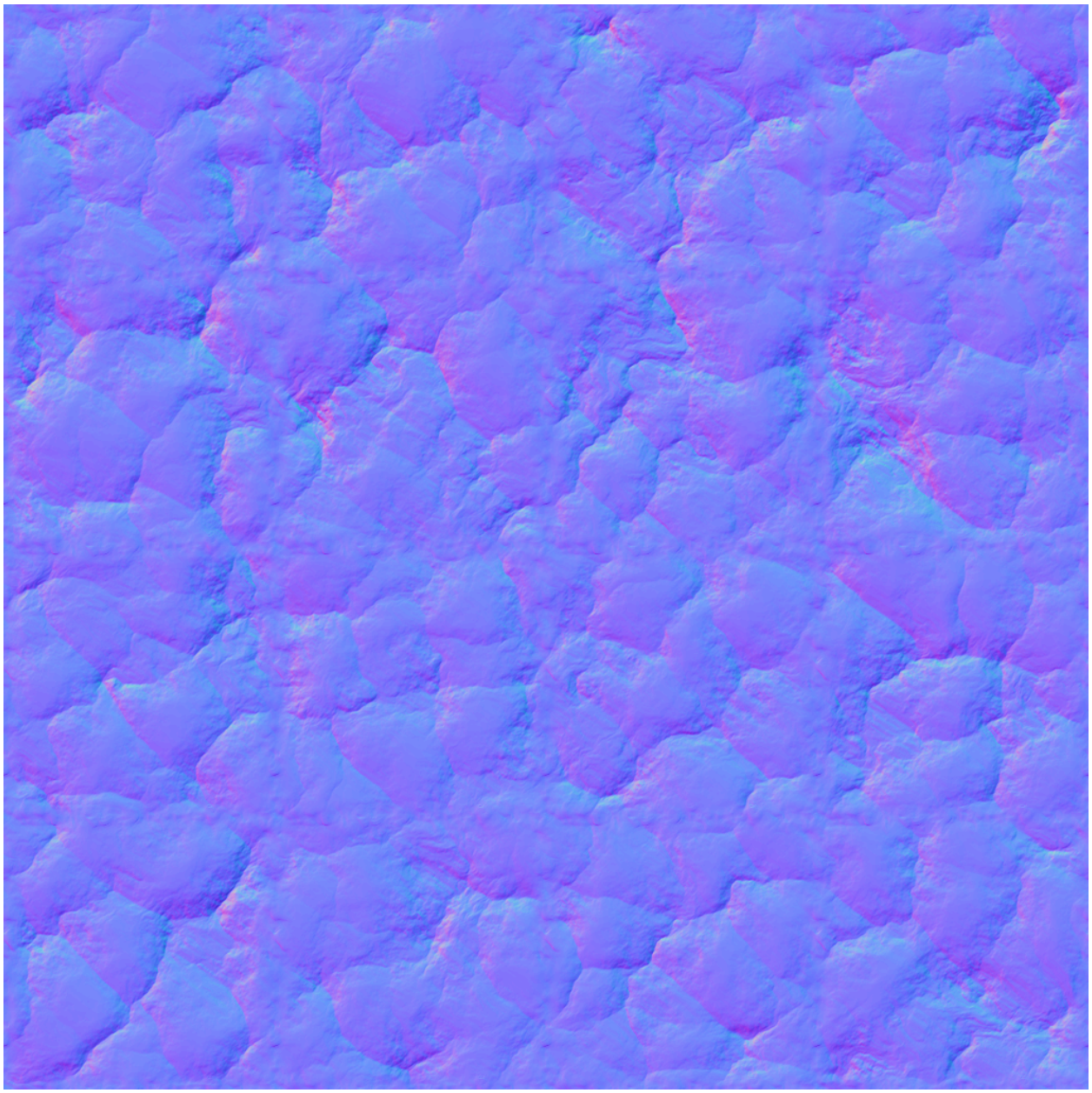
New info: You can now change bumpwater params live in BAR, enable /devmode and see F10 settings last tab. A great example is now Downs Of Destruction V2.

```
water = {
	damage =  0,

	repeatX = 10.0,
	repeatY = 10.0,

	absorb    = { 0.05, 0.005, 0.001 }, --absorption coefficient per elmo of water depth
	 basecolor = { 0.3, 0.5, 0.5 }, -- the color shallow water starts out at
	mincolor  = { 0.0, 0.3, 0.3 },

	ambientFactor  = 1.0,
```

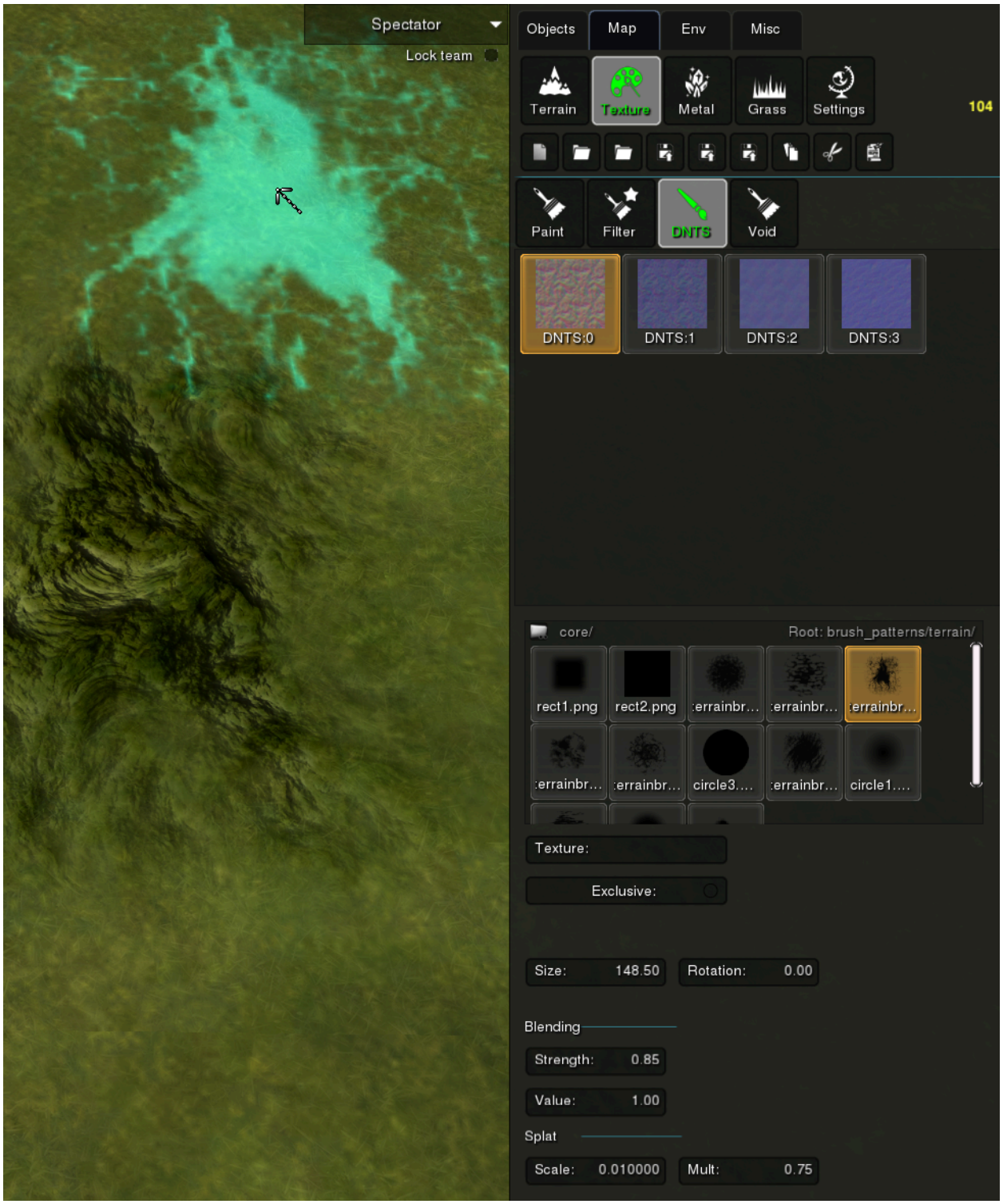
```
      diffuseFactor  = 1.0,
      specularFactor = 1.4,
      specularPower  = 40.0,

       surfacecolor   = { 0.67, 0.8, 1.0 }, --color of the water
texture
      surfaceAlpha  = 0.02,
      diffuseColor  = {0.0, 0.0, 0.0},
      specularColor = {0.5, 0.5, 0.5},
      --planeColor = {0.00, 0.15, 0.15}, --outside water plane color
--

       fresnelMin     = 0.08, --This defines the minimum amount of
light the water surface will reflect when looking vertically down on
it [0-1]
      fresnelMax    = 0.5, --Defines the maximum amount of light the
water surface will reflect when looking horizontally across it [0-1]
      fresnelPower = 8.0, --Defines how much

      reflectionDistortion = 1.0,

      blurBase       = 2.1,
      blurExponent = 1.5,

      causticsResolution = 100.0,
      causticsStrength = 0.16,

      perlinStartFreq  =  8.0,
      perlinLacunarity = 3,
      perlinAmplitude  =  0.85,

      shoreWaves = true,
      forceRendering = false,

      numTiles = 4, -- default 1
      windSpeed = 0.5, -- default 1.0
      waveOffsetFactor = 0.3, -- default 0.0
      waveLength = 0.37,
      waveFoamDistortion = 0.10,
      waveFoamIntensity = 1.0,
      normalTexture = "maps/waterbump_4tiles.png",
```

```
      --hasWaterPlane = true, --specifies whether the outside of the
map has an extended water plane

    },
```

This is waterbump_4tiles.png :



Hand-painting Splats in SpringBoard



Once you are satisfied with your painting (it's a really good idea to paint a small patch of DNTS under features, makes them blend oHh-so-well with the terrain), you need to look into the folder:

\SpringBoard\data\springboard\projects\archsimkat_sb.sdd\sb_project_files\textures\shading-splat_distr.png, convert it to DDS with the tools shown above, and link it in mapinfo.lua.
ep

Verifying and fine-tuning passability in SpringBoard

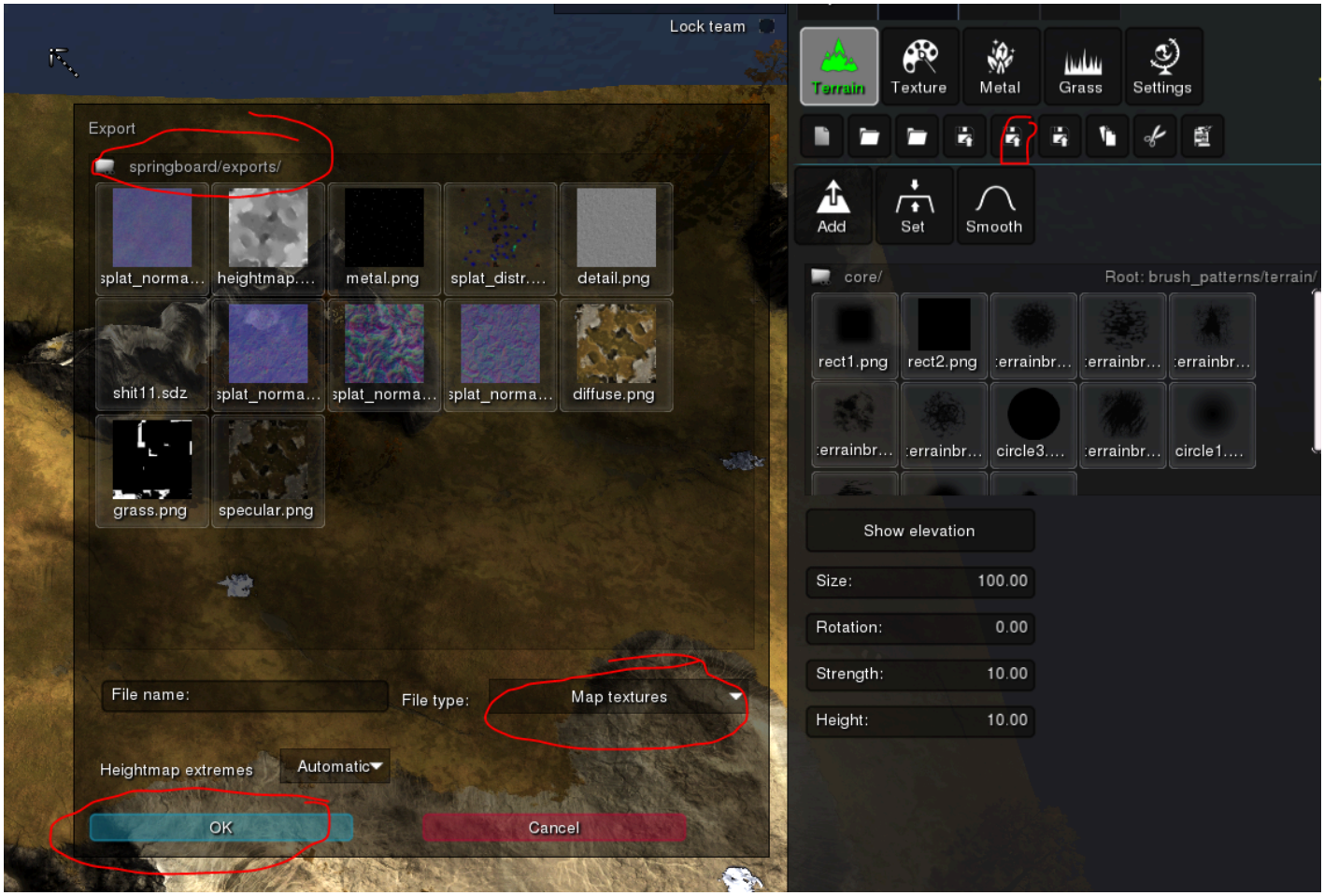
You'll need to copy over some units into your map. Your best bet is **armpw**, and **armstump**, and **armch** (construction hovercraft)

Start from the springboard map folder that we've been working with so far: (C:\Users***\AppData\Local\Programs\SpringBoard\data\maps\Archsimkats_Valley_V1.sdd)

Copy the contents of this repo into the map.sdd, and then you can /give these units, select them and use f2 to view passability:

https://github.com/Beherith/bar_springboard_passability
You don't have to put this in your map.sdd you can just put it in the springboard project and you don't have to worry about accidentally distributing it

Use the brushes in the terrain tab to fine-tune passability. Once you are done, you can export your modified heightmap:

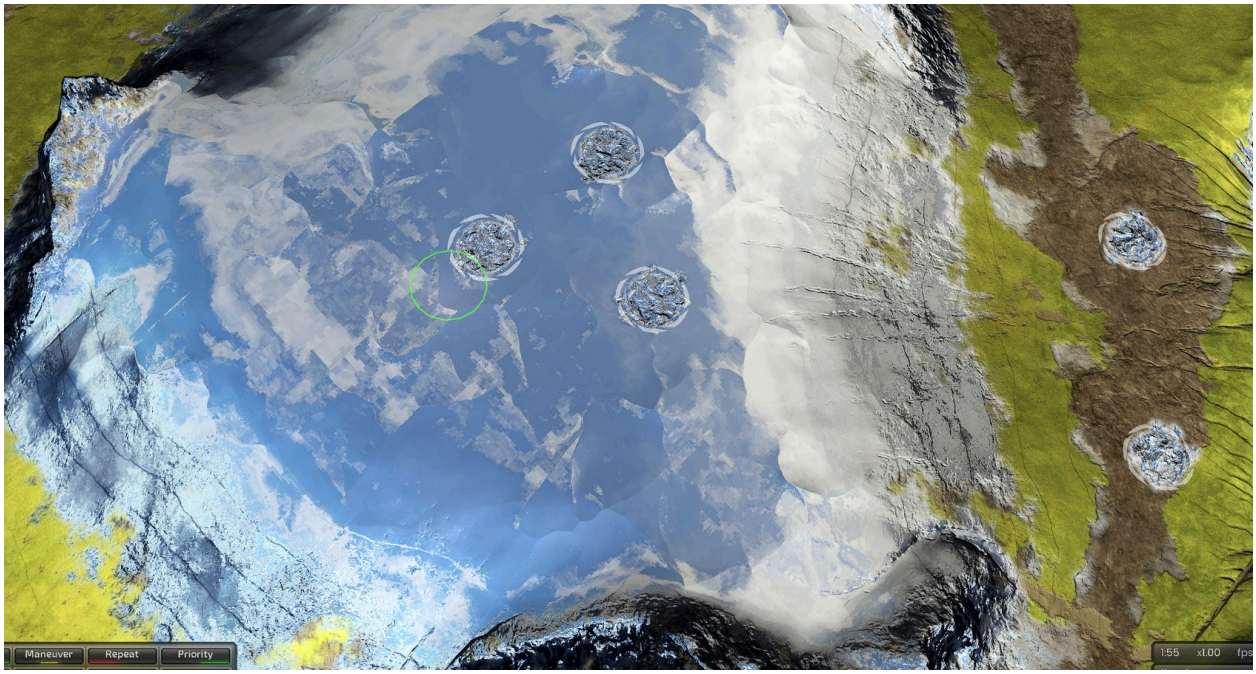


Retrieve this heightmap.png, and recompile your map (there is probably an easier way, feel free to detail it here if you wish).

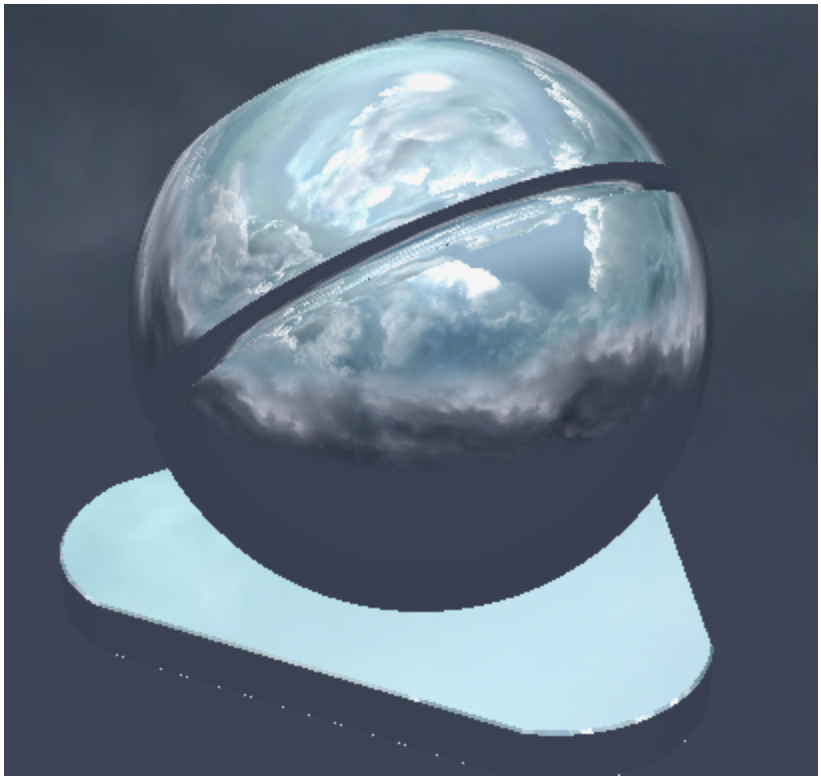
Additional Details

Sky reflections

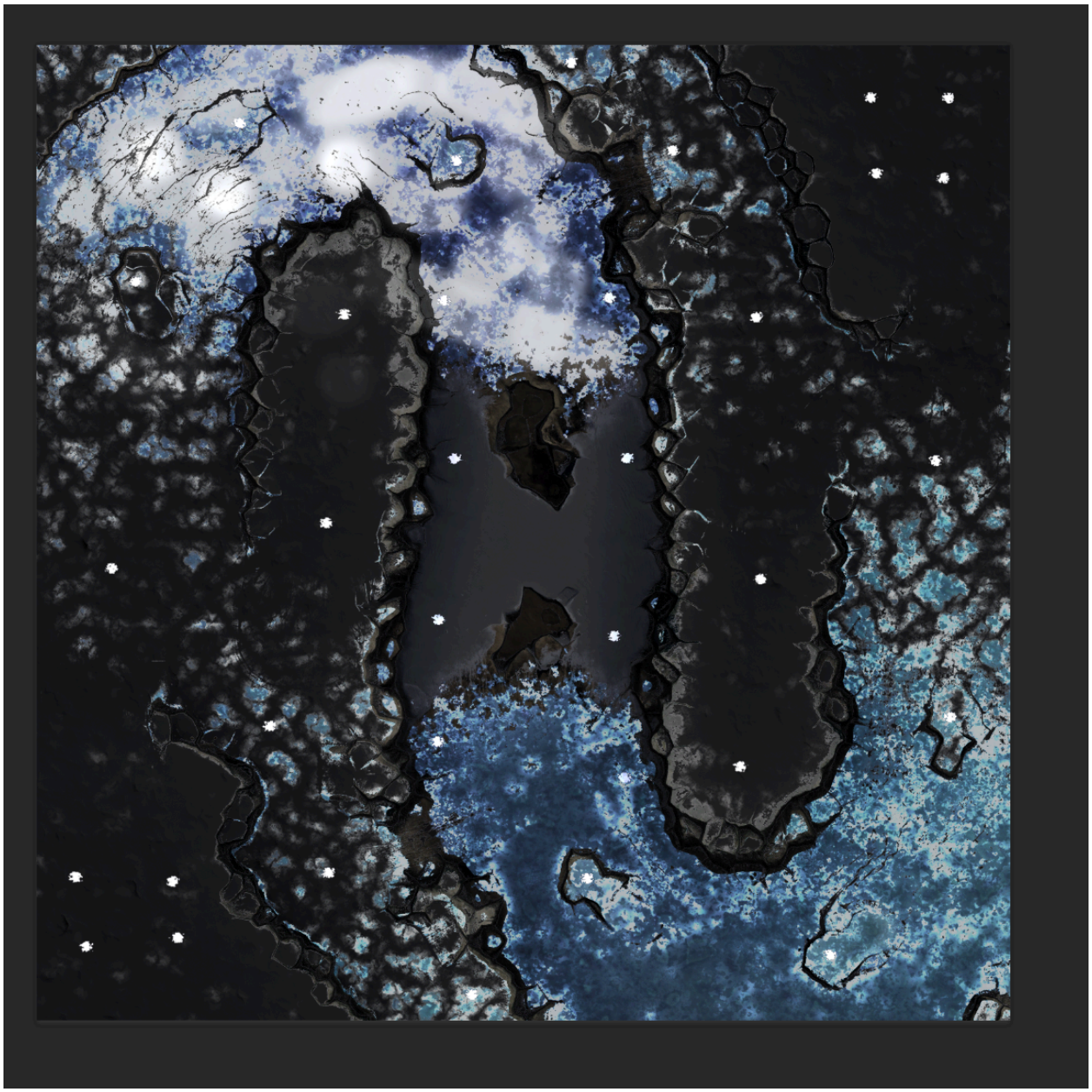
You can specify skyreflectmodtex in mapinfo.lua, and it will take that RGB texture, and blend the color of the map with the skyreflections according to the skyreflectmodtex color. In practice, this means that black values will result in no sky reflections, white values will result in the diffuse texture color being fully replaced with the sky reflections (fully metallic surface), and colors in between will blend between the two. For practical purposes, unless you want to tint the sky reflections, using low grayscale values give good effects on ice and metal. Note that any compression or other artefacts in detailNormalTex will be highly amplified by this, so only use it on surfaces that have DNTS splatted onto them.



It is also important to mention that you need a skybox with at least some detail in the upper sky portion of it for this to be nice looking. A good example is this skybox:



(miramarclouds_v2)
And this skyreflectex:



Colorizing Features

It can often happen that after carefully adjusting the lighting for units and the map to be correct, the features just don't match as well as one would want. Fixing this can be done by loading up the feature's texture 1 in photoshop (e.g. ad0_aleppo2_1.tga), and adjusting the hue, saturation and lightness values of it, then resaving. Unfortunately this cannot be refreshed while ingame, so the map has to be reloaded on each adjustment. On newer BAR engines, you can **/reloadtextures s3o** to adjust colorization!

Grass in mapinfo

Grass can be tweaked in mapinfo.lua.

```
--------- HOW TO CONFIGURE GRASS (also important!) -----------------------
local grassConfig = {
    patchResolution = 32, -- distance between patches, default is 32, which matches the SpringRTS grass map resolution. If using external .tga, you can use any resolution you wish
   patchPlacementJitter = 0.66, -- how much each patch should be randomized in XZ position, in fraction of patchResolution
  patchSize = 4, -- 1 or 4 clusters of blades, 4 recommended
   grassMinSize = 0.3; --Size for grassmap value of 1 , min and max should be equal for old style binary grassmap (because its only 0,1)
  grassMaxSize = 1.7; -- Size for grassmap value of 254
  grassShaderParams = { -- allcaps because thats how i know
    MAPCOLORFACTOR = 0.6, -- how much effect the minimapcolor has
     MAPCOLORBASE = 1.0,      --how much more to blend the bottom of the grass patches into map color
    ALPHATHRESHOLD = 0.01,--alpha limit under which to discard a fragment
    WINDSTRENGTH = 0.1,      -- how much the wind will blow the grass
    WINDSCALE = 0.33, -- how fast the wind texture moves
    WINDSAMPLESCALE = 0.001, -- tiling resolution of the noise texture
    FADESTART = 5000,-- distance at which grass starts to fade
    FADEEND = 8000,--distance at which grass completely fades out
     SHADOWFACTOR = 0.25, -- how much shadowed grass gets darkened, lower values mean more shadows
    HASSHADOWS = 1, -- 0 for disable, no real difference in this (does not work yet)
    GRASSBRIGHTNESS = 1.0; -- this is for future dark mode
  },
   grassBladeColorTex = "LuaUI/Images/luagrass/grass_field_medit_flowering.dds.cached.dds", -- rgb + alpha transp
   mapGrassColorModTex = "$grass", -- by default this means that grass will be colorized with the minimap
    grassWindPerturbTex = "bitmaps/Lups/perlin_noise.jpg", -- rgba of various frequencies of perlin noise?
  grassWindMult = 4.5, -- how 'strong' the perturbation effect is
  maxWindSpeed = 20, -- the fastest the wind noise texture will ever move,
  -- The grassdisttex overrides the default map grass, if specified!
     grassDistTGA = "", -- MUST BE 8 bit uncompressed TGA, sized Game.mapSize* / patchResolution, where 0 is no grass, and 1<= controls grass size.
}
```

Load the map up in BAR, and hit Settings (F10), enable advanced, and the dev tab by entering the /devmode command into the chat prompt. Tweak the following sliders, and type the values into mapinfo.lua once satisfied:

MapOptions.lua

You can set various map level options, like should there be speed bonuses for roads, should the water be acidic, and how high the water level should be (for a dry/wet) version of the same map. This is a little bit involved, and I recommend taking a look at mapoptions.lua inside of Throne_V8, with the corresponding mapconfig/mapinfo/0_apply_options.lua file. Currently, sanely testing any of these requires SpringLobby, and in skirmish mode. Select the game and map, then select the Game Options tab. Resize the window to see the scrollbar appear and to be able to see the map options at the bottom of the options tab.
Very important: To change/reload mapoptions for Springlobby, you must delete the cached mapoptions file for the map! E.g. delete:
C:\Users***\AppData\Roaming\springlobby\cache\Throne_V8-2318621211.mapoptions.json

Final Zipping

Use the official 7zip client to compress the contents of the .sdd folder, and make sure to set the **archive to non-solid** for faster loading. Rename from .7z to .sd7 when you are done.
Compress the contents of your map inside the .sdd, **do not include the .sdd folder**.