Modular Buildings Framework

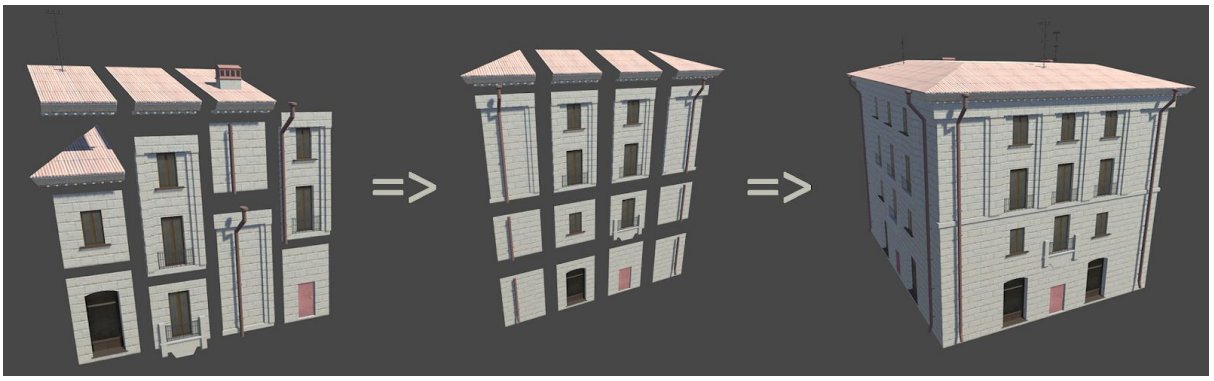
Version 1.0

Table of contents

General description

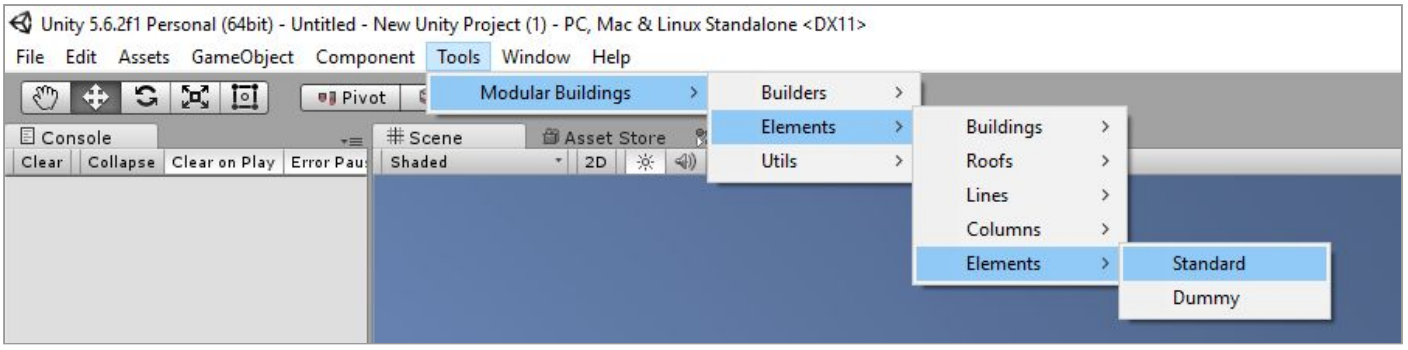
Buildings are made of different elements.

We take a few elements and combine them in columns, columns are combined in lines, lines - in walls. Add a roof on top and in the end we get a finished house.



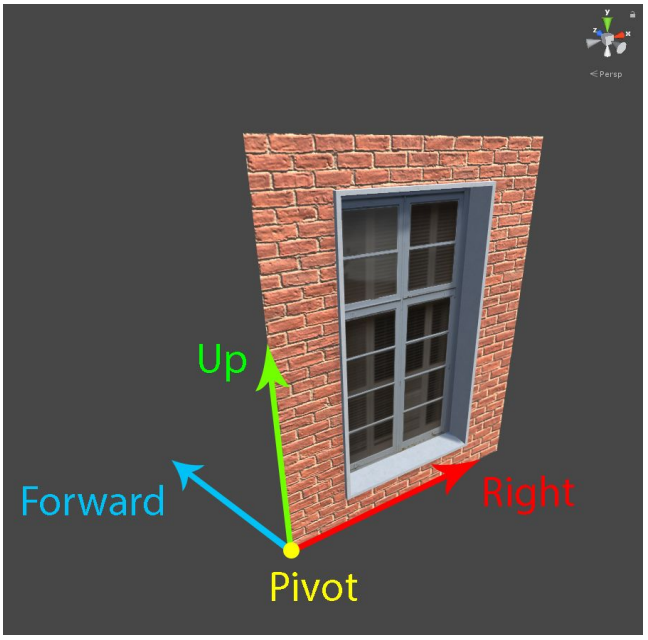
After the import of package you will get the new menu item: Tools/Modular Buildings/

- Elements - creation of assets with builders
- Builders - editor windows for generation different types of elements
- Utils - different useful utilities



Axis agreement

The following axes agreement is used.
Forward direction is a direction, in which we enter the house if it had a door. Usually pivot of element/prefab is bottom left point of an element.



Drafts

All elements, columns, etc. are built by using drafts.

Drafts describe the parameters of a generated object.

We create drafts, set up their parameters and send them in builders for further object generation.

Almost every draft has following parameters:

- **Parent** - an object which will be parental for future generated elements
- **Pivot** - point in the space, where the element is going to be placed.
- **Up / Right / Forward** - vectors defining elements' orientation in the space. Axis forward isn't usually set up manually, it's calculation is based on two others axes - right и up.

Element Builders

Elements – basic parts, which are used to build the rest elements and in the end - finished buildings.

Element builders must implement ***IElementBuilder*** interface, which has a method ***Build***. It takes ***ElementDraft*** structure as an argument.

Usually there is no need in manual draft calculation; this is what column builders do.

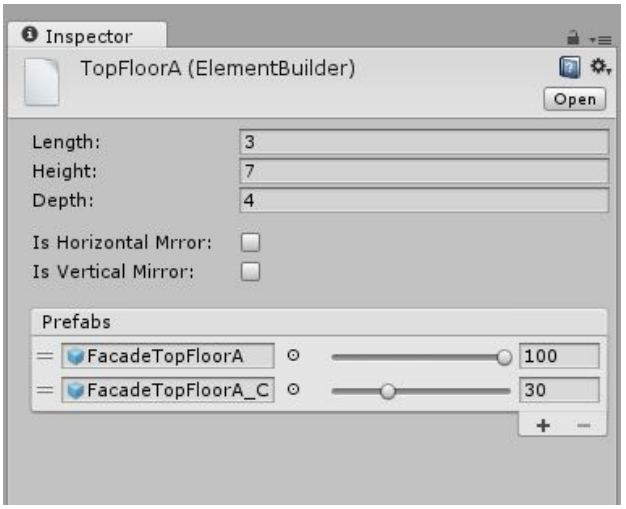
Elements draft parameters:

- **Length** – the size of an element along the axis “Right”
- **Height** – the size of an element along the axis “Up”.
- **IsHorizontalMrror** – should/shouldn’t an element be generated horizontally mirrored.
- **IsVerticalMirror** – should/shouldn’t an element be vertically mirrored.
- **ForwardScale** – axis “Forward” scale, it is used to calculate the scale along Z-axis and elements’ fitting during building process.
- **Parent** – parent object of a generated element
- **Pivot** – point in the space, where the element is going to be placed
- **Up / Right / Forward** – orientation axes

Standard Element Builder

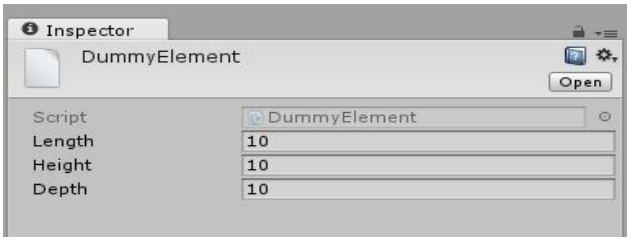
Standard Element Builder can be used to build common elements of the building.
It has size parameters, which define physical size of prefabs.
If you set up parameters ***Is Horizontal Mirror*** / ***Is Vertical Mirror***, this element will be generated mirrored by one or both of the axes. This allows to add a little variety.

The list of prefabs allows to set up several prefabs, and one of them will be chosen in random for generation. A chance, that the prefab is going to be chosen, is defined by its “weight”. The more is weight, the more are the chances for the prefab to be chosen.



Dummy Element Builder

Element builder, which however doesn’t build anything, but it can be used as a stub, that is quite useful in some situations.



Column Builders

Column – intermediate stage of building.

Usually column builders take a column draft as an input argument, make the calculations of elements amount, their scale and other parameters, form elements drafts and pass them to Element builders, which in their turn generate objects.

Column builders must implement ***IColumnBuilder*** interface, that uses ***Build*** method, which takes ***ColumnDraft*** structure as a parameter.

Usually there is no need in manual draft calculation, this is what Line builders do.

Columns draft parameters:

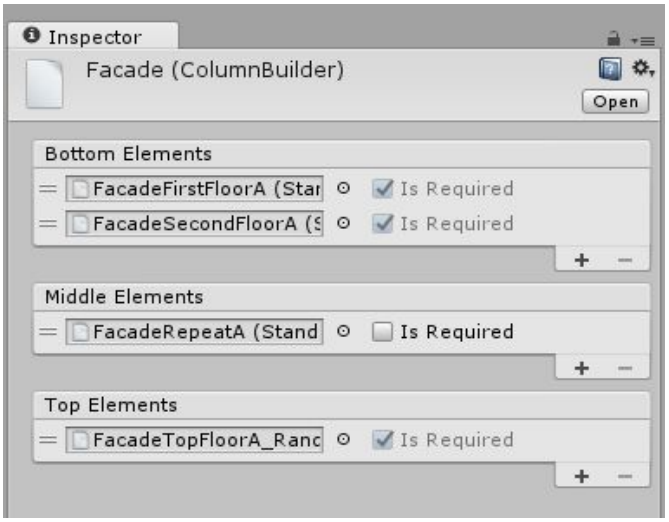
- **Length** – the size of an element along the axis “Right”. Usually a column stretches the elements by this parameter.
- **Height** – the size of an element along the axis “Up”. A column calculates the amount and the scale of elements, using the information about column height, and builds them one over another.
- **IsHorizontalMrror** – should/shouldn’t column elements be generated horizontally mirrored.
- **ForwardScale** – axis “Forward” scale.
- **Parent** – parent object of a generated element.
- **Pivot** – point in the space, where the element is going to be placed.
- **Up / Right / Forward** – orientation axes

Standard Column Builder

This builder has three lists, which consist of elements builders.

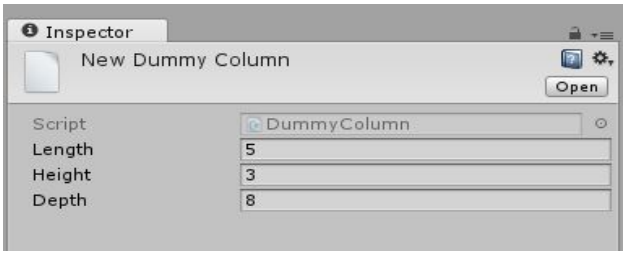
- **Bottom Elements** – elements, which are going to be generated in the bottom of the column and won't repeat.
- **Top Elements** – in the same way, but the elements are going to be generated in the top of the column.
- **Middle Elements** – elements, which are going to be generated in the middle part of the column and will repeat in cycle until the size of the column allows it. At the same time the column will take in a count the size of the elements and will try to scale them close to their real size.

If one of the column elements has a parameter ***IsRequired***, it will surely be generated at least once, it doesn't depend on the height of the column.



Dummy Column Builder

Fake column builder. Works in the same way as Dummy Element Builder.



Line Builders

Lines – objects, that form buildings walls.
Line builder uses line draft to calculate the amount and size of columns to form them in line.

Line builder should implement ***ILineBuilder*** interface, that has ***Build*** method, which takes ***LineDraft*** structure as a parameter.

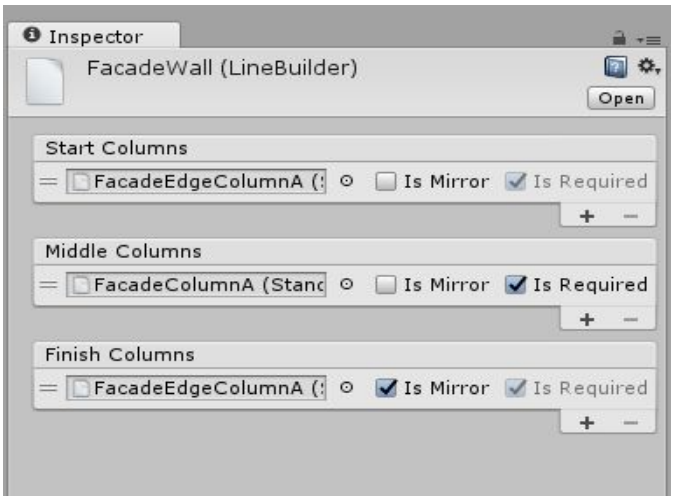
Usually there is no need in manual draft calculation, this is what box builders do.

Lines draft parameters:

- **Length** – the length of an element along the axis “Right”
- **Height** – the size of an element along the axis “Up”.
- **ForwardScale** – axis “Forward” scale
- **Parent** – parent object of a generated element
- **Pivot** – point in the space, where the element is going to be placed.
- **Up / Right / Forward** – orientation axes

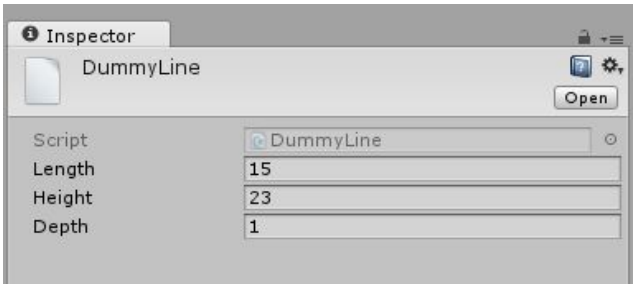
Standard Line Builder

This builder works in the same way as Standard Column Builder, but uses column builders.



Dummy Line Builder

Fake line builder



Box Builders

Box builder – the first builder, which defines building itself.

However it's not the only purpose of this builder. For example, roof builder can build only a roof.

Box builders work with lines, by forming them in walls, and they also can use other builders to form the complete building (e.g. roof builders).

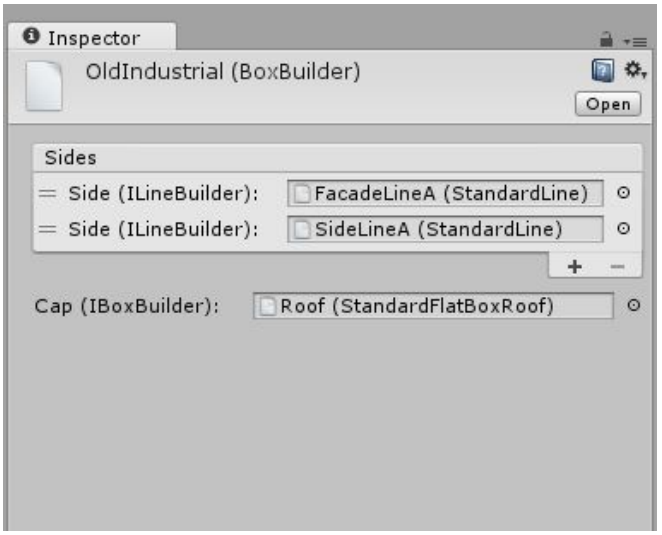
Box builders should implement ***IBoxBuilder*** interface, that has ***Build*** method, which takes ***IColumnDraft*** structure as a parameter.

Box draft parameters:

- **Length** – the length of an element along the axis "Right"
- **Height** – the size of an element along the axis "Up".
- **Depth** – the size of an element along the axis "Forward".
- **Parent** – parent object of a generated element
- **Pivot** – point in the space, where the element is going to be placed.
- **Up / Right / Forward** – orientation axes.

Standard Box Builder

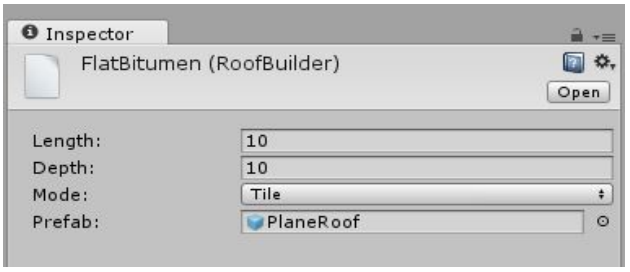
This builder has a list of lines, which it uses to build walls by choosing them one after another in cycle. It also has a link to a Roof Builder, which is used for adding a roof after building a line.



Roof Builder

It's a builder, which implements ***IBoxBuilder*** interface, that allows to add flat roofs to the buildings.

Mode – parameter defines how exactly prefab (or prefabs) will be generated to create a roof. In mode **"*Tile*"** builder is going to generate a few prefabs, trying to scale them for defined size. In mode **"*Scale*"** builder is going to generate only one roof prefab, which will be stretched to the draft size.



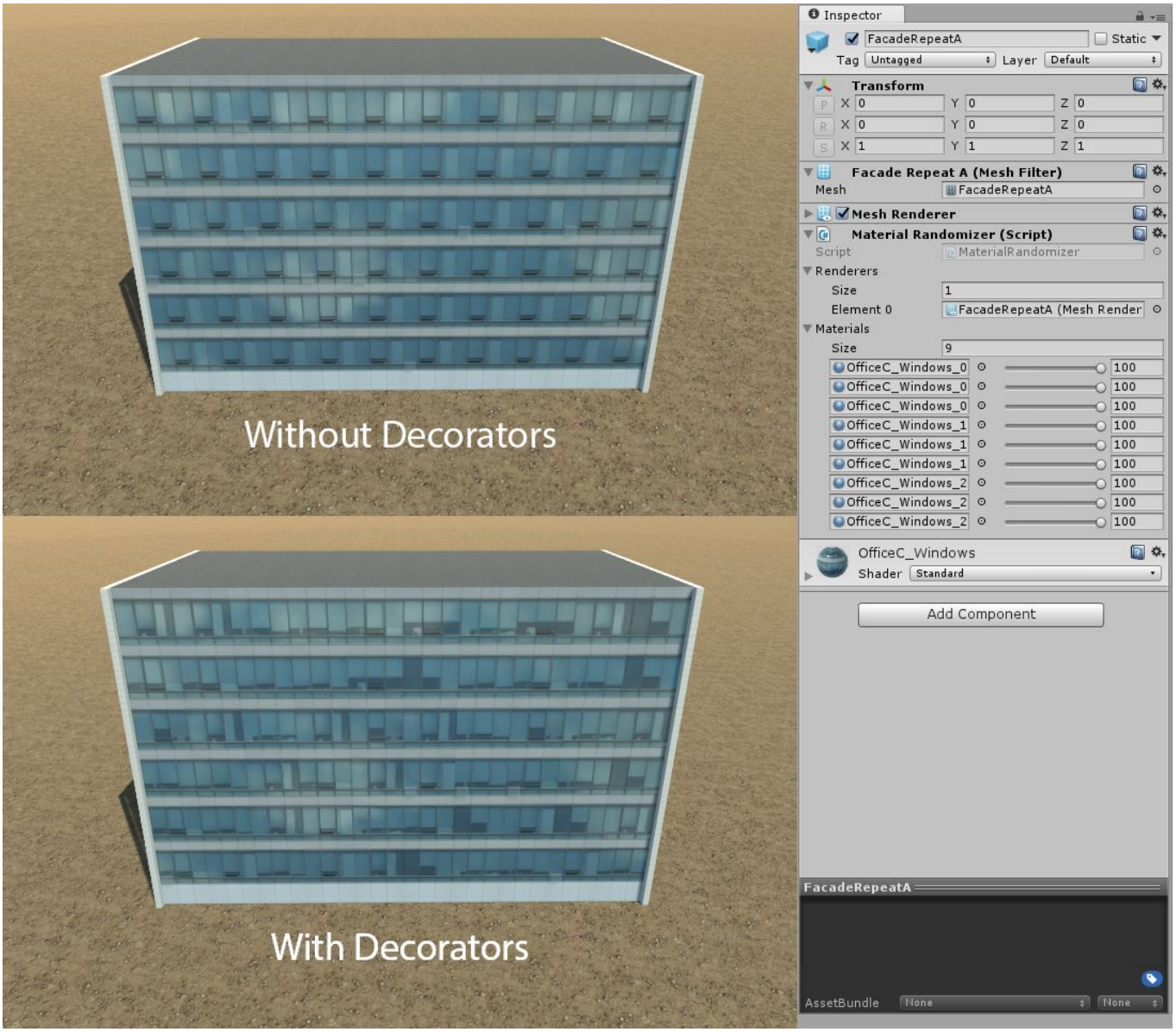
Decorators

Decorators – objects, which allow to add a variety in your generation result.

For decorators you can use MonoBehaviour Components, ScriptableObjects or just simple C# classes. There is only one condition – they should implement ***IDecorator*** interface with method ***Decorate***.

The easiest way to use decorators is to choose parent object of a generated building and send it as a parameter to ***Utils.RunDecorators*** static method. This method finds all the components, which implement ***IDecorator*** interface and restarts it one after another.

Picture illustrates decorator “MeshRandomizer”.
This is a component, which allows to set up a random material for your object.



Utilities

This asset has a few useful utilities to make your work much easier.

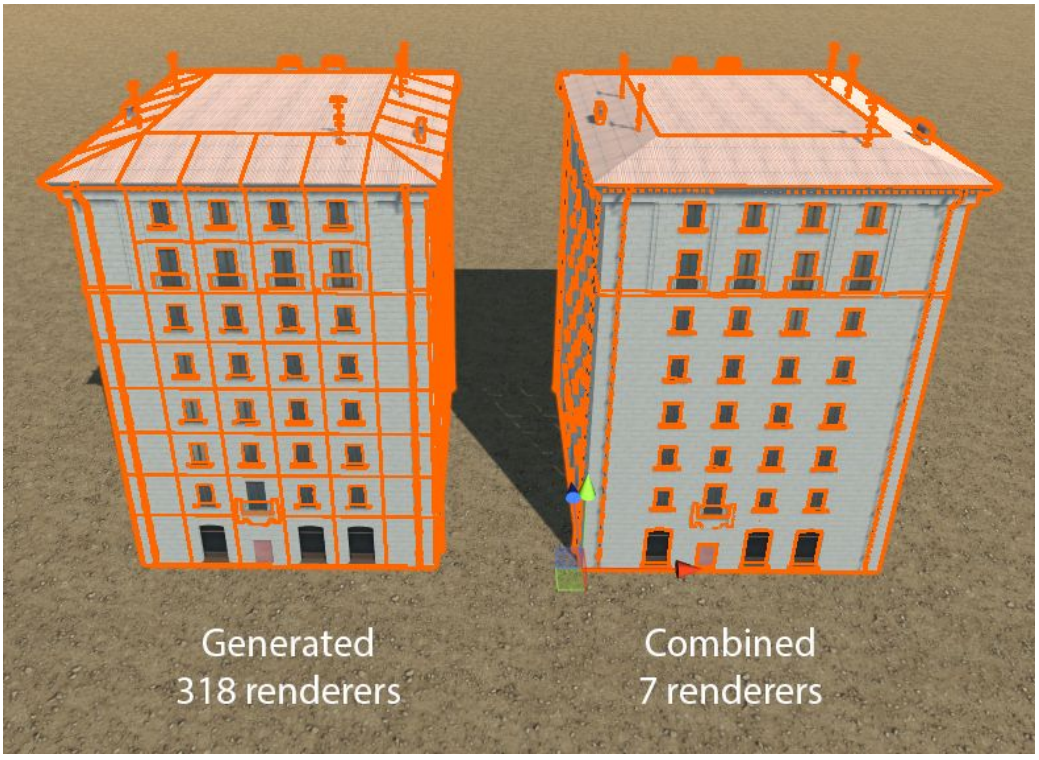
Meshes Combiner

An editor window, which allows to combine generated buildings in one asset and use it as a prefab.

Combiner joins meshes by material, which allows to reduce amount of objects, so the amount of Draw Calls is reduced and the efficiency is higher.

After combining a building in one object you can export it as an asset, that contains array of generated meshes, which can be used for further work.

Watch the video with example of usage: https://youtu.be/kwxEn0hb9UI



Additional and Contact Info

Unity Asset Store link: http://u3d.as/1fBi
WebGL Demo: http://alt3d.ru/stuff/modular-buildings-framework/web/
Trello Roadmap Board: https://trello.com/b/TPY61Asq/modular-buildings-frameworks

Youtube videos:
Preview: https://youtu.be/gFlozTCmSCw
Manual: https://youtu.be/2H5Tzj62n2U
Example of usage: https://youtu.be/9Rke5CUoOfc
Mesh Combiner Utility: https://youtu.be/kwxEn0hb9UI

Email: alt3d@yandex.ru
Web: alt3d.ru

Feel free to contact me with any inquiries.
Thanks for buying this package.