Functions and Timers

Steven Harris

TABLE OF CONTENTS

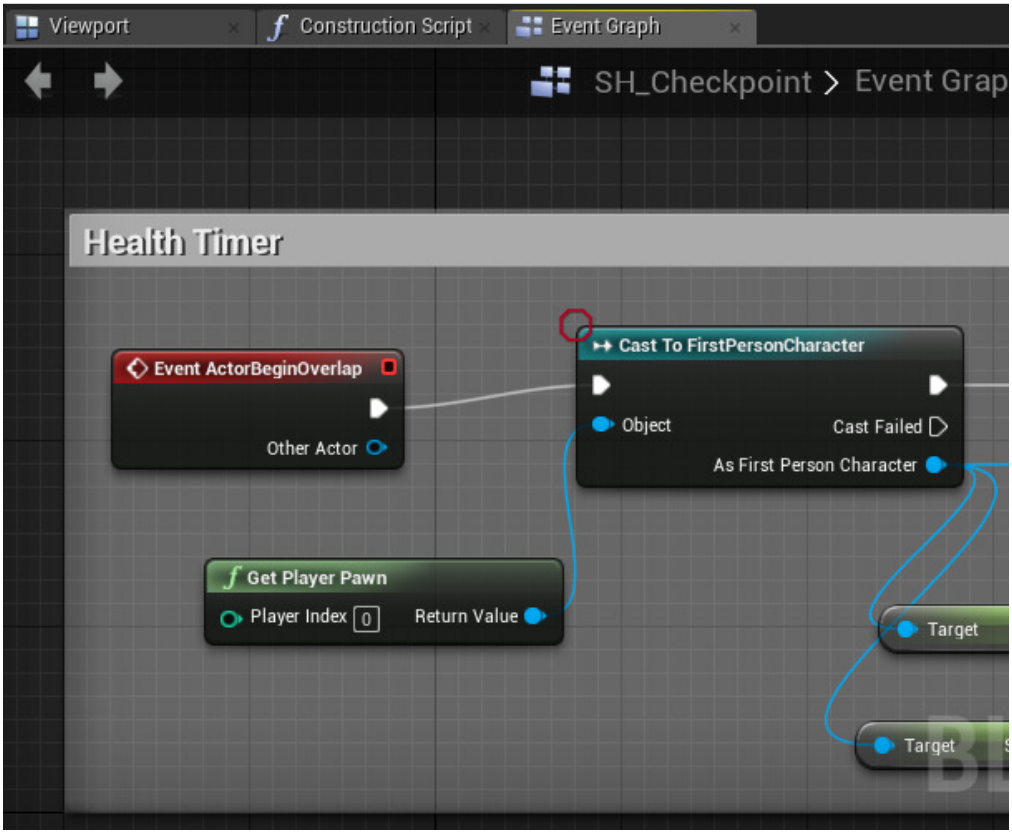
1. CORRECT TERMINOLOGY

1.1 A BLUEPRINT

A BLUEPRINT is the collection of node graphs and 3d Assets brought together as a single game object. The 3D assets are visible through the VIEWPORT tab in the blueprint.

1.2 NODE GRAPH

A NODE GRAPH is the correct name for the collection of commands linked together as a visual script in Unreal Engine. A Blueprint typically has two node graphs to start with, the CONSTRUCTION SCRIPT and EVENT GRAPH tabs, as below. It is possible to add further NODE GRAPHS by adding FUNCTIONS into the blueprint, explained later.



1.3 FUNCTIONS

As with traditional programming, whenever you have a section of code which is repeated twice or more, it is more efficient to put that code into a FUNCTION, and call that function whenever you need that same code to execute. This saves you having to copy and paste large chunks of code over and over again.

In UE4, functions which belong to one blueprint, can be called from another blueprint. This is useful when we need to change a variable in a blueprint from another blueprint. For example, lets say you have a HEALTH variable in the player character blueprint. If you make the health variable public, this would allow any other blueprint to modify it in any way. This can be dangerous, esp when working in a large team. Other developers could manipulate the health value in ways you do not intend, intentionally or accidentally.

A more efficient and more robust approach is to keep the health variable private so no other blueprint can modify it. You then create a function which modifies the health variable in a way which you specify. For example, you may decide that health can only increase in increments of 10 units, and your 'Add Health Function' modifies the health variable by only adding 10 units. That function is then made public, so the only way for another blueprint to modify that variable is to call the 'Add Health Function' you have made public, thereby restricting how that variable can be modified. Any other blueprint which wants to increase the health of the player must go through your 'Add health Function' and modify the value in the way you have determined.

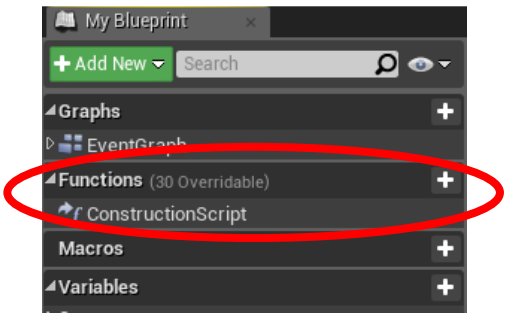
Blueprint Functions: https://goo.gl/mKBdhM

2. CREATING A FUNCTION

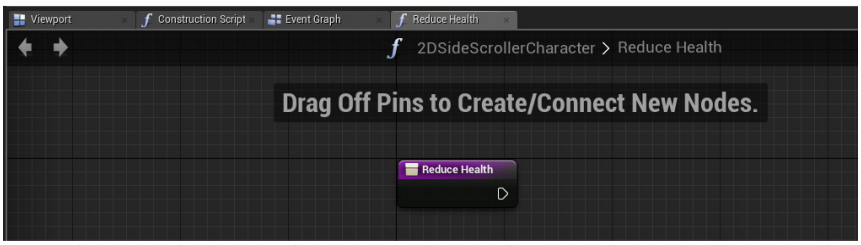
For this example, we will add a function to the player character which will reduce the health by an amount specified by another blueprint. We will do this be creating a FUNCTION which takes a float variable and then subtracts that float from the players current health value. This is preferable to allowing another blueprint to manipulate the variable directly. It is assumed that you have a 'current health' variable on your player character blueprint, in the example below this variable is called **HealthCurrent** and the maximum health value is 1.0 so that it works more efficiently with the slider HUD widget (see HUD notes).

To add a function:

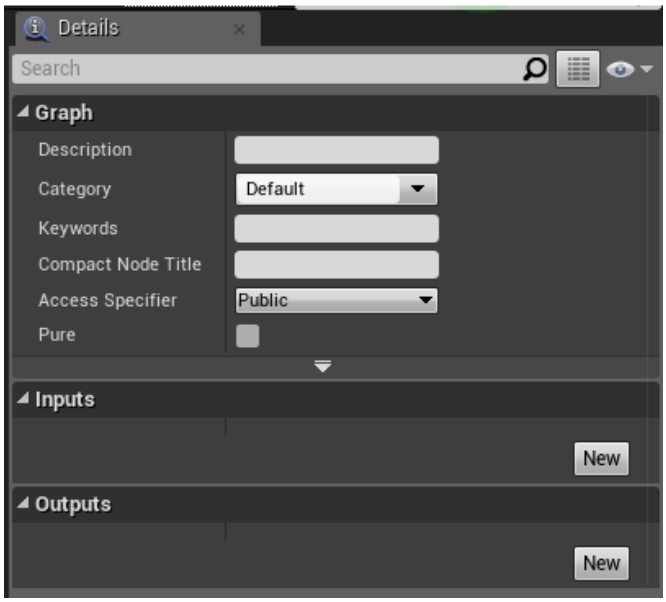
1. Open you player character blueprint with your health variable already added.
2. In the MY BLUEPRINT tab, click the + symbol next to FUNCTIONS, see below.



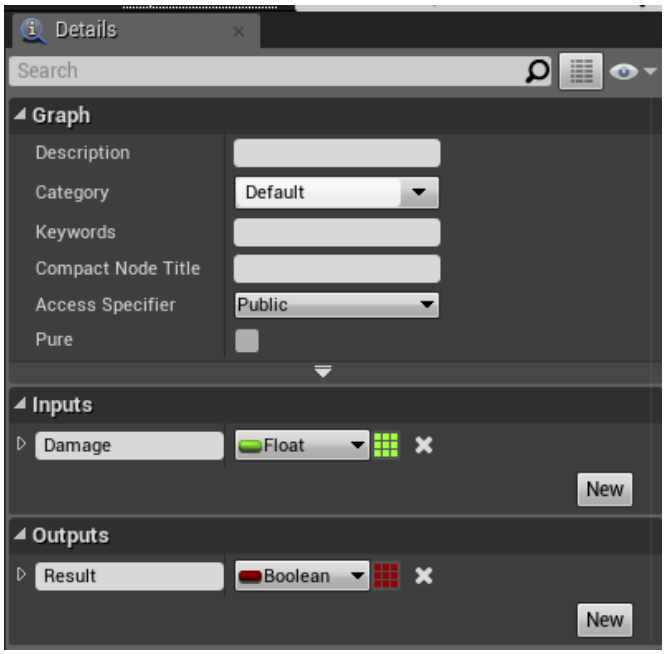
3. Call the new function ReduceHealth.
4. This create a new tab for the function alongside the Event Graph and Construction Script as below.



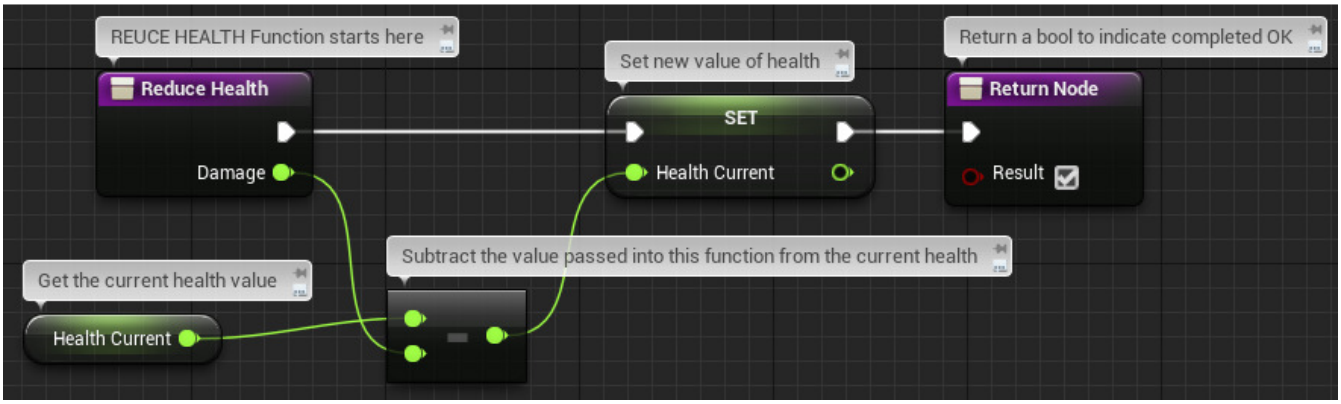
5. You will see a purple node with your function name on it. This is where your function code starts. Add the code below to this function.
6. In the Details tab on the right of the page, you can see the ACCESS SPECIFIER, set this to PUBLIC so that other blueprints can access this function.



7. The INPUTS section allows you to specify any data which you want to pass into this function. In the OUTPUTS section you can add any variables which this function will return back to the code which called this function. In this example, we need an INPUT value to take amount of damage which the other blueprint wants to subtract from the player health variable.
8. In the INPUTS section add a float variable called DAMAGE. Also add a boolen variable to the OUTPUTS section called RESULT, as shown below.



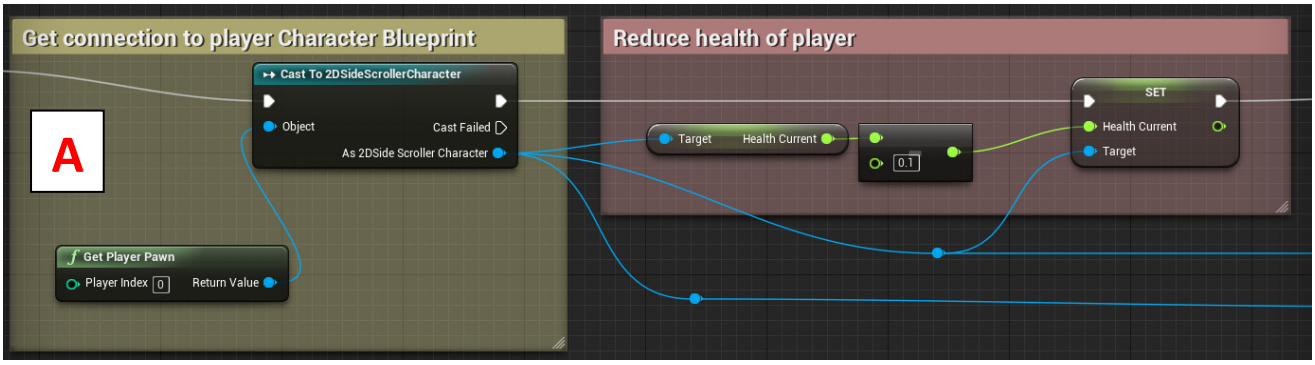
9. In the node graph, add the code below. The RETURN NODE command can be found by searching the command list when you right click on the workspace. Make sure you tick the RESULT variable in the RETURN NODE, so that a TRUE value is sent back from this function when it is called. Make sure you wire the pins in the correct order into the FLOAT – FLOAT node, the top node is the value from which the lower node will be taken away from.



10. Compile the code when compete. You now have a function which you can call from this blueprint or another blueprint to reduce the current health by a certain amount.

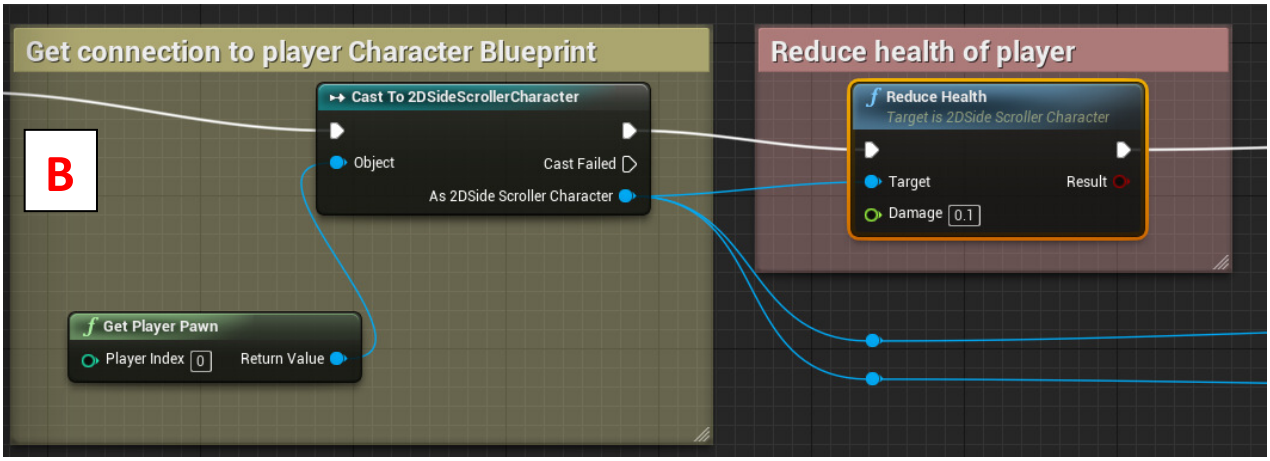
2.1 CALLING A FUNCTION

Open the blueprint of the actor which will cause the damage to player. In this example, we have a flag which when touched will reduce the health of the player, by directly manipulating the health variable in the player character blueprint, as seen below in example A.



We will change the red commented area to use the function we have just created.

1. To add the REDUCE HEALTH function node you must drag from the CAST TO node, this will ensure you get access to all the functions from the blueprint to which the CAST TO node is linked (in this case the PLAYER PAWN). If you right click on the workspace you will NOT find the REDUCE HEALTH function unless you untick the CONTEXT SENSITIVE option in the command list.
2. Connect the function as below, in example B



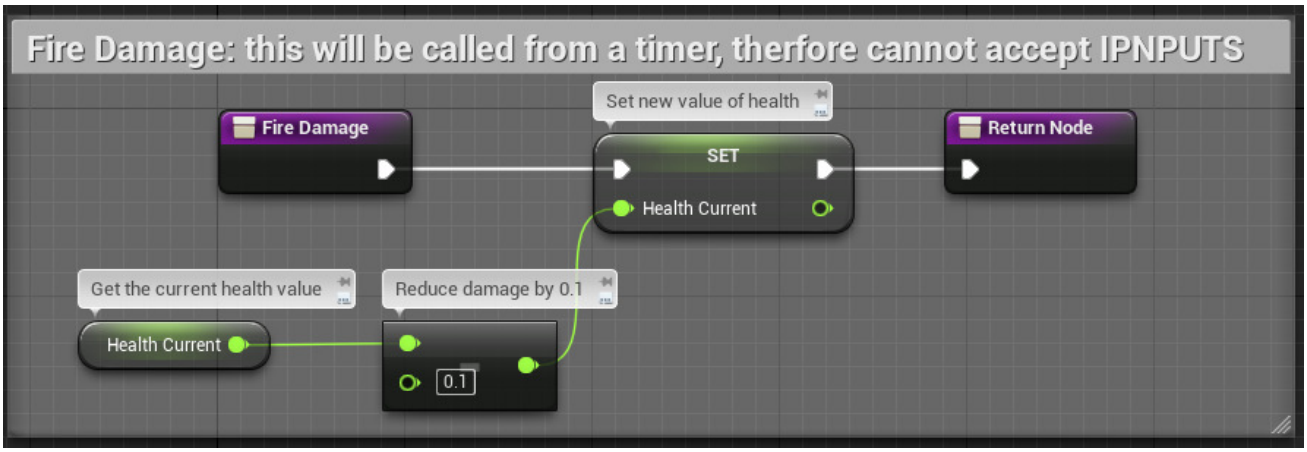
3. You will see that this makes the blueprint clearer, and also means that you no longer have to repeat the same code to manually reduce the health over and over again in different blueprints, as in example A above.

3. TIMERS

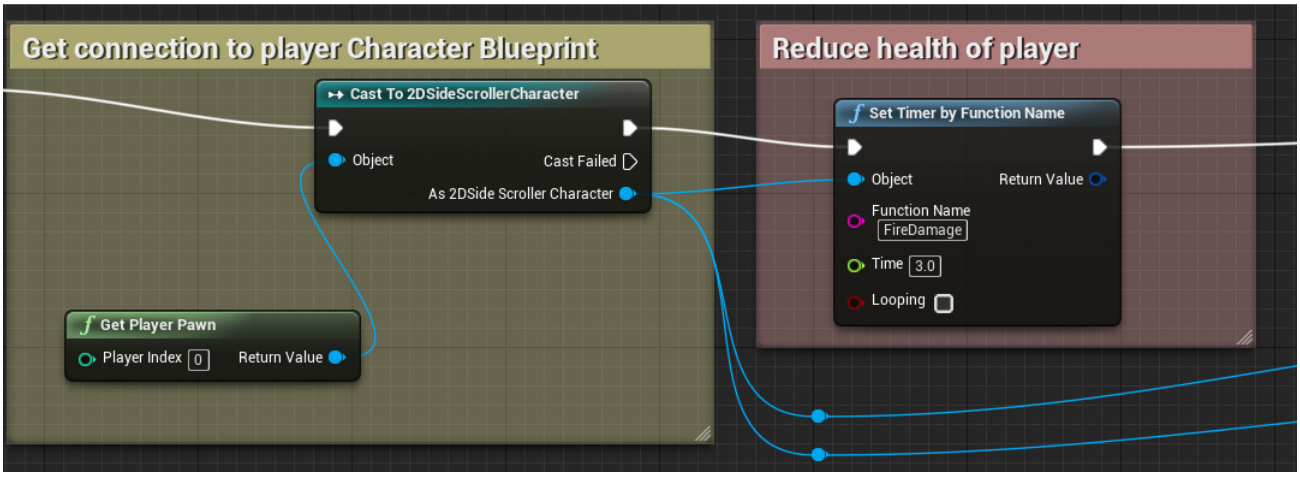
Timers can be used to repeatedly call a function in a blueprint. This can be useful in the example of the health variable to reduce the health value at regular intervals , for example when applying 'fire damage' so the player loses 10% health for every 2 seconds the character is in a certain area.

To call a function a function after a certain amount of time:

1. We will use the SETTIMER node in Blueprint. Currently there is no easy way of passing an input value into a function when using a timer, so you must plan carefully. The previous example where we could send the amount of damage to the function will not work with a time.
2. As an example, we have created a new function called FireDamage which subtracts a hard-coded value of 0.1 from the health. We will then use a timer to apply this damage after a certain amount of time.
3. In the player character blueprint where you created the previous Reduce Health function, create a new function called FireDamage with the nodes as below.



4. Note that there are no INPUTS or OUTPUTS, and the dame has been explicitly sated as 0.1.
5. Open the blueprint of the actor which will cause the damage to player. This is the same blueprint we used for Example B above.
6. From the CAST TO node, drag from the white arrow and search for the SET TIMER BY FUNCTION NAME node. In this node connect the object pin to the blue output pin from CAST TO, as below. Then manually type in the function name you want to call, in this case FireDamage. Then enter a TIME value which is the amount of time you want to pass before that function is called, 3 seconds in the example below.



7. Now when the event fires off the function will be called after a three second delay.
8. You can use the LOOPING option to constantly loop the timer once it has finished and start from the beginning again.
9. You can stop a timer and reset it back to its start time by using the CLEAR TIMER BY FUNCTION NAME node, as below. You will manually have t enter the unction name.

10. This can be useful if you place a collision box around your actor in the VIEWPORT tab of the blueprint, you can then use the OnComponenetBeginOveralp event on the box to start the timer when the player enters the trigger volume, and then a OnComponenetEndOverlap on the box to clear the timer when the player exits the trigger volume. This will repeatedly apply timed damage while the character is inside the volume and stop when it leaves