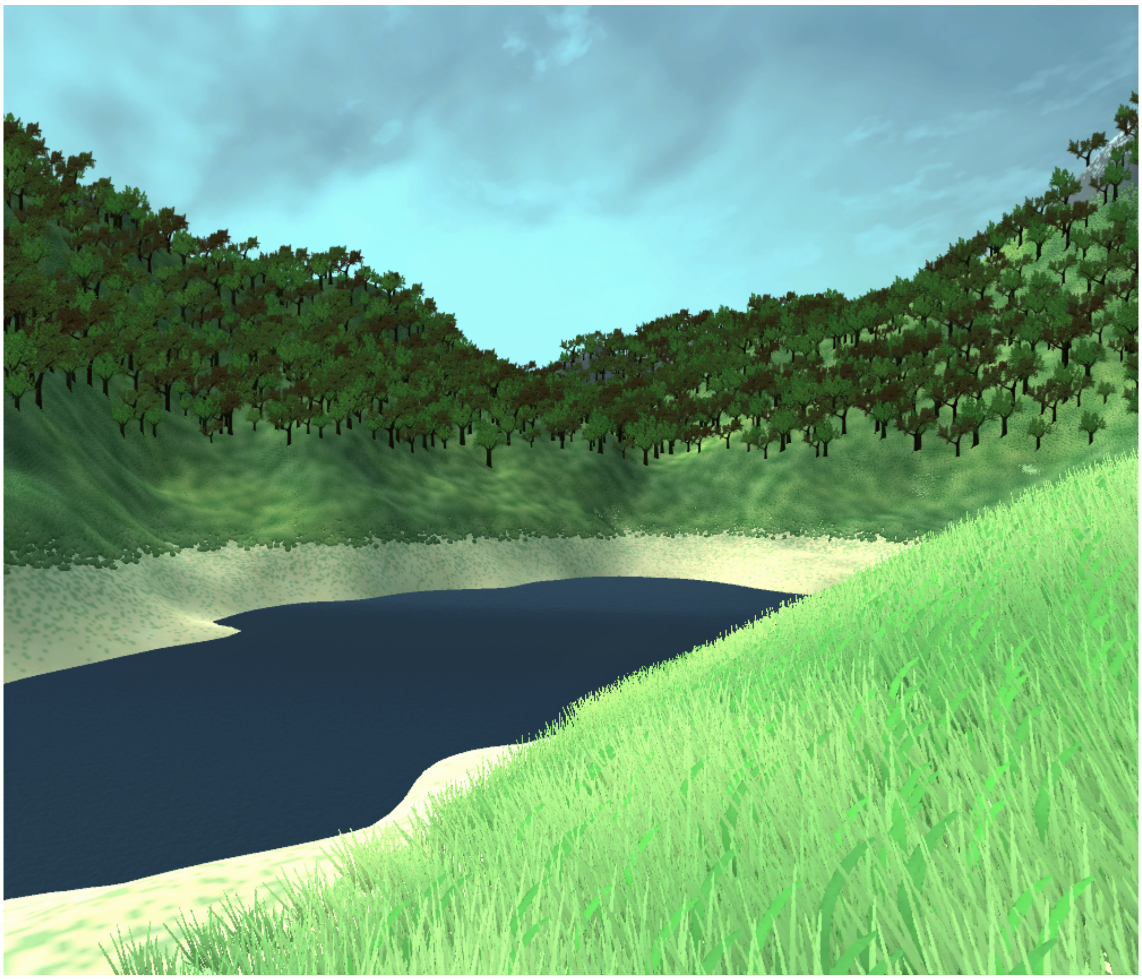
Procedural Landscape Generation with Unity



Susanna Brambilla (940755) - susanna.brambilla@studenti.unimi.it

Artificial Intelligence for Video Games a.y. 2020/2021

Table of Contents

1 Introduction

This project aims to develop a tool for Unity for generating procedural semi-realistic landscapes with textures, vegetation and water.

Procedural generation using terrains can be useful to assist the designer in creating game environments in an automated way, meaning algorithmically with minimum human input. The need for automatic terrain generation stems from the desire of providing the player new content without a large investment of resources and time, hence the goal is optimize the development process.

For achieving this goal, three different windows have been created for the Unity Editor: the Terrain Creation window, the Texture Window and the Environment Window.

The Terrain Creation Window allows to generate the landscape basic structure modifying the Terrain mesh and makes use of two different methods, the *Perlin Noise* and the *Diamond-Square* algorithm.

The Texture Window applies texture images using the terrain alphamap according to the different heights and slope data. In addition, it allows to change the color of the textures to generate new images, when needed.

The Environment Window makes it possible to give some level of completeness and realism to the terrain adding trees, details, such as grass, and finally water.

In this project, PCG is applied in offline mode: in this case contents are generated before the game is shipped. The content is semi-automatically created: a human support is needed to the algorithm's work. Each window has, indeed, a set of parameters that can be manually adjusted to change the configuration of the landscape in a controllable and intuitive way, but a certain amount of randomness is added to obtain unpredictability and novelty.

Once the landscape is ready to be used, an agent can be spawn on its surface. The agent has a wander behavior which helps to explore the terrain in order to understand its validity and navigability. For pathfinding and movement the Unity's Navigation Meshes system is used.

2 Terrain

The aim is to offer a simple tool with tab and sliders which allows the designer to generate landscapes with procedural methods, changing parameters for each specific aspect. Three different windows, working on different aspects, were created for this purpose. The windows can be found opening *Window* in the Editor.

2.1 Terrain Creation Window

This window, shown in Fig. 1, allows to create the structure of the landscape through the algorithmic manipulation of the Terrain Game Object's mesh.

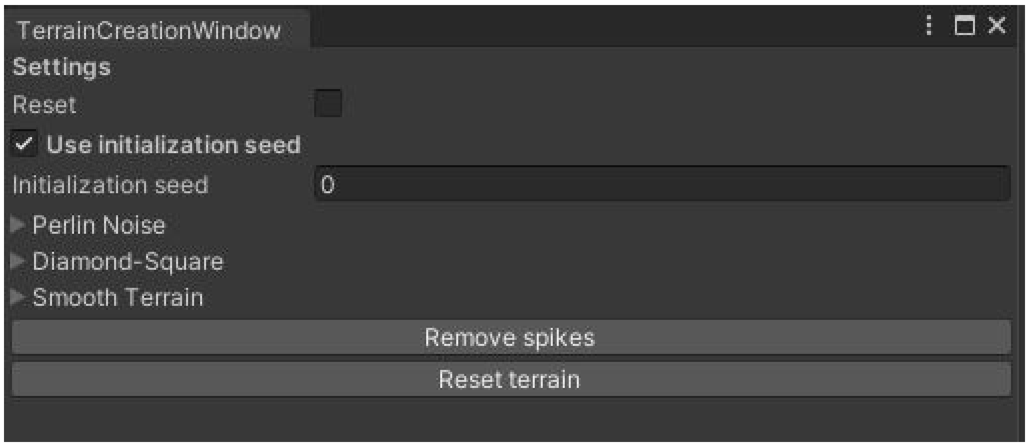

TerrainCreationWindow
Settings
Reset
✓ Use initialization seed
Initialization seed 0
Perlin Noise
Diamond-Square
Smooth Terrain
Remove spikes
Reset terrain

Figure 1: Terrain Creation Window.

Window components:

- *Reset*: if this toggle is enabled, the heightmap of the terrain is reset, setting the height of each point to 0, every time a change in the structure happens, before applying the change;
- *Use initialization seed*: if this toggle is enabled, allows to insert a seed for the terrain initialization, making it possible to obtain the same Terrain more than once using the same values with the same seed;
- *Perlin Noise*: this foldout allows to apply Perlin Noise to the terrain to generate the heightmap;
- *Diamond-Square*: this foldout allows to apply the Diamond-Square algorithm to the terrain to generate the heightmap;
- *Smooth Terrain*: this foldout applies smoothness to the Terrain;
- *Remove spikes*: this button makes the terrain rounded and flatter removing a quantity of height (the current height divided by 1.2);
- *Reset terrain*: this button makes the Terrain flat setting all heights to 0.

In the next sections, a more detailed explanation of what each of the three foldout contains will be given.

2.1.1 Perlin Noise

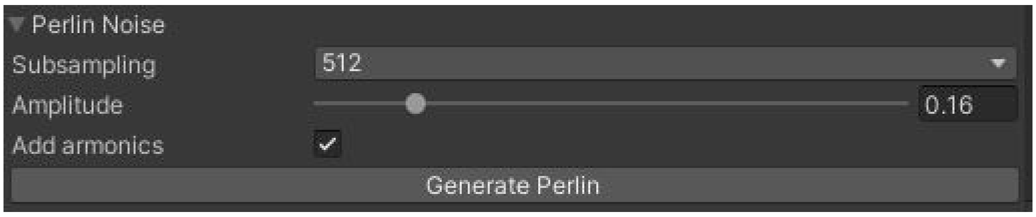
Perlin Noise is a procedural noise invented by Ken Perlin. It is also defined as gradient noise: the steepness and the direction of the slopes are randomly generated and only then the heights of the heightmap are inferred.

The Perlin Noise implementation has been provided by professor Dario Maggiorini and adapted for this project, where it is mainly used to explore the differences in results between this method and the Diamond-Square one.

A lattice is a grid on the terrain of equally spaced points. A random gradient vector is first generated at each lattice point, and each lattice point's height is set to 0. For every non-lattice point it is calculated the contribution to its height given by each of the four neighbouring lattice points. Its relative position inside the square is calculated and finally interpolated, in this context using bicubic interpolation.

To generate fractal terrains with Perlin Noise, thus making them more natural, amplitude and armonics are introduced. These parameters allow to run Perlin Noise multiple times reducing scale each step, and then add them togheter.

The fractal Perlin Noise foldout (Fig. 2) allows to apply fractal Perlin Noise using the *Generate Perlin* button.



Perlin Noise

Subsampling	512
Amplitude	0.16
Add armonics	✓

Generate Perlin

Figure 2: Perlin Noise foldout.

Parameters:

- *Subsampling*: represents the lattice subdivision and three parameters can be selected: [(heightmap resolution - 1) divided by 8], [(heightmap resolution - 1) divided by 4] or [(heightmap resolution - 1) divided by 2];
- *Amplitude*: measures the distance between the point position and its displacement, reduced each step;
- *Add armonics*: allows you to apply Perlin Noise more than once at multiple scales and then adds them togheter, as shown in Eq. 1.

$$noise(a) + \frac{1}{2}noise(\frac{a}{2}) + \frac{1}{4}noise(\frac{a}{4}) + \frac{1}{8}noise(\frac{a}{8}) \quad with\ a = amplitude \tag{1}$$

2.1.2 Diamond-Square

The Diamond-Square algorithm is a method for fractal terrain generation. This algorithm need a terrain of width and length of size $2^N + 1$, and starts by setting the four corners of the heightmap using random values, then the algorithm proceeds repeating two steps (represented in Fig. 3):

- Diamond step: for each square, find the center of the square defined as the average of the four corners' values plus a random value;

- Square step: for each diamond, find the center of the diamond defined as the average of the four corners' values plus a random value.

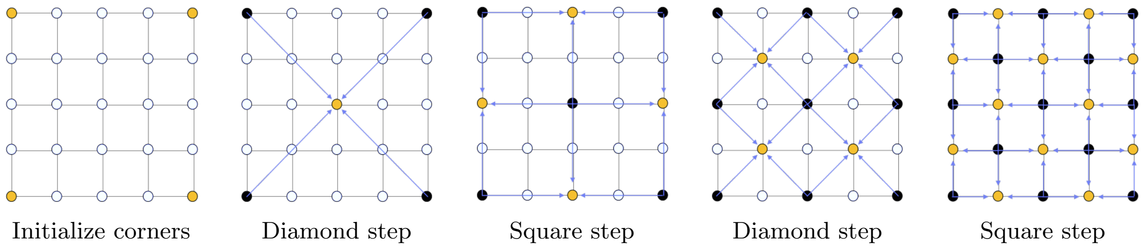


Figure 3: Diamond-Square algorithm.

The magnitude of the random value is called roughness and it is reduced at each iteration. In the Diamond-Square algorithm the two steps are repeated until a maximum number of iteration have been reached, as implemented in this project, the two steps are repeated N times with $N = log_2(terrainresolution - 1)$.

During the square steps, points located on the edge of the heightmap take the average of only the three adjacent values.

The Diamond-Square foldout (Fig. 4) allows to apply Diamond-Square using the *Generate DS* button.

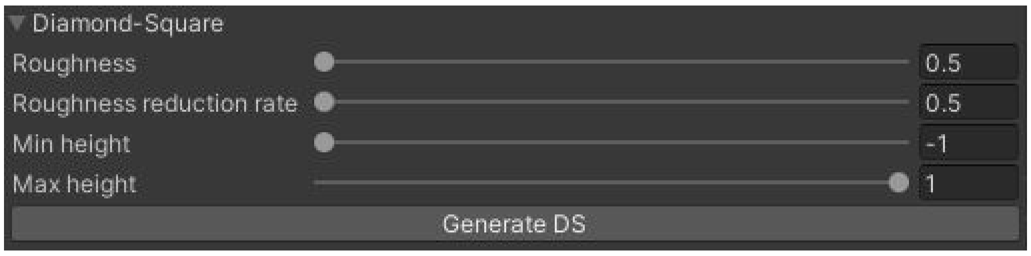

Diamond-Square
Roughness 0.5
Roughness reduction rate 0.5
Min height -1
Max height 1

Figure 4: Diamond-Square foldout.

The *Roughness* parameter is the value that reduces the magnitude of the random value using an exponentially negative function with the *Roughness reduction rate* parameter.

2.1.3 Smoothness

The smoothness function calculates for each point in the heightmap the average of its neighboring points plus the point itself and substitutes the point with this value.

The Smooth Terrain foldout (Fig. 5) allows to apply the smoothness function using the *Smooth* button.

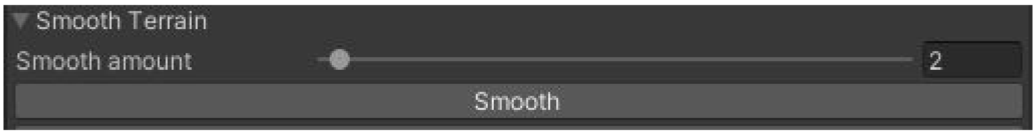

Smooth Terrain
Smooth amount 2

Figure 5: Smooth terrain foldout.

The *Smooth amount* parameter controls how many times smoothness will be applied.

2.2 Texture Window

The Texture Window, which is shown in figure Fig. 6 helps in the task of adding textures to the Terrain. You can assign different splatmap textures to the Terrain alphamap considering variations in heights and steepness, and blend them together.

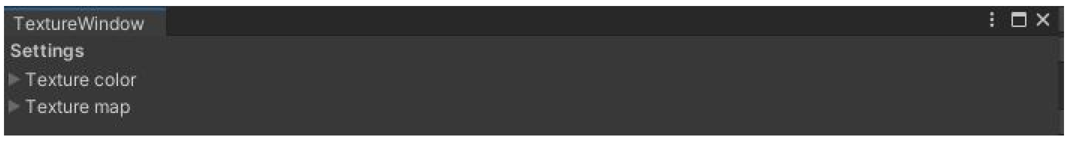


Figure 6: Texture Window.

Window components:

- *Texture color*: this foldout allows to take an existing texture image, change its color and save it as a new image;
- *Texture map*: this foldout applies existing texture images on the Terrain.

Now it will be given a detailed explanation of the content of these two foldout.

2.2.1 Texture Color

The Texture color foldout, in Fig. 7, gives you the ability to take a 2D texture image and change its tint. You can find some seamless images, useful for this purpose, in the *Seamless Textures* folder inside the *Assets*. All the images have been freely downloaded from Internet. There are different types of textures in order to match with different kind of environments, such as sandy, grassy, rocky, and snowy.

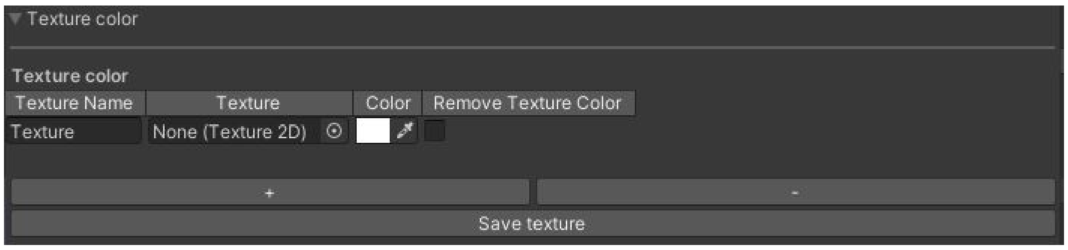


Figure 7: Texture Color foldout.

Let's now give an explanation of all the fields below the Texture color label:

- *Texture Name*: allows you to write a name for the texture image;
- *Texture*: you must select a 2D texture image from the *Seamless Textures* folder to use it as a base image;
- *Color*: select a tint to change the texture color;
- *Remove Texture Color*: if checked, you can remove this texture color line.

Explanation of the buttons:

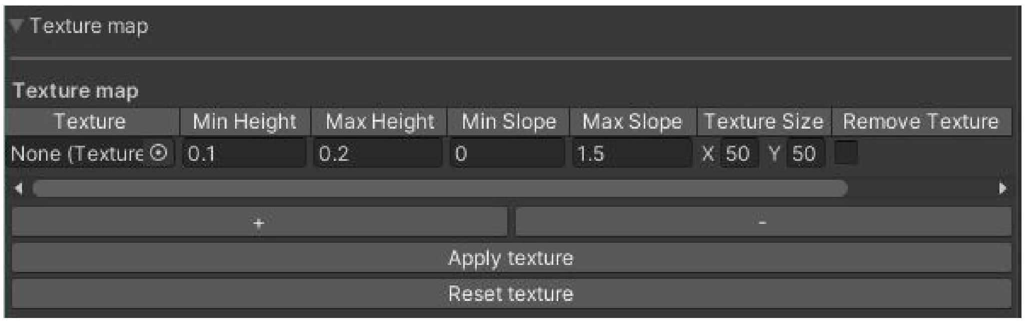
- (+): adds a new line to select more texture images at the same time;
- (−): removes all the lines toggled with Remove Texture Color;

- *Save texture*: saves all the new colored texture images;

All the colored texture images will be saved in the *Textures* folder inside the *Assets* folder. Before applying them on the terrain, you need to select them and toggle *Read/Write Enabled* in the *Inspector*.

2.2.2 Texture Map

This foldout, represented in Fig. 8, is needed to apply the textures on the Terrain alphamap. It gives you the possibility to choose the height at which each splatmap texture will be applied, based also on the terrain steepness and to change the tile size.



Texture	Min Height	Max Height	Min Slope	Max Slope	Texture Size	Remove Texture
None (Texture ⊙	0.1	0.2	0	1.5	X 50 Y 50	☐

Figure 8: Texture Map foldout.

Let's now give an explanation of all the fields below the Texture map label:

- *Texture*: select a 2D texture image from the *Seamless Textures* folder or a colored 2D texture image from the *Textures* folder;
- *Min Height*: the Terrain's height where this texture must start, between 0 and 1;
- *Max Height*: the Terrain's height where this texture must end, between 0 and 1;
- *Min Slope*: the value of the minimum Terrain's slope where the texture must be applied, between 0 and 90;
- *Max Slope*: the value of the maximum Terrain's slope where the texture must be applied, between 0 and 90;
- *Texture Size*: allows you to choose the UV tiling size by defining X and Y dimensions;
- *Remove Texture*: if checked, you can remove this Texture Map's line.

Explanation of the buttons:

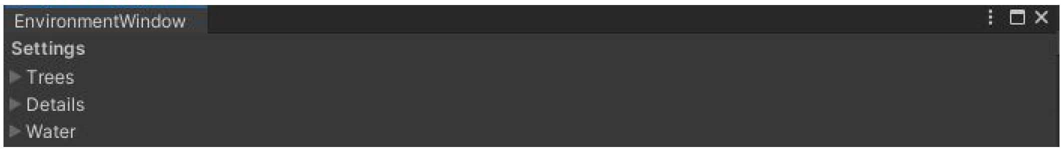
- (+): adds a new line to select more texture images;
- (−): removes all the lines toggled with Remove Texture Color;
- *Apply texture*: applies all the textures on the Terrain with the defined values;
- *Reset texture*: removes all the textures from the Terrain.

For each texture application the algorithm adds a component of randomness plus a sine wave to the height values to blend the textures together in a more natural way.

For more information about Splat Prototype properties for applying textures, here you can find the Unity documentation about it.

2.3 Environment Window

This is the window which allows you to add vegetation and other details to your landscape. You can populate it with trees, bushes, and grass to make it more realistic and complete. Its Settings are shown in Fig. 9.


EnvironmentWindow

Settings

▶ Trees

▶ Details

▶ Water

Figure 9: Environment Window.

Window components:

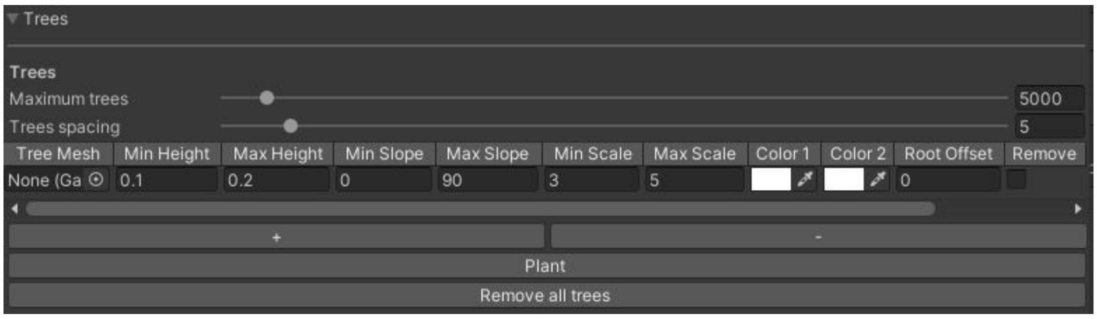
- *Trees*: this foldout allows to add trees on the Terrain;
- *Details*: this foldout allows to add details, such as grass, on the Terrain;
- *Water*: this foldout allows to add a water cube.

Now, it will be given an explanation of what each of these three environment foldout contains.

2.3.1 Trees

The Trees foldout (Fig 10) is used to add trees. Tree Prototypes can be selected among the prefabs in *Terrain Assets* ▷ *Trees* or in *Terrain Assets* ▷ *Bushes* folders, both in the *Assets* folder.

Each Tree Protoype can be placed according to the terrain steepness at a defined height on the terrain. Each Tree Instance it's planted within the range of heights you define plus a little random value to add more variability. You can also choose a range between each Tree Instance's scale can randomly change.


▼ Trees

Trees

Maximum trees 5000

Trees spacing 5

Tree Mesh	Min Height	Max Height	Min Slope	Max Slope	Min Scale	Max Scale	Color 1	Color 2	Root Offset	Remove
None (Ga	0.1	0.2	0	90	3	5	⬜	⬜	0	☐

Figure 10: Trees foldout.

Here below a description of each field contained in the Trees label:

- *Maximum trees*: this slider allows to select the maximum number of Tree Instances placeable on the terrain;
- *Trees spacing*: this slider allows to decide the distance of each Tree Instance from another;
- *Tree Mesh*: you must select the tree Game Object to generate the Tree Prototype;

- *Min Height*: the starting Terrain's height for this Tree Prototype, between 0 and 1;
- *Max Height*: the ending Terrain's height for this Tree Prototype, between 0 and 1;
- *Min Slope*: the value of the minimum Terrain's slope where the trees can be placed, between 0 and 90;
- *Max Slope*: the value of the maximum Terrain's slope where the trees can be placed, between 0 and 90;
- *Min Scale*: the minimum size of each Tree Instance;
- *Max Scale*: the maximum size of each Tree Instance;
- *Color1*: first color of each tree in this Tree Prototype, which will be interpolated with Color2;
- *Color2*: second color of each tree in this Tree Prototype, which will be interpolated with Color1;
- *Root Offset*: this value denotes how much the roots are placed inside the Terrain;
- *Remove*: if checked, you can remove the selected Tree Prototype's line.

Explanation of the buttons:

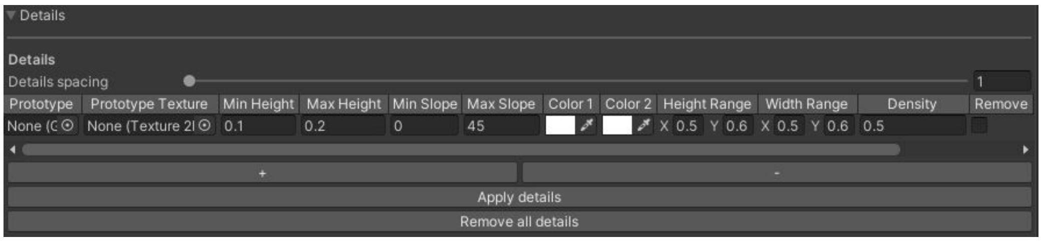
- (+): adds a new line to add another Tree Prototype;
- (−): removes all the lines toggled with Remove;
- *Plant*: adds all the Tree Prototypes on the Terrain;
- *Remove all trees*: removes all the trees from the Terrain.

You can find more information about Tree Instance properties on this Unity page.

2.3.2 Details

The Details foldout, expanded in Fig. 11, is in charge of adding Details, like grass, on the Terrain. The Game Objects suitable for this case can be found inside the *Terrain Assets* ▷ *Grass* folder contained in the *Assets* folder.

Even in this case, it is possible to adjust a set of parameters to control features such as the position on Terrain, which is also based on the Terrain steepness, the size of the Details, and their colors. For each Detail is added a random value to its defined position.



Details

Details spacing 1

Prototype	Prototype Texture	Min Height	Max Height	Min Slope	Max Slope	Color 1	Color 2	Height Range	Width Range	Density	Remove
None (C	None (Texture 2I	0.1	0.2	0	45	⬜	⬜	X 0.5 Y 0.6	X 0.5 Y 0.6	0.5	☐

Figure 11: Details foldout.

Below, it is given a description of each field of the Details label:

- *Details spacing*: this slider allows to select the spacing between Details Objects;
- *Prototype*: allows to add a Detail Mesh rendered in Vertex Lit Mode for solid objects;
- *Prototype Texture*: allows to add a Detail 2D Texture rendered in Grass Mode;
- *Min Height*: the starting Terrain's height for this Prototype, between 0 and 1;
- *Max Height*: the ending Terrain's height for this Prototype, between 0 and 1;
- *Min Slope*: the value of the minimum Terrain's slope where the details can be added, between 0 and 90;
- *Max Slope*: the value of the maximum Terrain's slope where the details can be added, between 0 and 90;
- *Color1*: color when the details are healthy (in the central patches of detail);
- *Color2*: color when the details are dry (in the patches of detail on the edge);
- *Height Range*: defines the height range for each Detail in the Detail Prototype;
- *Width Range*: defines the Width range for each Detail in the Detail Prototype;
- *Density*: allows to control the density of detail objects in the area;
- *Remove*: if checked, you can remove the selected Detail Prototype's line.

Explanation of the buttons:

- (+): adds a new line for another Detail Prototype;
- (−): removes all the lines toggled with Remove;
- *Apply details*: adds all the Details Prototypes on the Terrain setting Details Layers;
- *Remove all details*: removes all the details from the Terrain.

You can find more information about Detail Prototype properties on this Unity page.

2.3.3 Water

This foldout, whose details are shown in Fig 12, allows to add a water cube in your environment, using the *Add water* button. You can also remove the added object using the *Remove water* button.

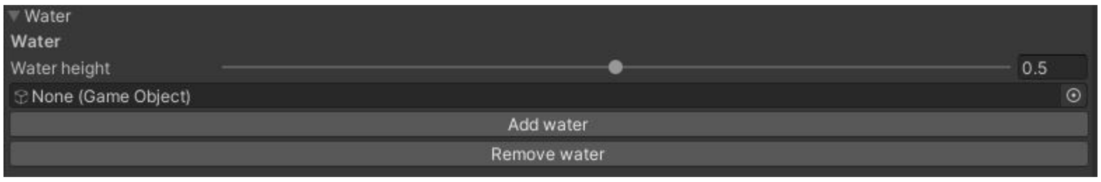

Water
Water
Water height 0.5
None (Game Object)

Figure 12: Water foldout.

In the Game Object field you can add a water Game Object, which you can find in *Standard Assets* ▷ *Environment* ▷ *Water (Basic)* ▷ *Prefabs* in the *Assets* folder. The slider *Water height* controls the level of the water starting from zero.

3 Area Baker

The Area Baker is the Game Object in charge of baking the Unity Navigation Mesh at run-time. This is, indeed, the Game Object to which the *NavMeshSurface* component is attached. This component is not in the Unity standard install, for more information about it and how to access it, see the documentation page. All the Area Baker's components are shown in Fig .

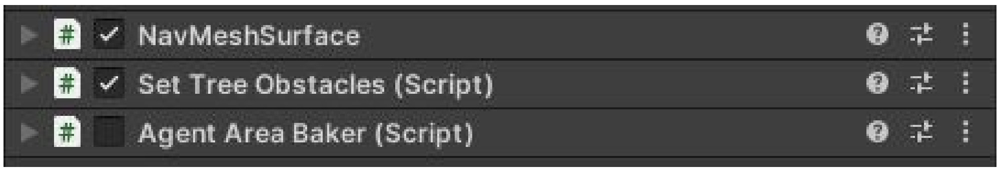


Figure 13: Area Baker components in the Unity Inspector.

In the next paragraphs the main functions of the Script components are shown.

3.1 Set Tree Obstacles

This Script will create an Obstacle on the Navigation Mesh for each Tree Instance, during the Play mode.

It takes the position and the rotation of each Tree Instance and creates a new empty Obstacle Game Object based on them, adding then a *NavMeshObstacle* component to it using the size of the Collider of the original Tree Prefab scaled to the size of the instance.

After this computation is done, the component enables the Agent Area Baker script, whose function is described in the next section.

3.2 Agent Area Baker

The main function of this component is baking the Navigation Mesh on Play. The Terrain is a walkable area, based on the Agent type and its characteristics, while the Water it baked as non-walkable.

After the Navigation Mesh is baked, the Agent Area Baker activates the button used for spawning the Agent. Each time you want to spawn the Agent this component selects a random feasible location on the Navigation Mesh as spawning point.

This procedure is done online, at run-time, because in this project it is used to test navigability of different landscapes. Baking the Navigation Mesh online is a computationally intensive operation and requires some time. When you will have the definitive Terrain to use in your game, the Navigation Mesh should be baked offline and even the spawn points for the Agent can be manually set.

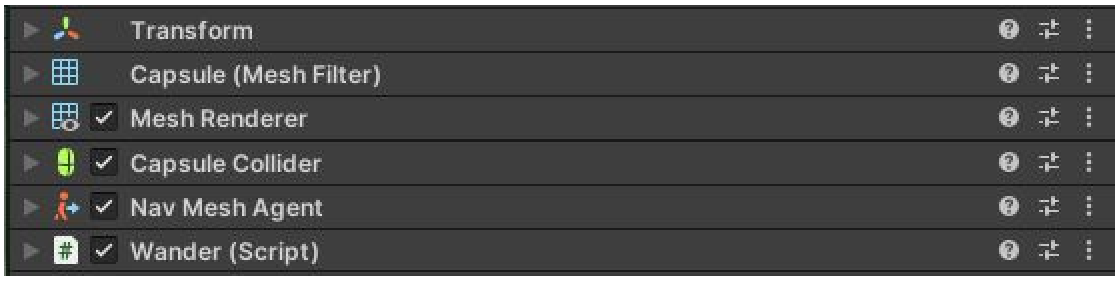
4 Agent

The Agent is used to test the navigability and explore the Terrain from a third person view. For this purpose a capsule GameObject has been created, and named *Agent*.

The Agent Prefab has a Camera as a child, which follows it during the landscape exploration.

4.1 Components

In the figure below (Fig. 14) the Agent's components of the Inspector are represented.



Transform
Capsule (Mesh Filter)
Mesh Renderer
Capsule Collider
Nav Mesh Agent
Wander (Script)

Figure 14: Agent's components in the Unity Inspector.

Here below, the explanation of each component:

- *Transform*: Unity's component for position, rotation and scale;
- *Mesh Filter*: Unity's component which takes the mesh from the assets and passes it to the Mesh Renderer;
- *Mesh Renderer*: Unity's component which receives the mesh from the Mesh Filter, and renders it on the screen;
- *Capsule Collider*: Unity's component for collision detection;
- *Nav Mesh Agent*: Unity's component for Agent's pathfinding and movement system;
- *Wander*: is the script which makes the Agent wander.

A Capsule Agent Type has been created specially for this NavMesh Agent. It is possible to change Agent Type depending on the tests that are needed to be done.

4.2 Wander

This script makes the Agent wander around the Terrain. The Wander script has three parameters editable from the Inspector:

- *Wander Radius*: the distance within will be calculated the next position;
- *Min Time*: the minimum time before the next destination's computation;
- *Max Time*: the maximum time before the next destination's computation.

As represented in Fig. 15, to decide the new destination the Agent first finds a point in a random range between a minimum and maximum value of angle degrees, in this case [−40, +40] in the direction it is currently going and within a the Wander Radius. After that, it sets the nearest position to that point on the NavMesh Surface as the destination.

Changing the direction of the Agent in a limited angle degrees' range each computation of the new destination, allows to obtain a more natural wandering avoiding the abrubt change of direction.

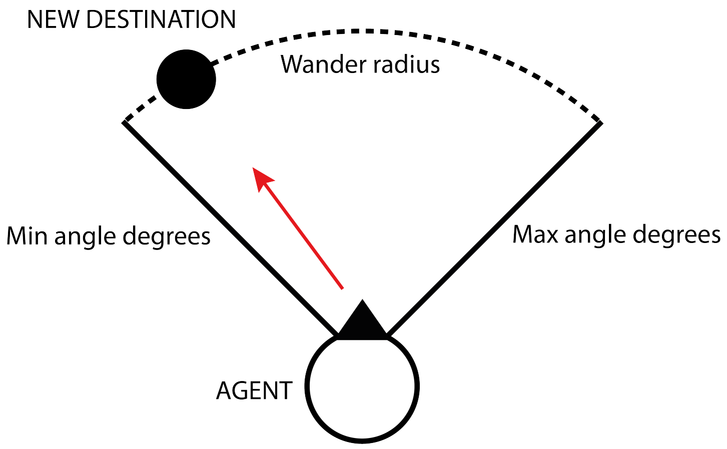


Figure 15: The Agent computes the new destination selecting a random point between an angle degrees' range and within a given distance. In the next step, as the red arrows shows, the agent turns towards the new destination point and starts to go there. Remember: this image approximates the new destination point. The right new destination point is its the nearest on the Navigation Mesh.

A new destination is computed when the previous destination is reached or after a certain period time in a random range between Min Time and Max Time.

Calculating the cross product between the Agent's x axis and the Terrain normal in each point, the Agent is rotated according to the Terrain steepness.

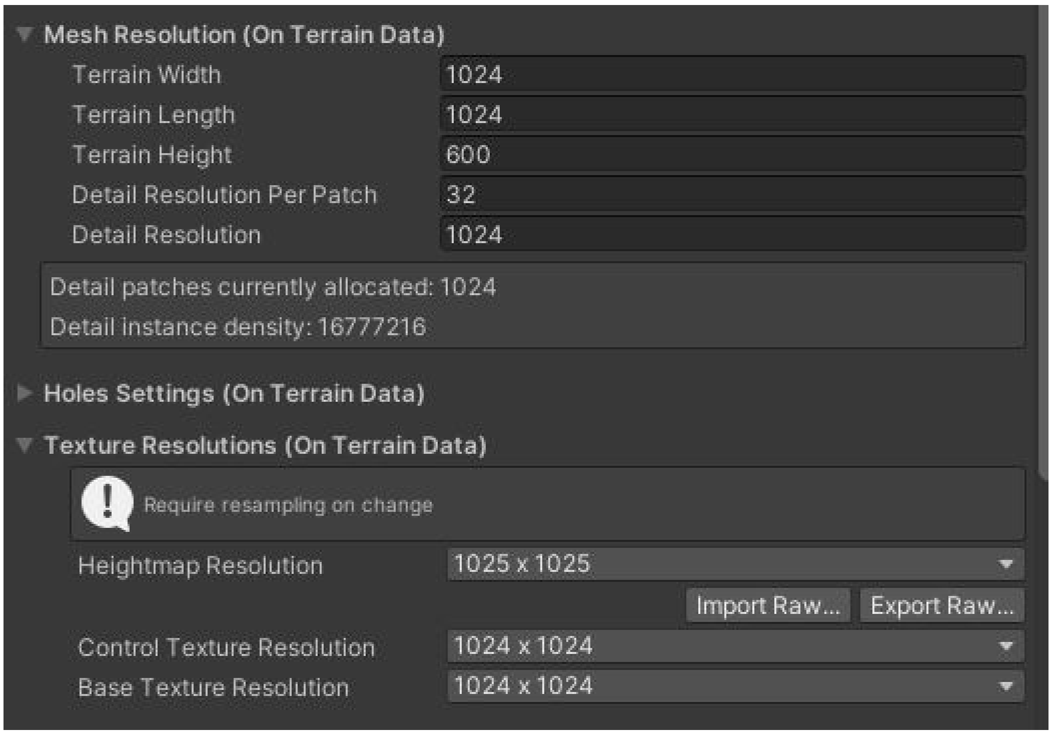
5 Test

First of all, the project must be loaded in Unity. The landscape creation step must be carried out in the Edit mode, while, to navigate the Terrain using the agent, it is necessary to enter in the Play mode.

In the project, the scene *PCG Terrain With Wander Agent* is already set up: it contains all elements and a Terrain Game Object initialized with the needed requirements and ready for the landscape generation. Alternatively, it is possible to test on a completed landscape, you can find it in the folder *Terrains* inside the *Assets* folder.

5.1 Setup

First of all it is necessary to create a Terrain Game Object, which must have the *Terrain* Layer assigned. For this project needs, the Terrain width and length must have the same size, which needs to be a power of 2. The Heightmap Resolution must be set to $(width + 1)x(length + 1)$, the Detail Resolution must be equal to $width = length$, while the Texture Resolution must be $(width)x(length)$, we can see an example in Fig. 16.


Mesh Resolution (On Terrain Data)
Terrain Width 1024
Terrain Length 1024
Terrain Height 600
Detail Resolution Per Patch 32
Detail Resolution 1024
Detail patches currently allocated: 1024
Detail instance density: 16777216
Holes Settings (On Terrain Data)
Texture Resolutions (On Terrain Data)
Require resampling on change
Heightmap Resolution 1025 x 1025

Control Texture Resolution 1024 x 1024
Base Texture Resolution 1024 x 1024

Figure 16: An example of the Terrain initialization in Terrain Settings in the Inspector.

5.2 Create landscape

The landscape must be created following the steps described in the Terrain section. The three windows can be found below Window▷ Procedural Landscape. First, the Terrain Creation Window must be used (Fig. 17); it allows you to generate the structure of the Terrain, secondly the texture should be added with the Texture Window (Fig. 18) and finally the Environment Window helps you to obtain a more natural looking environment including vegetation and other details (Fig. 18).

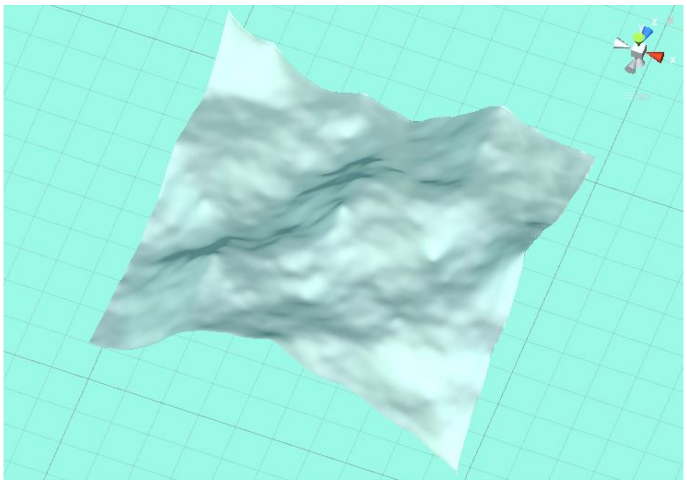


Figure 17: Terrain Game Object modified with Terrain Creation Window.

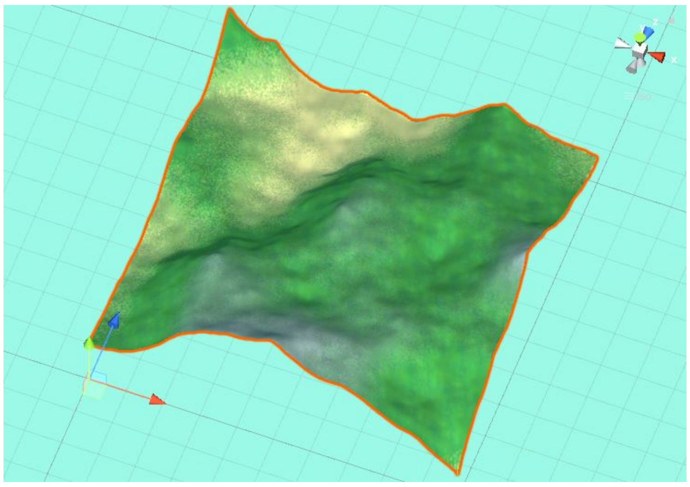


Figure 18: Terrain Game Object modified with Texture Window.

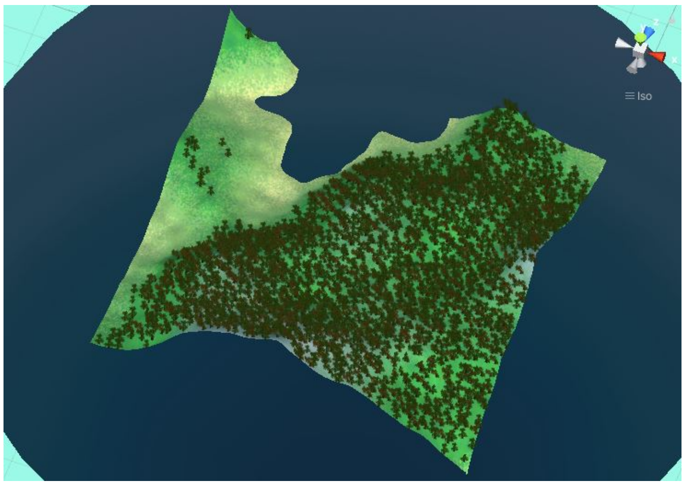


Figure 19: Terrain Game Object modified with Environment Window.

5.3 Spawn agent

Once the landscape is generated and complete, you can spawn the Agent to navigate it (Fig. 20). Pressing Play in the Unity Editor, the Navigation Mesh is created, and the Spawn Button to add the Agent appears.

The agent is placed at a random point on the Navigation Surface and starts to wander around the terrain. A third person Camera, always behind him, helps you to follow him and see the the landscape from the agent point of view.

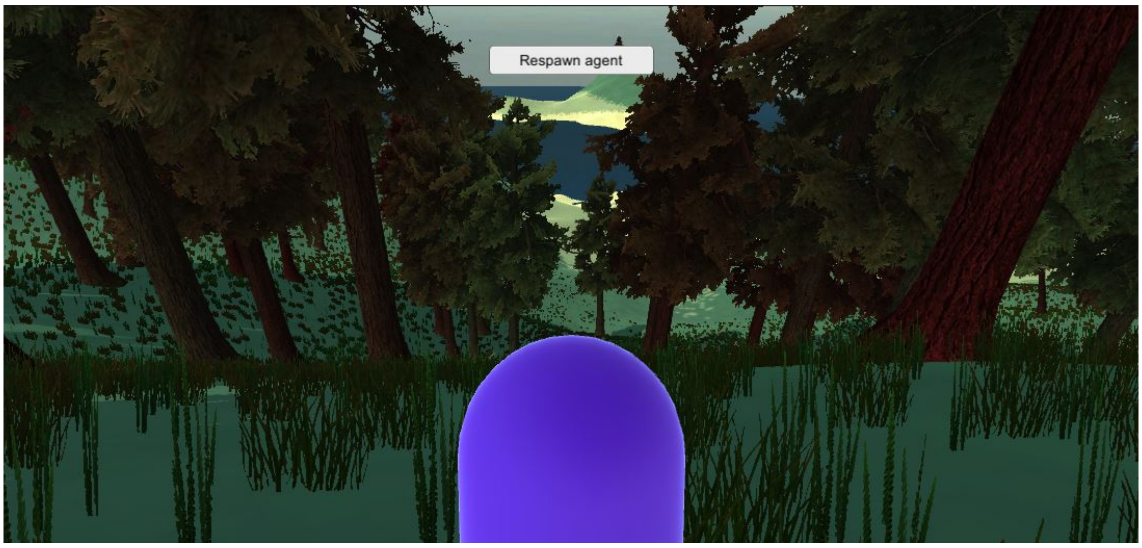


Figure 20: A screenshot of the Game view after the Agent is spawn.

This operation can give you hints about possible problems linked to navigability and an overview of the goodness of the result.

6 Conclusions and future works

The aim of this project is to explore tools that can help developers produce massive content saving time, money and other resources.

This analysis on some Procedural Generation of Terrains techniques shows only a small fraction of the potential of this method. In future works will be interesting to delve deeper in this method testing other approaches for the generation of fractal Terrain or agent-based approaches, or them to obtain different results.

In this work the landscape is made up by only one single Terrain Game Object. It is fundamental to give the possibility to generate a larger world combining different chunks and this could be the next area of investigation, maybe adding more kinds on environment and assets.

The presence of the Agent can give an idea of the of the possibilities of movement in the Terrain and a close view of its structure. Other elements, such as other Agent types, group of Agents, or a human-player, are necessary to test the playability. Even though the Agent helps in exploring the landscape, it is difficult to evaluate the validity of the generated Navigation Mesh: a designer should check it.

To conclude, I think that a manual refinement of the Terrain will be always necessary to create an usable map and validate the results.