Terrain Engine 2D

A 2D Block Engine for Unity

Out now on the Unity Asset Store

BUY NOW!

Terrain Data

This page explains what Terrain Data is and how to create a custom TerrainData script for procedurally generating the block world.

Table of Contents

General

The Terrain Data refers to all of the data used to procedurally generate the block terrain. This includes the Block Type, Bitmask, Variation, Render Block, and Fluid Data. In Terrain Engine 2D, this data is stored in 2D arrays where every single grid unit/tile position has this information stored. All of this data is handled by the block engine, there are various levels of abstraction making it easy for the developer to make changes to the Terrain Data without worrying about all the extra complexity.

- Block Type The type of block
- Bitmask The bitmask of the block for rendering
- Variation The variation of the block texture
- Render Blocks Whether this block should be rendered
- Fluid Data FluidBlock properties including Weight (amount of liquid) and Stable (whether the liquid is flowing)

In Terrain Engine 2D, the Terrain Data is initially generated on Start, and then it can be dynamically modified at runtime using the included Tools. The Terrain Data is generated using a custom TerrainData script created by the developer.

Framework

Terrain Engine 2D contains a framework for procedurally generating all the Terrain Data when the World is initialized. This framework will generate the bitmasking values, block variations, and set the Render Blocks, but it requires a custom Terrain Data script to set the block types, and Fluid Data. This is so the developer has full control over how all the blocks are placed in the terrain. This script must extend the base TerrainData script which includes many helper functions so you do not have to access any variables directly.

TerrainData Script

The TerrainData script is a base Monobehaviour script which is meant to act as an extension class for creating a custom Terrain Data script. This class makes it very easy to setup the block data for the World. This script can be found in the asset package under: *TerrainEngine2D/Assets/Scripts/Terrain/TerrainData.cs*.

Helper Functions

- GenerateData This is where all the block data is set
- PerlinNoise Gives the perlin noise value at a specific coordinate
- AddBlock Sets the block type of a specified layer to a specific block
- DoAddBlock Determines if a block should be placed based on a probability factor
- RemoveBlock Removes a block from a specified layer
- RemoveAllBlocks Removes blocks from all layers at a specific coordinate
- IsBlockAt Checks if there is a block at a specific coordinate or area
- SetFluidType Set a Fluid Block to either Empty, Solid or Full
- AddFluid Adds liquid to a Fluid Block
- RemoveFluid Removes all liquid from a FLuid Block
- GeneratePool Generates a pool of fluid
- ClearFluid Removes all fluid from blocks below a threshold (floodfills the area, stops at terrain blocks)

Creating a custom Terrain Data script

Script Setup

1. Start by creating a new c# script or using the TerrainDataTemplate script as a starting point. If you choose to use the template some steps are already completed for you.
2. Inherit the TerrainEngine2D namespace by adding: `using TerrainEngine2D;` at the top of the script.
3. Extend the TerrainData script in the class declaration: `public class CustomTerrainData : TerrainEngine2D.TerrainData`
4. Override the GenerateData function and call the base function:

```
public override void GenerateData()
{
    base.GenerateData();
}
```

5. (Optional but recommended) Add a Layers enum containing the name of all your layers. Example: `private enum Layers { Background, Trees, Decoration, Main, Ore, Foreground}`
6. (Optional but recommended) Add enums for all your Layers containing the name of all of its Block Types. Example: `private enum Background { Dirt, Stone, Wood }`
7. Now you can start adding to the GenerateData function and begin generating your block data (next section).
8. When you are finished with your script, save and add your script to the Terrain Data Script Field in the World custom inspector.

World (Script)
Main Properties
Block Setup
Objects
Toggle OSD:
OSD Update Rate:
0.05
Terrain Data Script:
World (TerrainDataExample)
Toggle Cursor:

Terrain Engine 2D TerrainData Script Field

Generating Block Data

As the developer you have full control over generating the block data for your terrain. This means there is no proper or correct way to code this section, it is up to you to figure out your own cool and unique ways of generating terrain. You can use any kind of noise you wish, algorithms, libraries, etc. That being said everything you need to know to get started can be found below and the included TerrainDataExample script gives you further detail and examples.

Pass

Block data is generated in Passes, inside each Pass is where you can modify the block data. A Pass is where you loop through all of the blocks in the world to modify the block data. You can have as many passes as you wish inside the GenerateData function. A pass looks something like this:

```
//Pass
for (int x = 0; x < world.WorldWidth; x++)
{
    for (int y = 0; y < world.WorldHeight; y++)
    {
        //-----Set block data here-----


        //----------
    }
}
```

Modifying Block Data

Block data modification refers to adding and removing blocks and/or fluid. Block data is modified inside a Pass which loops through every block in the World. This means anything you do inside the Pass is applied to every single block in the World. Generally we do not want to set every block to the same type, so to avoid this we use conditional statements. These conditional statements make use of noise functions, random functions, and other values or functions in order to decide whether to add or remove a block.

To add or remove blocks there are two helper functions: `AddBlock(int x, int y, byte layer, byte blockType, float probability = 100f) RemoveBlock(int x, int y, byte layer)` When calling these functions you need to specify which Layer and if you are adding a block which Block Type is being set. The functions take this data as a byte referring to the index of the element in the Layers/Block list of the World custom inspector. It is recommended to use Enums in your script to keep track of all your Layers and Block Types.

Example - Removing clumps of blocks to create caves

```
AddBlock(x, y, (byte)Layers.Background,
(byte)Background.Dirt);
AddBlock(x, y, (byte)Layers.Main, (byte)Main.Dirt);

if (PerlinNoise(x, y, 10, 10, 1) > 5)
{
    RemoveBlock(x, y, (byte)Layers.Main);
}
```

Terrain Engine 2D Terrain Caves

Perlin Noise

Perlin Noise is a type of gradient noise generated through the Perlin Noise algorithm. It is a popular algorithm used in many different applications, one of those being procedural generation of graphics. Unity contains a built in function 'Mathf.PerlinNoise' which can be used to get values of Perlin Noise from a given x and y float value. The TerrainDataExample script shows how you can use Perlin Noise to procedurally generate data. There is also a helper function called PerlinNoise which makes it easier to manipulate and generate Perlin Noise values.

Perlin Noise can be used and manipulated to create natural pseudo-random block terrain. Some things Perlin Noise can be used for includes; calculating terrain height, generating clumps of blocks, and creating caves.

The PerlinNoise helper function contains a number of parameters for manipulating the output Perlin Noise value.

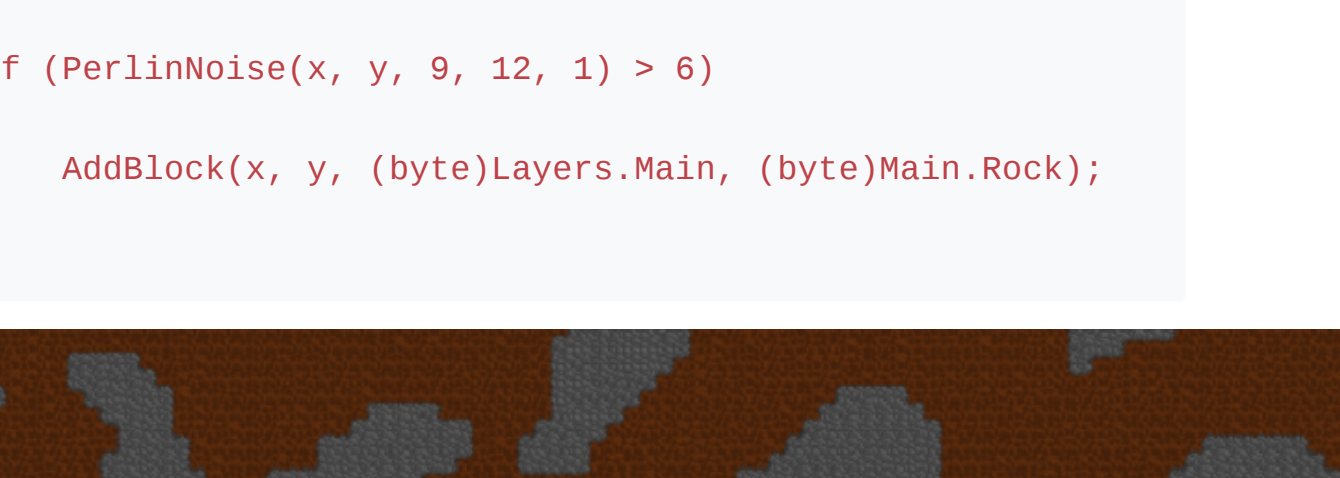
Perlin Noise Parameters `PerlinNoise(int x, int y, float scale, float mag, float exp)`

- x The x coordinate of the block
- y The y coordinate of the block
- scale Scaling; lower values gives more varying height output
- mag Magnitdue; higher values produce a larger height output
- exp Exponential; greatly increases height output

Example - Adding clumps of rock

```
AddBlock(x, y, (byte)Layers.Background,
(byte)Background.Dirt);
AddBlock(x, y, (byte)Layers.Main, (byte)Main.Dirt);

if (PerlinNoise(x, y, 9, 12, 1) > 6)
{
    AddBlock(x, y, (byte)Layers.Main, (byte)Main.Rock);
}
```



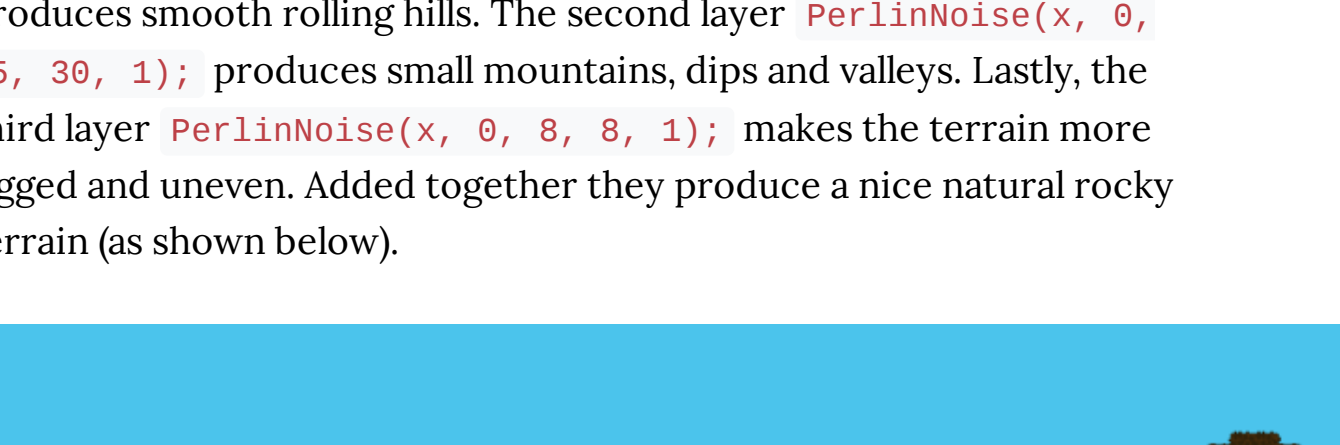
Terrain Engine 2D Terrain Rocks

Height Variables

Height Variables are variables used to set the height of the terrain. They are initialized inside the first loop of the Pass, so that for every x value there is one consistent height value.

```
//Pass
for (int x = 0; x < world.WorldWidth; x++)
{
    //-----Set Height Variables here-----
    int groundLevel = PerlinNoise(x, 0, 75, 20, 1);
    groundLevel += PerlinNoise(x, 0, 45, 30, 1);
    groundLevel += PerlinNoise(x, 0, 8, 8, 1);
    groundLevel += 60;
    //----------
    for (int y = 0; y < world.WorldHeight; y++)
    {
      if (y <= groundLevel)
      {
        //Any blocks added here will be at or below the
groundLevel
      }
    }
}
```

In this example 'groundLevel' is the height variable. It has various Perlin Noise values added to it in order to produce the resulting height value for the current x coordinate. Each additional Perlin Noise value adds another layer of noise to the height variable. In this case the first layer of Perlin Noise `PerlinNoise(x, 0, 75, 20, 1);` produces smooth rolling hills. The second layer `PerlinNoise(x, 0, 45, 30, 1);` produces small mountains, dips and valleys. Lastly, the third layer `PerlinNoise(x, 0, 8, 8, 1);` makes the terrain more jagged and uneven. Added together they produce a nice natural rocky terrain (as shown below).



Terrain Engine 2D Height Variable Example

Random

When generating terrain it is often necessary add a level of randomness to your blocks. In Terrain Engine 2D this can be done a number of ways. Unity contains the class 'UnityEngine.Random' which can be used for generating random values. Terrain Engine 2D has two functions which make use of this Random class: AddBlock and DoAddBlock. AddBlock can take an optional percent probability to determine whether the block should be added or not. DoAddBlock takes a percent probability and returns a boolean value representing the evaluation of the odds. Below are some examples making use of these functions.

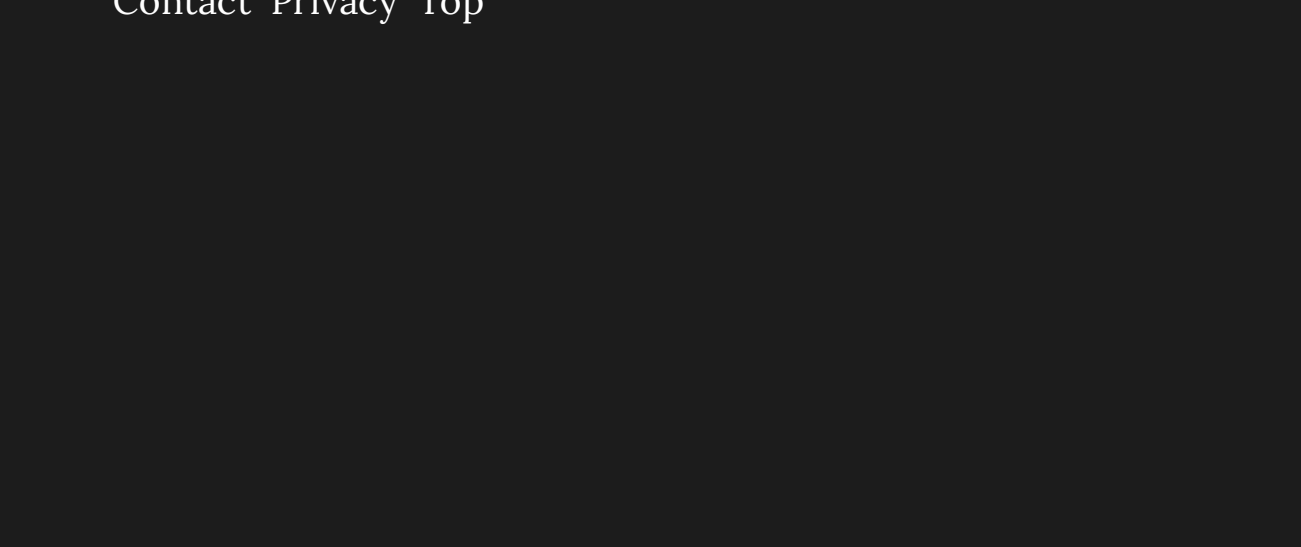
Example - Randomly adding gems

```
AddBlock(x, y, (byte)Layers.Main, (byte)Main.Rock);
//Add Gems with a probability of 1%
AddBlock(x, y, (byte)Layers.Ore, (byte)Ore.Gems, 1);
```

Terrain Engine 2D Random Gems Example

Example - Random Table Sets

```
if(y < 50)
  AddBlock(x, y, (byte)Layers.Main, (byte)Main.Wood);
//Add Table set with 10% probability if the space is free
if (y == 50 && DoAddBlock(10) && !IsBlockAt(x, y,
(byte)Layers.Decoration, 5, 2))
{
  AddBlock(x, y, (byte)Layers.Decoration,
(byte)Decoration.LeftChair);
  AddBlock(x + 1, y, (byte)Layers.Decoration,
(byte)Decoration.Table);
  AddBlock(x + 4, y, (byte)Layers.Decoration,
(byte)Decoration.RightChair);
}
```



Terrain Engine 2D Random Table Sets Example