Kevin Loddewykx

Documentation v1.0

Unity® Framework

Tested with Unity® v2018.2.1f1

Contents

1 Introduction

A open-source framework for Unity® , by Kevin Loddewykx. The framework can be found at GitHub

2 Transform Inspector

A custom inspector for the Transform component, which allows the user to easily reset the position, rotation and scale. Or copy the settings from one gameobject to another or to the clipboard. The script is a customized version of the TransformInspector.cs script created by LuGusStudios, which can be found in some of their open source projects on their GitHub. Added the [▲] and [▼] buttons, and changed the padding, margins and the text of the other buttons.

Fig. 1: Custom transform inspector.

- [0] and [1]: Sets the X, Y and Z values of the row it belongs to, to the specified number.
- [C] and [P]: Are for copying and pasting respectively. Allowing the user to copy the transform data from one gameobject to another. The buttons are backed by three Vector3 variables, so you can not copy a position and paste it into a rotation.
- [S] Copies the content of the row to the clipboard as a string, for example copying the position will put "Vector3(6,58f, 0f, 0f)" on the clipboard.
- [Uniform X scale]: Takes the X scale and assigns it to the Y and Z values of the scale.
- [▲]: Selects the parent object if the gameobject has one, otherwise the button will be disabled. It will ignore/skip over parent objects with the hideFlag "HideInHierarchy" set.
- [▼]: Selects the first child without the hideFlag "HideInHierarchy" set, if none exists the button will be disabled.
- [+]: Adds a new empty child to the gameobject and selects it. With its local position and rotation set to zero and scale to one.

3 Weighted Array

A "WeightedArray", is an array consisting out of elements of the class "WeightedObject". A WeightedObject has a weight property which determines how much weight it contributes to the total weight of the array. The WeightedArray class has methods to get a random WeightedObject or index with the chances determined by its contribution to the totalweight of the array, and other useful methods. The constructor takes in a bool parameter which tells the array to ignore empty elements or to take them into account.

Before calling any methods on the array, after changing something to its elements, make sure to call "RecalcTotalWeight" to initialize the total weight properly.

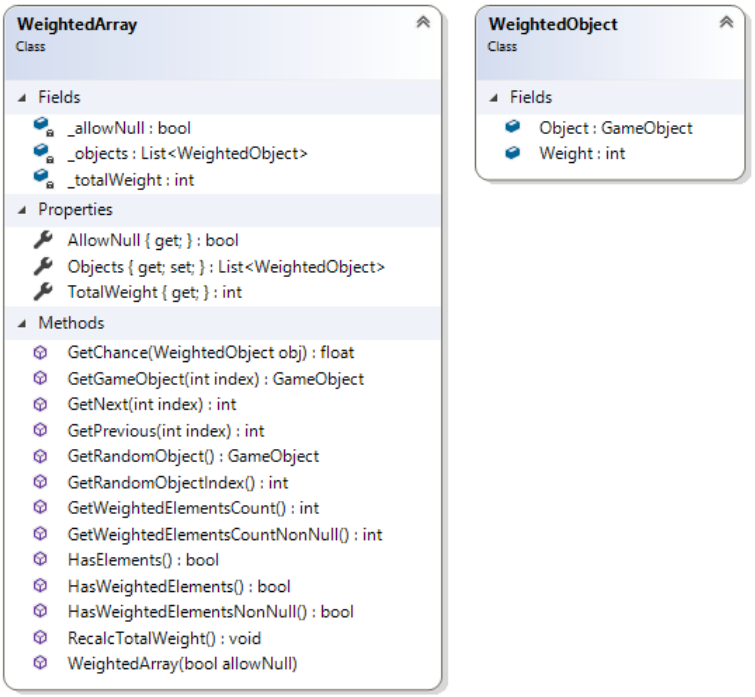

Fig. 2: The class diagram of WeightedArray and WeightedObject.

4 Weighted ScriptableObject

"WeightedScriptableObject" is and abstract class inheriting from "ScriptableObject", containing two abstract get properties, namely "Elements" of the datatype "WeightedArray[]" and "Names" of the datatype "string". The abstract class also contains some pass-through methods for calling the methods on the element accessed with the passed in "elementIndex" argument and a property to get the first element from the Elements array. The names property is only used inside the editor and is therefore compiled away in none editor builds.

The class has a custom editor script for rendering the inspector, see image below. This script automatically triggers the "RecalcTotalWeight" method, when a change to the WeightedArray occurs, and shows the chance percentages of the gameobjects in the UI. You can add new items to list by dragging or assigning an item to the objectfield behind the "Add Element" label. Removing and item from the list can be done by pressing the [X]. When you set a gameobject to none inside the list and the array doesn't allow for empty elements, it will be automatically removed.

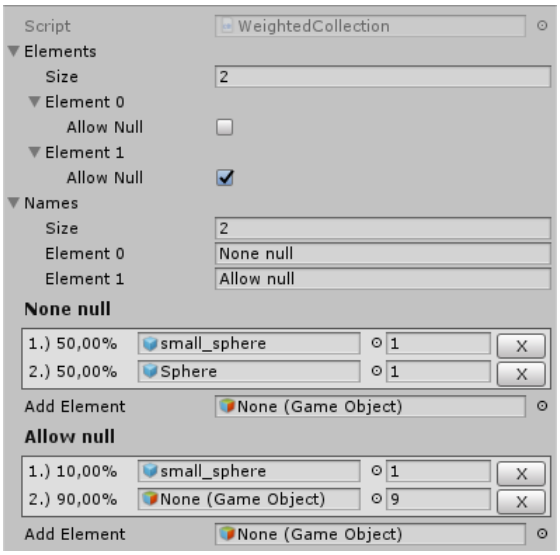

Fig. 3: Inspector of a derived class of WeightedScriptableObject.

The code below belongs to the scriptableobject in the image above, you can add the [HideInInspector] attribute to "_elements" and "_names", if you don't want the user to change the amount of arrays.

```
using Helpers_KevinLoddewykx.General.WeightedArrayCore;
using UnityEngine;

namespace Helpers_KevinLoddewykx.General
{
    [CreateAssetMenu(menuName = "Resources/Weighted Collection")]
    public class WeightedCollection : WeightedScriptableObject
    {
        [SerializeField]
```

```
        private WeightedArray[] _elements = new WeightedArray[] { new
        WeightedArray(false) };

        public override WeightedArray[] Elements
        {
            get { return _elements; }
        }

#if UNITY_EDITOR
        [SerializeField]
        private string[] _names = new string[] { "Weighted Gameobjects" };

        public override string[] Names
        {
            get { return _names; }
        }
#endif
    }
}
```

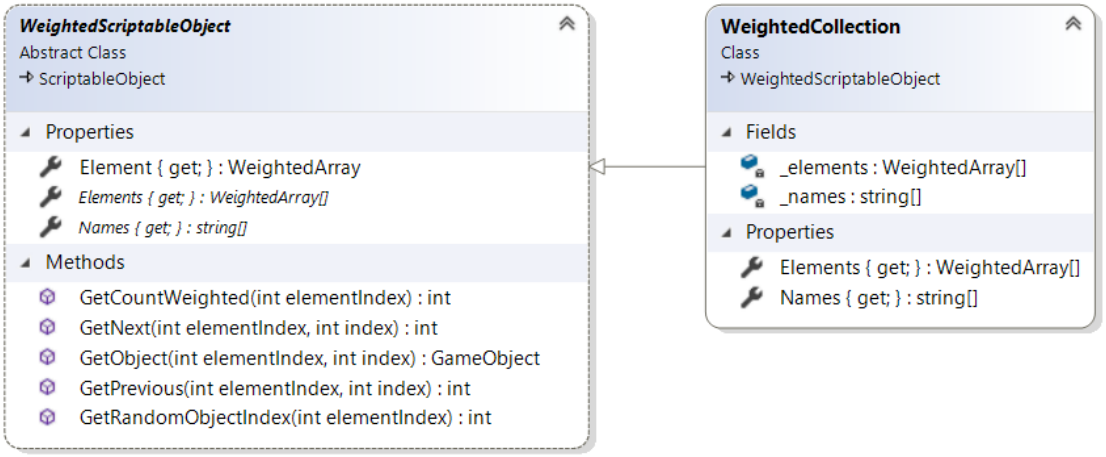

Fig. 4: The class diagram of WeightedScriptableObject and WeightedCollection.

5 Poisson Disk Distribution

> Poisson-disc sampling produces points that are tightly-packed, but no closer to each other than a specified minimum distance, resulting in a more natural pattern.
>
> *Jason Davies*

The distribution method is implemented inside the Editor project and therefore can not be used at runtime. The algorithm is computational intensive and will lock up unity® until it is finished.

It can be run from an EditorWindow, which can be opened from "Menubar -> Tools -> Poisson Disk Distribution". Or from a gameobject inside the scene when you attach the PoissonPlacer component to it.

It supports multiple levels/layers of Poisson disk sample points. Allowing distributing gameobjects between already placed objects from the previous level, when the minimum distance setting [5.5] is smaller than that of the previous level. Each level has it owns WeightedScriptableObject [4] for determining the gameobjects to place. Each gamebobject inside the WeightedArray [3] has it owns settings [5.5].

The main reference for the implementation is from: [Tul09].

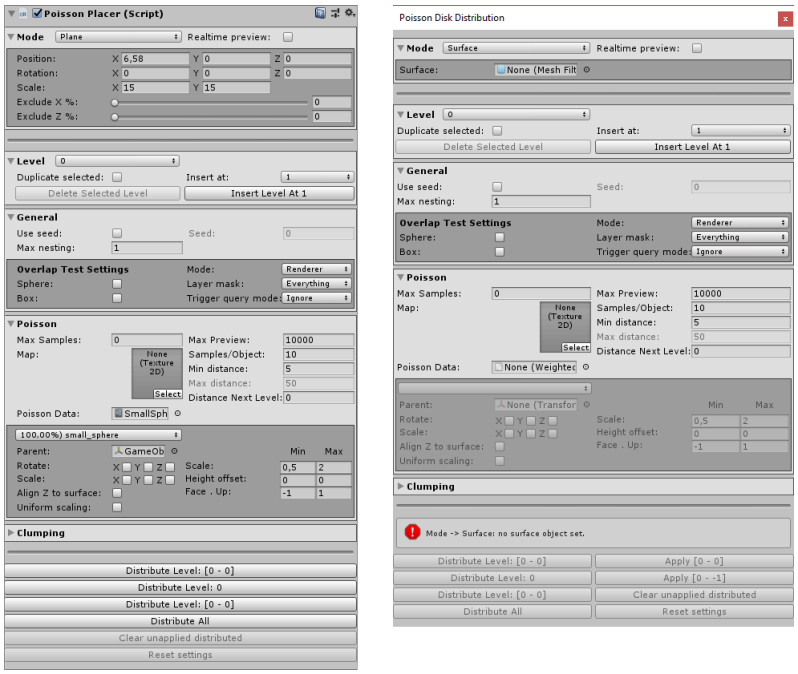

Fig. 5: Left the monobehaviour inspector, right the editor window.

5.1 Known issues

- The inspector its horizontal scrollbar only shows up when the vertical is also needed.
- Presets don't work properly.
  - The distribution buttons aren't removed from the inspector when inspecting it.
  - When assigning a preset it will overwrite the variables responsible for keeping track of the placed objects.
- When the inspector or EditorWindow is rendered, it will set the left and right margins of EditorStyles.numberField and EditorStyles.layerMaskField to zero, otherwise the controls wouldn't align properly inside the user interface. These changes are permanent until the Editor is restarted or overwritten somewhere else. Tried resetting them at the end of OnInspectorGUI and OnGUI, but then the changes would not affect the rendering.
- When you have a visual helper selected and trigger a recompilation of the Engine a "NullReferenceException: (null)" will be logged in the console. This exception won't affect the workings of the editor or the distribution. This error is due to the visual being destroyed inside the OnDisable and recreated inside the OnEnable of the MonoBehaviour or EditorWindow.
- When you remove the PoissonPlacer component from a gameobject, and afterwards undo this operation a "CheckConsistency: Transform child can't be loaded" will be logged in the console. This is due to visual helper being a child of the gameobject containing the MonoBehaviour and having the hideFlags DontSave set. Which causes a mismatch in the childCount. The visual will be automatically re-added afterwards in the OnEnable of the MonoBehaviour.
- You can take copies of the visualizer

5.2 Foldout: Mode

There are 5 modes, for setting up the region in which the objects are distributed. The mode determines also if raycasting is used for placing the objects, so objects are only placed on top off other colliders. Each mode has its own specific sub-settings.

The mode tab also contains a toggle for enabling the real-time preview. Like mentioned before the algorithm is computational heavy, so use it with care. Under the Poisson foldout [5.5] you can set for each level the maximum amount of samples to calculate, when setting it to <= 0 the algorithm will run until completion.

The modes **Plane**, **Ellipse**, **Projection Plane** and **Projection Ellipse** will show a visualizer inside the scene view and have similar sub-settings:

- **Layermask** (only projection modes)
- **Trigger query mode** (only projection modes)
- **Position** of the visualizer
- **Rotation** in euler angles of the visualizer
- **Scale** of the visualizer (projection modes: 3D, others: 2D ignores Y)
- **Exclude _ %** sliders [0 - 1] for excluding a region starting from the center

The visualizer will not show up in the Hierarchy panel, due to the hideFlag hideInhierarchy being set, but is still selectable and with an editable transform. So you can control the distribution region from inside the UI or inside he scene view by dragging, scaling or rotating the visualizer. Changes to the visualizer will be directly reflected in the UI and vice versa. The position, rotation and scale are in world space for the EditorWindow, due to the visualizer not having a parent. And in local space for the MonoBehaviour, due to the visualizer being attached to gameobject the MonoBehaviour resides on.

The projection modes are using raycasting. The rays are casted with the method Physics.Raycast, from a sampled point inside the top of the visualizer taking into account the transform of the visualizer, with the height of the viualizer as ray length, and utilizing the layermask and trigger query mode for collision filtering. If the ray hits a collider and the point is valid a new gameobject will be instantatiated at that location.

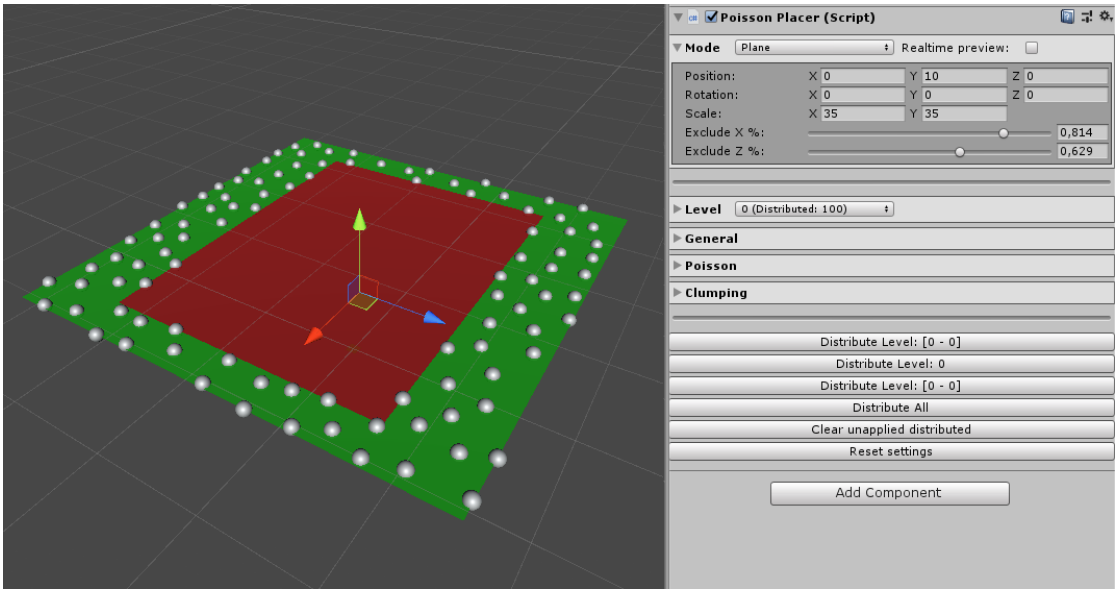

Fig. 6: Mode: plane with exclude region set.

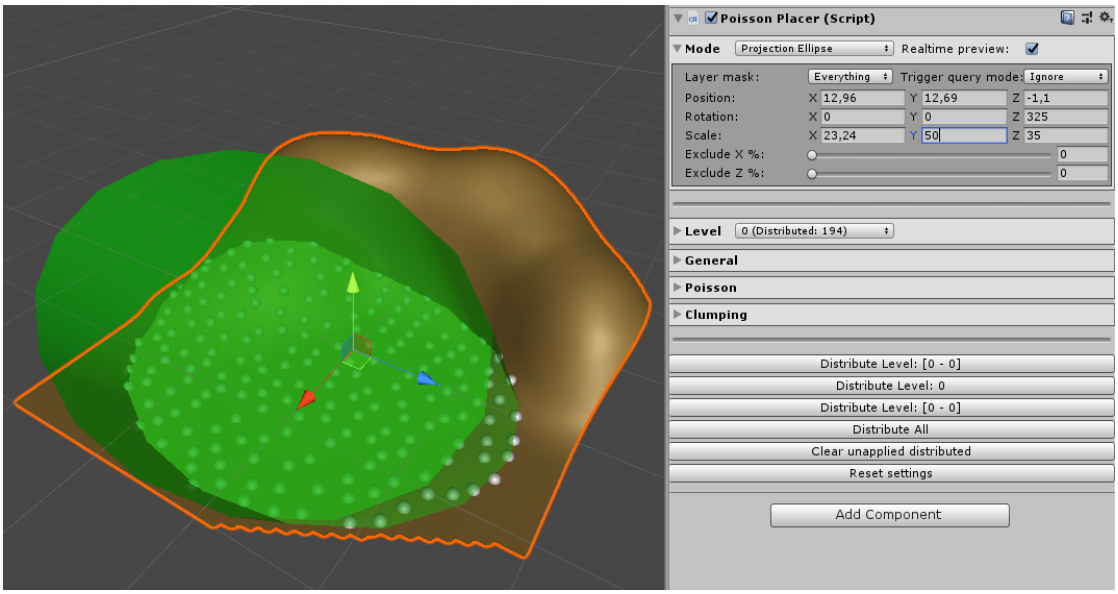

Fig. 7: Mode: projection ellipse.

The mode **Surface** has one setting, namely the **Surface** GameObject to use. When using this mode from a Monobehaviour this setting is disabled and will show the gameobject it is attached to. For the EditorWindow the user can pick freely an object from the scene as long as it has an active collider. The region is then determined by combining the bounds of all the active colliders on the gameobject and it childs.

This mode also uses raycasting, but doesn't take the rotation of the object into account and will always cast straight down from the top of the bounds, with a ray length of bounds.size.y. These rays aren't casted with Physics.Raycast, but the algorithm loops over all the active collliders inside the surface and uses the Raycast method from the Collider class to determine if a point is valid.

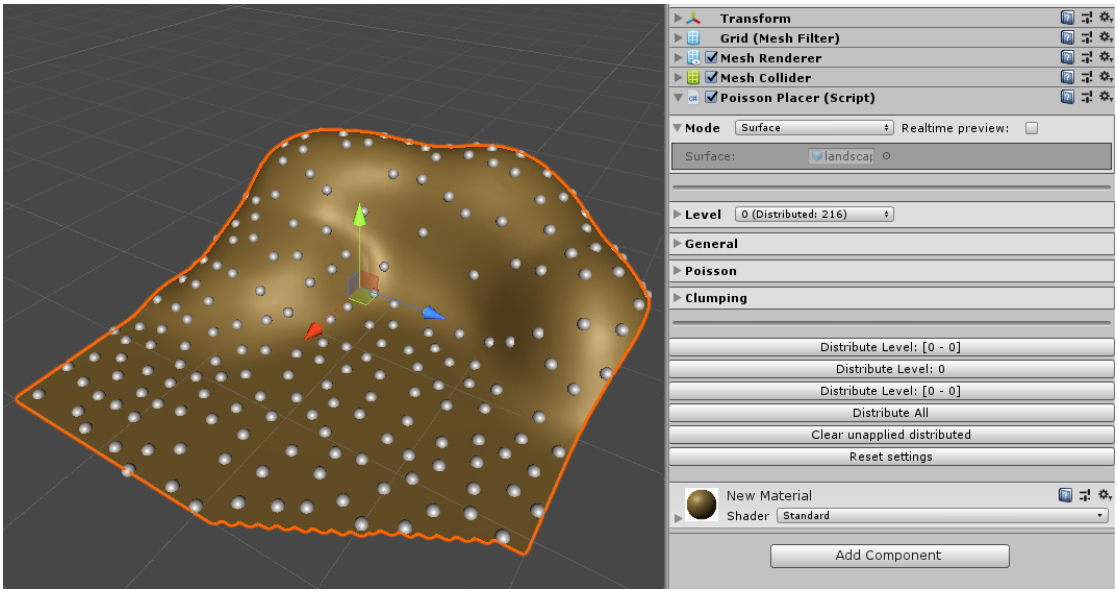

Fig. 8: Mode: surface

5.3 Foldout: Level

When using multiple levels with decreasing min and max distances you can distribute multiple WeightedArrays in the same region, while taking into account the levels above.

The dropdown field behind the foldout label, determines from which **Level** the settings are shown inside the General [5.4], Poisson [5.5] and clumping [5.6] foldouts.

When clicking **[Insert Level At _]**, you insert a new level. You can choose where in the stack to insert it with the field **Insert At**. And with the toggle **Duplicate Selected** you can choose to duplicate the current level its settings or add a level with the default settings. After insertion the newly added level will be selected.

The button **[Delete Selected Level]** will remove the active level. If a level has been applied, see section [5.7], you won't be able to insert a level before it or delete the level.

Checking if the point is to near to a gameobject of a previous level is done with:
max(PrevLevel.DistToKeepNextLevel, finalMinDist) < distToObject
whereby:
PrevLevel.DistToKeepNextLevel <= max(PrevLevel.MaxDist, PrevLevel.MinDist)
and when no map is set:
finalMinDist = CurrLevel.MinDist"
or when a map is set:
CurrLevel.MinDist + (mapSample(x, y).grayscale * (CurrLevel.MaxDist - CurrLevel.MinDist))

The DistToKeepNextLevel can not be set higher then the maximum of the two inputted distance values due to a optimization which is being used and described in [Tul09]. This optimization ensures that the algorithm doesn't need to do a distance check with all the placed objects.

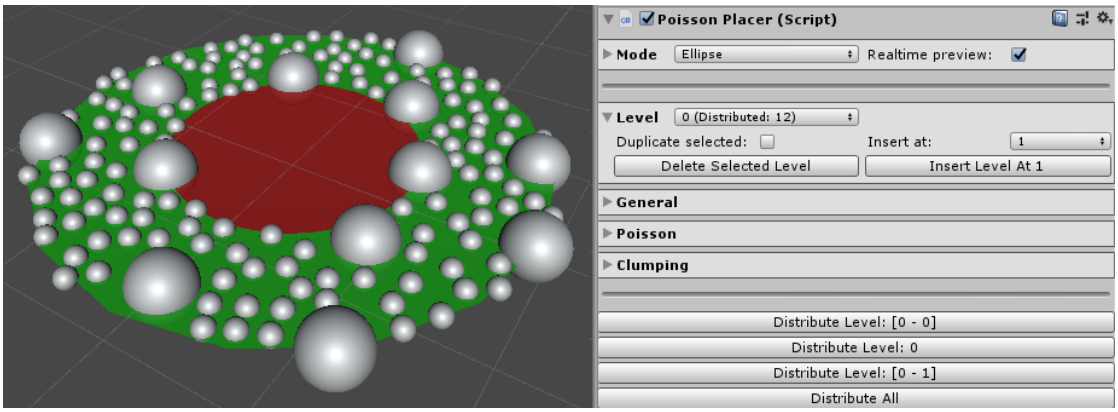

Fig. 9: 2 levels, large spheres (min dist = 5, dist next level = 2) and small spheres (min dist = 1.1).

5.4 Foldout: General

The general foldout is level dependent, and affects the placement of objects declared under the Poisson and Clumping foldout.

If you want to put in a seed value for the algorithm you can enable the toggle **Use seed**, which enables/disables to input field **Seed**, to input your own integer value as seed. The value is passed into the static InitState method of the UnityEngine.Random class.

When the distributed gameobject contains a PoissonPlacer MonoBehaviour at its root, this placer can be automatically triggered. The value **Max nesting** controls how deep the triggering goes. Whereby zero = no triggering, 1 = one level deep, 2 = two levels deep, ... This is to prevent infinite loops when you have a cyclic dependency, like a WeightedScriptableObject containing a placer which distributes that scriptableobject and then trying to distribute the scriptableobject.

The last 5 settings in the foldout are for controlling the overlaps checks, **box** and **sphere**, default these are turned off. When turned on the algorithm will calculate the objects its bounds, these can be calculated from the active Renderer or Collider components depending on the **Mode** setting. Then it will use these bounds in the corresponding Physics.Overlap[Box/Sphere] method, together with the settings **Layer Mask** and **Trigger query mode**. The overlap checks are always against the colliders in the scene, never against the renderers their bounds. When the calculated bounds overlap with something in the scene, the gameobject is not placed.

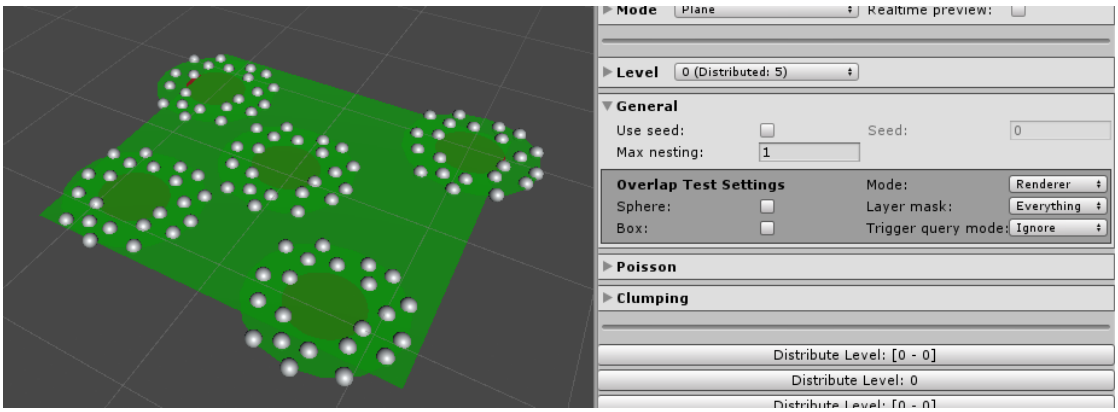

Fig. 10: Distributing nested placers

5.5 Foldout: Poisson

The Poisson foldout is level dependent and exists out of two parts, the first part affects how the samples are sampled and the maximum amount of samples. The second part contains the per object settings.

The **Max Samples** and **Max Preview**, determines how many samples can be calculated before the algorithm is stopped, when one of distribution buttons [5.7] is pressed or the realtime button [5.2] is enabled respectively. When the inputfield is set to 0, the algorithm runs unitl completion. The **Samples/Object** sets how many new samples are calculated around the current sample point if it is valid.

When a point is found, new points are sampled around the point in a circle/disk shape. The minimum radius and max radius of the disk is based on the input fields **Min Distance**, **Max Distance** and **Map**. The minimum radius equals to the inputted minimum distance when no map is given or

"min distance" + (map.Sample(x, y).grayscale * ("max distance" - "min distance"))

when a map is assigned. The maximum radius is two times the calculated minimum radius. When a map is given the color black equals to the minimum distance and white to maximum distance. The **Distance Next Level** tells the algorithm how far the levels below the current level need to keep their distance from the placed sampled points. This value can not be bigger then the maximum of "Min Distance" and "Max Distance". "Max Distance" can be smaller then "Min Distance".

When using a map/image, **Read/Write** needs to be enabled inside the Import Settings under the Advanced foldout, otherwise the image can not be sampled by the CPU.

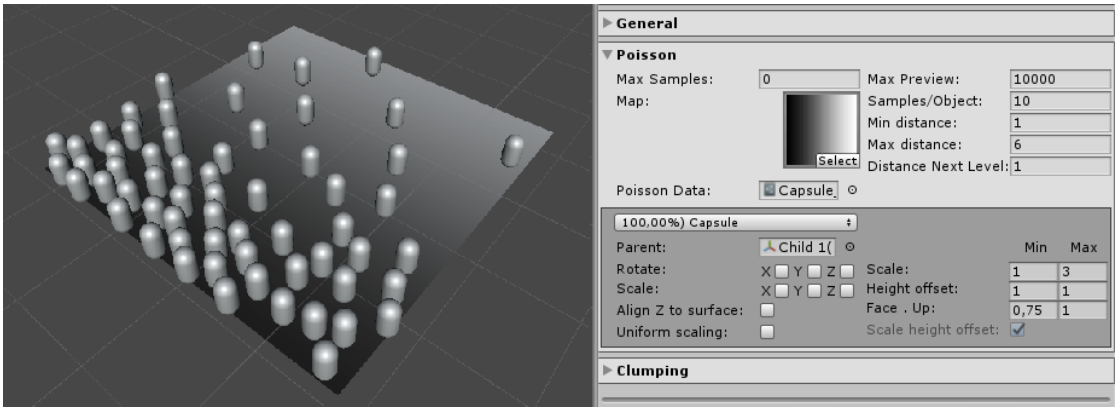

Fig. 11: Distributing with a map

The inputfield **Poisson Data** requires a WeightedScriptableObject [4], for determining which objects can be placed, atleast one entry inside the WeightedArray needs to contain a value, none null. With the dropdown below it you can select the none null objects for visualizing and editing their associated settings.

- **Parent** sets to which gameobject the generated gameobjects need to be parented.
- **Rotate** toggles to which axis a random rotation is applied, X and Z can only be turned on when "Align Z to surface" is turned off.
- **Scale** toggles to which axis a random scale is applied.

- **Align z to surface** toggles if the up vector of the gameobject needs to algign with the surface normal.
- **Uniform scaling** when enabled will make the axis which have scaling enabled have the same random scale.
- **Scale (min - max)** sets the interval of the random scale.
- **Height offset (min - max)** sets the interval for a random height offset in the direction of the surface normal.
- **Face . dot (min - max)** sets the interval for how flat or steep the surface needs to be. Calculated by dot(surface up vector, world up vector)
- **Scale height offset** when enabled the height offset is scaled by the calculated random Y scale, only enabled when the Y scale toggle is enabled.

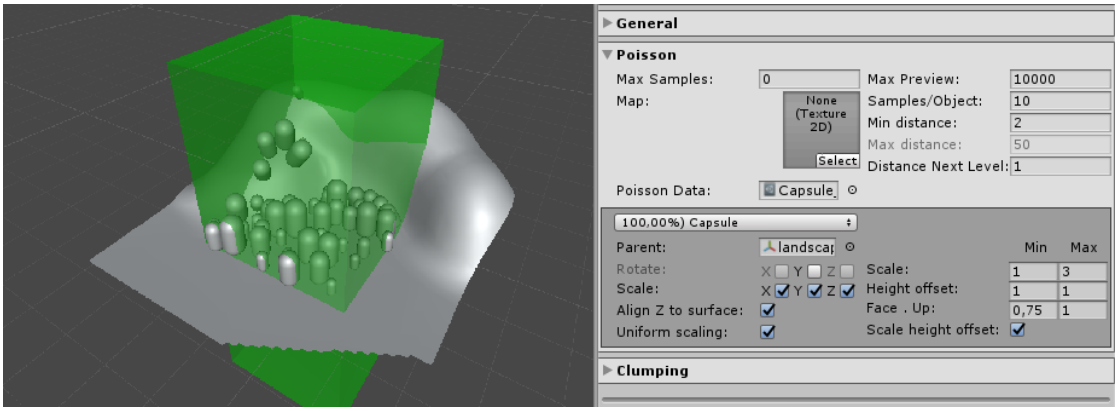

Fig. 12: Distribution with several settings turned on.

5.6 Foldout: Clumping

The Clumping foldout is level dependent, and controls a small very basic optional clumping algorithm, which is controlled by four input fields: **Min clumping**, **Max clumping**, **Min clump range** and **Max clump range**. The clumping logic is ran for each succesfully placed object, coming from the Poisson sampling. It will place objects around the created object its pivot point, with the min/max clump range inputfields controlling the nterval how near and far the objects can be placed from the pivot point and the min/max clumping fields the amount of objects to be placed. The objects will also be placed with raycasting if the mode requires it, overlap checks if enabled and region checks, but will not check if the object is in the neighbourhood of other objects. The created objects by the clumping algorithm are also not taken into consideration when doing the in neighbourhood check for the Poisson samples.

The **other settings** are the same as the second part of the Poisson foldout [5.5].

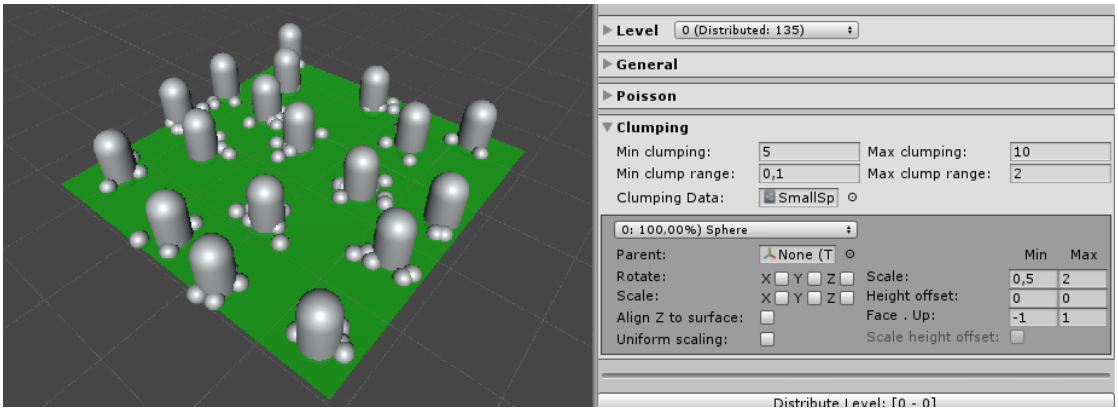

Fig. 13: Distribution with clumping.

5.7 Buttons

At the bottom of the UI the errors are shown and the buttons.

The first four buttons are for triggering the distribution.

- **[0 - current level]**
- **[current level]**
- **[current level - last level]**
- **[0 - last level]**

The buttons will be disabled when a level in the interval is not in a valid state, or a level before the start of the interval hasn't been distributed yet.

The two apply buttons only exists in the EditorWindow. They are enabled when the levels of the interval are distributed, and contains a none applied level. When applying a level you can no longer change the settings inside the Mode foldout, redistribute the applied level. and insert a new level before it.

- **[0 - current level]**
- **[0 - highest distributed level]**

The **[Clear Unapplied Levels]** button will clear all the placed objects which are from a unapplied level. The **[Reset Settings]** button will reset all the settings back to their default value and destroy all the objects of the unapplied levels.

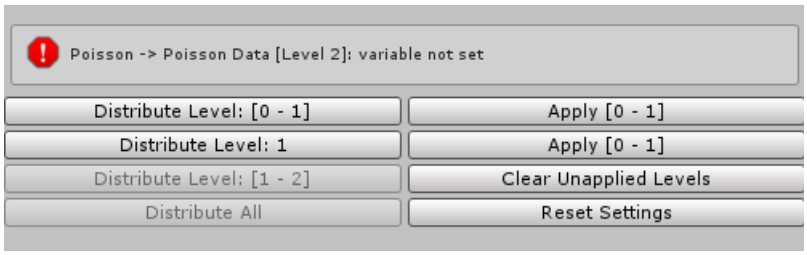

Fig. 14: Buttons.

References

[Tul09] Herman Tulleken. Poisson disk sampling.
http://devmag.org.za/2009/05/03/poisson-disk-sampling/, 2009.