Tutorial – Adding an Upgrade Action

by JNA Mobile

Table of Contents

Quick Start

Goal

In this tutorial we add the ability to upgrade the Sawmill building and increase the output of the Gather activity based on the level of the sawmill.

This tutorial shows several features:

1. Extending BuildingData.
2. Adding new Activities.
3. Customising building behaviour.

Location

The finished scene and all related assets for this tutorial can be found in the folder:

Assets/CityBuilderStarterKit/Extensions/BuildingUpgrades

Video

The video for this tutorial is available at:

http://youtube.com/JNAMobileHidden/

and is called:

CBSK – Tutorial 2 – Building Upgrades

Code Updates

The Building Data

The first part of this change is to extend building data so that the buildings level is stored. To do this we create a new class that extends the existing **BuildingData** class. We call this class **UpgradableBuildingData**.

Here it is:

```
/**
 * Extends building data by including a building level which can be upgraded.
 */
[System.Serializable]
public class UpgradableBuildingData : BuildingData {

   /**
    * The buildings current level.
    */
   virtual public int level {get; set;}
}
```

There's not much too this class, by inheriting from BuildingData the class automatically becomes part of the saving and loading system. All we need to do is mark it as serialisable with the **System.Serializable** annotation.

We then add a new property called **level**. We use the pattern from all the data classes, namely publicly accessible virtual properties. This means we can further extend this class in another class.

The Building

We also need to extend the **Building** class so we can ensure two things. Firstly that we use **UpgradableBuildingData** instead of **BuildingData** when we create a new building and secondly that we generate rewards differently. We call the new class **UpgradableBuilding**.

There's quite a bit of code in this file, but most of it is copied straight from the Building class.

First we copy the **Init()** method from the Building class and make the following change (changes in bold):

```
   /**
    * Initialise the building with the given type and position.
    */
```

```
override public void Init(BuildingTypeData type, GridPosition pos){
    data = new UpgradableBuildingData();
    ((UpgradableBuildingData)data).level = 1;
    uid = System.Guid.NewGuid ().ToString ();
    Position = pos;
    this.Type = type;
    State = BuildingState.PLACING;
    CurrentActivity = null;
    CompletedActivity = null;
    view = gameObject.GetComponent<BuildingView> ();
    view.Init (this);
    view.SetPosition(data.position);
}
```

This code just ensures data is set to the correct type and that our default building level is 1.

Next we copy the **AcknowledgeActivity()** method from the Building class and make the following change (changes in bold):

```
/**
 * Acknowledge generic activity.
 */
override public void AcknowledgeActivity() {
    if (!((this.data) is UpgradableBuildingData)) {
        Debug.LogWarning ("Upgradable buildings should use upgradable building data");
        return;
    }

                                ... some code removed for clarity...

    if (data != null) {
        switch (data.reward) {
        case RewardType.RESOURCE :
            ResourceManager.Instance.AddResources(data.rewardAmount * ((UpgradableBuildingData)this.data).level);
            break;
        case RewardType.GOLD :
            ResourceManager.Instance.AddGold(data.rewardAmount * ((UpgradableBuildingData)this.data).level);
            break;
        case RewardType.CUSTOM :
            if (data.type == "UPGRADE") {
                ((UpgradableBuildingData)this.data).level +=1;
            } else {
                SendMessage("CustomReward", data, SendMessageOptions.RequireReceiver);
            }
            break;
        }

                                ... some code removed for clarity...
```

Our changes do two things:

- Firstly we multiple the reward from any activity that creates resource or gold by the buildings level.
- Secondly we add a special handler for the UPGRADE activity. When this activity completes we increase the buildings level by 1.

Note that alternatively we could have added this code in to a **CustomReward** method which would respond to the **SendMessage()** call.

Configuration Updates

Activity Data File

We create a new Activity Data file by copying the existing one. We call the file **UpgradableActivityData** and ensure it is in a Unity Resource Folder.

To this file we add a new activity of type UPGRADE:

```
<ActivityData>
  <type>UPGRADE</type>
  <durationInSeconds>20</durationInSeconds>
  <description>Upgrade the buildings level.</description>
  <reward>CUSTOM</reward>
  <rewardAmount>1</rewardAmount>
  <rewardId>UPGRADE</rewardId>
</ActivityData>
```

Notice that the reward type is CUSTOM.

Building Data File

We create a new Building Type Data file by copying the existing one. We call the file **UpgradableBuildingTypeData** and ensure it is in a Unity Resource Folder.

In this file we update the Sawmill building so it includes a reference to the new UPGRADE activity:

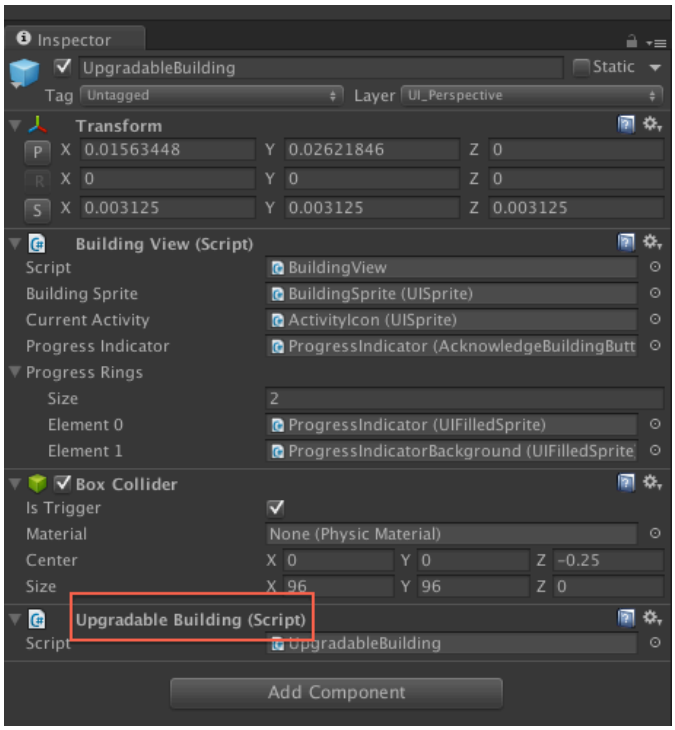
```
<activities>
    <string>GATHER</string>
    <string>UPGRADE</string>
  </activities>
```

A button for this activity will be automatically generated if you are using the default UI.

Scene Updates

Create a Building Prefab

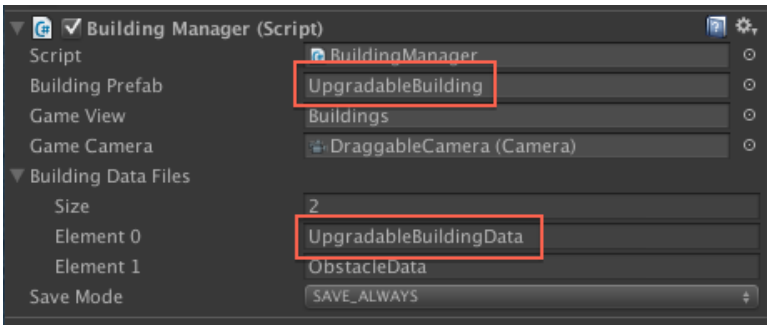
In order to use our new Building class we create a new prefab based on the existing Building prefab. We remove the **Building** behaviour from the prefab and add the **UpgradableBuilding** behaviour in its place. Finally we save the prefab.



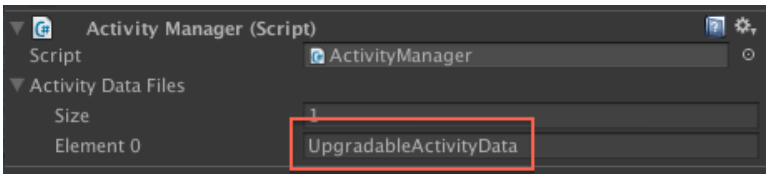
Update Manager Objects

Finally we need to associate all the new data and the new prefab with the manager objects. We make the following changes:

1. Drag the new prefab on to the **BuildingManager** and change the name of the data file to match our new new building data file:



2. Update the **ActivityManager** to refer to the new activity data file:



3. Change the name of the saved game in the **PlayerPrefsPersistenceManager** so as to avoid any conflict with existing saved games: