CODE NINJAS®

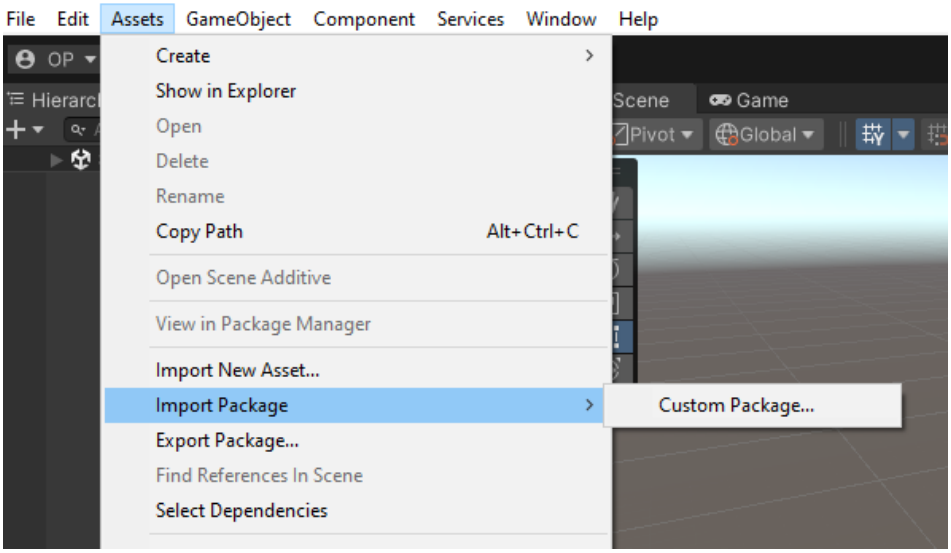
Bronze Belt Ninja Guide

Activity 08: Dropping Bombs Part 2

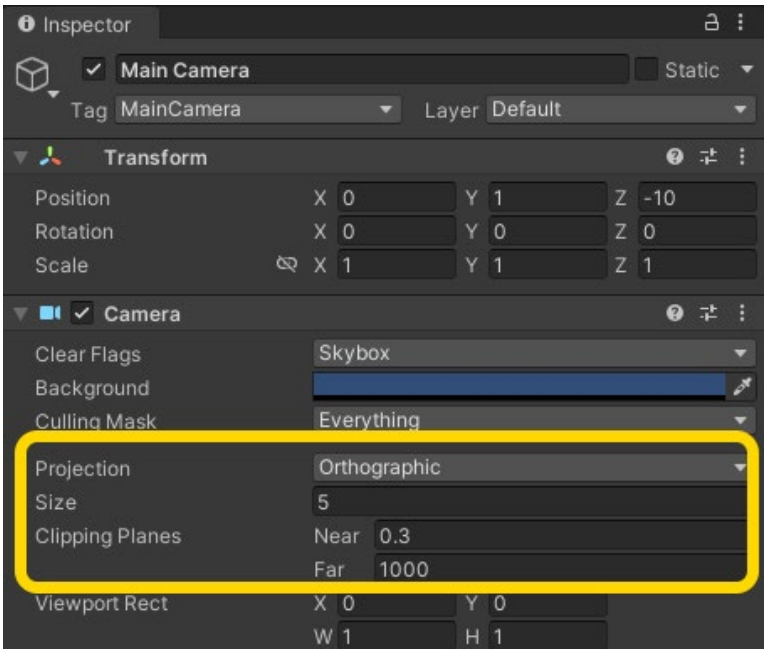
Activity 8: Dropping Bombs Part 2

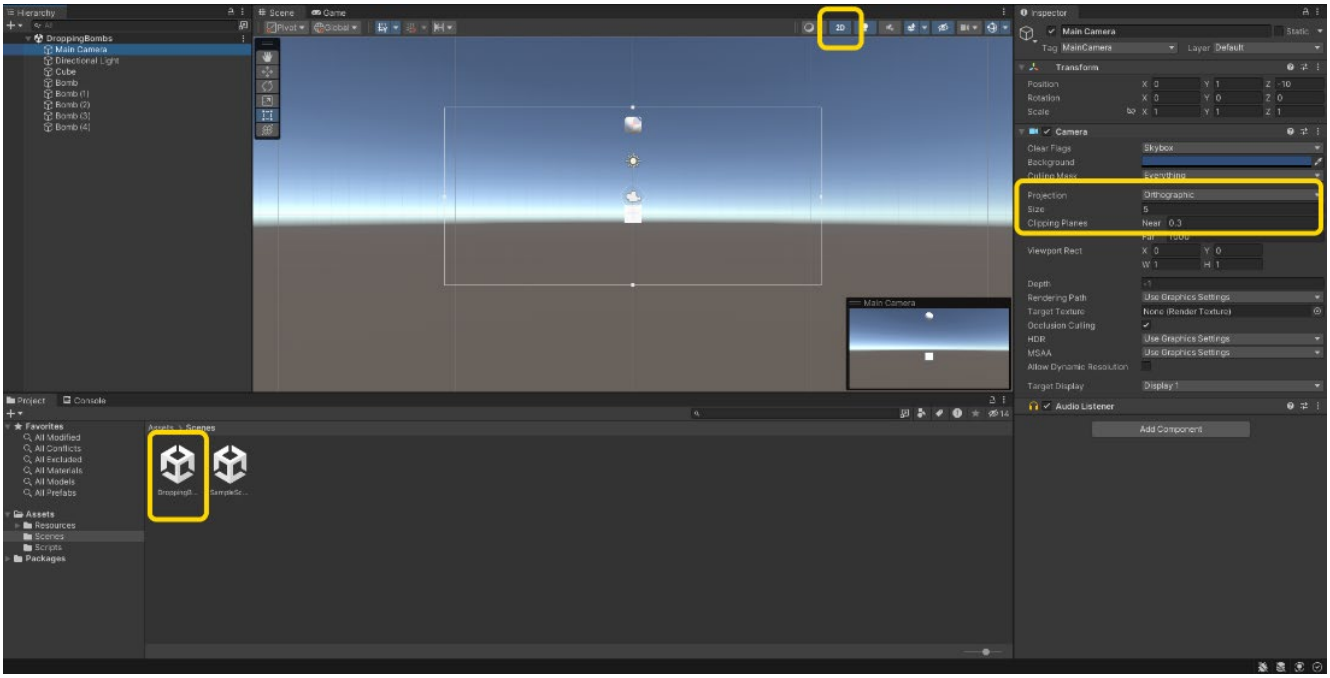
As promised, this section and the next are devoted to taking the rather simple Dropping Bombs game from the beginning of the book and giving it a good update to make it look like a professional game. You may either start with your files from that activity or open up Unity and create a new 3D project, giving it a name like **JS-DroppingBombs2**.

1 If you are starting with a new project, go to the **Assets** tab, select **Import Package**, and click on **Custom Package**. Go to the folder with all your Bronze Belt files and select **Activity 08 – MyDroppingBombsPart1.unitypackage**.

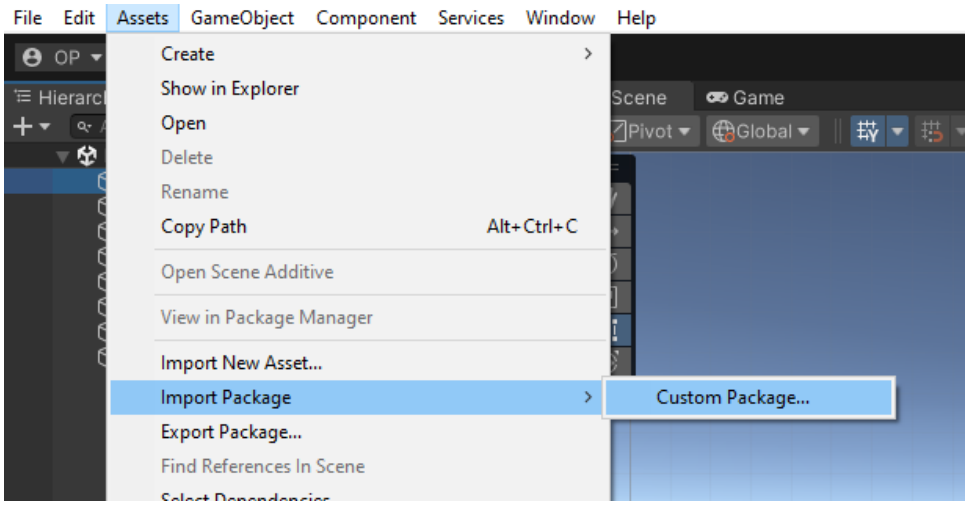


2 If the Hierarchy seems empty, go over to the **Scenes** folder, and select **DroppingBombs** (or whatever your Scene was named). Just as with the original Dropping Bombs game, we will be using the **2D** camera to edit our scene. Select the **Main Camera** GameObject in the **Hierarchy** panel and change the **Projection** from **Perspective** to **Orthographic** as shown below.

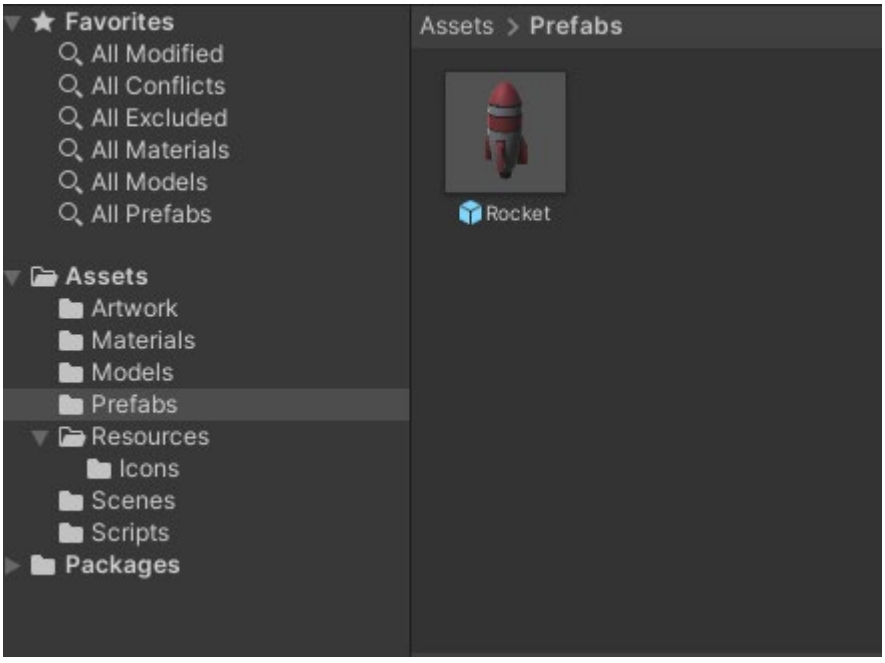




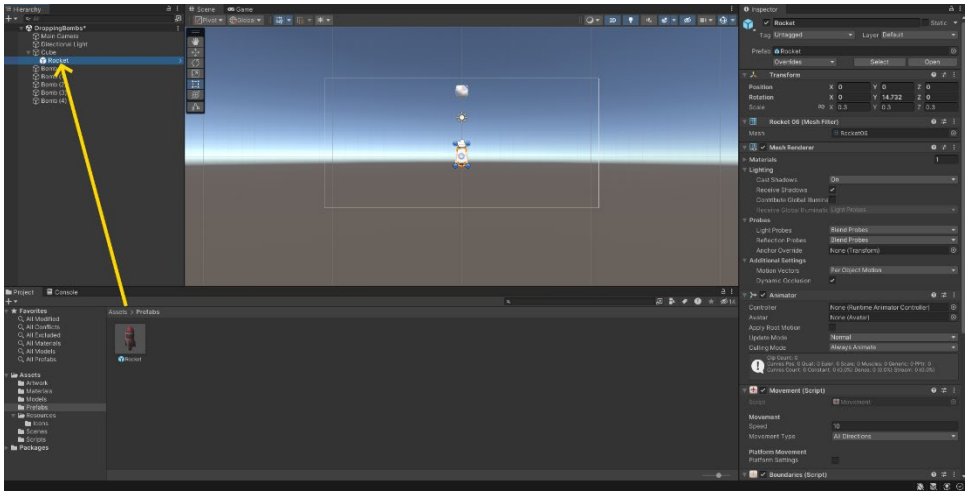
3 The first thing you'll want to do is replace the cube and sphere in the game. Go back to the **Assets** tab and select **Import Package**, then select **Custom Package**. This time, select and load **Activity 08 - RedRocket.unitypackage**.



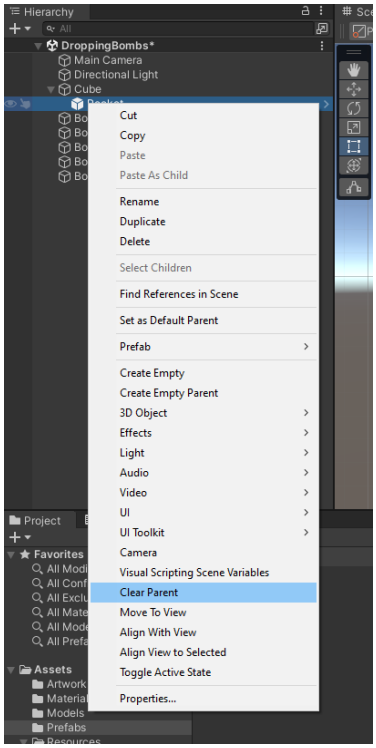
4 To verify that the new assets are loaded, go to the **Project** panel and select the **Prefabs** folder to see the **Rocket** assets.



5 To replace the cube with the **Rocket**, hold down the **Alt** key while you're dragging the **Rocket** prefab over to the cube object in the Hierarchy. The rocket will copy all the scripts on the cube. However, we still see the cube, so we will need to remove it.

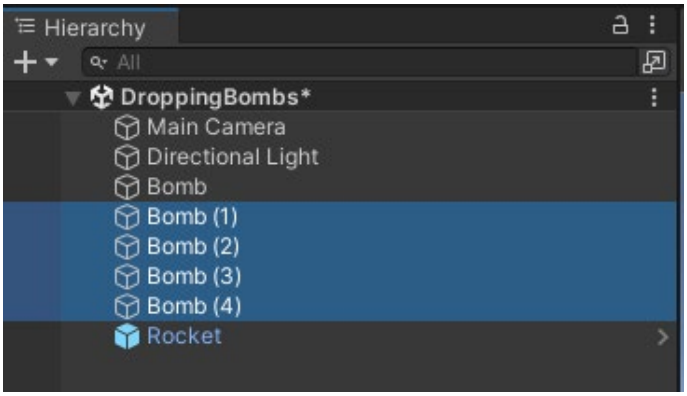


6 To remove the old cube, right click on the **Rocket** and select **Clear Parent**. This will remove our rocket from the parent but will still leave the cube. After that, select the **cube** and delete it.



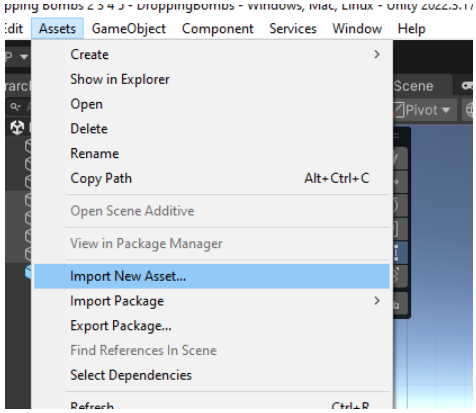
7 Just like with the rocket, we'll need to do the same for the bombs.

Remove the extra bombs in the **Hierarchy** by highlighting them and deleting them. You should be left with one bomb.



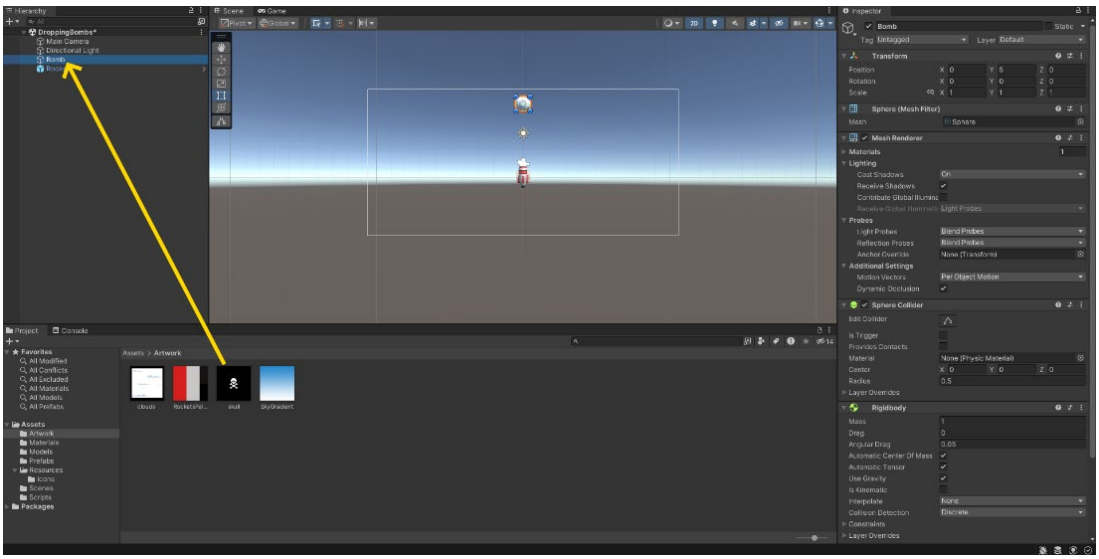
8 Before we can work on our bomb, we'll need to change how they look. We'll start by importing some assets. Instead of importing a Package, we want to import **Assets**.

Import the **Activity 08 - Skull.png**, **Activity 08 - SkyGradient.png**, and **Activity 08 - clouds.png** files from the Bronze Belt assets folder into your **Artwork** folder.

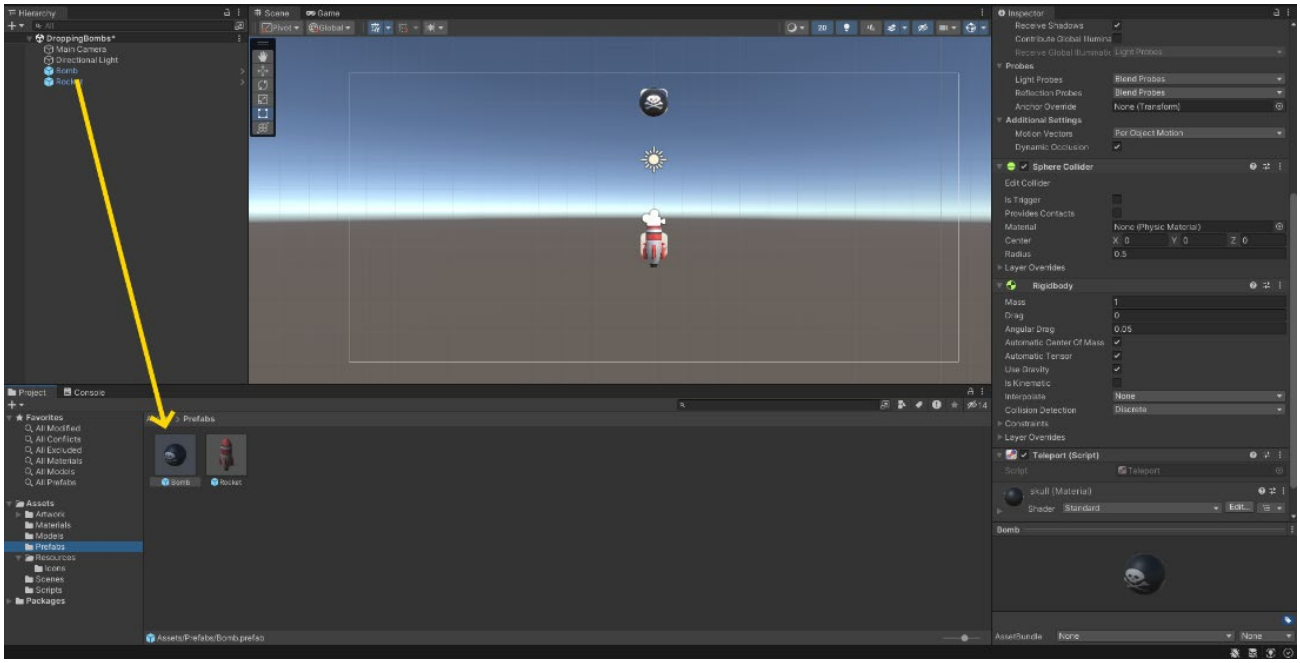


9 With the new imported assets, drag the **Skull** texture from the **Artwork** folder onto the **Bomb** GameObject in the **Hierarchy** to give it a new look. This will automatically create a new **Material** and give it to your bomb.

To make sure you can see the skull, rotate the **GameObject** by 90 on the Y axis.

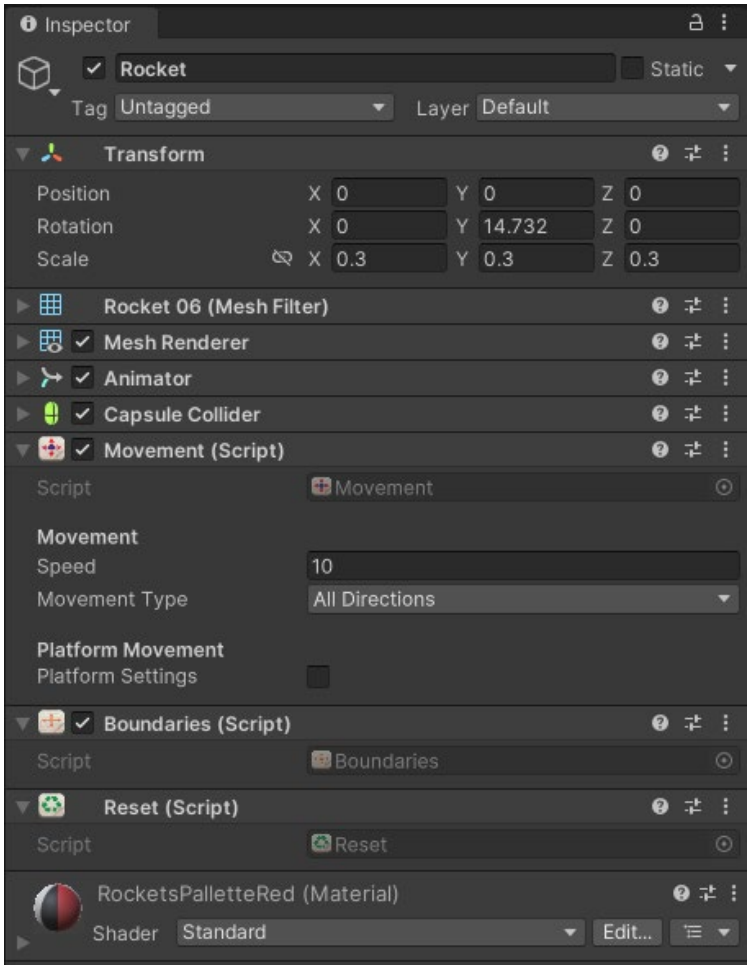


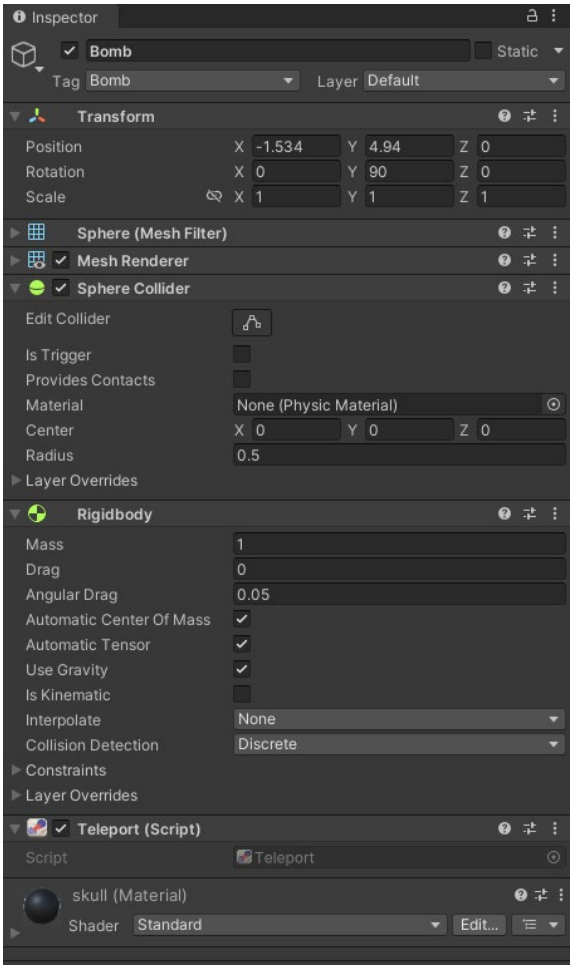
10 Drag the **Bomb Gameobject** from the **Hierarchy** to the **Prefabs** folder, making it into a **Prefab**. Now any changes you make to the **Bomb** prefab will automatically get added to any bombs in the Hierarchy



Don't panic if some of the scripts attached to your GameObjects have vanished! We'll put them back in the next step.

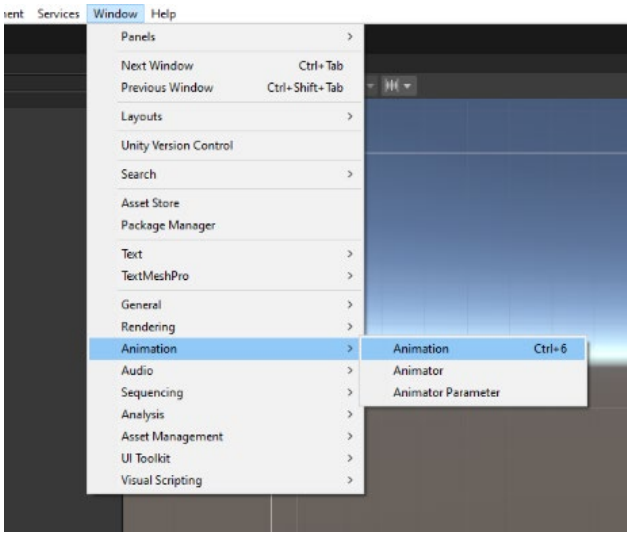
11 If any scripts get lost while replacing the objects, just drag them back into the **GameObject** Prefabs and use the settings below (The Teleport script goes on the Bomb while the other three go on the Rocket). Once complete, try playing the game!





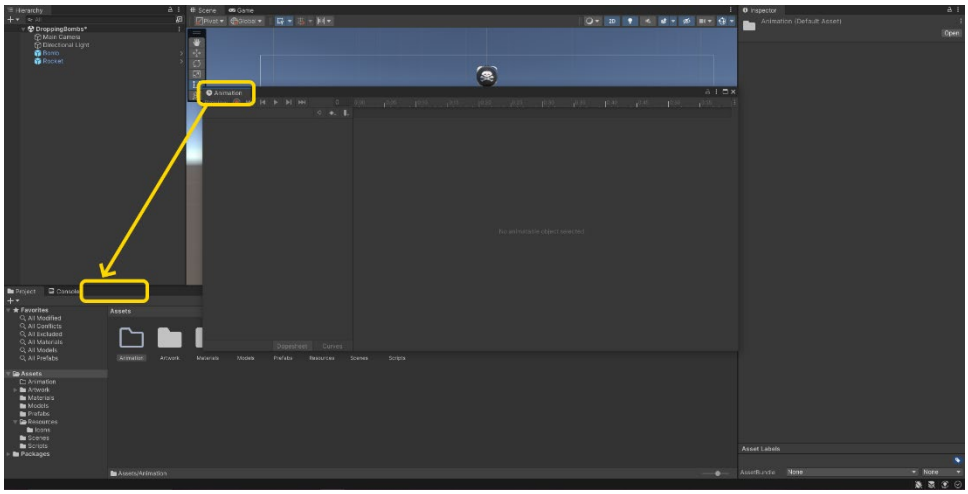
12 To make this game more interesting, we are going to add some animations to make the rocket more realistic. First, create a folder just for animations. In the **Projects** panel, select the **Assets** folder, right-click to create a new folder and call it "***Animation***".

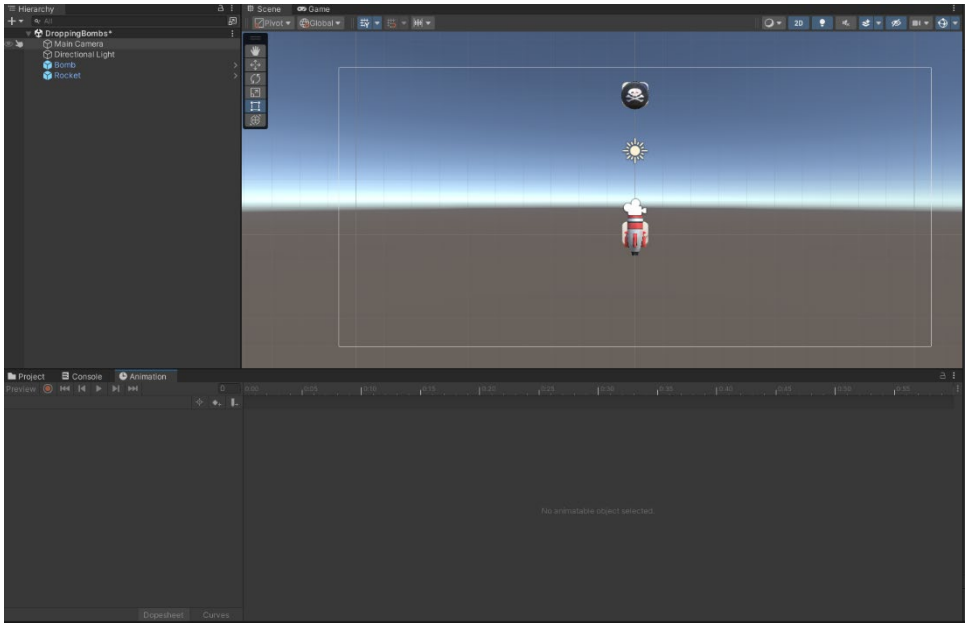
13 To edit animations, we need to open the Animation Window.
There are two windows, but for now, we want the one called Animation. Click the **Windows** tab and select **Animation**, then select **Animation** again.



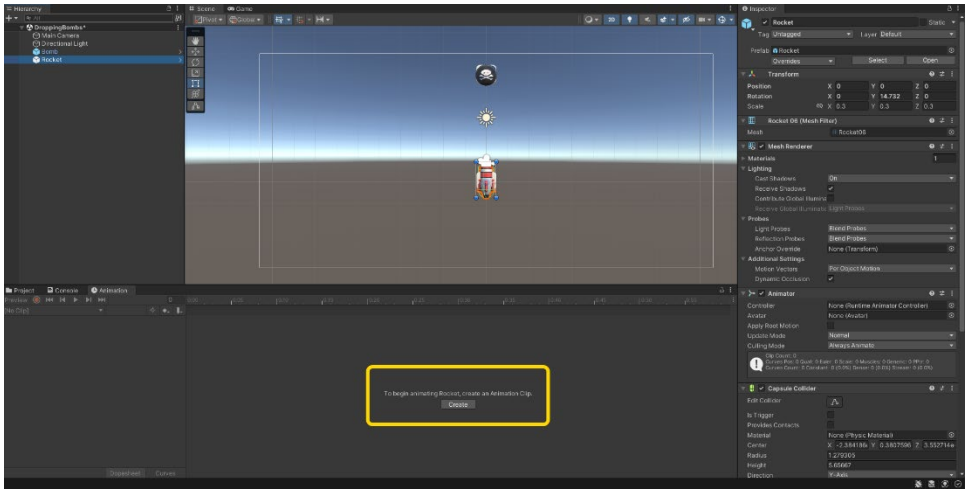
14 This has opened the **Animation** panel. This panel should appear in the middle of everything! If you need some more space, you can move it around or add it to another panel.

By dragging the **Animation** tab in the **Animation** panel and dragging it around, you can see how it interacts and fits into various places. Find the one that is most comfortable for you. Click the Windows tab and select Animation, then select Animation again.panel since we won't be needing that.

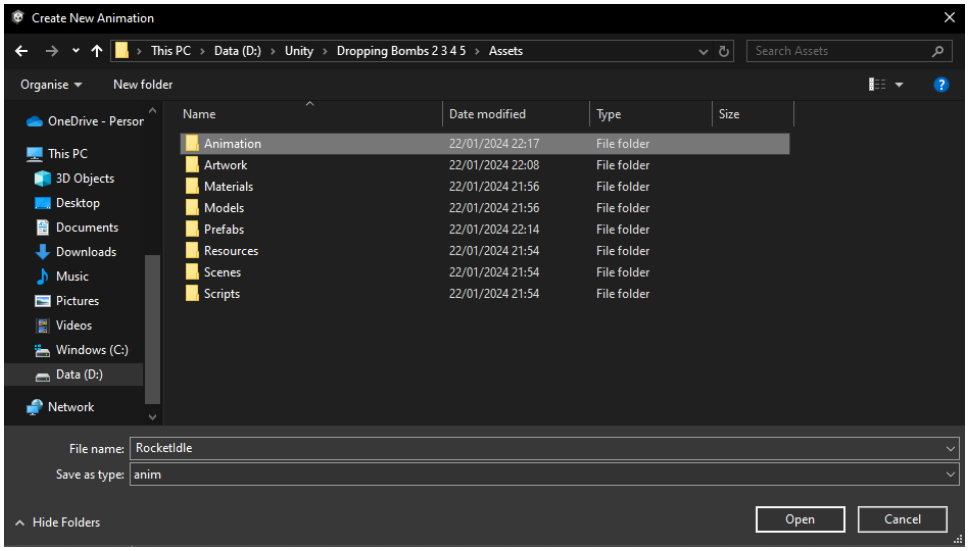




15 Make sure you select the **Rocket** in the **Hierarchy** and click the **Animation** tab. Since you don't have any animations attached to the Rocket, you will be asked to create one. Click the **Create** button.

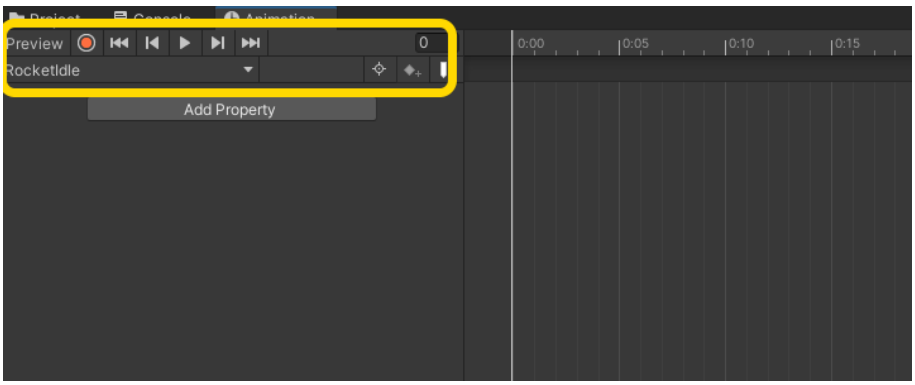


16 This opens the **Create New Animation** window. Open the **Animation** folder that you just made – we'll be putting all the animations in there. Give it a name like **RocketIdle** and click save.

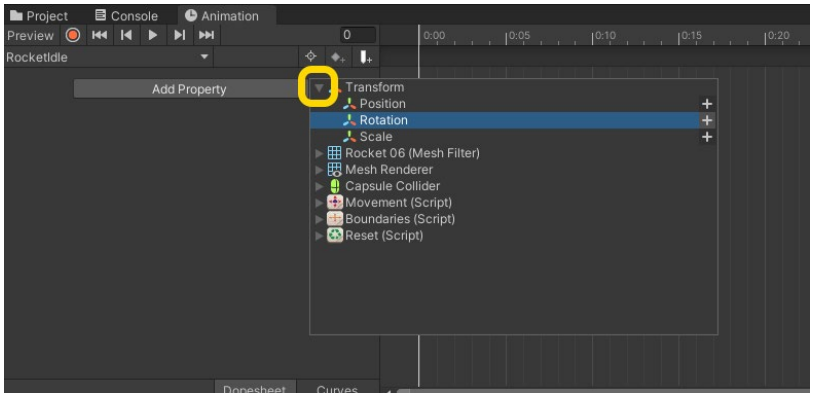


17 Now the **Animation** panel is ready for you to begin creating an animation for your rocket. Look at the control panel in the upper left. These let you record and play your animation to make sure it looks right. It also lets you know what animation you're currently working on and gives you the means to add additional animation to the object.

Always make sure you are on the correct animation when editing!



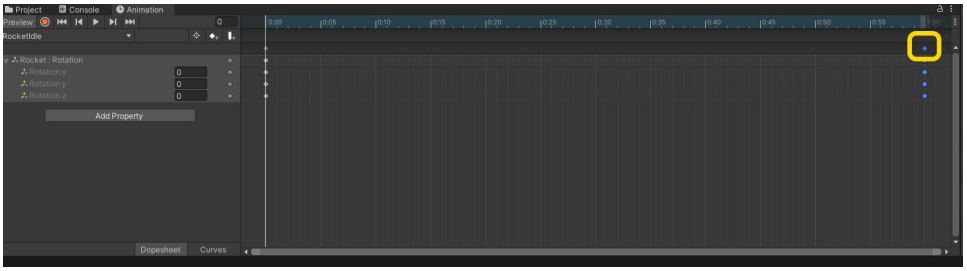
18 In the **Animation** panel, click **Add Property**. This opens a window of all the possible properties in the **GameObject** that you can modify to create an animation. Most Often, animations are applied to an object's **Transform** component (Position, Rotation and/or scale). In this case, we want to edit the **Rotation**, so select that and click the plus symbol (**+**) on the right to add that property to the panel.



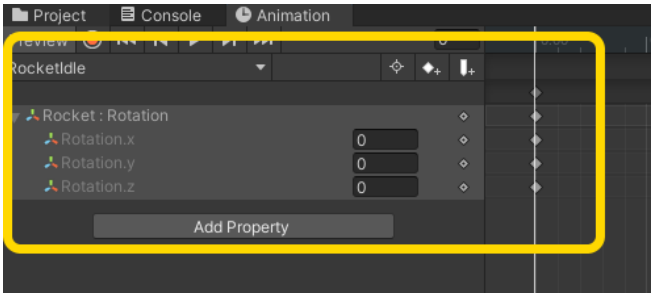
19 This will add two diamonds, these are **Keyframes** to our animation Timeline. One at **0:00** and one at **1:00**. This refers to the amount of time in seconds. So, our Keyframes are happening over 1 second. The Timeline refers to the area in which we will be placing the **Keyframes** and creating the animation.

20 Currently, we see one major keyframe for the whole component, but by clicking the little arrow on the left, next to the rotation, we can expand it and see 3 new keyframes one for the X, Y and Z rotation.

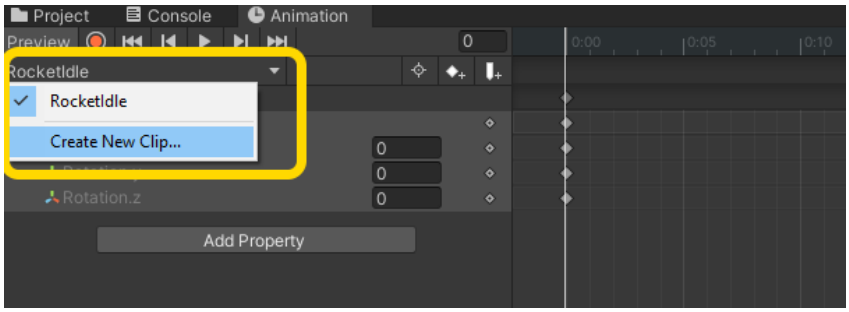
For our first animation, we don't need the keyframes at the 1:00 mark. Select the **top parent keyframe** and press the **delete** key. You will be left with only the keyframes at the 0:00 second mark, meaning this animation only happens for 1 frame.



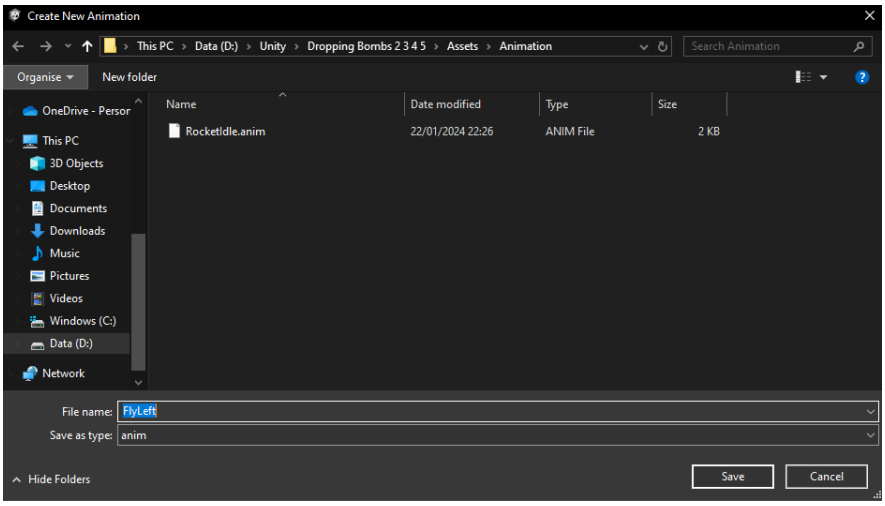
21 That's it for our first animation! The final thing is to make sure the numbers in your keyframes are set to 0, 0, 0. Since this is our idle rocket, it will be set to a rotation of 0, 0, 0.



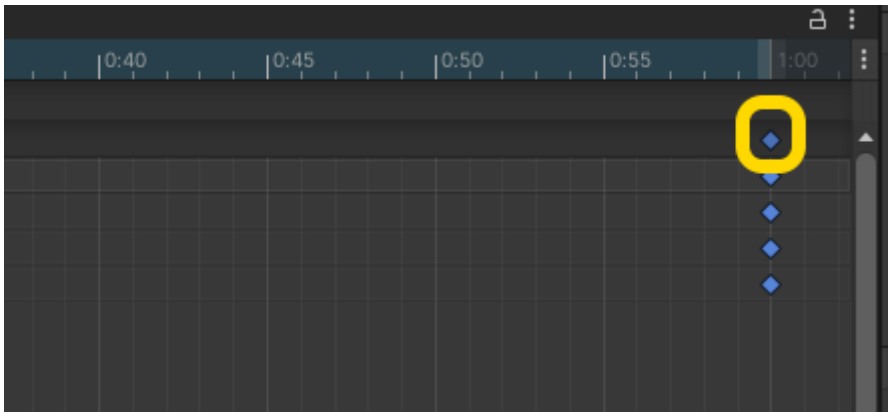
22 Now to create animations for the rocket moving left and right. In the upper left corner of the animation panel, click on the name of the current selected animation. This will open a tiny menu that shows all animations of the current selected GameObject. Select **Create a New Clip**.



23 Make sure that you're in the **Animation** folder and give your new animations a descriptive name, like ***FlyLeft***.

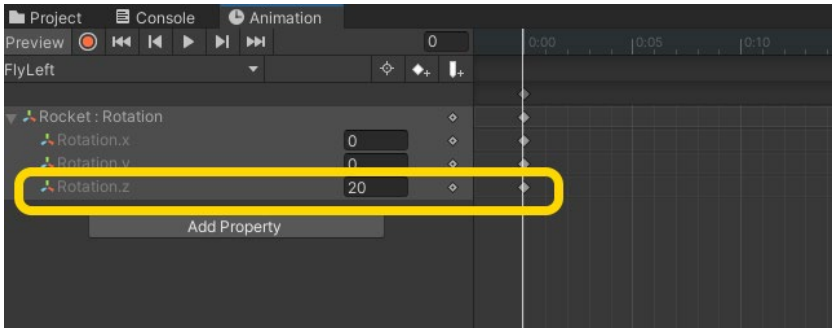


24 Repeat Steps 18 and 19 above to add in the **Rocket's Rotation** component and **delete** the extra Keyframes at the end, as you did with the first animation.



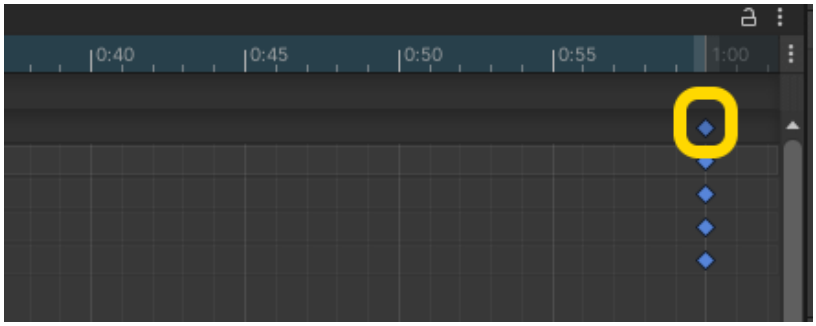
25 Once you've set up your single keyframe for the **FlyLeft** animation, we can start making the rocket fly to the left. To do so, select the **rotation Z** keyframe and type in the number **20**, giving it a rotation of that value.

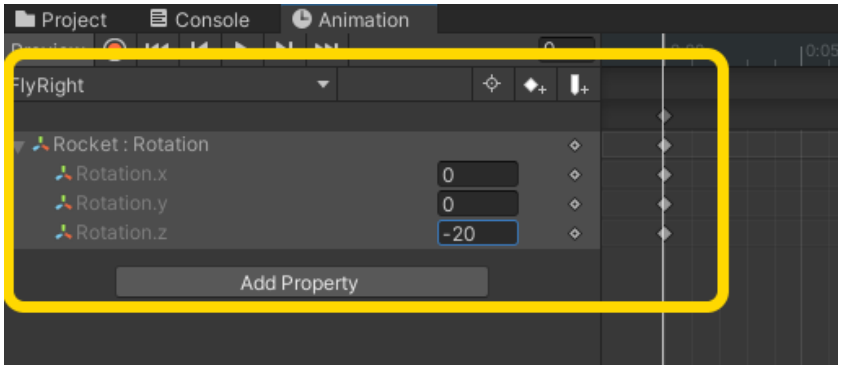
If you don't see the rocket change right away, make sure the Preview button in the top left is selected.



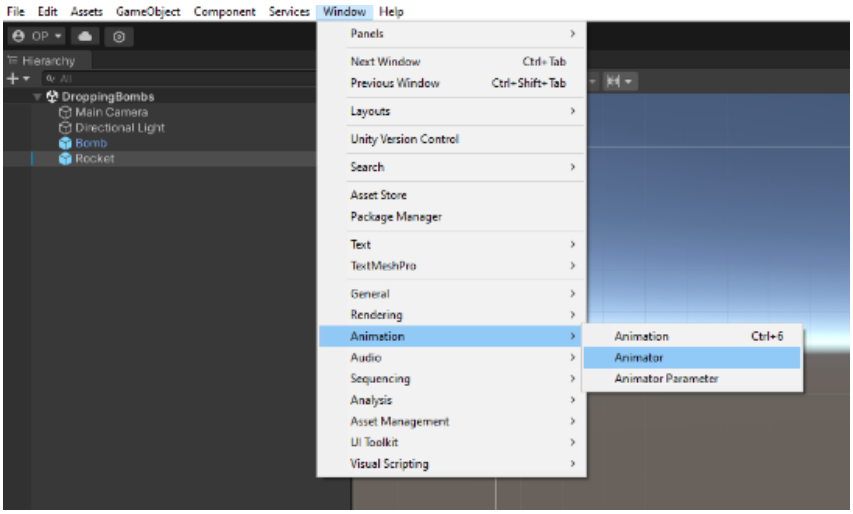
26 When you are done, we can repeat the above steps but for a third animation called ***FlyRight***.

You'll need to **Add a New Clip**, **add the rotation value**, and **remove the extra keyframes** before changing the Z value to **-20**.



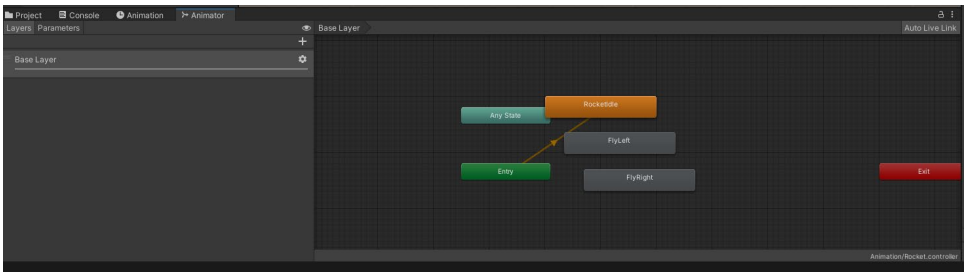


27 Now we have Three Animations: one for leaning left, one for leaning right, and one for the middle. The next step is to let **Unity** know when each animation needs to be used. For that, we need the **Animator** panel (not to be confused with the ***Animation*** panel we just used). To do this, click **Window**, select **Animation**, then select **Animator** to load it.



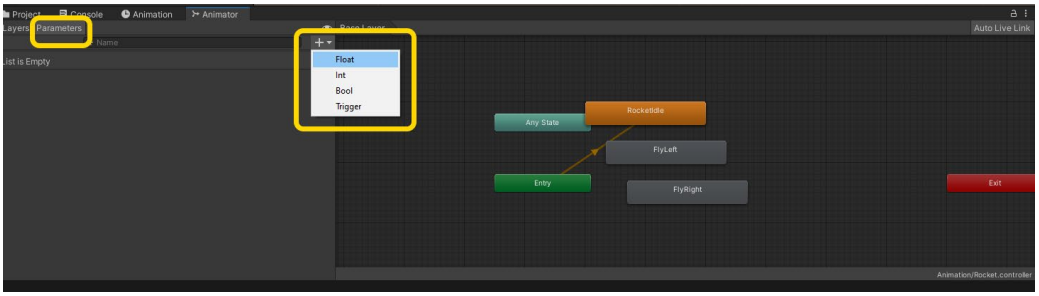
28 Just like with the other panels, you can move the **Animator** panel around until you find a place that feels comfortable. Just like before, placing it down by the project and the animation tab works well.

This tab may look confusing, but it's actually very simple. The rectangles are the states of animation that we just made. The green one is the **Entry** state for when the game starts, and the orange one is the default state. Currently, our default state is the one we made first, RocketIdle. This means our first animation will be RocketIdle, when the game loads.

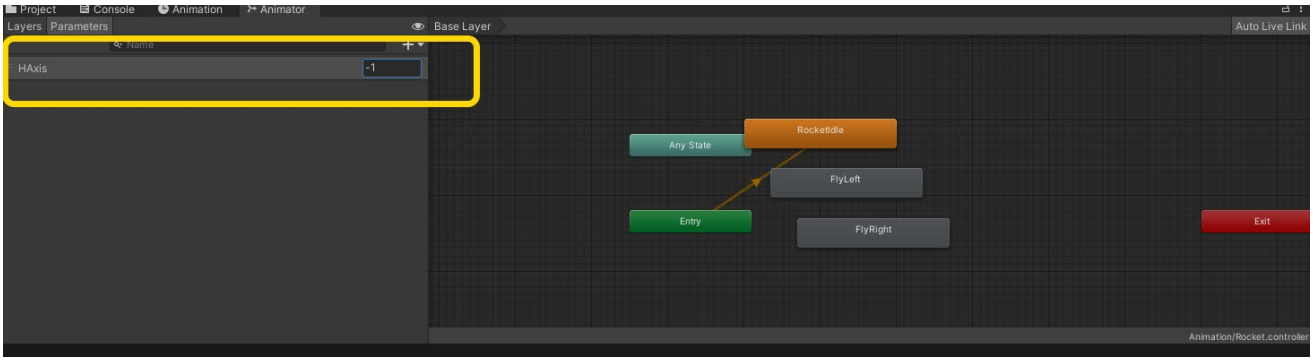


There can only be one default animation state for each object, but any animation in an object can be designated as the default animation state.

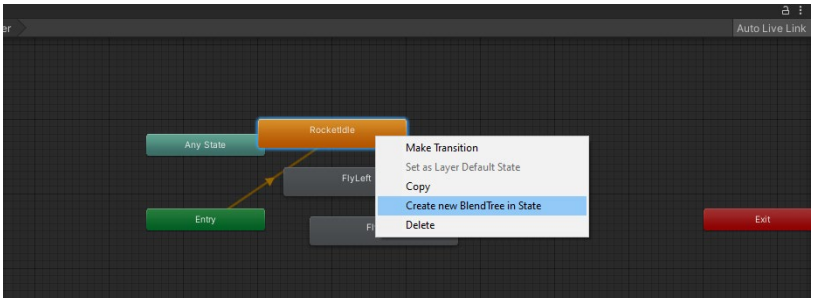
29 We can connect the **Animator** to a script, so the script can tell the animator which animation it should play. For our rocket prefab, all we need is a single parameter. Click the **Parameters** tab and add a new parameter by clicking the plus (**+**) symbol then **Float**. This Parameter will be a floating-point number (meaning it can be a decimal number).



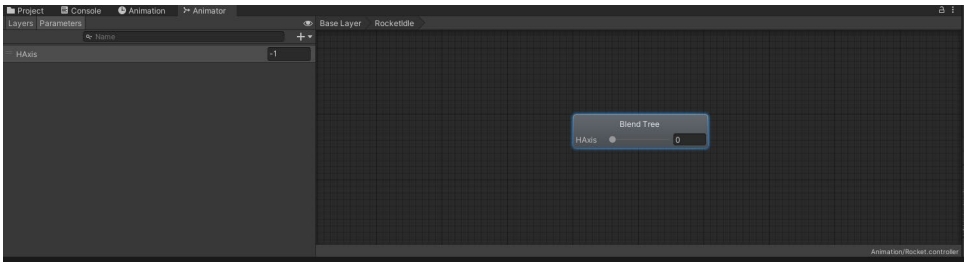
30 After the float appears, rename it **HAxis** (meaning Horizontal Axis). This parameter will tell the **Animator** whether the player is moving to the left or right. Change the value of HAxis to **-1**.



31 You can create links between animation states (like going from Entry to RocketIdle), but there is an easier way. Right-click on the orange default animation (RocketIdle) and select **Create New BlendTree In State**.

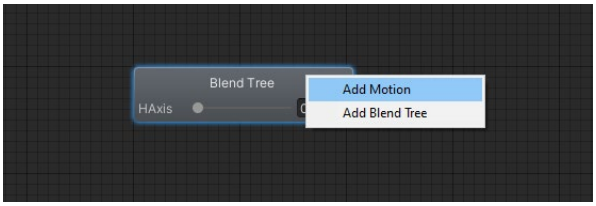


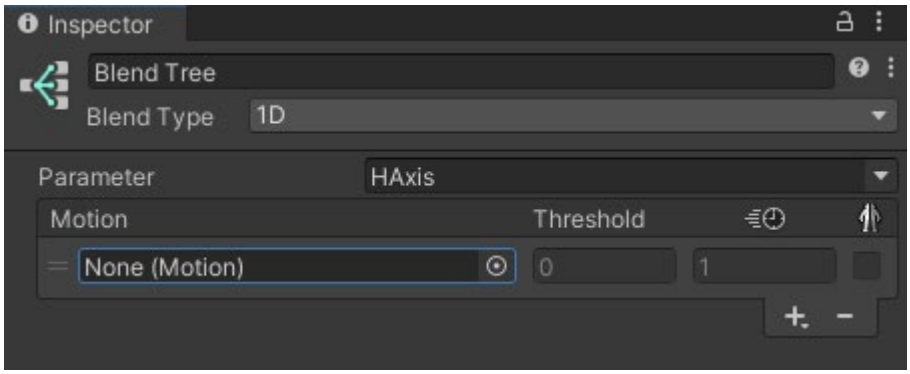
32 It will look as if nothing has happened, but it has! Double-click the default animation (Rocket Idle) to open the **Blend Tree** that you just created.

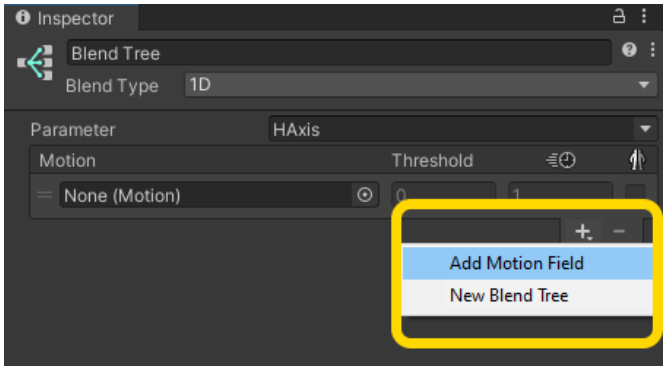


33 Once in the **Blend Tree**, right click the center node and click **Add Motion**.

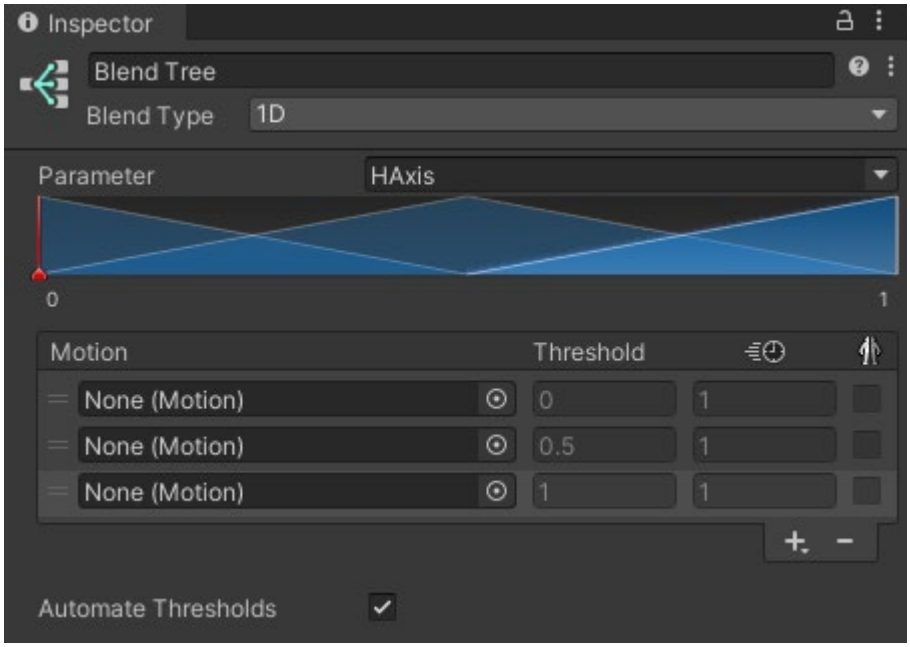
This will create an option in the top right that will allow us to select our animations, but before that, let's create 2 more.







34 Once we have our 3 Motions, we can click the little dots to select what animation will go in them. The top one should be **FlyLeft**, the middle should be **RocketIdle**, and the bottom should be **FlyRight**.



Using a Blend Tree

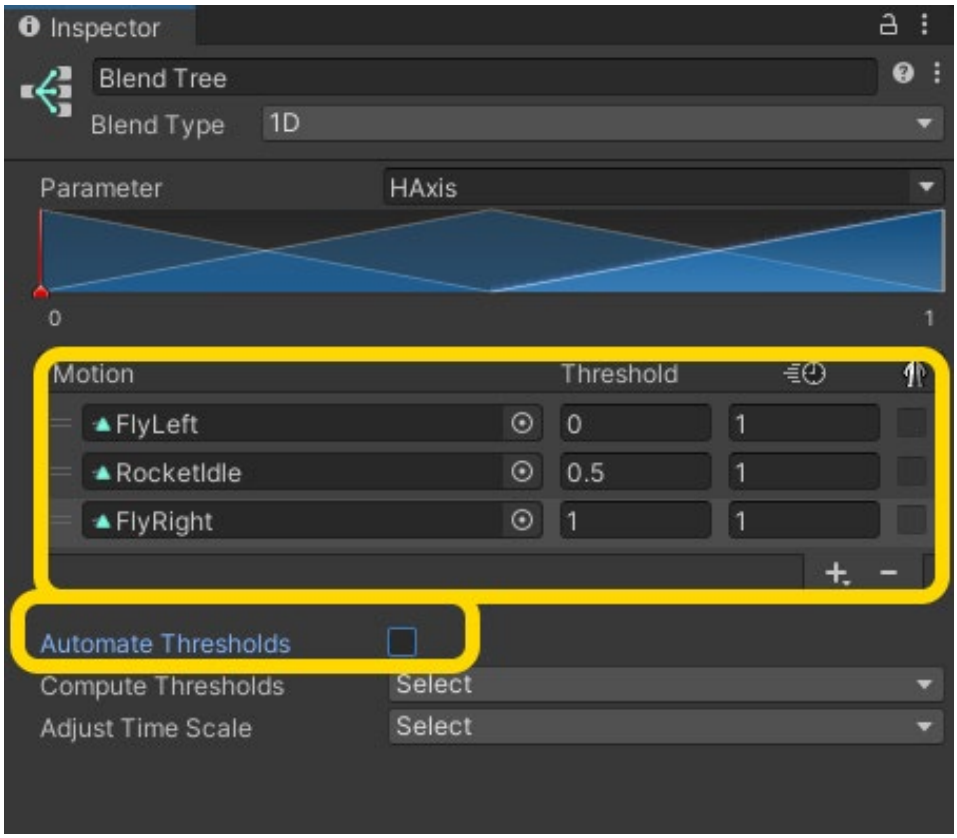
Most animation in Unity can be handled with a simple transition from one animation to the next. For instance, the avatar in Scavenger Hunt has separate falling and landing animations. We don't know how long the avatar will be falling, but when it lands, Unity can smoothly transition from the falling to the landing animation.

When there are more than two animations, handling the transitions can become a bit more complicated. That's where Blend Trees come in handy. With the Rocket, there were three animations. The Blend Tree determines which animation to use based on the data it's receiving – in this case, the data is the player input and the ship animation plays based on Left or Right input.

While our animations may be single frame animations, we can use a Blend Tree to easily transition between them. It can also handle transitions from multiple inputs, such as from moving to the right to moving straight up.

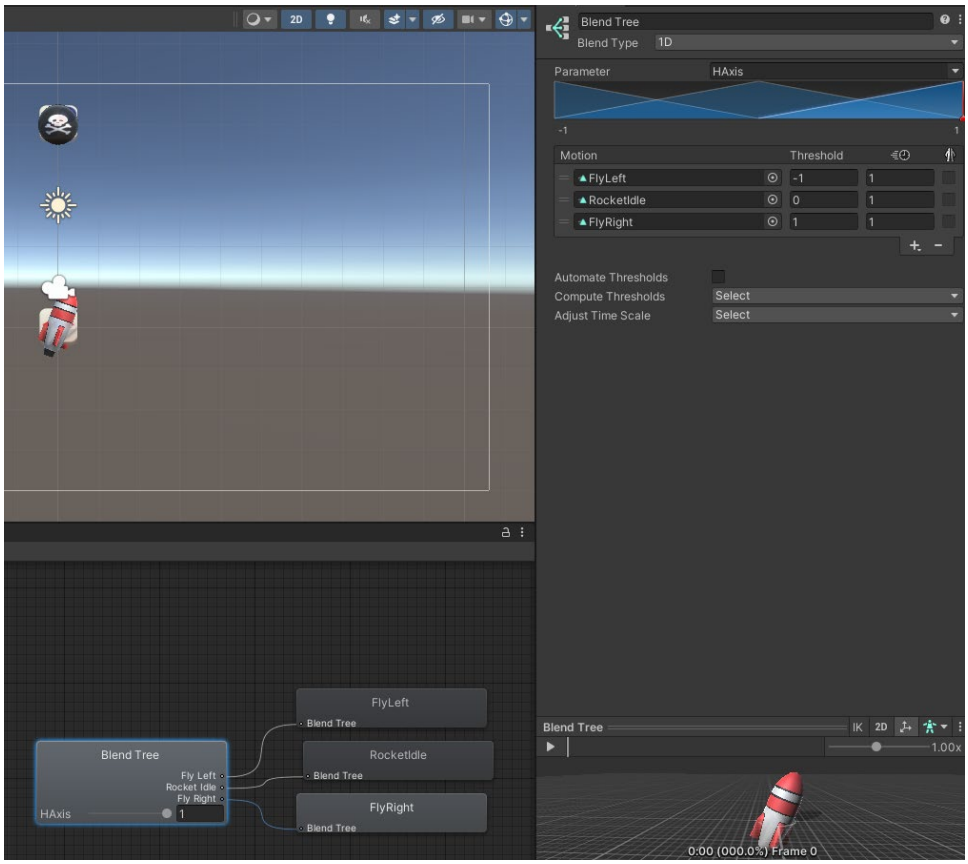
35

Once you've added the three animations, make sure to uncheck **Automate Thresholds** so that we can change the values. The **Thresholds** tell Unity what animation to play based on what number **HAxis** is equal to. FlyLeft should have a **Threshold** of -1, RocketIdle 0.5, and FlyRight 1.

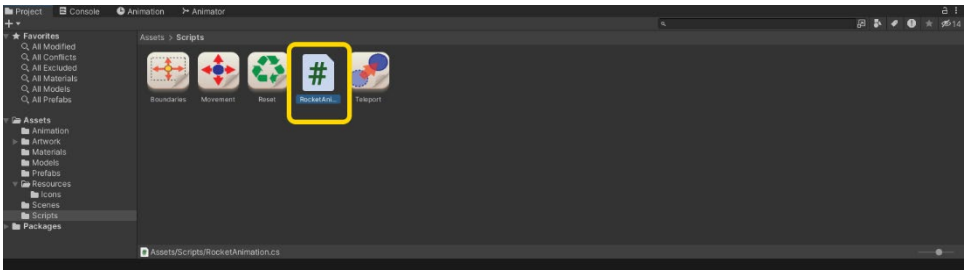


36

Once you've done that. You can preview the animation. Drag the slider in the Blend Tree and see the animation move! Just like that! The blend tree smoothly transitions between animations.



37 With the animations done, we can create a script to allow the **Rocket** and **Animator** to communicate with each other. In the **Project** panel, select the **Scripts** folder and create a new **C#** script. Call it ***RocketAnimation***. Double-click on the new script to open the **C#** Editor.



38 Each new **C#** script has some basic code already. We need to add into it to give instructions on what we want the script to do. For these next steps, try to follow along on your own, but if you get stuck, there will be an image to check to see if you got it correct.

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script | 0 references
5   public class RocketAnimation : MonoBehaviour
6   {
7       // Start is called before the first frame update
        Unity Message | 0 references
8       void Start()
9       {
10
11      }
12
13      // Update is called once per frame
        Unity Message | 0 references
14      void Update()
15      {
16
17      }
18  }
```

39 We want our **variables** to be put before the **void Start** function. In this case, we are creating a **reference** to the animator component. This variable will be **public** so that you can see it in the **inspector**. For this variable, we have given it the type of **Animator**, and the name animator.

40 When the script starts, assign a **component** to the animator variable. In **C#** we use **GetComponent<>()** with **Animator** as the component to retrieve.

41 We need to change the **Update** function to **FixedUpdate**. While Update is called once per frame, FixedUpdate can be called several times per frame, making it ideal for situations where force or other physics-related functions are applied. In a game like this, it's not necessary, but it can help smooth things out.

Next, we declare a **float** variable named **horizontal** and set it to be equal to **Input.GetAxis("Horizontal")** - in other words, when the user moves left or right, that becomes our floating variable, horizontal.

Finally, tell the **animator** (which we set up above) that the floating variable **HAxis** should be equal to **horizontal**.

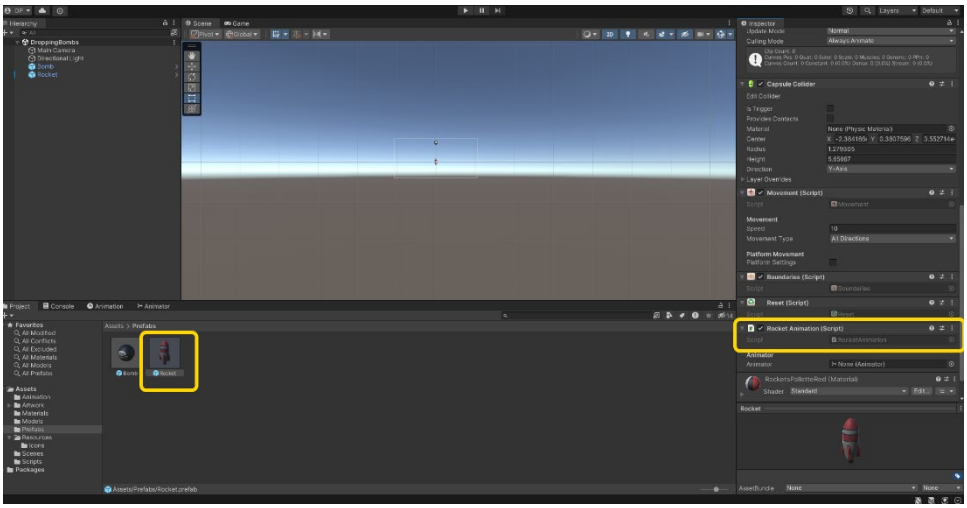
Then save the script by hitting **Ctrl+S** and return to Unity.

RocketAnimation.cs*

Assembly-CSharp | RocketAnimation

```
1   using System.Collections;
2   using System.Collections.Generic;
3   using UnityEngine;
4
    Unity Script | 0 references
5   public class RocketAnimation : MonoBehaviour
6   {
7
8       [Header("Animator")]
9       public Animator animator;
10
11      // Start is called before the first frame update
        Unity Message | 0 references
12      void Start()
13      {
14          animator = GetComponent<Animator>();
15      }
16
17      // FixedUpdate is called several times per frame
        Unity Message | 0 references
18      void FixedUpdate()
19      {
20          //Get Axis Horizontal gets the Left and Right movement
21          //Which is a number from -1 to 1
22          float horizontal = Input.GetAxis("Horizontal");
23
24          animator.SetFloat("HAxis", horizontal);
25      }
26  }
27
```

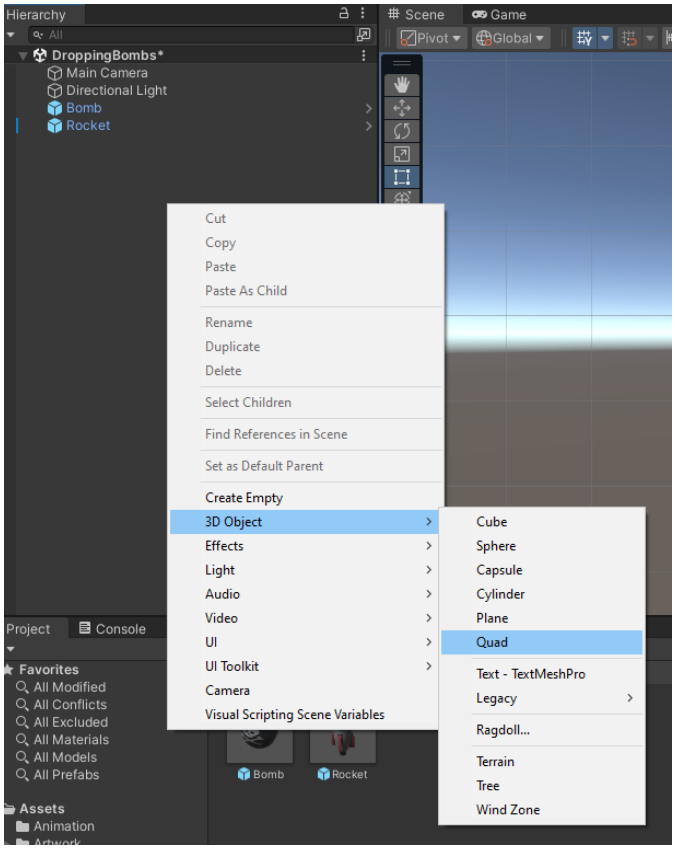
42 Once back in Unity, attach the **RocketAnimation** script to the rocket **Prefab**. Since we are using **GetComponent<>()** in our code, we don't need to do anything else. You can enter play mode and see the animation work. You can also see the **Animator** variable we made in the inspector find the animation on the Rocket.



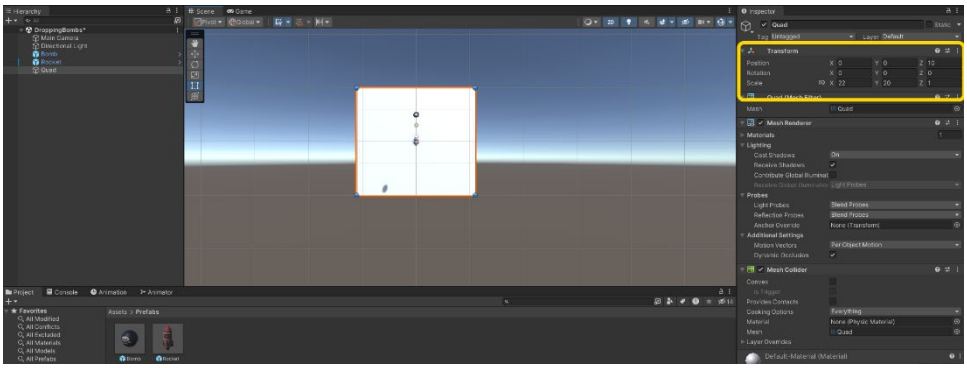
43 Once you've playtested your game, you should notice the rocket leans left and right as it flies. There's still more we can do to make it feel like it's flying.

When you are finished testing, stop the game by clicking the **Play** arrow again.

44 Where do rockets fly? In the sky, of course! Before we can add a sky, we need a place to put it. Right-click on the **Hierarchy** and select **3D** Object, then select **Quad**.

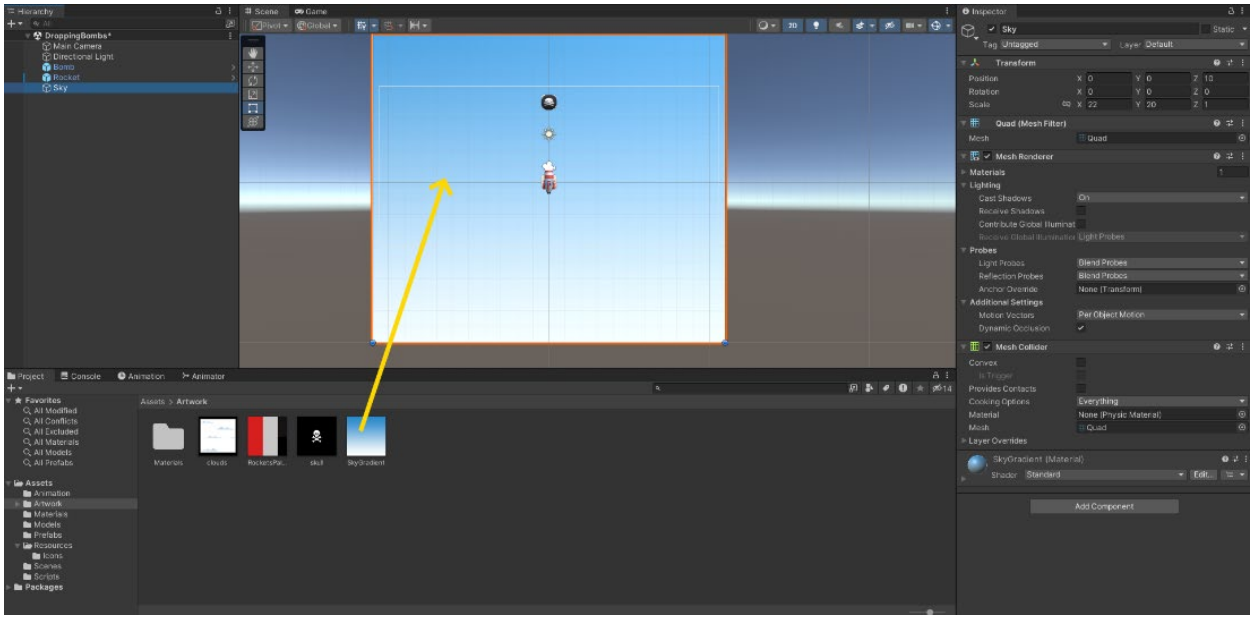


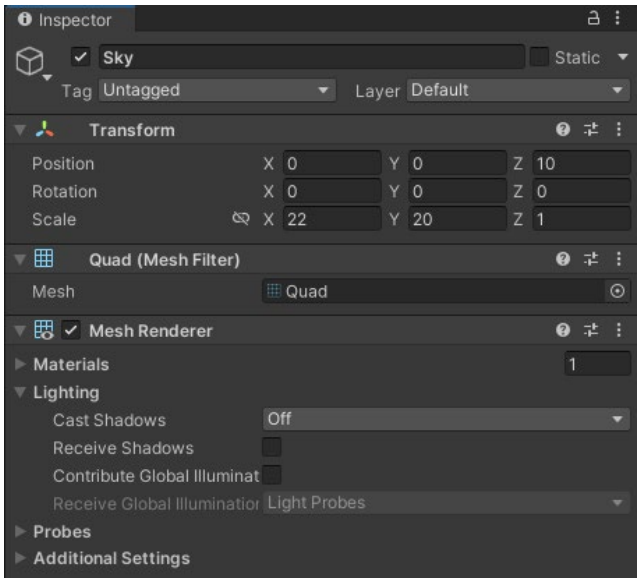
45 Make sure the new **Quad** is in the middle of the scene at X:0, Y:0, and set the Z:10. Resize the Quad so it fits the white outline of the camera from as seen in the image below.



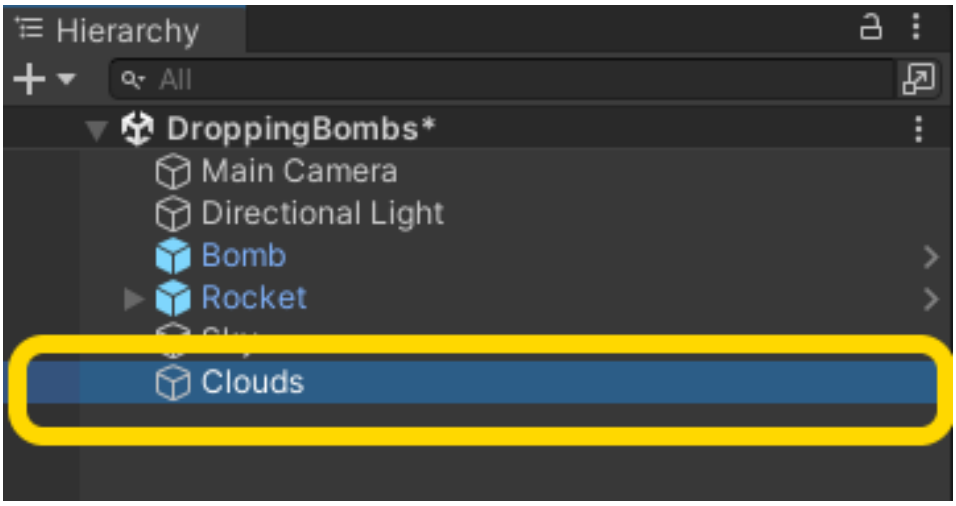
46

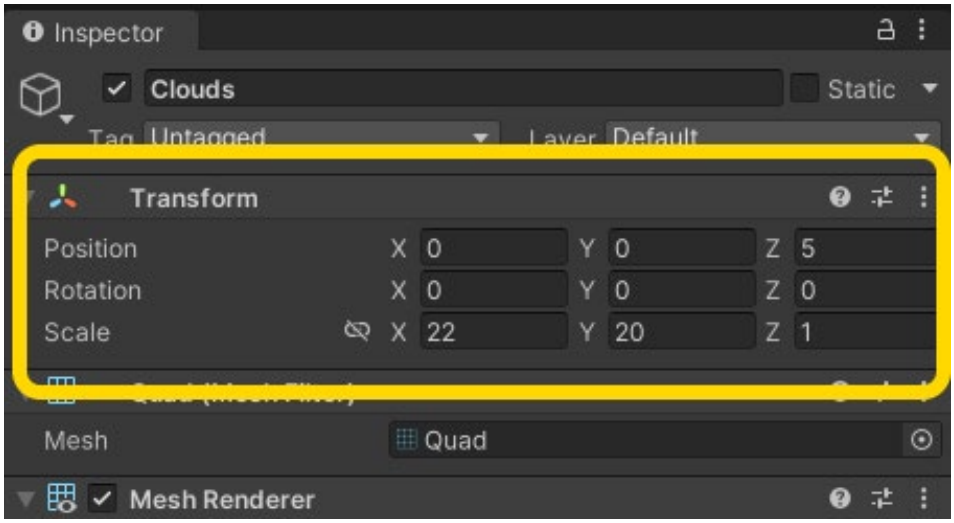
Rename the **Quad** to ***Sky***. Drag the **SkyGradient** texture from the **Artwork** folder onto the **Sky GameObject**. In the **Mesh Renderer** component, change **Cast Shadows** to **Off** and make sure that **Receive Shadows** in the Mesh Renderer is unchecked.



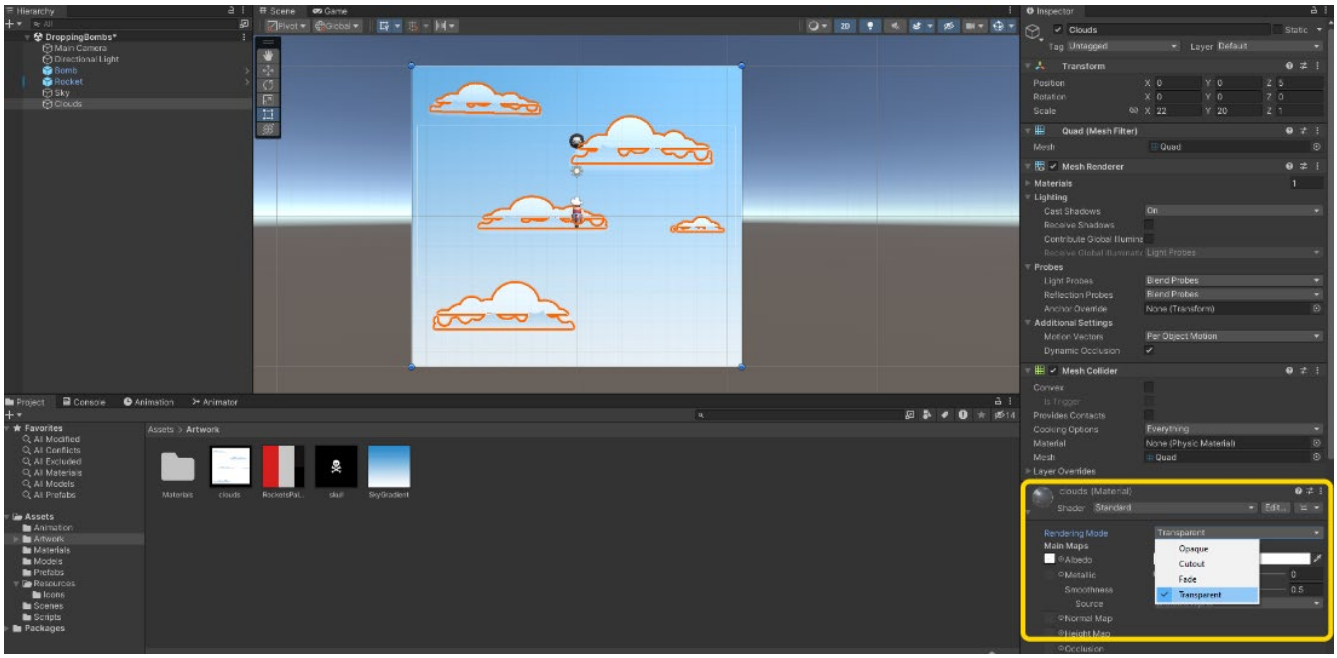


47 Press **Ctrl+D** to duplicate the **Sky GameObject** and rename the new object ***Clouds***. We want the clouds in front of the sky, so change the Z value of **Clouds** to **5**.

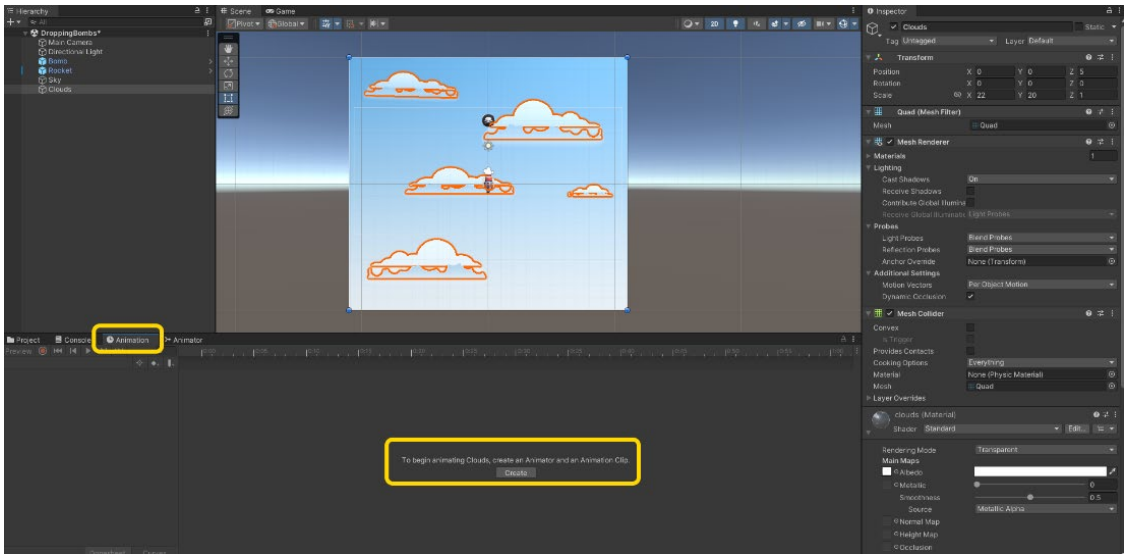




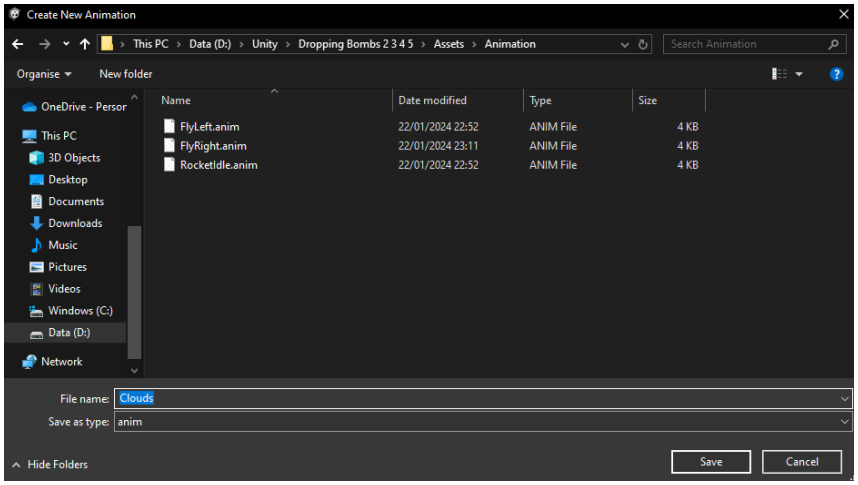
48 Drag the **clouds** texture from the **Artwork** folder onto the **Clouds GameObject**. In the **Inspector** panel, find the **Mesh Renderer** component and set **Cast Shadows** to **Off**. Then, go to the **clouds Shader** and change the **Rendering Mode** to **Transparent**. If you can't see this option, click the arrow on the material to reveal all the material options.



49 With the **Clouds GameObject** still selected, click the **Animation** tab and select **Create** to make an animation for the clouds.

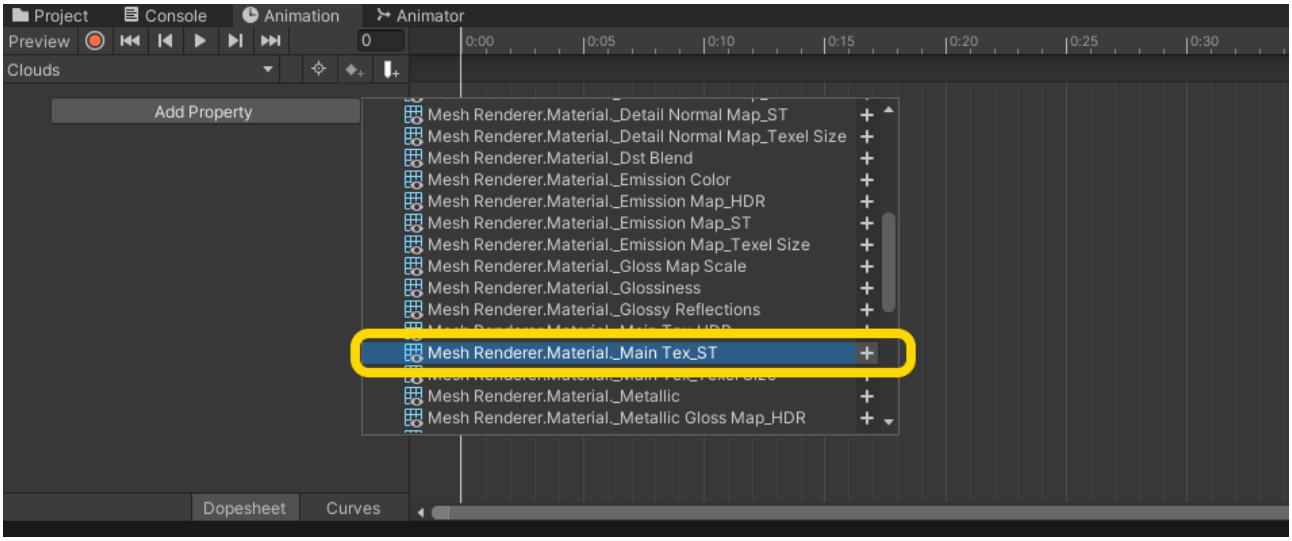


50 Make sure that the new animation is in the **Animation** folder and name it ***Clouds***.



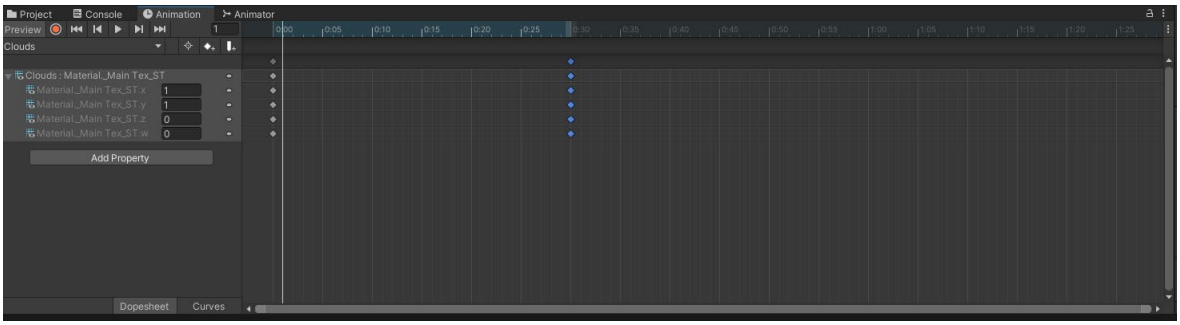
51 Just as you did with the **Rocket** animation, click on **Add Property**. This time, you'll animate the cloud texture which is part of the **Mesh Renderer**.

Expanding the Mesh Renderer shows other properties that can be animated. We want **Material_Main_Tex_ST**. Click the plus (**+**) symbol to add it to the **Animation** panel.

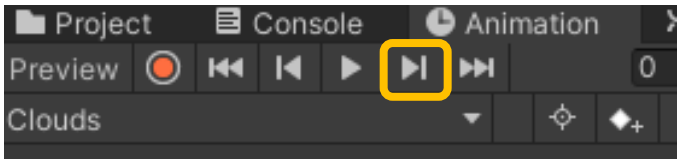


52 As with the **Rocket** animations, the length of the animation is 1 second long by default. Let's make it half that length. Click the top key frame diamond at the 1 second position and drag it over to the 0:30 second position as shown.

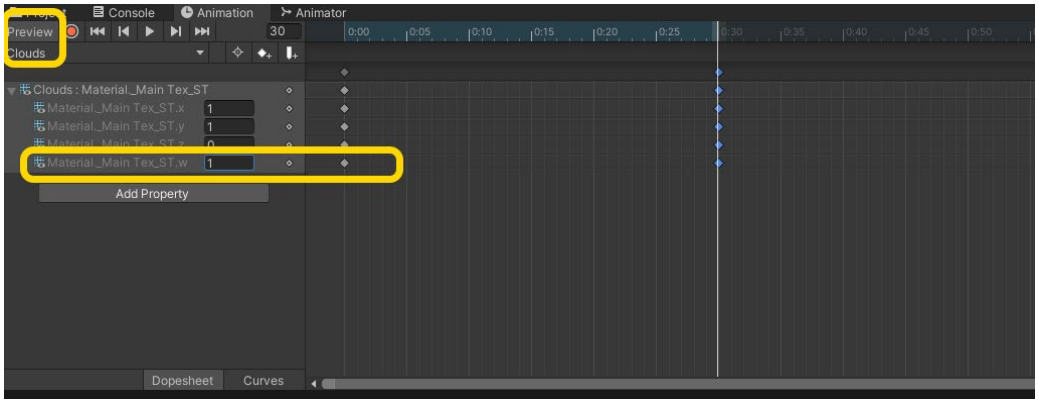
53 The animation timeline has a vertical white line (known as the **Playback Head**) at the 0 position. This line indicates where in the timeline that you're making your changes. For the clouds, we want to make our change at the very end of the timeline, at the 0:30 second position. Drag the white line over to that end of the timeline.



You can also use the preview controls to send the playback head to the previous or next keyframe.

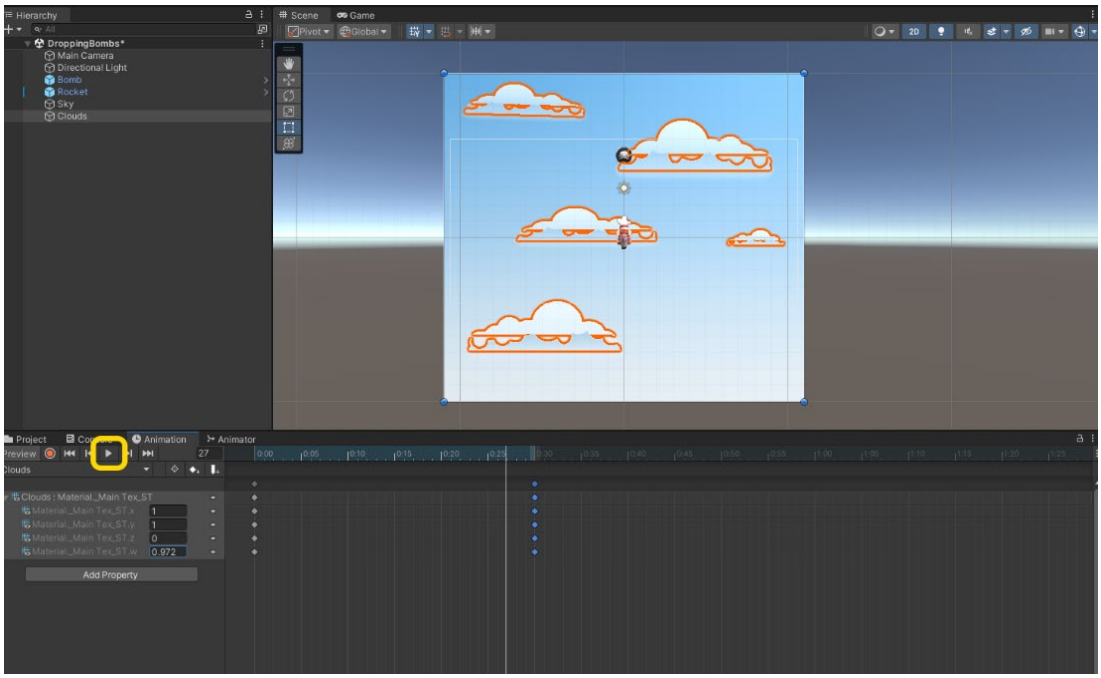


54 Make sure that you're at 0:30 in the timeline as shown below. Of the four properties, the only one that's being changed is **Material_Main_Tex_ST.w**. This controls the vertical offset property of the texture. Change it to **1**.

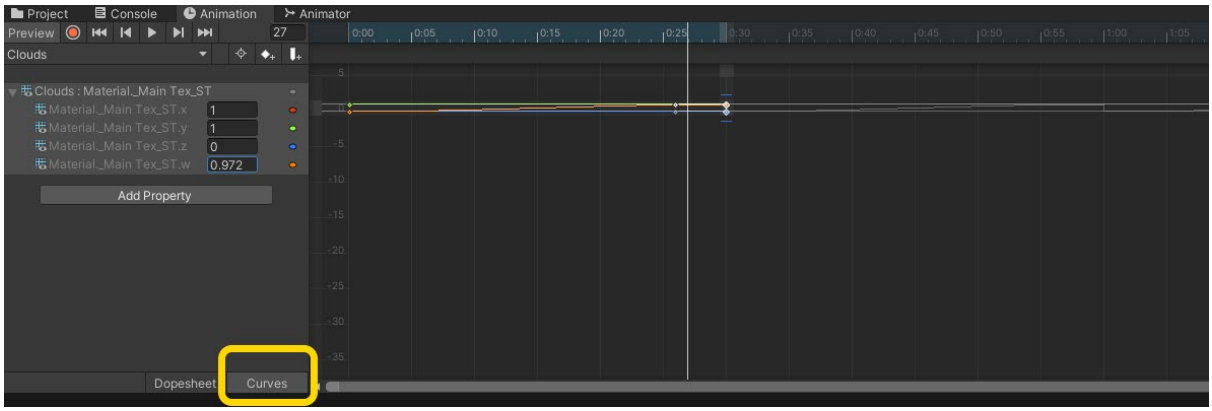


55 Click the black "**play**" arrow in the **Animation Preview** panel to check your animation. From 0:00 to 0:30, Unity is gradually changing the offset value from 0 to 1 and making it look like the clouds are moving.

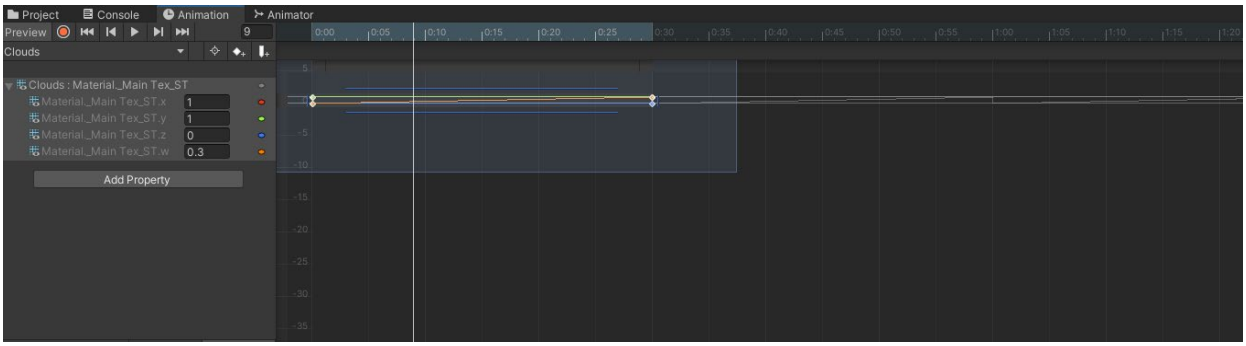
If it looks like the clouds aren't moving smoothly, there's a fix for that in the next few steps.



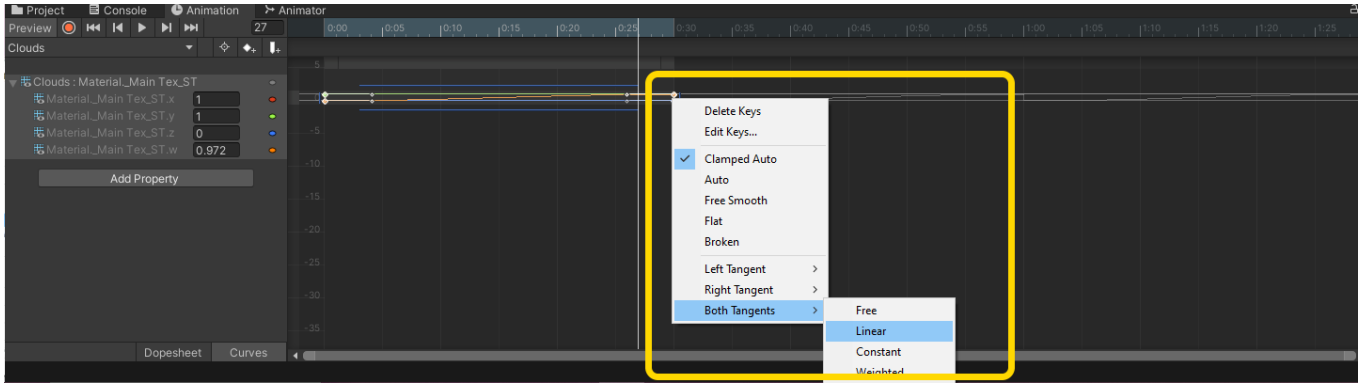
56 If the clouds look like they're lurching (speeding up and slowing down over and over again), it's because Unity **Animation** has automatically added "*easing*." Instead of a straight line from the beginning to end, the animation is a curve. To fix this, click **Curves** in the **Animation** panel.



57 This view is a diagram showing how the properties change as the animation plays. Select all of the points in the diagram by dragging a rectangle around them as shown.



58 With all of the points selected, right-click on one of the points on the right side to open the menu shown below. Click **Both Tangents** and select **Linear**. This changes the animation from a curve to a straight line.



59 Now play your game. It should look like the rocket is flying higher and higher. However, the game doesn't have scoring and the way it resets is rather abrupt. We will tackle that in the next activity.