CODE NINJAS®

Bronze Belt Ninja Guide

Activity 10: Dropping Bombs Part 4

Activity 10: Dropping Bombs Part 4

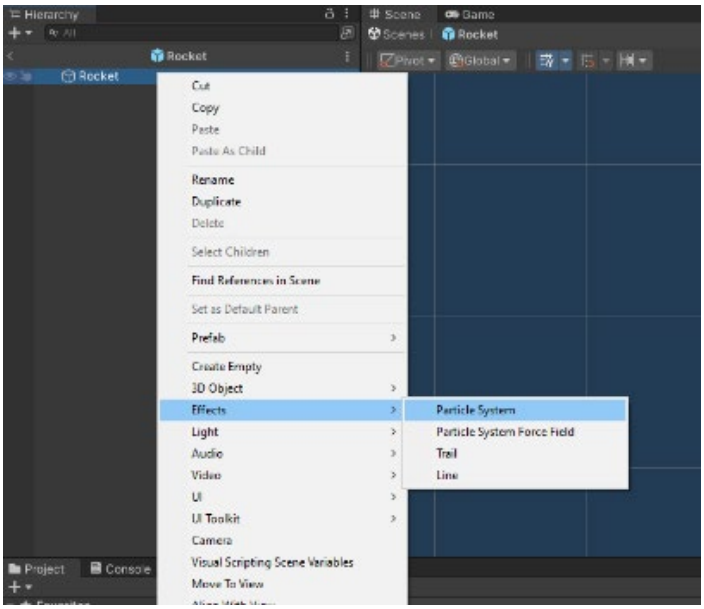
The Dropping Bombs game has come a long way since creating the simple cubes and spheres at the beginning of this book. In this activity, you'll make your game even more exciting with fire and explosions!

1 Before making additional changes to your project, it's important to make a backup of the entire game. In the **Projects** panel, make sure that the **Assets** folder is selected. Then click on the **Assets** tab at the top of the screen and select **Export Package**.

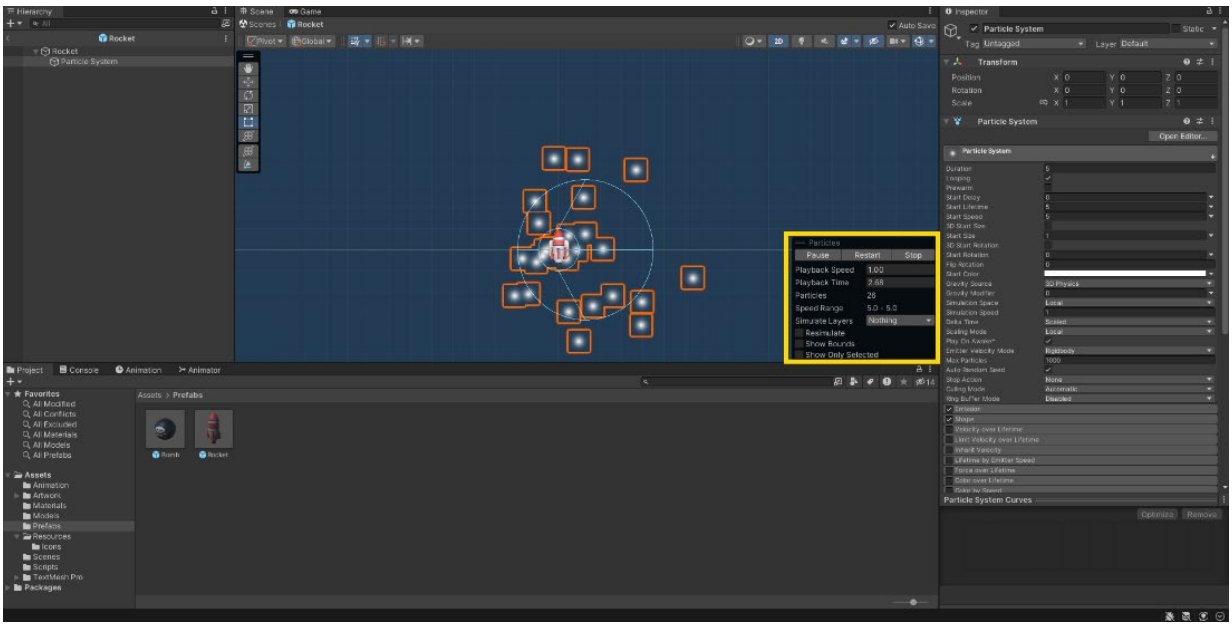
Make sure that all of the assets are selected before clicking the **Export** button. Give the package a name like ***JS-DroppingBombsPart3***.

2 Somehow, the rocket is flying without any thrust. Which might be great for the environment, but it certainly doesn't look like a rocket should. Let's add a particle system to simulate rocket thrust.

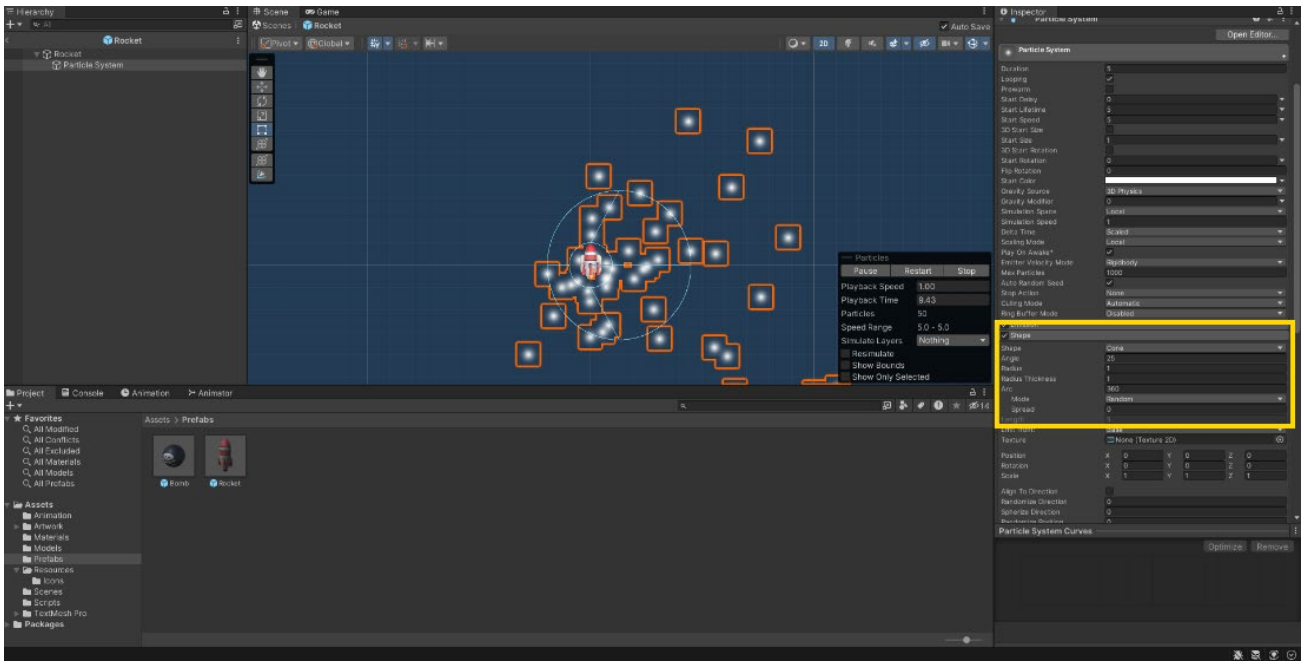
Find the **Prefabs** folder in the **Project** panel and click on the **Rocket** to edit it. In the **Hierarchy** panel, right-click on the **Rocket** and select **Effects**, then select **Particle System** to add it to the **Rocket**.



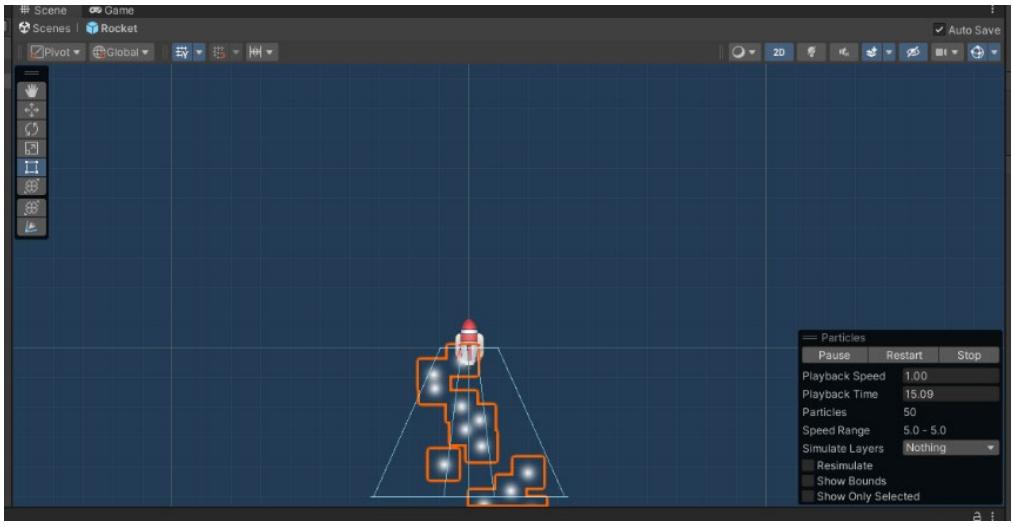
3 When added, the **Particle System** gets to work right away by showing default particles spreading away from the **Rocket**. In the corner of the scene, there's a control panel that allows you to control the particle system and provides you information on what the system is doing. Let's leave it alone for now.

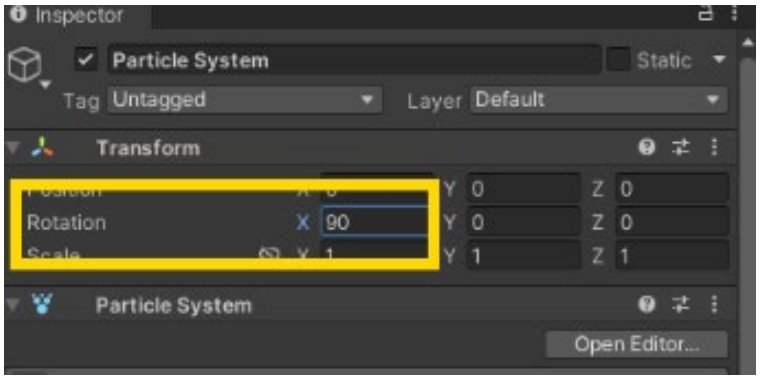


4 The particles seem random, but they're actually confined to a shape which you cannot see. In the **Inspector**, scroll down until you see the **Shape** menu and click on it to make it expand. Now you can see the cone that the particles are coming from. It appears that they're coming from the back side of the rocket instead of the nozzle.



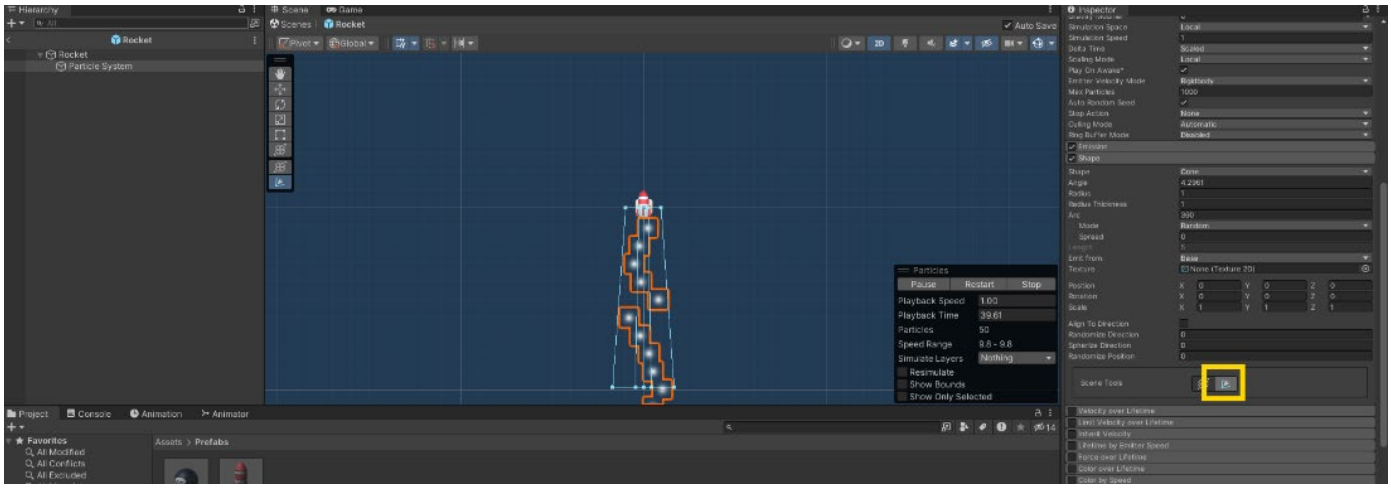
5 Let's rotate the cone so the particles emerge from beneath the rocket. In the **Transform** component, change the X rotation by **90** so that the cone is facing down. (You might need to use a rotation of **-90** or even **180** to get the desired effect.)



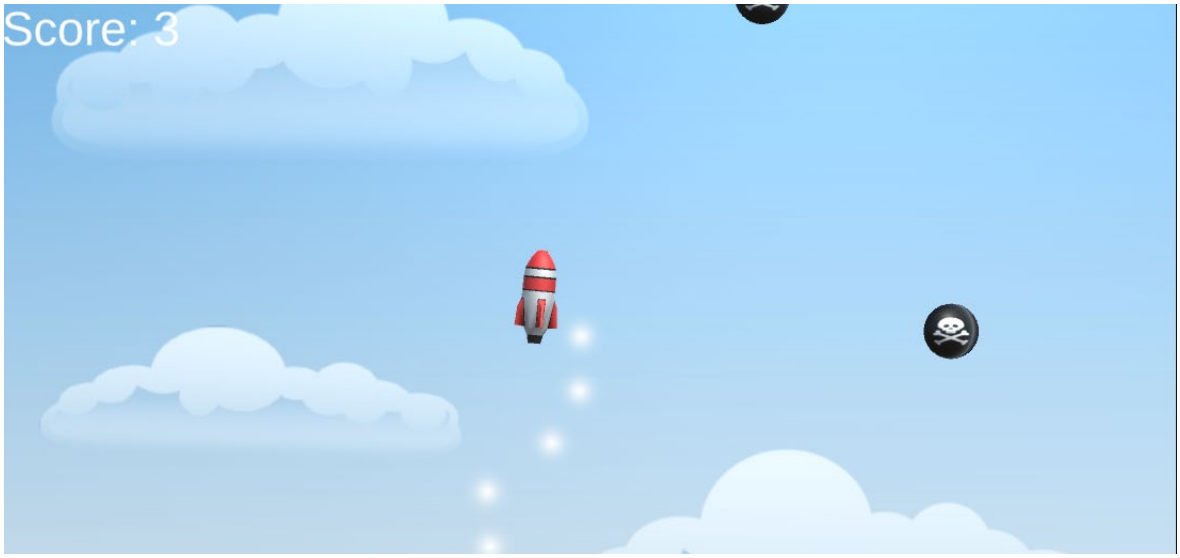


6 The cone seems a bit wide at the moment. Fortunately, at the bottom of the **Shape** component are tools to let you adjust it. The first icon lets you adjust the width at the top and bottom of the cone, while the others work just like the tools in the **Scene** editor. Use the tools to adjust the cone and move it so that it is at the bottom of the rocket as shown below.

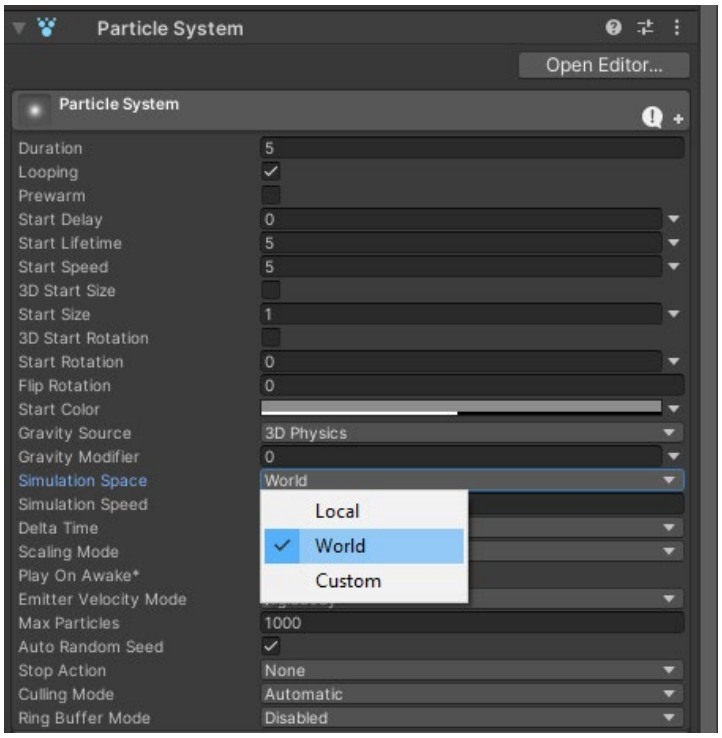
You want the button that looks like a cone being stretched!



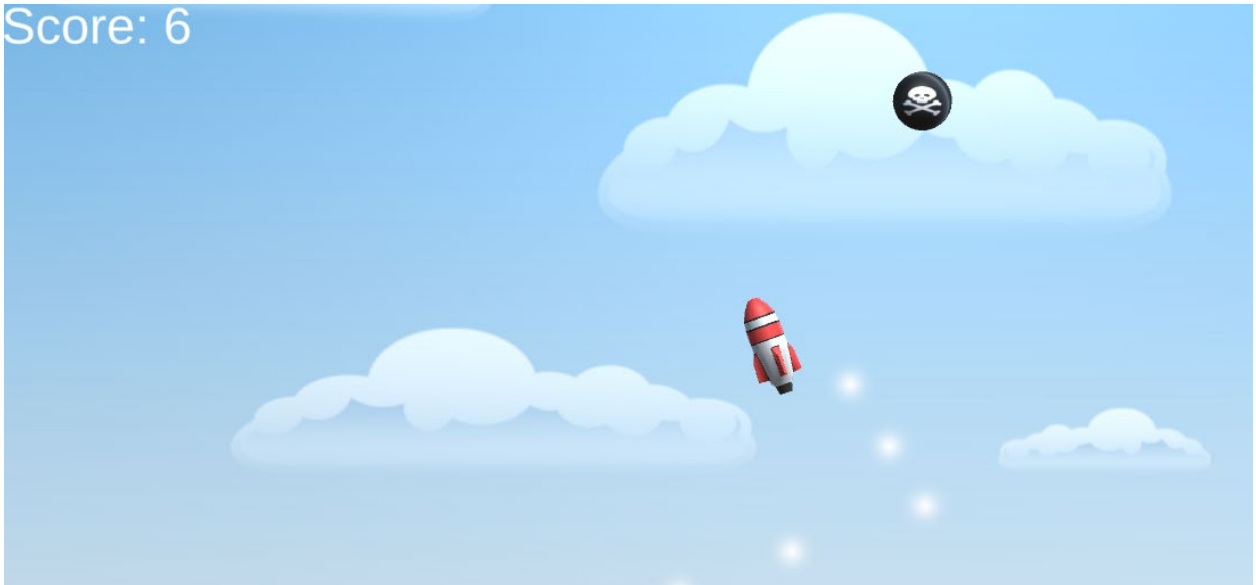
7 Click the **Play** arrow to see how the particles look in the game. They're coming from the rocket, but when the rocket moves, the whole particle system moves with it. Stop the game.



8 While we want the particles to originate at the rocket nozzle, we expect them to behave like they're part of the world that the nozzle sends them out into. In the Inspector, find Simulation Space and change it from Local to World.

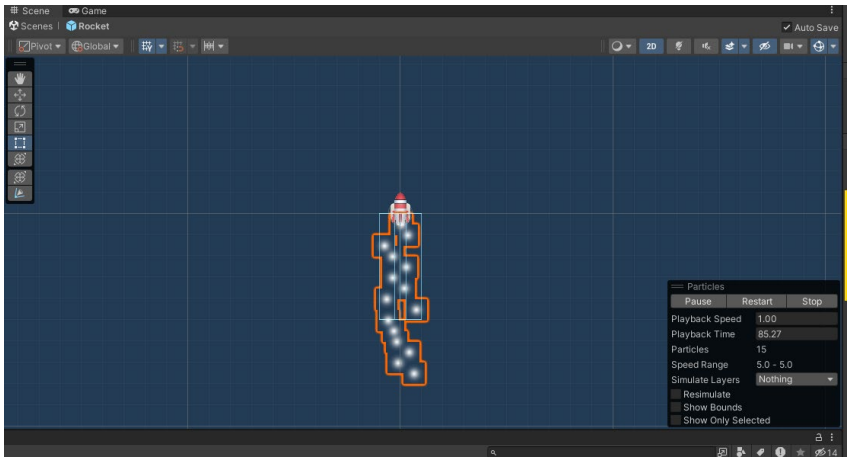


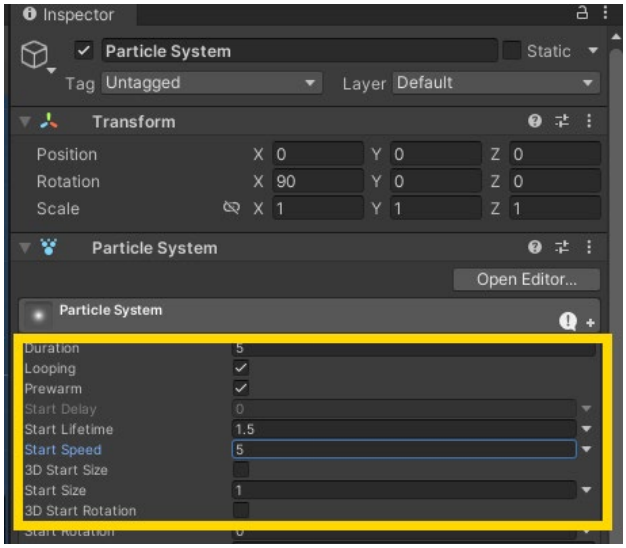
9 Start the game again. Now you can see that the particles behave more as expected. Stop the game before continuing to the next step.



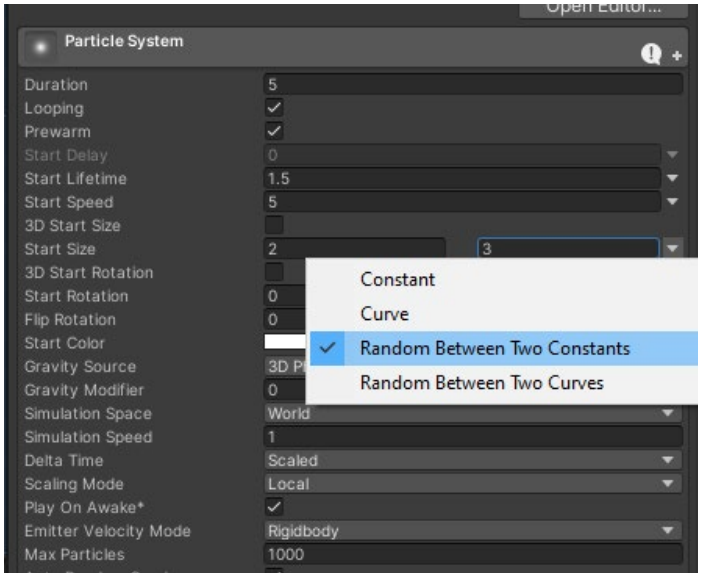
10 You may have noticed that it takes a moment for the particles to get started when the game begins. There is a setting called **Prewarm** that animates the particles as if they had been running for a while and not starting "cold." In the **Inspector**, make sure that the box for **Prewarm** is checked. Also, let's make the trail a little shorter by reducing the **Lifetime** of the particles.
Try **1.5**.

You may also wish to change other properties. If you haven't, set the **Start Speed** to **5.**



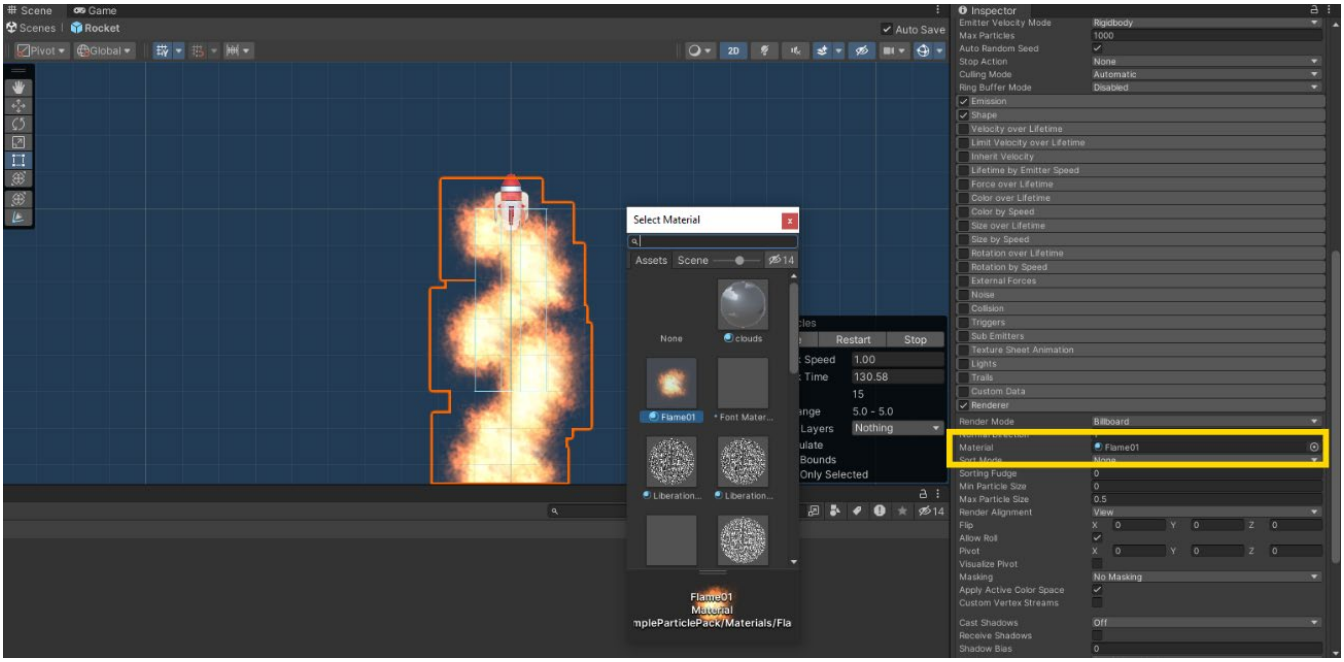


11 We can add a little variety to the particle by randomizing the size of the particles. Click on the triangle to the right of the slot for **Start Size** and choose **Random Between Two Constants** and pick a range of **2 to 3** as shown.

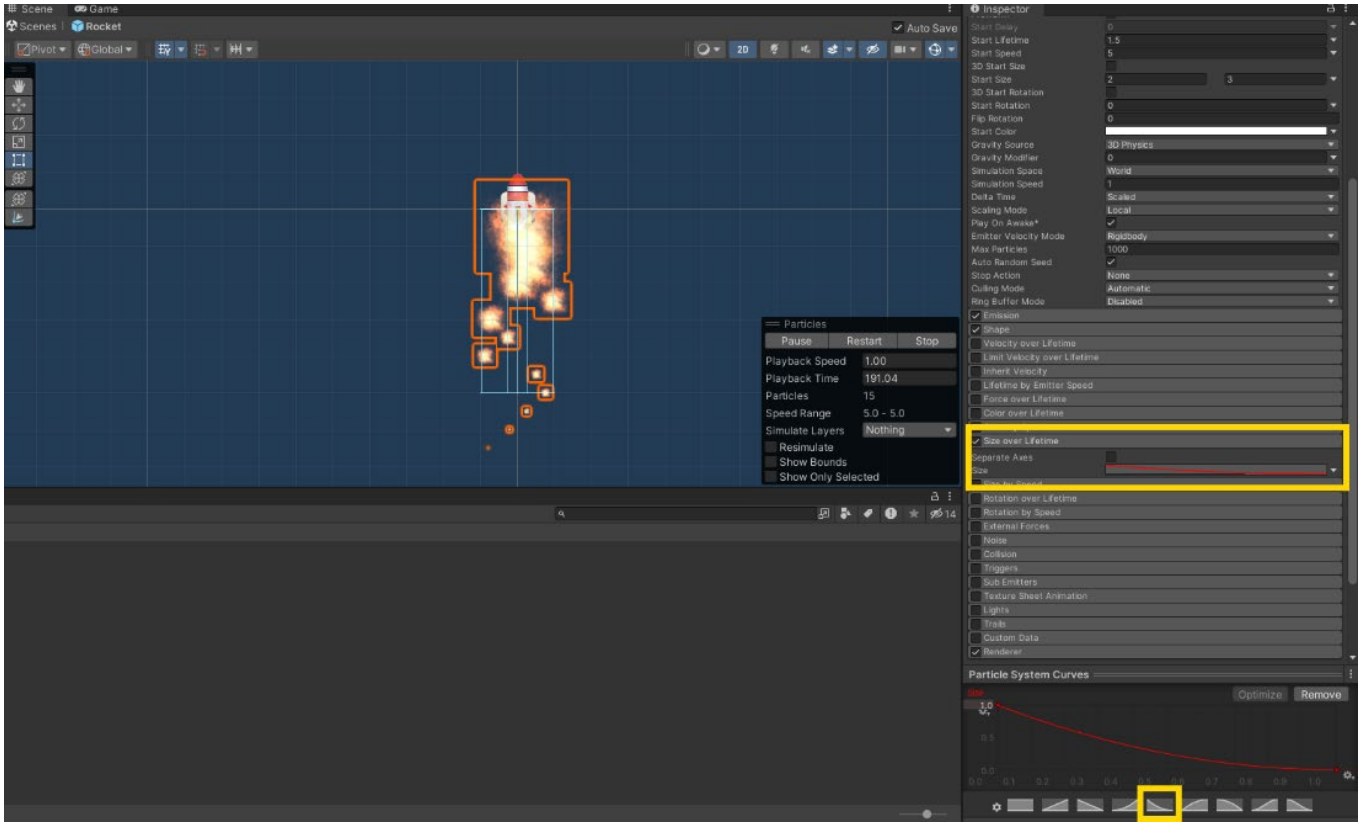


All of the properties in the Particle System have options for constant or curves just like with Start Size.

12 The rocket thrust is looking better, but still looks odd. Let's use a different particle. Click the **Assets** tab and select **Import Packages** and load **Activity 10 - Flame.unitypackage**. At the bottom of the **Particle System** component is a property called **Renderer**. Click on it to expand it and select a new **Material** by clicking on the circle to the right of the **Material** slot. From the menu that opens, select **Flame01** (which you just imported).

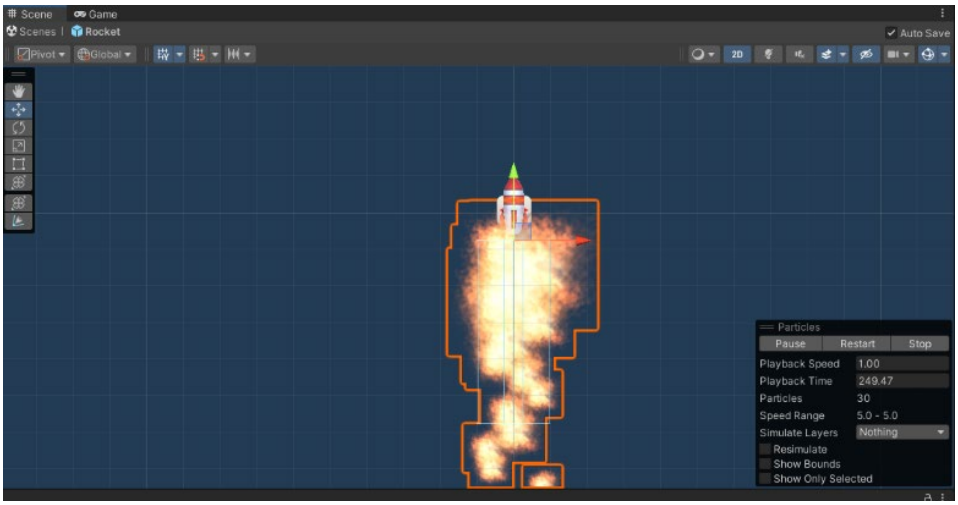


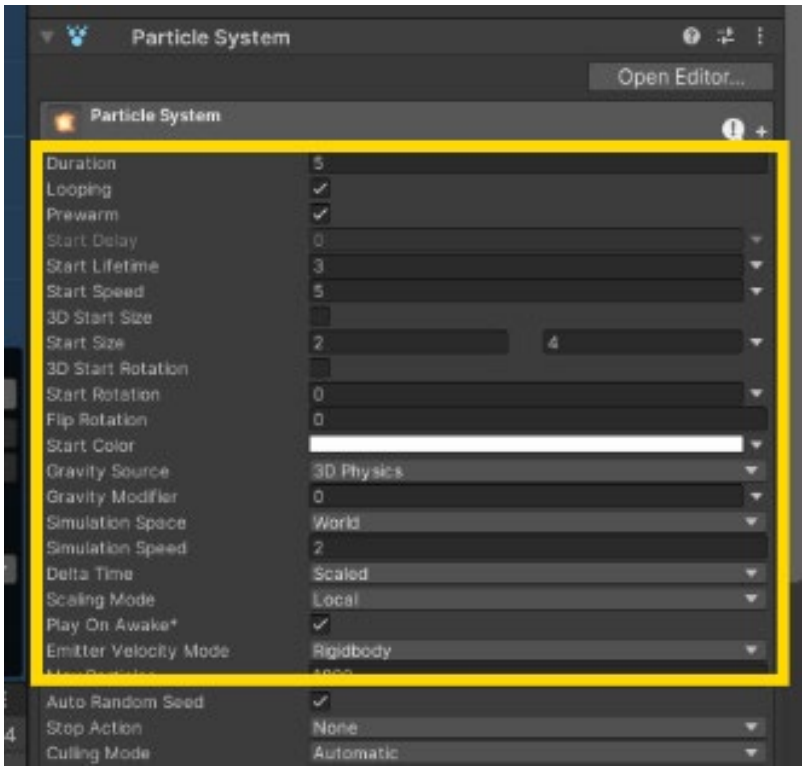
13 In reality, the rocket thrust would taper off as the flames got farther and farther from the fuel source. The **Particle System** has a component to simulate that. Look for the component called **Size Over Lifetime** and check the box to enable it. Then click on the window next to **Size** to access the curves for that component. This opens a new window at the bottom of the **Inspector**. Select a curve that starts high and fades to nothing.



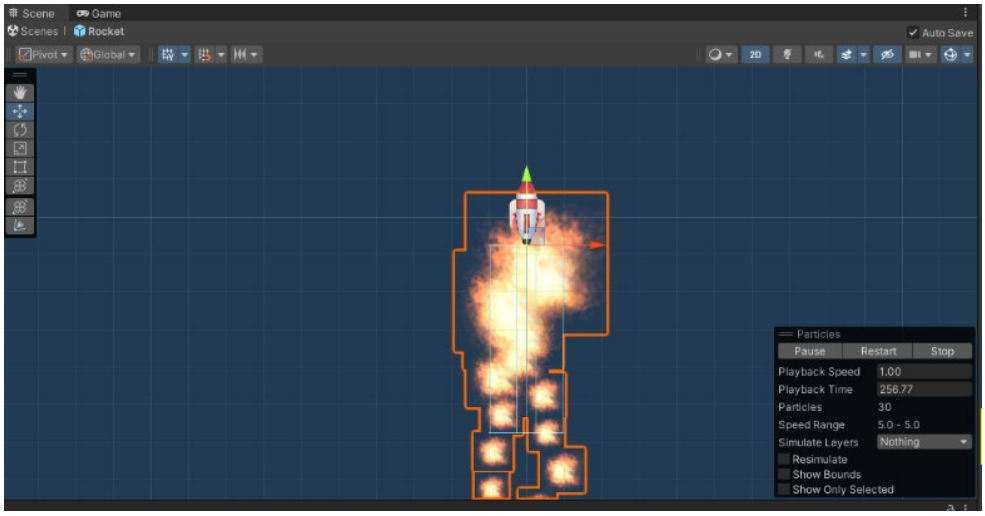
14 Let's adjust some of the numbers to get the thrust where we want it. If we want a longer thrust, increase the **Start Lifetime**. If we want it to be denser, decrease the **Start Speed**. If we want a more powerful flame, increase the **Simulation Speed**.

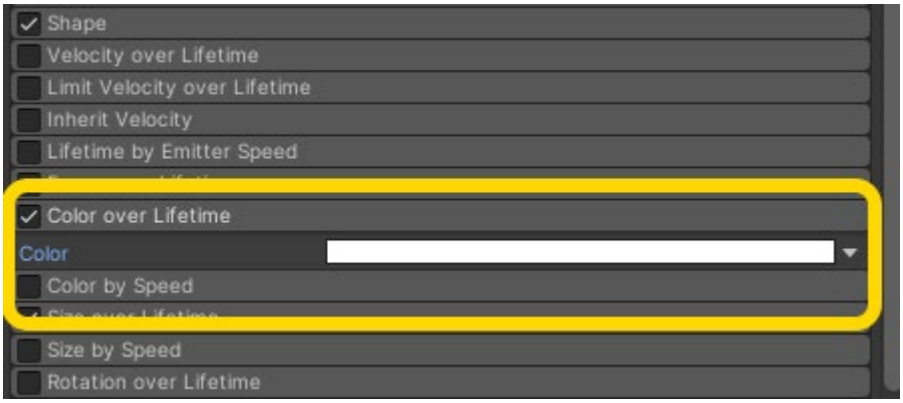
You can use whatever numbers you want, see below for an example.



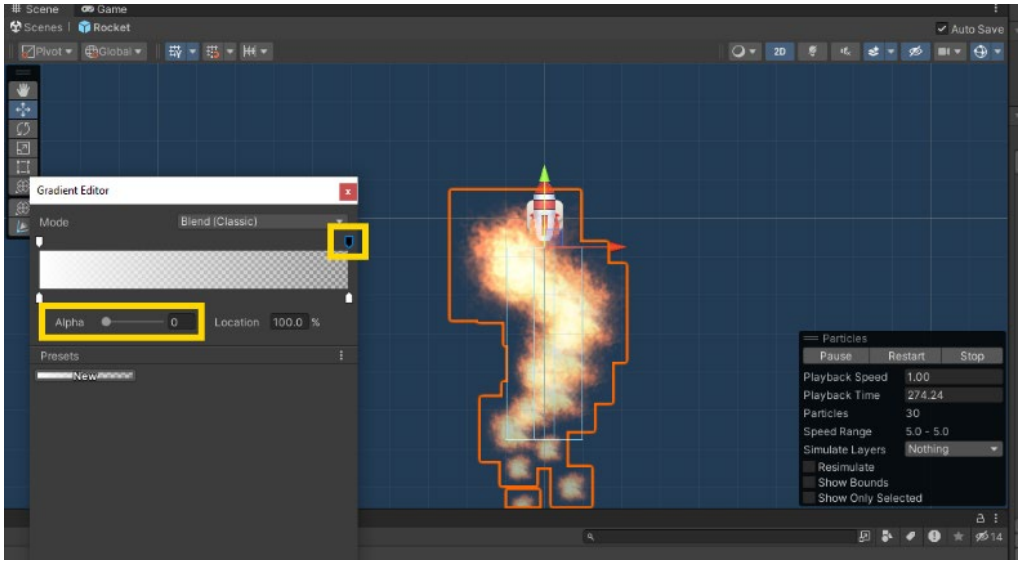


15 The flames abruptly vanish when they reach the end of their lifetime. Instead, let's make them more transparent over time. In the **Inspector**, find **Color Over Lifetime** and check the box to enable it. Then click on the **Color** window.





16 Clicking on the color opens a new window for adjusting color. It works a lot like the timeline we used for the animations, beginning at the left and ending on the right. If we wanted to, we could have the particle gradually change color over time. Right now, we just want to change the transparency which is controlled by the arrows at the top. The left arrow is fine as it is. Let's click on the arrow at the top right. The panel at the bottom says the alpha transparency is 100% (completely opaque). Change it to 0 so that the color is completely transparent at the end.

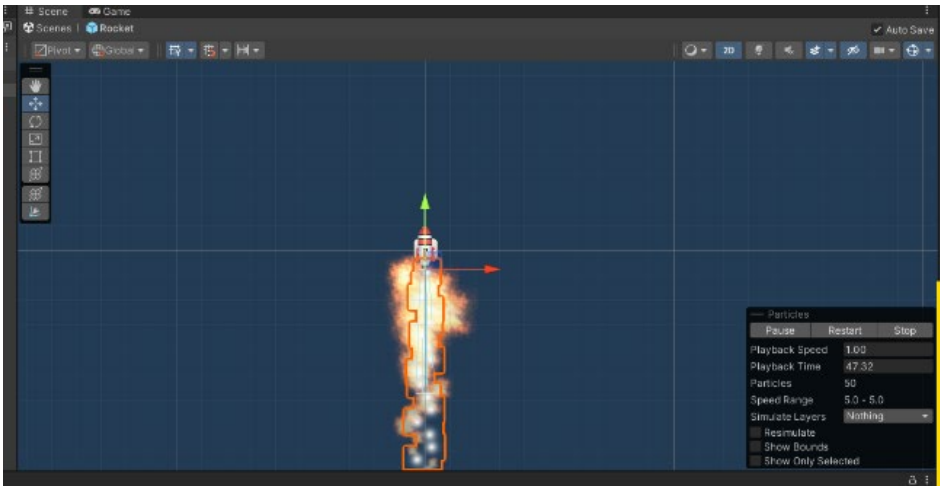


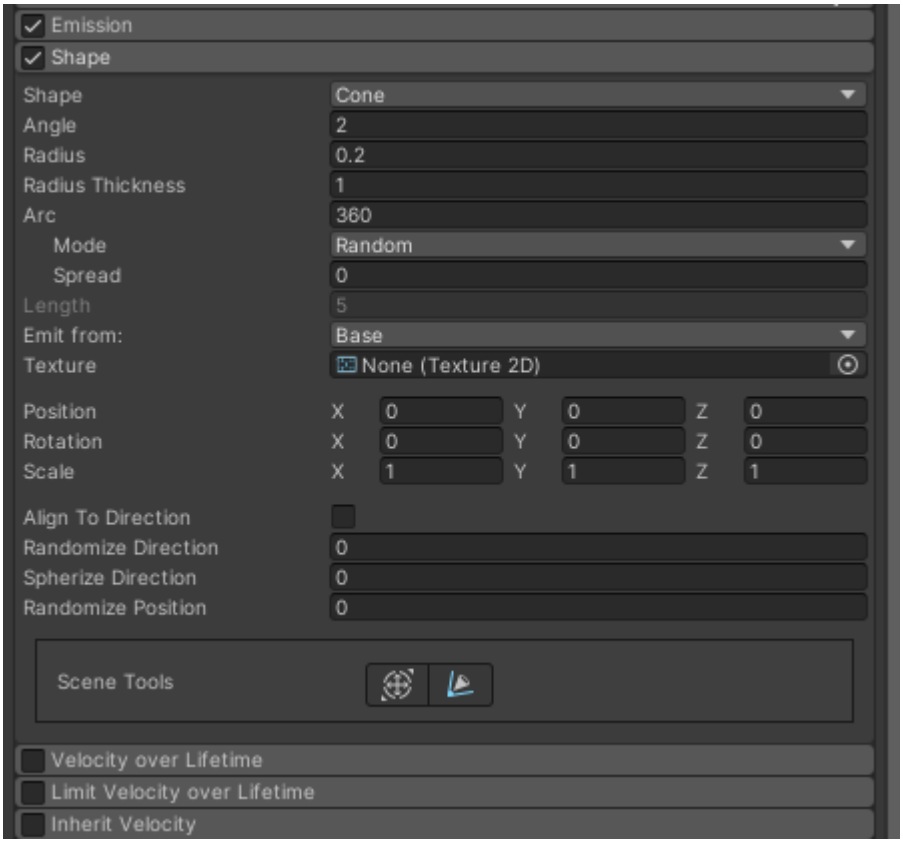
17 Let's test the thrust by playing the game. Not too bad, right? Though real rocket thrust would have some smoke. Stop the game before continuing.

18 In the **Hierarchy**, right-click on the **Particle System** to add a new particle system inside the first one (select Effects and click on **Particle System**). See the Pro Tip below if the new Particle System isn't playing.

Sometimes, when adding additional Particle Systems, the new system doesn't seem to work. The control panel shows 0 particles being emitted. When this happens, restart or pause the particle system and start it again.

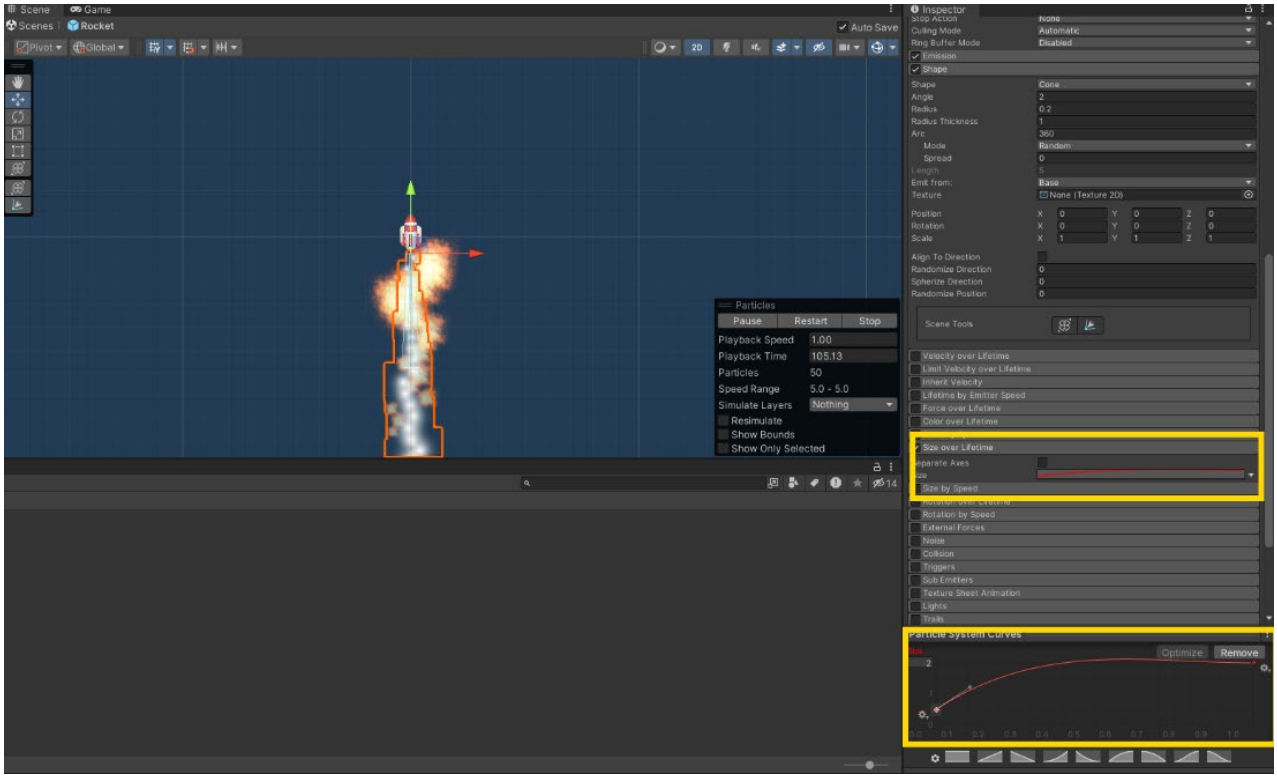
19 The shape of the smoke should be similar to the flame. Click the **Shape** component and use the tools at the bottom to adjust the shape of the cone. Then move it so that it starts a bit further away from the rocket nozzle than the flames.



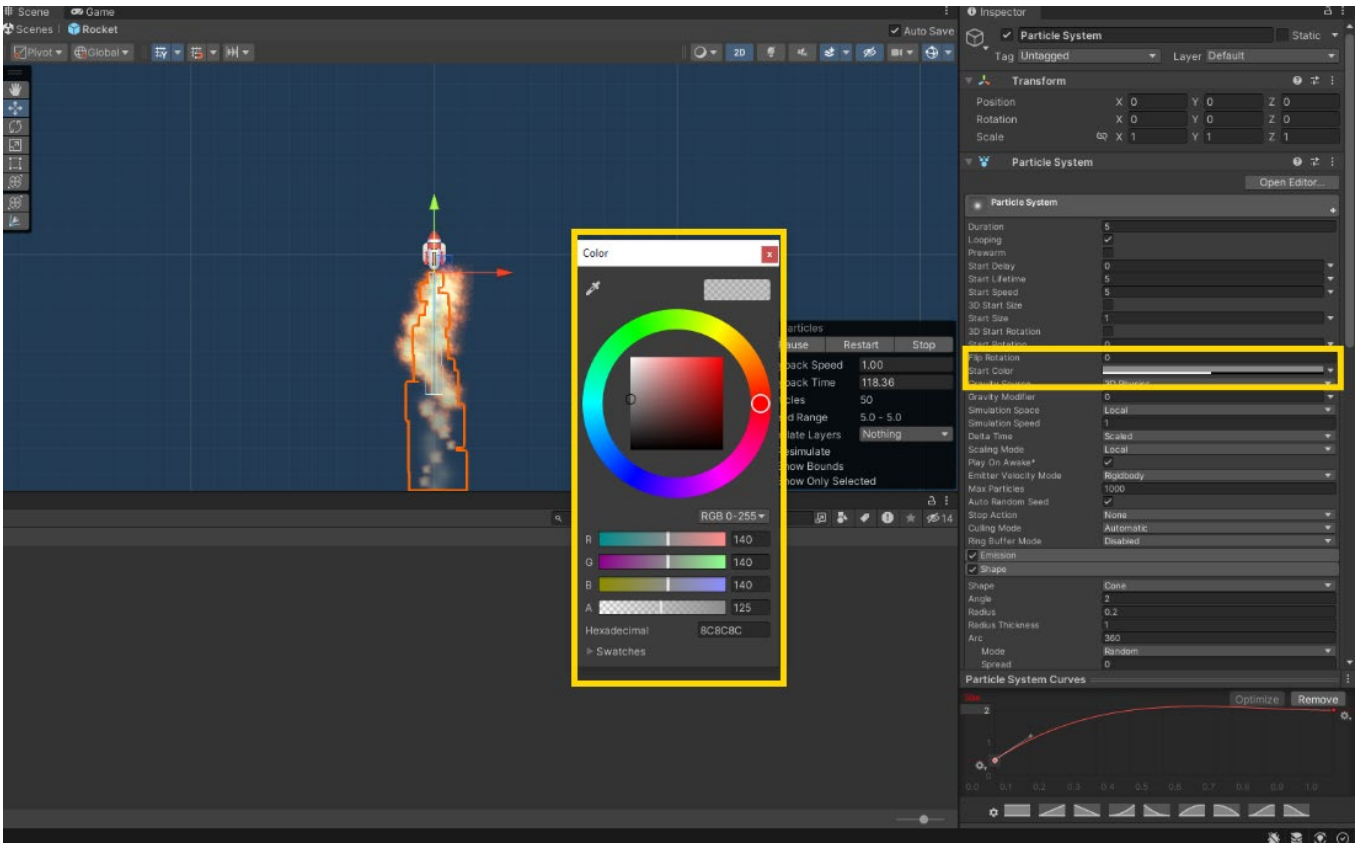


20 Over time, smoke spreads out. So go to the **Size Over Lifetime** component and enable it. Choose a graph that starts small and grows. To ensure that it starts out at a reasonable size, click and drag the left point of the curve so that it's about a third of the way up the scene.

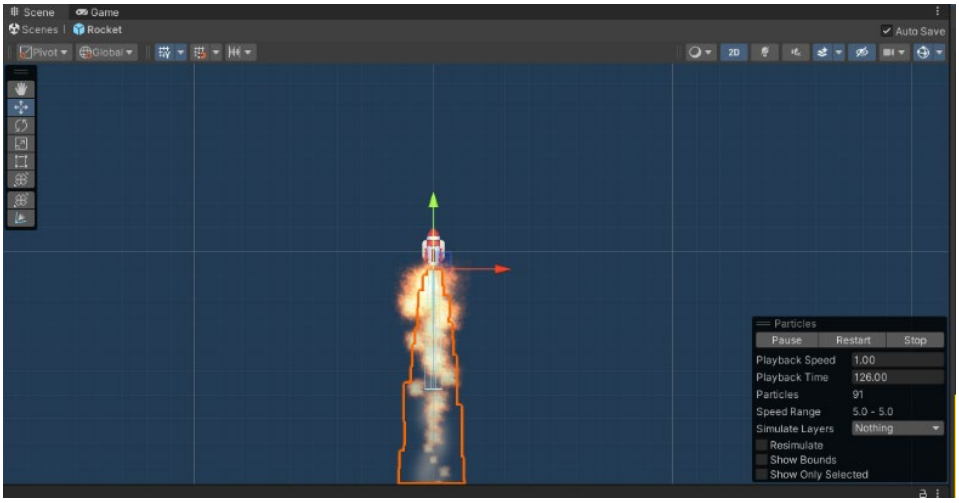
Your curve doesn't have to look exactly the same; you may wish to use a curve that you think looks better.

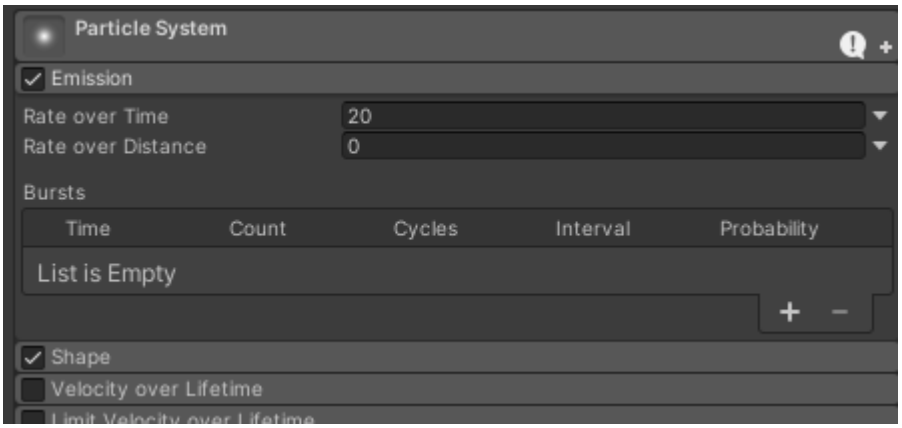


21 Now let's adjust the color of the smoke. Find **Start Color** in the component and click on the white rectangle to open the **Color** editing window. Drag the selector in the square to get a light grey color. Let's also make it a bit transparent by dragging the Alpha slider ("**A**") a bit to the left.



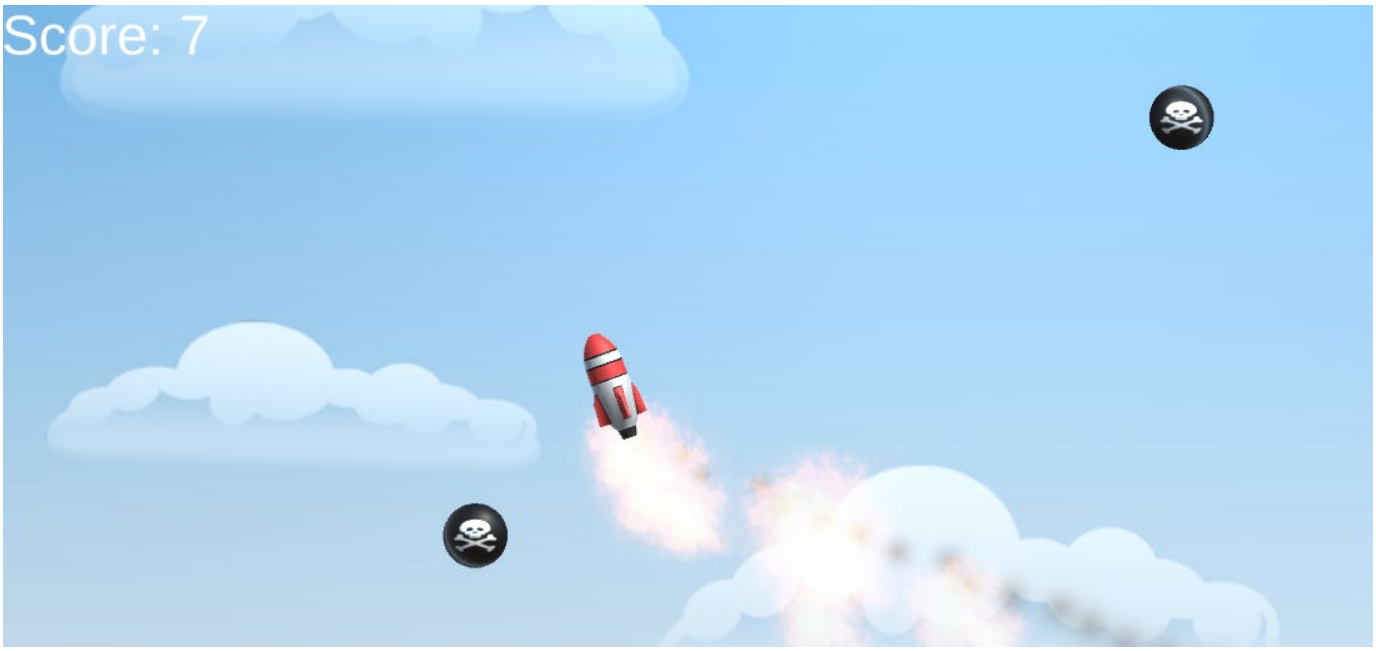
22 We can increase the smoke by selecting the **Emission** component and increasing the value of **Rate Over Time**. Let's double it.



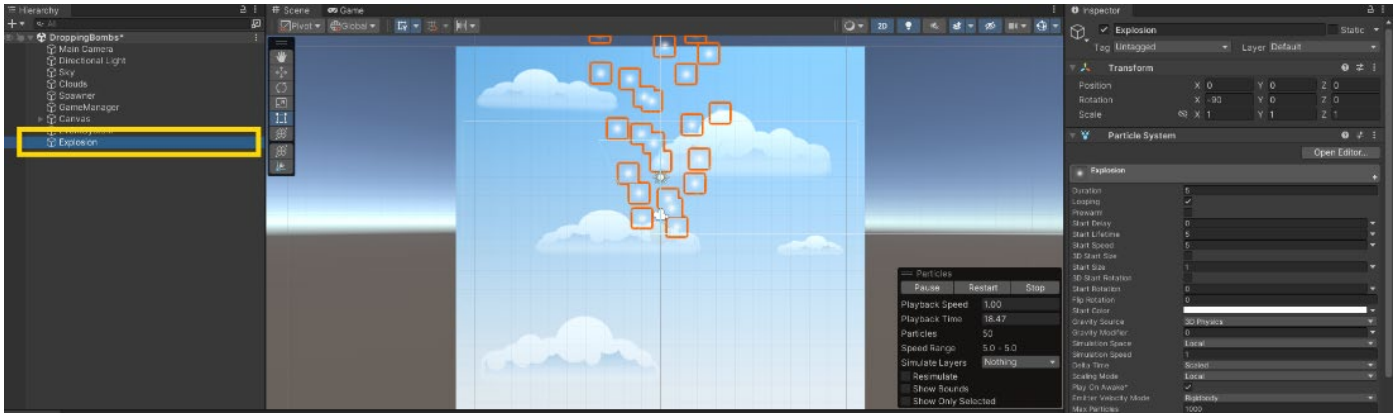


23 Play your game to see how the Rocket Thrust effect looks. Do you want more fire? Darker smoke? Adjust the Particle System to get the results that you like. Don't forget to update the Simulation Space to World.

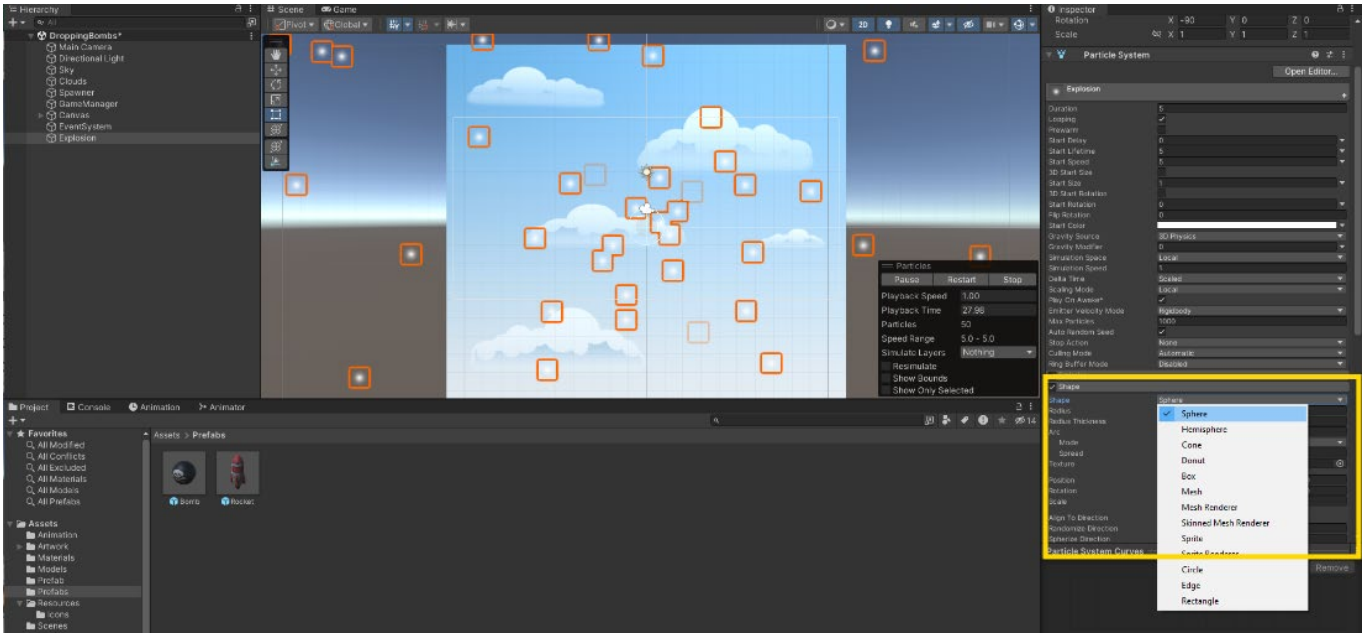
Stop the game and save your project before continuing.



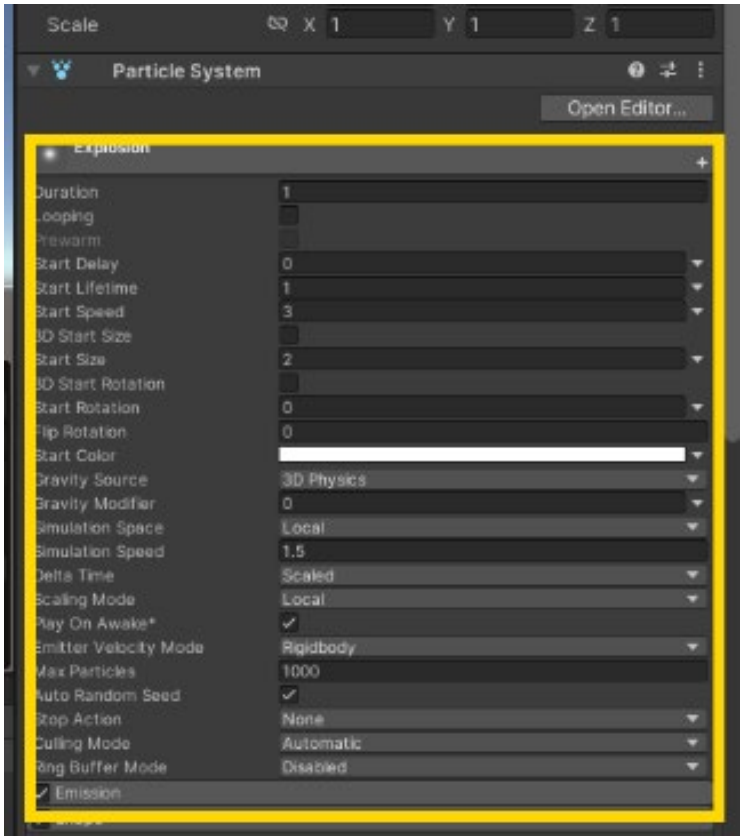
24 What the game needs now is something to happen when the Rocket collides with a bomb. In the **Hierarchy**, click **Create**, select **Effects** and then select **Particle System**. Name it ***Explosion***.



25 The default shape of the new **Particle System** is a cone, but that's not right for an explosion. In the **Shape** component, go to the **Shape** menu and select **Sphere**.

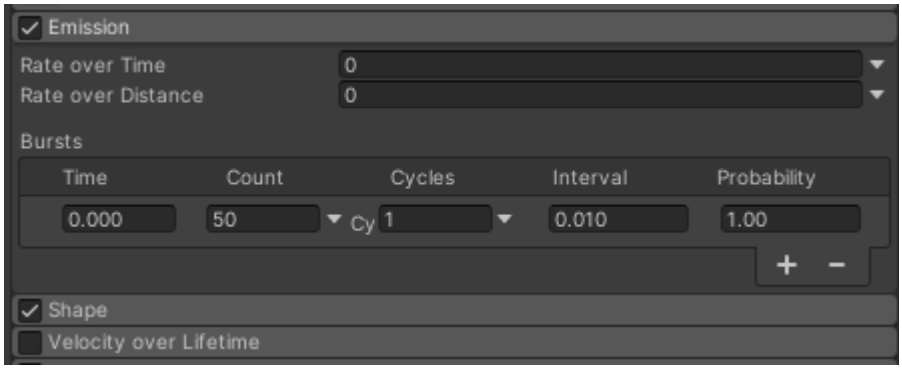


26 The explosion should only play once. Go to the top of the **Particle System** component and uncheck the box for **Looping**. Change the settings for **Duration**, **Start Lifetime**, **Start Speed**, **Start Size** and **Simulation Speed** as shown below.

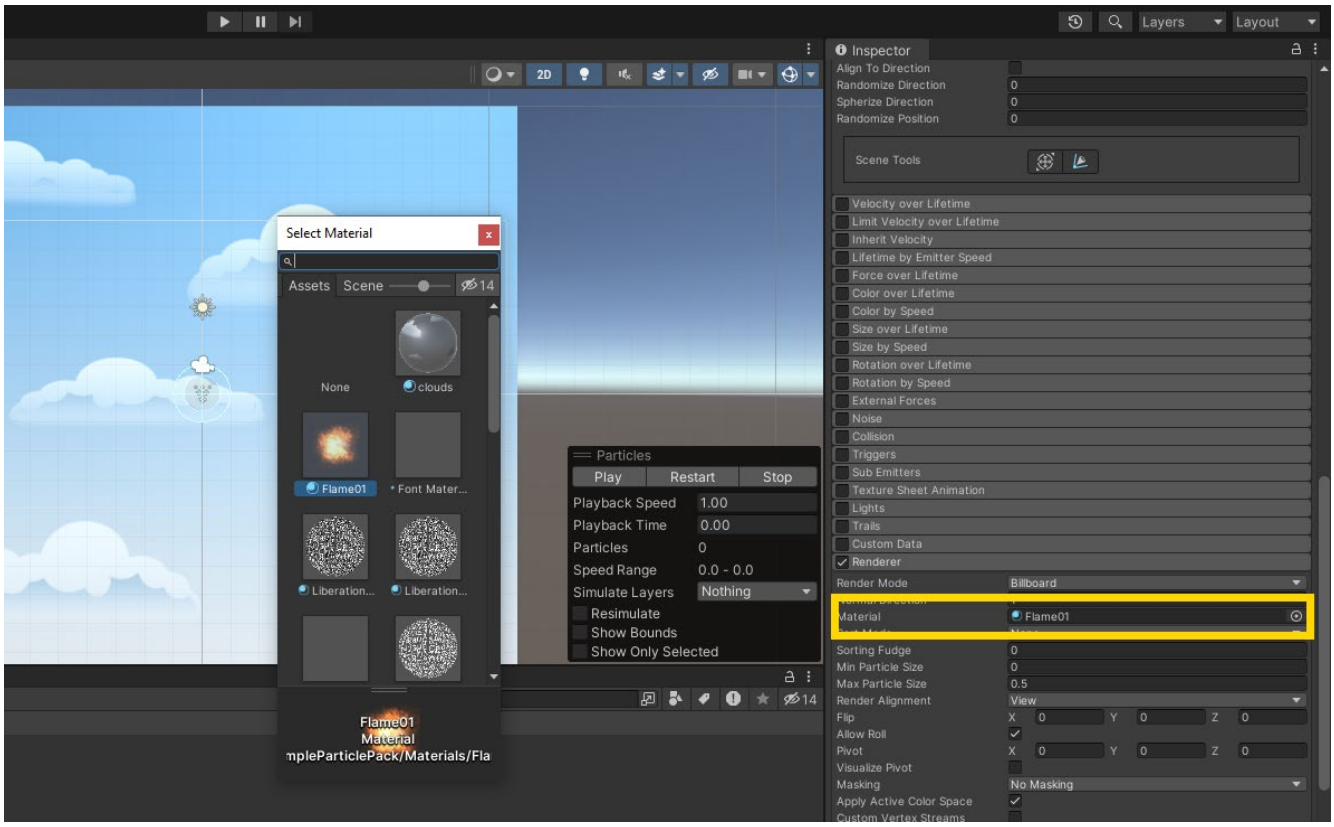


With Looping off, the Particle System no longer plays constantly in the Scene panel. From now on, you will need to press Play in the Particle System control panel when you want to preview this Particle System.

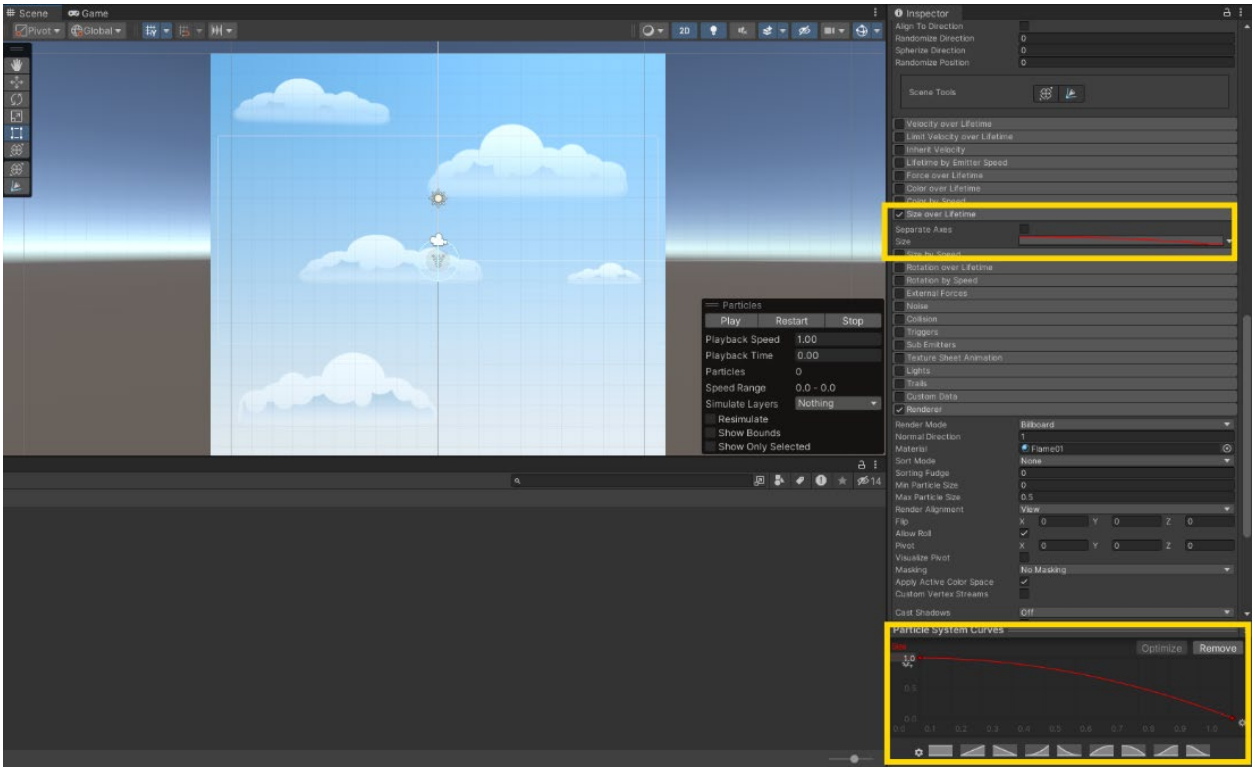
27 At the moment, the explosion seems a bit weak. Go to the **Emission** component, but instead of adjusting the **Rate Over Time**, add a **Burst**. Click on the plus (**+**) symbol at the bottom of the **Bursts** panel to add a **Burst**. Keep all of the default settings as they are but increase **Count** to **50** and reduce the rate over time to **0**.



28 Go to the bottom of the **Particle System** component and expand the **Renderer**. Select a new **Material** by clicking on the circle to the right of the **Material** slot. From the menu that opens, select **Flame01** (the same as what you used for the Rocket Thrust).



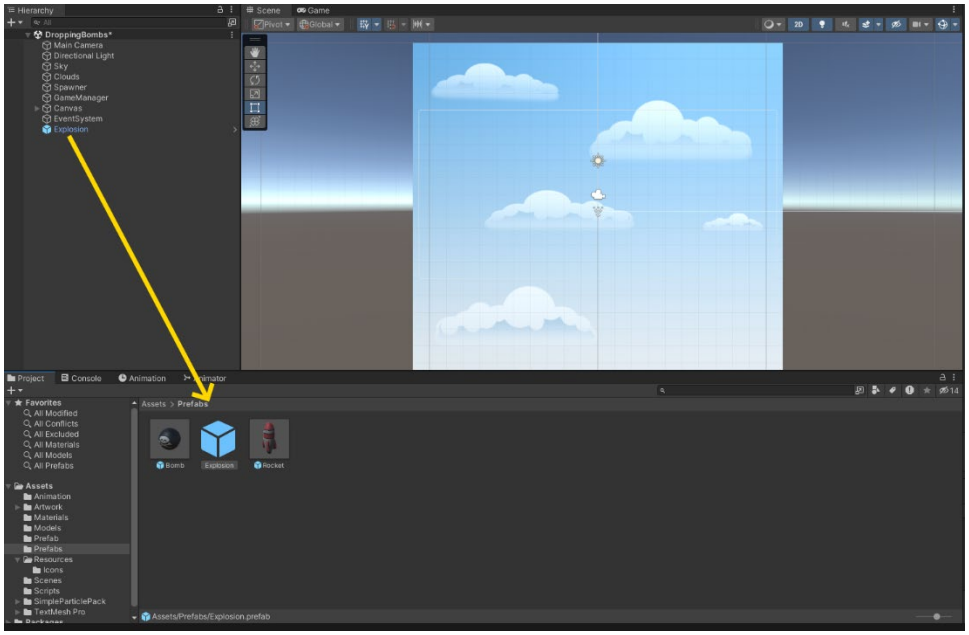
29 The explosion particles should get smaller over time. Find the **Size Over Lifetime** component and enable it. Then create a curve that starts out large and eventually dwindles to nothing.



30 Just as you did with the rocket thrust, select the **Color Over Lifetime** component and enable it. Click on the **Color** rectangle to edit the color. Click the arrow on the top right of the color timeline and set the **Alpha** to **0**.

Let's save this setting so that from now on we can just load it when we want to change Color Over Time. Just click the **New** button and the next time you need this. Click on the **Preset** you just made.

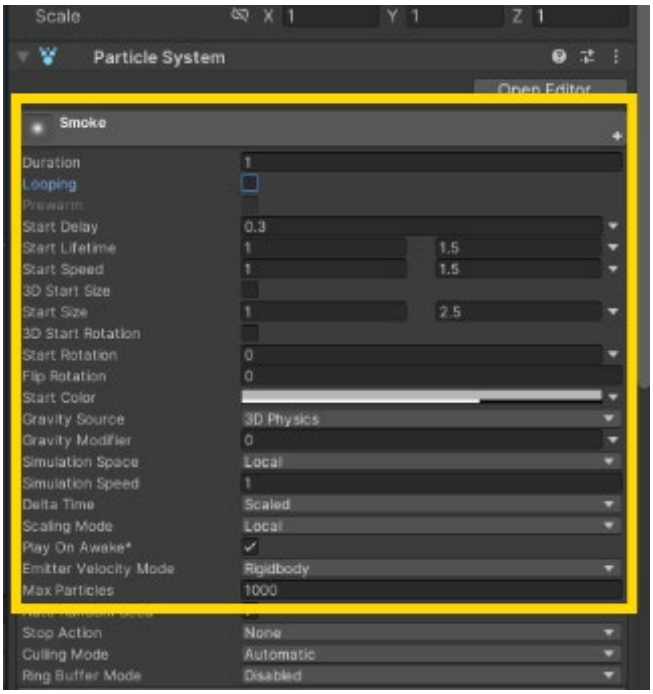
31 Before going any further, let's drag the **Explosion** object into the **Prefabs** folder. Double-click on the **Explosion** prefab to edit it.



32 Just as you did with the Rocket thrust, right-click on the Explosion to add another Particle System and name it ***Smoke***. Turn off looping and change the **Duration** to **1**. Change the **Start Delay** to **0.3** so that the smoke follows a little bit after the flames.

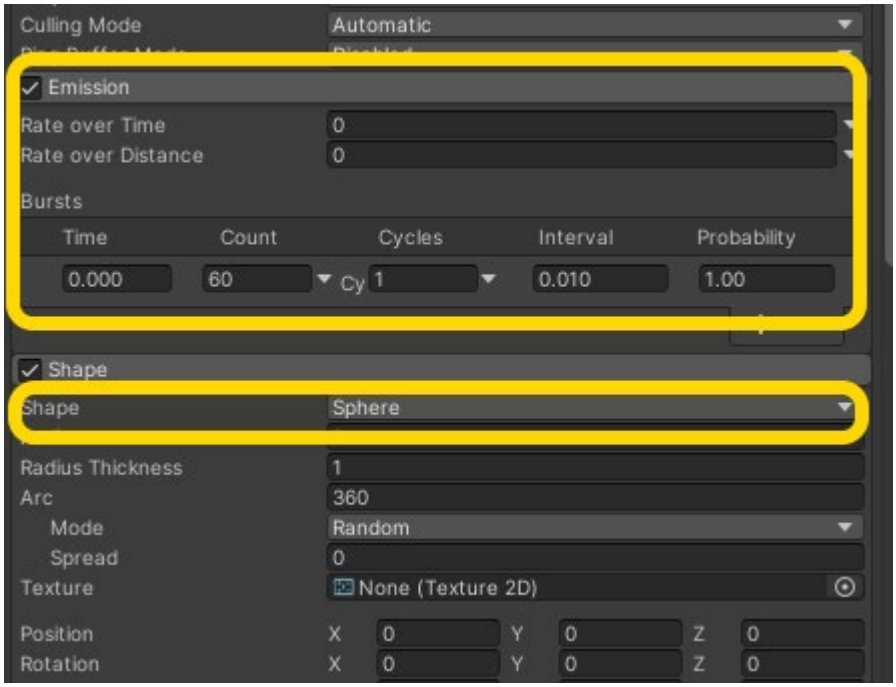
Add random between two ranges for the **Start Lifetime**, **Start Speed** and **Start Size** as shown below.

Change the color of the smoke to a medium grey with an **Alpha** of about **75%**.

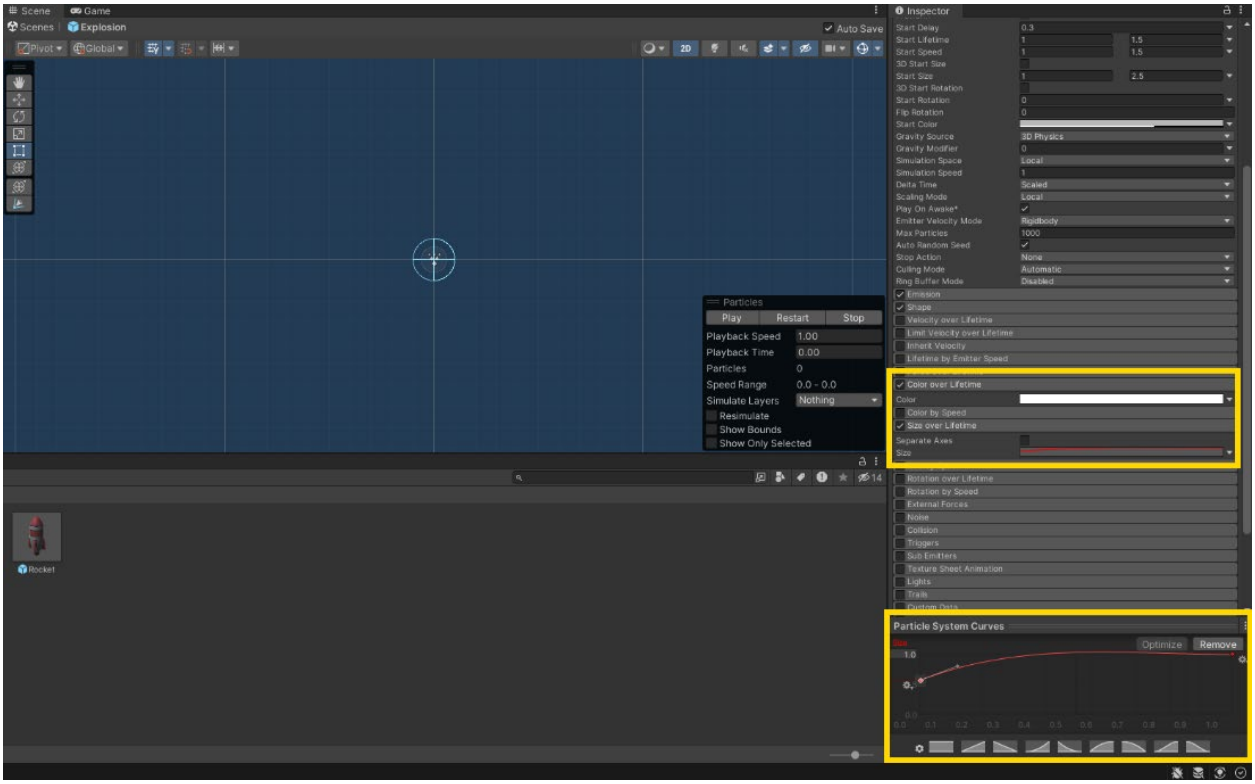


33 Just like the **Explosion**, the **Smoke** will also have a **Burst**. Expand the **Emission** component and change the **Rate Over Time** to **0**. Add a **Burst** to the **Bursts** panel by clicking on the plus (**+**) symbol in the lower right corner. Change the **Count** to **60**.

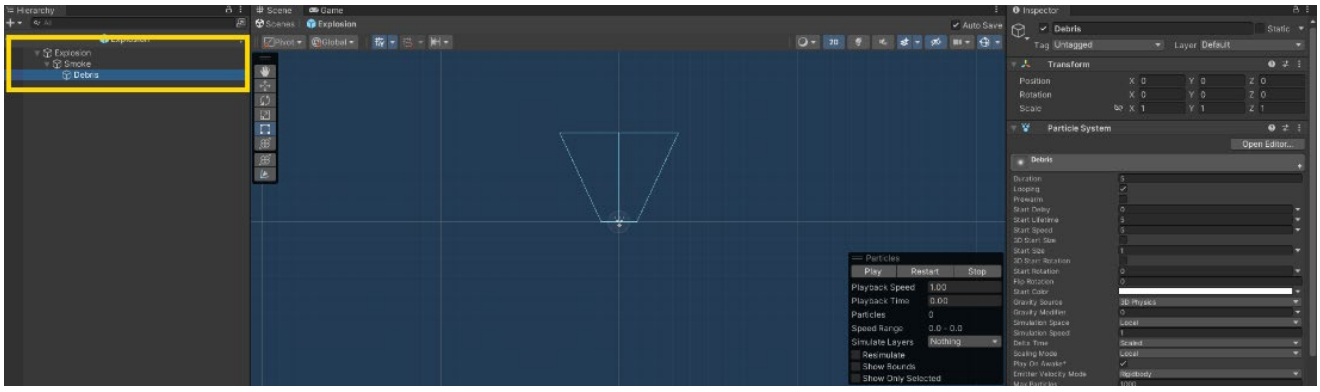
Expand the **Shape** component and change the **Shape** from a **Cone** to a **Sphere**.



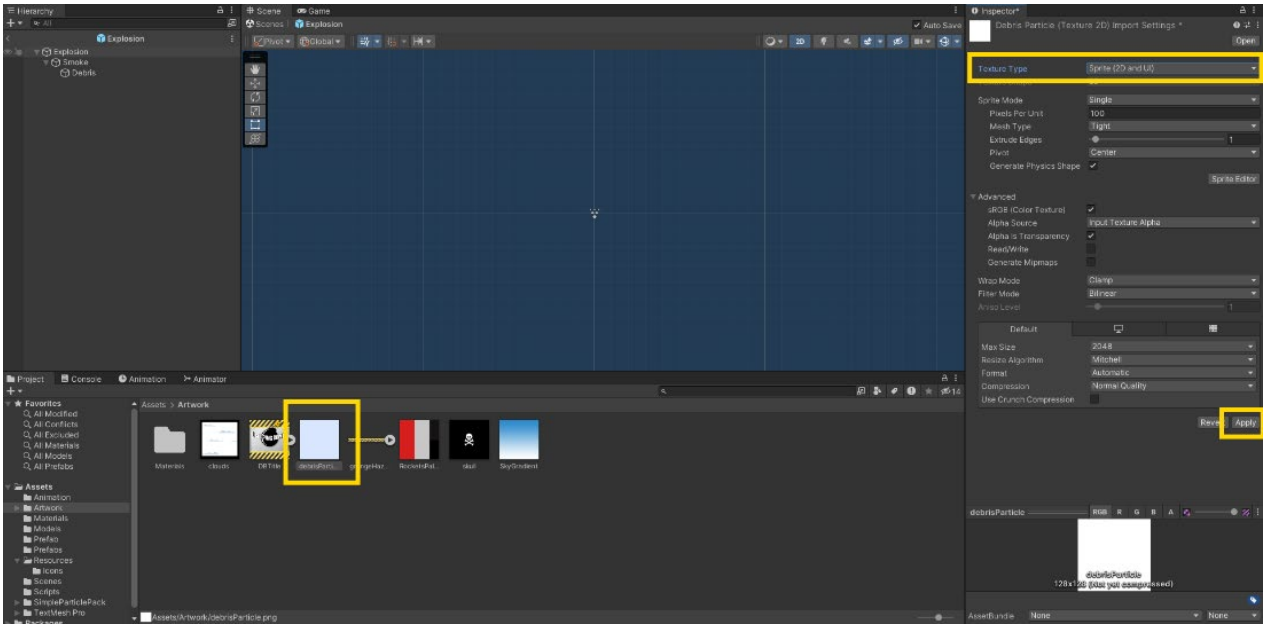
34 Enable **Color Over Lifetime** to add the preset gradient that you recently made to have the smoke fade away. Enable **Size Over Lifetime** to give a gradually increasing curve to the size of the smoke. Save your progress.



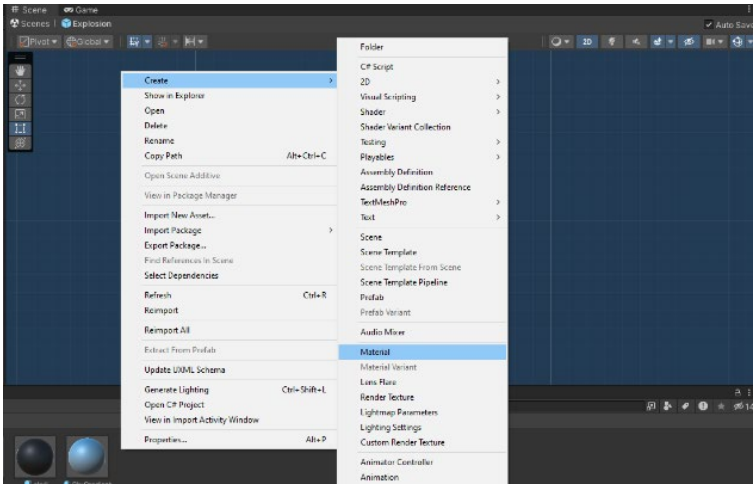
35 Right-click on the **Smoke Particle System** to add a third **Particle System** and call it ***Debris***.



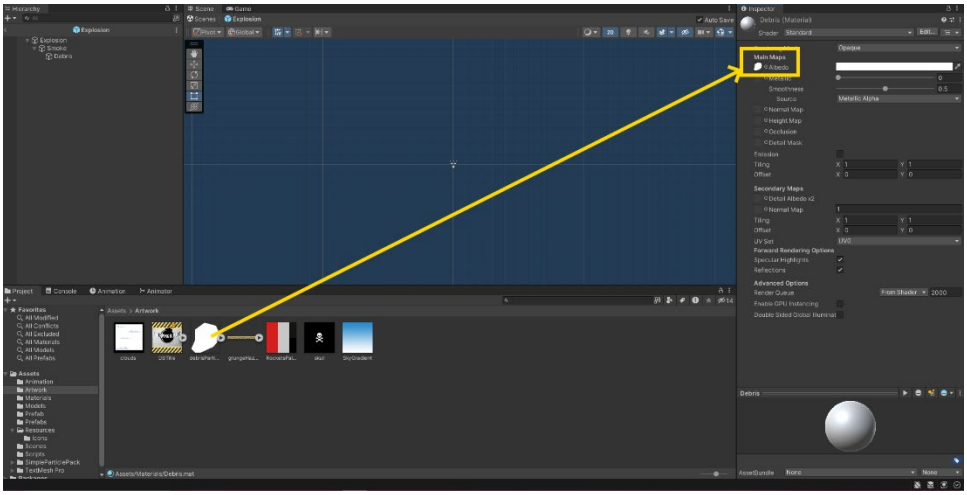
36 We need a new particle for this and this time, we're going to make it ourselves. In the **Project** panel, open the **Artwork** folder. Right-click inside the folder and select **Import New Asset** and select **Activity 10 - debrisParticle.png**. After the image is loaded, select it and then go to the **Inspector** panel where you'll change the **Texture Type** to **Sprite (2D and UI)**.



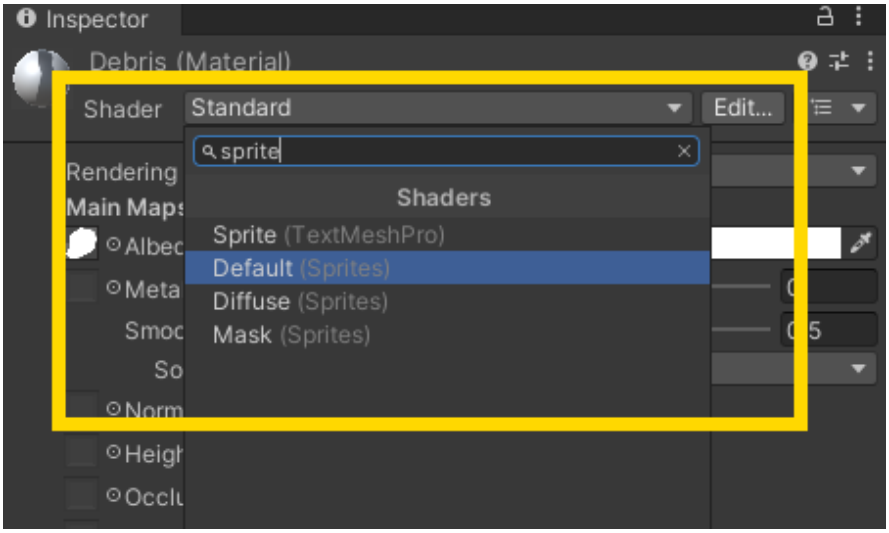
37 The **Particle System Renderer** needs a **Material**, so we need to create one for the **Debris** image. Go to the **Materials** folder and right-click inside the folder. Select **Create**, then **Material**, and name the Material ***Debris***.



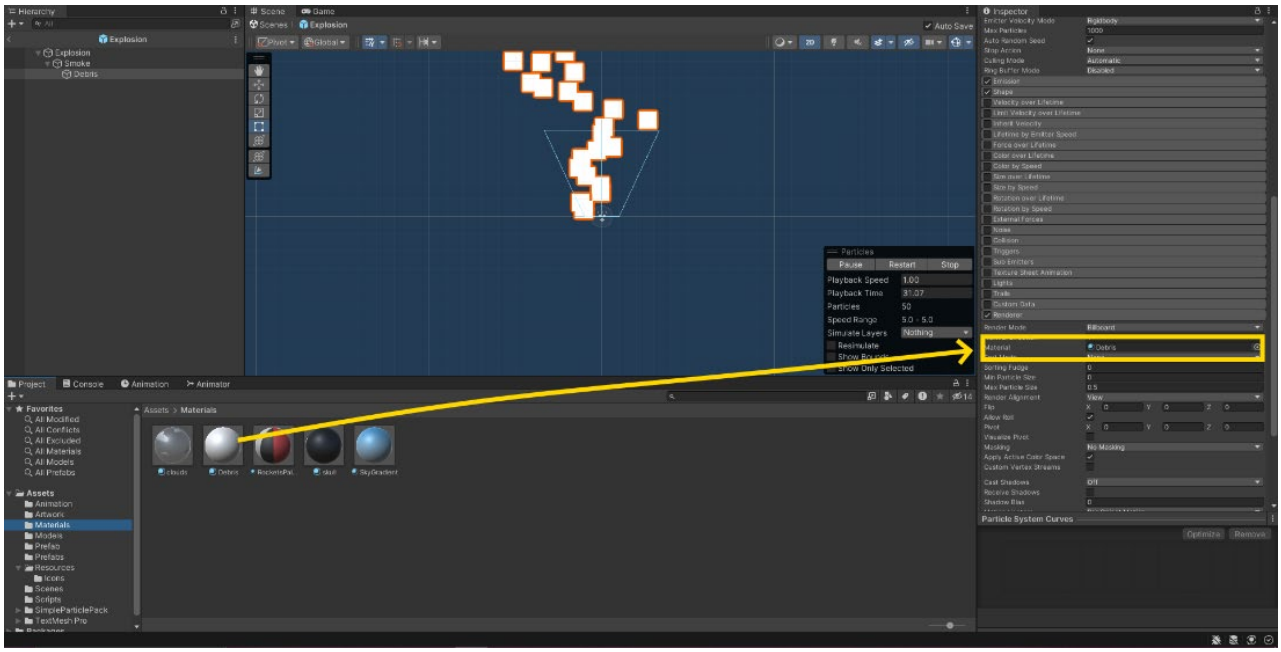
38 To add the image to the **Material**, simply drag it from the **Artwork** folder into the box for **Albedo** in the **Material Shader**.



39 The debris particles need to be rendered as sprites, so you'll need to change the shader. In the **Inspector**, click on **Shader** and search **Sprites then Default** from the menu.



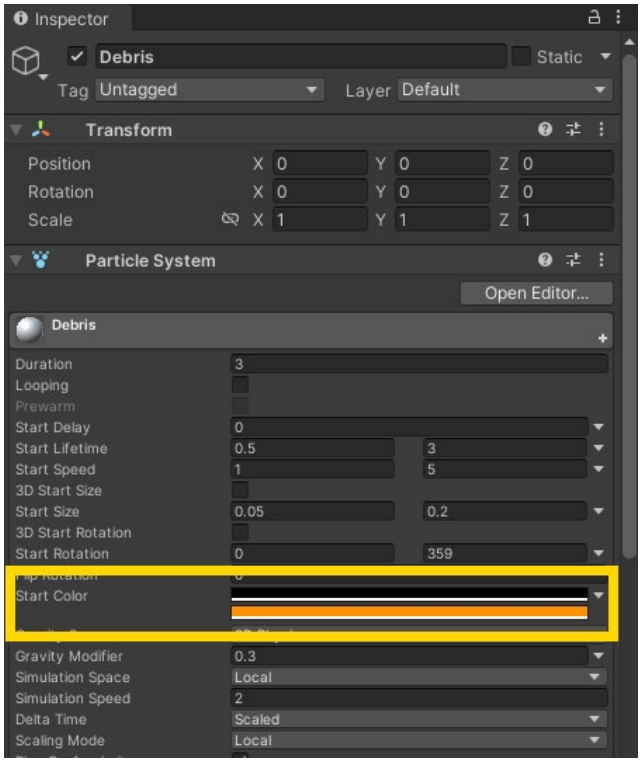
40 Go back to the **Explosion** in the Prefabs folder. Select the **Debris Particle System** and drag the **Debris** material from the **Materials** folder into the **Material** slot in the **Particle System Renderer** component.



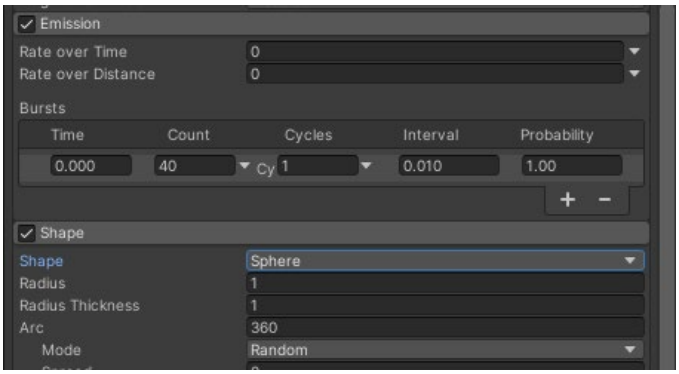
41 By now, you should be well acquainted with adjusting the settings for a **Particle System**. We want the debris to be little bits that fly out from the explosion and fall to the ground, so this time we are applying gravity. The following settings worked best for us.

- Duration: 3.0
- Looping: off
- Start Lifetime: 0.5 - 3
- Start Speed: 1 - 5
- Start Size: 0.05 - 0.2
- Start Rotation: 0 - 359
- Start Color: see below
- Gravity Modifier: 0.3
- Simulation Speed: 2

Click on the arrow to the right of the color window to select Random Between Two Colors and choose black and orange.

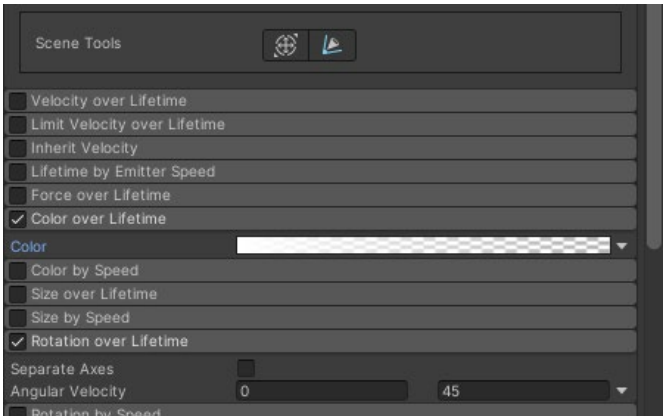


In Emission, set Rate Over Time to 0 and add a Burst with the count of 40. In Shape, change the Shape to Sphere.

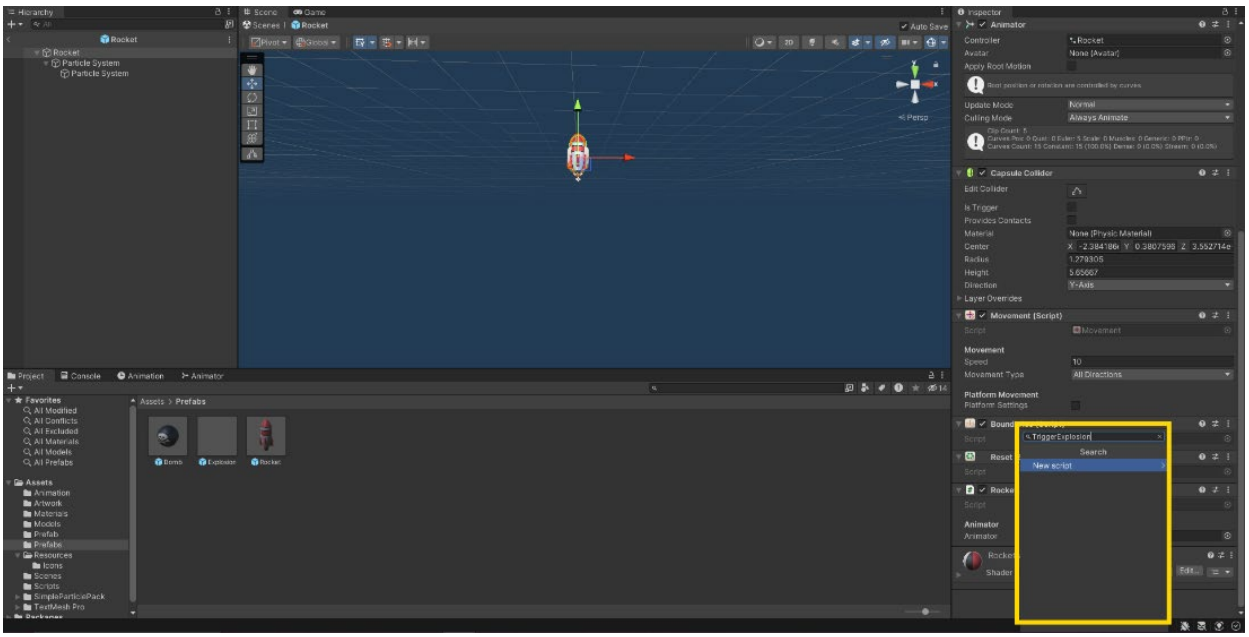


Enable Color Over Lifetime and use the fade to 0 Alpha preset that we saved earlier.

Enable Rotation Over Lifetime and set a Random range between 0 and 45.



42 Go back to the **Hierarchy** and make sure that the original explosion has been deleted from the scene. We still need to trigger the explosion when the **Rocket** hits a **Bomb**. In the **Prefabs** folder, select the **Rocket** and at the bottom of the **Inspector**, click **Add Component** and select **New Script**. Name the script ***TriggerExplosion***.



If you've lost track of any of the particle systems that you been working on, double-check their position in the transform component.

43 Open the new script in the script editor. The script needs a variable to identify the explosion prefab, so add a **public** variable of the type **GameObject** and name it ***explosion***.

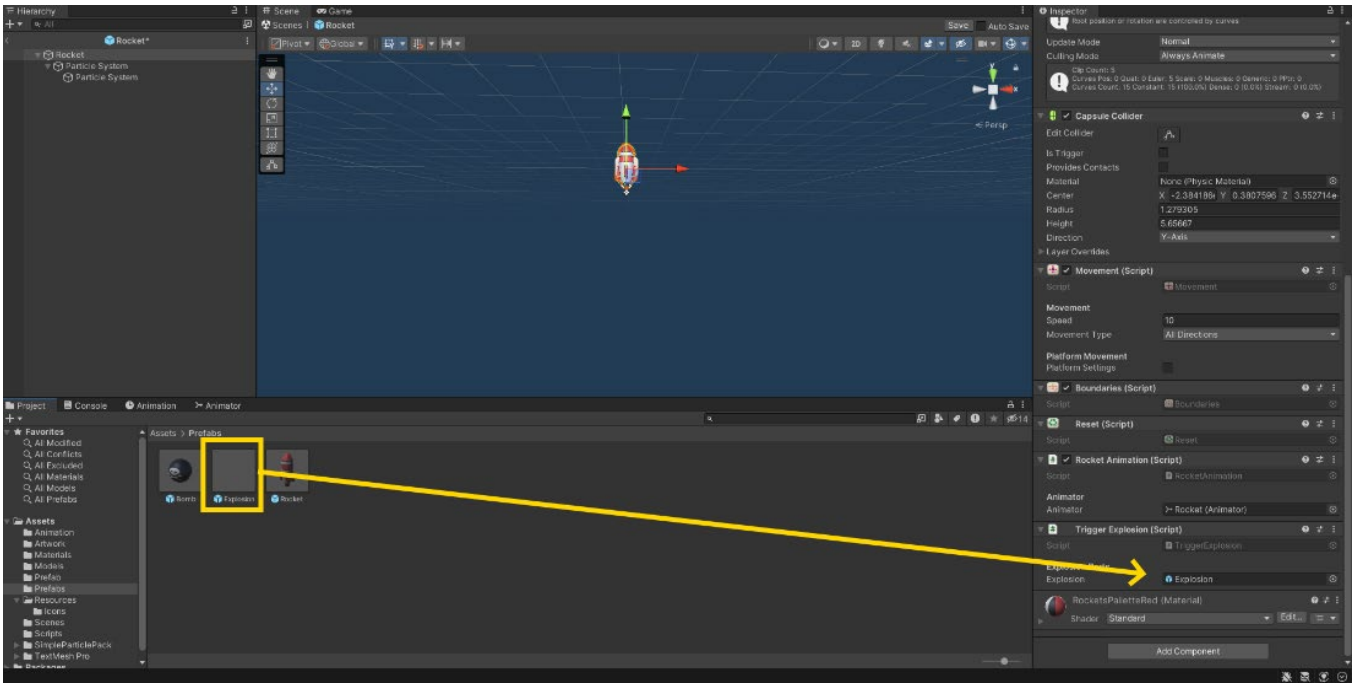
You want the script to activate when anything touches the object this script is attached to (the Rocket) so let's add a function that is a private void OnCollisionEnter (the parameter is type of Collision and is named collision).

Inside the function Instantiate the explosion prefab at the position and rotation of the GameObject this script is attached to (the Rocket).

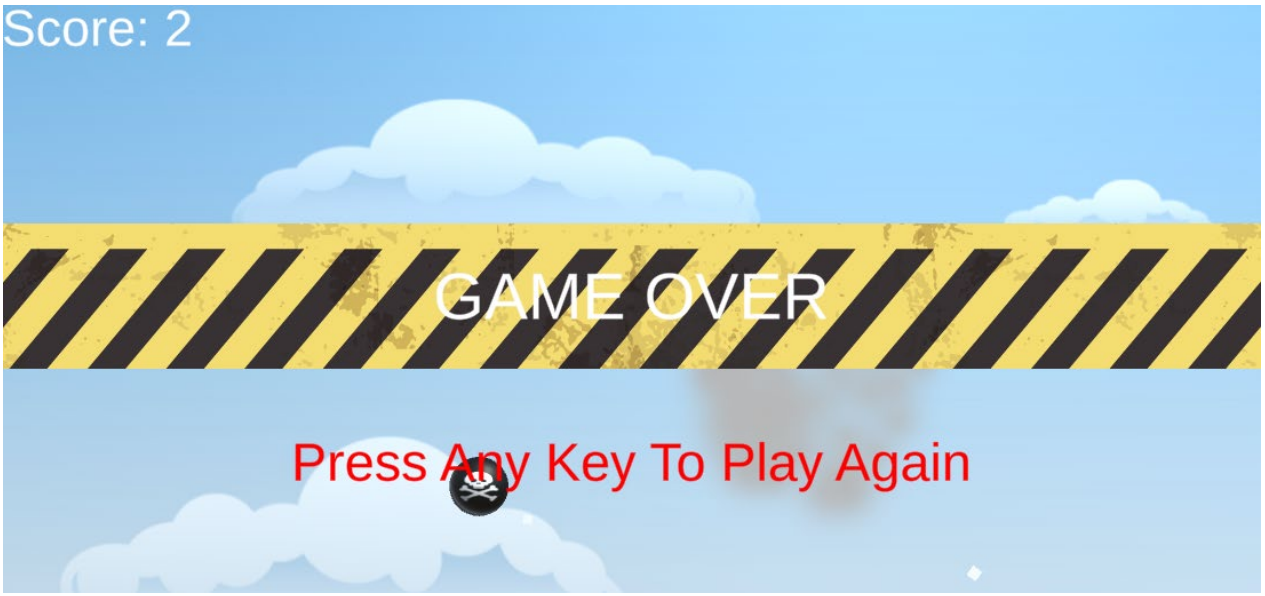
Save the script.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script | 0 references
5   public class TriggerExplosion : MonoBehaviour
6   {
7
8       [Header("Explosion Parts")]
9       public GameObject explosion;
10
        Unity Message | 0 references
11      private void OnCollisionEnter(Collision collision)
12      {
13          Instantiate(explosion,transform.position,transform.rotation);
14      }
15  }
16
```

44 Switch back to Unity and find the **Trigger Explosion** script attached to the **Rocket** prefab. Drag the **Explosion** prefab into the slot for the **Explosion** variable.



45 Now when you play the game, hitting anything with the rocket causes an explosion. The only problem is that the splash screen appears at the same time!



46 To fix this, you need to add a delay before the **GameManager** shows the splash screen. Open the **GameManager** script. To know when the explosion smoke has cleared, add a **private** variable of the type **bool** named ***smokedCleared*** and set it to **true**.

```
Unity Script (1 asset reference) | 0 references
public class GameManager : MonoBehaviour
{
    private Spawner spawner;
    public GameObject title;
    private Vector2 screenBounds;

    public GameObject splash;

    [Header("Player")]
    public GameObject playerPrefab;
    private GameObject player;
    private bool gameStarted = false;

    [Header("Score")]
    public TMP_Text scoreText;
    public int pointsWorth = 1;
    private int score;

    private bool smokeCleared = true;
```

47 Go down to the **OnPlayerKilled** function in the script. This function gets called as soon as the rocket is destroyed and that's where the splash is set to active. Replace **splash.SetActive(true)** with **Invoke("SplashScreen", 2f)**.

Invoke will call the named function after an indicated period of time. In this case, the time is 2 seconds.

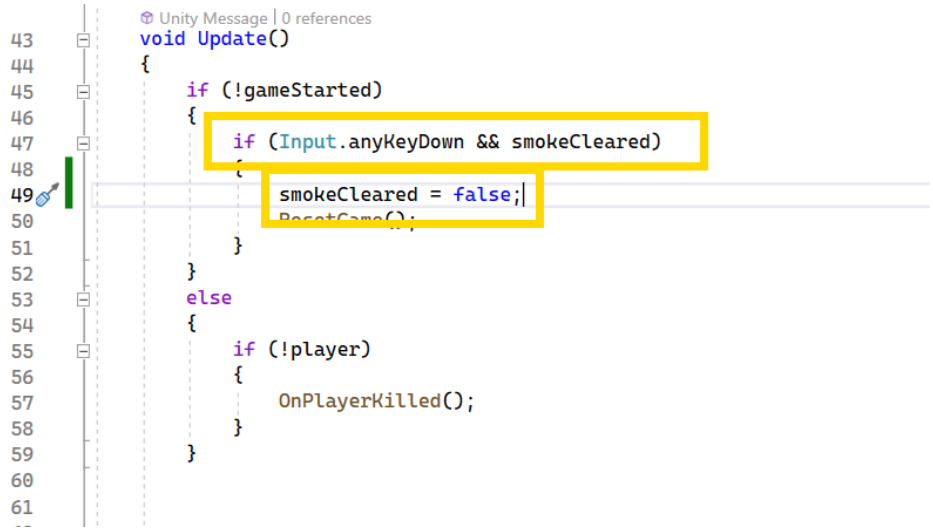
While here, we are going to add a new function called **SplashScreen** that sets the splash screen to appear.

```
1 reference
void OnPlayerKilled()
{
    spawner.active = false;
    gameStarted = false;

    splash.SetActive(true);

    Invoke("SplashScreen", 2);
}

0 references
void SplashScreen()
{
    smokeCleared = true;
    splash.SetActive(true);
}
```

48 Go to the **Update** function in the script. Right now, it is possible to restart the game by pressing any key before the splash screen appears. To fix that, modify the conditional so that nothing happens unless both **Input.anyKeyDown** *and* **smokeCleared** is true (in **C#** 'and' is represented by "**&&**").

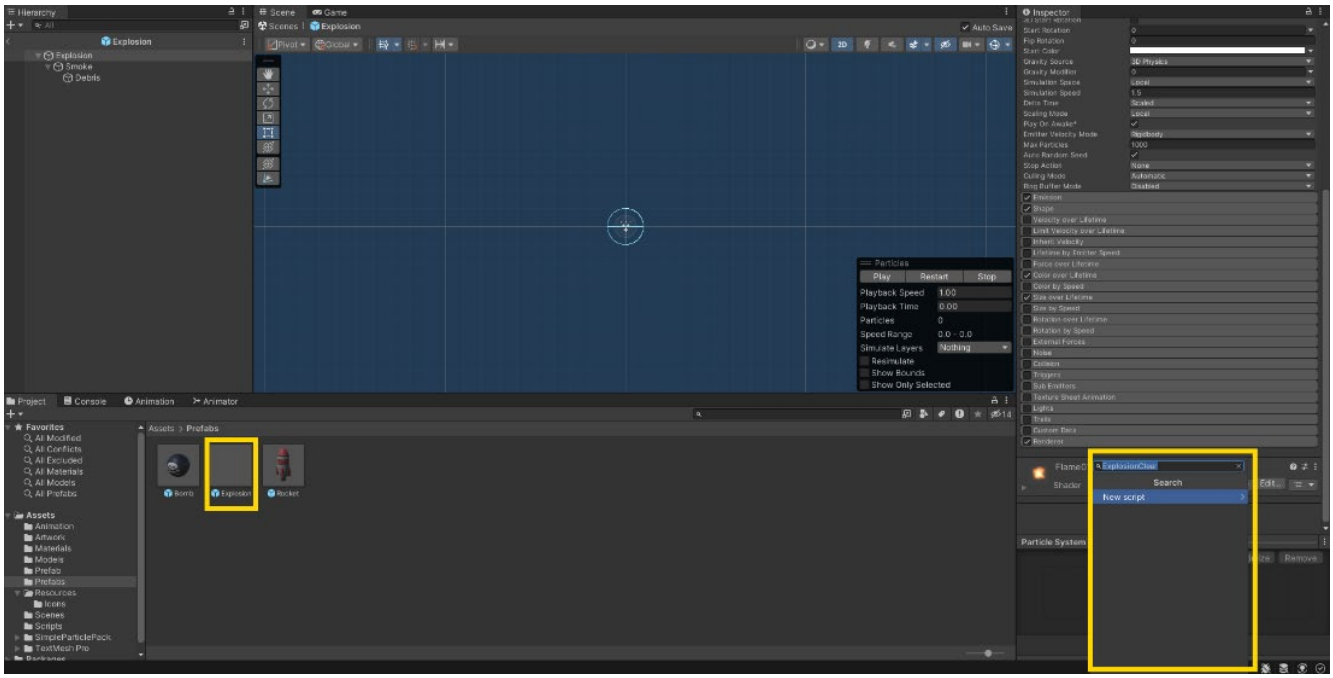
Inside the conditional, let's add a line to set **smokeCleared** to **false** so that a new game can't be started again until the **SplashScreen** function has been invoked.

Don't forget to save your script! Return to Unity.



49 One thing left to do. You are using **Instantiate** to make copies of the **Explosion** prefab, but there's nothing to clean them up. In the Prefabs menu, open the **Explosion** and create a new **C#** script called **ExplosionClear**.

Open it in the script editor.



50 Since the explosion only plays once, it's safe to remove it after the particle systems have stopped playing. In the explosion, the smoke takes the longest to clear, so let's track that.

Add a **private** variable of the type **ParticleSystem** and name it ***particleSmoke***.

The moment the object is instantiated, (**public void Awake**) you'll need to define **particleSmoke**. Since it is a child of the explosion object, you can set **particleSmoke** to **gameObject.GetComponentInChildren()**

Finally in the **Update** function, we check to see if **particleSmoke** is still running (**IsAlive**). If it isn't, **Destroy** this **GameObject**.

Save your script and return to Unity.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script (1 asset reference) | 0 references
5   public class ExplosionClear : MonoBehaviour
6   {
7       private ParticleSystem particleSmoke;
8
        Unity Message | 0 references
9       private void Awake()
10      {
11          particleSmoke = GetComponentInChildren<ParticleSystem>();
12      }
13
        Unity Message | 0 references
14      private void Update()
15      {
16          if (!particleSmoke.IsAlive())
17          {
18              Destroy(gameObject);
19          }
20      }
21  }
22
```

51 Now the game runs and looks pretty sharp! Too bad you can't remember your best score. If only there was something to fix that...

52 Before we move onto the final part, let's change how some things look. Using what you know about UI, change your UI elements to look how you want. You can copy the image below or use what you think looks best. You can also try messing around with the particle systems, try changing the color of the smoke to an alien green or maybe try changing it to a golden yellow.