Simple House Maker

DOCUMENTATION

(Made by Laszlo Timar)

SHORT USAGE AND ADVICES

There are two major parts of this package, the house and the houserow. The first one is the simplest house imaginable, just a rectangle as base. The houserow makes it a bit more interesting, this concatenates these houses into a row, by the given parameters.

The simplest use of this, if you instantiate the premade prefabs, and tweak the settings a little then press the UPDATE button. This is the first step to understanding the parameters, however I recommend that you should read this documentation, to fully understand the working of this simple house generation.

With the additional system you can easily make windows, doors and other objects placed on the house. The windows are a bit tricky: since I recommend this tool to make building that mostly appear in the background (or even in foreground), but it's first purpose is not making houses enterable for players (however that is also possible, just not so easy), so the windows are just masks, you can see through, but if you see in, you can not see out or if you see out, you can not see in (the difference is: maskscript is on outerwalls or on innerwalls, in the prefab I made this is on the outerwalls). This solution is easy and enough for most cases. Thinking about the cases, where this wouldn't be enough I implemented the „cut" parameters, that will cut the selected mesh after placing it on the house (this requires a cutmesh component on the window prefab). The problem is, that this mesh boolean operation is really hard and complicated, this will cause an error after too much calls. If you really want to use that I recommed placing houses on top of each other, thus making more outerwalls and innerwalls objects, and separating these operations on different meshes.

I would also like to recommend using some meshcombining methods for the additionals, since they can easily result hard work for the CPU. I did not make this, because the optimization highly depends on the usecase. (And this would be one of the first thing I would improve in the future…)

Thank you for purchasing this asset.

(And if something feels wrong, or if you find a bug or just can't figure out everything about this asset, feel free to contact me.)

laszlo.tim0112@gmail.com

Discord name: Timar Laci#2749

USING THE SCRIPT FOR RUNTIME GENERATION

This asset generates the houses runtime (the editor version is just a preview), this is why it is also working for runtime creation. You can create new houses, adjust the parameters after you have hit the play button.

Since it wouldn't be effective to update repeatedly or check if some parameters have changed and then update the whole house, after everything is assaigned, you have to call an updating function to see the changes. This is similar to the Update button in the inspector, you have to call it yourself this time (in runtime).

So if you want to update it from script after you assaigned the parameters:

- in the houserow, call:

  houserow.CoroutineCaller()

- in the house, call:

  for the meshes: house.UpdateParts(),

  for the addition: house.Addition()

- in the houseBridge, call:

  for the meshes: houseBridge.UpdateParts,

  for the addition: houseBridge.Addition

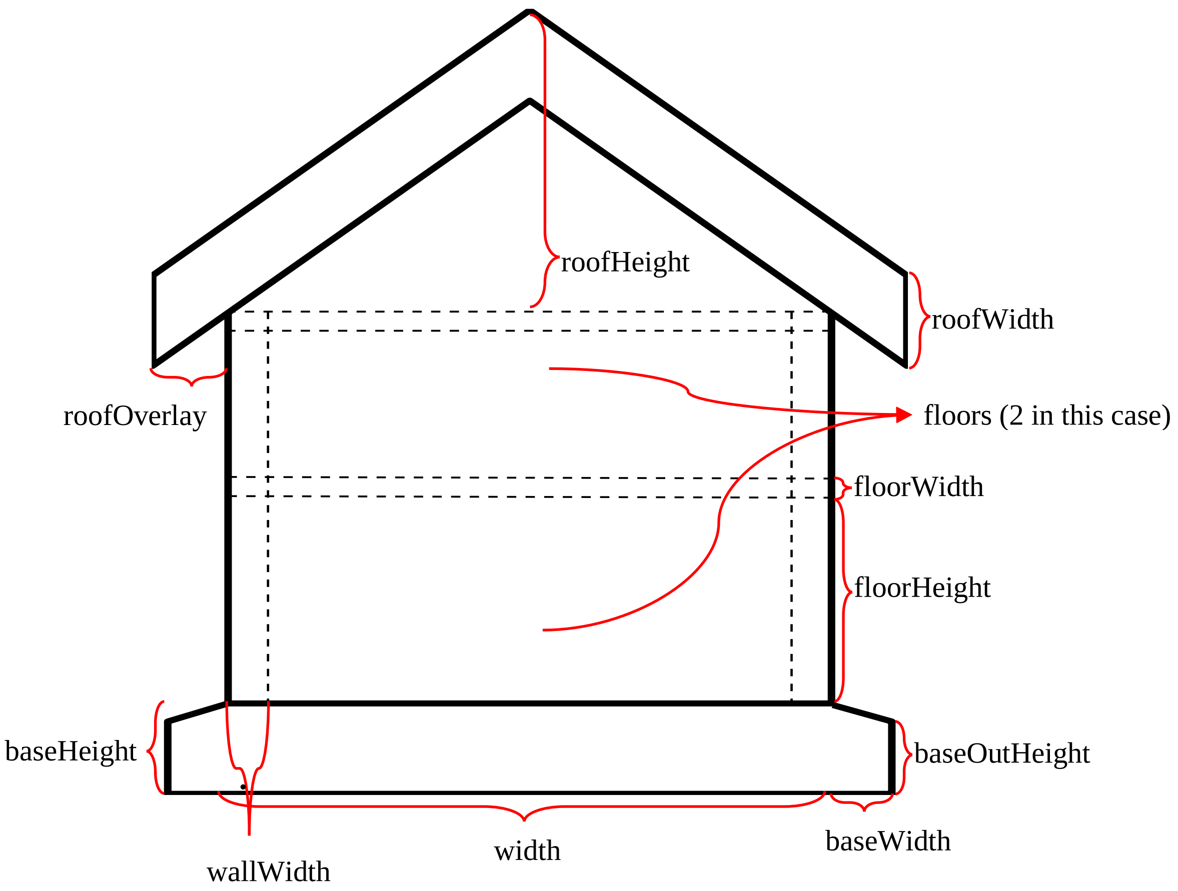
The houserow updating is a bit problematic, this is why a coroutine gets called. Every other option I tried, was not trustworthy enough.

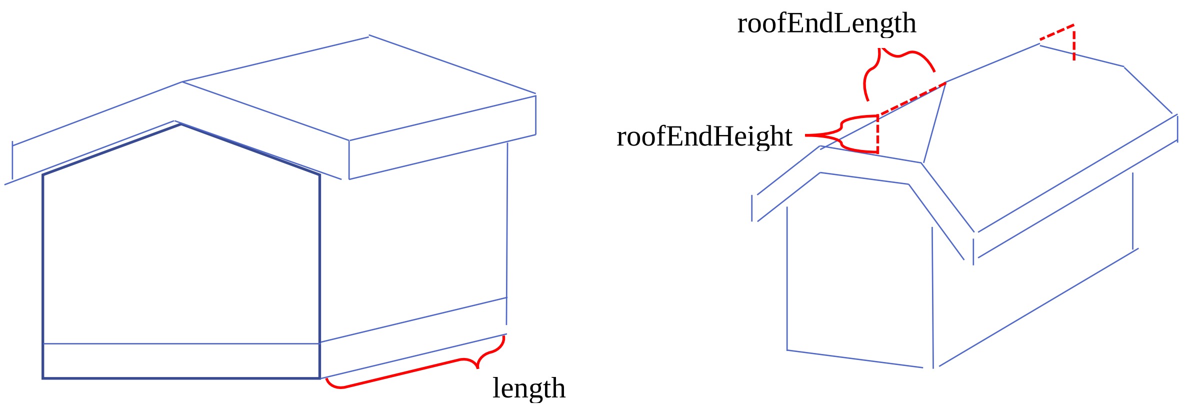
NAMING

Some variables are named not so catchy, which may confuse the user. To clarify every questionable name in use, here is a small explanation of all the names used. (Except the ones in the Draw() functions, these are just mathematical expressions, used for calculating the coordinates.)

bridge - when you put two or more houses in a row, a bridge fills the gap in the given angle, the name is from "bridging the houses"

Names in the house system:





Names in the additionals system:

The prefab that will be placed in the house by the given parameters.

if you want to place object on the roof, enable isRoof (owerwrites everyFloor)

if isRoof enabled and follow roof angle is enabled, objects will have a default angle in the roof

places the object on every floor if enabled

this will set the objects in the middle of each side (useful in windows)

minimum and maximum distance between two object (random between the two value)

randomly deletes objects, given by the percentage

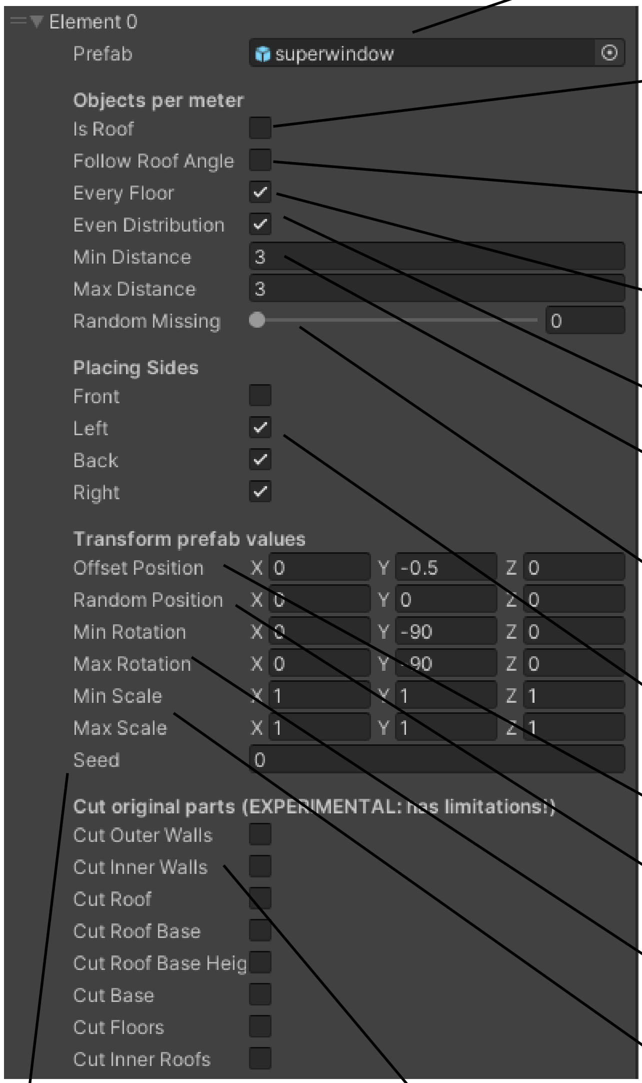
placing objects on the enabled sides

offset the prefabs position

adding a random position (this is the maximum vector)

minimum and maximum rotation of the prefabs

minimum and maximum scale of the prefabs



Element 0

Prefab	superwindow		
Objects per meter			
Is Roof	☐		
Follow Roof Angle	☐		
Every Floor	✓		
Even Distribution	✓		
Min Distance	3		
Max Distance	3		
Random Missing	0		
Placing Sides			
Front	☐		
Left	✓		
Back	✓		
Right	✓		
Transform prefab values			
Offset Position	X 0	Y -0.5	Z 0
Random Position	X 0	Y 0	Z 0
Min Rotation	X 0	Y -90	Z 0
Max Rotation	X 0	Y -90	Z 0
Min Scale	X 1	Y 1	Z 1
Max Scale	X 1	Y 1	Z 1
Seed	0		
Cut original parts (EXPERIMENTAL: has limitations!)			
Cut Outer Walls	☐		
Cut Inner Walls	☐		
Cut Roof	☐		
Cut Roof Base	☐		
Cut Roof Base Heig	☐		
Cut Base	☐		
Cut Floors	☐		
Cut Inner Roofs	☐		

the seed used in the randomly generated values

when enabled, this will cut the corresponding part of the house (modifying the mesh)(the prefab requires a cutout component)

SCRIPTS

The „house.cs” script (and “bridge.cs”)

A house has multiple parameters. This script holds the parameters used for generation. The children of this GameObject (with house component) can access the parameters, and then they generate the mesh.

The script also handles the additionals-system, which means you can add GameObjects to the house (windows, antennas, air conditioners…).

The „MeshRecognizer.cs” script

This script is used in the house objects, and only purpose is to update the collider by making the MeshCollider recognize the generated mesh.

When a new mesh is generated, the collider usually won't update. This problem can be solved by reassigning the mesh in the Meshcollider component. That is what the MeshRecognizer does.

Since this only occurs when the mesh is updated and a MeshCollider component is on this GameObject, this normally shouldn’t update. So this reassigning only happens in the start frame.

If you, for some reason, would want to update the collider, open the MeshRecognizer.cs script, there will be seen options for the update, but the recommended is, that the MeshRecognizer.Recognize() function should be called only when it is needed, from another script.

The „HoleCutter.cs” script

This is only an example use of CSG, which is able to make boolean operations on meshes.

This example subtracts the given objects from the base object.

The „stairs.cs” script

Stairs can be a bit tricky, so there is a little script for that. You need a GameObject as a stairstep (this will be instantiated), you also need a start and end point. Then you can also edit the prefabs scale, rotation and offset. You can update with checking the UPDATE field: this serves as a button. The start and end points shouldn’t be the children of the object with the stairs component: the stairs component deletes all of its children during updating.

The „TextureFlip.cs” script

Textures come sometimes with a wrong tiling scale or needs to be flipped with 90°. No need to mess with the material, or make different versions of it, the problem can be solved with the

TextureFlip component. This also updates in playtime, but only if some of the input values have changed.

The “cutout.cs” script

If you have windows, you want to see through the window (most of the time). But with the basic geometry of the houses, it is not default, that when you place a window, you cut out the given wall area. So we have to work with a CSG mesh-boolean operation, called subtraction. You should place a cube (or any other mesh) on your window, that doesn’t even have to render, and assaign this mesh to the cutmesh property. When this window appears in one of the houses' additional list, it will know, that the wall should be cut by the windows (with the window’s cutmesh variable).

The “MeshCombine.cs” script

This script is not directly for the house making, I just came across a problem, and first tried to reduce the calculation (which is always a good thing), the script is still in use.

This script combines the meshes of the given GameObjects, thus reducing the draw calls made by the CPU.

The „CombineAnyMesh.cs” script

This meant to combine meshes just like the MeshCombine script, but this can handle multiple materials.

The „ColorRandomizer.cs” script

This will randomly overwrite the color of the material given by the colors array or the gradient. This will only happen in the start of the game.

The house-parts generating scripts: “baseCube.cs”, “floors.cs”, “innerRoofs.cs”, “innerWalls.cs”, „outerwalls.cs”, “roof.cs”, “roofBase.cs”, “roofBaseHeight.cs”

Each script generates the corresponding part of the house:

baseCube.cs	base of the house (cube-like under the whole house)
floors.cs	planes in the house, one in each level
innerRoofs.cs	planes in the house, one in each level (ceiling)
innerWalls.cs	walls, visible from the inside

outerwalls.cs	walls, visible from the outside
roof.cs	part of the roof, best visible from top view (top part)
roofBase.cs	part of the roof, best visible from under the house (bottom part)
roofBaseHeight.cs	part of the roof, best visible from side view (side part)

The bridge-parts generating scripts: “baseCube.cs”, “floors.cs”, “innerRoofs.cs”, “innerWalls.cs”, “outerwalls.cs”, “roof.cs”, “roofBase.cs”, “roofBaseHeight.cs”

Each script generates the corresponding part of the bridge (very similar to the house):

bridgeBase.cs	base of the bridge (cube-like under the whole house)
bridgeFloor.cs	planes in the bridge, one in each level
bridgeInnerRoofs.cs	planes in the bridge, one in each level (ceiling)
bridgeInner.cs	walls, visible from the inside
bridgeOuter.cs	walls, visible from the outside
bridgeRoof.cs	part of the roof, best visible from top view (top part)
bridgeRoofBase.cs	part of the roof, best visible from under the house (bottom part)
bridgeRoofBaseHeight.cs	part of the roof, best visible from side view (side part)

All scripts of the above contain a Draw() function, which is responsible for generating the coordinates of the corresponding part and calling the UpdateMesh() function, thus updating the Mesh itself.

The CSG scripts

These scripts are responsible for mesh boolean operations: Union, Subtraction, Intersection. the cutout script uses the Subtract() method, but this only works with small call-count on one mesh – at least in my experience. This is also in the HoleCutter script. Use it carefully, it can cause a stack overflow!

PREFABS AND COMPONENTS

Since I made just a few house prefabs I feel I must explain all the given components and why are they in the GameObjects.

The House PREFAB:

First of all you need an empty GameObject with the **house component**: this will call the Draw functions of the children. You should then create children with the wanted **components**: **outerwalls**, **innerwalls…** So each child should have now one housegeneration script (like outerwalls), a **MeshFilter** and a **MeshRenderer**.

The **MeshCollider**: if you will have collisions with the houses, then you should also add MeshCollider components to the houseparts. And since the generation will trick the MeshCollider, you also have to add in this case the MeshRecognizer component.

Then you should assign these objects in the parent’s house component.

And this is it.

The little tunning I did, is that I added a **ColorRandomizer** to the roof and outerwalls objects, that will randomize the colors in the given scale of the roof and the outerwalls, when the game starts. This will make the houses a bit more unique.

Then the additionals:

At this point there is no window on the house and most houses have windows in real life…

You can add windows, doors and anything else you want on the house with the additionals system (this is at the bottom of the house component). You can easily find out the parameters, but the tricky part is when you want to see in or out of a wall.

There are 3 options:

- You make a window with a not transparent glass material.
- You add a GameObject to your window prefab with the maskmaterial, and attach the maskscript component of the part of the house, which should transparent in some places.
  (This is easy and fast, but you can only use it with a few parts: this will render the windows maskmaterial first, which is transparent, then NOT render the GameObjects with the material of the GameObject with the maskscript. And yes, this can cause weird looking things)
- You can add a cutout component to your window prefab and then tick to cut some parts of the house in the additionals part. But this should be not used, when too many cuts should be made. When this really needed I recommend to slice up first the original housepart and then cut it. (Then the cuts will be called on different meshes. THIS IS NOT DEFAULT, SO THIS TRICK IS NOT IN MY SCRIPTS (maybe in the future…))

After all of that if you placed a lot of additionals, you should combine them or use some other optimization technique, that your project excepts. (THIS IS ALSO PARTLY MADE, BECAUSE THIS HIGHLY DEPENDS ON THE USE-CASE!)

The HouseRow PREFAB:

This is a bit easier to use:

You don't really need a prefab for this: this is working with just an empty GameObject with the houserow component.

You need to assaign the cluster part: this will determine the chain of houses (and bridges).

Don't forget to add the house and bridge prefabs under the cluster: these will be instantiated. Also if you have special houses to place you can overwrite these default prefabs by assaigning the special prefabs in the cluster part.

And give the values of the other variables (the general variables/parameters which will be se ton each element of the row).