PROCEDURAL GENERATION

Unreal engine Procedural Generation Plugin

Introduction:

Procedural generation is a plugin Consists of multiple Tools to develop A Hybrid Generation System in Unreal Engine

The Goal Is to build a system where the generation System will give you flexibility to Generate Procedurally as well as give you have manual control to manually.

For Example, the base Grid Can be done procedurally either organic or Tile Based And then the roads will also get generated procedurally, vegetation System will spawn vegetations around procedurally around and the construct system will be manual so that you can have manually control over construction of the buildings from Modular Parts.

Key Features:

- **Performance Efficiency**: it uses Instance static mesh component (ISMC) for each instance.
- **Data Table Driven:** All the meshes can be imported with a single Data table and placement constraints can be defined in the data table.
- **Modular:** the generation system follows Modular structure for easy modification
- **Debug Runtime Editor UI :** you can Use the runtime debugger UI for debugging

Manual Building System:

The Manual Generation/Building System is aiming to provide features to be able to manually construct building or another structures.

Key Features:

- A specific implementation for Top-Down View
- Top Down Controller has the Ready to use Input Setup Configure
- Configured with new Enhanced Input System
- Mouse Tracing to Grid
- Panning View, Zooming, And other Top-down Features
- Item Drag and Drop From UI
- Item Spawning TO Grid and Snapping to Other Objects
- UI Written In Completely C++ (Slate +UMG)

This is a work in progress Project!!!!

High Level Overview

The Algorithm Used to generate the base level follows wave function collapse algorithm.

Main Classes

I. The generation starts by making a grid of tiles.
II. Each Tiles is Represented By this class UTile : public UObject.
III. We Represent Each Spawn Able Mesh by This Class UTileMesh : public UObject.
IV. The Actor Handling the Generation is called ABaseProceduralActor : public AActor
V. The Tile Grid generation is managed by UCoreGenerator : public UObject
VI. The Struct Used for Data Table - struct FTileMeshData : public FTableRowBase

UTile

This class holds data about tile.

1. The ID Of tile
2. 2D Position in the Grid
3. World Location
4. The Available Meshes The algorithm can Choose from (this gets updated Depending Surrounded Tiles TileMesh)

UTileMesh

This Class hold data of each Spawn able Mesh

1. The Actual Static Mesh
2. Is the Mesh Tile Able
3. The Tag Representing the Tile (Each Mesh Should have its own Tag, I am using FGameplayTag for this
4. 4 Tag Container (FGameplayTagContainer) , this container stores Tags of supported TileMesh which can be place around this mesh so for 4 sides we have 4 tag container each container can contain multiple tag
5. And a UInstancedStaticMeshComponent , This component can have multiple instance of the same mesh spawned in the word with single draw call

FTileMeshData

This struct exposed to blueprint and it Used to make the data Table

It exposes necessary variable to data table

Mainly it Contains variable from the UTileMesh .

Algorithm Implementation

The implementation of this algorithm consists of multiple steps.

Lets start with the **Begin Play** Function

ABaseProceduralActor::BeginPlay()

1. This is the beginning of the function we starts with initializing the class **ABaseProceduralActor**
   we call the **Init()** function , in this we import the Data Table using the **StaticLoadObject** function
   once we have the data for each row of the data we create a corresponding UTileMesh and pass that data to the tilemesh
   TileMesh->Init(this,tilemeshdata) and store all of them in the **TArray<UTileMesh*> TotalTileMesh**

2. Next After initializing the class we call RunGenerator() , this class creates a instance of provided generator
   The base generator class is UCoreGenerator , this class handles the base grid generation logic (you can make custom, generator by inheriting from UCoreGenerator and overriding the base implementations)
   After creating instance of we call the **Generator->Init(ControllerWidget,FloorMesh,Map_Height,Map_Width**
   then run the generator it takes a array of utile* Reference as input .

3. Now As we have the Array of tiles as well as the tile mesh We call the **WaveFunctionCollapse()** Function

WaveFunctionCollapse()

1. In the WaveFunctionCollapse() function first We Copy all the Tile to Another array calls **RemainingTiles** which stores them until they get collapsed (means Until it gets a specific mesh allocation) ,

2. For the First Tile we generate a random seed and from that seed we choose a random tile from the Remaining tiles array ;

3. Now We Call the AddInstanceMesh(FirstRandomTile) BY Providing he selected tile it calls the RandomMeshFromAvailableMesh(SelectedTile); this function returns a random tilemesh from the **AllAvailableMeshToChooseFrom** array of the tile , after that it takes the **instanceComponent** mesh from the TileMesh and add a instance at the location also removes the tile from **RemainingTiles**

   After adding a instance in the tile it Updates its surrounding By calling **UpdateSurroundingMesh(UTile* SelectedTile, TArray<UTile*>& TotalTile)** , this function calls 4 other function they updated 4 surrounding tiles left , Right , Up ,Down (these are base on the 2D Grid)
4. So now as it Added a mesh and updated its surrounded tile it Choose the next tile base on the lowest entropy for that it calls **ReturnTileWithLowestEntropy(RemainingTiles)**
5. Now until the **RemainingTiles gets** empty it checks for tile With Lowest Entropy And Repeats Step-3 For each of them

6. **Your Map Is Ready !!!!!**

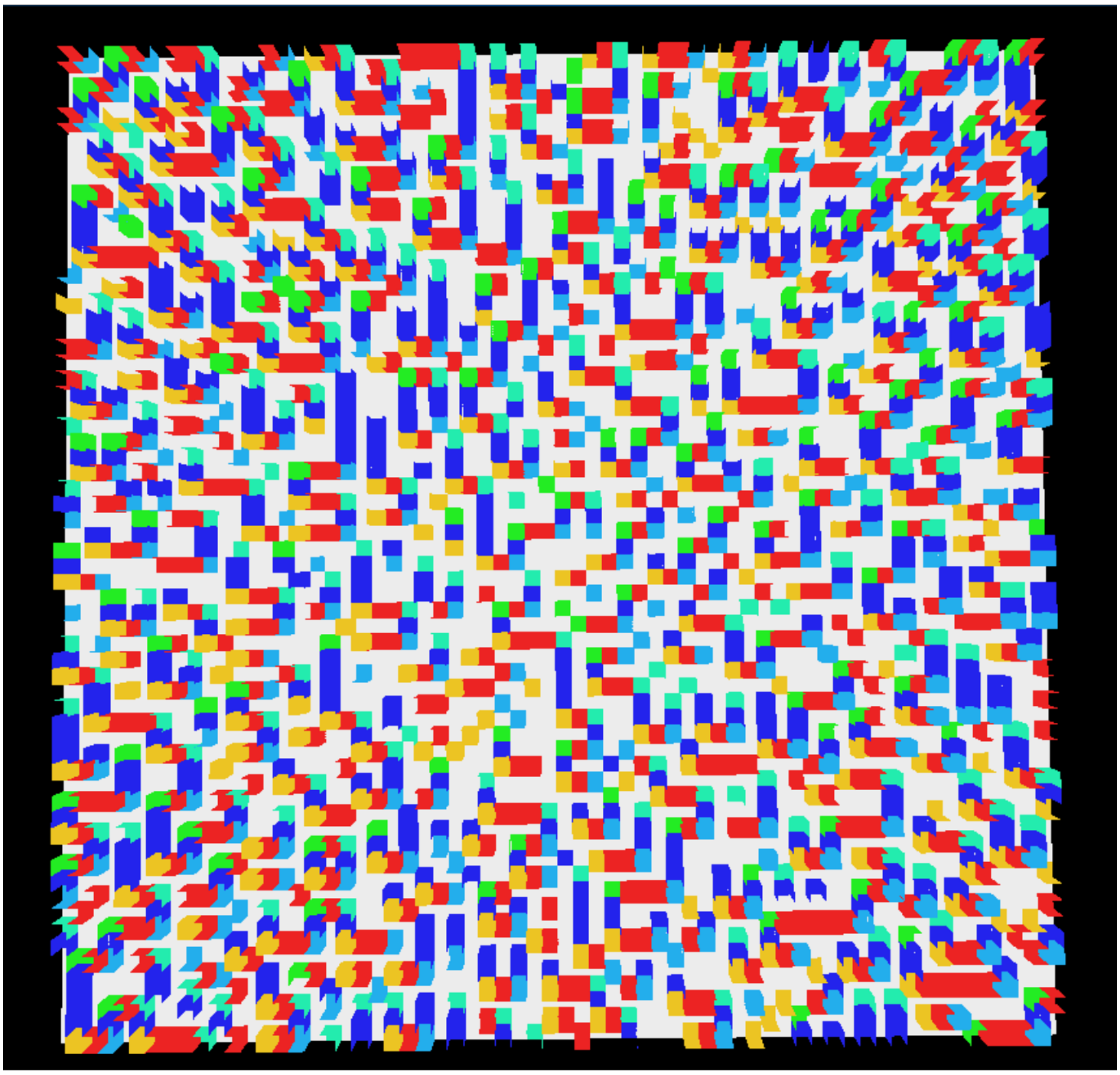
Cleaning UP

At the end play function we don some clean up

We iterate through the Array And Call the ConditionalBeginDestroy() and remove from the array

You Can Watch this video to learn more about the Wave Function Collapse algorithm: https://youtu.be/2SuvO4Gi7uY

Result Without Custom Frequency :



Raw Generation Without Frequency Control 4096 Tiles

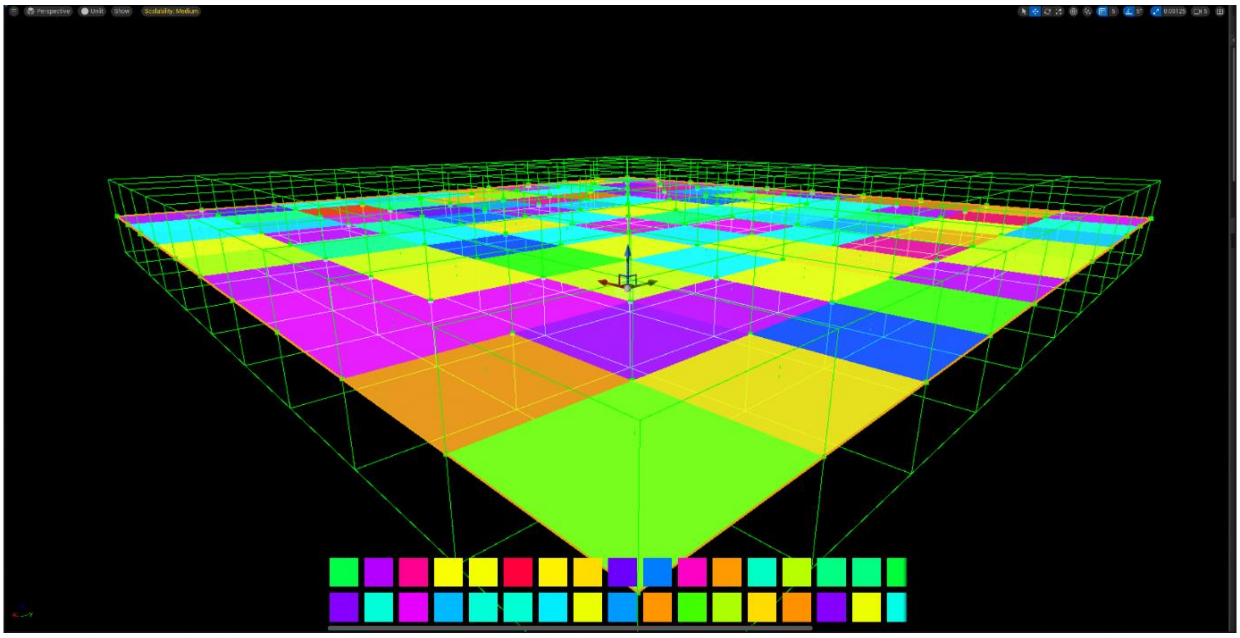
Top-Down Implementation

Program Mesh Grid:

This Actor uses the UProceduralMeshComponent to generate the grid Can be modified at runtime

Features :

1. Runtime Modification
2. Can be Animated.
3. Dynamic Resizing
4. Can be used to Make Organic Landscape Base


Perspective
Unlit
Show
Scalability: Medium