

Grid Building System

Overview.....	2
Asset setup.....	3
Package dependencies:.....	3
Built-in, URP & HDRP.....	4
Grid Placement Layer.....	4
Demo Scene & Controls.....	6
Predesigned Level.....	6
Start with empty level.....	6
Input Control.....	7
User Interface.....	7
Item Shop.....	8
GridType based removal.....	8
Action Tooltip.....	9
Asset Configuration.....	10
Centralized Configurator.....	10
Grid configuration.....	11
Grid cell size restriction.....	12
Creating a grid scene from scratch.....	13
New scene setup.....	13
Create a placeables database.....	14
Create Placeables.....	15
Placeable properties.....	17
Prepare your 3D models.....	19
Requirements.....	19
Problem: Wrong pivot.....	20
Solution: Center pivot of your 3D model.....	20
Shaders.....	21
SF Grid.....	21
SF Preview.....	22
Deep dive.....	23
Extending Placeable properties.....	23
Layer based removing.....	25
Placement System.....	26
Placement Handler.....	27
Placeable Object → Placed Object.....	28
Save System.....	28
Game Manager.....	28

Overview

The Grid Building System is a powerful and highly customizable grid placement tool, ideal for creating city builders, farm simulators, and other grid based style games.

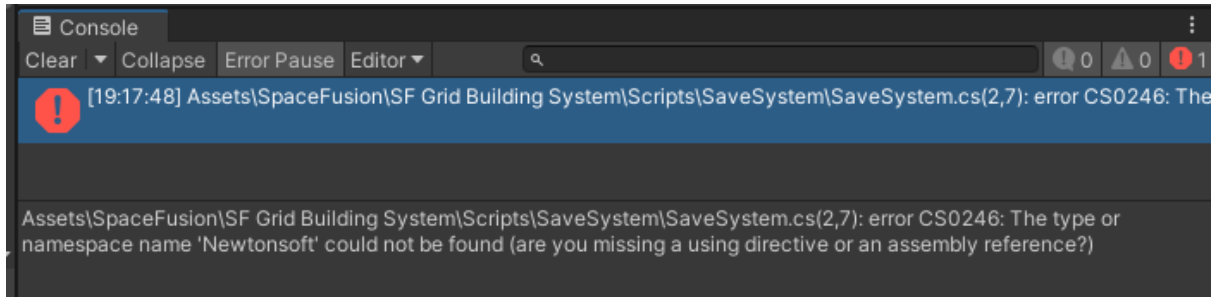
It features adjustable grid sizes with custom cell visualizations, multi-layered GridData for advanced placement mechanics, and building rotation support. An integrated tab-based Shop UI allows for intuitive object selection, while the smart Remove UI streamlines object deletion by filtering specific grid types. With the range of smart tools, this asset makes the development workflow fun and easy.

Asset setup

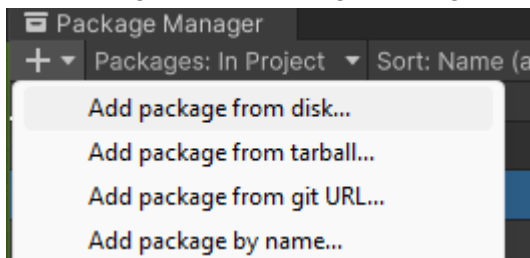
Package dependencies:

This asset uses the newtonsoft json library for the save system.

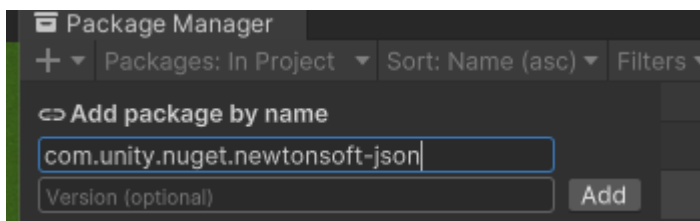
After importing the asset you may get following error:



To fix this go to the package manager and click on the + icon on the top left corner:



Click on Add package by name and add “com.unity.nuget.newtonsoft-json”

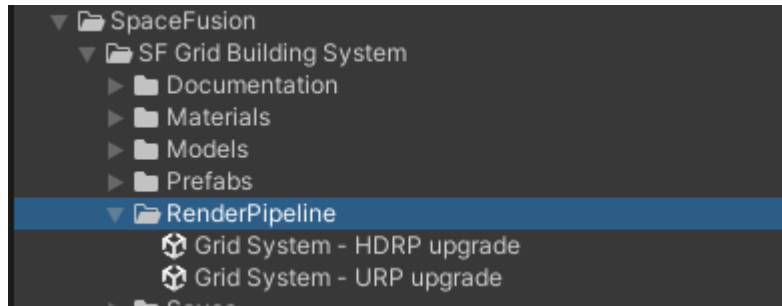


Unity will then recompile and the error should be gone.

Built-in, URP & HDRP

This asset was developed in the built-in render pipeline so it should work out of the box if you are using the same pipeline.

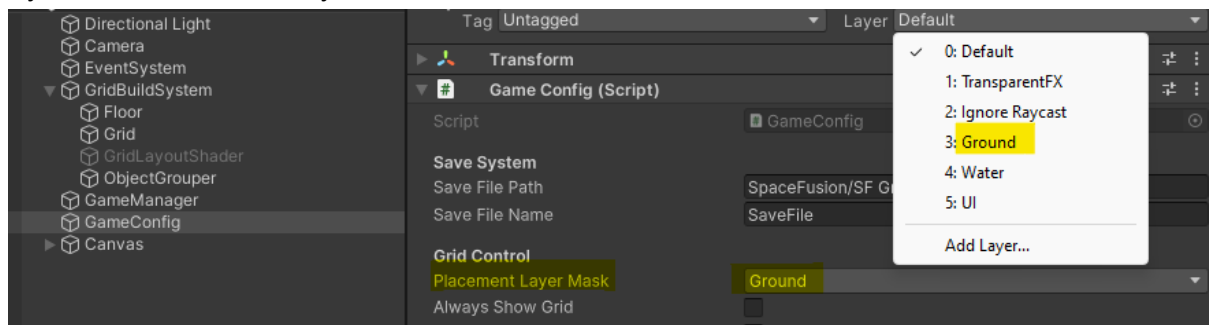
For URP & HDRP simply navigate to the included RenderPipeline folder



Then double click on the upgrade for your render pipeline and import the upgraded assets. This will overwrite the materials and the shaders according to your used render pipeline.

Grid Placement Layer

Please locate the GameConfig in the scene, create a new Layer and assign the placement layer mask to the new layer.

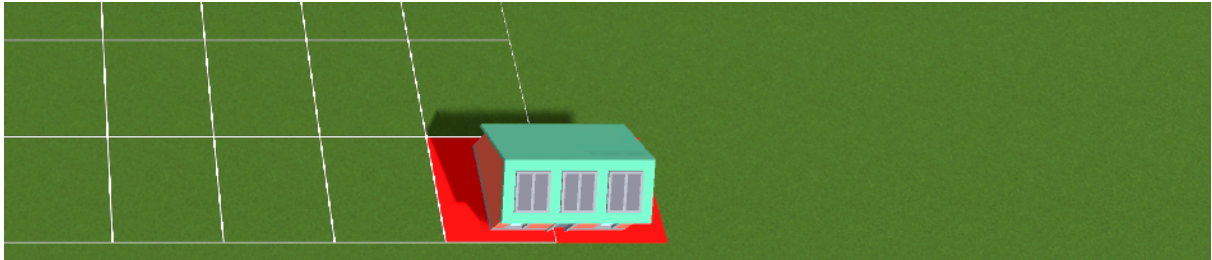


This will automatically assign the layer to the needed objects on game start.

That's basically all you need to do configure in order to start your demo scene.

You can find some detailed description about the layer based system and an option on how to extend the input handling beyond the grid boundaries on the next page.

The grid placement system operates on a **separate layer** to give the user a visual feedback of where it is allowed to place objects. This means if the mouse is moving on some other object that is not on the same layer (like the grass floor), the input system won't recognize its movement and will not update the preview until you move back to the allowed grid layer.



When you move back to the grid with the mouse, the preview will be updated again.

If you do not want the previously described behaviour you can also directly assign your newly created layer to the floor object so the placement system will recognize mouse movements across the whole floor (not only the defined grid). This would look something like that



With this you will achieve that the preview is updated according to your mouse.

So you can drag the objects outside of the grid, but obviously you will not be able to place it since it is not anymore within the grid size boundary.

Demo Scene & Controls

In this section we will explore the included Demo Scene, the input controls of the scene and the included UI components like the shop, remove options and action tooltips to give you an easy start into this asset.

Predesigned Level

This Asset includes SaveFile with a predesigned demo level. It will be loaded by default when you start the *Grid Demo* scene.

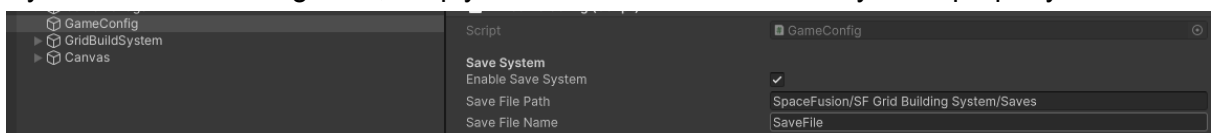
It should look something like this:



Start with empty level

If you want to start completely from scratch, simply locate the GameConfig in the scene and choose a new save file name.

If you don't want saving at all, simply uncheck the "Enable Save System" property.



Input Control

The demo scene supports following mouse and keyboard controls:

General	
Drag Camera	Hold the mouse scroll wheel and move the mouse allows you to move your camera
Rotate Camera	Holding the right mouse button and then moving the mouse will allow you to rotate the camera
Zoom	Simply scroll with your mouse wheel and the camera will zoom in or out based on your mouse position
Camera auto movement	This setting can be adapted in the global game configuration : Moves the camera in mouse direction when the mouse reaches near the screen corners. Per default only activated in placement/removal state
Placement / Removal State	
Rotate Objects	Press R to rotate the previewed object before placing it on the grid
Place Objects	Placement state: Left mouse button to place the object on the selected grid cell
Remove Objects	Removal State: Left mouse button to remove the selected object
Placed Object	
Trigger Action Tooltip	Press and hold the left mouse button on an object to trigger the Action tooltip. This allows you to move or remove the selected object

User Interface

Item Shop

On the bottom of the screen you can see a simple shop that allows you to place different kinds of objects. You have different tabs based on object groups:

Buildings, Decorations, Nature & Terrains.



Clicking on an object icon will trigger the placement state and will allow you to choose where to place the selected object. You hold and move the left mouse button to scroll through the entire list of objects.

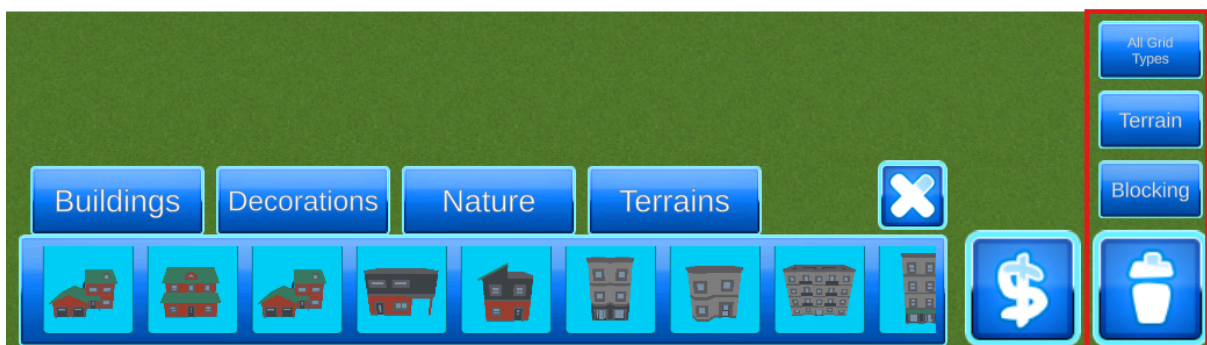
Clicking on the Dollar sign will show or hide the shop panel.

GridType based removal

This removal option uses the underlying GridType and GridData classes which will be explained later in the documentation. For now it's enough to know that objects of the same GridType are stored in the same GridData. This allows you to store e.g. terrains and buildings on the same grid cell.

Clicking on the bin icon will open some removal options for you. You can choose to remove only from specific GridTypes (like only the terrain data, only the blocking data like buildings and decorations, or directly choose to remove objects from all grid data).

Choosing a specific option will hide all objects that are not of this type and you can focus only on the necessary items. Choosing the all grid types option will allow you to instantly remove all data on a selected position. This means you can directly remove the terrain and a building that is on the same position just with one click.



Action Tooltip

Clicking the left mouse button on an object and holding for a short time will trigger an action tooltip for the selected object. You can decide to remove the object or to move it to another direction.

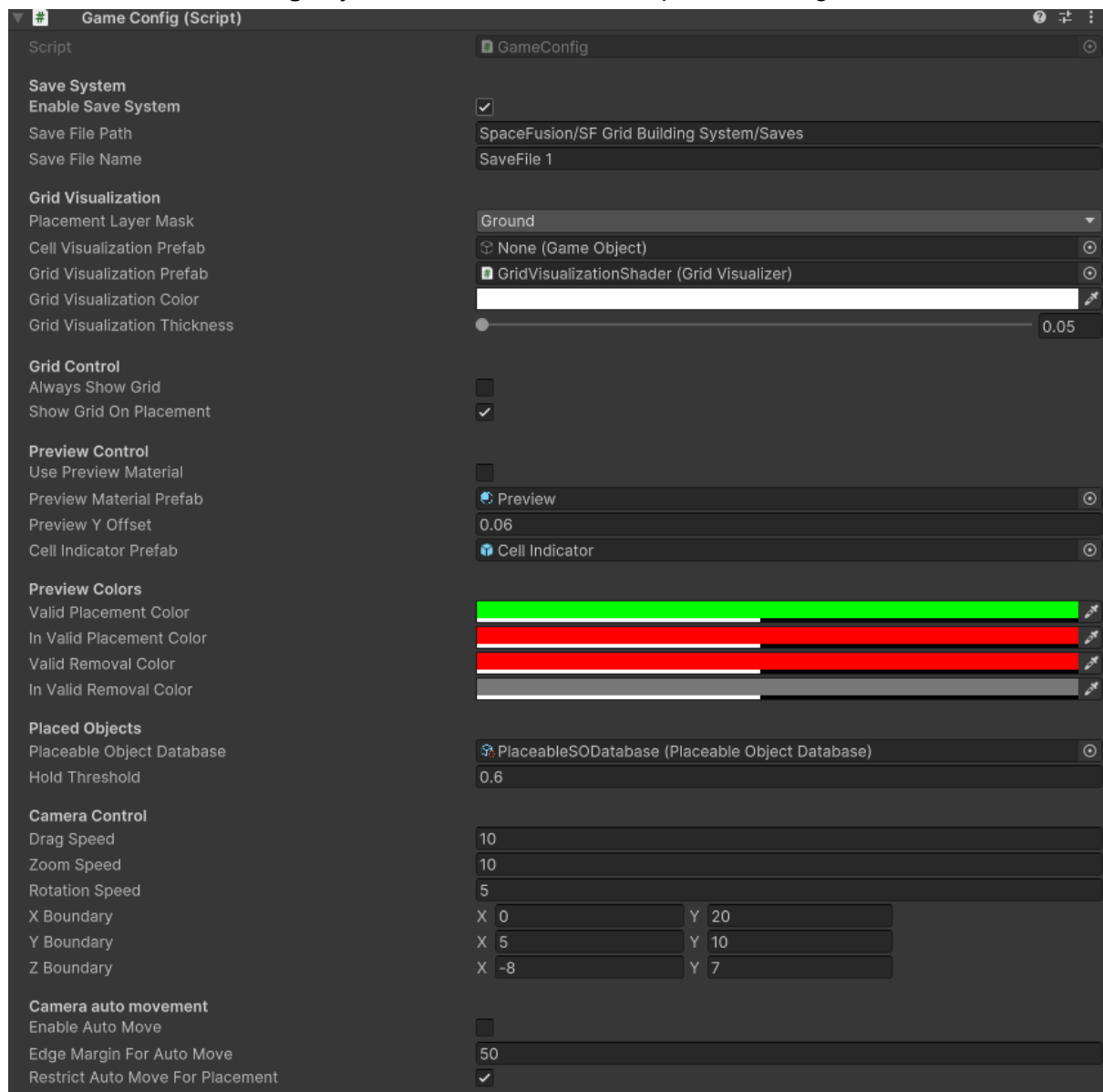


Asset Configuration

This section will show you where you can find the configurations to customize the asset for your needs

Centralized Configurator

Configure nearly everything you need for your gameplay in just one place. Simply locate and click on the **GameConfig** object in the demo scene to open the configuration



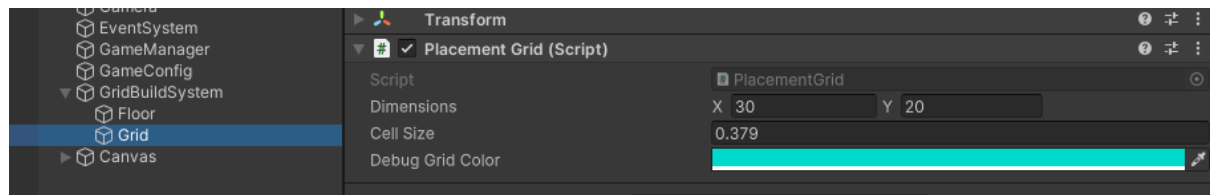
Tip: Hover over a property and a tooltip will be shown with a more detailed explanation for the selected property.

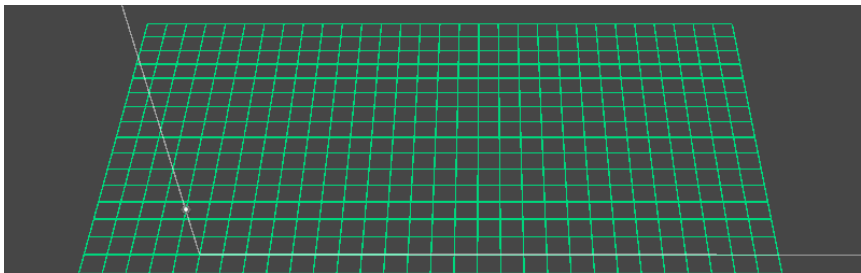
Grid configuration

The only thing that you can not configure in the global configurator are the grid dimensions, the grid cell size and the debug color. This is to keep the grid configuration flexible, in case you want to create multi grid systems with different dimensions and cell sizes.

Note: This asset currently does not support multi grid management

Locate the Grid GameObject under GridBuildSystem → Grid and specify your grid size.



Dimensions	Define the amount of cells in x and y direction. Y is actually the z direction in unity world space, since unity uses the Y for the Upwards vector
Cell Size	Define the size of each individual cell
Debug Color	<div>Specify a grid debug color for the scene view. If you don't want the grid to be shown in scene view, simply set the color alpha to zero.</div> <div></div>

Grid cell size restriction

Do not change the grid cell size again for scenes with existing saveFiles.

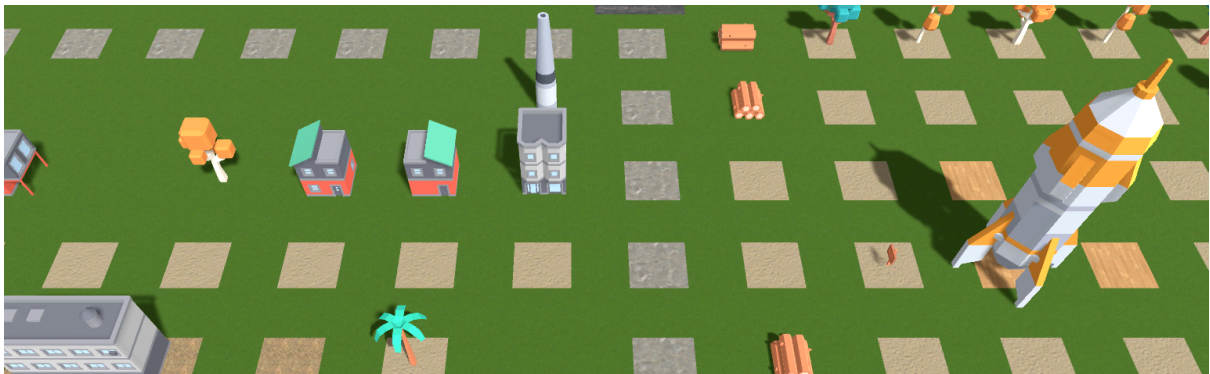
Feel free to test different cell sizes and explore what fits your use case the best, but once you have saved a scene with a specific cell size, do not change the size again. The reason is that the saved object positions and occupied cells are stored in the saveFile and are based on the cell size that you used when placing your objects. When loading the saveFile again there is no additional check that if the cell size is still the same as it was previously

Example of loading a game state created with cell size of 1 but using a new cell size of 0.33.



Actually all gameobjects are still on the same grid positions than previously, but since the cells are smaller, there will be a lot of overlapping.

A similar effect can be seen with a higher cell size (2)



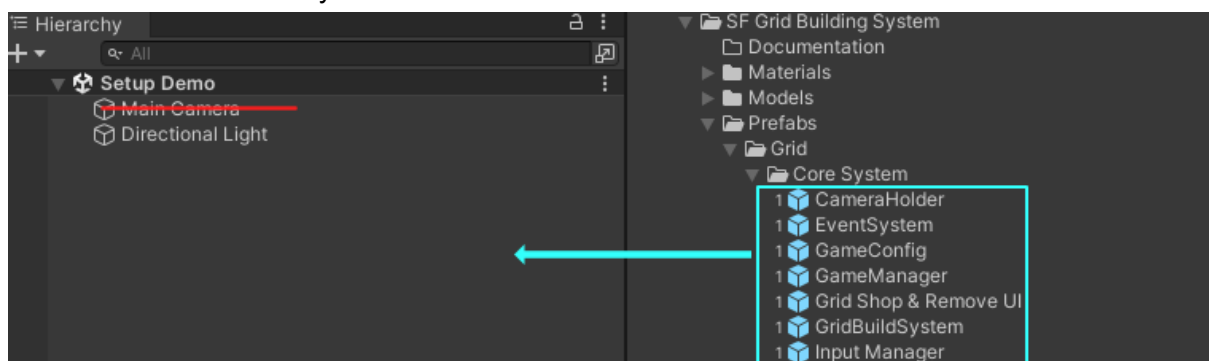
Still the grid positions are all the same, but the object now would occupy less cells than it was stored in the saveFile. A Rocket of size 2x2 only occupies 1 cell of size 2, but with cell size of 1 it would occupy 4 cells.

Creating a grid scene from scratch

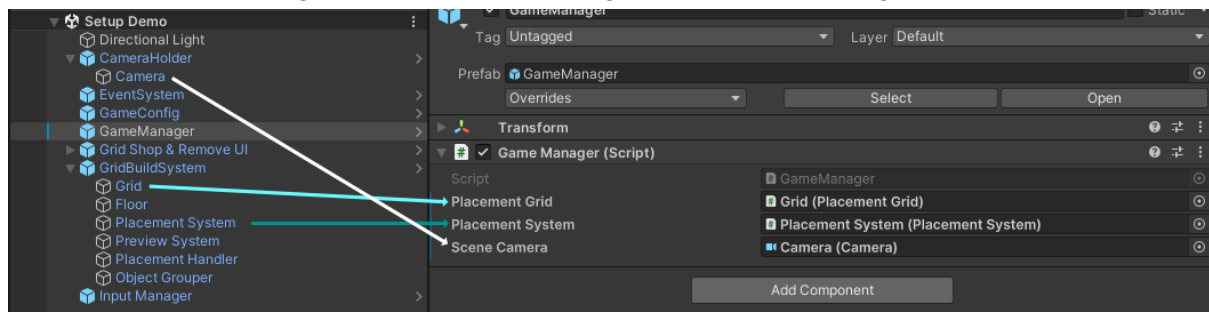
This section will guide you through a step by step instruction on creating a fully functional grid scene from scratch with your own 3D models. You will learn how to get from a simple 3D model imported in Unity to a fully functioning placeable object that you can use for your grid. Each section has a detailed explanation on how to configure everything and why the stuff needs to be done in this way.

New scene setup

As the first step, let's create a new empty scene in your project. Name the scene however you want. Delete the main camera from the new scene and add all prefabs that are under Prefabs→Grid→ Core System to the scene



Select the GameManager and fill in the missing properties according to the screenshot:



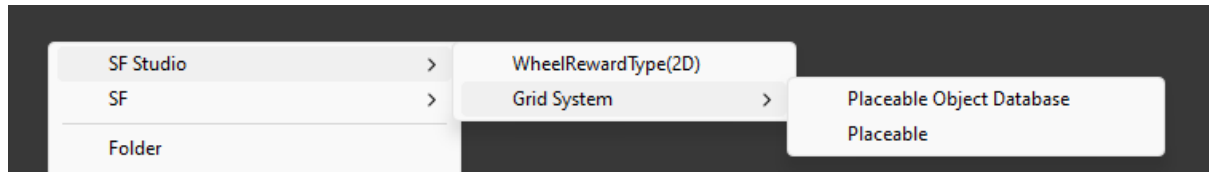
This should now allow you to start the scene without any errors, using the predefined placeable object database.

The GameConfig is pre configured to use the included PlaceableSODatabase. However if you want to use your own models for the grid system, you need to prepare your models as a placeable object and add them to your custom database.

In the next section you will learn how to replace the database with your own empty database. Then we will learn how to actually create placeables and assign them to your database, so you can actually use them in your game.

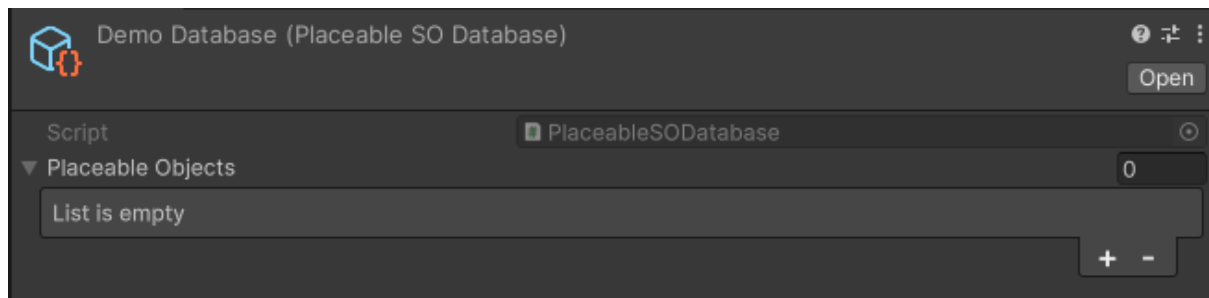
Create a placeables database

The database itself is a simple scriptable object that stores a list of placeable objects. Simply right click somewhere in the project folder and locate “SF Studio → Grid System → Placeable Object Database”

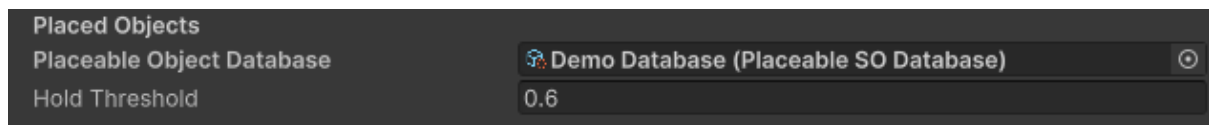


This will create an empty database for you.

It should look something like this:



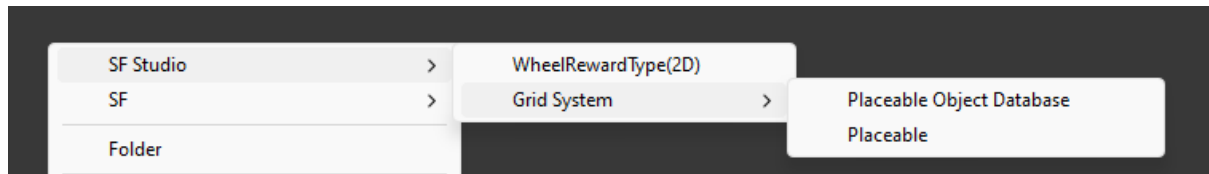
Go to the GameConfig in your scene and switch the existing database with your newly created one:



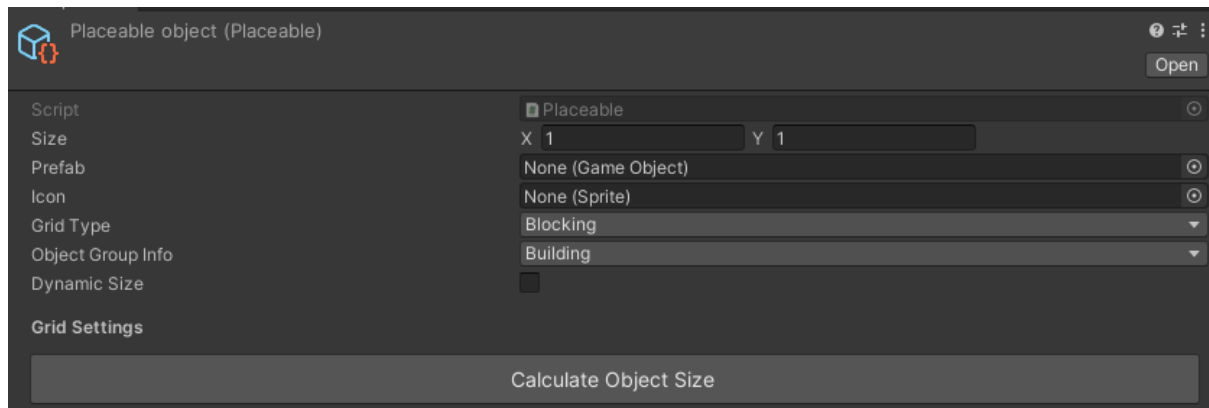
Now that we are using our new database we need to create some placeable objects and store them in the database, so that we can actually place them on the grid.

Create Placeables

Create a new placeable in the same way that you created the placeable database. Just choose “Placeable” now instead of “Placeable Object Database”



You will see following preview in the inspector:



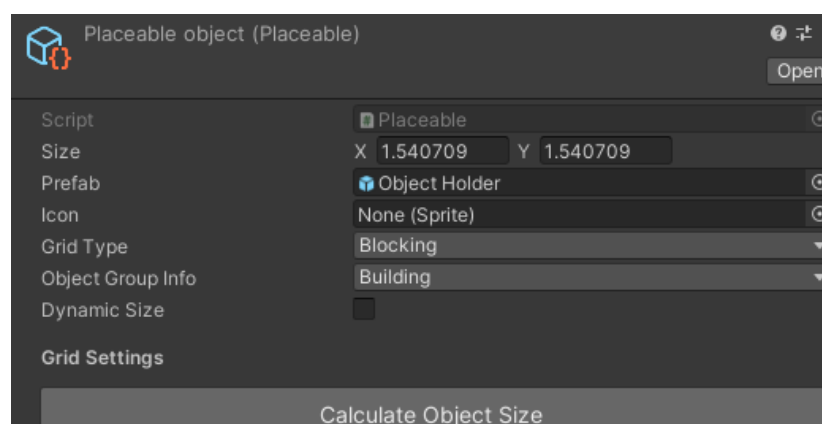
Use the prefab of a 3D Model of your choice and assign it to the Prefab holder.

There are some requirements for 3D models that you need to consider for a flawless experience:

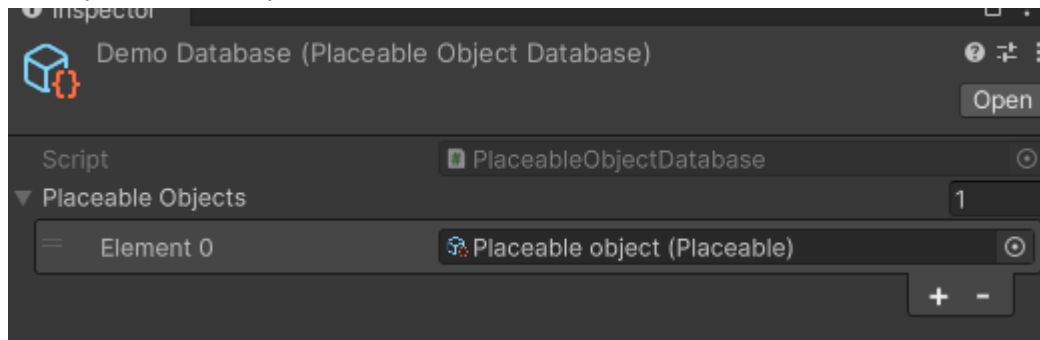
- The 3D model should have a collider assigned so we can interact with it after placing it on the grid.
- Also the pivot of the model should be centered in x and z axis to ensure smooth rotation on the grid

If the model already meets these requirements, you can just continue with the documentation, otherwise either first read the section “Prepare your 3D models” or simply continue with the documentation and fix the model later on.

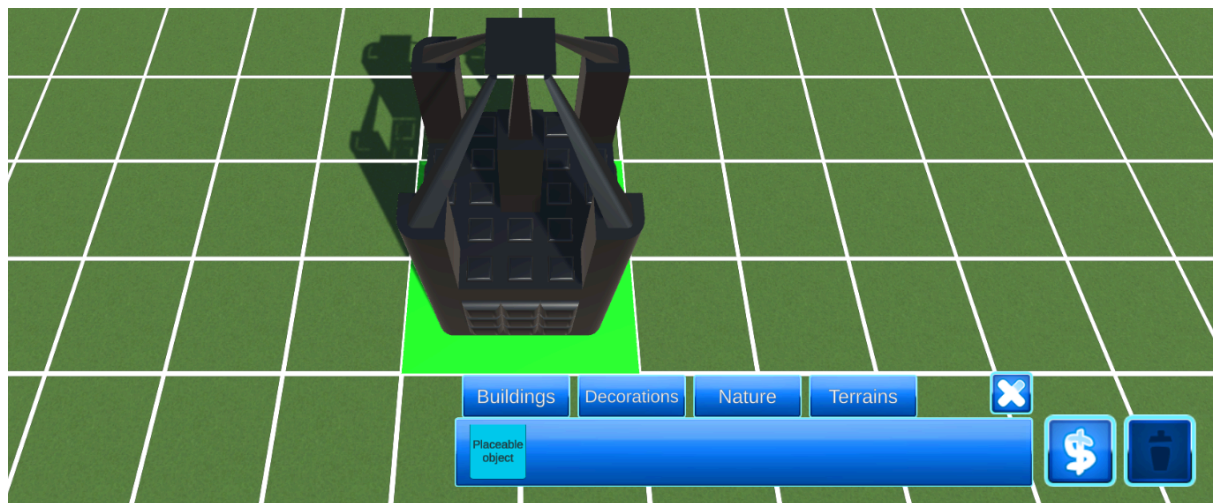
After adding your 3D model, click on *Calculate Object Size*, which will pre fill your Size property



Go to your previously created database and add this placeable to the database:



Now start the demo scene and try to place your object. The shop UI buttons should be prefilled automatically based on the placeables that are in your database:



Placeable properties

As you probably already noticed there are some other properties for the placeables that you can configure. You already know that you need to assign a prefab and then click on the calculate button to prefill the size property. Now let's go over the other properties.

Icon

The icon is used for instantiating the shop buttons with the item icons.

Simply create a sprite of an image of your object and assign it to this property.

If the property is empty, the shop ui buttons will fallback to the placeable name and display the name instead.

Grid Type

The grid type is used as an info to specify in which grid data the object position is stored after placement.

Detailed explanation

The placement system of the grid internally handles multiple GridData classes. A GridData represents an encapsulated state that holds the grid positions and the PlacementInfo for each position, but only from objects of the same gridType. This means that all objects of type "*Blocking*" will be stored in the same GridData and also the available positions will be calculated based on this grid data. The data does not know anything about other object types. This allows us to store one object of type blocking and one object of type terrain on the same grid position (since it's handled by different grid data classes). This is pretty handy, since we may want to have the option to place a building on top of a terrain.

Note that you can not place 2 objects of the same type on the same grid position, but you can have as many grid types as you want (just extend the GridType enum script)

Object group Info

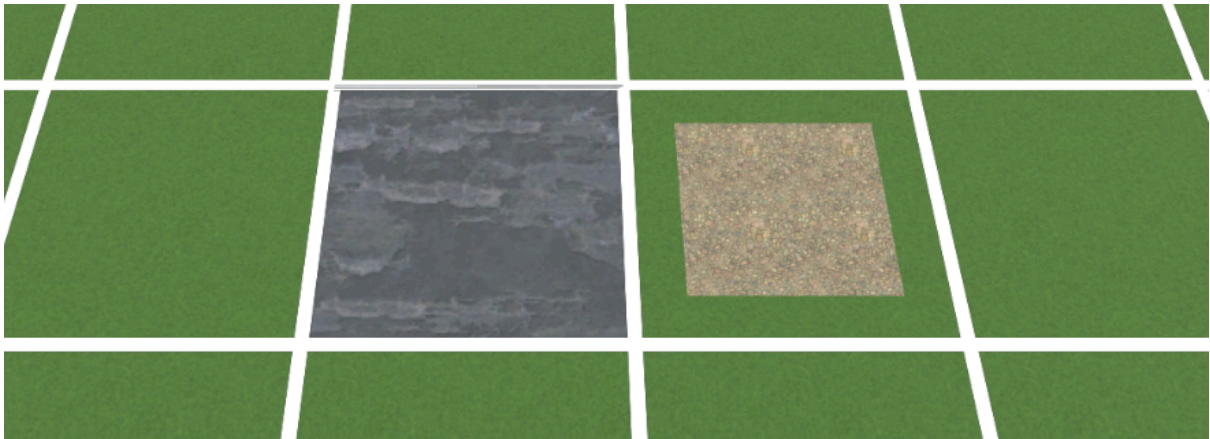
The object group info is basically only for grouping the objects into different tabs for the shop UI. The shop currently has 4 different tabs (Building, Nature, Decoration & terrain)

For each tab only the objects that belong to the same group will be shown in the shop

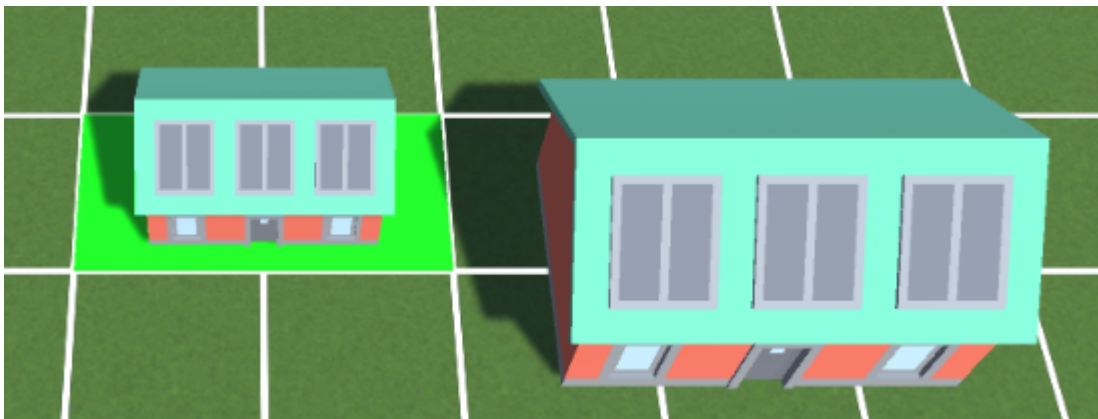
Dynamic Size

This is an additional grid system feature that allows you to scale your object dynamically on runtime, based on your grid cell size. The object transform scale will always have the cell size as value, so that a dynamic object of scale 1x1 will always occupy exactly 1 cell, independent of the cell size. I added this feature because of the terrain objects when using a bigger cell size. You basically always want to fill out the full occupied cells with the terrain, therefore we need to scale it. But this also works for all other objects, not only terrains. If a dynamic object would occupy for example 2x3 cells with a cell size of 1, it will be scaled to also occupy 2x3 cells with any other arbitrary cell size.

Here you can see an example of terrain objects, one with and one without dynamic size.
(left is dynamic and right is fixed size)



The same works with arbitrary object like houses:
(left is dynamic and right is fixed size)



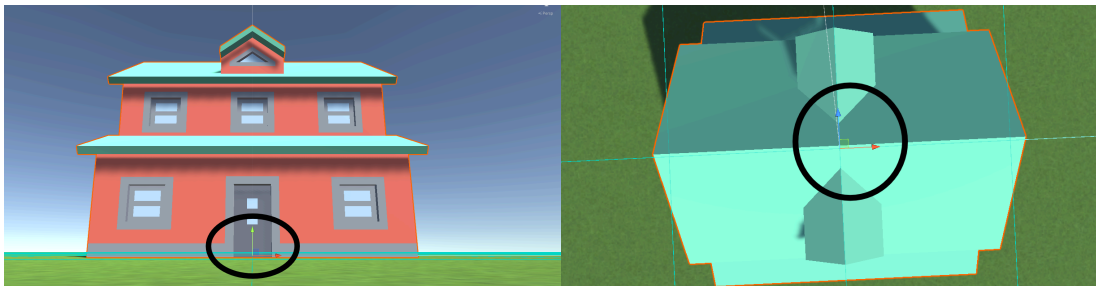
Prepare your 3D models

Requirements

- Pivot of the 3D model should be centered in x and z axis
- The 3D model should have a collider attached so we can interact with it after it is placed.
- The Y axis of the pivot point is handled automatically on runtime, no need to worry about that

Most of the used 3D models in this asset already have the correct pivot point.

E.g. For the house type 06 the pivot is already on $y=0$ and centered across the x and z axis.

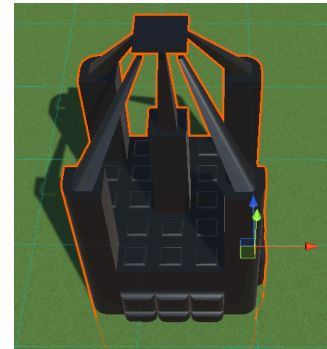


If this is the case for your models you can simply create a prefab out of it and assign it to the proper placeable object.

Problem: Wrong pivot

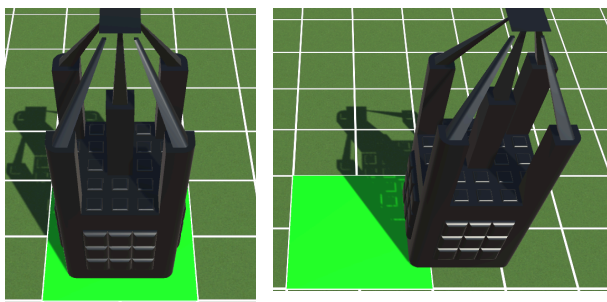
Obviously not all 3D Models will use a perfectly centered pivot.

For showcasing the issue I have created an example object with a wrong pivot. As you can see the pivot is not centered in the x and z axis. The y position of the pivot is not even at the bottom of the mesh, so instantiating the object at 0,0,0 position would result in the object being half under the floor.



Let's try out to place this object on the grid:

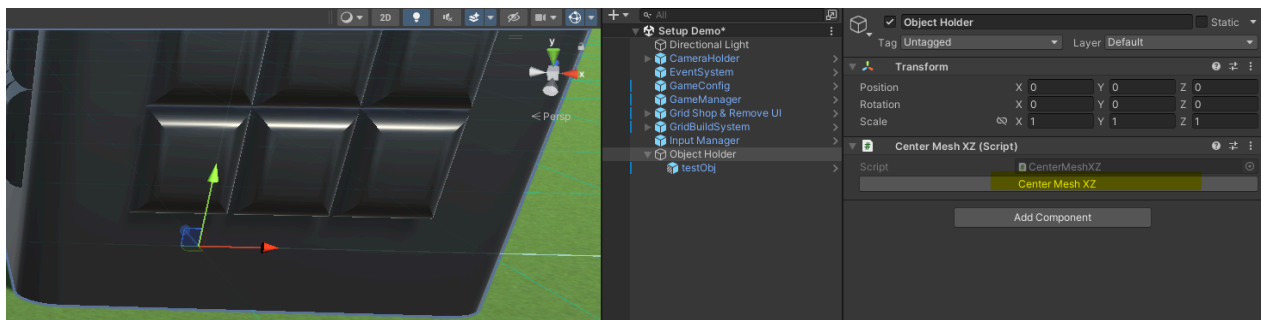
The left example is without rotation and the right one with rotation applied.



You can see that the automatic grid pivot offset calculation at runtime perfectly handles the height offset. However if you try to rotate the object you can see that the object is not anymore in the marked grid cells where it actually should be.

Solution: Center pivot of your 3D model

To fix the previously described issue you simply need to create an empty gameobject and add the CenterMeshXZ to this object. Then add the 3D model as a child of this object.



You can simply click on “Center MeshXZ” and the pivot is automatically fixed for you. However you do not need to use the button, since the script will be executed on runtime which will fix the pivot. The most important thing is just that the script is attached to the parent object.

Please make sure that you have a collider assigned to your model

Now your model is ready to be added to a placeable object and used in the grid system.

Shaders

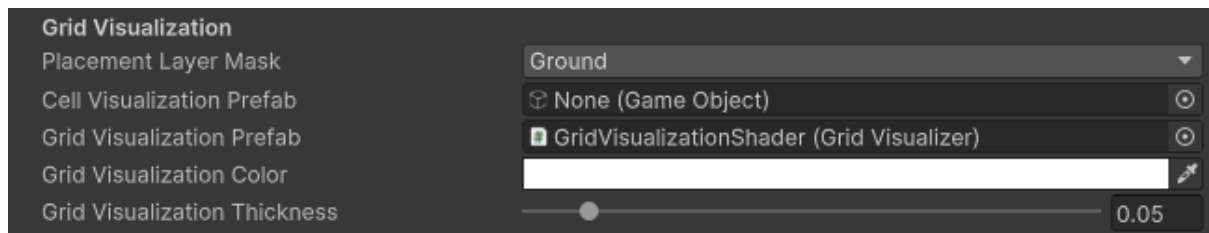
The custom shaders are located in the ShaderGraphs folder of the asset.

SF Grid

The Grid shader is used for the overall grid visualization.

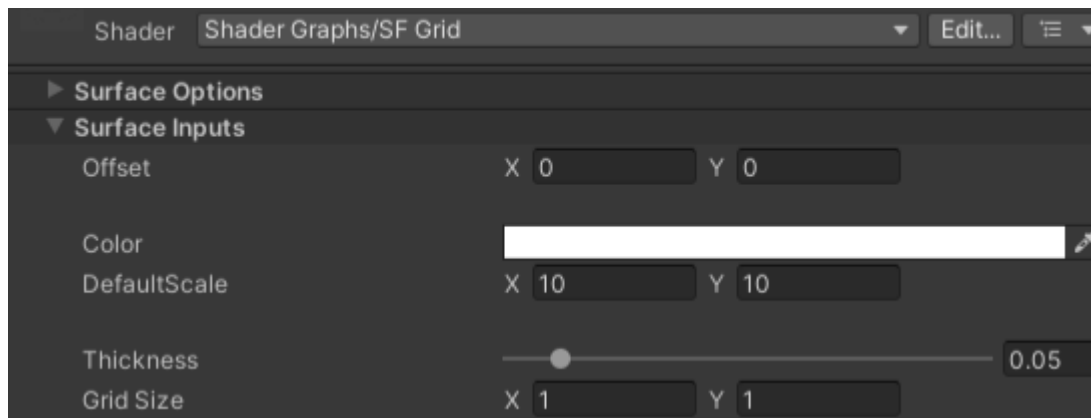
You can find the prefab under Prefabs → Grid → Visuals → GridVisualizationShader

This prefab is already assigned in the Game Config in the demo scene:



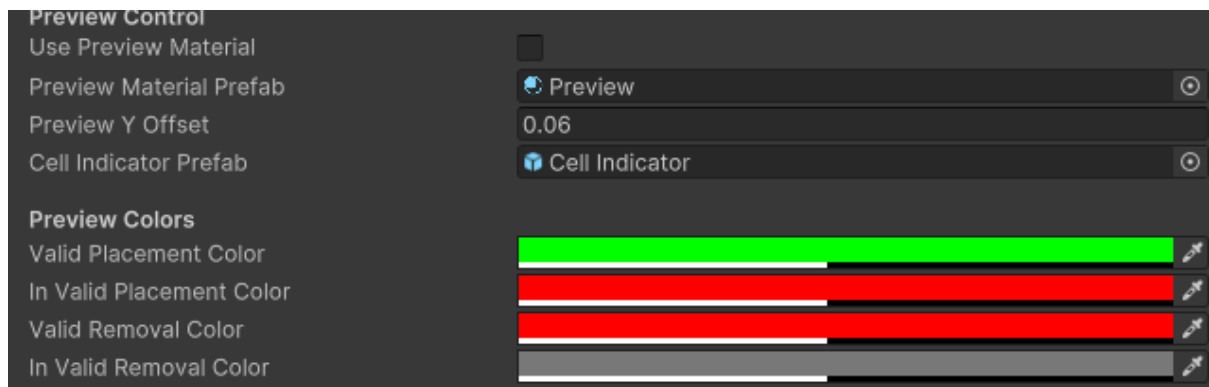
In The Game config you can already configure the color and the thickness of the shader

The shader itself has some more properties which you don't really need to worry about.



The grid size is automatically handled by the Grid visualizer script attached to this prefab. Offset is not really needed since the grid visualizer already handles position the grid at the proper position.

SF Preview



The preview shader has only a color value and displays objects in a slightly transparent way.

The Preview material assigned to the GameConfig has this shader attached.

To change the color of the shader you can configure the preview colors.

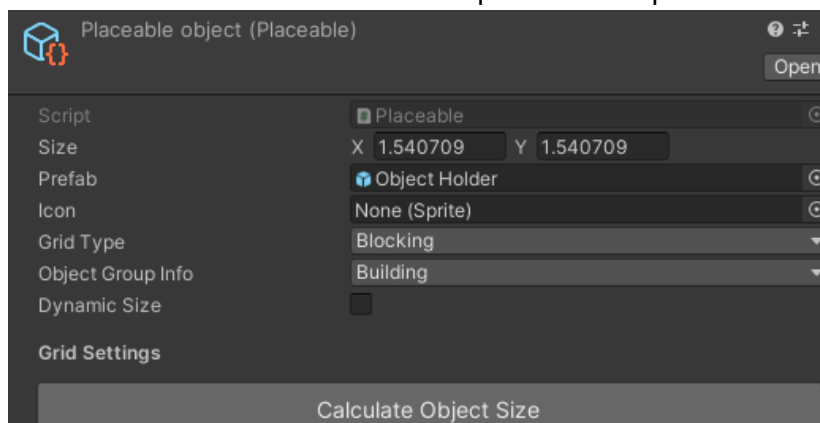
All colors that you can configure are applied to the same material on runtime, thus the color will change on runtime depending on the state(placement or removal) and the validity of the removal/placement action.

Deep dive

This section has some additional information that can be helpful for customizing and extending the grid system. You don't necessarily need to go through it to use the grid system but I would encourage you to do so, so you will get a better understanding of the whole system.

Extending Placeable properties

Just for the overview here is a example of the scriptable



The grid type and object group info are simple enums which can be found under Scripts → Enums. Simply extend the enums as you like to add more groups or grid types. You can have as many as you want. Adding a third grid type would allow you having up to 3 objects at the same grid position.

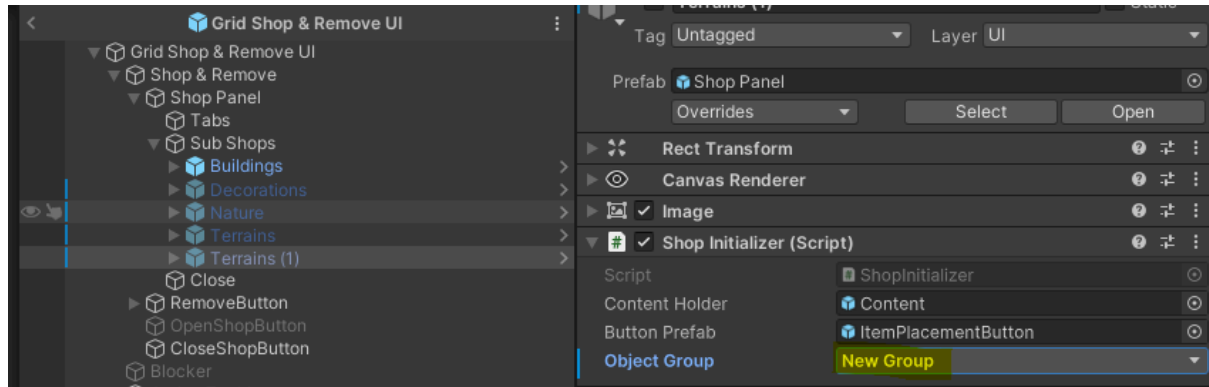
The **Remove UI dropdown** will automatically handle the new type at the same time you add it to the enum. It just shows all enums in the dropdown, so nothing to adapt here.

For the **tab based shop UI**, you would need to add a new sub-shop.

The tab creation is handled automatically. The only thing you need to do is to locate the Grid Shop UI prefab and navigate to the Sub Shops.

Then simply duplicate one of the sub-shops (I duplicated the Terrains one).

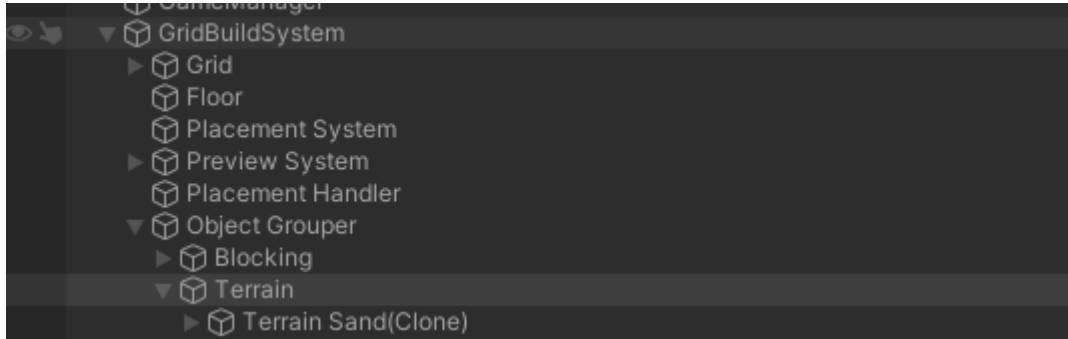
Select the duplicated shop, rename it as you want and change the object group in the attached shop initializer script



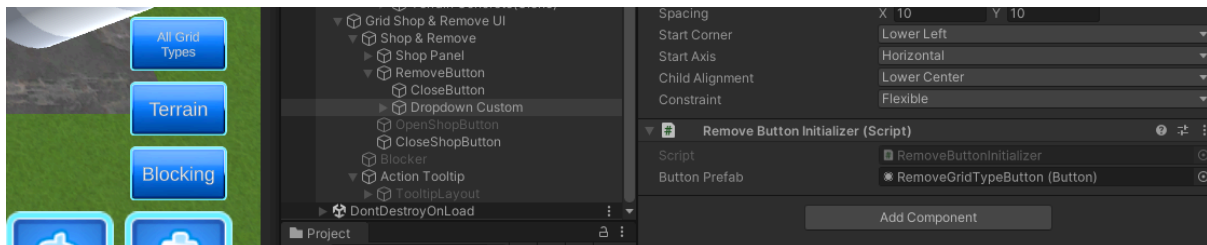
After you create new placeables for your new group and add them to the database, everything should work fine with the shop and the remove-UI.

Layer based removing

You already learned about the GridType property of the placeable object and how the GridData classes handle those types. Additionally the system automatically groups the instantiated objects by the grid type in the scene. When you start the demo scene you will notice an Object grouper in the GridBuildSystem that holds all the different Grid types.



Also when you click on the bin icon you will see a remove dropdown for different grid types.



This UI uses a custom Dropdown implementation and an Remove Button Initializer that triggers removal for either all types or only one specific type.

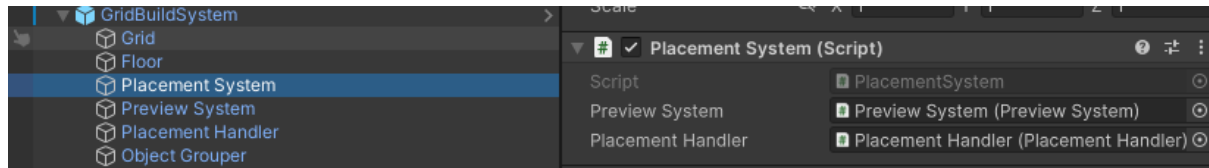
You probably have or want to create your own UI for your game, so for your buttons you would need to call one of those functions:

- *PlacementSystem.Instance.StartRemoving(gridType)*
- *PlacementSystem.Instance.StartRemovingAll()*
- *PlacementSystem.Instance.Remove(PlacedObject placedObject)*

So you have the option to either remove from all grid types at the same time, only a special grid type or to remove a placed object directly.

Placement System

The placement system is one of the core parts of the grid building system:



It is a state based system that handles place, load, move, rotate and remove actions. Currently there are 5 states implemented which all have a corresponding function call in the PlacementSystem script.

State	Method call in Placement System	Description
PlacementState	<i>StartPlacement</i> (string id)	Handles placing new objects on the grid. Buttons in the shop ui will trigger the StartPlacement Method
LoadObjectPlacementState	<i>InitializeLoadedObject</i> (PlaceableObjectData podata)	Similar to the Placement state, only that it uses data from the saveFile to rebuild the scene as it was saved on the last game run.
MoveState	<i>StartMoving</i> (PlacedObject target)	Triggered by the move button in the action tooltip when you hold on a placed object. Enables moving and rotating the selected object.
RemoveState	<i>StartRemoving</i> (GridDataType gridType)	Triggers removal of only the selected type. Objects of all other types will be hidden so you have a better view of what to remove.
RemoveAllState	<i>StartRemovingAll</i> ()	Triggers removing all grid types at once. When you select a grid position while in remove all state, all objects on this position will be removed

Note: The Rotate Action is handled directly in each state where it is needed.

Placement Handler

Each state described previously has an OnAction method that is triggered when you select a grid position while you are in one of those states. Those actions then call one method of the PlacementHandler which finalizes the action. The placement handler has 4 methods. The called method depends on the current state:

State	Called Method
PlacementState	PlaceObject
LoadObjectPlacementState	PlaceLoadedObject
MoveState	PlaceMovedObject
RemovStatee and RemoveAllState	RemoveObjectPositions

E.g. When you select an object from the shop UI you will enter the PlacementState which will allow you to place your object. Clicking the right mouse button on a grid cell will trigger the OnAction method of the PlacementState, which will then call PlaceObject from the placementHandler.

The *PlaceObject()* method then handles following things:

- Initialize the object
- Calculating pivot offset
- Considering rotation
- Considering scale if dynamicSize is activating
- Calling ObjectGrouper to add the initialized object to the correct group
- Adding a new script to the object called *PlacedObject*

Placeable Object → Placed Object

When a placeable object is placed on a grid it gets a new script attached called *PlacedObject*. *PlacedObject* is responsible for handling the mouse hold on the selected object and then trigger opening the action tooltip



It holds a reference to the *Placeable* object itself (in case of moving the object we need to access the object size from the placeable properties).

And it also initializes a *PlaceableObjectData* and holds a reference to it. This data is needed for the *SaveSystem*, since it stores the current object position, the asset identifier, the rotation direction and a unique guid that allows each placed object to be uniquely identified..

Save System

The save system is based on simple JSON serialization and simply stores and loads a json file from a specified location. For each object it stores a *PlaceableObjectData* with the needed information for the object.

On each creation and deletion of a *PlacedObject* the *saveData* will be automatically updated, so that you do not need to manually save anything before exiting the game.

Game Manager

The *GameManager* handles the initial loading of a specified *saveFile* and on game exit the saving of the existing game state, by calling the according methods from the *Save System*.

It also handles initializing different scripts in the proper order (using *Start* method in every script would lead to some race conditions between the placement system and the grid) For this reason the manager handles first initializing the *Grid* itself, and then the placement system:

- `PlacementGrid.Initialize();`
- `PlacementSystem.Initialize(PlacementGrid);`

I hope you enjoyed this asset and had a smooth integration into your project!

If you liked this asset I would really appreciate it if you could take a few minutes of your time and help me out by leaving a review on the Unity asset store for this package!

Something in the documentation is missing or not clear enough?

Any other questions/complaints/suggestions?

Please contact me under: ***gamedevibk@gmail.com***