Procedural Tools Example: Open World Manager

For one of my prior jobs I created this complex tool and I will explain it in the following. The underlying pipeline was a procedural world generation, where we would create hundreds of configuration files. The most basic type would define what objects are placed under certain circumstances, the configuration object layer on top would define Sub Biomes (so to say group the objects), then there would be biomes (which would group the Sub Biomes) and in the end maps, which are collections of biomes. This workflow would allow numerous people to work together at the same time, setting up the biomes, painting them out (size, density, growth and type) without having to think what should be placed or how many. But it came with one big downside: a huge amount of setup overhead and the connected complexity to manage it.

The tool I'm going to present here had the following requirements:

- know about all setup files without the user pointing it to it
- search through all files based on numerous criteria
- add, remove or create setup files
- multi-object edit those setup files
- check for sanity of those setup files
- full perforce support

Along the way a stretch goal appeared that overall fit logically quite well into the tool, even though it was quite different from the remaining functionality: placing VFX and Audio in the world. Here are the major requirements for this sub-tool:

- place audio and particle system based on the properties of the placed mesh, e.g. falling leaves VFX with every tree of a certain size
- avoid large quantities of actors
- allow for controlled spacing between audio and VFX emitters, e.g. every 100m one emitter
- communicate the placed objects information about the mesh, type of tree or size
- allow for manual placement of the audio / VFX emitter, e.g. bird audio sound on top of the crown or ant VFX emitter on the bottom of the grass

Here are some general thoughts when I got to know about all those requirements:

Due to all the different functions, I chose a tabbed layout. It allows for good categorisation and is very flexible in case more needs to be added. There will be a lot of buttons involved, color coding them is an excellent way to guide the users' eye. I chose the following concept:

- blue: chosen tab highlight
- green: non-destructive action
- red: destructive action
- pink: lengthy non-destructive actions
- white: inactive tabs and support / setup buttons

To avoid lots of scrolling through the lists, I added wherever I saw fit auto scrolling functions.

Such a large tool will have an incredible amount of potential errors. So to make the user aware of those more conveniently, I wanted to add its own console, including export and filtering functionality to help the Tech Art department to track down issues.

Adding new meshes to the system

This section allowed for selection of any mesh in the content browser to be able to assign it conveniently to any biome setup. 'Faulty' setup like missing setup, duplicate categories or 0% probability would be highlighted and auto-scroll was added.

	Mesh	Biome	Appendix	Console	Help	
			Add Search			
Add Se	lection	Add Entries	Amount Cle	ar		
▶ Defau	lt Entry Settin	gs				
Add T	o Biome(s)	Replace in I	Biomes Override	e All Scroll To	Faulty	
Static Mesh	Settings	Algae_01 🗸				Ū
Map W Biome Subbiom Biome La Probabili	/L01 V Plains V e Forest4 V yer GroundCo ty 1.0	Bi ver ❤ ↔	ome Categories + Biome Category Default	Clear -		
Static Mesh	Settings	Algae_02 🗸				Ű

Searching for existent meshes

Search functionality would allow for any type of property to be looked for. The found setup files could then be highlighted in the content browser or deleted in bulk as well as moved to a different biome.

	Mesh	Biome	Appendix	Console	Help
			Add Searc	า	
🔻 Searc	h Settings				
	Hap WL01 Biome Pla Subbiome Biome Layer Probability Name	Forest4 V GroundCover V 1.0 Statio	: Mesh None	ne 🗸	
Sear Rem	ch To T ove All R	op To Bo emove All Select	ttom		
	Static Mesh Biome Set 	ttings	V_Grass_L_01 ∨		Ū

Biome Values

The values tab allows for checking any setup of a settings file. Every properties file had three different masks, one for its growth (is it getting scattered?), scale (what size should the scattered object have?) and alternative (should it be a dry version of the same biome or maybe snowy?).

	Mes	sh	Bi	ome	A	ppend	ix	Conse	ole	Help	
	[Val	ues	Adm	in	Sani	ty	Sanity S	Setting	js	
🔻 Biom	e Search										
Map Biome	WL01 🗸 Plains 🗸										
Subbiom	e Fores	t4 🗸	¢								
Biome La	ayer Gro	undCov	ver 🗸								
Show (Growth M	lateria	l)	Show Sca	ale Ma	aterial	Sh	ow Alternati	ive		
Search				Show Spe	ecies						

Due to the nature of the system, there were a lot of duplicate properties (different masks depending on the same noise settings for example). This tool merged them into one entry but when set, would always override every similarly named entry altogether, avoiding lots of human error in the process.

 Noise
NoiseScale 1
NoiseMin 0
NoiseMax 1
FineNoiseScale 1
FineNoiseMin 0
FineNoiseMax 1
NoiseDisable 1
FineNoiseDisable 1
NoiseSeed 1 1 3.1
FineNoiseSeed 1 1 3.1
▶ Scale
Weightmap Influence Strength
Rotation
▶ Misc

Biome Property Controles

This area allows for interacting with existent / new parameters for any singular or set of settings objects, allowing for quick multi-object-editing which would otherwise be not as convenient.

	Values	Admin	Sanity	Sanity Settings	
Action					
Mask 🛛 Growth 🗸	Action T	ype Add 🗸			
Parameter Type	Weightmap 🗸	S None	arameter 🗸		
Parameter Name	e Parameter N	Nam Replace			_
New Mask New	/ Mask	Renam Remov	Like 'Add' but ov	erwrites pre-existent entries	
Add					
Targeted Sp	ecies				
Add Entry	Clear Speci	es Category C	ommon 🗸	Add Full Category	
 I Plains - For 	est1 - Ground	Cover 1 💼			

Biome Sanity Checker

Since the whole system depends on massive amounts of properties which by default would need to be added manually, this tool points out exactly which ones are missing. Naming was the most crucial part, so it would not only point out missing properties but excess ones as well. Maybe there was just a typo.

	Values	Admin	Sanity	Sanity Settings			
		Current Typ	e Common 🗸				
Update							
Bushes 🔍	Growth m Scale ma Scale ma	aterial does terial doesn terial doesn	sn't need m 't need mas 't need mas	ask FILLME sk FILLME sk BushesNoiseOver	rride		
Groundcove	Groundcover C Growth material doesn't need mask FILLME Scale material doesn't need mask FILLME Scale material doesn't need mask GroundcoverNoiseOver						
Trees_Sapli	ngs 🔍 (Growth mat Scale mater Scale mater	erial doesn' ial doesn't i ial doesn't i	t need mask FILLME need mask FILLME need mask TreesSa	E plingsNoise(

Biome Sanity Settings

The biome sanity settings contained all the naming parameters, important settings and any other general logic for the system to make sure it exists.

		Values	Admin	Sanity	Γ	Sanity Settings	
			Current Type	Common	~		
-	Default						
▶	Indicators		10 Array elem	nents	Ð	τ ΰ	¢
•	Growth Paramet	ter					¢
	 Handpainted 	Masks	2 Array eleme	ents	Ð	Ū	¢
	Index [0		NoiseOverri	ide 🗸 🗸			¢
	Index [1		Density	~			¢
	Weightmaps		6 Array eleme	ents	Ð	τ ΰ	¢
	LandscapeM	asks	3 Array eleme	ents	Ð	Ū	Ś
	BiomeCheck		~				
►	Alternative Para	meter					¢
►	Scale Parameter	r					
►	Material Mappin	ig	3 Map elemer	nts	Ð	ū	¢

Creating appended components

The idea was to set up differently spaced object grids, where based on the properties defined in the optimisation and audio settings, the user could define that, e.g. every 10 meters there would be a bird and every 50 meters the bird sound. This was Implemented in a very lightweight way with a kd-tree in python, easily allowing point clouds of 20 million plus in around a second of computational time.

	Mesh	Biome	Appendix	Console	Help	
			Grid Debug			
Мар	WL01				~	
Shor	tcuts					
Optin	nisation Settin	ngs Audio	Settings Check	kout Check in	n	
Actor Grid						
Upda	te Scatter	Remove Sca	tter Create Exc	clusion By Selectio	n	

A grid had not only a name but could be assigned Niagara systems as well as AK Events (this would accept any type of assets. Supported assets are highlighted green). Every of those Objects could be assigned numerous parameters. In this example here, the TreeSize and the TreeType. In the audio settings, it is defined that a tree of 10m would be considered 'Medium' and that would be passed to the AK Event to choose the correct sound. The distribution of those objects is determined by the chosen setup, as shown in the bottom with Map, Biome, Subbiome etc. 'To Point cloud' would convert the current setup into a debug point cloud and visualize it. In the end those components would have a secondary grid that would define the bounds of the actor and would automatically add the components. For example every 10 meters there's a bird sitting on a tree. If the actor has a grid size of 100m, then 10 x 10 niagara components would then be added to it.

Component Grid					
Name New Name Grid Size 0	Add Grid	1			
AUD_Foliage_Generic - 1500 [1 Objects - 2 Biomes]	To Point C	Cloud 🔟			
Add Biome Clear Biomes	Add Object	Clear Objects			
 Objects 					
Object	ageRustle 🗸	Ū			
 Parameter 					
+ Clear					
AK Event Parameter TreeSize 🗸	Ū				
AK Event Parameter TreeType 🗸	Ū				
▼ [Plains - Forest1 - TreesLarge]	Ŵ				
Map WL01 V Bio	ome Categories	+ Clear			
Biome Plains 🗸	Biome Category De	fault 🗸 🛛 –			
Subbiome Forest1 🗸					
Biome Layer 🛛 TreesLarge 🗸					

Debug Settings

In order to have a clear idea about the spacing, a debug view was implemented. This would draw debug shapes in the viewport to showcase what that seed and grid size would look like based on the provided meshes.

	Grid	Debug
Settings View Distance 30000		
Test Grid		
GridSize 15000 Iterations 20000	Seed 0	
Add Selection Add Empty	Preview	Clear

The preview result would look like this. The filtered instance would be highlighted in the viewport and could be adjusted to the user content.



Console

The console shows a colored frame based on the most critical type of error that occurred. Filtering and clearing allowed for changing the amount of visible entries, export would create a text file. When the tool would be closed or crashed, this log would be sent to the UE5 internal log.



Help The help button leads directly to the Confluence documentation.

Structure

The overall UI consists of more than 30 sub-widgets and more than 100 structs and data asset types to allow for a very much customisable experience. This tool is 60% Blueprint, 30% C++ and 10% Python. It ended up having more than 20 pages of documentation and allowed for 15-20 people to create an open world simultaneously without much training time. Overall time to create including lots of feedback was around 3 months.

There's an incredible amount of things that were not mentioned due to the sheer size of the overall tool. If you have any more questions, I'ml be happy to answer!