BuildableObject.cs

Properties:

public bool canBeModified;
This bool determines if the player can interact with an object to start the upgrade process. Setting it to false will disable the child object "triggerPoint" making the player unable to interact with this system.

Private set group

public bool isUpgrading {get; private set;}
This boolean is true during the duration of the upgrade and false otherwise. To be used for player animations and other events or systems that require this information.

public bool hasBeenUpgraded { get; private set; }
This boolean is true after an object has been "upgraded"

The following member variables MUST be set in the editor for the system to work:
[SerializeField]

private Mesh upgradeMesh; (optional)
This is the mesh that we will use to replace the current mesh of the object in the upgrade event. Set to null if you want your object to have no mesh after the upgrade, otherwise a mesh must be provided.

private GameObject particlePrefab; (optional)
This variable stores a custom prefab of the particle effect that will be played to cover the mesh transition. It is not mandatory for the rest of the system to work but without it you will not have anything to hide the mesh swap.

private GameObject triggerPoint; (required)
The trigger point is a child object(required) of this object that has a collider(current) or interface (to be implemented) upon which the player can interact with to start the upgrade process. The object only requires a trigger collider, everything else is optional.

private GameObject particleSpawnLocation; (required)
This is to be set to an empty object where the particles will be spawned in relation to the current object. I recommend setting this as a child object however this is not required.

NOTE: The following 2 list MUST be equal in length as they work in unison with the inventory system. If these objects are left empty then the player can just interact with the object and start the upgrade process without any further requirements

private List<string> requiredItems; (optional)
This is a list of strings containing the key values of the materials needed for construction, these key values will be passed into the inventory's function to check if the player has the required amount of the required items.

private List<uint> requiredQuantities; (optional)
This list contains the quantities of resources needed for each required item in "requiredItems". As such the index of the element matches the index of the element in "requiredItems"
Example: requiredItems = { "stone", "wood", "slime", … }
requiredQuantities = { 5, 3, 1, … }

This means that the player requires 5 stones, 3 wood, 1 slime etc.

private float EffectDuration = 0; (optional)
This value defaults to 1 second if left untouched or set to a negative. Otherwise, it can be set in the editor by a designer. Upon start of the building/upgrading process the mesh will be swapped in half of the time provided and the particles will last for the amount of this duration upon the start of the event.

private MeshFilter objectMesh; (required)
This is used internally to process the mesh swapping logic. Without it, the mesh cannot be swapped or rendered.
Note: This variable only requires the matching component to be attached to the object

Methods:

NOTE: ALL methods in this class are private, all logic and functionality is processed internally based on the current set parameters. They are only listed here for development/debugging purposes.

private void BeginUpgrade();

This function is called once an interaction with the trigger point has been made with the player AND the player has the necessary resources and quantities.
private IEnumerator SwapMesh(float time);
This function is called inside of BeginUpgrade().
Parameter:
Float time, the value passed in here is EffectDuration / 2.0f aka half of the duration.
This function handles the mesh swap.

private IEnumerator ShutDownParticles(float time, ParticleSystem particles);
This function is called inside of BeginUpgrade().
Parameters:
Float time, the time the call waits before destroying the particles.
ParticleSystem particles, the particle system component attached to this object.

This function will wait and then destroy the particle prefab instance. At the end of the upgrade/build event.