NWN CTT 1.0 Beta

(Neverwinter Nights Community Tileset Tutorial)

Created By
The Community
For
The Community

The following people all had a hand in creating the information found in this Tutorial.

Pstemarie – Project Q Admin

Bannor Bloodfist – CTP Admin

Merricksdad – Tileset and Tutorial Creator

BioWare Corp. – Tileset Construction Tutorial

Tonden Ockay – Put this whole document together

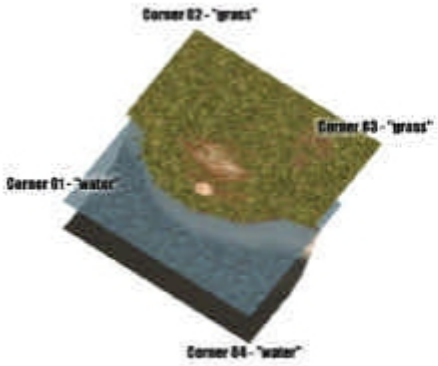
* I created this tutorial by taking most of the information found in it from tutorials/guides that the people above created, in order to have all the information in one place.

Table of Contents

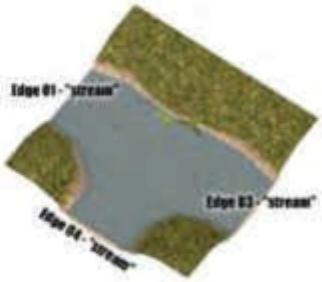
Tilesets

As I’m sure you know by now, a tileset is a group of tiles that are referenced into one list, any of which are able to be painted in one Neverwinter Nights map. The models for the tiles can be shared between tilesets, but in the interests of consistency and design, most of them are unique to a single tileset. In order to consider what you want in a tileset, you have to understand some of the aspects of how the Neverwinter Toolset handles painting.

For one tile to be positioned next to another, it HAS to have the same edge information. Each edge will have three attributes… the two corner types and one crosser type. A “corner” simply states what kind of terrain occurs at a specified corner (e.g. grass, water and pit). “Crossers” refers to the flag type of what happens in the middle of an edge (e.g. stream, road and fence). Most tiles do not contain a crosser flag, but for it to match up with another tile the next tile has to have an empty edge too.



Example 1



Example 2

Example 1 Example 2 Example 1 shows the corner types for that specific tile. All of the crosser types for that tile are empty. Example 2 shows a tile which has grass on all four corners, but three edges have “stream” crossers and one is empty.

Height Transitions are a specific type of corner, and in the case of the rural tileset, the height transition corners simply have the “grass” flag, but with a height value of “1”. This means that more grass tiles can attach themselves to the upper parts of this tile, even though they are 4m above the original grass height. I’ll show you how to define these things later on…

Groups and features are our way of creating “stamps”. A Group is a number of tiles that all fit together, that have to be placed on the map all together or not at all. In that way they cannot come up by painting crossers and corners. A Feature is a group made of one tile… again, preventing it from coming up in just normal painting situations (even though it may have all the same corner and crosser attributes as other tiles).

It’s important to note that, as tile are rotate-able, the number of tiles you have to produce is greatly reduced. Try to make sure you don’t inadvertently create two tiles that do the same job (unless this is desired).

On that subject, “Variation” is the name we give to any tile that has all the same attributes as another tile or tiles, all of which are not features or groups. When the toolset is provided with a number of tiles to choose from for a spot, it decides randomly which one to use. Variations are important to remove the uniform tiled nature of a landscape, and should be made for the more commonly used tiles in a tileset.

Planning

With these things in mind, it is possible to design a tileset in it’s entirety on paper… but you’ll find that it’s not as easy as it sounds. If you’re starting a tileset from scratch I would suggest starting small and growing, or looking at a tileset that has already been created for reference. On average, each of the Bioware created tilesets held around 200 tiles, the largest of which were the rural and city environments (which had 249 and 320 tiles respectively). Outdoor situations tend to grow a tileset a large amount. Something else I would suggest for the development of a tileset is to start with very simple geometry for a each tile, so that you can get a feel for a tileset and the numbers involved before you start on the look.

I would estimate that a good quality generic tileset would take one person 3 months to do (barring interruptions). Some of the artists here at Bioware could maybe do it in 2 months, but only because they’ve used this system before and are familiar with the Neverwinter Export Scripts.

It’s important also to remember that while tiles are simply BioWare Aurora Engine models with some specific attributes, I would suggest that you use our system for organization. I will continue this document with the assumption that you are using this system. But, with that being said, you’re welcome to try things in your own organizational method, and maybe even improve on what we’ve started (maybe when you understand our way of doing things J).

Basic Organization & Tileset Models

I will be making example files as I write this document (available in the Tutorial/MaxFiles directory), and I have labeled them TTE01 or Tiles Terrain Example. Follow along this document with these example files, and it should make a little more sense. I’ll start at the beginning and let you know exactly where you will be as we proceed. Once we have seen a new tileset working in the toolset, please read on to learn how to make things a little more interesting.

N.B. These max files were created using 3DS MAX 3.0. If you are using later versions of the program, it will ask you to re-save the file, which is fine. If you encounter problems with the scripts and these files, see the readme.txt I have placed in the Max4.2Max5.0 directory

First you need a name for your tileset. Once you have this name, it's good to stem the rest of your names with this prefix, as then you can tell which tileset a certain file or object belongs to. For this reason we keep it short…

TCN01 is "Tiles City Neverwinter" (originally we were going to have more than one city tileset)

TTR01 is "Tiles Terrain Rural"

TDC01 is "Tiles Dungeon Crypt"

TIC01 is "Tiles Interior Castle"

The "01" in all these names leaves room for expansion, as we could make two different rural terrains… this seems unlikely, but it's also good for version numbers.

Open TTE01_01.max. Here you will see the basic layout approach, with a title at the top (always handy inside the file as max file names can change). Below the title is a simple grid, with letters on the X-axis and numbers on the Y-axis. All of the numbers and words are only important to organize things… these pieces of geometry are not linked to Aurabases and therefore will not get exported. Now take a look at the Aurabase X's. The name of each Aurabase is generated from three aspects:

TTE01	_	A01	_	01
Tileset Name		Grid Position		Variation Number

= TTE01_A01_01

A variation number of greater than "01" only really comes into play when you want to make a variation, which can be placed in a row under the original piece down the Z-Axis. Again, this is purely for organization, but it does help to keep your files straight.

The other thing you can see on these Aurabases in the modifier stack is the Classification of "Tile". This is fairly important, so make sure you have this checked before you copy a large quantity of Aurabases (otherwise you may have to do a lot of clicking).

Also you will see that I have placed Aurabases on all the grid points that were available. This does not mean that I have to use them all, however, as Aurabases with no geometry connected to them do not export a model. And since the naming is irrespective of the game dynamics, we can leave spaces in the grid anywhere we choose. It's for these reasons that I like to keep the Aurabases around, spacing the models as much as possible, as there's always a chance I might want to slip another tile in somewhere, or shift them around.

Now I'm going to choose two corner types. I will choose (for example's sake) "dirt" and "pit". I do not have to model a final version of the tile at this point, so I won't. If you open TTE01_02.max, you will see a mockup of a very simple tileset, made of 6 tiles. These tiles allow you to have areas of pit, areas of dirt and a transition in any grid-like way. Simple, huh? Having similar edges next to each other in the grid means that I lower the chances of having dissimilar edges on the geometry… which is useful, but it's good to keep an eye on the shape of a tile edge at all times.

Try exporting all the models in the file. If you look at the most recent files in your export directory, you should see two files for every one Aurabase that you exported, an mdl, and a wok file. This is because the Aurabase had the classification of "Tile". You should be familiar with the mdl file, but the wok file should be new… it contains the information for pathfinding and hit checking. Why are they all so small? Because they

don’t contain any information. We’ll get into that soon… it has to do with the AABB modifier that I didn’t get into in the Neverwinter Scripts document.

Now I’m going to mock up three more tiles, a single tile group (or feature), and a two tile group. You can see those in the max file TTE01_03.max. Export these to your game directory, and you have the models ready for the next step. We’ll get back to finishing these models after we have successfully seen them in the game.

Below you will find a list of resources associated with tilesets.

File	Example	Purpose
*.set	ttr01.set	Primary Tileset InformationFile
*palstd.itp	ttr01palstd.itp	Toolset Placement Options
doortypes.2da		Tileset Specific Doors List
genericdoors.2da		'Type=0' / Standard Doors
*edge.2da	ttr01edge.2da	Area Edge Model Placement
areag.ini		Tileset default Area Properties
loadscreens.2da		List of loadscreen options
tilecolor.2da		controls available sourcelight colors
footstepsounds.2da		the name says it :)
surfacemat.2da		walkmesh materials
*.mdl	ttr01_a01_01.mdl	A 3d model, in this case either Tile or Door
*.wok	ttr01_a01_01.wok	Tile-Walkmesh File
*.dwk	t_door01.dwk	Door-Walkmesh File
*.tga	MIttr01a01.tga	Texture file, MapImage, Loadscreen Image
*dds	MIttr01a01.dds	as above, but in Bioware-DDS compressed format

Color Coding :

Necessary file for any tileset. Without these, it won`t work.

These files *might* be needed, but the tilest will work without them.

Indicates secondary files, the finishing touches.

These files are for pros that that wanna dig a bit deeper.

Tools needed / recommended for use with these sites...

Name	Purpose	Link1	Link2
NWN-Explorer mod. by Roboius	View and extract all things NWN	NWN-Vault	
GFF-Editor, mod by Roboius	Edit ITP type (itp,utd etc) files	NWN-Vault	
Textpad	Texteditor, edit .set files or any other ASCII file	Homepage	
Bioware ExportScripts, mod by Velmar	modelling, 3dsMax5 or lower	Vel`s Homepage	
Bioware ExportScripts	modelling, 3dsMax5 or lower	Bioware	
Aura Importer	for use with Bio`s Scripts	NWN-Vault	
MDL-Suite	modelling, 3dsMax4 - 6	NWN-Vault (outdated)	Roboius HP

NWmax by Joco	modelling, Gmax 1.2 (get this one if you don`t have 3dsMax)	NWN-Vault	Joco HP
Gmax	Free 3d modelling program	Discreet	

This file tries to explain the various entries in a tileset`s ".set" file.

I use the rural "ttr01.set" for this.

Note that not all lines are explained. That is because I either don`t know what they do or am at least unsure about it.

This is not carved in stone. If you notice an error or think something is badly explained, let me know.

You`ll have to have some basic understanding on how a .set is used to understand this.

; NEVERWINTER NIGHTS TILESET FILE
; DO NOT EDIT MANUALLY - UNLESS YOU KNOW WHAT YOU ARE DOING

[GENERAL]	Begin of General Section
Name=TTR01	The "internal" name. Other files like "doortypes.2da" and "loadscreens.2da" reference to it.
Type=SET	Seems obvious.
Version=V1.0	I edited this once and the toolset crashed.
Interior=0	Set this to "1" to have weatherand day/night FX enabled by default
HasHeightTransition=1	If the tileset uses raise/lower and looks for height=1 in [TILE] description
EnvMap=ttr01__ref01	The default environment map (used for water textures and the like)
Transition=5	The height difference of model placed (in meters) at Height 0 and Height 1. This only describes the models height, and does not change anything in the tilemodel itself.

Elevation and the Height Transition Setting

NWN works on an engine that is really only 2d. Unlike many other games today, you cannot walk under a portion that you can also walk over. Ie. there is only one level of walkable game. You can't hide up above another player on a ledge and jump down on them as they come from a hallway under you. You can't walk over a stone arch bridge while another unit moves under the bridge (without some interesting scripting). The game simply isn't made for it.

As I mentioned before, the tileset only lets you raise or lower tiles by +1/-1 of a pre-set height transition in relation to it's adjacent tiles. This means you can't have elevational changes that vary by 2 meters here, and then 3 meters there, and then 5 meters way over there. So no areas with big cliffs and smooth hills, unless you do more work. Can it be done? You probably know it already has. Is it easy? Well, like with everything else about NWN, it has some fundamental limitations. With every type of terrain change, you need to make an increasingly exponential number of mixing-tile types. You can mix water with flat ground, or water with a cliff, or water with flat ground AND a cliff. You can basically mix any four things at once (one per corner of a tile).

Elevation changes can either be a terrain type, or it can be a height transition. Depending on your area's needs, you might need a single elevation change, and you might use it often to create a slope, so you'd use the height transition. But you might also want multiple heights in your area, all of different levels, and then you want

to also transition directly from flat ground to a really high cliff, so you'd make all your height transitions terrain types.

Can you mix the two? Yes. Does it have some issues? Let's just keep with the theme: yes it has issues. At some point, your height transition and your terrain-type elevation changes will run into an inescapable issue, which is that they become incompatible at a certain height combination (two height transitions). I spent an entire year trying to work around this and found that the game has no issue here, it is the toolset. I won't go into that further here, but if you go so far as to attempt what I did, and you want the scoop, just ask.

So what is the best height transition value? Well, many of the tilesets in NWN use 2 meters, or approximately the height of a human-sized bounding box. I've also used both 1 and 4. For any other height transitions, you might as well just use terrain types to make your elevation changes.

DisplayName=1606	The name displayed when selecting a tileset in the toolset. It references an entry in dialog.tlk. Change this to "-1" and add a line "UnlocalizedName=MY_TILESETS_NAME" to make that name appear instead.
Border=Grass	The Terrain of the Area`s edges
Default=Grass	The default terrain that is placed everywhere when creating a new area using this tileset.
Floor=Grass	The terrain placed when using the "eraser".
[GRASS]	The settings for the grass that appears if a walkmeshnode of the tilemodel is set to "3 / grass"
Grass=1	Set this to "0" to turn of grass completely. You can then remove the following lines.
Density=5.000	How strong the grass grows.
Height=1.200	The height of the grass in meters.
AmbientRed=0.500 **AmbientGreen=0.600** **AmbientBlue=0.300**	The colour settings of the grass. RGB settings.
DiffuseRed=0.500 **DiffuseGreen=0.600** **DiffuseBlue=0.300**	
[TERRAIN TYPES] **Count=3**	The number of terrain types used in the tileset.
[TERRAIN0]	First Terrain entry. Make sure ALL count entries in your tileset begin with "0" and the last entry is count-1
Name=Grass	The name of the terrain as it will be used in the [TILE] entries later. (CASE SENSITIVE !)
StrRef=63635	The reference to name-string in "dialog.tlk" leave this out if you create a custom terrain and use RESREF

RESREF
Grass
STRING
NAME
MyTerrainName

with the ITP-Tool

You can use this combination with CrosserTypes, too.

Terrain and Crossers

Your area is made up of one or more terrain types, patched together like a quilt. Crossers are like a neat pattern you can add between the squares. Terrain is what is at the corners of your tile, and crossers are what is on your edges. Crossers can be a type of terrain, like a stream, or they can be an obstacle, like a wall. Crossers can also match one of your terrains, but not too many people make use of that.

Common terrain in NWN includes:

1. Basic floor, or the primary terrain style of the area, such as trees in a forest
2. Secondary or more decorative floor type
3. Water or Lava
4. Pits and Chasms
5. Mountains and unreachable areas, commonly used as edges

Common crossers in NWN include:

1. Walls/anything you can attach a door to
2. Corridors/thin hallways
3. Streams
4. Roads/paths

When crossers pass between two terrain types, they often make some kind of separation. But in some cases, they change the functionality of the terrain they cross through. For instance, a stream crosser which passes between two edge terrain tiles (such as in the vanilla rural set) makes something you can use as a door. The same with a road crosser and edges.

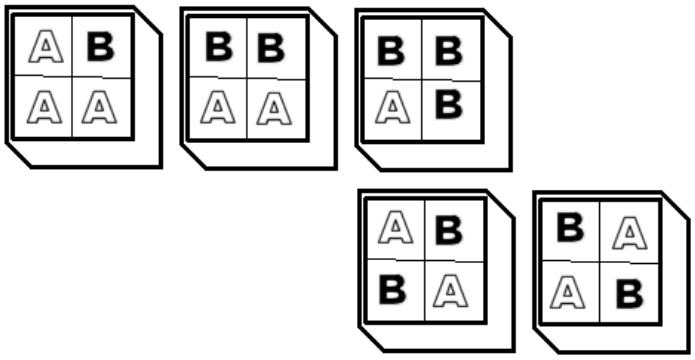
Likewise, when crossers meet in the middle of a tile, they may become something else. For instance, when a stream crosser meets with a blank crosser in the middle of a forest tile, it can make a spring (the water has to come from somewhere). Another example: when a stream and a road crosser meet, they make a little bridge.

For every one of these possibilities, you need to make at least one tile to suit the occasion. If you don't, then the toolset won't allow the crossers to meet. The same is true of terrain. If you want your mountain edge tile

to be able to meet a forest edge tile, then you have to have a tile for that, or the toolset won't allow you to attempt it. Height transitions are handled the exact same way

OMG! How much work is this going to be?

Maybe not as much as you think. Maybe more. Let me show you the 6 basic tile types: Full, Half, Quarter, Three-Quarter, Diagonal, and Inverse Diagonal.



The Full Tile

To create any tileset, you need at least one terrain. Let's call that TERRAIN A. You need a single tile where all four corners are represented by that terrain. There you go, you are done. The most basic tileset.

If you want a second terrain type, TERRAIN B, then you need a second full tile with all four corners representing that terrain. But, to actually use TERRAIN B in a map specified as being TERRAIN A by default, then you need a minimum of 3 mixing tiles.

The Half, Quarter, and Three-Quarter Tiles

I think the image above is fairly self explanatory, but just to be sure: If you have a tile with mostly or partially any two terrains, then you need these three tiles, or you can't use any of them. With these three, you cover the possibility of a diagonal touch (corner to corner) or an edge-on touch (the Half tile). The only reason you would not need all three is if you were using TERRAIN B as an edge tile only, in which case you would still need the three-quarter tile, and the half tile, but not the quarter tile. Again, if you omit any of these three (or two in the case of edge-only use) then you can't use TERRAIN B in any map with TERRAIN A because the toolset map editor does not allow what it cannot complete.

Diagonal and Inverse Tiles

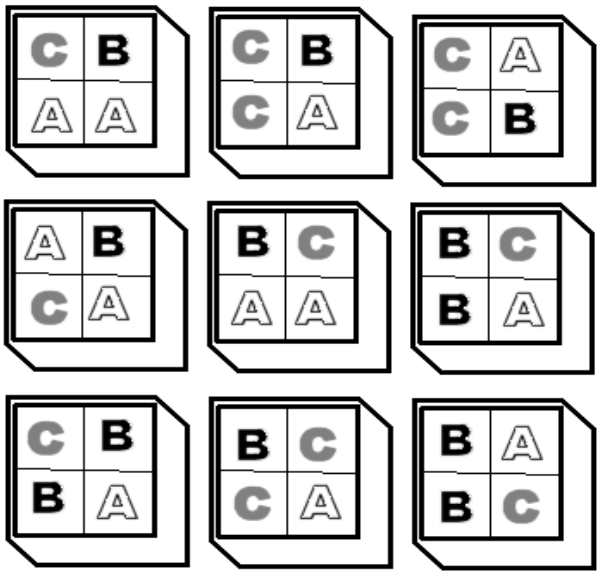
In the event that you want to have two of the same tile intersect with two of another tile, in the pattern of an X, then you need at least one diagonally terrained tile. This means that in the upper left and bottom right corners of the tile, you need TERRAIN A, and in the opposite corners you need TERRAIN B. There are many ways of doing this, but I prefer to make two tiles.

In the first tile, TERRAIN A is dominant in that it is assumed you can walk in the terrain A top corner to the terrain B bottom corner, but it may be an obstacle to transition to terrain B in the other corners. Think of TERRAIN B as a chasm or something with the TERRAIN A corners meeting as a bridge in the middle. In the second tile, I do the opposite: the chasm meets in the middle and leaves two ledges at the opposite corners. I call this the Inverse Diagonal Tile.

Do you need both? No, but if you don't then you have less variety. I actually like to make no less than 3 of any terrain combination tile. In the case of diagonals, I may make two of each instead of having 6 total diagonal mixes.

Mixing Two or Three Terrains

Chart 21

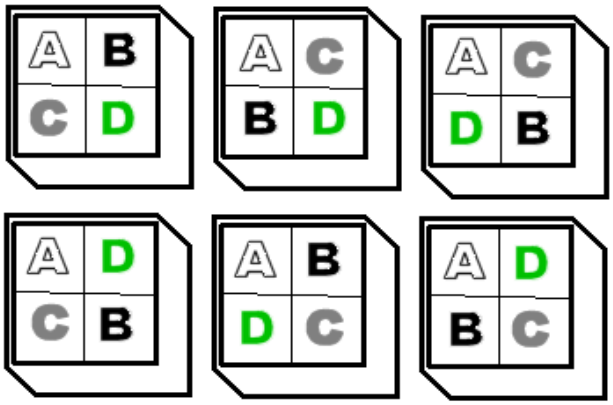


Now, let's say you have TERRAIN C and TERRAIN D, which might be like a mountain and water. To mix three terrains, you need first a set of 6 more tiles, shown in the diagram above, to mix TERRAIN C and TERRAIN A, and than ANOTHER set of 6 to mix TERRAIN C with TERRAIN B, and THEN, you also need another set of nine more tiles shown on Chat 21.

Yeah! That diagram shows that you need to have a pair of mixed-halves for any of the three terrains, and a mixed-diagonal for each terrain. Without these, you can't mix three types. So, what is that now? 1 + 6 + 6 + 6 + 9 = 28 tiles just to mix 3 terrains. Bla!

But NWN lets you mix up to 4 terrain types. To do that, you need another set of 6 to mix D with A, 6 more to mix D with B, 6 more to mix D with C, a set of 9 to mix ABD, another set of 9 to mix ACD, another set of 9 to mix BCD, and then a panel of combinations as shown Chart 22.

Chart 22



Mother of !...

OK, this is probably getting scary for you right now. And this is just 4 terrain types. You can't get much more basic of a tileset than that. So what do we have now? 28 + 6 + 6 + 6 + 9 + 9 + 9 + 6 = 79 tiles! And that is just counting one variation of each combination. If you didn't have respect for anybody making tilesets up until now, then you probably either think they are insane, or you just gained that respect needed to jump into it yourself. Also keep in mind, you can still make special groups and individual tiles that are much like placeables. And we haven't even gotten to tiles with crossers yet.

So as not to scare anybody off that has made it this far, let's just skip the diagrams for crosser creation and move on to another part.

Defining Crossers, Terrain and Height Transitions

In video one, you can see I constructed an obvious height transition. But you don't have to use the height transition that way. You can use it as a terrain type, as I was mentioning in the video. The only difference with a height transition is that the toolset lets you repeat it from tile to tile, going up, up, up from the base level. With a terrain type, you can only use it once and never go higher, without of course making more height transition terrain types.

But how do you decide how to define them? Well, it totally depends on how you want your tileset to be when finished. Lets say you have a very Scottish Isles tileset. You will have large cliffs, and short cliffs. You'll have rolling hills, and some flat places. In that situation, you can either use repeated short cliffs to represent a larger one, and then use larger cliffs as a terrain type. You'll never be able to mix those two except at 1x multiplier of the hight transition. What I mean is, you can never use your very high terrain type adjacent to a high place built up from multiple height transitions. The toolset just won't understand that.

Still not sure what you want to do? Well, remember NWN is made up of both role playing areas and combat areas. For combat, you have specific needs. For puzzles and conversation bits, you may have totally different needs. Let's focus on combat. Many games have large open areas, let's call them community combat areas. Many players may be in the area fighting random creatures. This area should probably be generally open, or should have large open areas with definite dividers. So in this situation, you might want a tileset which has no height transition, but uses one or two height-change terrains for dividers. The player might be able to reach the higher levels, but it is not a requirement for the area.

So how about other area types. Maybe you have discovery type areas. These might have very few monsters, but might have a puzzle. The entire area might be a puzzle that you have to keep traversing. In that situation, you want the area to be visually interesting, because the player is going to see it a lot. I'd suggest a height transition, used multiple times, and then mix with a few other terrain types.

Let's look at some common terrain types:

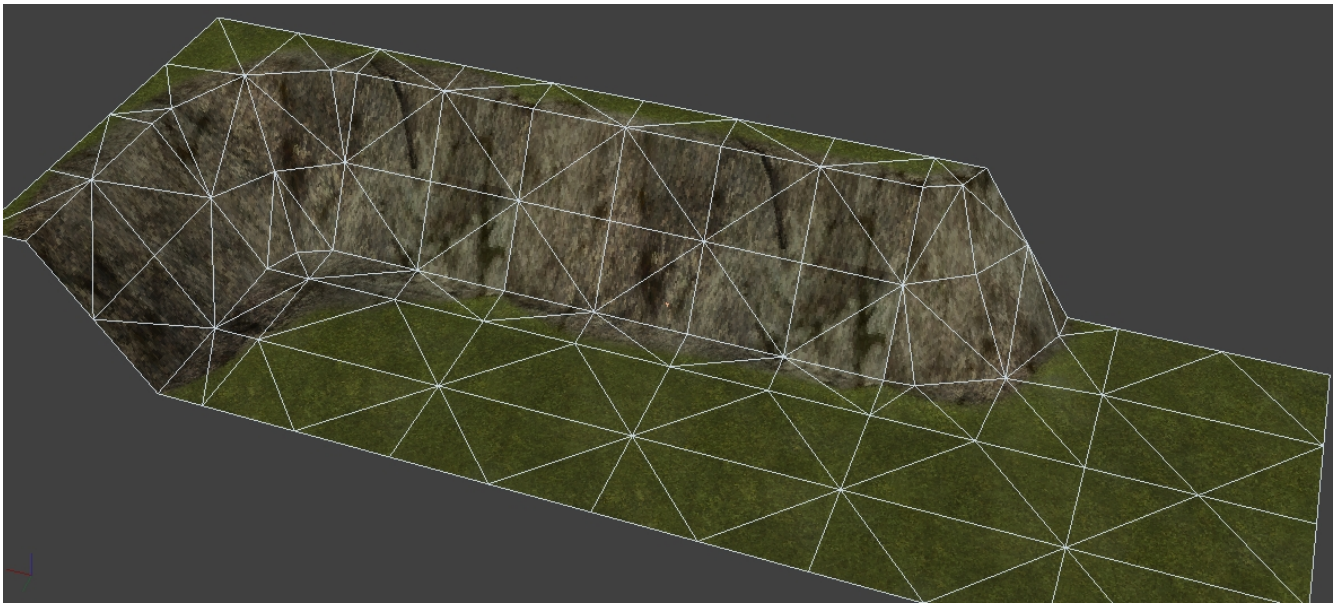
Grass (or base terrain)

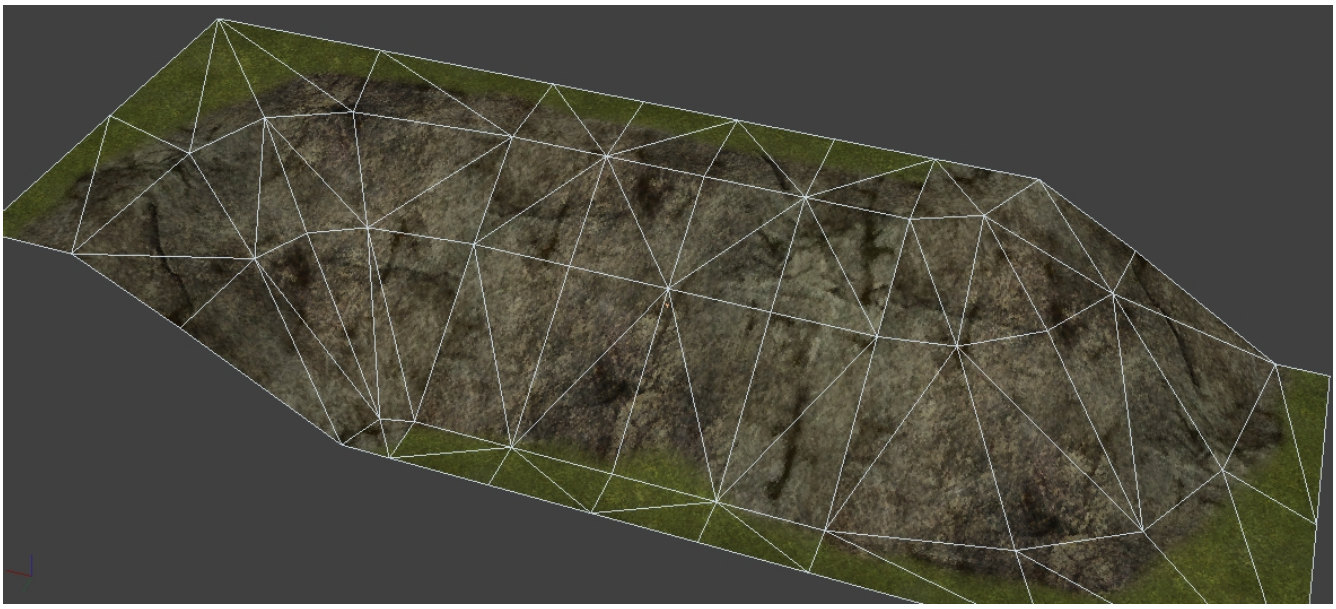
Most outdoor tilesets make use of a base terrain. The area is set up to automatically paint with that terrain type. Vanilla rural areas are simply 0-height transition grassland. Grass is most commonly used also as the height transition terrain type. It is probably best to make your base terrain, whatever that happens to be, the same one you use as a height transition.

Rock (something impassible)

Outdoor and cave-style terrains often have a border terrain type. In many of my tilesets, I simply call it impassible rock. I don't let players walk on it, and I generally use it to wrap my areas so players can't see the engine-dimmed edges (I hate how the engine does that). I also include impassible rock as a crosser. Whenever I want to elongate the unusable area at a height transition, I insert some impassible rock. Let me show you.

...a very basic height transition...





Trees

Vegetation, such as trees, can be used in a lot of ways. First, trees were used in the vanilla rural tileset as an edge type. It could also be used to create separations within the area. There were no pathways which could pass through the forest sections. Second, in the forest tileset, (tall) trees were the base terrain, and so you could walk through the areas dominated by trees.

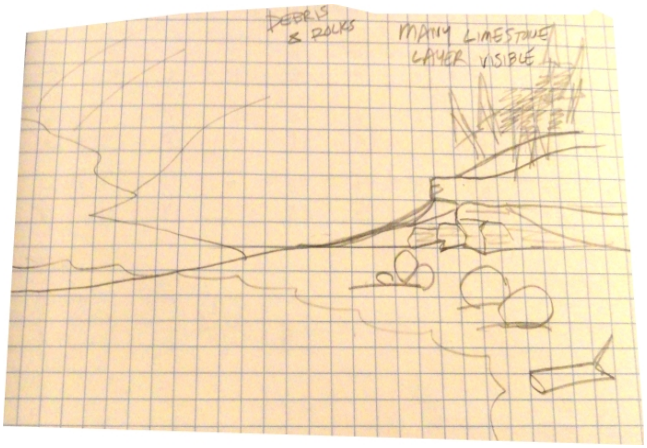
But you can do more with trees. When I was playing Dungeon Seige one summer, I took a lot of interest in how they used trees as both a border terrain, as a mid-tile terrain, and as something that looked more like a crosser. So I started experimenting with trees as a crosser and came up with shrub colonies and fence rows. I made them so you can walk through them, at least most of them, without issues. Both players and monsters

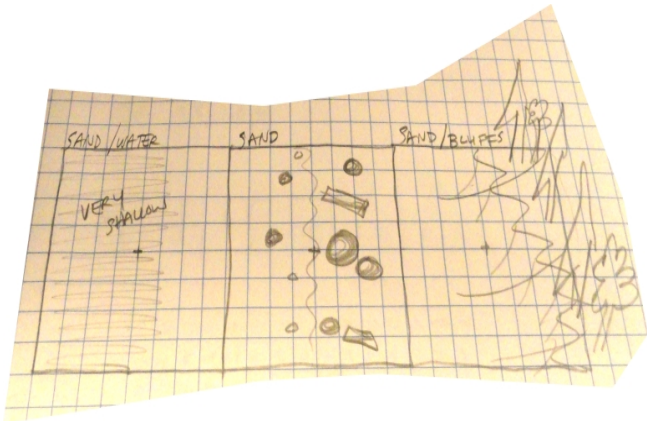
could use them as cover, so I also used them to hide spawn points, especially for creatures with custom appear/disappear animations. Take for instance our scottish isles terrain: a common terrain crosser might be a property/plot separation, such as a fence, a rock wall, or even just a tight line of trees.

Water

In standard rural tilesets, water was immediately so deep you were never allowed to walk in the water. But water was also used as a stream crosser that you could easily walk through. Both were fairly unrealistic in their depths, but I could imagine the world being created by dwarves with digging machines if I stressed my brain hard enough.

When I took on water in 2008, I differentiated water based on its adjacent terrains. You may need instantly deep water, like you might find in a mountain setting, where water collects in natural or man-made reservoirs. But you might also want something more representative of a freshwater lake, in which case, you definitely want a gradual transition. On a trip to Pictured Rocks here in Michigan, I studied and took time to sketch the shorline. I wanted to know how the sand, rocks, berms and the shoreline acted on each other. I found in many sites that you could have a sharp drop from a cliff directly to the water, which might be deep. You could also have a shallow area that stayed shallow for quite some time, and in some cases, you might have a wide beach area. Or in some places, the beach might lead eventually into a berm, or a cut in the area's normal height level.





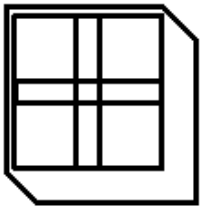
Decide what you need to do with water, and whether or not the player and monsters can walk in it.

Lava and Chasms

Like water, these terrain types might be an obvious divider to the area. Unlike water, there really isn't much possibility that you would want your players to travel into them. Instead, you might want special tiles that cross them, such as bridges. Again, think about if you want a larger portion of your area to be filled with these types of terrain, or if you want to use them as crossers. As a crosser, lava could flow like a stream, or a chasm could be a crack in the ground. As terrain, lava could be a lake, or a chasm could be as deep as the Grand Canyon (or whatever you can see with the fog level set in your area).

[CROSSER TYPES] **Count=4** **[CROSSER0]** **Name=Stream** **StrRef=63302**	Similar to [TERRAIN TYPES], just for crossers like road and stream

So Let's Talk Crossers



To be extremely general about it, crossers are the same as corner terrain, except that they are in the middle of each side. That means, just as with corners, you have four edges, or crossers, to a tile. If we divide the tile icon further, this is what you'd see.

Basically, the number of alternative tiles for any combination of 4 corners just multiplied by a power of two. You can either ignore that little square in the center, or like I do, you can count it as the center section when you name your tiles.

So how would you use this expanded grid? Instead of 4 parts per tile, we now have 8 or 9. Well, let's take a look at streams/rivers as crossers. In vanilla rural areas, streams travel down the center of tiles. To make a complete set of streams, you need these combinations… (Chart 25)

..or to make rivers cross or join, you might want these additional tiles… (Chart 26)

Chart 25

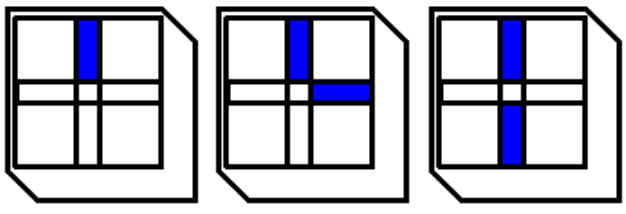
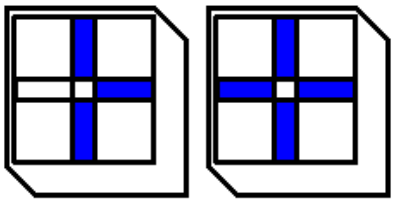
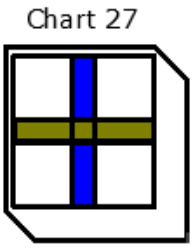


Chart 26

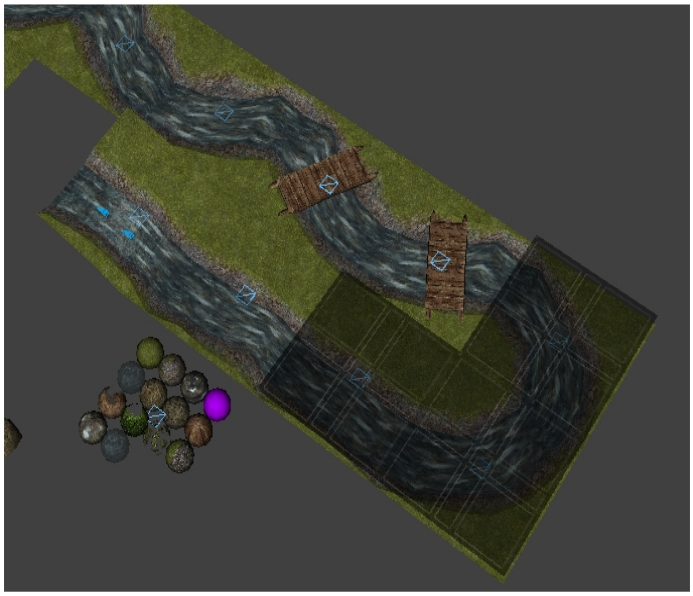


That seems simple enough. And the same set would be required for roads, paths, river variants, and on and on. Five specific tiles and as many variants as you like. But let's say you want a bridge crossing your stream. Would your bridge be a crosser? That depends. For instance, if you have a randomly placed bridge over your stream, but

no real path leading up to the stream, then your bridge can simply be represented as a variety of the stream tile. However, if you have a road leading up to your stream, and you want the bridge to be represented by the road, be able to attach to it, then you'd want a second set of crossers for roads in which meeting in the middle creates the bridge over the river, like Chart 27

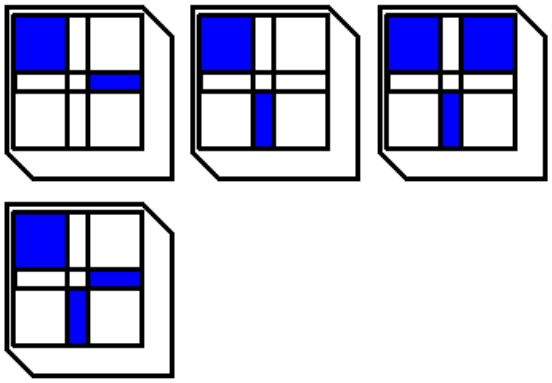


...you would still need the set of 3-5 tiles mentioned above for roads, and the set for streams, and then you would need this one for where they meet. I've overlaid my crosser box icon over some old river tiles I made, see below:



Note that in my river crossers, instead of running the river down the middle of the tile, I ran it along an edge. There is nothing that keeps you from doing this, as long as you have enough tiles made that they line up. See in the picture, I have rivers that run on the right hand side, and rivers that run on the left hand side. But always, when my river goes around a bend, it is on the outside (really just for ease of creation). You can also see I have snaking tiles where the crosser changes handedness. I also have a non-crosser bridge which happens to cross at one of those snaking tiles. Just remember that if your texture moves or otherwise animates in a direction way, that you need two varieties of each tile. One going upstream, and one going downstream.

And if you want a stream to meet at a water terrain, then you might like these diagrams…

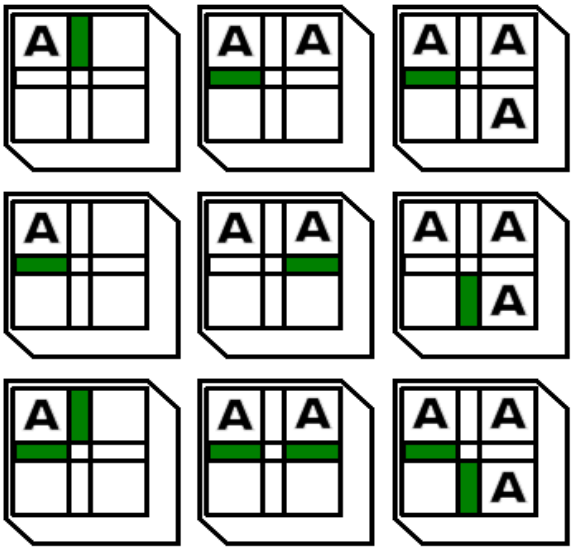


... where the top two on the left are where your river comes into the corner of a lake, or otherwise leaves a lake, and your top right being where the river meets a lake in a more perpendicular manner. The bottom icon could be where maybe a river meets a low spot but continues on its way after exchanging some water with a pond. Can you do more river-water combinations? Sure, but you start to run out of useful tile space when you have a river parallel to the edge of a lake. You need to draw a line for your area, and that is: Are you running a national geographic special for your scenery, or are you building an arena for players to kick the shit out of monsters and bad guys? Find your line.

But get creative with your rivers and crossers. Mix your terrains and crossers and make spectacular waterfalls, which maybe hide a cavern. Make a lava waterfall that pours directly into the ocean. Make springs well out of the side of a rock face. It is completely up to you. In any case, this simple combination of crossers is what you would use for roads, chasms, lava streams, small rivers, paths, walls, and so on.

Ok, but let's say your crossers are for use with terrain changes. For instance, if you want to smooth off the hills we made in the first tileset video, you would need a crosser called "smooth" (or similar). You would then place that crosser at your terrain boundaries to make the hill much less angular.

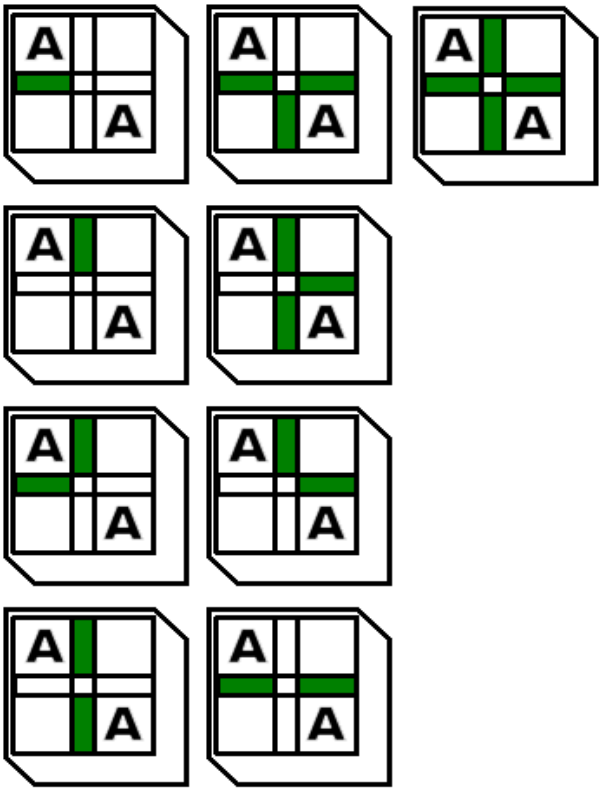
Chart 23



Let me show you on Chart 23 the tiles are your basic crosser plans. When you think of crossers that make terrain transitions, you have to think of handedness. You will need a crosser which handles the right side of any combination, the left side, and then both at once. If you follow the columns down, you can see that the first box handles smoothing on one side of a quart tile, and then the next handles the other side, and then finally the bottom one handles them both smoothed.

Not too hard right? But did you notice I left out the diagonal tile type? Please don't hate me, I am only the messenger. Here are your diagonal crosser plans on Chart 24...

Chart 24



Again, you need right, left and both. But with diagonal tiles you need to take into consideration all four quadrants and their combinations. One more thing...if you have diagonal tiles, and you use the inverse diagonal tile like I do, then you need two sets of these, one for each dominant tile (terrain type). All that, just for one terrain type, in addition to your base terrain. By now, either your respect for tileset makers is growing, or your confidence in yourself to make a tileset is falling fast. Or both.

In my example above, I made crossers out of impassible stone and joined those to my height transitions. In that scenario, I used the exact same crosser patters you would use to make smoothing crossers. You can use these same patterns to add extra sand at a grass-lake border, to make overhangs on your cliffs and rock outcrops, or even to simply change the rock type shown on your rock faces. Do whatever you want with your crossers, and be creative.

[PRIMARY RULES] **Count=28**	The primary rules section. Here is determined what terrain type is allowed to place, and what happens if a certain terrain type is placed on and next to another.
[PRIMARY RULE0]	The first primary rule.Again, watch the count !
	The use of Primary Rules is hard to explain. You got to know that terrain is always determined using the corners (NOT the edges) and if you place something, all tiles adjacent are placed again, too. (That`s why tiles change around a placed one, even if you just place basic floor tiles) According to BioWare`s TilesetTutorial, these aren`t really needed and just add some logical pattern to the .SET ( ?! ) .  I`ll explain it the way I read it:
Placed=Grass	If "Grass" is placed....
PlacedHeight=0	...and the height of the Tile is 0
Adjacent=Grass **AdjacentHeight=0**	...and the adjacent tile`s corner is "Grass" with a height of "0"
Changed=Grass **ChangedHeight=0**	...the terrain at the adjacent gets changed to "Grass" with a height of "0".
[SECONDARY RULES] **Count=0**	UNKNOWN - it`s never used in any tileset
[TILES] **Count=259**	Begin of the [TILE] section. The settings in this section mainly describe the looks and use of a given tilemodel. We`ll use "ttr01_a11_01" for this example.  The same tile from above and aligned to fit [Tile] entry....

[TILE9]	The number of the tile entry
Model=ttr01_a11_01	The file reference to "ttr01_a11_01.mdl" and it`s walkmesh file "ttr01_a11_01.wok"
WalkMesh=msb01	
TopLeft=Grass	The tile-description section.
TopLeftHeight=0	Take a look at above picture and you`ll understand how it`s read.

TopRight=Water **TopRightHeight=0** **BottomLeft=Grass** **BottomLeftHeight=0** **BottomRight=Grass** **BottomRightHeight=1**	Remember tiles are always read over corners and if you place a tile, adjacent corners are placed, too. This starts a "snowball system" of placing tiles. Note the edges that are left out. It looks like they aren`t affected if you place something, but in fact the tiles there are affected by the tiles placed at the corners.
Top= **Right=** **Bottom=** **Left=**	Here are the entries for crossers. Crossers are additionally used over edges, not only over corners.

MainLight1=1 **MainLight2=1** **SourceLight1=1** **SourceLight2=1**	This section is used to turn the various tile options on and off. The options have to be set in the model, else nothing will happen. If there are no "sl" nodes in the model , no sourcelight (torches) will appear for example. You`ll see the checkboxes for this section if you rightclick on a tile.

Tile Light Sources

There are four lights available on a tile: 2 general tile lights, and two source lights with flames. Not all of these have to be used, but a good rule of thomb is to make sure that the two general lights are always available and only put the source lights where the look would allow. Here’s how to implement

Main light 1: This is the primary light, so make an Auroralight at [0,0,500] above the tile (five meters above the Aurabase). You need to name the light with the prefix of the tile name, followed by “ml1” for main light 1. For example, “TTR01_D03_01ml1”.Use these settings for the best effect:

Main light 2: This is the secondary and smaller than the other light, so use it to highlight areas of interest on the tile, i.e. you can place this one wherever you see fit! The name is similar to the previous except with an “ml2” instead. Here are the settings:

Source light 1: Because these light positions spawn in another model with the light attached, use a dummy box instead of an Auroralight. Make a dummy, and use the tile name and add “sl1” to the end, e.g. “TTR01_Q04_01sl1”. Source light 2: Same as above, with an “sl2”.

AnimLoop1=1 **AnimLoop2=1** **AnimLoop3=1**	This section is used to turn the various tile options on and off. The options have to be set in the model, else nothing will happen. If there are no "sl" nodes in the model , no sourcelight (torches) will appear for example. You`ll see the checkboxes for this section if you rightclick on a tile.

Tile Animation

Animation for tiles has to be handled differently than most other models, as tiles are inherently loaded as static objects. For the engine to know which parts are to be loaded statically and which are to be loaded dynamically, we use the linking/tree system again. We create a dummy object that has the exact same position and orientation as the Aurabase for the tile that we want to have animation on. Then we call it the exact same name as the Aurabase itself, with the suffix “a” (for animation). We link it to the Aurabase, and then link all our

dynamic, animated objects to this dummy. For example, here's what the tree would look like for the tile TTR01_A01_01:

So basically, when the engine loads a tile that has this structure, it filters out all the objects that are linked to this anim dummy and renders them dynamically, and thus is able to use the keys for animation they might have.

There are only certain animation names that will be recognized by the game. The first group consists of general loops for the tile. The end user is able to switch these animations on and off in the toolset, so try and plan them being as generic as possible.

Animloop01
Animloop02
Animloop03

The second group consists of animations for if you want a tile to have different visible effects during night or day (e.g. torches at night)… then you can use these animation names for a tile:

Day
Day2Night
Night
Night2Day

N.B.: If you want to have a loop that is always on whether or not the map painter wants it, then you can use the Day/Night animations. These play as loops.

As a further note on animated geometry, if you are having problems with transparent objects not sorting correctly with other objects then try linking them to the animation dummy. Our graphics engine renders dynamic objects after it renders static objects (per frame), so it can sort through transparent polygons better when they are all done in the dynamic stage. It will calculate the depth of a polygon/pixel based on the axis point of a piece, so watch your center points here too.

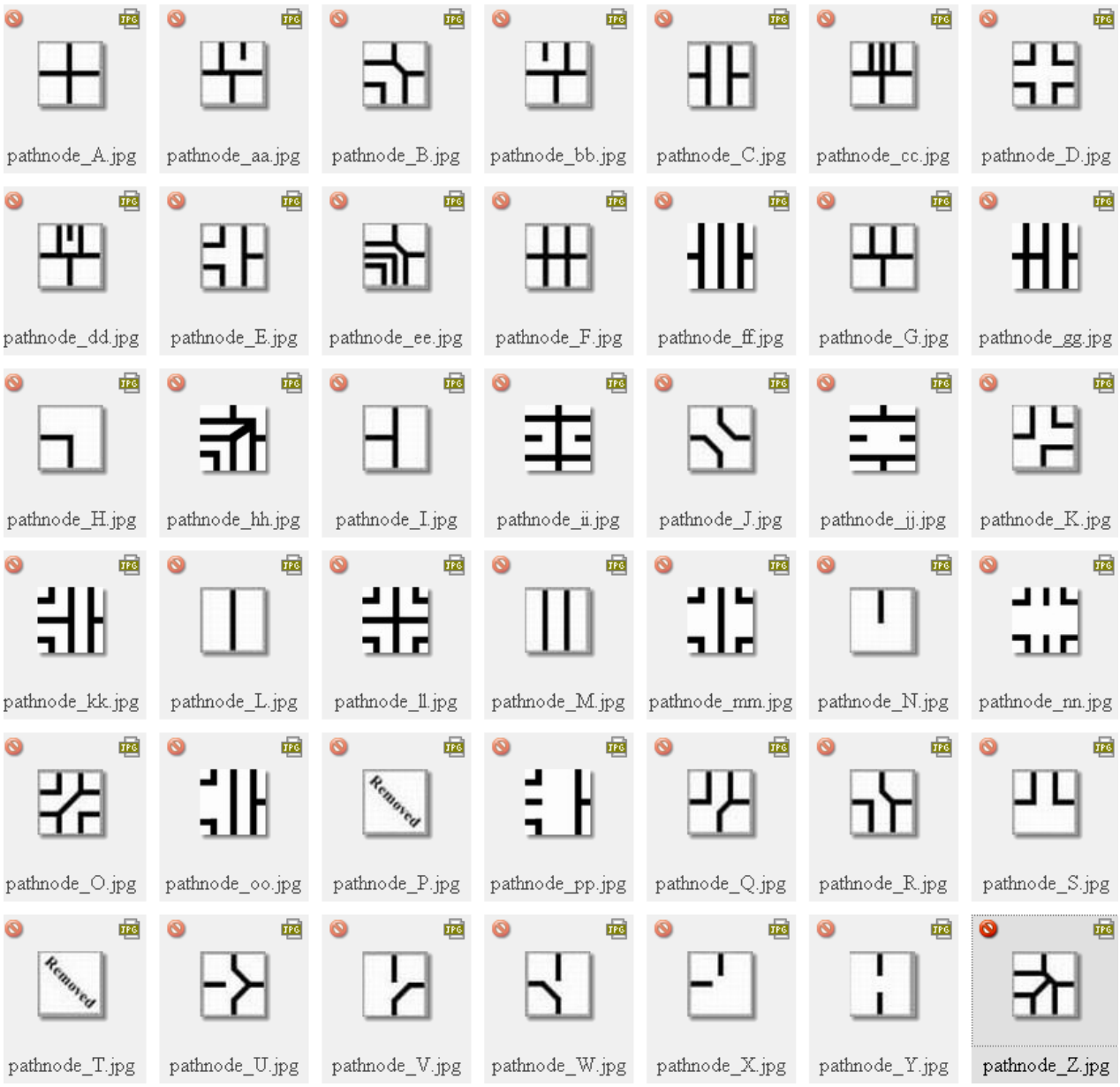
Doors=0 **Sounds=0**	The number of doors in the tile. I`ll do an door example further down....

PathNode=Q	This is a descriptor for the walkmesh type and is used in the pathfinding routines.

Appendix 1: Path Nodes

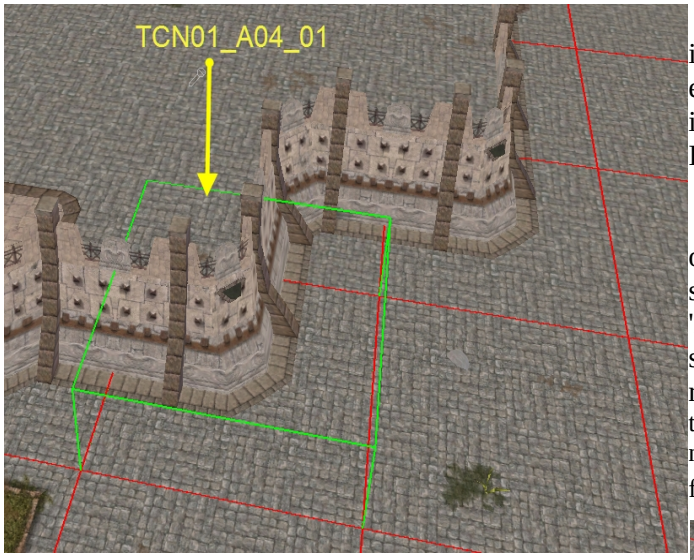
There is no such thing as a 3d pathnode, 3d doesn't mean anything to the game engine and technically doesn't mean anything to the pathnodes. ALL pathnodes are 2d. They can be used in a 3d environment, provided you pay attention to them. The picture below has all the 2d versions listed. Some of these pathnodes can be used for multiple tiles, some flat, some no, and some both…. If that makes any sense.

Once a tile has a walkmesh it can use, the pathfinding will be able to traverse inside this tile. Bioware has found that for large areas, it was better to have a simpler system in place to know if a character could walk from one spot on the map to another. So they implemented a system of exits on a tile and how a tile works in general. Take a look at the picture below:



Example: That “C” pathnode used for that wall section? It could also be used for a non-walk able stream, or possibly for a crack in the ground like a narrow pit.

Each tile that we make has to have a reference to one of the symbols above. Each line that hits a side of the square represents an exit of that tile. In the case of “A”, there is only one line per side, and they all meet in the center. This is the representation of a flat open tile. Take a look at some of these other examples:



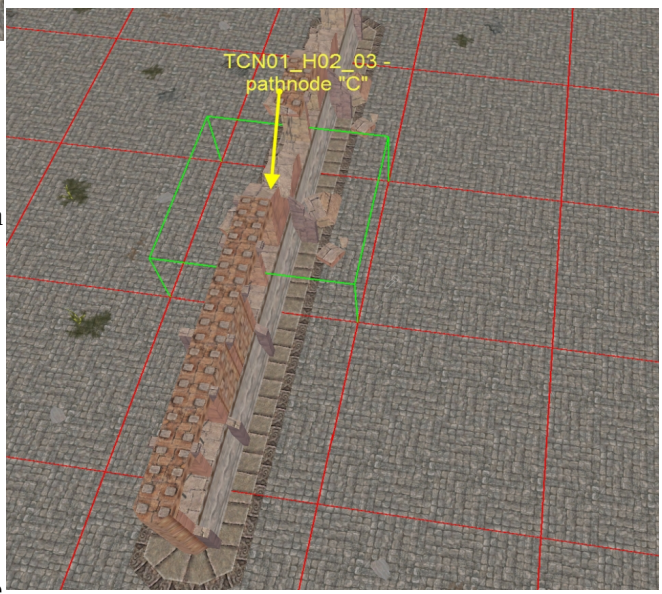
The way they work, is the black lines are what is considered walkable. Most specifically, along the edges of a tile, the points when the black lines intersect those edges, is where someone can walk INTO/ONTO the tile.

As an example, take a look at this screen shot of the standard Bioware City Exterior, with a raised section, on a corner, you would need to use pathnode 'B' (capitol B) to access both the top and bottom sections of this tile. It of course, has to be correctly rotated to match the way the tile is defined as well. IE, if the set starts that tile using non-raised, non-raised, raised, non-raised then the default position for pathnode B works fine.

In the above pic though, that tile is actually rotated inside the toolset, from the camera perspective.

Anyway, the real key is that any line that intersects an outside edge of the pathnode, SHOULD be connected to a matching line in the next tile over. This is handled by the pathnode you choose AND by how the tile is defined in the .set file.

Another example, using TCN01_H02_03 (a wall tile) which uses pathnode "C"



If you take a close look at C, it allows you to walk onto the tile from either side, and from top or bottom, but, that wide white section in the middle prevents you from walking across the middle... just like the wall in the above pic.

Basically, when there are two exits from one edge of a tile, there is a wall or an elevation difference that leaves the edge of the tile. The lines inside the square depict how things are connected inside the tile. You will notice that the path nodes don't account for all cases, but a majority of the missing ones are when you rotate the symbol in 90 degree increments. This is why whenever you enter a path node in our editors, you will be asked for an orientation (a positive rotation being counter-clockwise). Also, if you want to put in a tile that doesn't have any walkable areas on it, then use the "P" or "T" letters, as these have been removed and will be ignored when used.

When you are specifying the nodes for a tile, you have three different nodes available:

1. The first is the general path node and its orientation. This is how the tile would look with no doors on it.

2. The next is the visibility node. This is for things like sound and general “I can see you!” calculations. If all the exits defined in your general path node are because of high walls then the visibility node will be exactly the same as the first. But if there are blocking things that are visible through or over then the node will be different. A straight piece of wire fence across a tile would have a “C” path node, but an “A” visibility node.

3. The last is a door node. When doors occur somewhere in the middle of a tile and effectively changes the path node style when they are closed, it gets entered here. If there are no doors or the doors are only meant for changing areas, feel free to skip this entry.

Take a look at the TTE01.set file to see what flags I have given the tiles in our tutorial exercise

Orientation=180	Turn the Pathnodes with this (in +/- 90 degrees) to tell the engine how the tile can be passed through. Positve values are Counter-Clockwise Rotations, Negative Values are Clockwise.
VisibilityNode=A	Same as PathNode, but for Line-Of-Sight only - Can be left out if same as PathNode
VisibilityOrientation=0	Same as Orientation, but for Line-Of-Sight only- Can be left out if same as PathNode
ImageMap2D=MITR01_A11	This is the reference to the image that is displayed on the map for this tile. (MItr01_a11.tga in this case) It should be 16x16 px , 24bit TGA.

ON DOORS

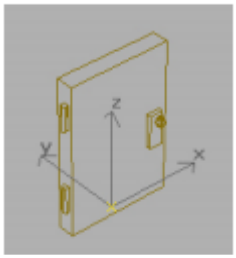
Remember the Doors=0 entry above ?	
Let`s assume it would have been Doors=1 or more.	
In that case the [TILE] entry must be directly followed by	
[TILE9DOOR0]	This is the entry for the FIRST door in a given tile. The first part "[TILE9" references to the tile, while "DOOR0]" is the sequential number of the door. If there would be more than one door in the tile, the next would have to be [TILE9DOOR1] and so on.
Type=65	This references to the entry number in "doortypes.2da". If it`s not "=0" it is a TilesetSpecific door. Entries of "=0" let you place doors from "genericdoors.2da" (simple doors). Tileset specific doors get auto-placed when the tile is placed, while you have to place simple doors by hand.
X=2.78 **Y=0.00** **Z=0.00**	The position of the door. Values are in meters.
Orientation=-90.0	The facing of the door.

Doors

I want to warn you that this is a generally offensive system, and I do apologize right off the bat. That being said, as long as you have the correct tree structure, the correct names for objects and the 2da updated with your door, you needn't know why things are the way they are, they will just work. If you could care less about the details, I invite you to try and skip any techno-babble and go straight for the meat.

Firstly, there are two different types of doors: generic and unique. Generic doors are simple 3m x 2m doors that can be used whenever and wherever you like. Most building doors and edge doors use this kind of door. Unique doors are ones that are created specifically for one place (but can be used multiple times). Examples of this type of door would be the city gates, the crypt corridor doors and the evil dungeon fence doors. So the only real difference is that some door slots are open to generic doors while some only allow one type of door, and thus the actual making of the door in 3DS MAX differs little. I'll get back into how to enter them into the system after I've told you how to make them

An important thing to note is the orientation of a door relative to its Aurabase. It should resemble this:



..... where when the door is closed, the x-axis is the length of the door and the y-axis shows the direction that the door blocks. For some tiles, the direction you will be blocking with a door might be the x-axis. In these cases you can either rotate the door geometry after you've created it, or rotate the Aurabase in 90 deg increments.

Door Model Structure

Here is an example of the tree structure you'll need… I have supplied this door in the example files, just so you can see how it works:

TTR_UDoor_01
- + [Visible Mesh]
- +TTR_UDoor_01_DWK
  - +01_DWK_dp_closed_01
  - +01_DWK_dp_closed_02
  - +01_DWK_dp_open1_01
  - +01_DWK_dp_open1_02
  - +01_DWK_dp_open2_01
  - +01_DWK_dp_open2_02
  - +01_DWK_wg_closed
  - +01_DWK_wg_open1
  - +01_DWK_wg_open2
- +sam
- +TTR_UDoor_01grnd
- +TTR_UDoor_01head

+TTR_UDoor_01hhit
+TTR_UDoor_01impc

First, I would suggest copying the naming scheme as much as possible from here on in as the exact count and position of letters is important. The most important part is to have the last two characters as numbers (in this case “01”).

Second, you should know that there are three states for a door. Opened inwards (open1), opened outwards (open2) and closed (closed).

So, what is each object for?

TTR_UDoor_01 is the name of the aurabase for the door. This important to note as the door will get exported as a single entity.

The [Visible Mesh] is obviously linked to this aurabase.

TTR_UDoor_01_DWK is the name of a dummy placed at exactly the same position as the aurabase. The name “_DWK” refers to Door Walk, as everything under this node is for pathfinding purposes.

Now, you will notice that everything under this node starts with a “01_DWK”. This is important to follow through on as the last 5 characters of its parent. Anything with a “dp” is a dummy box that is placed on the ground where the character should stand to do things.

If the door is closed and the player wants to open it, he/she will walk to the position of either “01_DWK_dp_closed_01” or “01_DWK_dp_closed_02”, depending on which is closer and not blocked. These two are placed on either side of the center of the door. Obviously you will need both dummies for the door to work from both sides. If the door is opened inwards, then he/she will go to “01_DWK_dp_open1_01” or “01_DWK_dp_open1_02”, thus they are positioned on the outer tips of the door/doors. If you only have one door, you only need one node (“01_DWK_dp_open1_01”).

The same is true for the outward swung door, substituting the open1 for open2.

Anything with a “wg” is for walk geometry. In order for the character to be blocked by the door, there needs to be geometry there that he cannot stand on. It uses a system similar to the walkmesh, in that it uses the same multi-sub texture to determine what polys are what. Create some loose boxes around your door/doors for the three states of open1, open2 and closed. It would be best to make these boxes wide enough to fit the frame and then some, as you want to make sure that the door blocks correctly. Name them similarly to above, and apply the walkmesh material with the seventh material tag (or non-walk), and link them to the DWK dummy. Please check your names carefully as everything is case sensitive as well.

Next, the object called “sam”. After making such an effort to have every object in the max file have a different name, here is the only object that got through. And if that wasn’t bad enough, it was called “sam”. Please don’t ask me why. What does it do? It’s quite simply the object that you click on when the door is an area transition. In the game, you can cursor over a door, and it will show a blue polygon (from both directions). This is “sam”. In order for sam to work, it will also need an animation to reference it… see later for how to do this.

The last group of objects are concerned with spells and impact nodes. They imitate the ones that characters have, so that doors will not only receive spells but can cast them too. Again, make sure to name them correctly.

Animations

That's it for the actual objects. But in order for the visible geometry to swing/slide in the way they should, you'll also need animations. The door should use these animation names:

closed
opened1
opening1
closing1
opened2
opening2
closing2
trans
die
dead

The first 7 should be obvious. "trans" concerns the sam object. The way we do it is have a blue self-illumination on the aurapoly for that object. For most animations the poly is at 0.0 alpha (transparent), but for the two frames of the "trans" anim, the alpha changes to 0.25.

"die" is the anim when the door breaks apart, or falls through the ground, whereas
"dead" is for the state where it is lying in pieces or gone.

Appending the 2da

So now you have a door model. It will need to be entered into a 2da…

For Generic Doors:
Open "genericdoors.2da", copy a line from a similar door and append the 2da with your model.
For Unique Doors:
Open "doortypes.2da", copy a line from a similar door and append the 2da with your model.

Placement/Orientation of Nodes on Tile

Your model will also need a place to be spawned, which is handled by placing a dummy object on the tile where you want it to occur. Again, this requires two different systems based on whether the door is generic or unique.

For Generic Doors:
When you place a dummy on the tile and link it to the Aurabase, you have to name the dummy with the prefix of the tile name, and the addition of "_D01", "_D02" and so on for up to four nodes. The orientation is important!

For Unique Doors:
When you place a dummy on the tile and link it to the Aurabase, you have to name the dummy with the prefix of the tile name, and the addition of "_U", and then the last two numbers of the unique door's name! It will look like this, for example:

“TTR01_A01_01_U02”, which will use the door “TTR_UDoor_02”.

(N.B. If you want multiple tiles to use the same unique door model, then you can use the same last two digits.)

Again, the orientation of the dummy is important.

genericdoors.2da

This file lists all generic doors.
Generic doors are those doors that can be placed when a Tiles Dooretry is set to Type=0....
For Example:
[TILE7DOOR0]
Type=0
X=2.50
Y=0.00
Z=-0.40
Orientation=90.0

They are listed in the toolset under the door placement options.
The genericdoors.2da is a child of doortypes.2da, technically it replaces line0 of that file.

Let`s take a look at some genericdoors.2da lines...

	Label	StrRef	ModelName	BlockSight	VisibleModel	SoundAppType	Name
1	Wood_strong	5349	T_DOOR01	1	1	0	63750
2	Fancy	5349	T_DOOR02	1	1	0	14445

(The list goes on)

So what do these collumns mean ?

#	The numbers are the *sequential* entrynumbers. Keep them sequential !
Label	This is for editors` eyes only, it has no use ingame.
StrRef	The type reference, keep it to 5349, else it won`t work. This is hardcoded.
ModelName	The actual model without the .mdl extension.
BlockSight	If a player can see what is behind the door, and if it blocks casting etc. Options are '0' and '1'
VisibleModel	Display the model ingame yes/no ('0'/'1')
SoundAppType	This is the sound that is played when the door is opened. It refererences 'placeableobjsnds.2da' entries.
Name	This is a reference to dialog.tlk and is the name of the door as displayed in appearance in the toolset.

doortypes.2da

This 2da contains all information needed for tileset specific doors.
These are, for example the door in the rural walls or the city`s tower building.

The .set references a linenumber of this list in a door entry.
For example:

[TILE7DOOR0]
Type=68
X=2.50
Y=0.00
Z=-0.40
Orientation=90.0

The 'Type=68' references entry number 68.
These doors get placed automatically when you paint the tile, unless the type is '0'
If the Type is '0' then a generic door can be placed, these are the doors in the
toolset door item list.
This door is from the list in 'genericdoors.2da'.

Back to doortypes.2da.
Like every 2da it can be edited with any texteditor (textpad is a good choice, but notepad will do).
Let`s take a look at the collumn items.

	Label	Model	TileSet	TemplateResRef	StringRefGame	BlockSight	VisibleModel	SoundAppType
0	Generic	****	****	****	66717	****	****	****
1	Wall1Door	TTR_UDoor_01	TTR01	nw_door_ttr_01	63491	1	1	1

(This list goes on)

So what do the collumns mean ?

#	The numbers are the *sequential* entrynumbers. Keep them sequential !
Label	This is for editors` eyes only, it has no use ingame.
Model	The model that is place in the doors position, without the .mdl extension.
TileSet	What tileset the door can be used in. If you use a door in a tileset that has not this name, you`ll get a 'model not found' error. Enter '****' to make a door available in every tileset.
TemplateResRef	This is a reference to another file, a .utd file. It holds the doors properties like hardness, scripts etc. Think of it like an .erf for doors. If you create an utd in the toolset and pack it to your hak, then reference it in doortypes.2da, every door of this type will have these properties.
StringRefGame	This is a reference to dialog.tlk and is the name of the door as displayed in appearance in the

	toolset.
BlockSight	If a player can see what is behind the door, and if it blocks casting etc. Options are '0' and '1'
VisibleModel	Display the model ingame yes/no ('0'/'1')
SoundAppType	This is the sound that is played when the door is opened. It refererences 'placeableobjsnds.2da' entries.

[GROUPS] **Count=63**	The Groups section and it`s count. Everything that is not bound in a group is subject to be placed either through Terrain or Crosser placement and is either considered a Feature or Group, based on your ITP entry.
[GROUP3]	The FOURTH group entry (remember, 0 is the first !)
Name=DragSkel_1x2	The name of the group. It`s "for your eyes only" as it`s not used elsewhere.
StrRef=63637	The reference to dialog.tlk entry. You can leave this line out if you use a STRING NAME combination with ITP tools.
Rows=1 **Columns=2**	The size of the Group. When placing a group it first reads all row "Tile=" entries and then the collumns. (hope to be right on this)
Tile0=127	The first Tile to be placed with a group and the tile that you have to reference to with the ITP-entry. So look up the model name of [TILE127], what is "ttr01_o06_01" And create your ITP: RESREF RESREF ttr01_c06_01 STRING NAME DragonSkeleton_1x2
Tile1=128	The subsequent tiles. Once a tile is bound, it`s unavailable for Terrain/Crosser placement. A small trick: if you set a "Tile#=" to "-1" (example: Tile3=-1), a random tile will be placed, according to terrain.

Edge tiles

At the edge of a map is a dead zone of tiles that we do not want to have to send between the server and the client, so we came up with a system to make this space generic and flat shaded, but still look like they were part of the world. Edge tiles are more like models than tiles, they have no walkmesh, the have no lights; they are as simple as they can be. If you want to see some examples of our edge tiles, they are all available on the “Z” column of each of our tilesets. It’s important that these tiles DO NOT get entered into the .set file. They have their own file that tells about their information. First though, you have to make the geometry, which is a tile that has all the variations of corner types and edge types for one edge that stretch over the length of the tile and has the same edge on the other side. So for example, if you have two corner types (grass and water) and one crosser type (wall) for a tileset, then you’ll need these edge-of-world tiles:

1. A flat grass tile .....

2. A flat water tile .....

3. A straight grass to water tile .....

4. A flat grass tile with wall going all the way across it .....

You’ll most likely have already built tiles like this anyway; you can just copy the geometry and give the Aurabases a new name. It’s also important for the edges that you are representing to be on the left side of the tile. They will automatically be rotated 180degs for edges that are backwards.

To get them used in the game, you have to make a 2da file, named the same as the tileset itself, with the addition of “_edge”, so the one for the rural tileset is “TTR01_edge.2da”. An example of this kind of 2da would look like this:

2DA V2.0

	Corner1	Edge	Corner2	Height	Model
0	Grass	****	Grass	0	Temp1_Z01_01
1	Water	****	Water	0	Temp1_Z02_01
2	Grass	****	Water	0	Temp1_Z03_01
3	Grass	Wall	Grass	0	Temp1_Z04_01
4	Grass	****	****	0	Temp1_Z01_01

...

Corner 1 is the top left, and corner 2 is the bottom left. Look at line 4, where there is a “****” in the Corner2 column. This line is for the four corners of the map… if it only has one corner to fill, it will only check the Corner1 column. You would need a line like this for every terrain type.