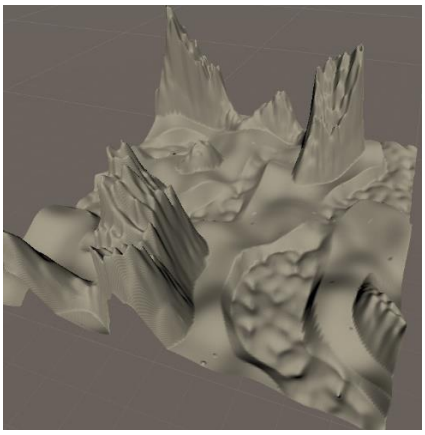
Procedural Content Generation with Perlin Noise



In this assignment you will be utilizing Perlin Noise generation and a hierarchical set of generation rules to create complex heightmap terrain.

Perlin Noise is a type of gradient noise that is useful as a starting point for procedural content generation (PCG) of textures and terrain. Perlin Noise can be scaled to reflect random details at different frequencies. Noise can be processed and layered to create different effects.

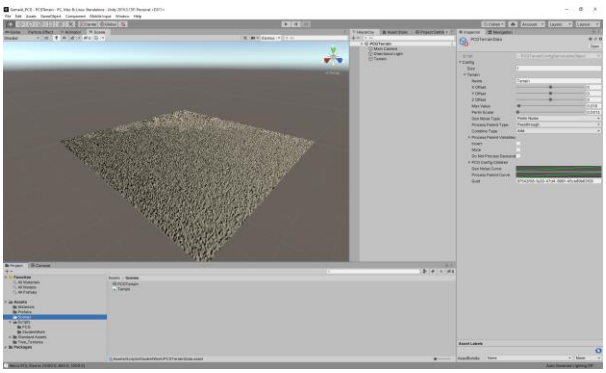
For this assignment, you will complete the implementation of a simple recursive PCG rules engine. These development tasks include generating noise from an existing Perlin Noise implementation and completing configurable preprocessing and combination/mixing steps.

Additionally, you will use your completed PCG generator to design a terrain with three visually distinct areas (or biomes). This will involve segmenting your terrain according to boundaries generated by Perlin Noise. For instance, you might make mountains in one biome, sand dunes in a second, and a canyon in a third. This configuration will be saved in a Unity ScriptableObject.

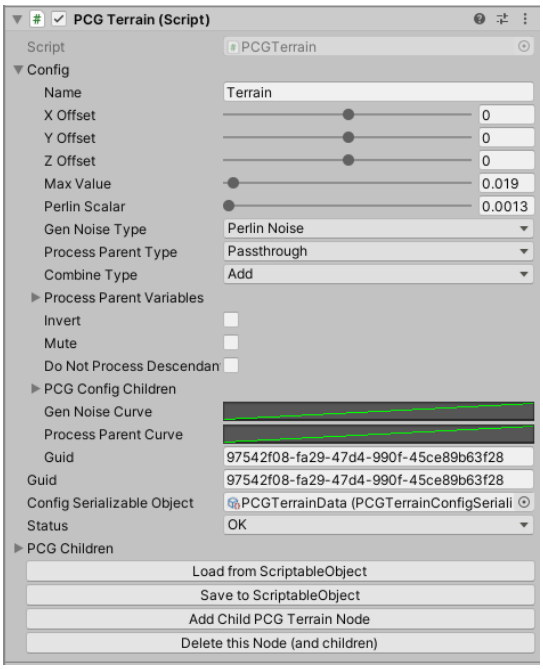
You will submit your PCGTerrain.cs and your designed terrain PCGTerrainData.asset (ScriptableObject) and four (4) screenshots that best demonstrate what you developed. There are more details in the Submission Section.

Development Resources

A Unity Project (GameAI_PCG) is provided as a starting point. Open the project in Unity and then open scene, PCGTerrain. You will not need to “Play” the game at any point. All the work can be done in the Editor.



You will notice that there is a terrain with lots of bumps in the Scene view. You can select it in the Hierarchy view (named “Terrain”). In the Inspector view you can scroll down to the PCGTerrain component. Here you have access to several configurable actions. They will not be very useful until you implement the unfinished parts of PCGTerrain.cs.



But you can move the “Max Value” slider to see the effects of scaling the noise output.

Code Changes

Open PCGTerrain.cs. You will need to complete the following steps:

1.) Implement: `FindPerlinNoiseAtPos()`
2.) Complete the parent processing step in `ApplyPerlinNoiseContribution()`
3.) Complete the combination step in `ApplyPerlinNoiseContribution()`

`ApplyPerlinNoiseContribution()` is a recursive function. It allows for a variety of PCG terrain effects to be realized. It is executed according to a rules graph affecting generation and combination at each level. It generates a value for the current level in the graph, then evaluates child nodes to see if the children modify the current node's value. If there is more than one child, the sum of children’s values replaces

the current node's value. A single child replaces the current node's value. If there are no children, then the current node keeps its generated value. Next the value of the current node's parent (or a default value if the current node is root) is processed. This could involve keeping it as is or transforming the parent's value in some way. The processed parent value is then combined with the current node's generated value (which itself may have been modified by its children). Each of the parent processing and data combination steps have comments that start with “TODO” and a description of what is needed.

Building a PCG Graph

Once you have the code changes implemented, you can build your graph. You will use your completed PCG generator to design a terrain with three visually distinct areas (or biomes). This will involve segmenting your terrain according to boundaries generated by Perlin Noise. For instance, you might make mountains in one biome, sand dunes in a second, and a canyon in a third. This configuration will be saved in a Unity ScriptableObject.

This is probably the most difficult part of the assignment as you will be declaratively defining rules/constraints on the noise generation to develop a compelling terrain scene. There are several workable strategies, but some methods are easier than others to organize things. Be sure to check out the related lecture video for tutorials.

The general steps involve configuring a node, then adding new nodes (via “Add Child PCG Terrain Node”) and then continuing the process of configuring and adding more nodes. Lastly, you need to save/serialize out to disk using the Save button. See below for more details.

PCG Configuration Details

The Name field is very helpful for keeping track of what layers do what. Be sure to change the name in the PCGTerrain Component and not at the top of the GameObject (PCGTerrain overrides the GameObject name).

The X/Y/Z Offset allow exploration of the noise space. This is useful for demonstrating how well your rules work if the terrain tiled out indefinitely. You can also use these to find the best screenshot.

The MaxValue is used to limit the amplitude of generated noise.

The PerlinScalar allows “zooming” in/out on the noise (change the frequency of gradient changes).

The GenNoiseType allows for PerlinNoise, PerlinNoiseWithMappingCurve, and None. The mapping curve involves processing the generated noise with GenNoiseCurve (a manually configured function). If None is selected then no noise is generated and a default value of 1.0 is used (scaled by MaxValue).

The ProcessParentType can be passed along unmodified (Passthrough), Inverted, processed/remapped with ProcessParentCurve, or Bandpassed (with cross fades). The latter is especially useful for segmenting noise for the purpose of creating regions/biomes. Note that the ProcessParentVariables is needed to define the bandpass region (4 values; the two outer values [index 0 and 3] define the start of the fade region and the two inner values [index 1 and 2] define the band). Anything outside the bandpass is mapped to 0.0. Anything inside is mapped to 1.0 (or faded from 0.0 to 1.0).

The CombineType selects how the current node’s generated value is combined with the parent. Refer to the code for a description of each operation.

Invert can be used to flip the final output value around (1.0 – outputVal).

Mute can be used to temporarily disable noise generation (value is 0.0).

DoNotProcessDescendant can be used to temporarily disable evaluation of children nodes.

Status is readonly and shows if an error occurred in an inner loop. This is the best way to show errors because print statements can slow Unity down too much.

The buttons at the bottom of the PCGTerrain component are also useful. You will need these for building your graph.

Load from ScriptableObject: Be cautious with this button! It will delete all children of the root node and reload/rebuild from the ScriptableObject. The load button is NOT smart. It is NOT aware of unsaved changes. The root and children will then match the ScriptableObject. Only available at the root of your graph.

Save to ScriptableObject: You must manually click this button to serialize your graph to disk (will be PCGTerrainData.asset with the default assignment). You should also save your scene from the Unity File Menu after saving the graph. Only available at the root of your graph.

Add Child PCG Terrain Node: This adds a child node that has PCGTerrain configuration to the current node. This button is available at every node of your graph

Delete this Node (and children): Be cautious with this button! There is NO confirmation! This button deletes your selected node and children. This button is available at every node of your graph.

Other Considerations

Feel free to share techniques for building rules graphs on Piazza with classmates. Also, you can extend the code in PCGTerrain.cs as you please (but do not delete options, even if you don't use them). Be aware that if you add to the PCGTerrainConfig properties, you will need to also add the new property to DeepCopy() so that serialization to the ScriptableObject works correctly.

If you are having performance problems, you can reduce your terrain resolution. Select the Terrain component settings (gear) and scroll to "Heightmap Resolution". Change to something smaller than currently. You might go higher for screenshots.

Submission

Submit your PCGTerrain.cs, your PCGTerrain.asset (be sure you click "Save to ScriptableObject" in the PCGTerrain Inspector Component View and also save your scene from the Unity File menu), and four (4) screenshots of your final terrain design. Place all in a ZIP/7ZIP file named LASTNAME_FIRSTNAME.[zip/7z].