uBuild

In-game modular building system

Contents

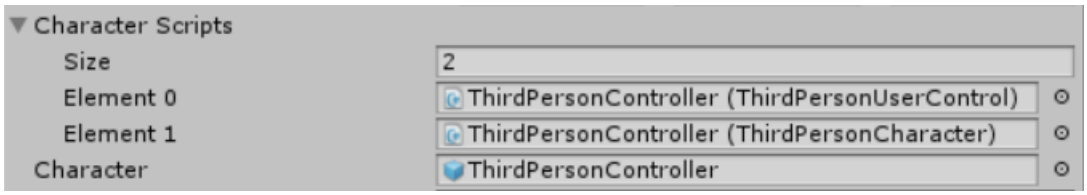
Introduction

I've always enjoyed drawing houses for Unity games in SketchUp and I think the layer system is very handy while drawing. Then I saw some first person Unity building systems and I thought, let's create such a system, featuring layers. uBuild is an in-game modular building system. Unlike other modular building solutions, this system allows the player to build using layers which you can turn on/off, place furniture and view your buildings from any angle.

Setting up the building system

To start using the building system, import uBuild into Unity. Then follow these steps:

1. If you have your own character, put it in the scene. If you don't have a character yet, just import the standard third person controller and scale it down to 0.7 (also make the capsule collider smaller).
2. Drag your character in the scene and give it the Player tag.
3. Now click the uBuildManager object.
4. Add your character to the save & load script and open the settings panel.
5. In the settings panel, add your character to the character field and add your character's script components to the 'character scripts' list (**if you don't know how to add the scripts, please see 'character scripts setup'**):



Important:

- The scene already contains a Ground object. The ground/floor object should be tagged 'Ground'.
- For the UI to work, the scene has an eventsystem. Send Navigation Events should be unchecked here.
- Under Edit-Project Settings-Physics, uncheck Queries Hit Triggers.
- If you enter build mode and the pieces list doesn't show up, please check the pieces object (canvas) for NaN values. If this is the case, replace all NaN values with 0.

Creating new pieces

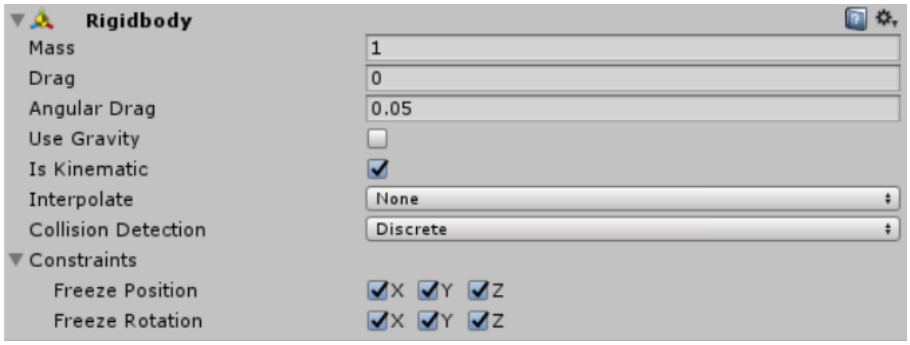
The main functionality of uBuild of course is creating modular buildings. To be able to create these, we need modules/pieces. To add a new piece, first create one:

Option 1: Creating a piece with a new material/texture

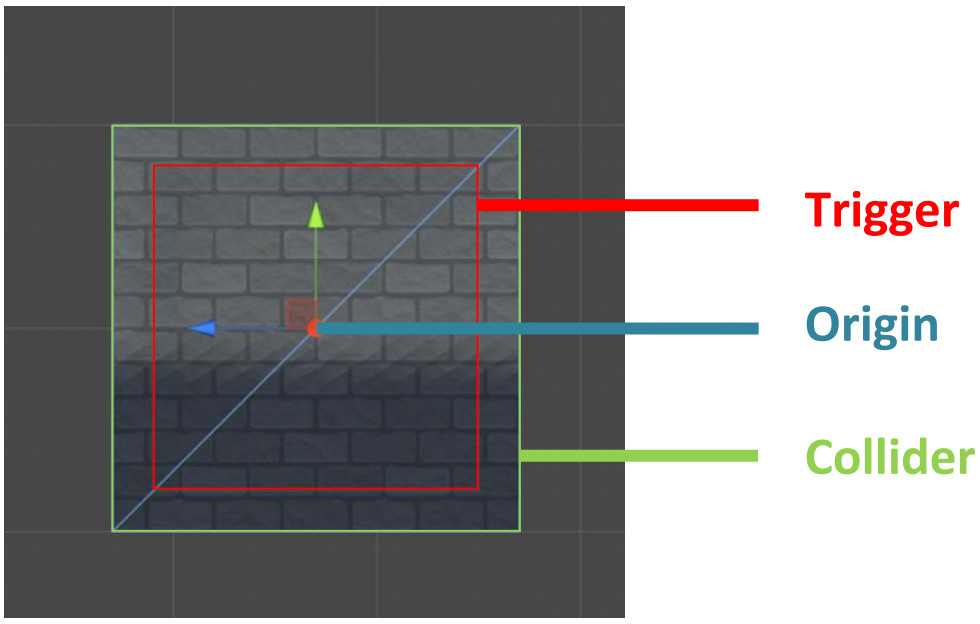
1. Just copy the type of piece you want from Prefabs-Pieces.
2. Rename the piece
3. Give it your new texture (drag & drop)
4. Drag your new piece into Prefabs-Pieces
5. Create an image for your piece (a screenshot or something like that)

Option 2: Creating a totally new piece

1. To create a new piece, first import your model and drag it into the scene. Make sure the origin is in the center of your object.
2. Add a normal box collider or mesh collider (depends on the object shape).
3. Add a rigidbody and use these settings:



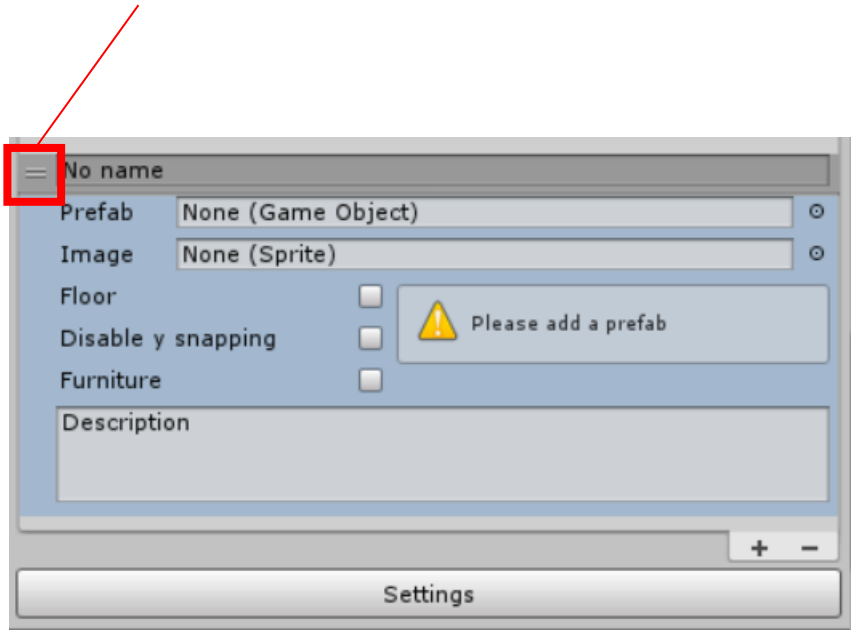
4. Add another collider and set it to 'is trigger'. Scale this one down to make it a little bit smaller than the actual collider. It should now look like the example below:



5. Add a ‘Piece Trigger’ script to your piece. Set the scale values to your piece’s in-game size
6. Name the piece and give it the ‘Piece’ tag
7. Drag your new piece into Prefabs-Pieces
8. Create an image for your piece (a screenshot or something like that)

Use your new piece in-game

1. Select the uBuildManager and click ‘edit pieces’.
2. Scroll down and click the + icon.
3. Add your new piece in the prefab field, name it, add your image and write a description for the piece.
4. Drag it to reorder your pieces.



Toggle settings:

1. Floor: This will make it snap to walls differently to be able to build multiple floors.
2. Disable y snapping: Disables y snapping of this piece. For example, doors and windows should not have snapping on the y axis to place them in the walls properly.
3. Furniture: Sofas and cabinets for example. This will put the piece in the furniture list and disables snapping on all axes.

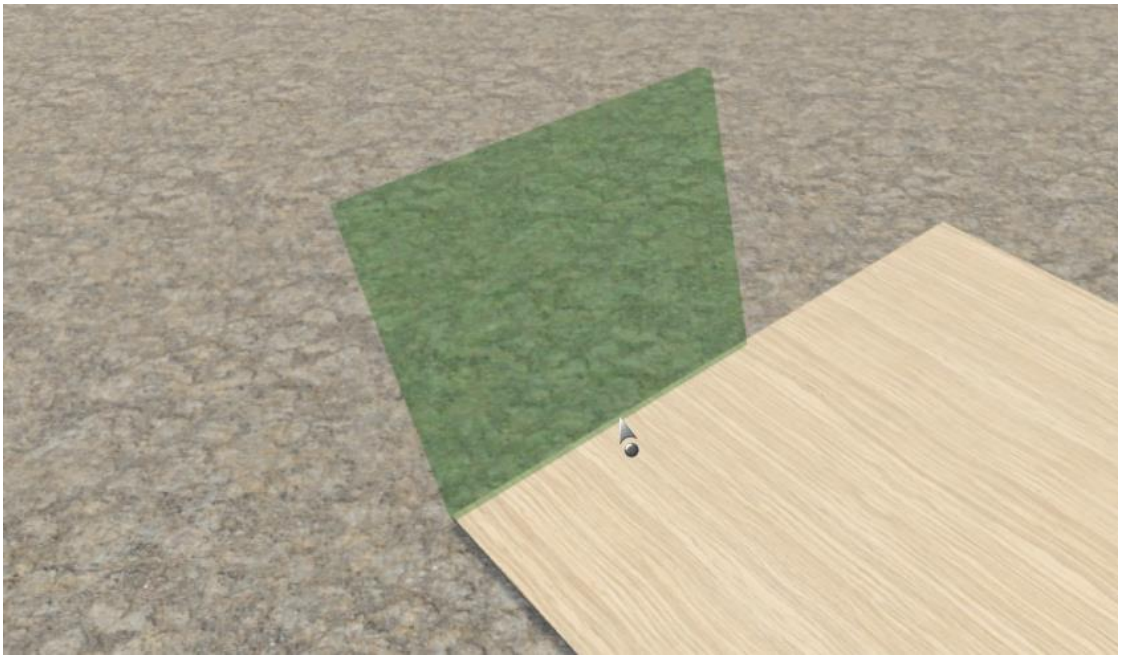
After making changes to the piece list, it’s a good idea to reset the PlayerPrefs data since the positions of your pieces in the list have changed. To reset PlayerPrefs, click the uBuildManager and open settings. Then scroll down and click ‘Delete Data’.

Placing pieces

One of the main features of uBuild is placing pieces. You can select pieces from the list and press a key to start placing. This probably works a bit different than in other systems.

When you click one of the pieces, it marks the piece as selected. When you then press the place key, it gets the selected piece and instantiates it from the list. It doesn't first create a placeholder or something; it immediately uses the piece itself. To use the piece, the main collider is disabled and the trigger detects if the piece can be placed using the piece trigger script. However, you can't just delete the piece trigger script since it also stores the piece type and layer. If you're placing a piece, it also has a nice transparent green color (adjustable in the uBuildManager settings). This is done by changing its shader via script.

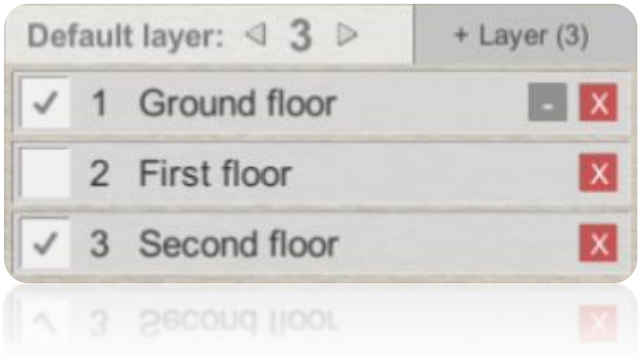
After you click the left mouse button and the piece was place-able, the shader changes back to diffuse and the collider gets active so characters cannot walk through walls for example.

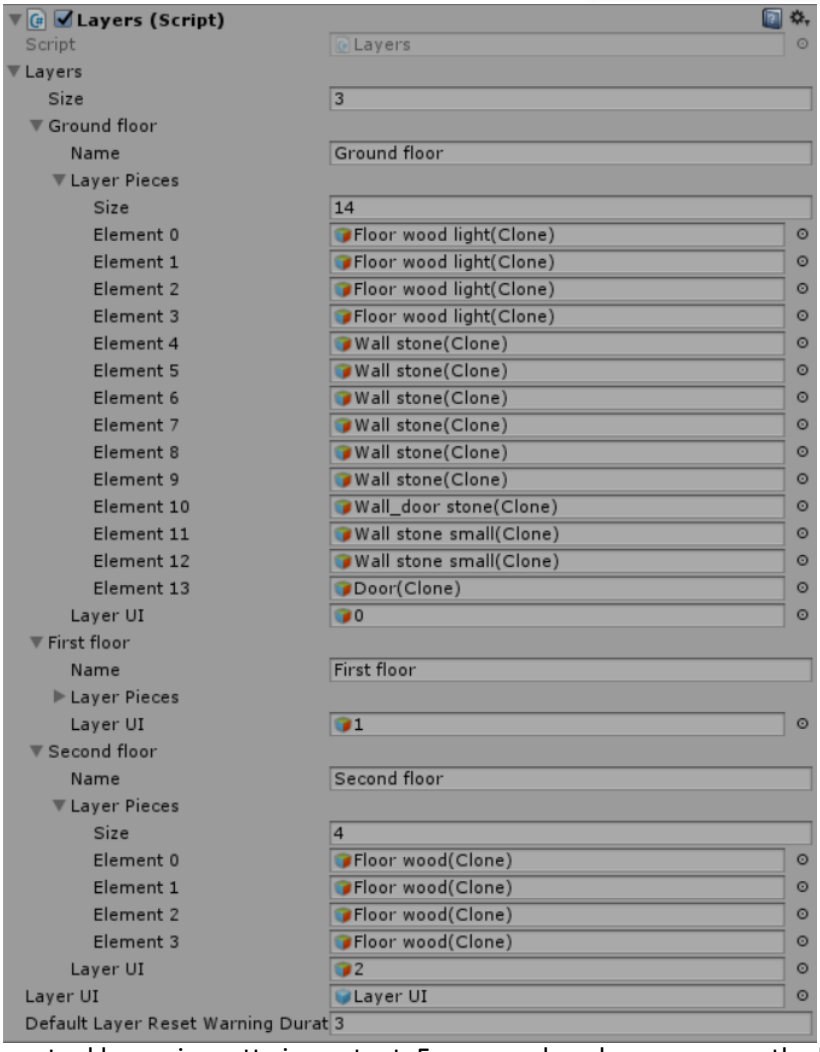


Layer system

In-game, layers just look like this:

A list of layers which you can turn on/off, add pieces to and remove. There's a default layer to automatically add your pieces to a layer and a button to add layers.





In the inspector it's easier to see what's going on. As you can see, each visible in-game layer is linked to an element in the layers list. A layer consists of a name, a list of added pieces and a UI element. Each layer saves the added pieces by storing the layer in the 'PieceTrigger' script. This way, when saving, all of the pieces save their current layer so they can be re-added to their layer once you restart the game.

By default (when the player places a new piece) the piece layer is 0. This means that it has not been added to a layer and it's not added to the layer list. This also means, that element 0 in the layers list is layer 1. Element 1 in the list is layer 2 etc. That's why you will sometimes see +1 & -1 in the layer script (to use the correct list element).

The link between UI elements and actual layers is pretty important. For example, when you press the button to add a new layer, it doesn't just add the visible UI. It creates a whole new element in the layer list so players can then add new pieces to it.

Settings (uBuildManager)

- The green & red colors are the colors while placing. They're called green & red since these are the most logical colors while placing. You can just change the colors and their alpha if you don't like them.
- The selected color is also a piece color. This is the selected color and as you can see it has a bit higher alpha. You can change this color as well.
- Button highlight is the highlight visible in the in-game list when the player has selected a piece. It's just the outline color of the selected button.
- The button is the button prefab visible in the in-game list of pieces. Please do not change this prefab; however, you can change the button prefab itself via 'Prefabs-UI-piece button'.
- Fp cam position is the position of the camera when the player is not in build-mode relative to the player position. It's called FP (first person) because I decided to make it third person later on.
- I've already explained character and character scripts before, please see the 'Basic Building System' part.
- Door key is the key to open doors when the player is not in build-mode.
- Build mode key, placement key & switch furniture mode key probably speak for them.
- Then there are 3 text areas. You can use these to write some help text for the player while building.
- The 'Delete Data' button is pretty important. It deletes all PlayerPrefs to reset the game. Please use this button before actually building your game.

Saving & Loading

After restarting the game, it's handy if buildings are saved. To do this, uBuild uses Unity's PlayerPrefs system.
Basically, each piece in the scene has a 'Piece' tag. The save & load script uses these tags to get all pieces. Just before saving all of the buildings get active. Then a list is created containing all the placed pieces. The length of this list is saved, and for each element in the list, it saves a type, a layer, a position and a rotation. The type is actually the position of the current piece in the main pieces list. Then all the buildings will get active again (except for inactive layers).
Saving layers is almost the same process, but then it saves states and names. It also saves player position & rotation, camera position & rotation, doors opened/closed, build-mode and furniture-mode.

To then load everything, it gets the amount of pieces to add and for each piece; it gets the type, layer, position and rotation and places it in the scene. This way it rebuilds all buildings.

Then it adds all layers, applies their names & state and it will check if buildmode is active. It checks furniture-mode, applies player position and rotation, checks all doors and gets the camera position and rotation.

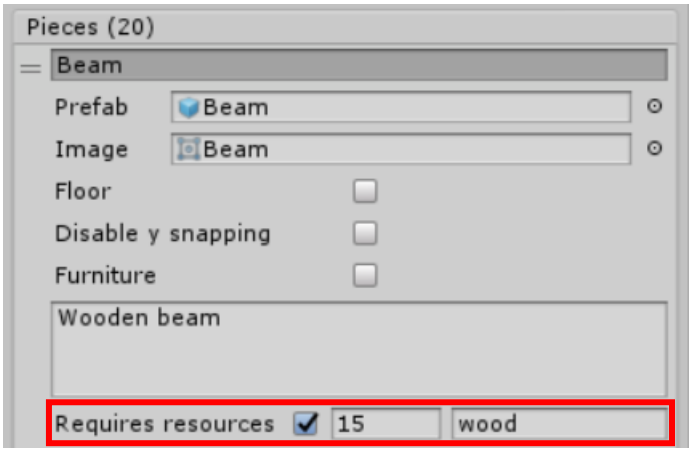
Currently, the save & load script saves after a certain amount of time and OnApplicationQuit. It might be a good idea to also save when opening a different scene or menu. If you have the uBuildManager with 'save & load' script in scene, save data like this:

GameObject.Find("uBuildManager").GetComponent<SaveAndLoad>().save();

When you're saving via a script added to the uBuildManager, you can remove the gameobject.find part and when you're saving directly in the 'save & load' script, you can just call save();

Using resources

To give your players an extra challenge, you can add a resource system to your game. uBuild includes an example of how to use resources ('Resources example' script on manager object). To let players only build with enough resources, check 'Requires resources'. Then type in the amount of resources needed and type the PlayerPrefs name used to save this resource.

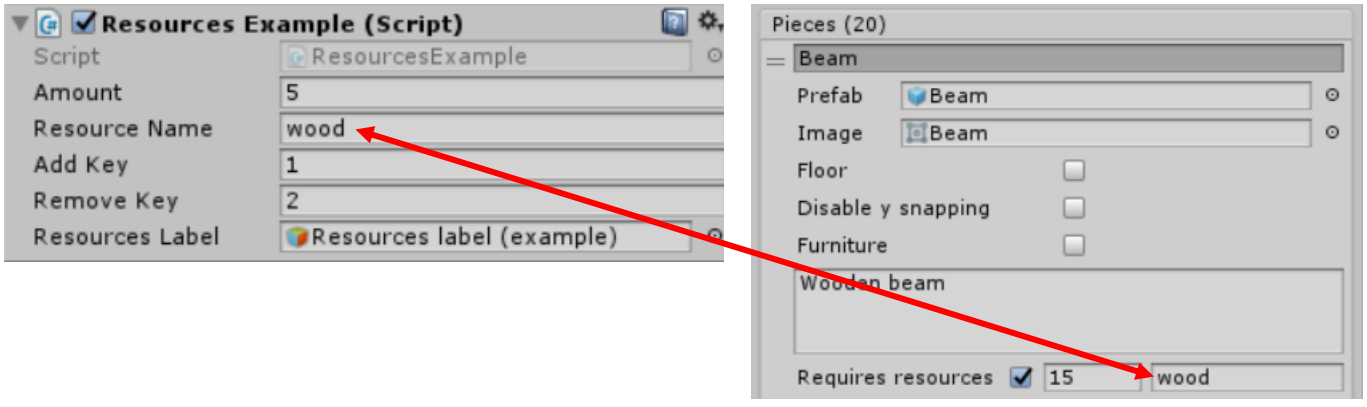


Each time a player wants to place a piece, it checks if the piece requires resources and if so, it removes the amount of resources when the piece is placed. When a player removes a piece, it restores the resources.

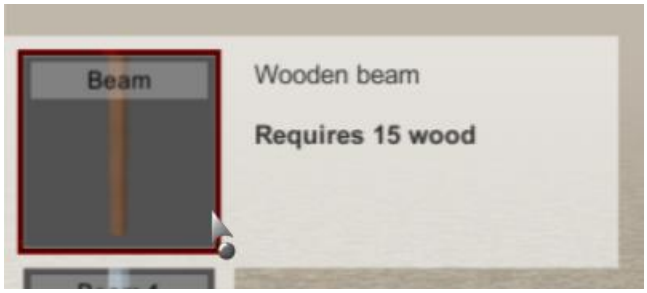
The resources example script isn't necessary for the building system at all, it just shows how to add and remove resources:

```
if(Input.GetKeyDown(addKey)){
    PlayerPrefs.SetFloat(resourceName, PlayerPrefs.GetFloat(resourceName) + amount);
}
if(Input.GetKeyDown(removeKey)){
    PlayerPrefs.SetFloat(resourceName, PlayerPrefs.GetFloat(resourceName) - amount);
}
```

The resourceName variable is the PlayerPrefs name used to save this resource, so when this name corresponds to the resources name of one of the pieces, the piece will use this value:



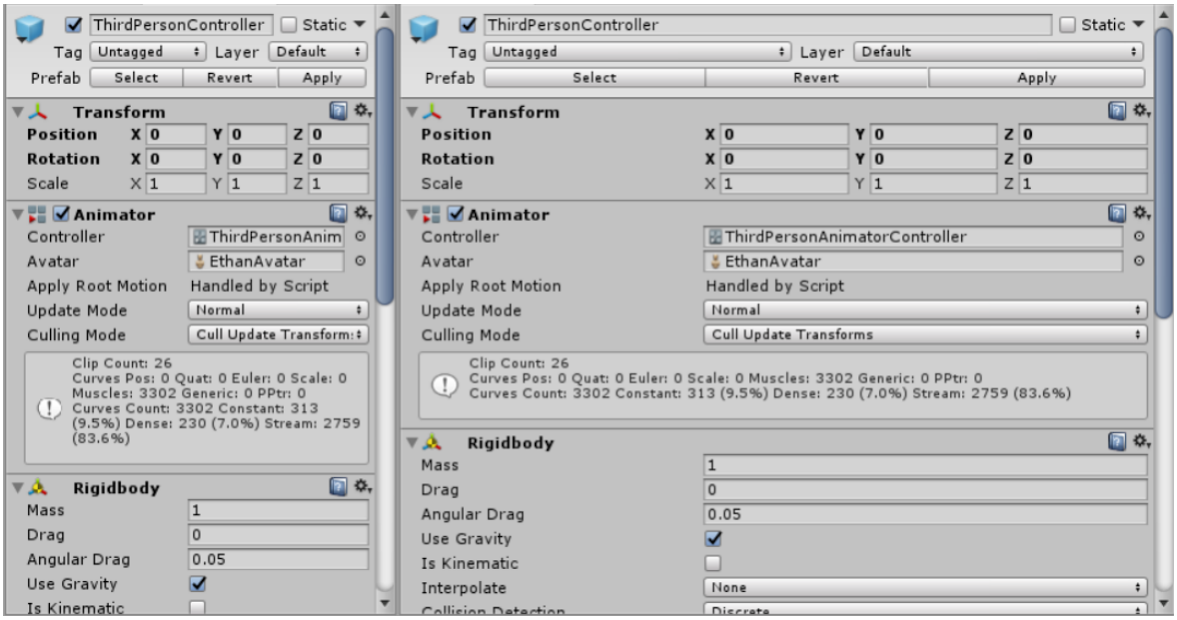
If a piece uses resources, it will be shown in the description:



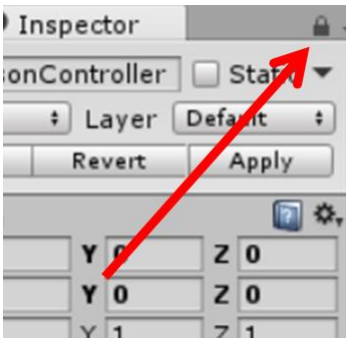
And you can check the example resources in the bottom-right of your screen:

Character scripts setup

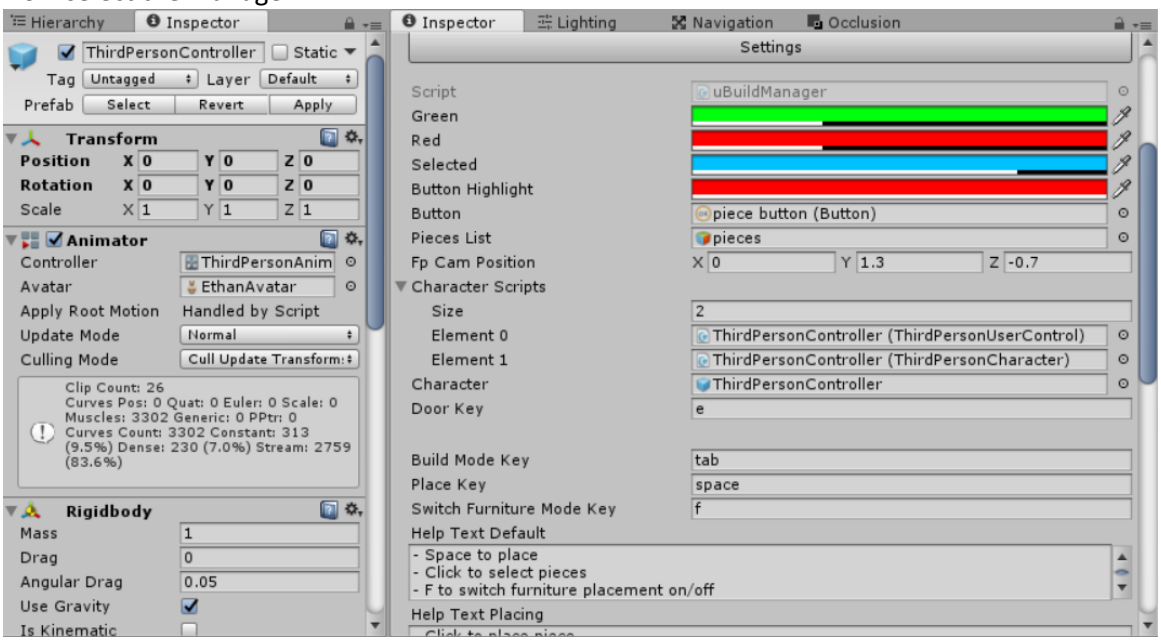
First open two inspector panels, and select your character (in the hierarchy):



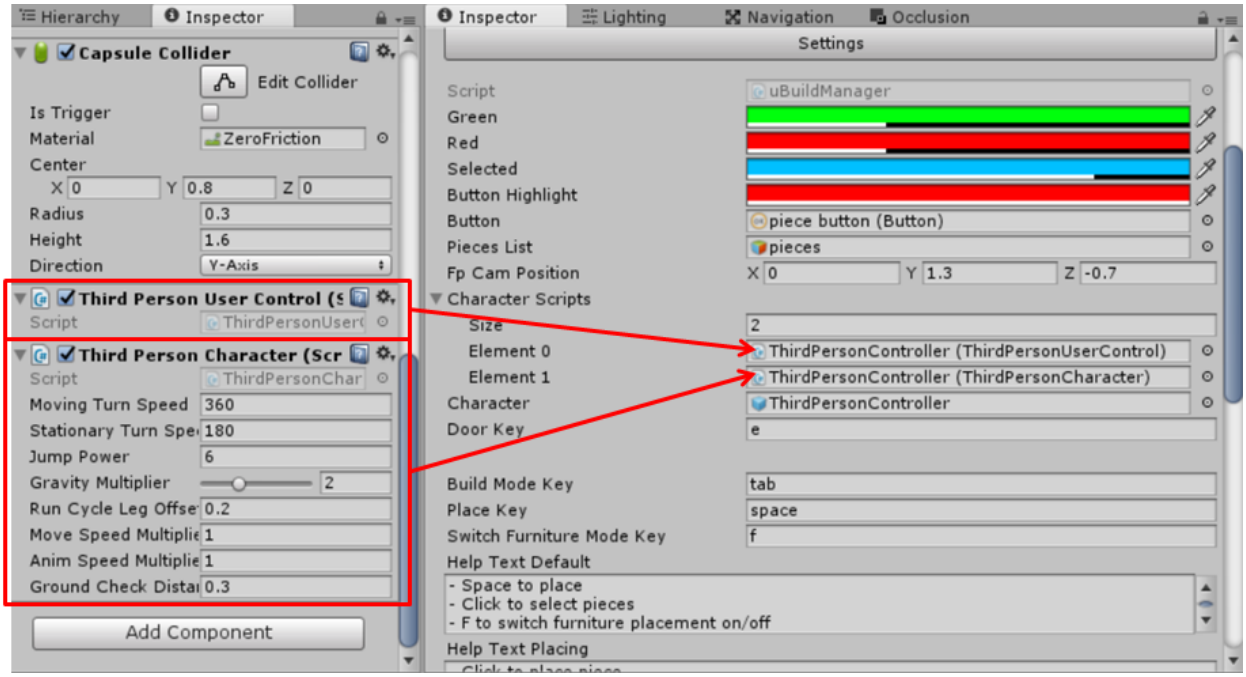
Then lock the first inspector to the character:



Now select the manager:



Than you can drag & drop the character scripts to the fields:



Conclusion

In this document, I've explained the uBuild asset. I hope it helps and you like uBuild. Good luck with your project and thank you for using uBuild. If you have questions/suggestions for uBuild, you can contact me via:

T3Dmake@gmail.com