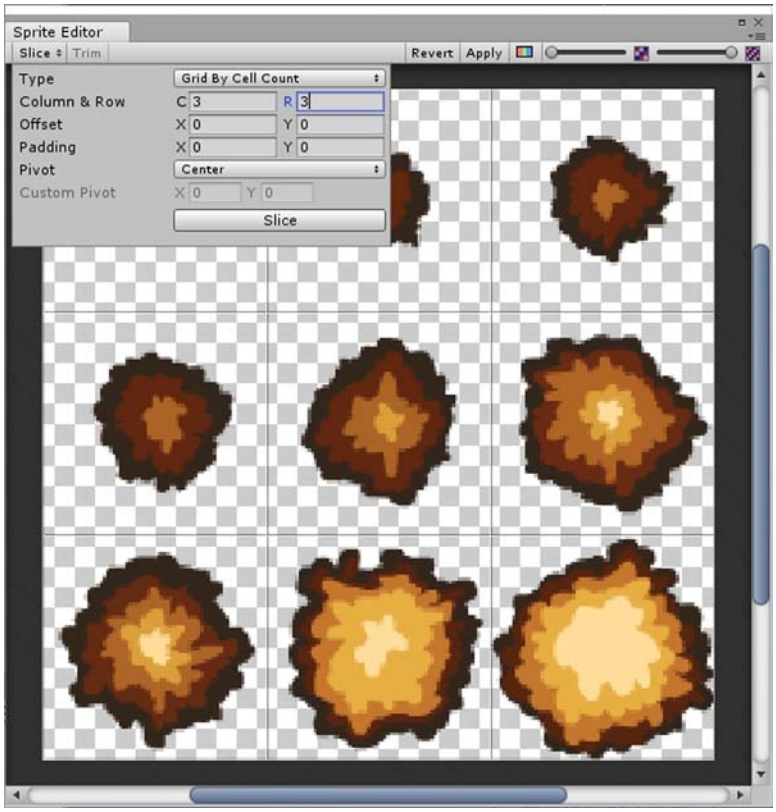
Building an Animated Unity Prefab with MonoDevelop

This document described how to build an animated Unity prefab for an explosion. Getting the visual portion of the prefab set up properly is more complicated than we've seen in the past because the explosion sprite is animated rather than static.

Select the explosion sprite in the sprites folder of the Project window and use the dropdown in the Inspector to change the Sprite Mode to Multiple; this tells Unity that you're using a *sprite sheet* rather than a single image. We still need to get the individual frames of the animation set properly, so click the Sprite Editor button in the Inspector.

When the Sprite Editor popup appears, select the Slice dropdown at the upper left and change the Type to Grid by Cell Count. Change the C and R values to 3 (because there are 3 columns and 3 rows in our sprite sheet) and click the Slice button; you'll end up with the image shown below. Click the Apply button near the top of the Sprite Editor, then close the Sprite Editor using the X at the upper right of the popup. If you click the arrow to the left of the explosion sprite in the sprites folder of the Project window, you'll see that the sprite now has 9 separate frames. You can left-click each of them and see that frame at the bottom of the Inspector.



Explosion in Sprite Editor

Now that we've got our separate animation frames set up, it's time to work on our Explosion prefab. Drag the explosion sprite (not one of the individual frames, the sprite with the arrow to the left of it) from the sprites folder of the Project window onto the Hierarchy window and change the name of the new game object to Explosion.

The next thing we need to do is add an Animator component to the Explosion game object. Click the Add Component button and select Miscellaneous > Animator (NOT Animation) to do that now. The Explosion still won't play as an animation, because we need to set up a Controller for the Animator component.

Create new empty animations and controllers folders in the Project window. Right-click the new controllers folder and select Create > Animator Controller. Name the new controller explosionController. Select the Explosion game object in the Hierarchy window and drag the explosionController from the controllers folder in the Project window onto the Controller field of the Animator component in the Inspector.

Next, we need to implement the explosionController. Double-click the explosionController in the controllers folder in the Project window to open the Animator window for the animator controller. Make sure you have the Animator window selected, then select Window > Animation on the menu bar to open a new Animation window. Select the Explosion game object in the Hierarchy window. In the Animation window, click the Create button to create a new Animation Clip. In the file navigation popup, select the animations folder, navigate into that folder, change the File name to Exploding, and click Save. Expand the explosion sprite in the sprites folder in the Project window, select all 9 children, and drag them into the Animation window where you see all the vertical lines. Change the Samples text box (near the upper left of the Animation window) to 30 so the explosion animation plays at 30 frames per second. Close the Animation window and the Animator window (right-click the tab for the window and select Close Tab).

Click the Play button in the Unity editor to test the animation; you should just see a looping explosion animation wherever you placed your Explosion game object in the scene. This might have seemed like a lot of work to get this simple animation working, but that's because Unity has a very robust animation system. If you want to learn more about the Unity animation system – which you'll definitely want to do if you start building character or other more complex animations – you'll probably find that the 2D Character Controllers tutorial from Unity is really helpful (it includes a discussion about building 2D animations from sprites).

Although it's kind of fun to have explosions loop forever once they're added to our game, that's not very realistic, so we'll fix that now. Create a new `Explosion` script in the scripts folder in the Project window and drag the script onto the Explosion game object in the Hierarchy window. Here's the complete script:

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

/// <summary>
/// An explosion
```

```
/// </summary>
public class Explosion : MonoBehaviour
{
    // cached for efficiency
    Animator anim;

    /// <summary>
    /// Use this for initialization
    /// </summary>
    void Start()
    {
        anim = GetComponent<Animator>();
    }

    /// <summary>
    /// Update is called once per frame
    /// </summary>
    void Update()
    {
        // destroy the game object if the explosion has finished its animation
        if (anim.GetCurrentAnimatorStateInfo(0).normalizedTime >= 1)
        {
            Destroy(gameObject);
        }
    }
}
```

Because we need to check if the animation is finished on every frame, we retrieve a reference to the Animator component and save it in the `anim` field in the `Start` method so we don't have to get that component every frame.

In the `Update` method, we check to see if the animation has finished by retrieving the `normalizedTime` field of the current animator state info. The `normalizedTime` field is a `float`, where the integer part is the number of times the animation has looped and the fractional part represents the percent progress in the current loop. That means that when the `normalizedTime` hits 1, it has just completed the first loop through the animation frames. When that happens, we destroy the game object the script is attached to.

Drag the Explosion game object from the Hierarchy window into the prefabs folder in the Project window to create the prefab. Delete the Explosion game object from the Hierarchy window, because we'll be creating the Explosion at run time.