



a.r.u. | Peterborough

Game Engine Workflows

Introduction to



SESSION 1 TASKS

TASK ONE: Adding keyboard controls to a game

Using both the guide below and experimentation, add the following controls to the side-scrolling game template provided.

Left Arrow = Move Left

Right Arrow = Move Right

Up Arrow or Space = Jump

TASK TWO: Explore the level. Can you find the item to collect? Can you add an action input?

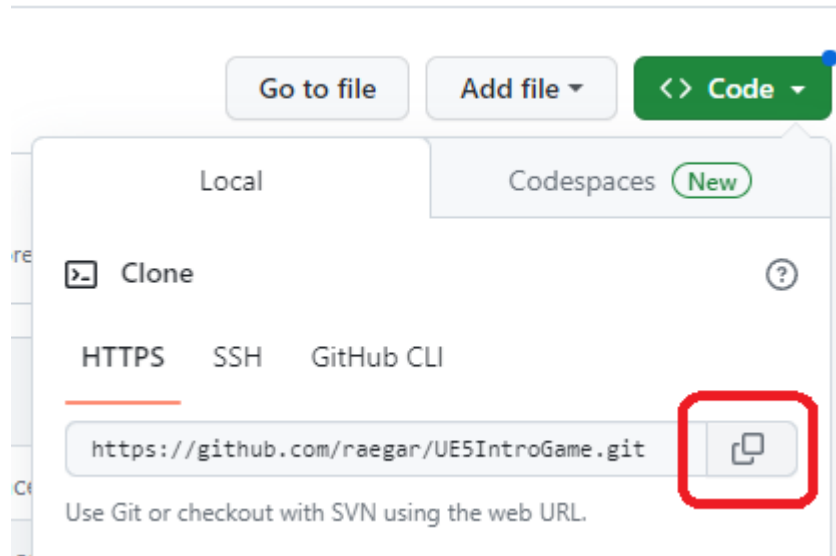
TASK THREE: Extension activities

TASK ONE:

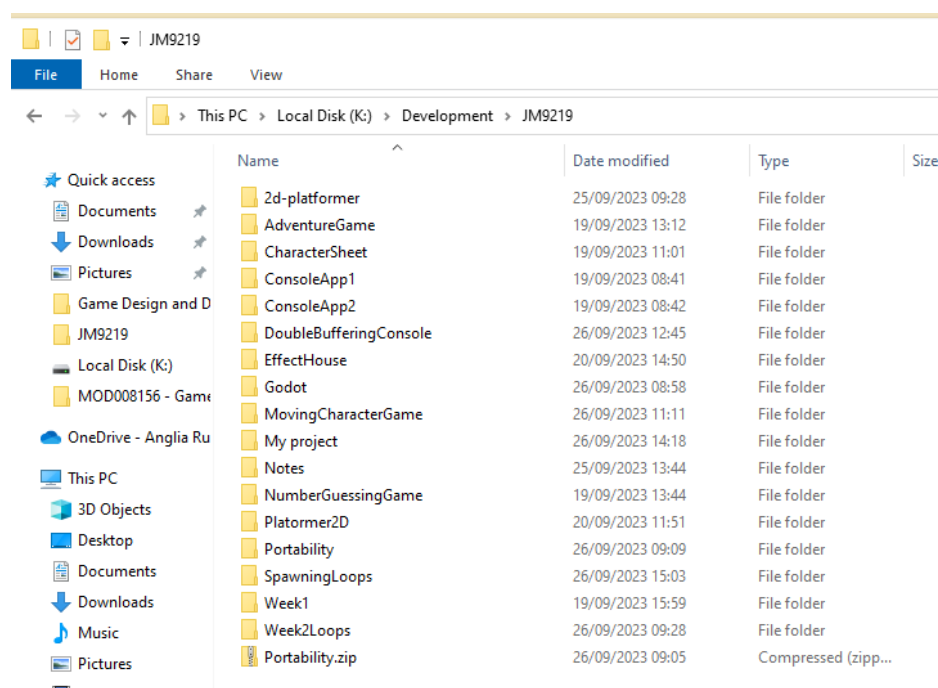
First we need to have a starting point for our project. Rather than starting from scratch, we will use a template to provide a simple level and character controller to begin with and then build it out from there.

Head to: <https://github.com/raegar/UE5IntroGame>

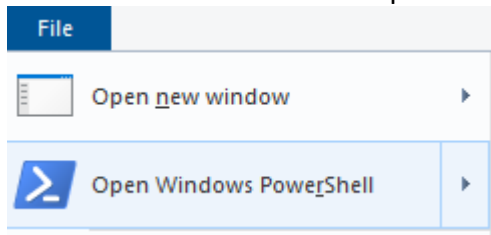
Click on the green [< > Code] button and then click on the copy link icon highlighted in red below.



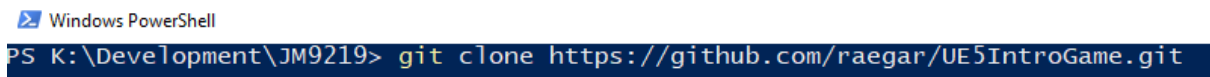
Now browse your computer to find the location where you want to save the game. In the lab this would be something like K:\YourUserName



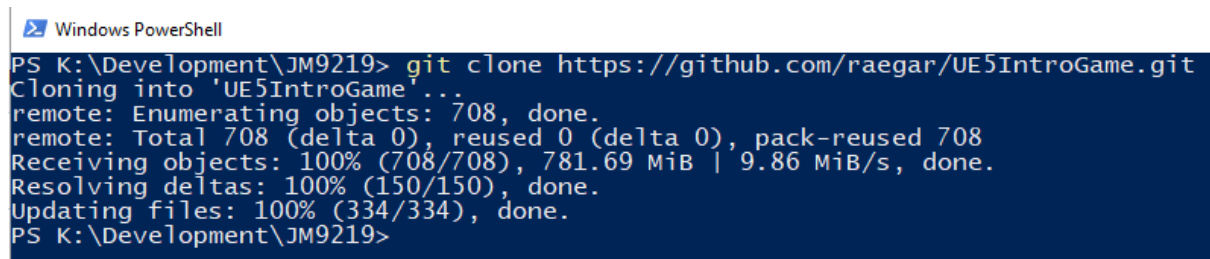
From the File menu choose Open **Windows PowerShell** or **Open Terminal**.



In the terminal type `git clone` then paste the url you copied. In terminal you can right-click to paste or use Ctrl-Shift-V



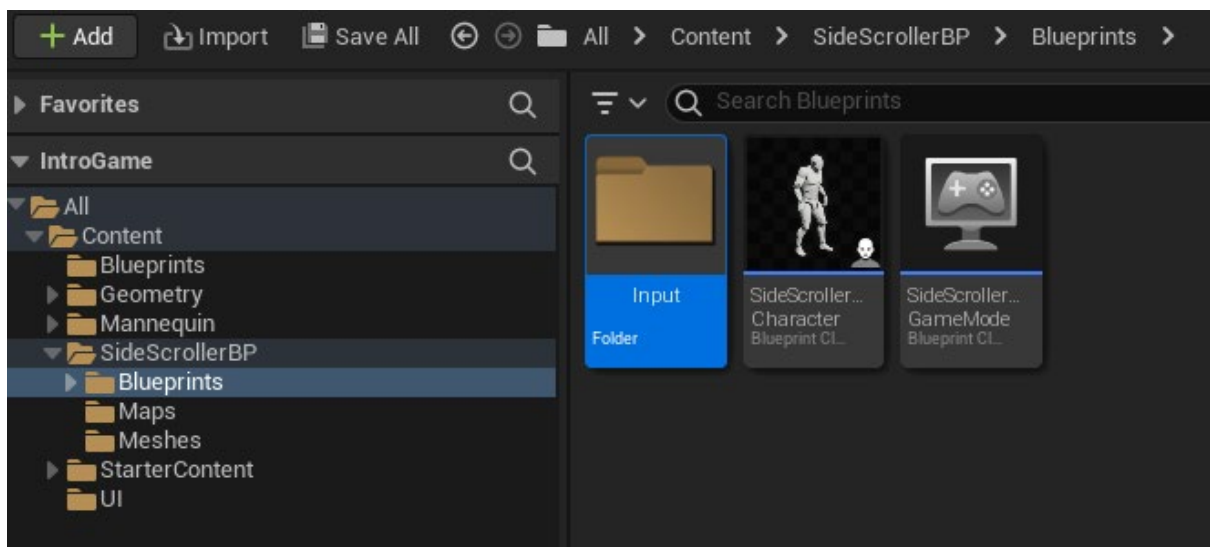
Press enter and the game will be cloned into a new folder.



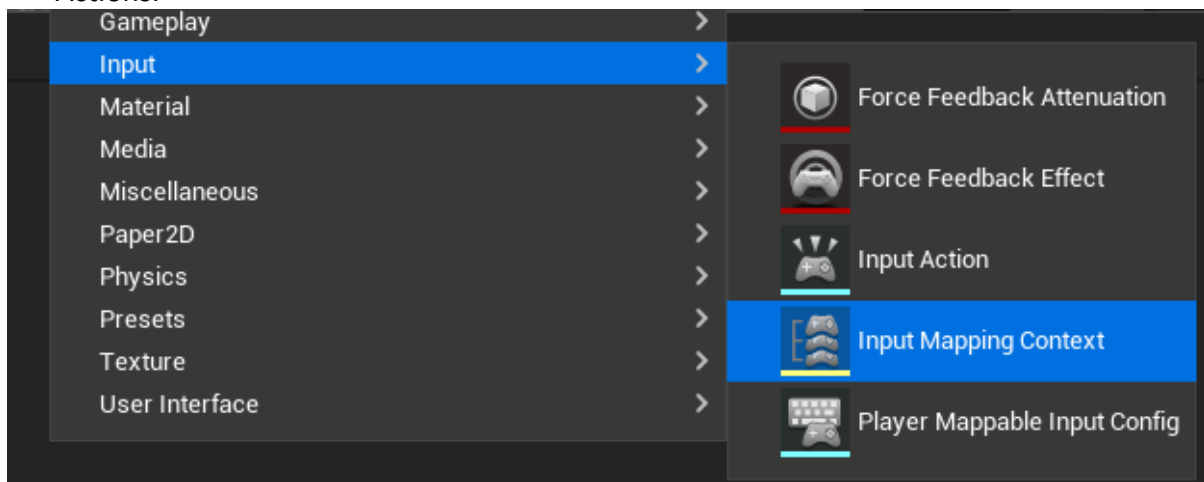
You can now open the IntroGame.uproject in Unreal Engine from **Recent Projects** then **Browse...**

Now, we will take a look at how we can add controls to our player using the new Enhanced Input Mapping system in UE5.

- 1) Navigate to **Content -> SideScrollerBP -> Blueprints** and create a folder named Input. This is where we will add our Input Mapping Context and Input Actions

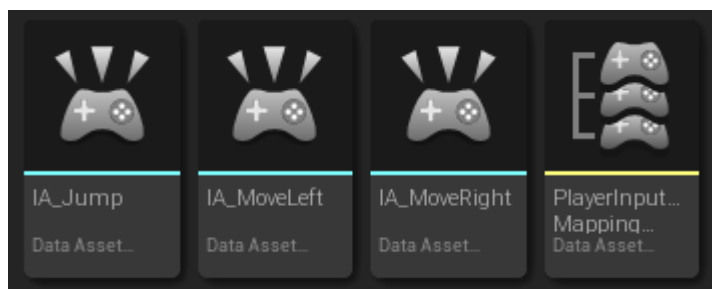


- 2) Inside this folder, right click and add one Input Mapping Context, and three Input Actions.

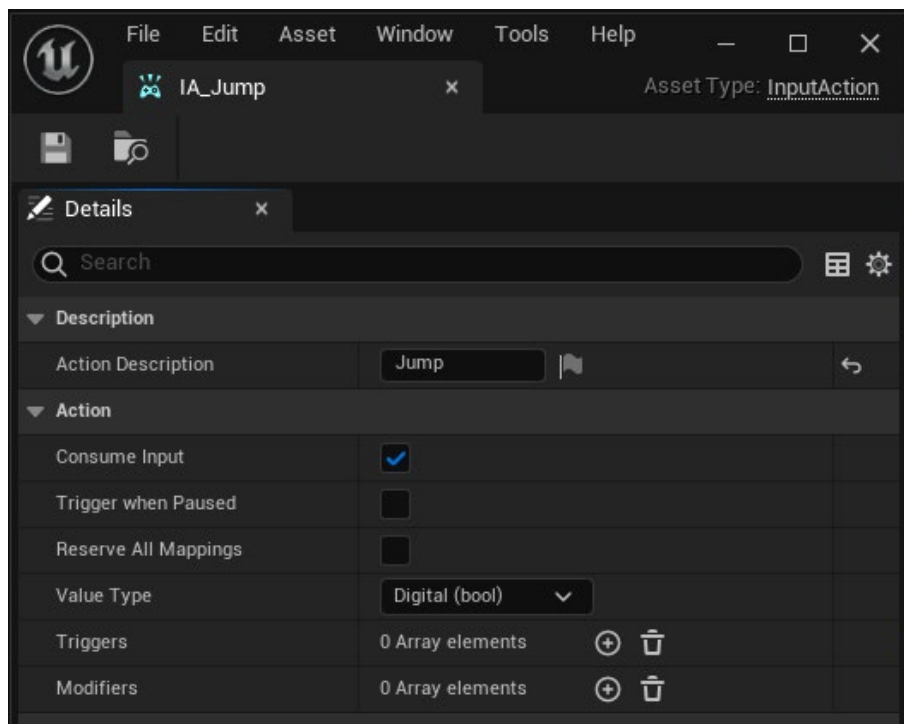


- 3) Name these

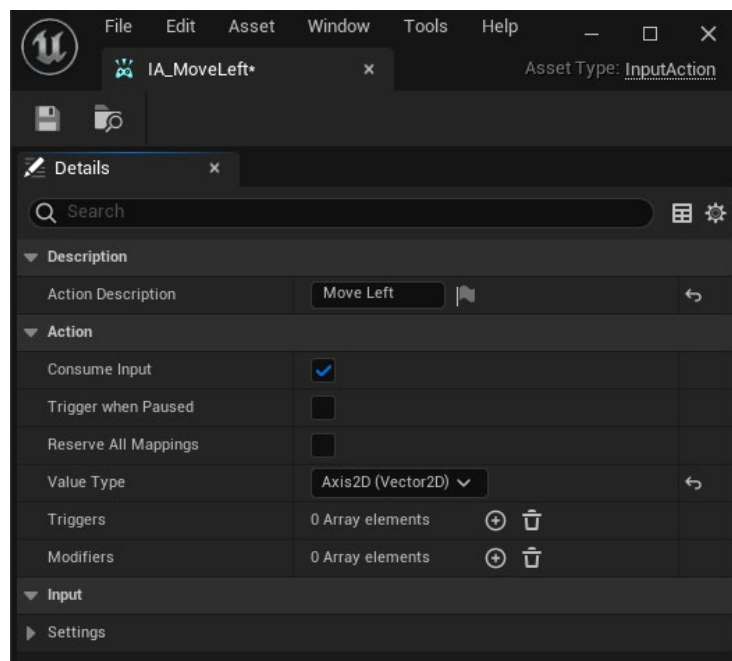
- PlayerInputMappingContext
- IA_Jump
- IA_MoveLeft
- IA_MoveRight



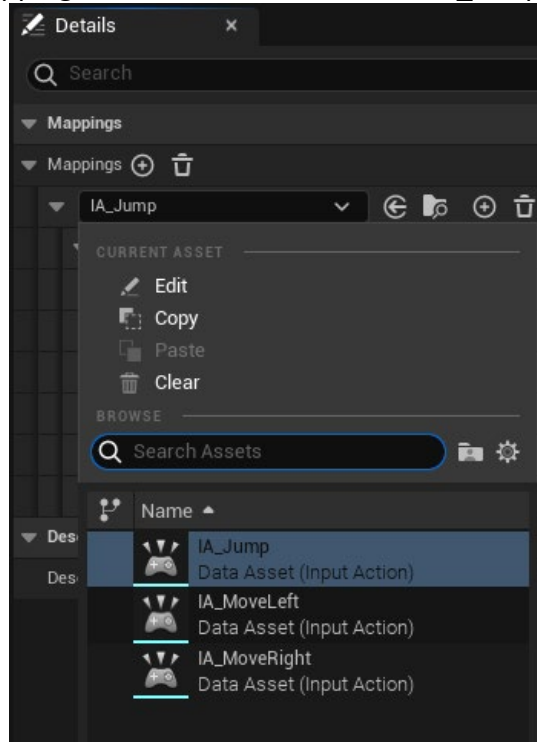
- 4) Set IA_Jump to have an appropriate Description and set the value type to Digital (bool) as it will essentially be a switch on or off, rather than an analogue input which can have a range of values.



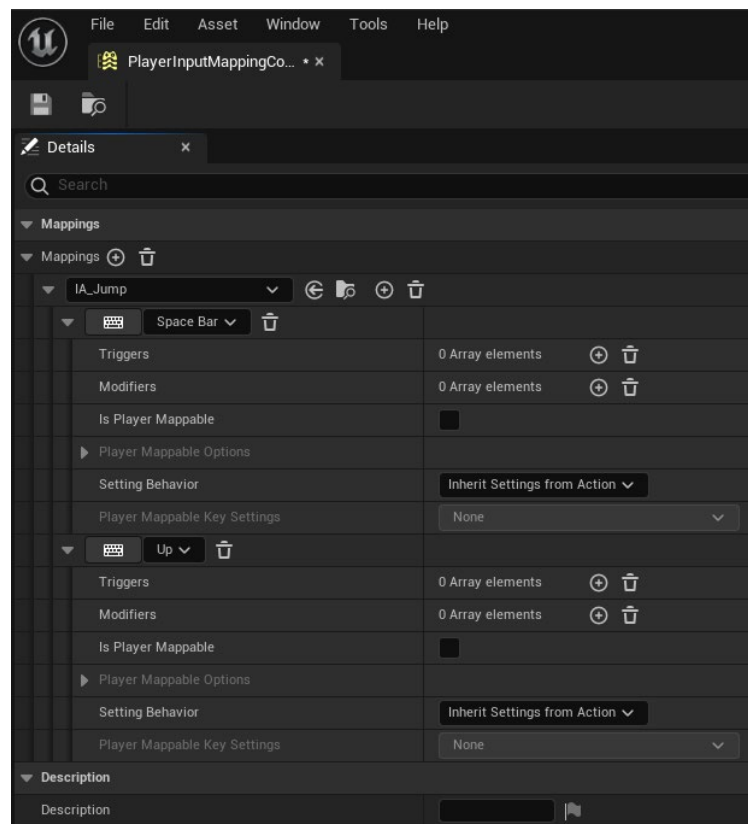
- 5) Do the same for **IA_MoveLeft** and **MoveRight**, but set their **ValueType** to **Axis2D** (Vector2D)



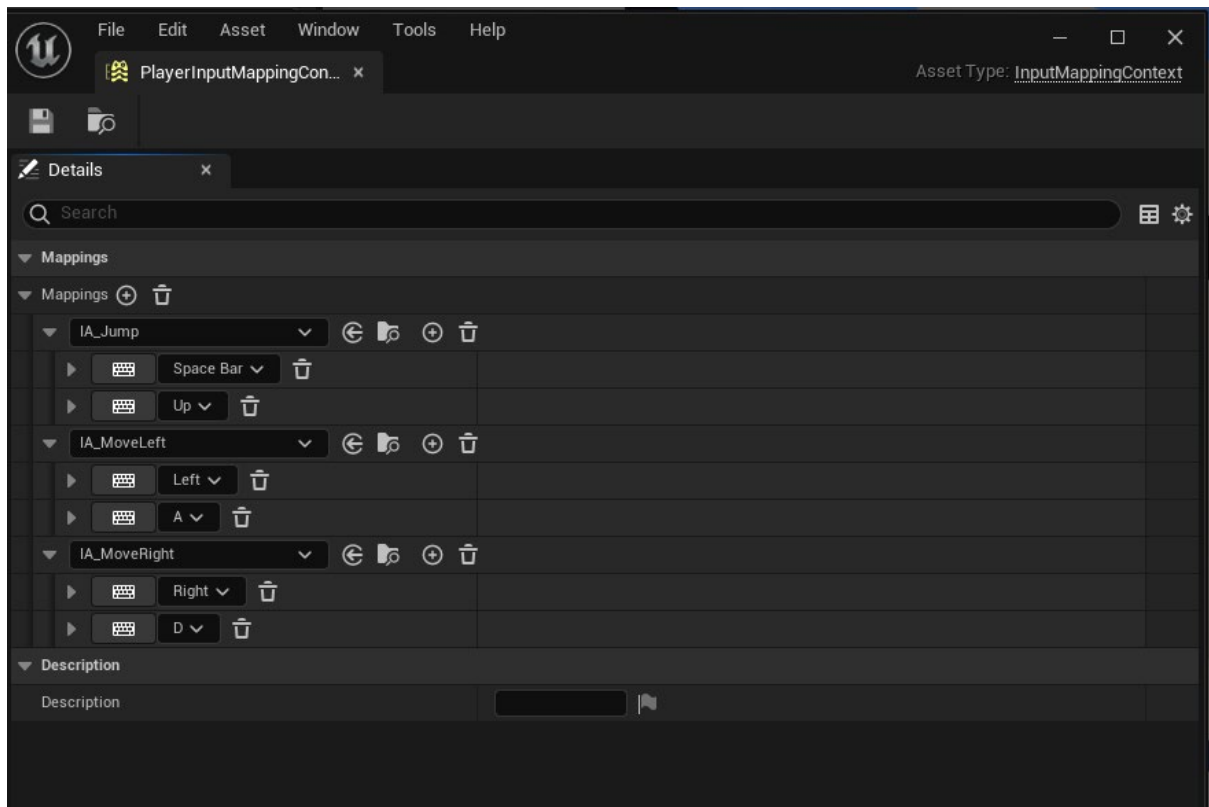
6) In the PlayerInputMappingContext add a reference to IA_Jump




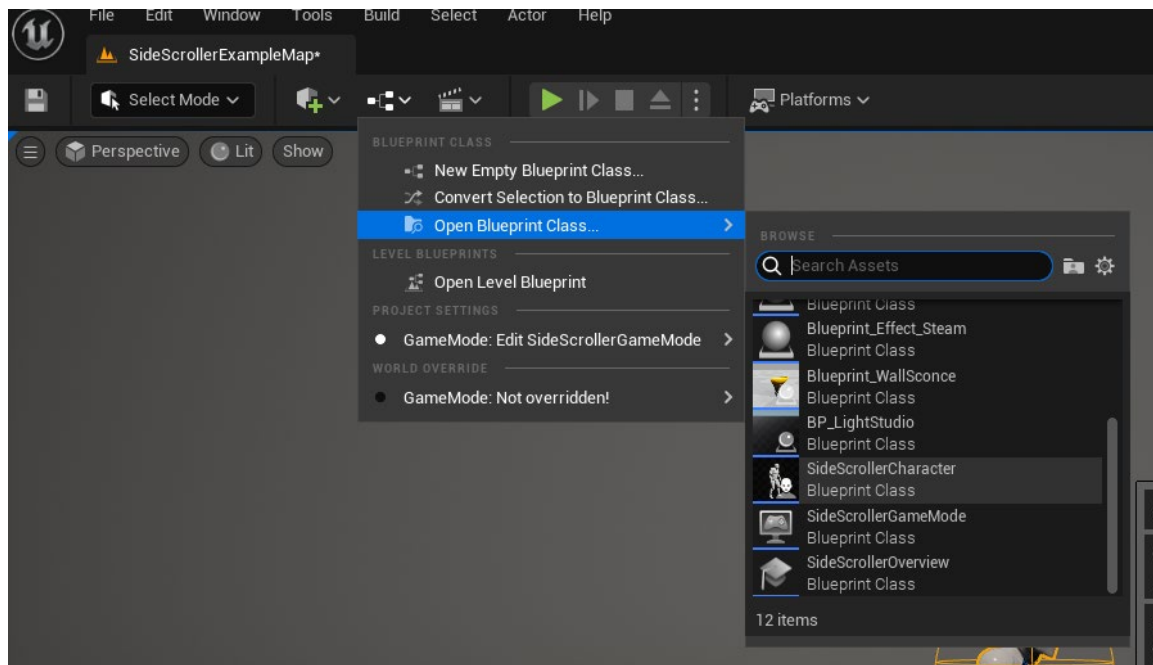
7) Then associate this action with the **Space Bar** and **Up** keys.



8) Now do the same thing for the Left and Right Input Actions, with appropriate key presses for each.



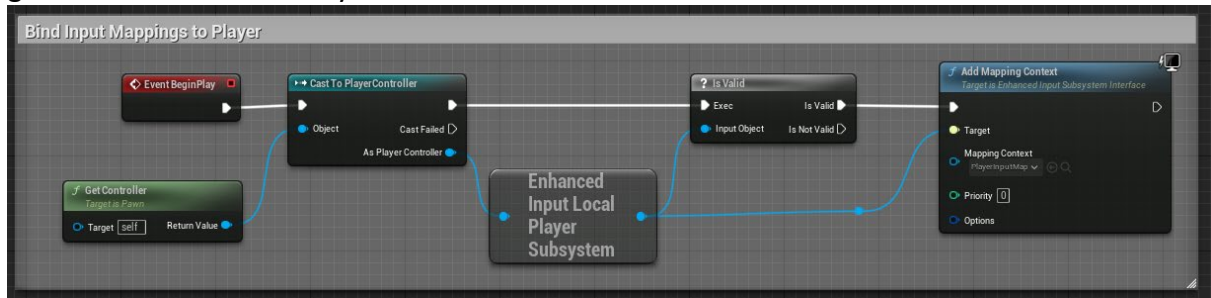
- 9) Now we have set up Input Actions, click on the Blueprints icon from the UE5 top taskbar, and choose  **Open Blueprint Class > SideScrollerCharacter**



A Blueprint is a visual scripting tool that enables you to program functionality quickly into your games without having to use code. However, Unreal Engine also includes the ability to fully code your games using the C++ language if you would like to have more control and the ability to optimise routines. Games made in the industry typically use a combination of both Blueprints and Code in the development process.

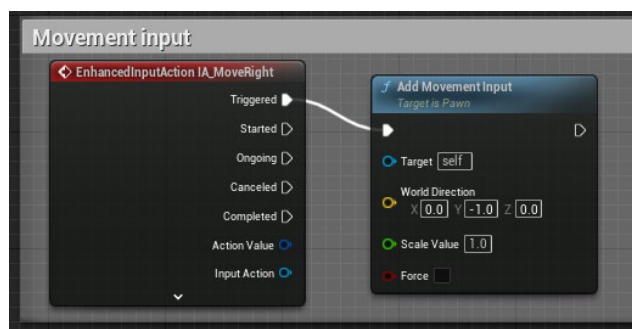
10) You will now see the Blueprint 'Event Graph' editor.

We need to bind the PlayerInputMappingContext to the Player so the input events can be assigned to actions in the Player.



11) To enable movement of our SideScrollerCharacter 'pawn' we need to tell UE5 what to do if the IA_MoveRight action is triggered (i.e. when the key bound to this action is pressed). Right-click on the blueprint and type in **IA_MoveRight**, then select the option that appears. This adds an action node to the screen. Now add an **Add Movement Input** node to the **Movement input** graph by right-clicking near to **IA_MoveRight** and searching for **Add Movement Input** in the menu that appears.

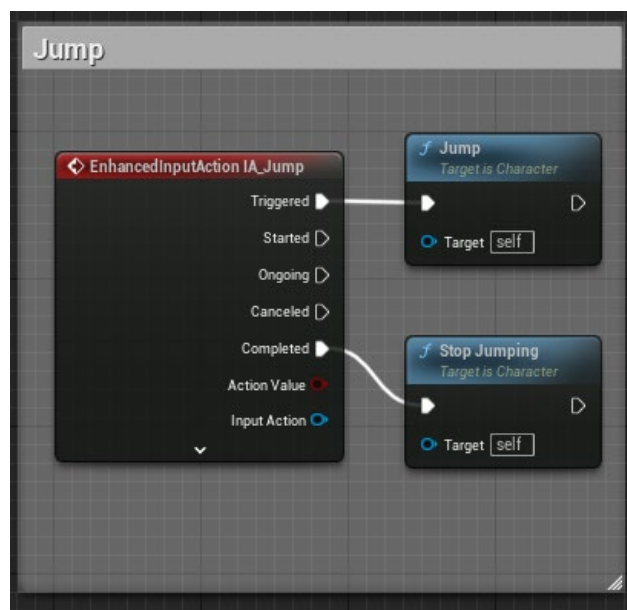
12) Finally, change the **World Direction** Y value from 0.0 to -1.0. This will tell the 'pawn', which is the SideScrollerCharacter to move in the 'negative Y-axis', which in our screen-space moves the player to the right.



13) Now do the same for **IA_MoveLeft**, with a Y movement of 1.0




- 14) Test the game by pressing the green Play button on the top taskbar. Make sure the gameplay preview window is active by clicking on the level editor near the player, and then press the Right key to test player movement. Once you are done, press the Escape key to regain mouse control.
- 15) Next, we need to do something similar for Jump, except we need to trigger two different actions, one for when we start jumping, and another when jumping ends. This allows the animation for jumping to start and stop at the right moments, which is crucial for making the jump look realistic, as it will have a beginning and end phase.



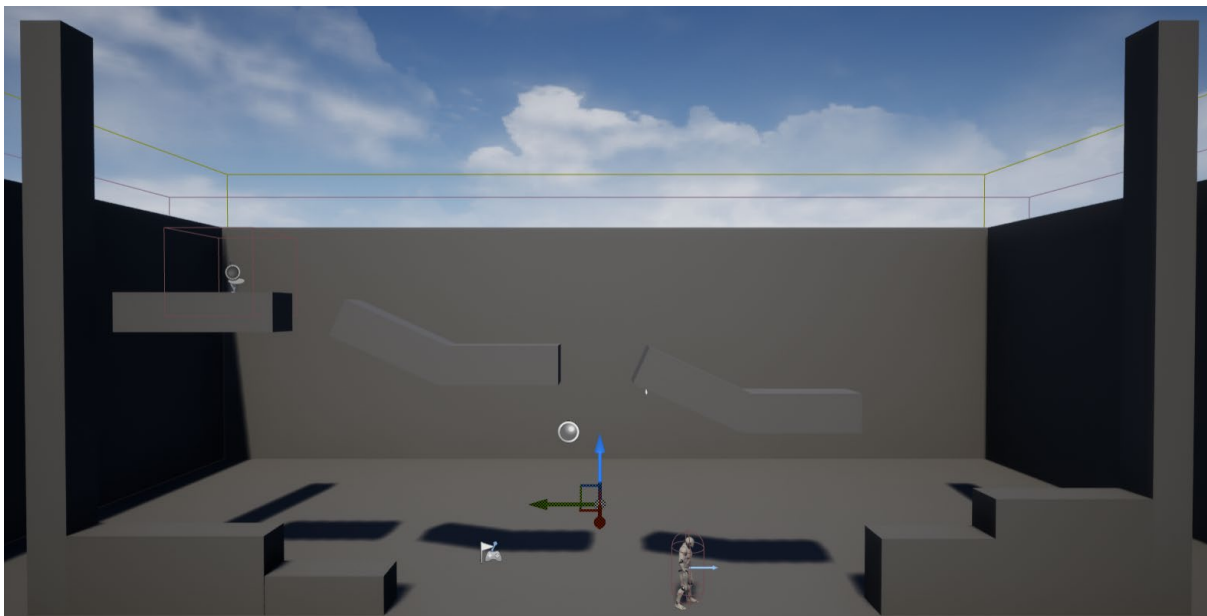
Now when you test your game you should be able to jump by pressing Space Bar or the Up arrow. Try exploring the level, and if you find the statue, press the **E key** to collect it; If you want to see where this is scripted, check out **Blueprint_InteractableItem** class.

Well done, you now have a simple working game!

1. Can you change the **Blueprint_InteractiveItem** to use an Input Action rather than a hard-coded keypress event?
2. Can you add to the existing Input Actions to support WA(S)D controls?
3. Can you change the text that is displayed on the screen when you collect the statue?
4. Can you add some more platforms to the level? You will need to add a cube from the **Quickly add to project** button  and selecting **Shapes > Cube**.

You will need to set the properties of the new cube to match the other platforms (Location, Material, etc).

Ensure that you add the new platform to the Ledges folder in the Outliner and name it to match the other items.



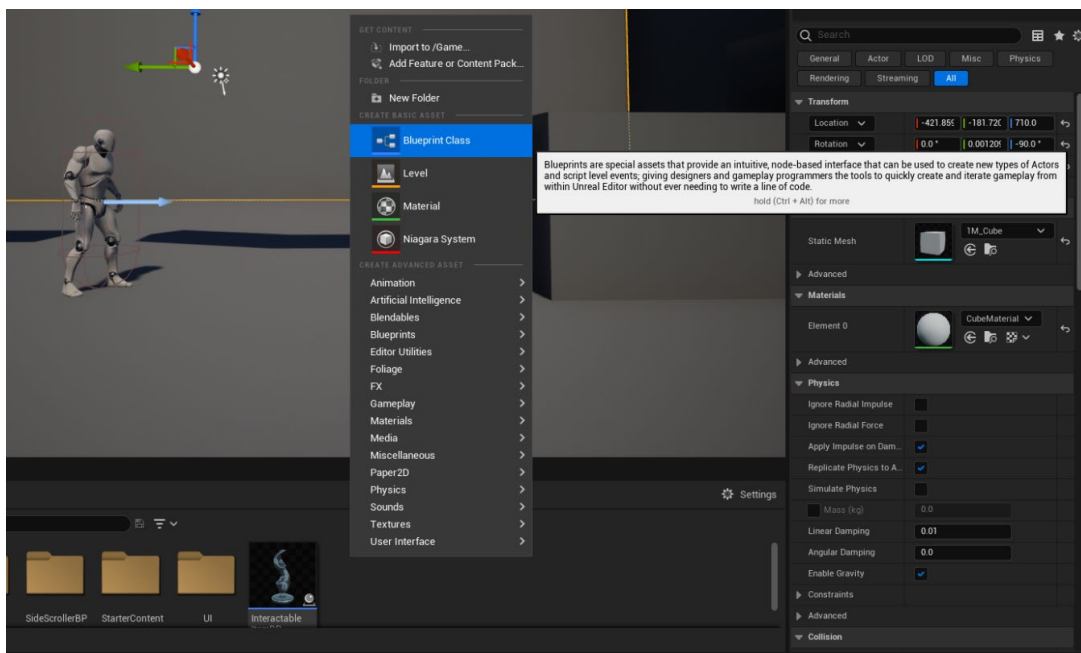
Part 2 TASKS

TASK ONE: Add a moving platform to the game.

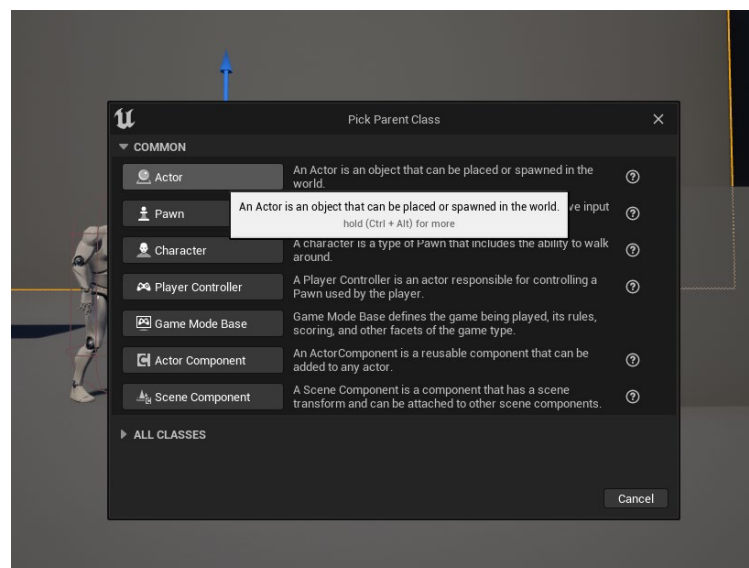
TASK TWO: Extend the game based on what you have learned.

TASK ONE:

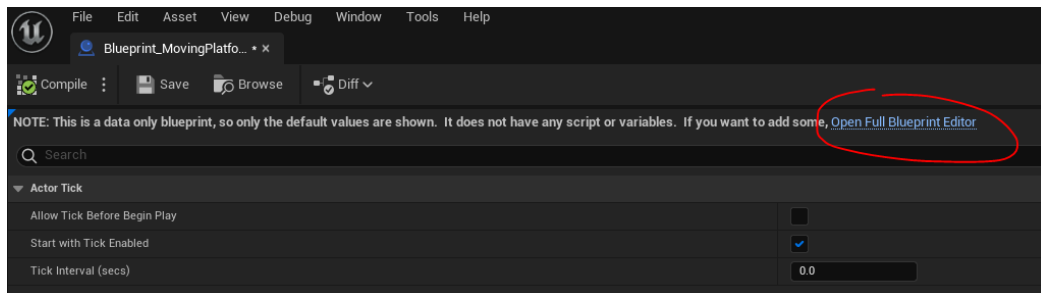
- 1) Right click on an empty space in the Content Drawer and choose Blueprint Class. Blueprint Classes are templates for creating new game objects. They define the object's properties and behaviours.



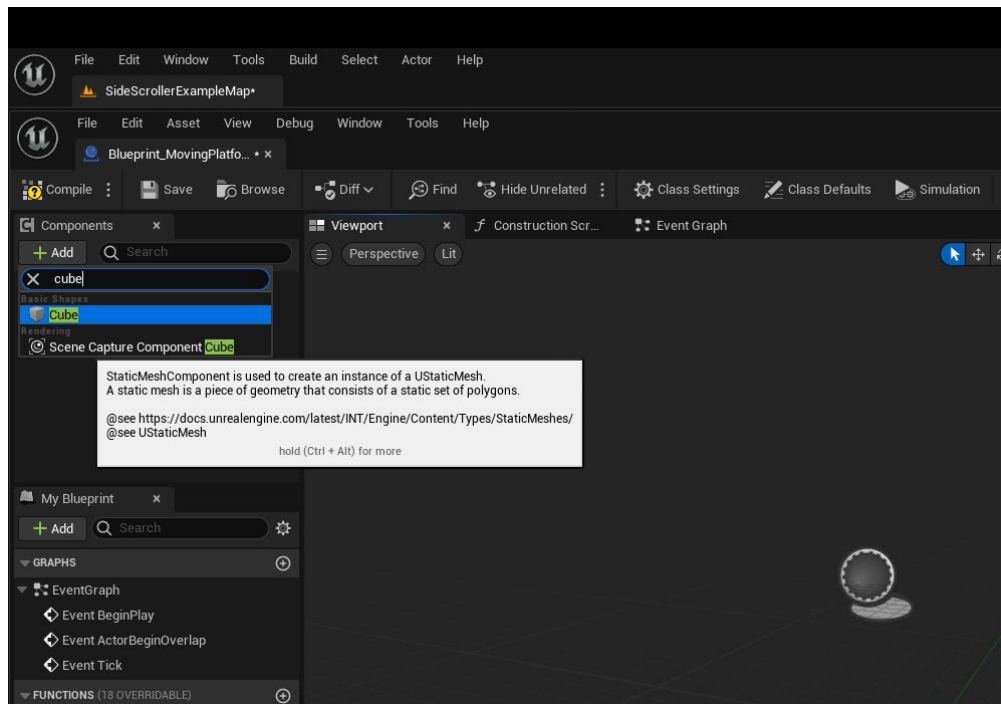
- 2) Set this new class to have a 'Parent Class' of the Actor type. This will provide us with some default values and settings for spawning an object into the game.



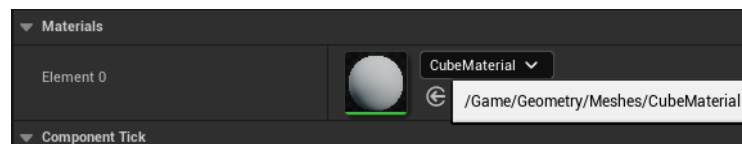
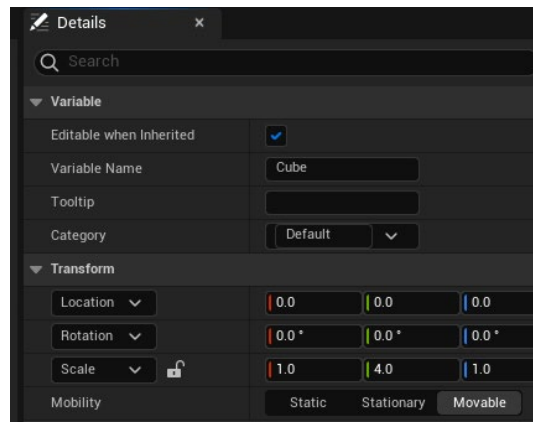
- 3) Call this new object **Blueprint_MovingPlatform** and then open it by double clicking on the object in the Content Drawer. If the following tab appears, click on the message circled below to open the Blueprint view.



- 4) Select the Viewport tab and then add a Cube component to the Component tree on the left of the Blueprint editor. The Cube component will serve as the visual representation of our moving platform.

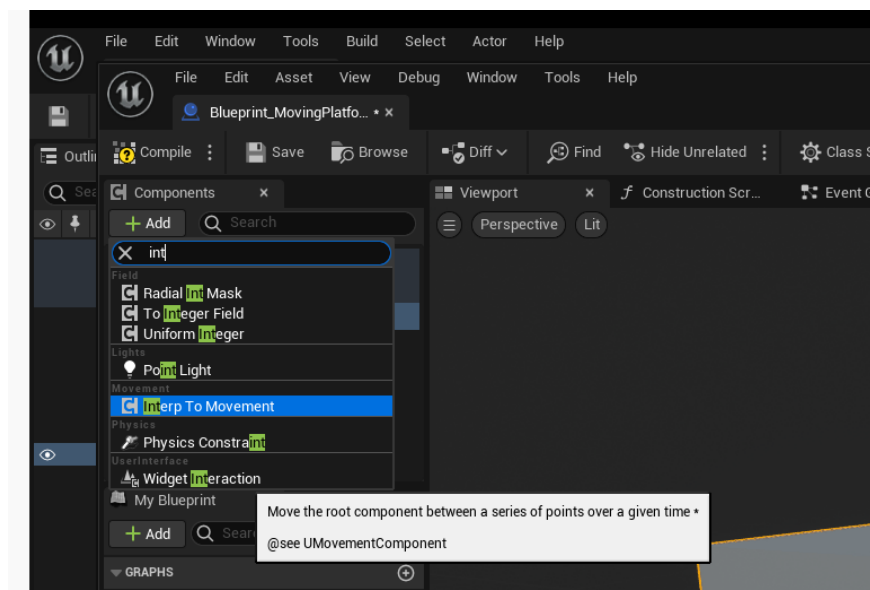


- 5) Change the Cube's Y scale to 4.0 and set the Material to CubeMaterial (You may have two CubeMaterials, if so, select the one with the path of Game/Geometry/Meshes/CubeMaterial)

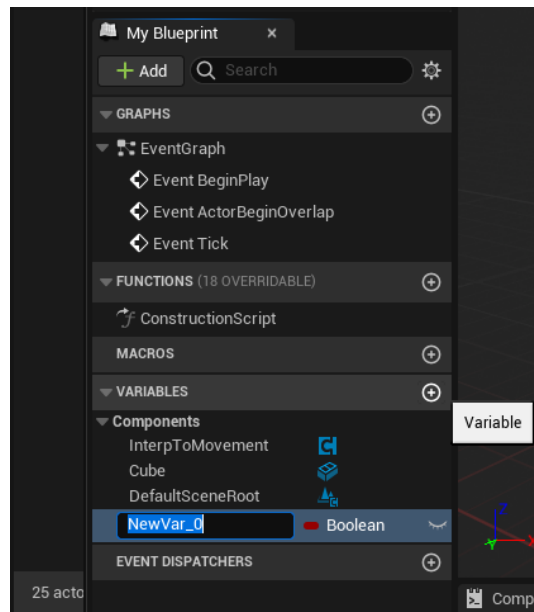


- 6) Add an **Interp To Movement** to the Components tree. This will enable us to move the object between two points by 'interpolating' the position between these points.

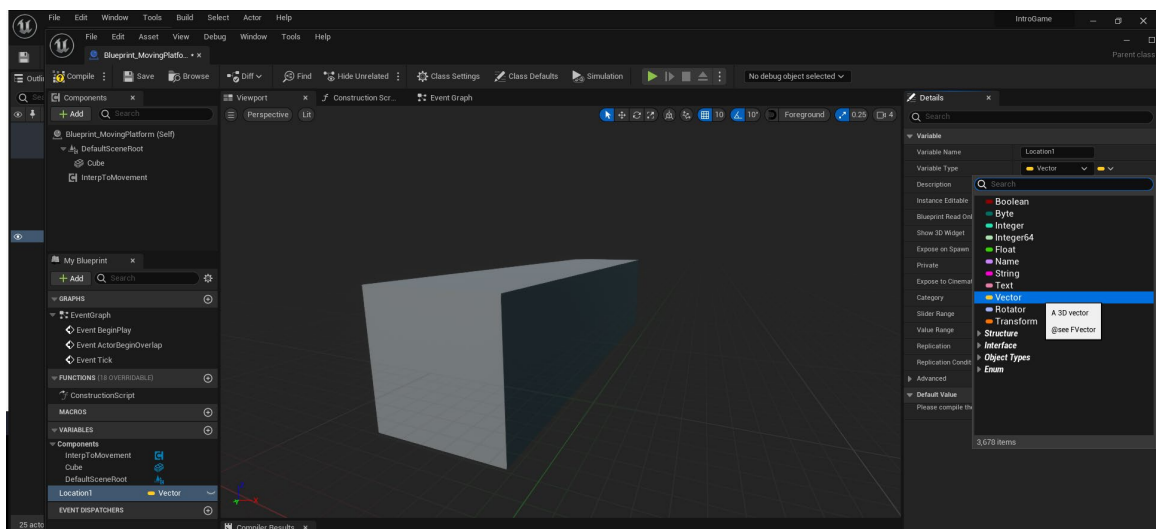
Interpolation is the process of calculating intermediate values between two points. In this case, it helps in smooth movement of the platform.



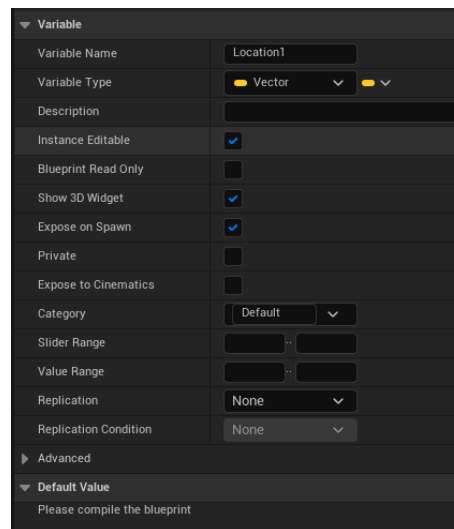
- 7) In the My Blueprint panel on the left, click the \oplus icon next to Variables to add a new variable, which we will use to set the starting location.



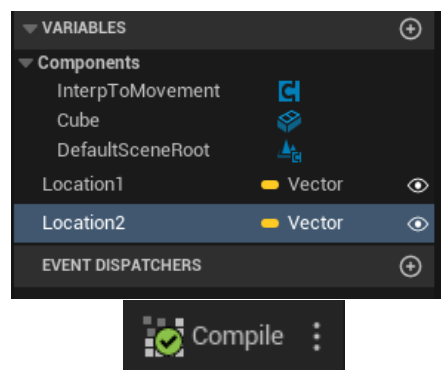
- 8) Rename this variable to Location1 and set its type to Vector.



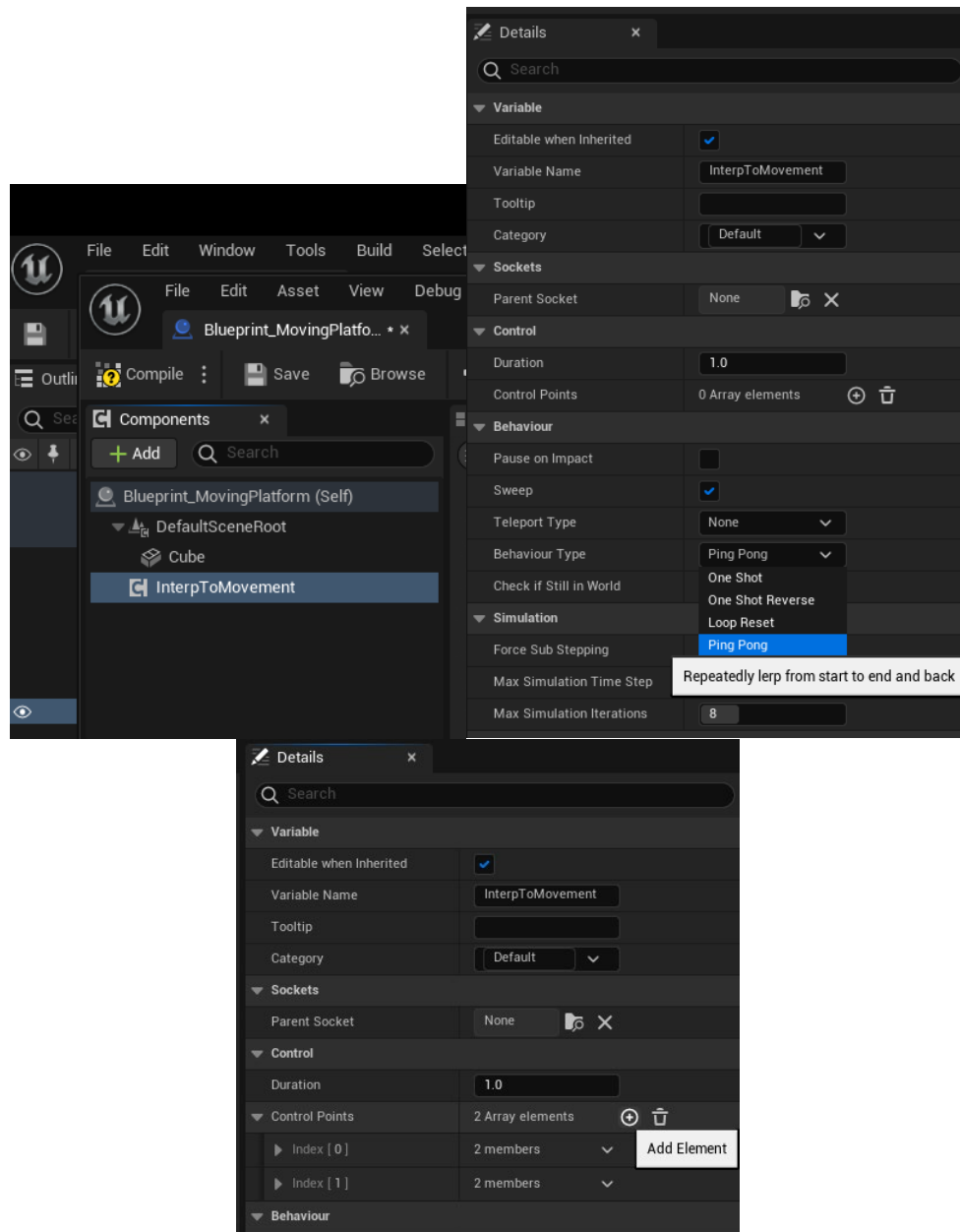
- 9) Set the Location1 variable to be Instance Editable, Show 3D Widget and Expose on Spawn. This will enable us to move and set the location in the main game editor window.



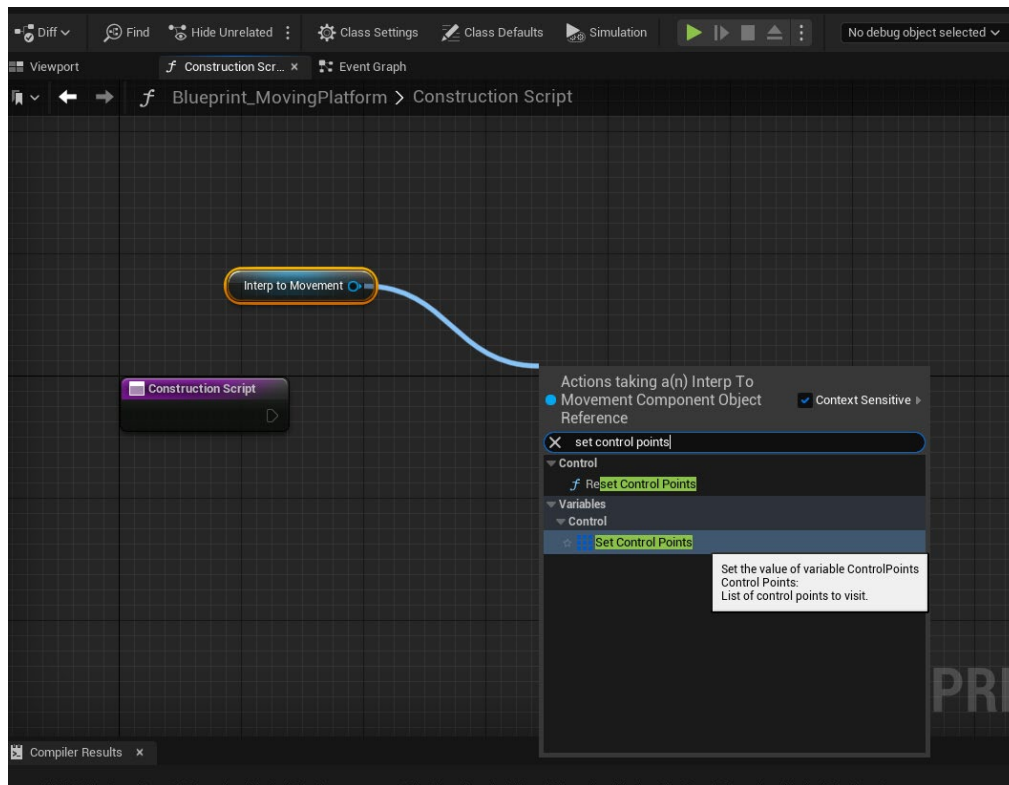
- 10) Repeat this for Location2, (or duplicate) then Compile the Blueprint.



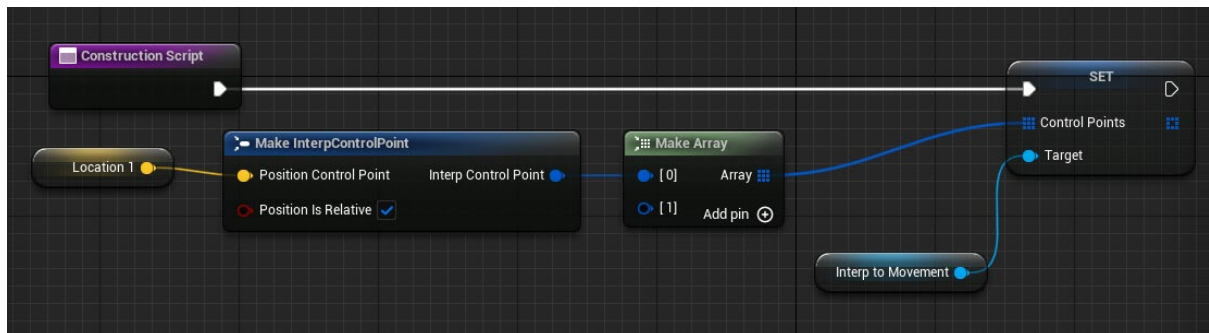
- 11) Select the InterpToMovement component and change its Behaviour Type to Ping Pong, which will allow it to move back and forth. Then click the \oplus icon next to Control points twice to ensure the array has 2 elements; This will store our start and end points.



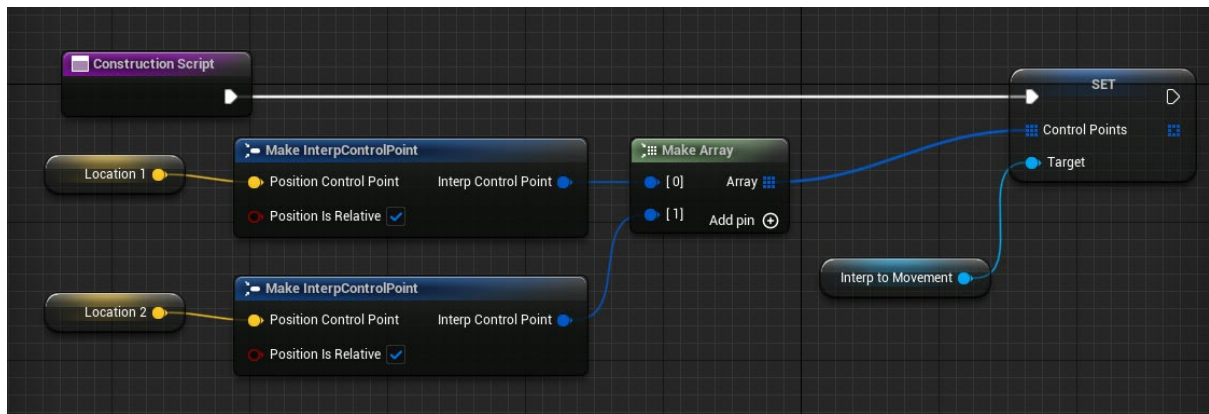
- 12) The Construction Script is executed when an object is created or modified in the editor, making it ideal for initial setup tasks. Here we can use this to set up the object's control points. In the Construction Script tab, drag the **InterpToMovement** component onto the Blueprint, then drag a connector from the blue circle to the right of the node. When the search menu pops up, look for and select **Set Control Points**.



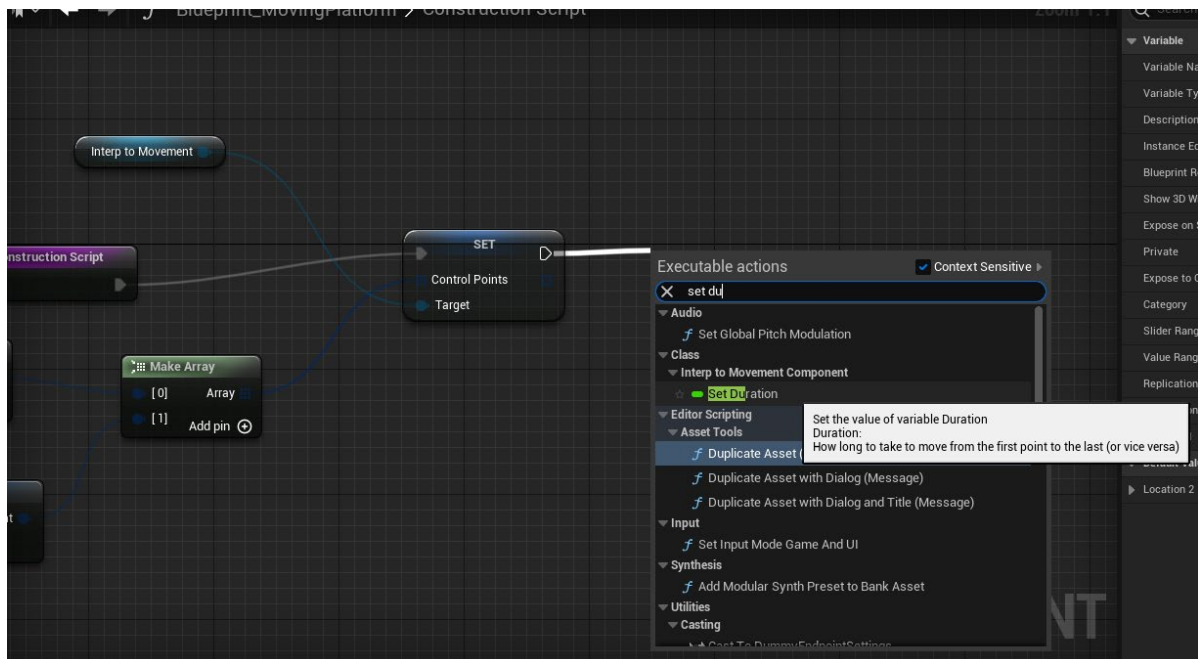
- 13) Connect the **Construction Script** node to the **Set** node so that this is triggered when the object is spawned, then drag a connector out of **Control Points** and search for **Make Array**. Connect element [0] of the array to a **Make InterpControlPoint** node, and drag a connector out of **Position Control Point** and search for **Get Location 1**. Your graph should now look like this.



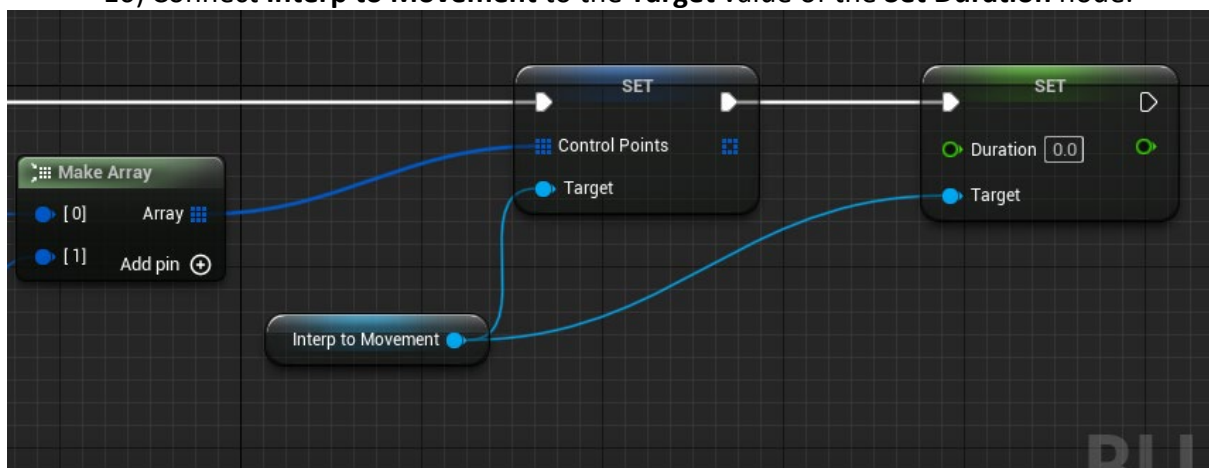
- 14) Make a **InterpControlPoint** for Location 2 via Array element [1] as you did above, so that the graph looks like this.



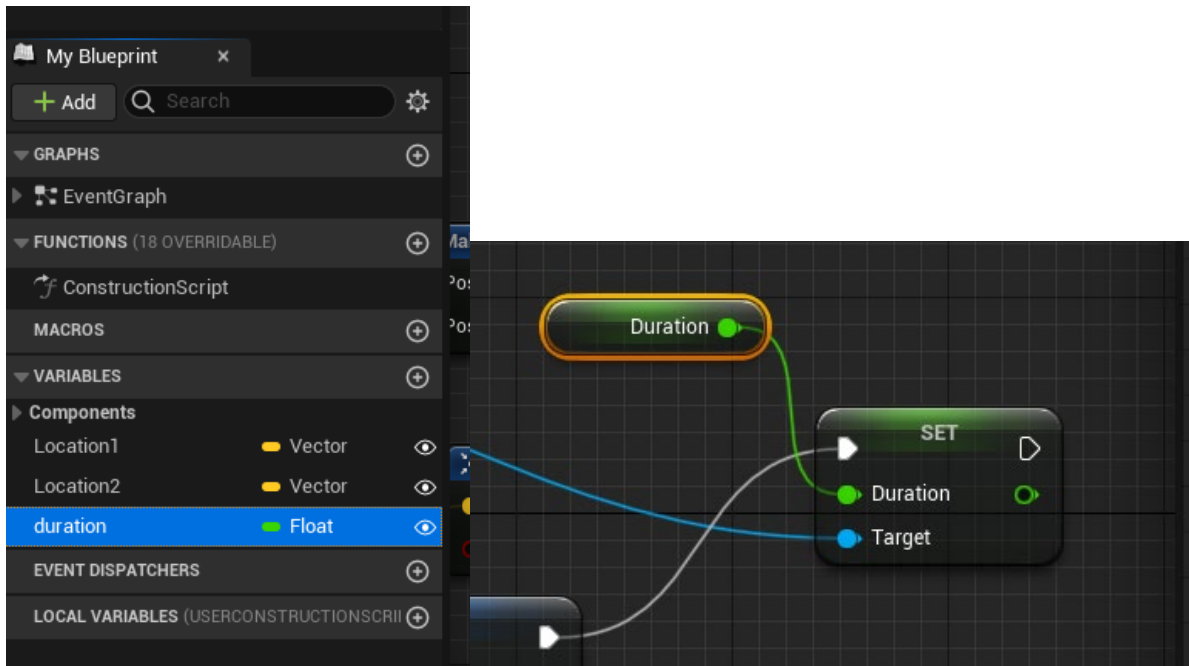
- 15) Drag a connector out of the **Exec** point of the **Set Control Points** node and search for **Set Duration**. This will be used to determine how long the program should interpolate the values to get from **Location 1** to **Location 2**.



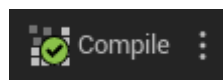
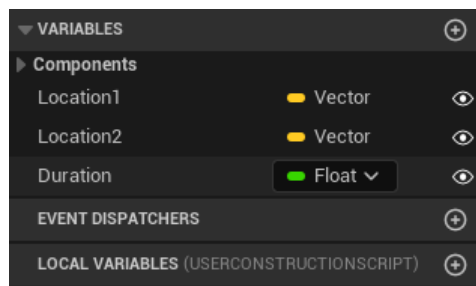
- 16) Connect **Interp to Movement** to the **Target** value of the **Set Duration** node.



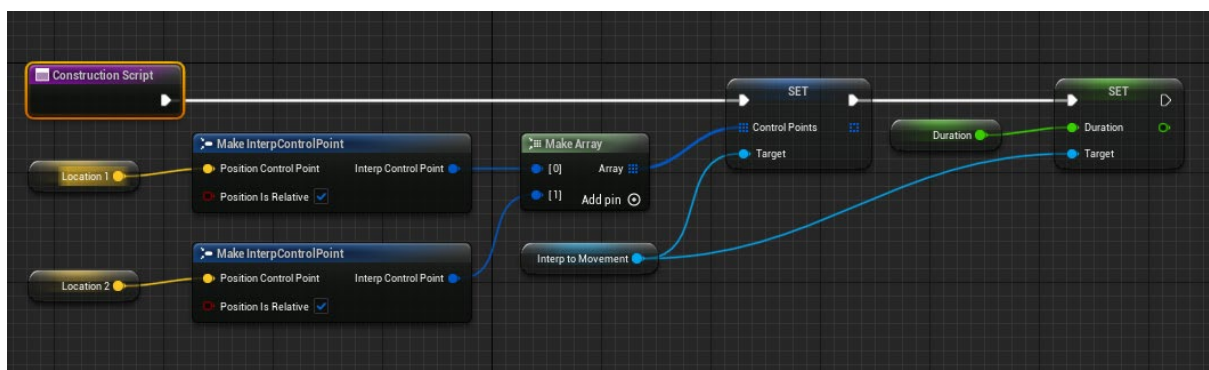
- 17) Right-click on the Duration property of the new Set node and choose **promote to variable**. Notice how you now have a new variable in the My Blueprint section.



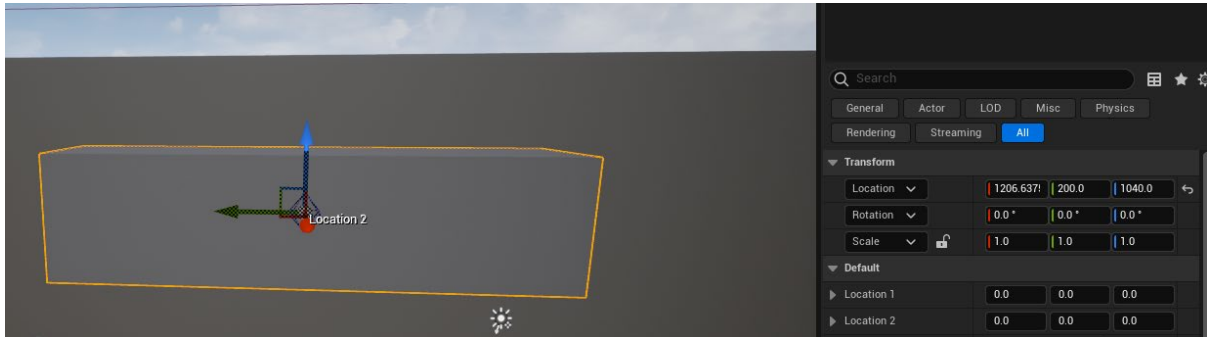
- 18) Make sure to click the eye icon next to Duration to make it Public, so it can be accessed from the main editor window.



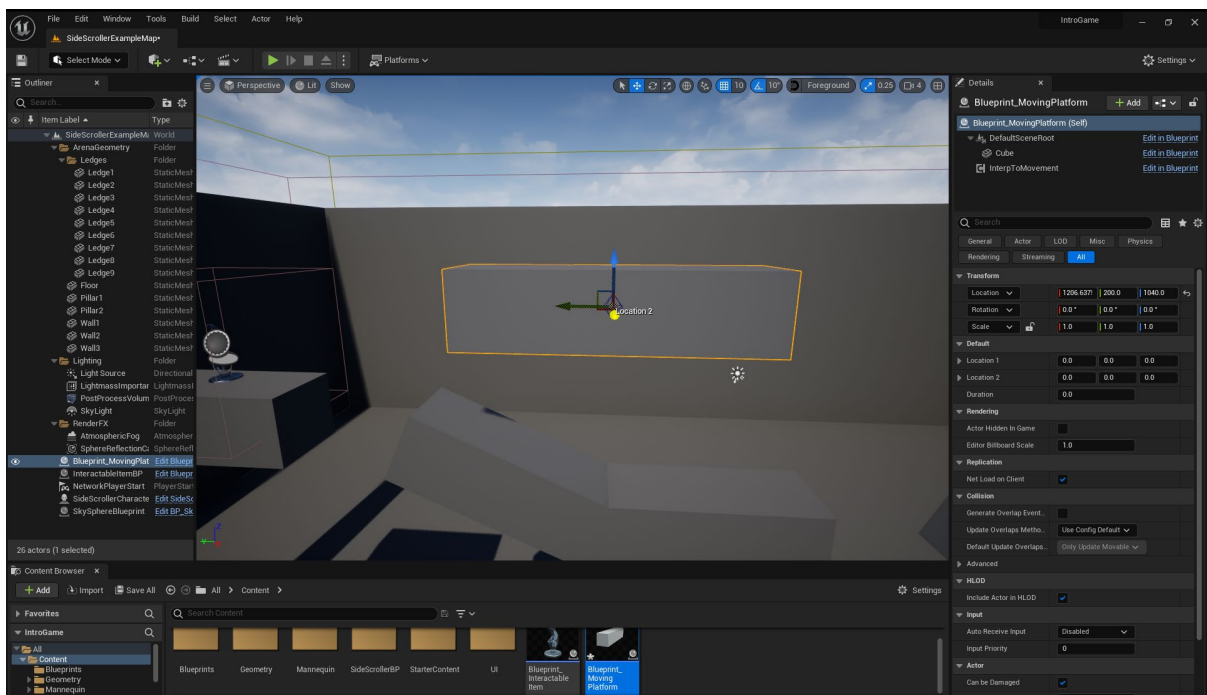
- 19) Your blueprint should now look something like this...



- 20) Save and close the blueprint and then drag an instance of your **Blueprint_MovingPlatform** from the **Content Drawer** to the game world in the main editor. Position the platform so that it is on the same X position as the other platforms, and move it to a location that the player can reach to jump on to.



- 21) Select **Location 2** by hovering over the circle near the centre of the control point, and then clicking on this when it is highlighted in yellow. You can now use the 3D widget to move the control point to the location you would like the platform to move to.



- 22) Set the **Duration** property to 5.0 (this means the platform will take 5 seconds to move from control point 1 to 2. You can adjust the speed of this later after playtesting.

▼ Default				
► Location 1	0.0	0.0	0.0	
► Location 2	0.0	-1010.0	0.0	↩
Duration	5.0			↩

23) Your game should now look something like this. Playtest the game and move the platform around the level to determine the optimal position. You can see below that I have also added a new platform (or 'ledge') at the top right of the level and placed the collectable statue here, so that the player must use the moving platform to get to it. Notice that I have also raised the height of the background walls as the level grows vertically.



TASK TWO: Extension challenges

- Can you build a more complex level, with multiple moving platforms?
- Can you add multiple items to find and collect?