Making an Endless Runner Game in Unity

GDV4000 Introduction to Games Industry Practice

Unity Workshop 1

Based on research and notes by Dr Glenn L. Jenkins

Problem: How do I add a Player Sprite in Unity?

1.1 Introduction

This worksheet was written for Unity **2022.3.31f1 but has been tested on Unity 6000.3.4f1 (Unity 6.3 LTS)**

The artwork used during this worksheet are credited to Free Art Sprites, but the site seems to no longer exist.

We are going to be focusing in this worksheet on importing images into Unity and creating and transforming them into sprites to use within a scene. We will start with bringing in a single sprite, and then a sprite sheet. The sprite sheet will be a precursor to animating a sprite. Here we are using a problem-solving strategy – Our overall aim is to add a player character, but the boulder is a single sprite and therefore a simpler problem to solve first.

Either using previous worksheets, or from your own practice, ensure you have created a Repo called 'Endless Runner' to create this project into (You can call it what you like, but remember you're going to have to find it again. Don't forget to add the Unity specific .gitignore, and save the scene so you can give it a more useful name than 'untitled'!

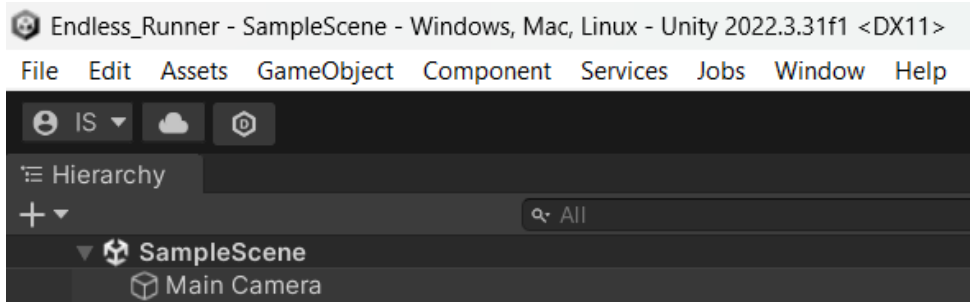


Figure 1.1: Screenshot: The 'Untitled' project now saved as 'Endless_Runner'

Problem: How do I add a Player Sprite in Unity?

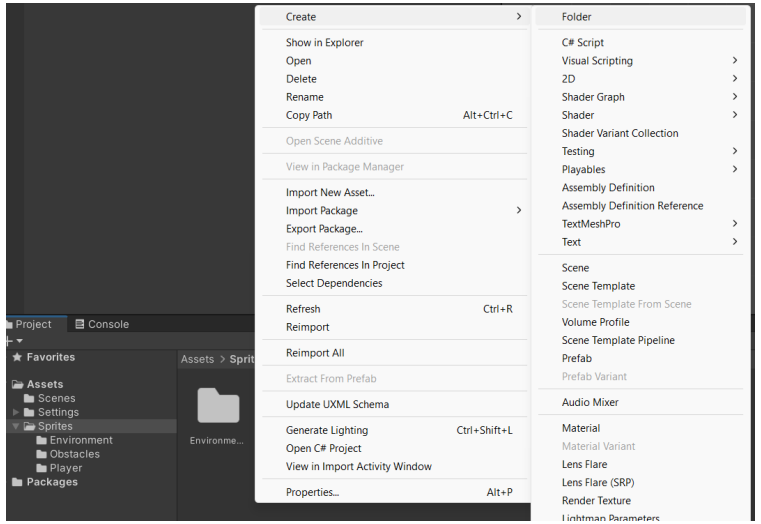
1.2 Importing a Single Image Asset

We need an image to use as a sprite. For this project the assets are provided for you to work with, but there is nothing stopping you from finding alternatives. The site that the sprites provided came from is no longer up, but there are plenty of others that you can search.

But before we add any assets, we need somewhere to store them where we can easily find them. Let's create some folders for our assets in the **Project Browser**. At this point we only have a few assets, but as our project grows in size it will benefit us to keep things organised.

This is a very good practice to get into for all your work.

To create sub-folders in the Assets folder, open it in the **Project Browser** window, right-click to open the context menu, and then select Create -> Folder. This will create a new folder that you can name. For this project we have a sub-folder in Assets called Sprites, and then three sub-folders in Sprites: Environment, Obstacles, and Player.



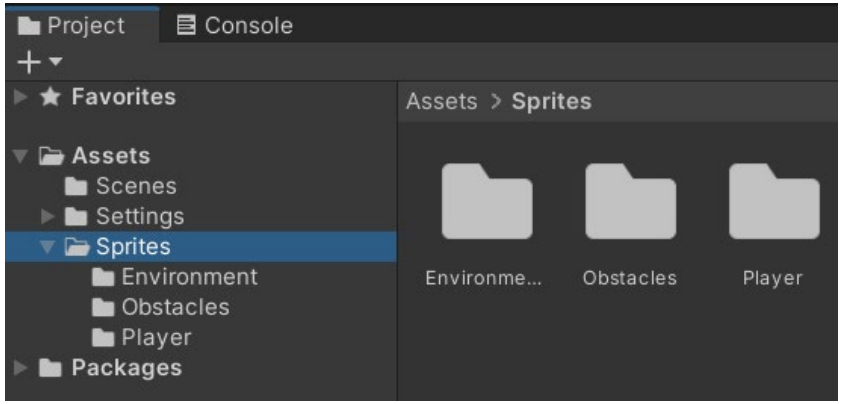


Figure 1.2: Screenshot: New folders – Structure is important!

Problem: How do I add a Player Sprite in Unity?

1.2.1 Getting the Image into our Unity Project.

This is a very simple step – We're going to drag the Boulder Image (Boulder.png) into our Obstacles folder.

You may have noticed that this is not our player sprite, which would seem to contradict the goal of the worksheet. What we are doing is employing a problem-solving strategy, by identifying and solving a simpler problem: "How to add a boulder sprite for the obstacles".

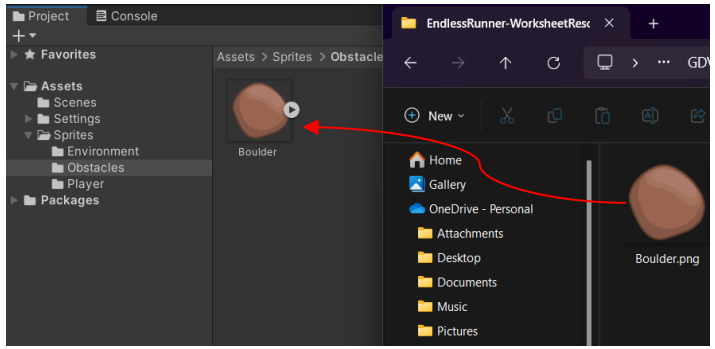


Figure 1.3: Screenshot: Dragging the Image into the Project Browser window.

1.2.2 Sprite Settings in the Inspector.

If we select our image in the Assets\Sprites\Obstacles folder, we can see some information about its properties in the Inspector pane (right side of the Editor). The sections we are interested in are:

- Texture Type
- Sprite Mode
- Pixel per Unit
- Filter Mode
- Platform Settings (at the bottom with three tabs)

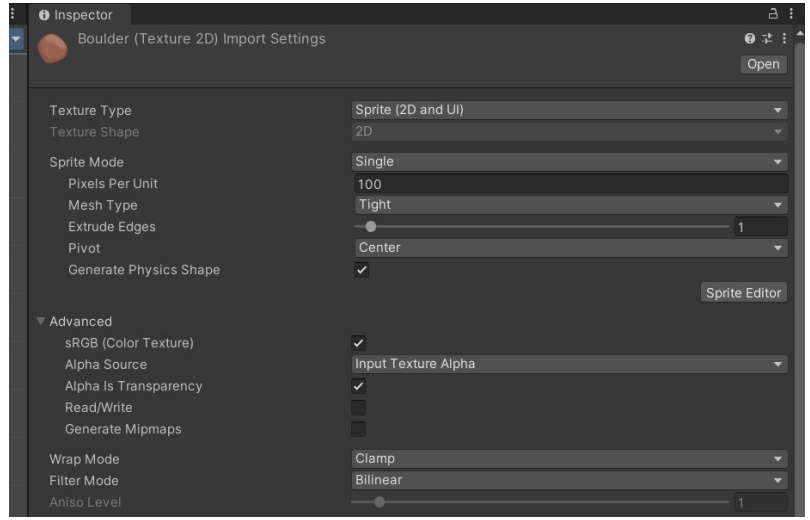


Figure 1.4: Screenshot: Our Boulder image's properties in the Inspector pane.

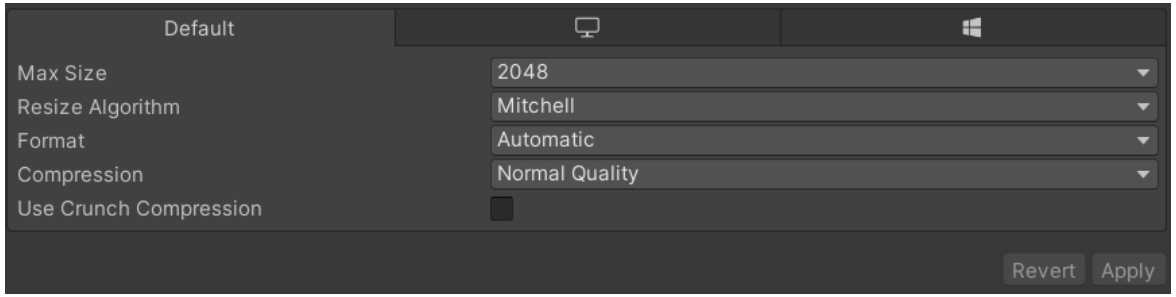


Figure 1.5: Screenshot: Platform Settings box.

Problem: How do I add a Player Sprite in Unity?

We can run with the default settings for this image, but we will review the options we have here.

Texture Type

The selected option is ‘Sprite’, which is intended for use with 2D sprite images. There are many different options for image types, e.g. Normal Map, Light Map, Cursor etc. These will likely come up in future studies (Normal Map definitely next year). It is important to use the correct texture type as this will dictate how Unity displays the image, and the options available in the Inspector.

Sprite Mode

We have two options: Single or Multiple sprites. For our Boulder image we are using Single, and the reason for this will become apparent later.

Pixels per Unit

The number of pixels in the image (Height and Width) that correspond to one distance unit in Unity’s world space.

Filter Mode

This determines how the sprite texture is filtered, essentially smoothing the image to prevent in looking blocky up close. Texture maps are generally square or rectangular (32x32, 64x64, 128x128, 256x265, 512x512, 1024x1024…), but after mapping to a polygon, or a surface, and transforming them into screen coordinates, the pixels in the texture (texels) rarely conforms to the pixels of the image. We are essentially either taking a group of pixels and spreading a single texel over them, or taking a group of texels and mapping them to a single pixel. Unity utilises the following algorithms to perform this magnification/minification:

Problem: How do I add a Player Sprite in Unity?

We can run with the default settings for this image, but we will review the options we have here.

1. Point: This is the option with the lowest computational cost and best for performance. Useful for small images at some distance from the player, but up-close they will appear blocky.
2. Bilinear: More expensive than Point, it applies some smoothness to the image to remove blockiness.
3. Trilinear: Similar to Bilinear but more expensive (takes longer to process) as it transitions between MipMaps (a technique where lower resolutions of the texture are created for when the player is further away, reducing memory cost).

For our purposes, we can stick with Bilinear as we are not using MipMaps.

Mesh Type

The two options here are Full Rect or Tight. Full Rect maps the sprite onto a rectangle, whereas Tight creates a mesh that follows the shape of the sprite more closely, based on the Alpha value of the pixels. Sprites smaller than 32x32 automatically use Full Rect.

Extrude Edges

Controls how much of an area to leave around the generated sprite mesh.

Problem: How do I add a Player Sprite in Unity?

Pivot

The location in the image where the local coordinate system originates. You can use a custom value or a preset one. A useful way to think about the pivot point is where you would want to move the sprite from. The image on the left of Figure 1.6 has its pivot in the centre at (0,0) in 2D world space. The image on the right has the pivot point in the top left corner, but it is still at (0,0). This option only appears in the Inspector when you select Single in Sprite Mode.

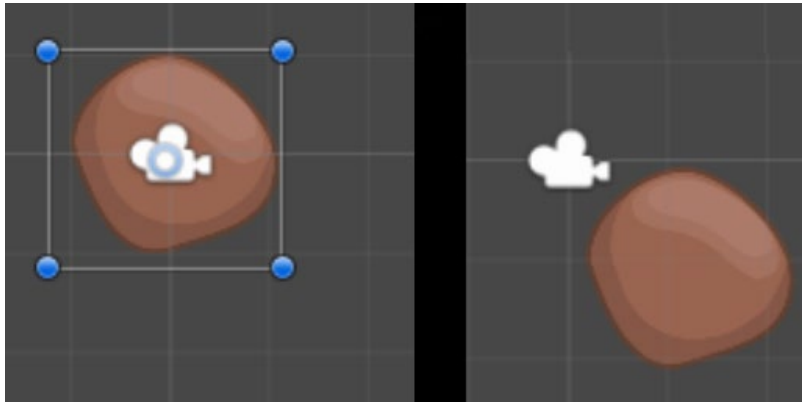


Figure 1.6: Screenshot: Illustrating the difference in Pivot Point on a sprite

These are the main options we may need to alter. For further reading on the others, I refer you to Unity's documentation: https://docs.unity3d.com/Manual/texture-type-sprite.html

Problem: How do I add a Player Sprite in Unity?

1.3 Adding a One Image Sprite to the Scene

This will be done in two short stages. Firstly, we want to create a Game Object in our scene to represent the sprite and thenwe will attach the sprite to it.

- ### 1.3.1 Creating the Sprite Game Object

  There are a few ways to do this, and with practice and research you will develop your own ways of working. But for now, we'll just look at one way.

  Right-click on the Scene in the Hierarchy Panel, GameObject -> 2DObject -> Sprites -> I have selected square, but as you can see there are many options here. Unity docs will help you to understand where and why these may be useful in different situations. For now, we just want to solve our problem of getting the asset into our game.

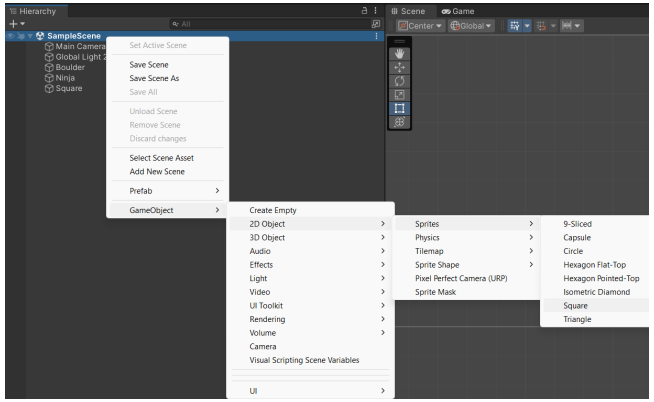


Figure 1.7: Screenshot: Creating a GameObject

- ### 1.3.2 Assigning the Asset

  The simplest way to assign the asset to the GameObject is to drag it from the Project window at the bottom onto the Sprite field in the Inspector (if you can't see this, make sure your empty GameObject is selected in the Hierarchy Panel first).

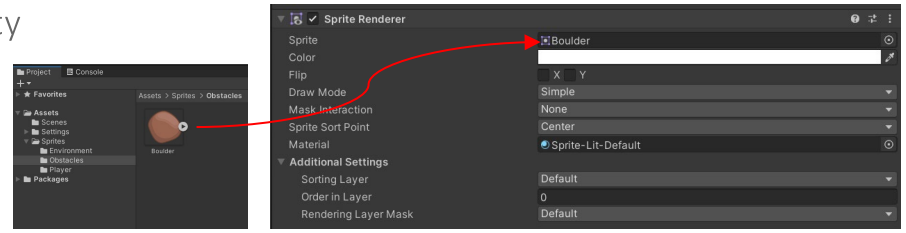


Figure 1.8: Screenshot: Assigning the sprite Asset to the GameObject by dragging.

Problem: How do I add a Player Sprite in Unity?

Now you can drag the Asset from the Project Window and into the main Editor window. We want to place it outside our Game Window for now. Why? Because a little later we are going to be making copies of it at this position on the X and Y axis by a random amount. They will move towards the player, with the background scrolling at the same speed so it looks like the player is moving and being tracked by the camera – This is a very old trick!

Use the same values as Figure 1.9 for your Boulder. Pay attention to the Scale too. We are scaling it down by half the size on the X and Y. You may notice that we are setting the Z Axis to -1, even though we're operating in 2D…

This is deliberate. Unity cheats a little when it comes to 2D, as it's 2D but still working within a 3D engine. Try to image that your sprites are like sheets of paper being placed on each other. The background would be at 0 on the Z Axis with the other sprites in front (Figure 1.10).

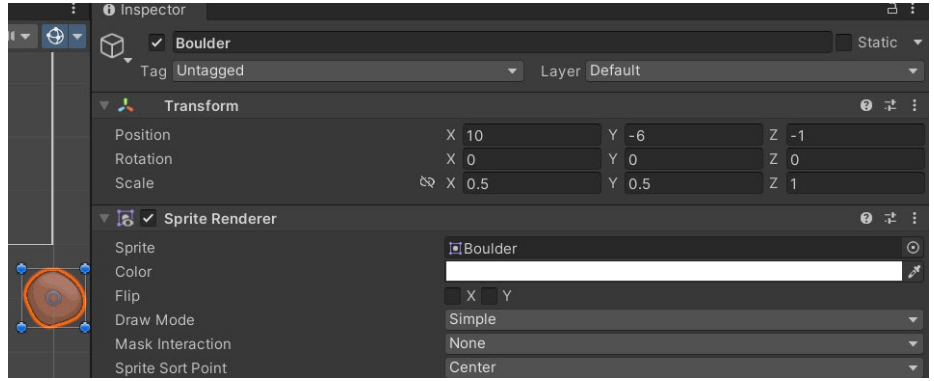


Figure 1.9: Screenshot: Placing the Boulder GameObject and scaling it.

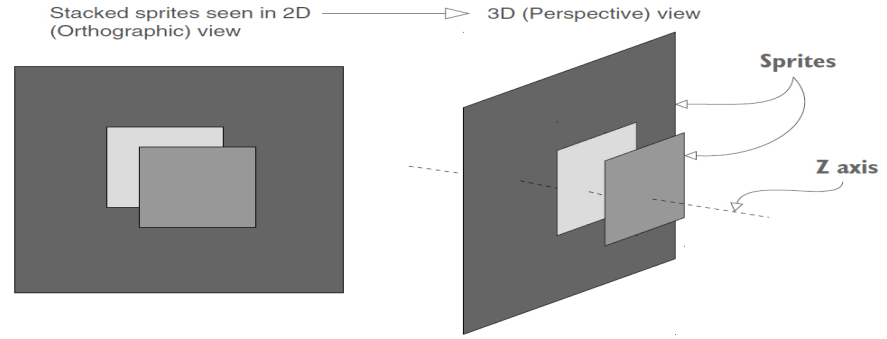


Figure 1.10: Screenshot: Sprites arranged along the Z Axis (Borrowed from Hocking *et al).*

1.3.3 Unity's Component Orientated Architecture

Game objects in Unity utilise a modular component system. In simple terms, the game engine is comprised of interacting software entities created from its own blueprint, unlike a more traditional inheritance model.

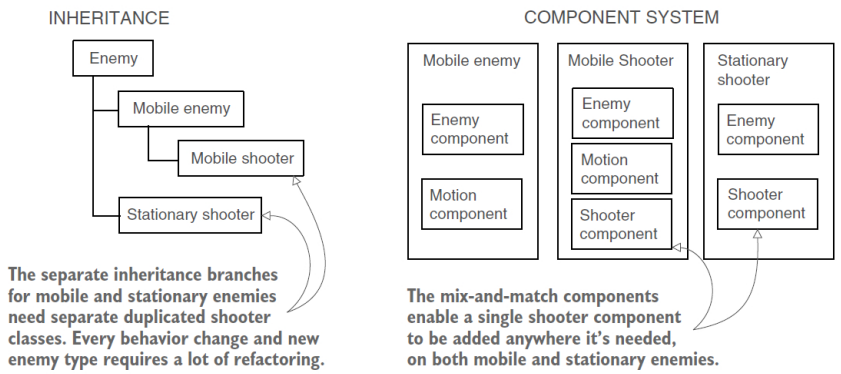


Figure 1.11: Inheritance will mean that refactoring how objects shoot will mean many changes to classes (which we want to avoid), whereas the Component System does not.

Problem: How do I add a Player Sprite in Unity?

There is a downside however, because the GameObjects will need to have their components connected together, either through the Editor or by code. For example, to ensure that a Particle Effect is activated when an object is hit. GameObjects also need to communicate with other GameObjects and initiate actions or behaviours in other GameObjects as a result. For example, a missile GameObject hitting an NPC GameObject must destroy itself and cause damage to the NPC.

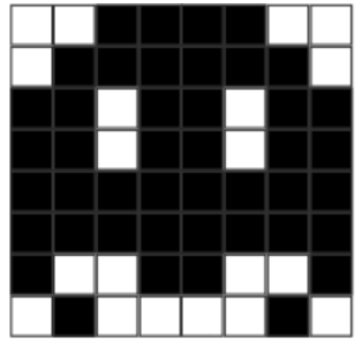
- 1.4 Adding a Sprite from a Sprite Sheet

We have added a single sprite from a single image, but what about a sprite from a Texture Atlas (also referred to as a Sprite Sheet)?

Wow there, what's a Texture Altas?

Back in the days of the ZX Spectrum, graphics were designed on graph paper and then coded using hex values converted to decimal. By the time of the 16bit era, interfaces and art packages for drawing sprites were more commonly used, such as *Degas Elite* on the Atari ST and Commadore Amiga.

The advantage of using a Texture Atlas is the difference on memory between loading in a lot of individual sprites and one single sheet with all the sprites on it.



Doing this in Unity is fairly straightforward, as the built-in tools do a lot of the work for us. The downside is that we need to learn those tools. If we were building our own engine for our game, we could set a standard for our sprite sheets and our assets to conform to it.

Sometimes what can seem to be a flexible workflow can in practice create more complexity!

Let's start by importing the sprite sheet we will use for our Player. For reasons best left to the imaginations of our players, it's a Ninja (**NinjaSpriteSheet.png** and it should be in the folder on Moodle marked **EndlessRunnerResources**).

Digit 1	Digit 2	Digit 3	Digit 4	Digit 5	Digit 6	Digit 7	Digit 8
1=1	1=2	1=4	1=8	1=16	1=32	1=64	1=128

Problem: How do I add a Player Sprite in Unity?

- 1.5 Slicing the Sprite Sheets with The Sprite Editor

Slicing is the process of defining which parts of the Sprite Sheet contain individual sprites. Unity can do this in two ways: Automatic and Manual. Automatic assumes that the sprites are ordered in a regular pattern, Manual allows for more irregular ordering.

The first step is the same as we did for our Boulder, but as we are dealing with multiple sprites and not a single sprite, we need to change Sprite Mode to Multiple.

Open the Sprite Editor to start working with the sprite. With the sprite sheet selected in the Project Window, click the Sprite Editor button. This will launch the Sprite Editor Tool.

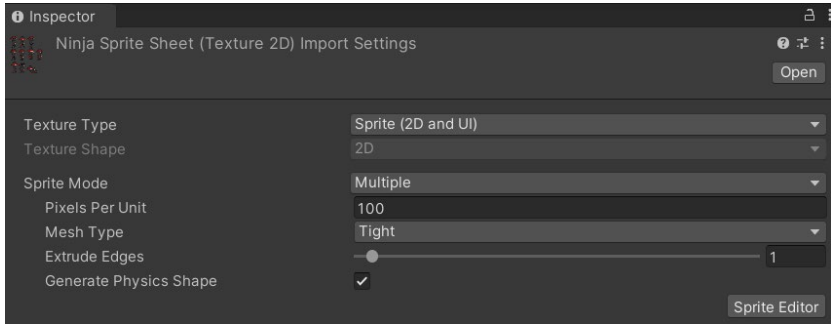


Figure 1.12: Screenshot: Player sprite settings – Note the Sprite Editor button bottom right.

- 1.5.1 The Sprite Editor - Manual Slicing

We are **not** going to use Manual Slicing for this sheet as it is more of a dope sheet with the animations arranged in rows. The Manual Slicing method allows us to create our own slices and size them, arrange them and name them. If you're not happy and want to start over, you can also Revert what you have done. If not, hit Apply.

- 1.5.2 The Sprite Editor - Automatic Slicing

There are various forms of automatic slicing in Unity. The first flavour of 'Automatic' best-fits a rectangle to a sprite, assuming that whatever isn't transparent is part of the sprite. In case of artifacts that we don't want to include, we can set Minimum Size to prevent these being included.

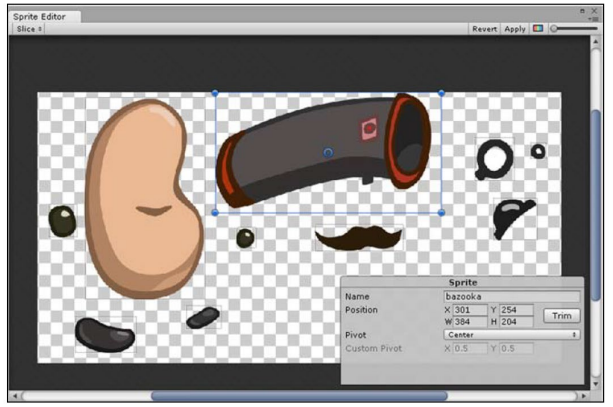


Figure 1.13: Screenshot: Sprite sheet with non-uniform sprites. This would need to be animated by Unity's animation system (From Pereira).

Problem: How do I add a Player Sprite in Unity?

We're going to use **Automatic** for our Ninja sprite sheet. **Grid By** could be an option as know the number of sprites and can work out their size.

Select Slice from the top left of the menu bar. Click the Slice button again and Apply or Revert depending on if you're happy with the selection.

In the Project Window, the Ninja sprite sheet now has a little arrow on the icon. Selecting this will show all the individual sprites contained in the sheet (after slicing).

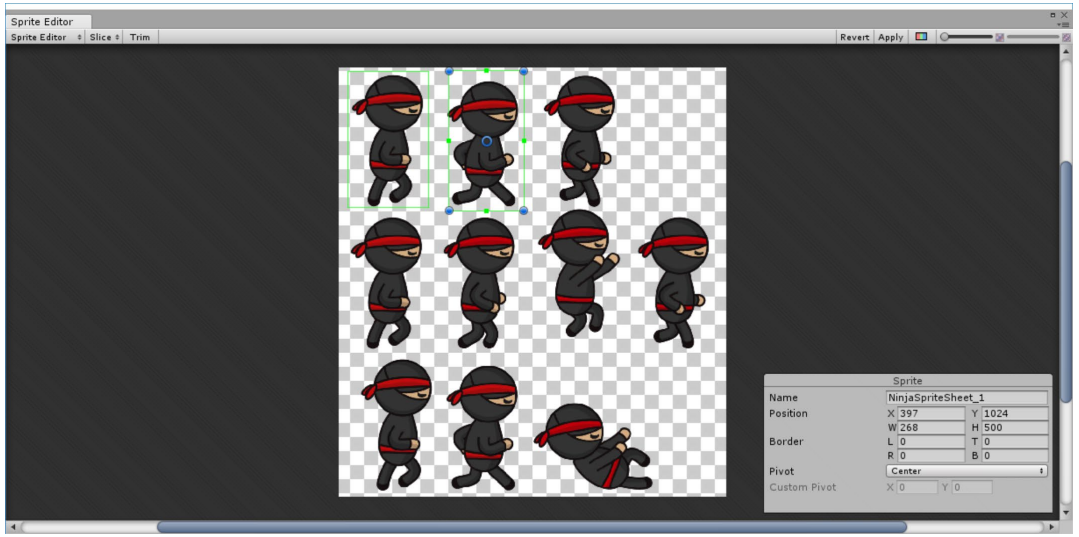


Figure 1.14: Screenshot: Manual Sprite Sheet slicing.

- 1.5.3 Adding the Player Sprite

Create a new GameObject (of type Sprite, just as we did for the Boulder) and attach the first image from the sprite sheet. Rename the GameObject 'Ninja' and set the position to -6.0, -2.0, 0.0, the rotation to 0.0,0.0,0.0 and the scale to 0.75,0.75,1.0.

Problem: How do I add a Player Sprite in Unity?

For your own practice –

1. We've added a boulder. Now add the Background image (background.png) the position is (-0.1, 0.0, 1.0), rotation (0.0,0.0,0.0), and scale (2.2, 2.5, 1.0).
2. Find some other objects that can be used as obstacles in our game.
3. Have a go at editing the Ninja sprite – change the colour of his clothes?

If you get stuck, try looking at Unity's online resources - https://learn.unity.com/ to help you.

Making an Endless Runner Game in Unity

GDV4000 Introduction to Games Industry Practice

Unity Workshop 2

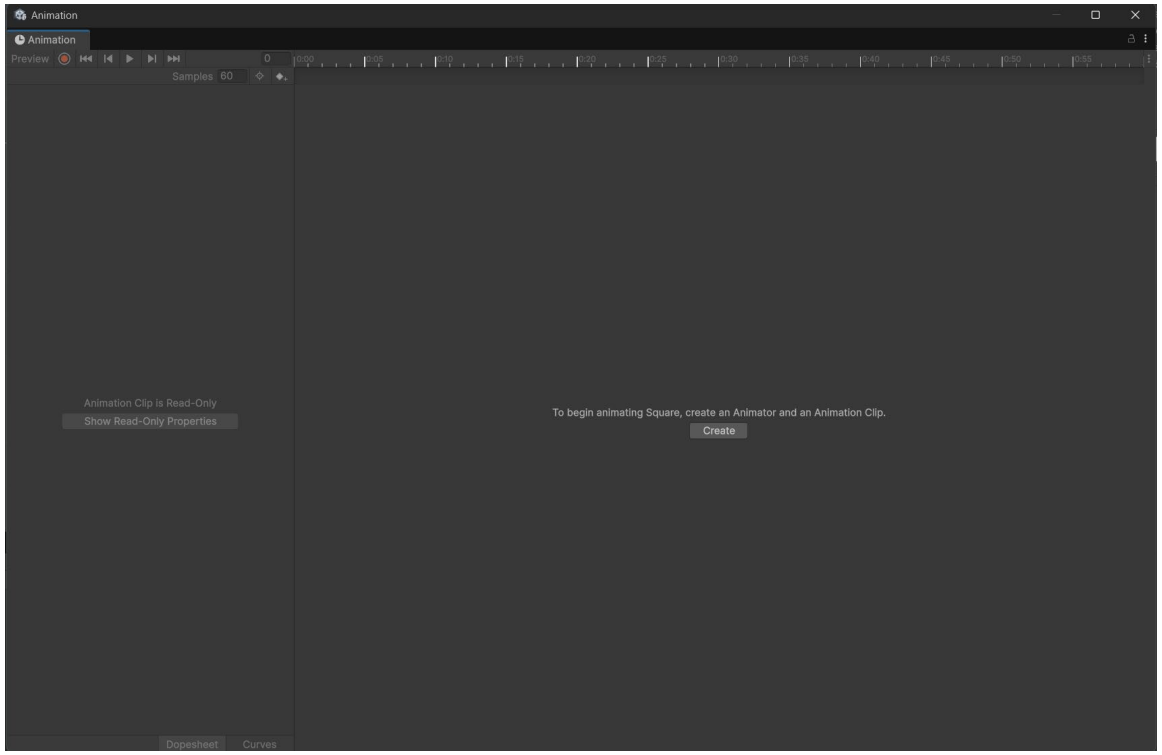
Based on research and notes by Dr Glenn L. Jenkins

Problem: How do I add animation to my Player Sprite in Unity?

So… How *do* I create a Player Sprite in my Endless Runner game?

The solution to this problem is to create an animated sequence from the individual sprites, making use of the Animation Editor.

To open the Animation Editor go to Window-> Animation -> Animation (or Ctrl+6)



If you do not have a component selected which can be animated, such as a sprite, you will be asked to select one before you can create an animation.

Using the Animation Editor, we can see our dope sheet 2D animations to use frames on a time-line and curves. The term ‘dope sheet’ comes from traditional animation in cartoons and film/TV. Like a storyboard, it is used to plan out the animation.

Problem: How do I add animation to my Player Sprite in Unity?

To set up a running animation for our Ninja character, we are going to follow these steps:

1. Select the player sprite in the Hierarchy panel on the left-hand side.

   You can do this either with the Animation Editor open or select the sprite and then open the Editor.

2. Unity will want to create the animation file before rather than after we have created our animation. Place this somewhere you will be able to find it easily. You may want to keep all of your animations together in an Animations folder, or you may want to keep them with their sprites. I am going to do the latter. Click Create and name your file something like 'Running'.

3. Click the arrow to the right of our NinjaSpriteSheet in the Project Window. It will expand to show all of our NinjaSprites.

4. Ctrl+Left-click the first three sprites and drag them onto the Animation Editor window. You should see three blips appear. These are key-frames.

5. Set the Samples to 10. This is our FPS, which at default is set to 60.

6. If you press the Play button on the Animation Editor, you should see the Ninja running.

Problem: How do I animate (rotate) an object with only one sprite?

Whilst not part of the original specification, learning how to use an animation curve to animate the object's Transform (in this case, rotation) will help us with creating our Jumping animation later; which will involve animating the sprite at the same time as adding movement.

Bu doing this we are learning about the problem we are about to solve…

As we did with our Ninja, select the Boulder sprite and open the Animation Editor. Click the Create button and name the animation something like Spin (remember to save it in the same place as the Boulder sprite).

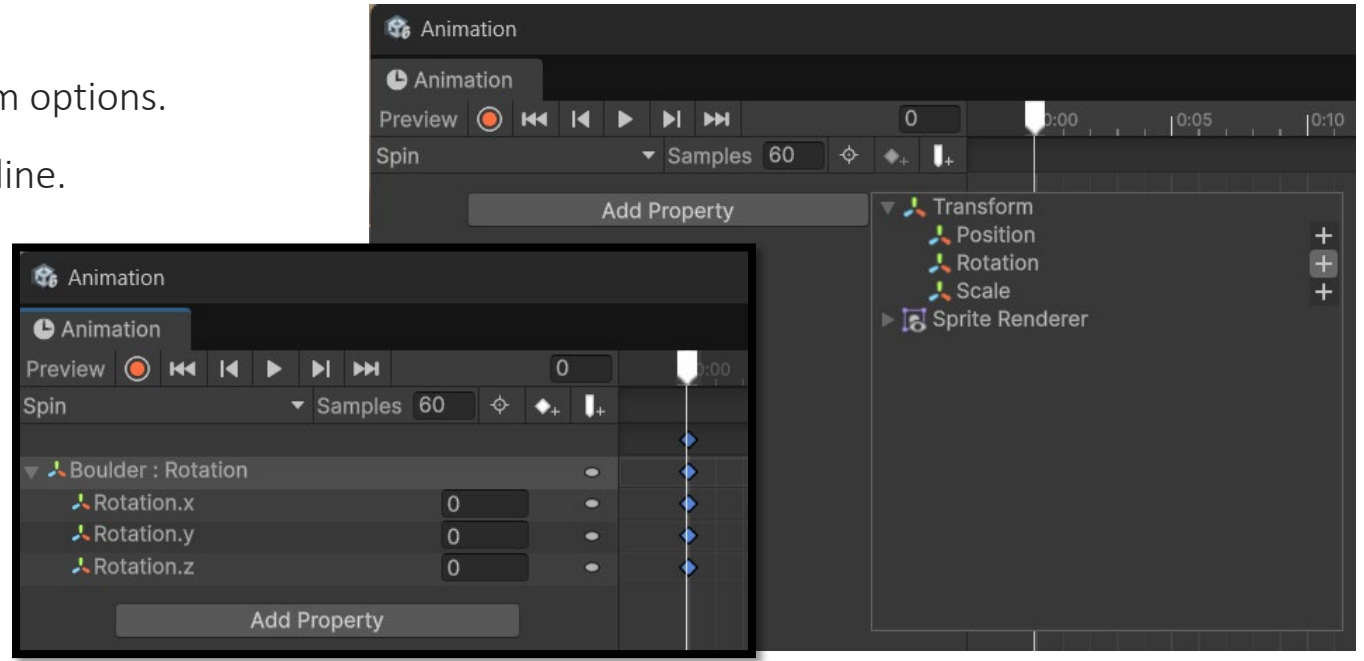
Rather than dragging sprites onto the Editor, this time we click the **Add Property** button.

From the drop-down menu, select Transform and Rotation.

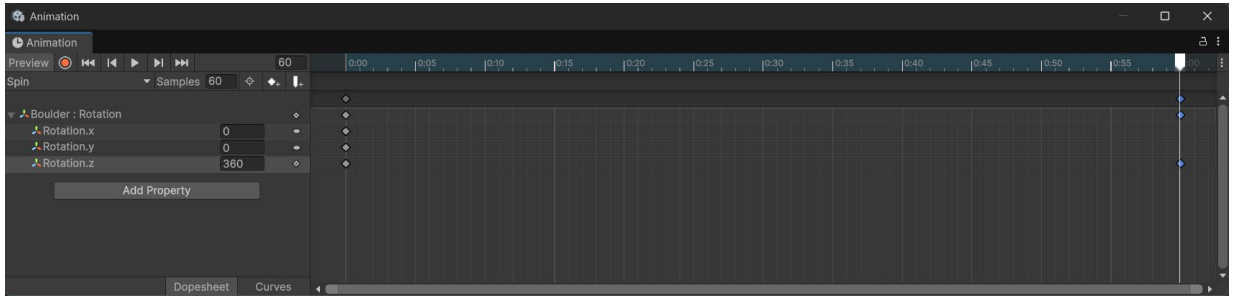
You need to click the arrow to the left to open the Transform options.

Click the + icon to the right of Rotation to add it to the timeline.

Click the arrow to the left of Boulder : Rotation to reveal the axis.
We can rotate it on any axis, so we need to be sure we are using the correct one.
The blue blip at 0.0 on our timeline is our first keyframe.



Problem: How do I animate (rotate) an object with only one sprite?



We want the Animator to create the animation for us by taking two rotation values and then interpolating between them. We have our first key-frame at 0 degrees. To complete a full spin, our last value needs to be 360 and we need to make sure that we are doing this on the Z axis only.

With Rotation.z selected, move the white bar to the last position (as above) and click the Add Keyframe button (white diamond with a + beside it). We should now have blue diamond blips at the end of our timeline. Change the Rotation.z value to 360.

Press Play and observe the result.

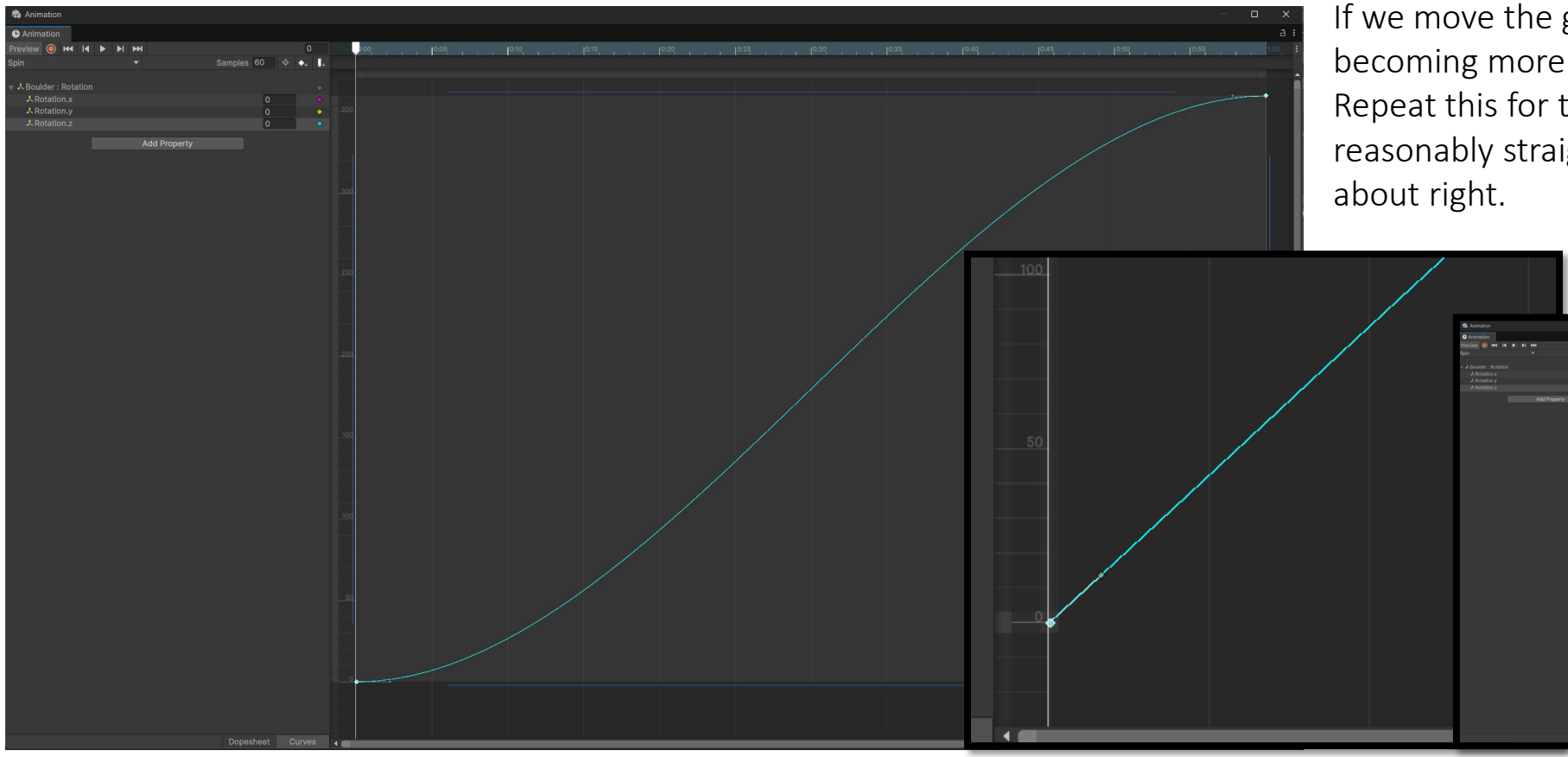
We can see that the rotation does work… but it looks a little odd. There is a slow down at the start and end of the animation cycle.

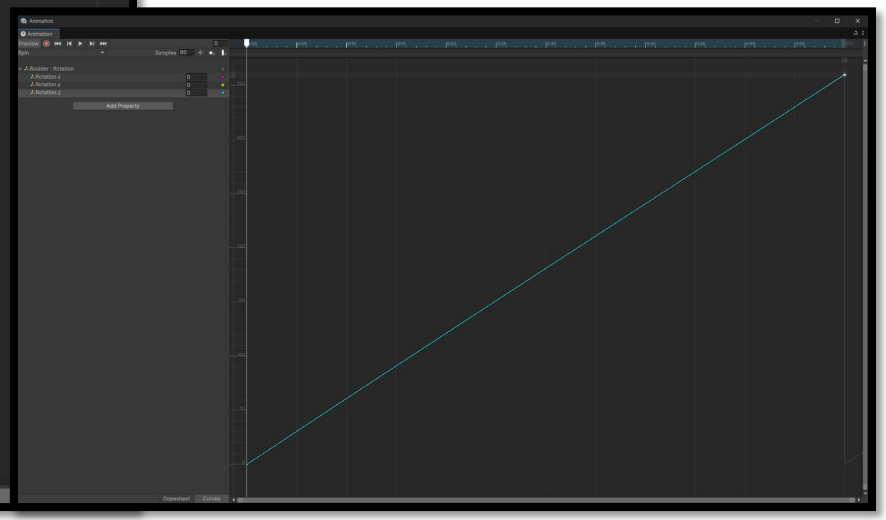
At the bottom right side of the Animation Editor, there are two tabs: Dopesheet (which we are on) and Curves. Click Curves

Problem: How do I animate (rotate) an object with only one sprite?

This is a graphical representation of how the numerical values of our rotation are changing over time. This shows us why our animation is behaving the way it is. In order to have a constant speed, our line should be a straight diagonal one. Thankfully we can change this.

If we click on the keyframe at 0, we should see a grey bar sticking out from the right-hand side. Our line is in fact a Spline with two control points (you'll encounter Splines more when you do GDV5002 next year). The grey bar can be used to control the curve amount.





If we move the grey bar up, we can see our curve becoming more straight.
Repeat this for the point at end until the line is reasonably straight. Tinker and experiment until it feels about right.

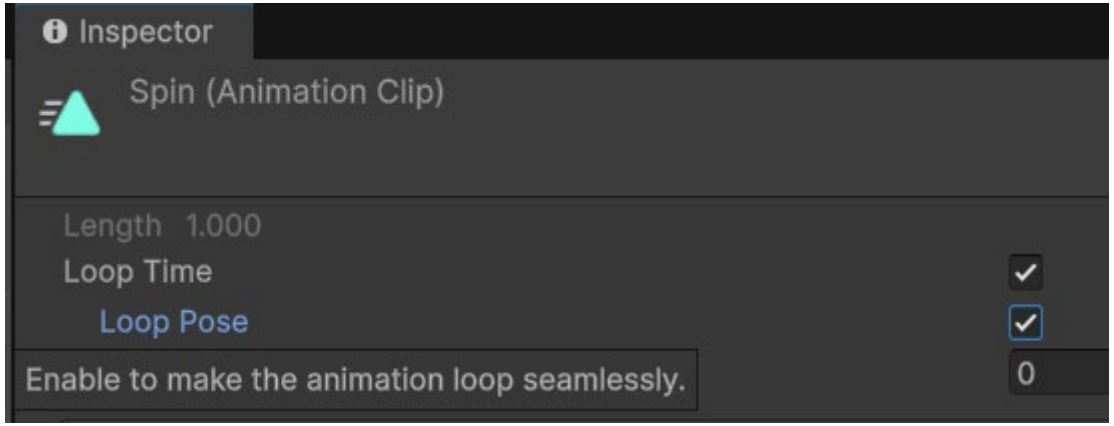
Problem: How can I make my Boulder animation loop?

Thankfully Looping is simple to setup. We can close the Animation Editor.

Now select the Animation Asset (Spin) in the Project window

In the Inspector panel on the right, we can see there is Loop Time and Loop Pose. Ensure both of these are ticked.

Our Boulder will now loop continuously through its animation cycle.



Making an Endless Runner Game in Unity

GDV4000 Introduction to Games Industry Practice

Unity Workshop 3

Based on research and notes by Dr Glenn L. Jenkins

Problem: I want the animation to change – Running must be different to Jumping

In our specifications we stated that we needed our player to be able to do:

Running

Jumping

Sliding.

We have ticked off Running (and a rotation animation on our Boulder), but we need to create Jumping and Sliding on the same Game Object.

This gives us a new sub-problem: **How do we get Unity to play and animate when running/jumping/sliding?**

We need to create animation cycles for Jumping and Sliding. Both utilise things we have already learned (creating a keyframe animation from a Sprite Sheet, and creating an animation based around manipulating the Object's Transform). The Jump and the Slide are the opposites of each other (Jump = Up, Slide = Down) so there will likely be a comparable logic to how we create them.

We do not want these to be Looped. They must be single-cycle animations. We will work in this order:

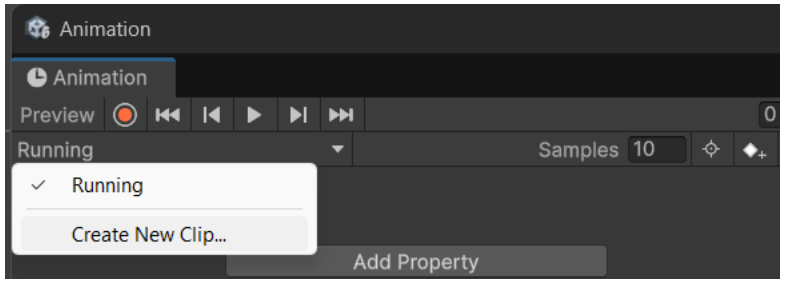
Create the Slide animation -> Create the Jump animation -> Look at how to connect them

Problem: I want the animation to change – Running must be different to Jumping

Adding a Sliding Animation

The process for adding further animations to a Game Object is a little different but still utilises the Animation Editor.

- Select the Player Ninja character
- Open the Animation Editor (Ctrl+6)



- Near the top you should see the name of the first animation cycle we created (Running) – click the drop-down arrow to the right.
- In the pop-up box, select Create New Clip.
- Name it Slide.anim and make sure you are saving it in the Player folder (you should see your Running.anim in there too).

For the Slide animation there is only one keyframe (the last frame of the animation). We will add movement to this.

The steps we will follow are:

- Drag the sliding sprite into the Animation Editor timeline.
- Change the number of samples to 4.
- Make sure the Frame Marker is at the start of the timeline and click the Add Property button. As we did before with Rotation, click the + icon next to Position. We will be manipulating the Y axis.

Problem: I want the animation to change – Running must be different to Jumping

Adding a Sliding Animation

- Move the marker to frame 0.2.
- With Position.y selected, click the Add Keyframe button (white diamond with + to the right). This will create our keyframes at two seconds in.
- Change the value of Position.y from -2 to -3.
- We should see our Ninja move down the screen.

- Note – Don’t be afraid to try out different values here. If you feel that the animation plays too quickly, drag the keyframes from 0.2 to 0.5 or 1.0, and then adjust the Sample value until it feels right.

Problem: I want the animation to change – Running must be different to Jumping

Adding a Jumping Animation

Here we are essentially doing the same thing as with our Slide animation, only we need more frames of animation, and the movement is up on the Y axis rather than down.

- Add a new animation to our Ninja and call it Jump (remember to check you are saving in the right place).
- Click and select frames 4-9 from the Sprite Sheet and add them to the Animation Editor.
- Set the Samples to 10.
- Add Property –> Transform-> Position
- We are going to set a position on the Y axis for each keyframe.
- Here we have hardcoded the values, but improvement could be made using Curves.
- Obviously, we have no forward motion as we will be using a scrolling background.
- Before we move on – Check that only the Running animation has Loop Time ticked.

Time	Y-Pos
0.0	-2
0.1	0
0.2	1
0.3	1
0.4	-1
0.5	-2

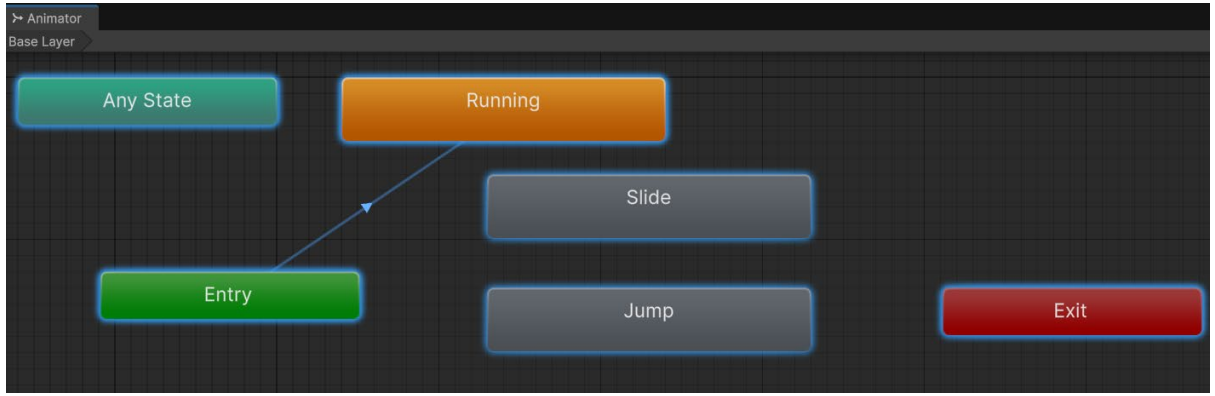
Problem: I want the animation to change – Running must be different to Jumping

How do we change between our animations?

This is something that modern Game Engines make easy for us. Unity uses a State Machine in its Animator, and Unreal Engine has a very feature rich animation system complete with rigging and blend spaces.

When animations are added to the same Game Object (our Ninja for example), an animation component is added. These become states that the Game Object animations can be in.

- Select the Ninja Game Object.
- Open the Animator Window (Window-> Animation-> Animator)



You should see a window like this with a Start and End State, an Entry and the animations we have made so far. The set-up comprises of three steps:

- Setup the transitions between states i.e. define what state changes are allowed.
- Add parameters to the animator.
- Associate these parameters with state changes.

Problem: I want the animation to change – Running must be different to Jumping

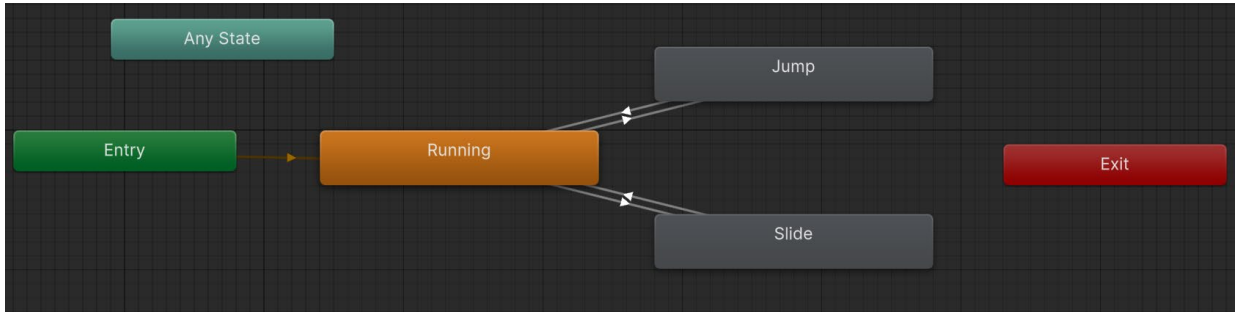
How do we change between our animations?

Adding Transitions

We need to show Unity which animations are possible from which states (**Run** -> **Jump** is okay, **Jump** -> **Slide** is not okay), and to do this we use Transitions.

- To create a Transition, right-click on the animation state (Running) and select **Make Transition.**
- Drag the Transition pin to the desired states (Running -> Jump, Running -> Slide, Jump -> Running, Slide -> Running).

You should end up with something like the image below (I shuffled the states around to make it more readable).

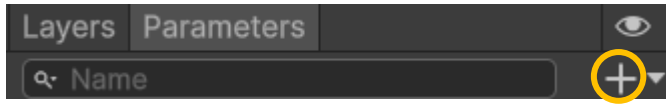


Problem: I want the animation to change – Running must be different to Jumping

How do we change between our animations?
Adding Parameters

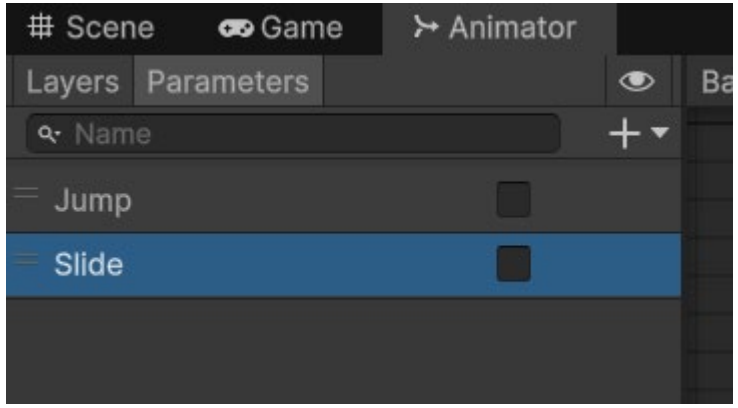
We need to add parameters to inform Unity about the events that have occurred. They are then passed to the current state and may result in a transition if the value of the parameter matches that which will result in a state transition.

To the left of the Animator should be an empty Parameters window. We want to add a new parameter in here.



- Add a new parameter by clicking the + to the right of the search bar.
- Set the type to **Bool** (Boolean) and the name to Jump.
- Repeat this for Slide and make sure both are set to false (unchecked) by default, otherwise the Player could start the game jumping and sliding.

You should end up with this.

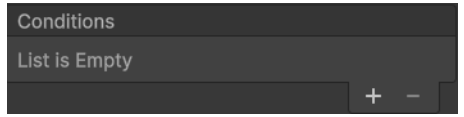


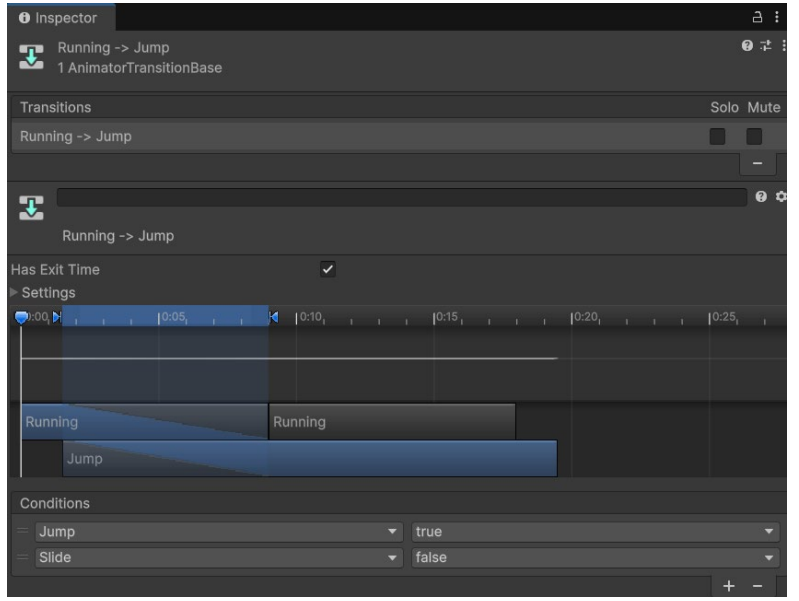
Problem: I want the animation to change – Running must be different to Jumping

How do we change between our animations?

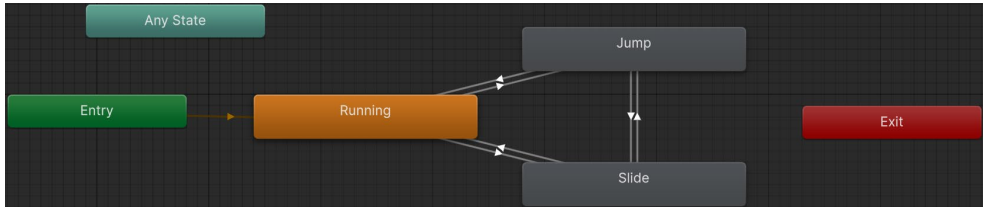
Linking Parameters to Transitions

Go back to the Animation window and select one of the Transitions (the arrows between the states). In the Inspector window we should see our Transition represented as a timeline. Underneath we have a place to add Conditions. By default, this list will be empty. The Conditions are the parameter Booleans we just created.





- Add a Condition by clicking the + beneath 'List is Empty'.
- We need to ensure that the Boolean logic for our Conditions makes sense.
- Use the table below to help populate the Conditions.



Source	Target	Jump	Slide
Run	Jump	True	False
Jump	Run	False	False
Run	Slide	False	True
Slide	Run	False	False
Slide	Jump	False	False
Jump	Slide	False	False

Problem: I want the animation to change – Running must be different to Jumping

How do we make this work??

We have set up a lot of infrastructure, but there is still nothing telling Unity that when we press W on our keyboard the player should perform the Jump animation. This is where we need a script.

As we are setting up a demo to get to know some basic operations, there is a PlayerMovement.cs script provided on Moodle.

- Create a new folder called Scripts and drag in the PlayerMovement.cs.
- Then, with your Ninja character open, drag the script onto the Inspector window on the right, just below Add Component (we are adding the script as a component).
- IMPORTANT – This script was written for a much earlier version of Unity and, by default, will not work with Unity 6's newer Input System. We need to make a few tweaks behind the scenes!
- Go to Edit -> Project Settings. Then in the new window under Player (on the left-hand column) find Active Input Handling and change it to 'Both'.
- Unity will need to restart. Make sure you save your project when prompted to do so.
- When we press Play, we should see our Ninja jump on W and slide on S.
- Note – The PlayerMovement.cs could be updated to the new input system and may well be later on if the old one is depreciated; but as the project only has two movements, the new system was considered a little overkill! For your own projects, I would advise looking into the newer system. For now, the old system is fine for prototyping.
- https://docs.unity3d.com/Packages/com.unity.inputsystem@1.18/manual/index.html

Making an Endless Runner Game in Unity

GDV4000 Introduction to Games Industry Practice

Unity Workshop 4

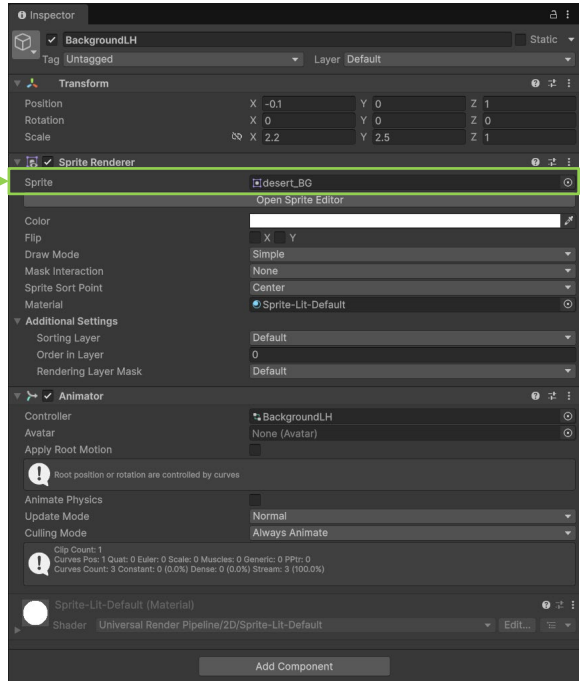
Based on research and notes by Dr Glenn L. Jenkins

Problem: I want to have an animated background in my Endless Runner game

Making the world move.

As part of the sprite assets we imported, we have a desert_BG.png. This is the sprite for our background. Nothing we are going to be doing in this section is essentially new, so hopefully as you go through this, there will be a lot of familiar practices.

- Create an empty sprite and call it something like 'BackgroundLH'.
- Transform = X -0.1, Y 0, Z 1.0
- Scale = X 2.2, Y 2.5, Z 1.0
- With BackgroundLH selected, drag desert_BG.png into the selection box to the right of **Sprite.**



Note that we are utilising the Z coordinate position on the **Transform** in order to layer our sprites.

Our Ninja sprite's Z Axis should be 0.0, placing it in front of the background.

We have a background in place, but it is static.

Our next problem will be how to make it move. This could be achieved through code, but as we have already successfully made use of the Animator and Animation Curves, the simpler option would be to use those.

Problem: How do I move the background image?

Making the world move.

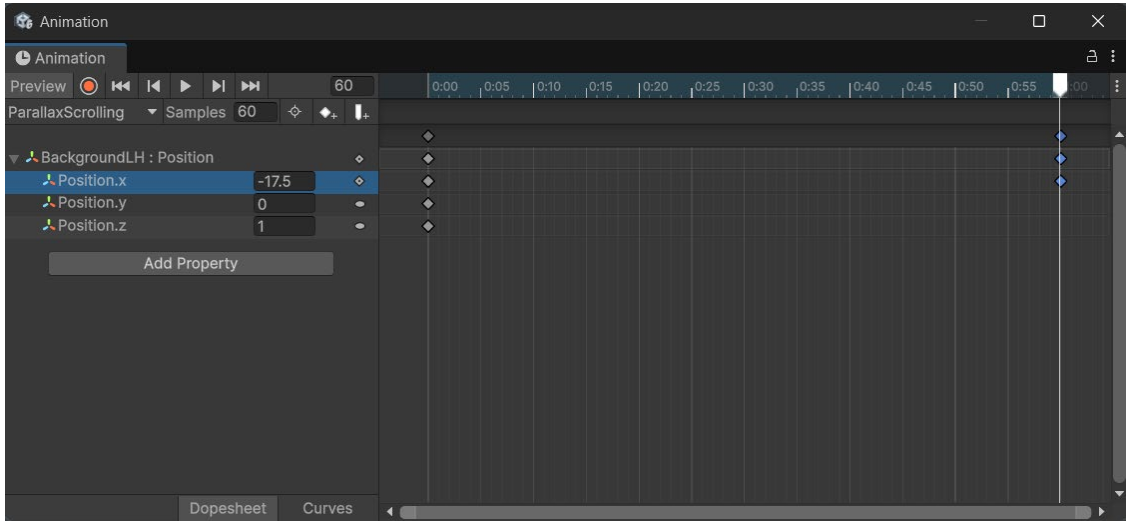
Start by selecting the component that we want to animate (**BackgroundLH**).

- Open the Animation Editor (CTRL+6) and create a new animation called ‘Parallax Scrolling’.
- Add a property. As we are looking to move the background to the left, we need this to be a Transform->Position.
- Samples and keyframes can remain at default.
- Move the keyframe marker to 1.00 and create a new keyframe on Position.x.
- Set the Position.x value at 1.00 to -17.5

Check the animation works. Make sure your X position is negative, otherwise the background will move the wrong way.

Close the animator and save the project.

When you play the game, you will notice that things are still not right…



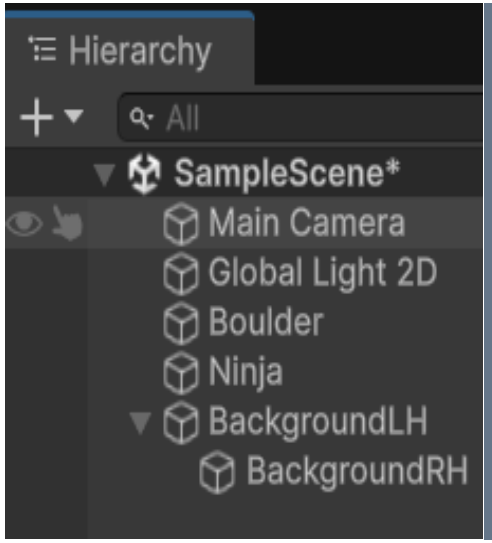
Problem: How do I make the background seem infinite?

Making the world move.

We need to create an illusion of an endless scrolling background, and we can leverage an important feature of our desert_BG texture – it is the same at both edges. This was designed to be a scrolling background.

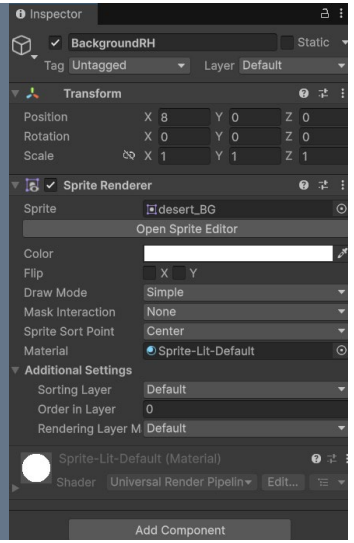
We are going to create another Game Object Sprite and call it BackgroundRH and then *parent* it to the Background LH. Parenting creates a link and a dependency between two objects. By parenting, when we move BackgroundLH, BackgroundRH will also move. This is because its position is local to that of its parent.

Note – This is not the same kind of parent/child relationship that you may have come across in other modules when discussing Object Orientation.



To parent, drag BackgroundRH in the Hierarchy window onto BackgroundLH. The result should be that the child is indented beneath the parent.

Copy these settings for BackgroundRH. Position.x should be 8.0 (moving it to join onto the righthand side of BackgroundLH). Note that the Scale Values are set to 1.0. This is because it is automatically taking the Scale used by the parent.



Problem: How do I make the background seem infinite?

Making the world move.

Save your scene and play the game.

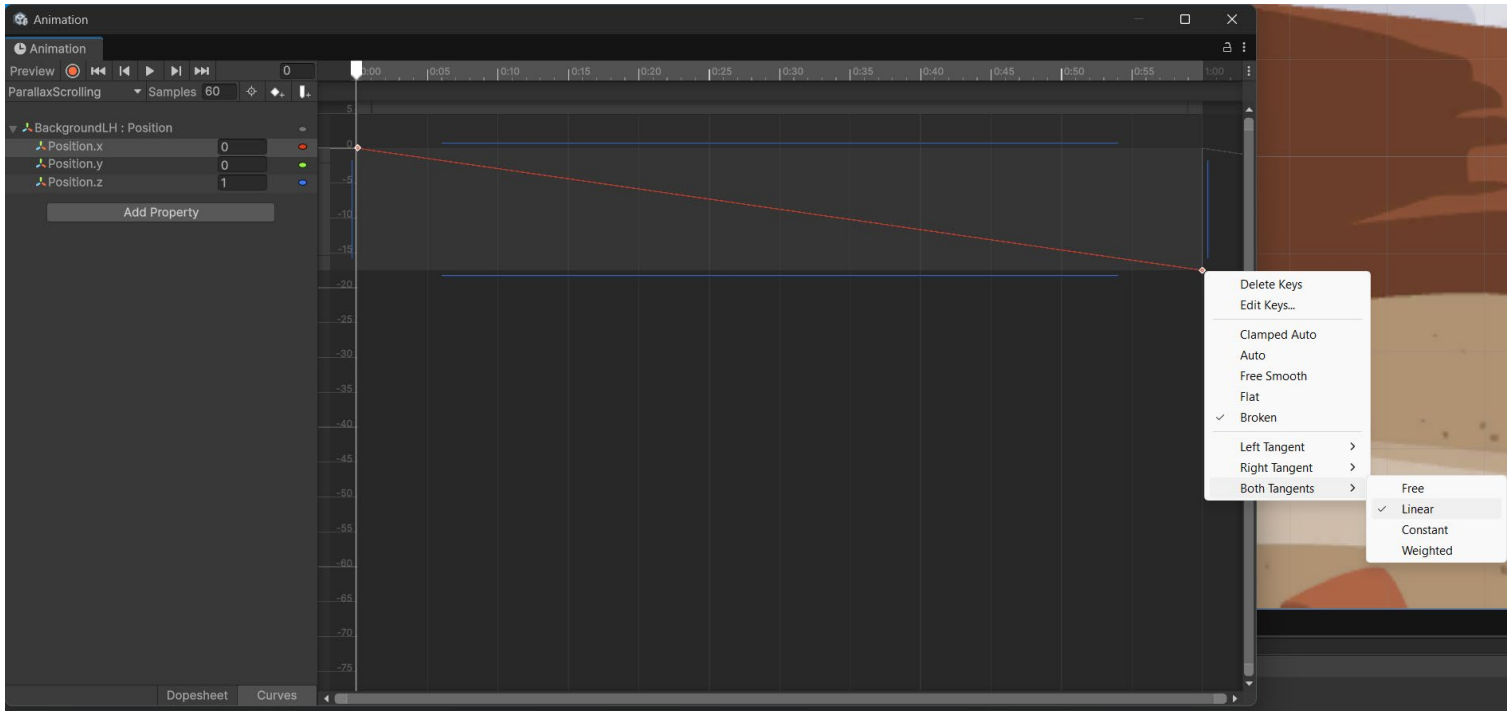
We should see a seamless scrolling background but… our animation curve is too smooth when it ought to be linear.

As we did with the Boulder, we just need to adjust the control points.

Select both control points in the Animation Editor under the Curve tab, and right-click.

Select Both-Tangents->Linear from the pop-up box.

This should straighten the line between the two points.



Try the game again.

Hopefully we have a smooth scrolling background and a Ninja that runs, jumps, and slides.

But to make it a game, we need to have some sort of challenge…

Extra Challenge

Now that you have had experience of creating a scrolling background, and you should be well-practiced in creating Sprites, Animations and using the Animation Editor, why not try to create some illusion of depth by introducing a scrolling foreground too?

Making an Endless Runner Game in Unity

GDV4000 Introduction to Games Industry Practice

Unity Workshop 5

Based on research and notes by Dr Glenn L. Jenkins

Problem: How do I make an obstacle that moves towards the player?

The Return of the Boulder

We have reached the stage where we need to draw together the components added to our Game Object to enable it to be created move towards the player.

- Boulder has a Transform and a Sprite Renderer attached to it (one provides position, orientation and scale, the other renders it to the screen). We also have our Animator, although that was created for us automatically when we created animations.
- We are going to add a Box Collider and a RigidBody2D. The Box Collider will provide a collision detection system. The RigidBody2D will link to Unity's physics system.
- The final component will be a script which ties the behaviour of the collision detection system to that of the physics system.

Preparatory work.

- Select the Boulder Game Object and in the Inspector select 'Add Component'.
- From the sub-menu, select Physics2D->Box Collider 2D.
- From the Inspector select 'Add Component'.
- From the sub-menu select Physics2D->Rigid Body 2D.
- Lastly, a C# Script – Move to or create a Scripts folder in your Project Window. Right-click, and from the Context Menu select Create-> Scripting->MonoBehaviour Script.

There are few choices of script in Unity 6. MonoBehaviour will give us some basic boilerplate code to start from.

- Name your script 'ObstacleMovement' and right-click it. Select Open C# project. This should launch Visual Studio.

Problem: How do I make an obstacle that moves towards the player?

Before we continue…

If the Boulder and our Ninja are to collide, then both must have a Collider component, as this provides information about the shape of the Object (the shape used for collision). When the Collider detects a collision, Unity calls a method in the class (if present) to handle the collision.

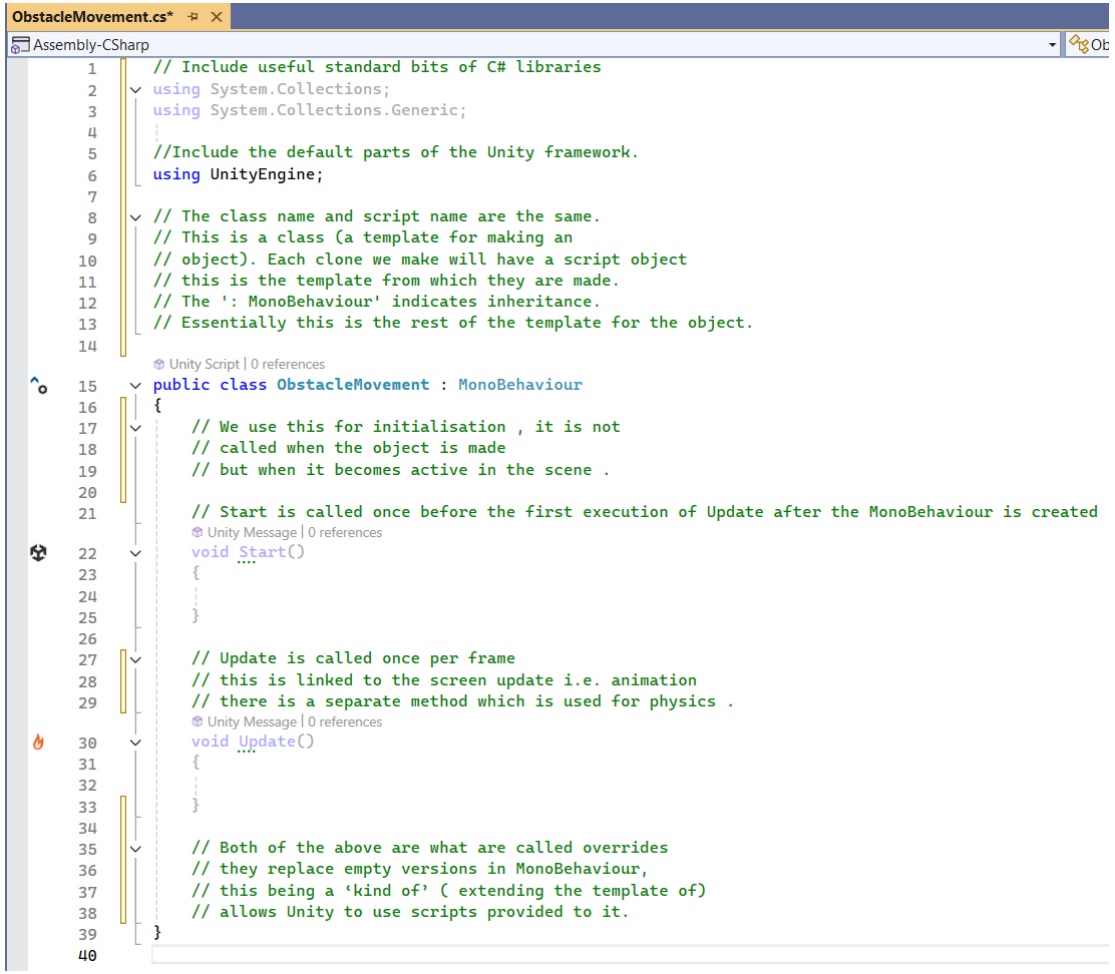
- Select the Ninja Game Object.
- In the Inspector, ‘Add Component’.
- From the sub-menu, Physics 2D->Box Collider 2D.

Problem: How do I make an obstacle that moves towards the player?

Sub-problem: How to create copies of the obstacle (Boulder)?

In Visual Studio (or IDE of your choice. The Games Lab has Visual Studio installed so we'll use that for consistency) we can start to populate our script with logic. There are a few things we need to add first.

- At the top of the script, above 'using UnityEngine;' add the following two lines:
- 'using System.Collections;'
- 'using System.Collections.Generic;'
- To the right is a version of our script with some comments (thank you Glenn).
- You may not understand the logic or terminology at this point, but with more practice it will become clearer.
- Note – You don't need to copy the comments in green, they are only here to give an overview of what's going on.



Problem: How do I make an obstacle that moves towards the player?

Sub-problem: How to create copies of the obstacle (Boulder)?

Ignore any red error lines for the moment (we will fix them with the next step because we have yet to define RigidBody2D and Obstacle Movement, or values for rockMinLimit and rockMaxLimit).

Here we have a method that will create an instance of the object. We will create public variables in our class for the rockMinLimit and rockMaxLimit so that they can be changed in the editor without having to go back to the code.

Finally, we will make use of Invoke, which calls a given method on all instances of a Game Object in a scene at a given interval.

```
// there is a separate method which is used for physics
Unity Message | 0 references
void Update()
{

}

// Both of the above are what are called overrides
// they replace empty versions in MonoBehaviour,
// this being a 'kind of' ( extending the template of)


// Spawns an Obstacle between rockMinLimit and rockMaxLimit
// with speed x velocity.
0 references
void SpawnObstacle()
{
    // Use a random number generator to give us a number within this range

    float height = Random.Range(rockMinLimit, rockMaxLimit);

    // Create a position from this as a Vector.

    Vector3 pos = new Vector3(10,  height, 0);

    // Create a rotation, this is made from Euler angles
    //Rotation around the x, y, and z. The mechanics of the
    //Quaternion are more complex - don't worry about it for now!

    Quaternion rot = Quaternion.Euler(new Vector3(0, 0, 0));

    // Create an instance of the Game Object using the original.
    // Use the position and rotation we have just created.
    Rigidbody2D obstacleInstance = Instantiate(obstacle, pos, rot);

    // Name the object
    obstacleInstance.name = "Obstacle(Clone)";
}
```

Problem: How do I make an obstacle that moves towards the player?

Sub-problem: How to create copies of the obstacle (Boulder)?

This will get a lot of Boulders spawning off-screen. The only way to check our code works is to run the game and monitor the number of components called ‘Obstacle(Clone)’ that spawn every few seconds.

Don’t forget to attach/drag our ObstacleMovement.cs script onto the Boulder Game Object.

```
Unity Script | 1 reference
public class ObstacleMovement : MonoBehaviour
{
    // We use this for initialisation , it is not
    // called when the object is made
    // but when it becomes active in the scene .

    // This is our reference to the RigidBody2D component of our object
    private Rigidbody2D obstacle;

    // These are public member variables in the class. They will be exposed in the editor to make
    // adjusting these values easier for tuning purposes.
    public float speed = -3.0f;
    public float rockMinLimit = -4f;
    public float rockMaxLimit = -1f;


    // Start is called once before the first execution of Update after the MonoBehaviour is created
    // Use this for initialisation.
    Unity Message | 0 references
    void Start()
    {
        // Get the RigidBody. Get Component looks for components attached to the same Game Object as the script.
        obstacle = GetComponent<Rigidbody2D>();

        // Invoke the method "SpawnObstacle" every four seconds.
        Invoke( methodName: nameof("SpawnObstacle", 4));
```

Problem: How do I make an obstacle that moves towards the player?

We have successfully addressed our sub-problem, now we can look at addressing the main problem.

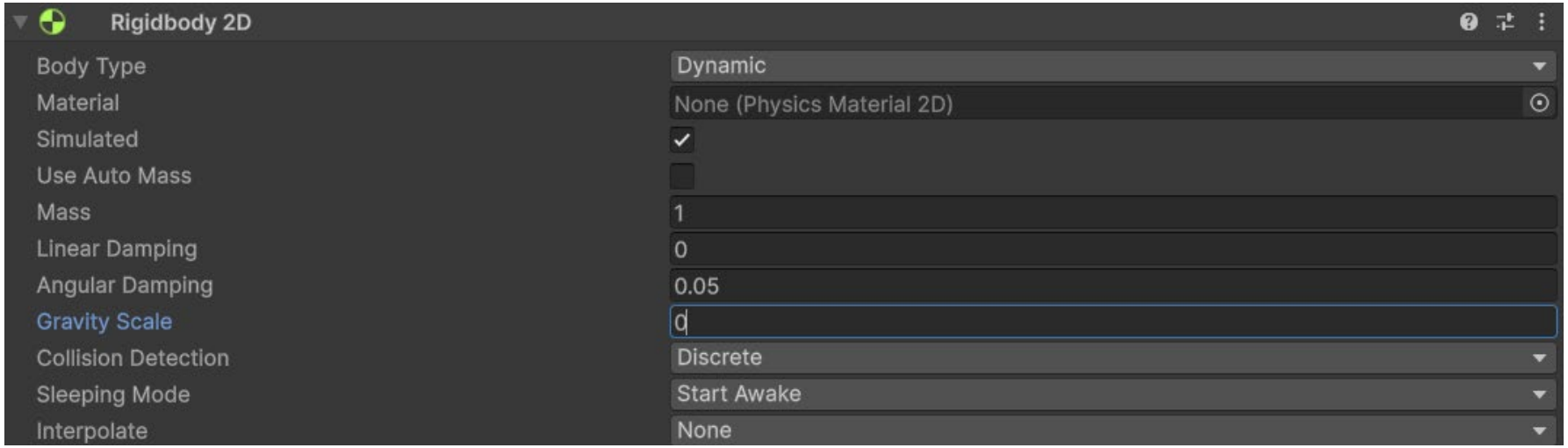
The RigidBody in our code refers to the position/orientation etc. of the object when interacting with Unity's physics system. To move the object towards the player, we just need to add a line of code to the bottom of SpawnObstacle.

- obstacleInstance.velocity = new Vector2(speed, 0);

One more change to make. By default, the RigidBody2D has gravity enabled and we need to disable it otherwise our Boulders will fall rather than fly across the screen. Select the Boulder Game Object and set the Gravity Scale to 0.

We can press play and see that it does exactly what we told it to do… but there are a few bugs to iron out.

Let it run and see what you notice happening.



Problem: How do I check for player hit?

A few slides back we added a Box Collision 2D component to our Boulder and Ninja.

We can use these in our code to create some logic around what happens when two objects collide. To start with, we want to confirm that the collision is happening, so we are going to use a Debug Message.

We have also used a few comments as pseudo-code to suggest other pieces of logic that we can implement here, such as checking that the intersecting collision belongs specifically to the player.

If you play the game again and open the Console tab at the bottom, every time there is a collision you should see “We have hit something!” appear.

Using Debug messages is a useful way of checking your code step by step, rather than trying to unravel a chunk of logic. When you know one step works, you can move to the next.

When you are more comfortable with coding, you can get away with doing it less.

```
            obstacleInstance.velocity = new Vector2(speed, 0);
        }

        // Collision Response
        Unity Message | 0 references
        private void OnCollisionEnter2D(Collision2D other)
        {
            //Debugging code - Comment this out for final version
            Debug.Log("We have hit something!");

            //Check this is the player

            // Do something if it is
        }
}
```

Problem: How do I end the game when the player is hit?

Here we are going to replace our comment with code to ‘do something’ when the player is hit – namely end the game. First, we need to check if the other component is the Ninja.

To do this we will ask the Collision2D object for the Game Object it is attached to and use GetComponent<PlayerMovement>(…) to find the attached Player Movement object if there is one. If there is, then we will end the game by stopping the timer running the animation. We will also destroy the Game Object for good measure.

The code will go inside our OnCollisionEnter2D(). We can comment out our Debug message too.

If we hit play, we can see that if the player is hit by a Boulder, then the game stops.

There are still refinements to make and bugs to fix, but this has provided a small game loop for you to play with and hopefully learn from.

```
// Collision Response
Unity Message | 0 references
private void OnCollisionEnter2D(Collision2D other)
{
    //Debugging code - Comment this out for final version
    Debug.Log("We have hit something!");

    //Check this is the player
    // If the object we collided with (other) has a PlayerMovement (our script) attached, then it's the Ninja
    // Get the GameOject the collider is attached to

    GameObject go = other.gameObject;

    // Call its GetComponent()
    PlayerMovement pm = go.GetComponent<PlayerMovement>();

    // If pm is null, we've hit another Boulder
    // If not then we have hit the player.
    if (pm != null)
    {
        // Stop time - End the game
        Time.timeScale = 0;

        // Destroy the root game object
        Destroy(gameObject);
    }
}
```

Further additions in your own time...

- Create a duplicate of the Boulder and change the sprite to something more likely to be at head height.
- Add the same script we used for the Boulder but alter the Min/Max values.
- Update the Min/Max values for the Boulder so it appears to be rolling rather than flying.

There are also optimisations that can be made to the design…

- Separate the code for movement and the code for collisions in the Boulder component into two separate scripts. Name them ObstacleMovement and ObstacleCollision.
- Create a PlayerCollision script and attach it to the Ninja, then copy across the code from ObstacleCollision. Change it to look for an ObstacleCollision rather than PlayerMovement during collision.
- Add a public variable called NumberOfHits to the player (set the value to 5) and change the collision response code to subtract one from player hits until it reaches zero, then end the game.

Bibliography

Pereira, V. (2017b). Learning Unity 2D Game Development by Example, chapter Setting the Scene, pages 27–44. PACKT Publishing, Birmingham.