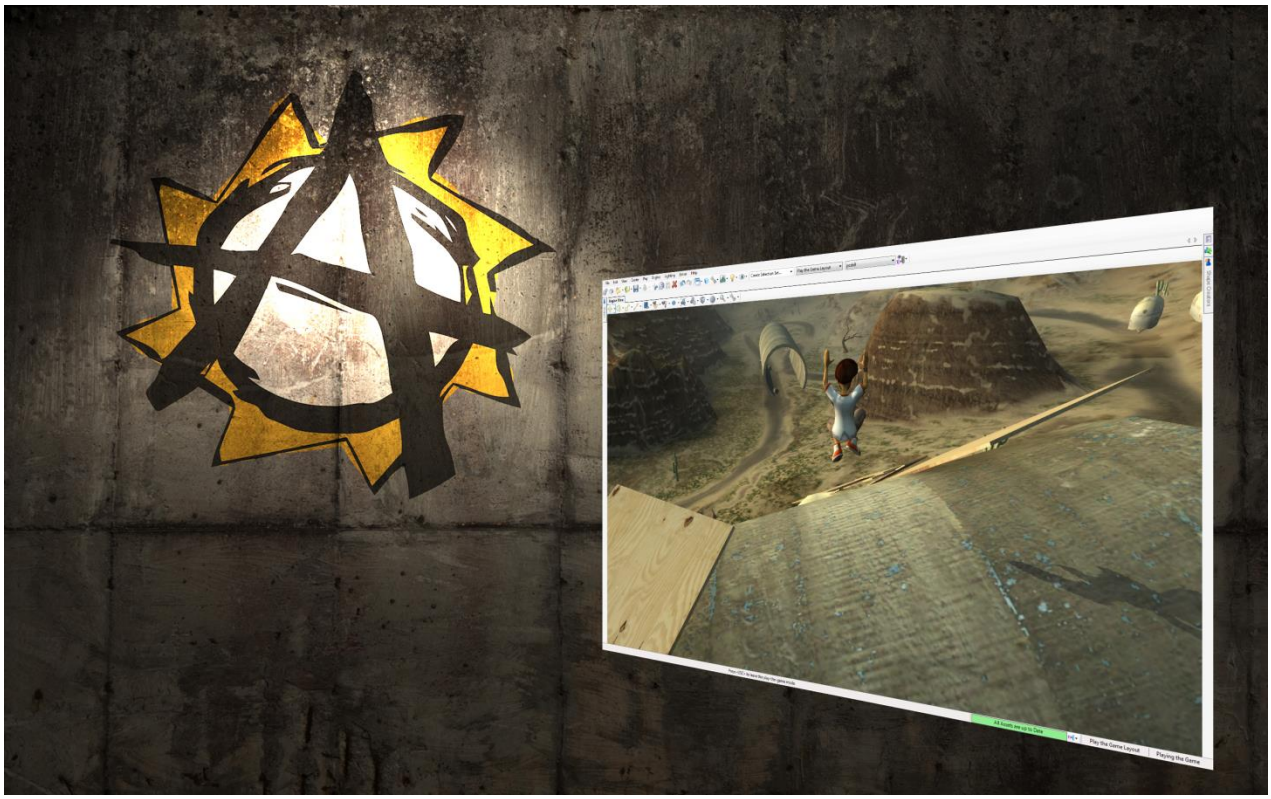
Anarchy for the USB 2

Rapid 3D game prototyping from idea to Device in less than an hour using Project Anarchy



Contents

Pre-requisites

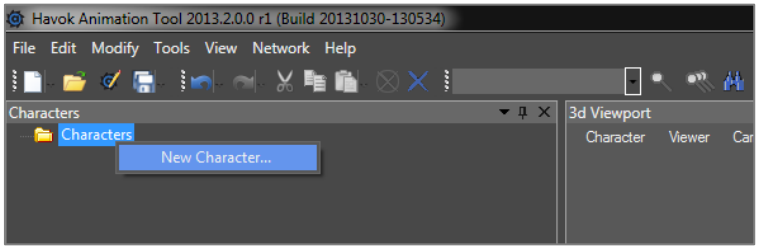
- A host PC with the Project Anarchy SDK installed.
- Android SDK and tools.
- A target Android device.
- The necessary device drivers to connect the device via adb.
- The supplied vPlayer.apk installed on the Android device
- The asset bundle CharlieTalk.zip file
- A Wi-Fi connection for both host PC and Android device†.
- A game controller connected to host PC

† *USB connection between device and PC may be used in lieu of Wi-Fi.*

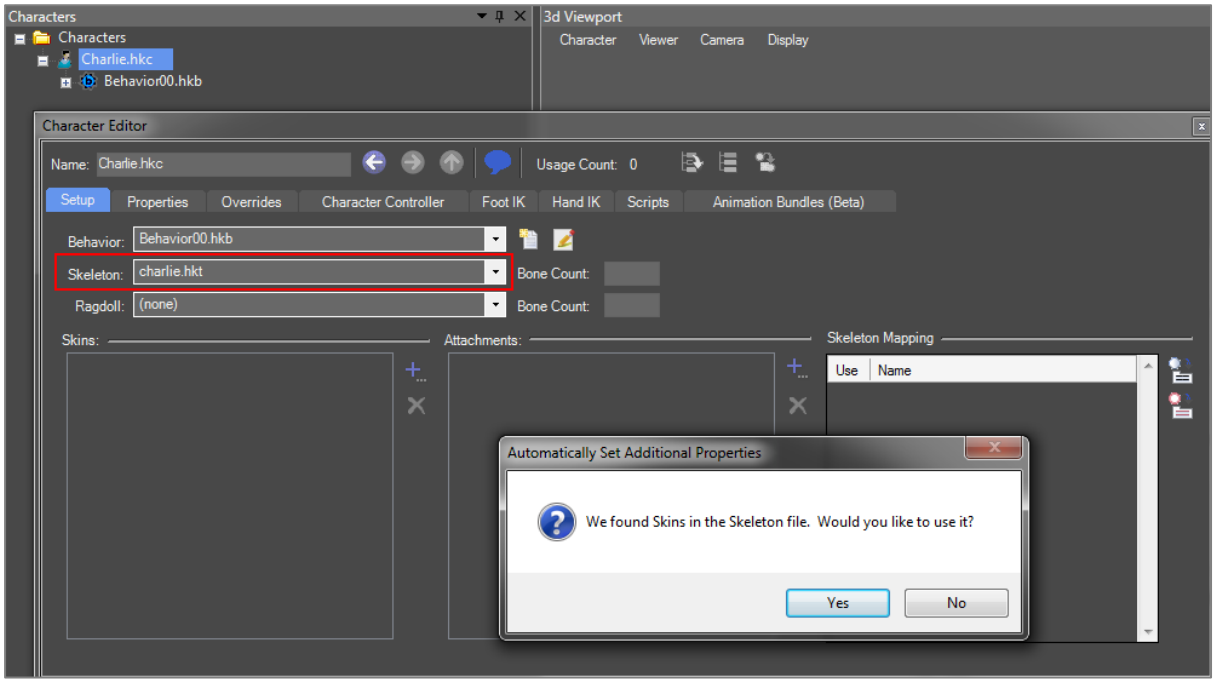
Creating the Character

Launch Havok Animation Studio and open the ***Charlie.hkp*** project.

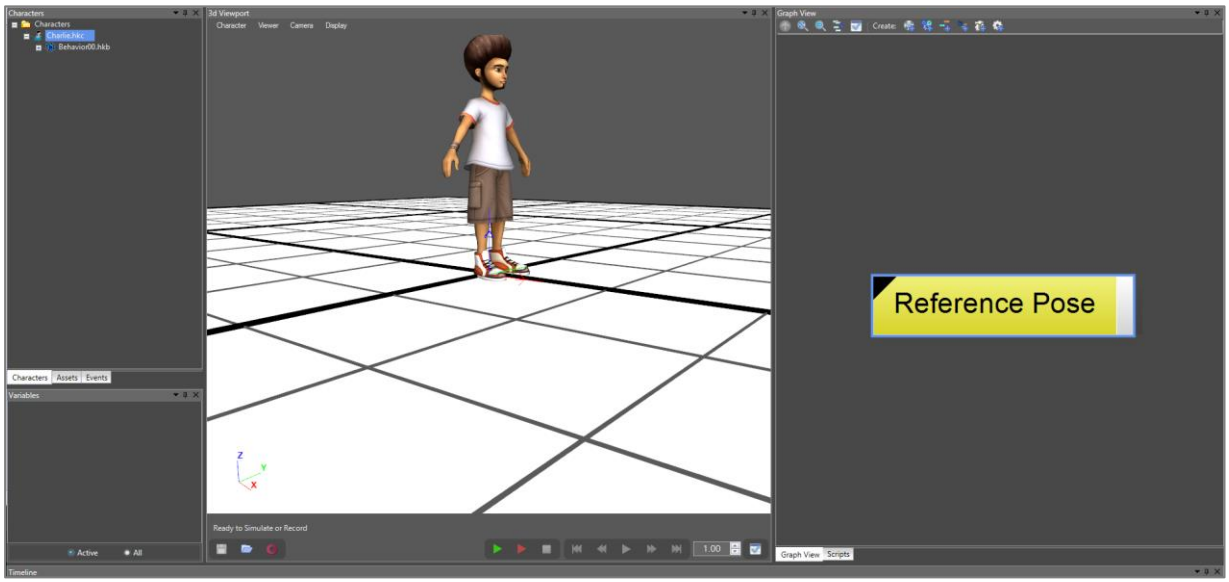
RMB on the **Characters** folder and select **New Character**. In the File Dialogue that opens, ensure the path is set to the **Characters** folder and enter ***"Charlie"*** as the filename and click on ***Save***.



Under the **Characters** folder a ***Charlie.hkc*** node will have been created with various other child nodes below. *Dblclick* on the ***Charlie.htc*** node to open the character editor. Using the drop-down selector set the skeleton field to ***charlie.hkt***. A pop-up will prompt using the skins found in the skeleton file. Select ***Yes***.

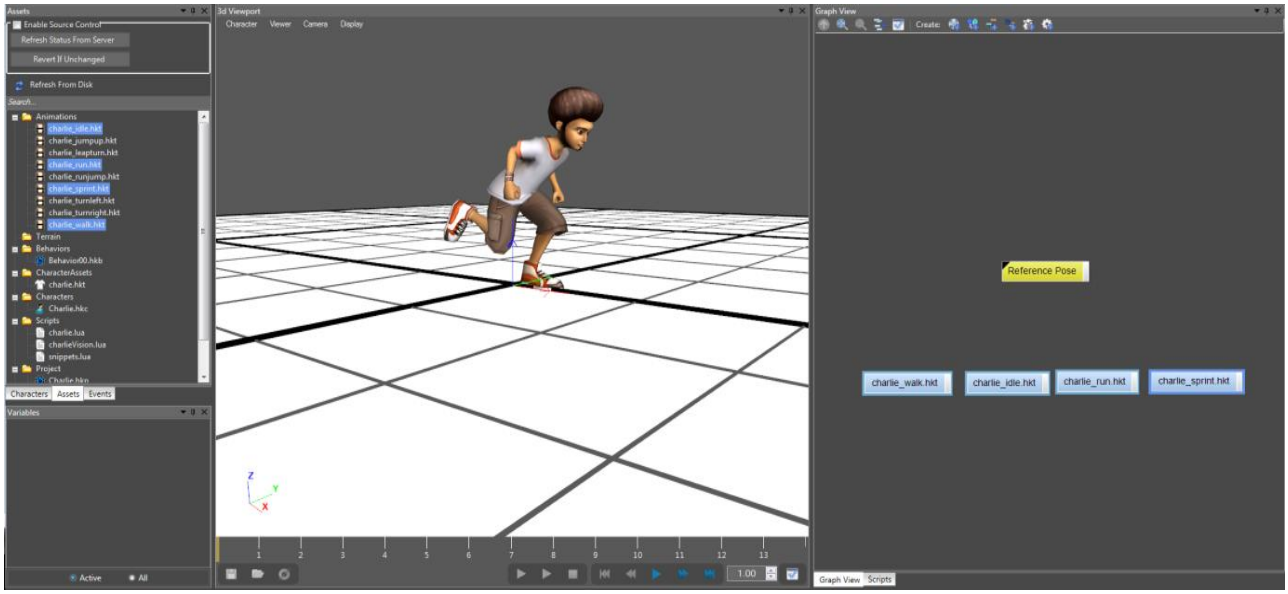


Close the Character Editor dialogue and all being well the Charlie character will be visible in the 3d Viewport displaying the Reference Pose.

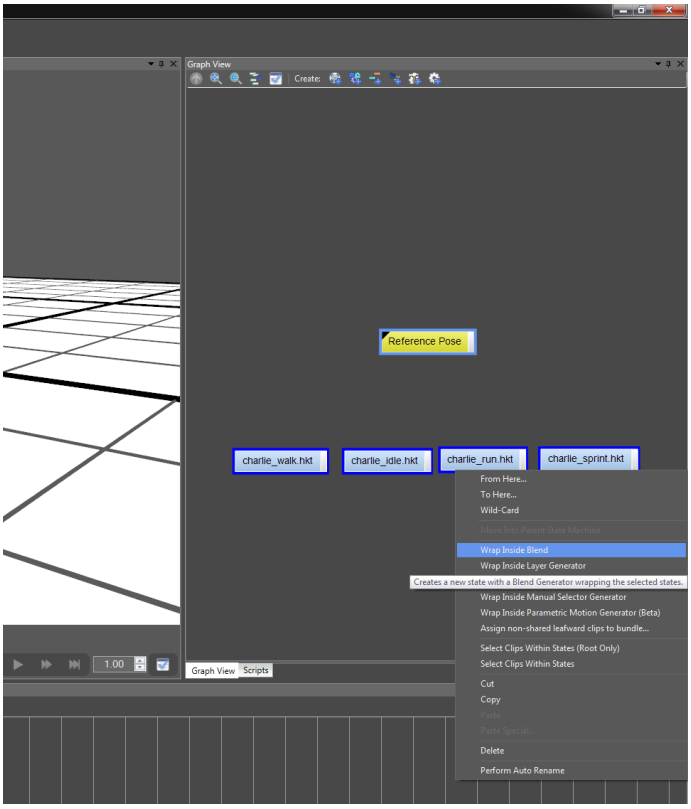


Adding forward motion

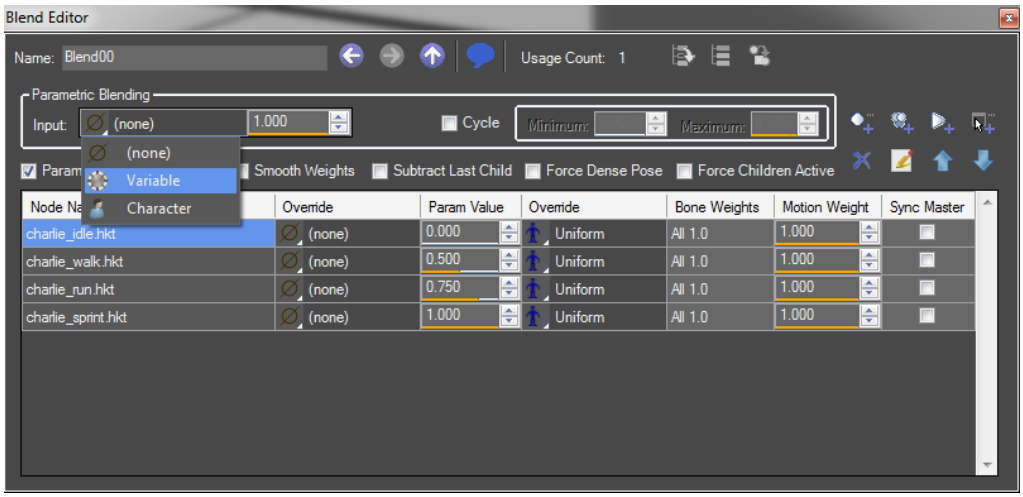
Open the **Assets** tab and select the following animations: ***charlie_idle.hkt***, ***charlie_walk.hkt***, ***charlie_run.hkt*** and ***charlie_sprint.hkt***. Drag them into the **Graph View** pane.



Now in the **Graph View** pane, use the box selection tool to highlight the four animations, and in the *RMB* pop-up menu select **Wrap Inside Blend**.



Returning to the **Characters** tab expand the newly created **State00** node and *Dblclick* on the **Blend00** node to open the **Blend Editor**.

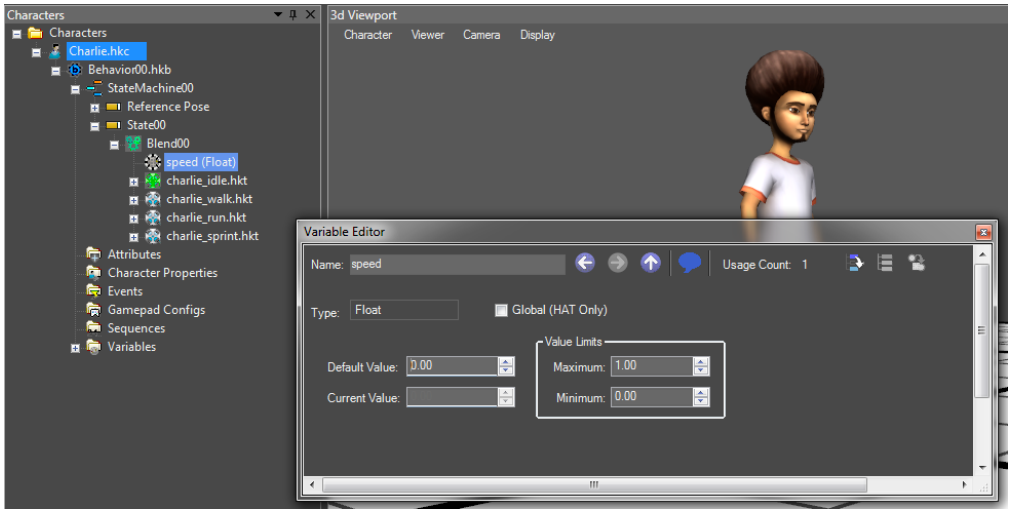


Check the **Parametric Blend** check box and set the **Param Value** fields as follows:

charlie_idle.hkt		0.000
charlie_walk.hkt		0.500
charlie_run.hkt		0.750
charlie_sprint.hkt		1.000

In the **Parametric Blending** field create a new **Input** variable using the pull down. Name the variable ***"speed"***.

The new ***"speed"*** variable will appear as a node under ***Blend00***. *Dblclick* on this to invoke the **Variable Editor**. (Alternatively *Dblclick* on the ***"speed"*** variable in the **Parametric Blending** field)

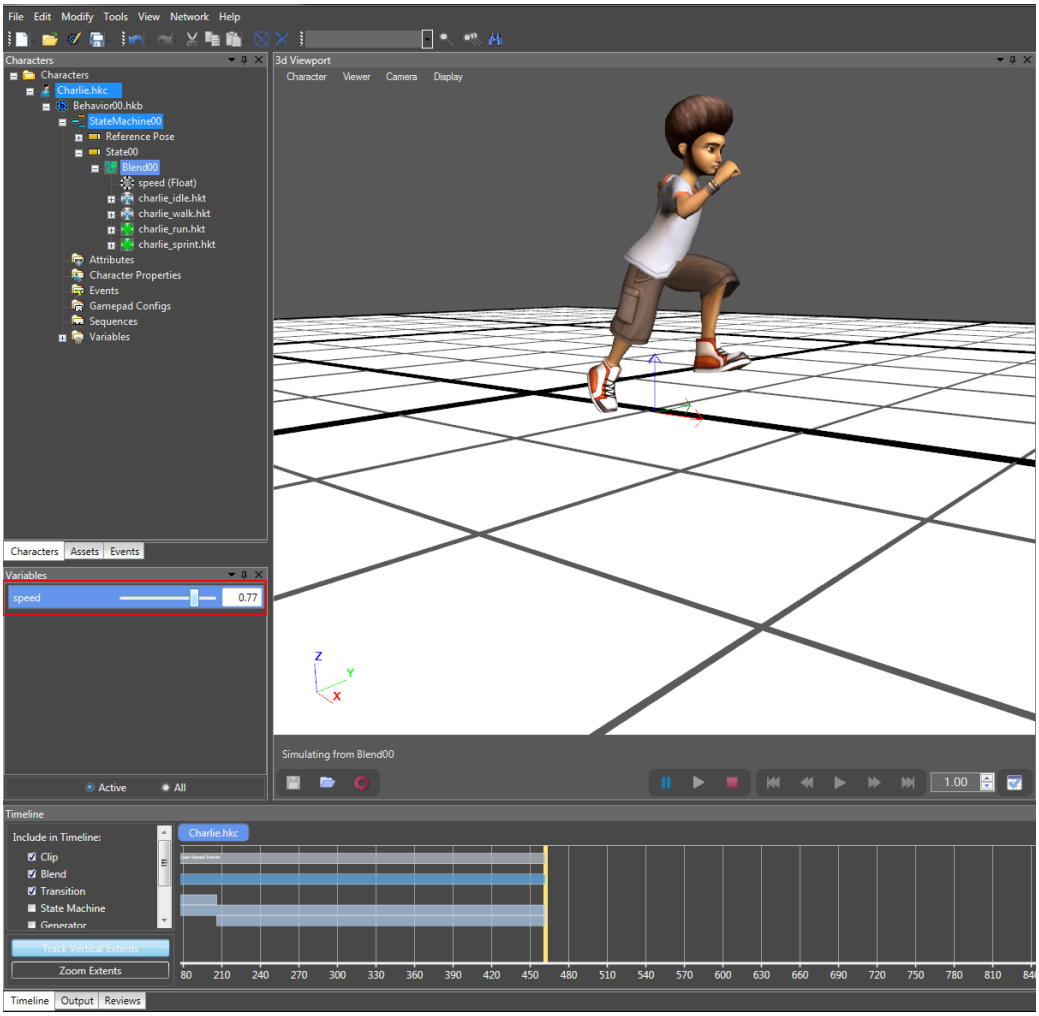


Set the properties such:

Minimum		0.00
Maximum		1.00
Default Value		0.00

Ensure the **Variables** view is enabled from the menu bar *View->Variables* and docked as a separate pane or tab. In the **Characters** tab select the ***Blend00*** node and run the behaviour by either pressing ▶ in the 3d Viewport or by pressing *SPACE*.

A slider representing the ***"speed"*** variable should be visible in the **Variables** view. Use this slider to increase or decrease the characters forward velocity.



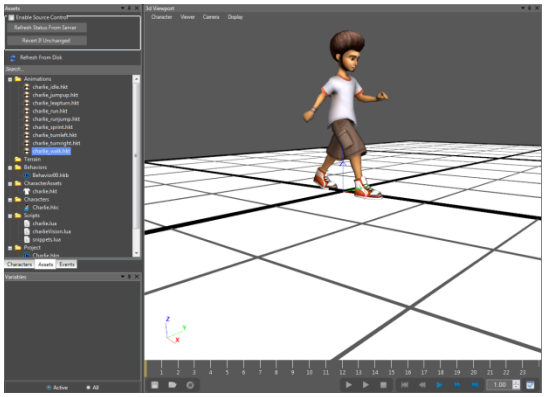
Note the seamless blending between the animations.

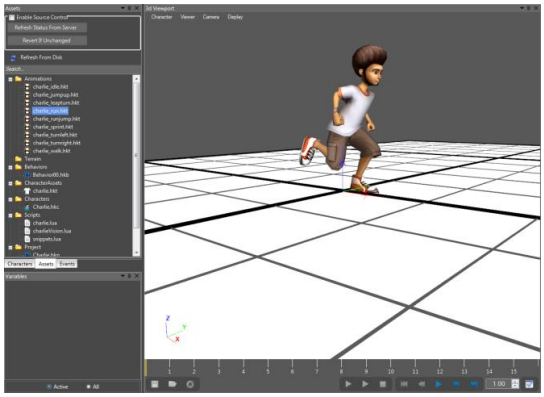
Or Not!!

There seems to be some strange behaviour with values between 0.5 and 0.75!!

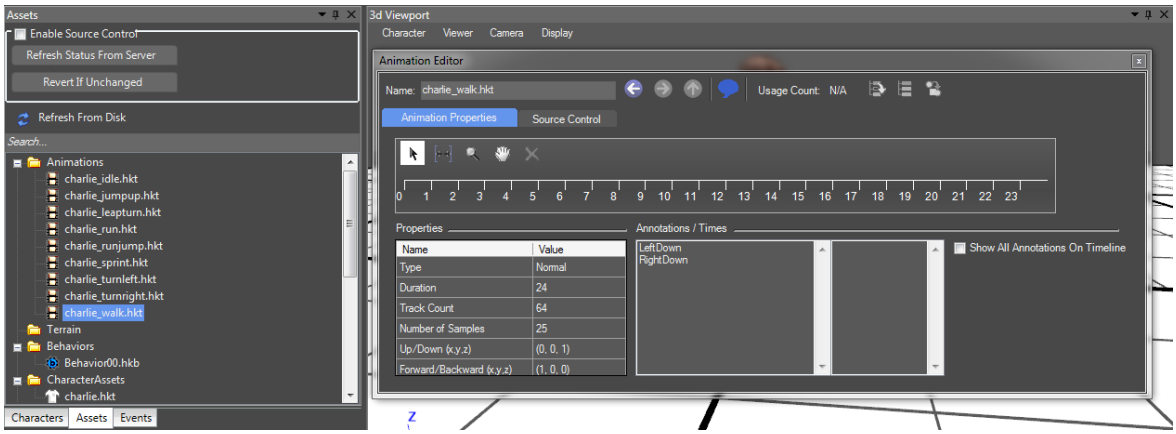
Synchronising the feet

Back in the **Assets** tab select first the walk animation then the run animation and the cause of this behaviour should be immediately obvious. We can see the walk leads with the right foot, whilst the run leads on the left.





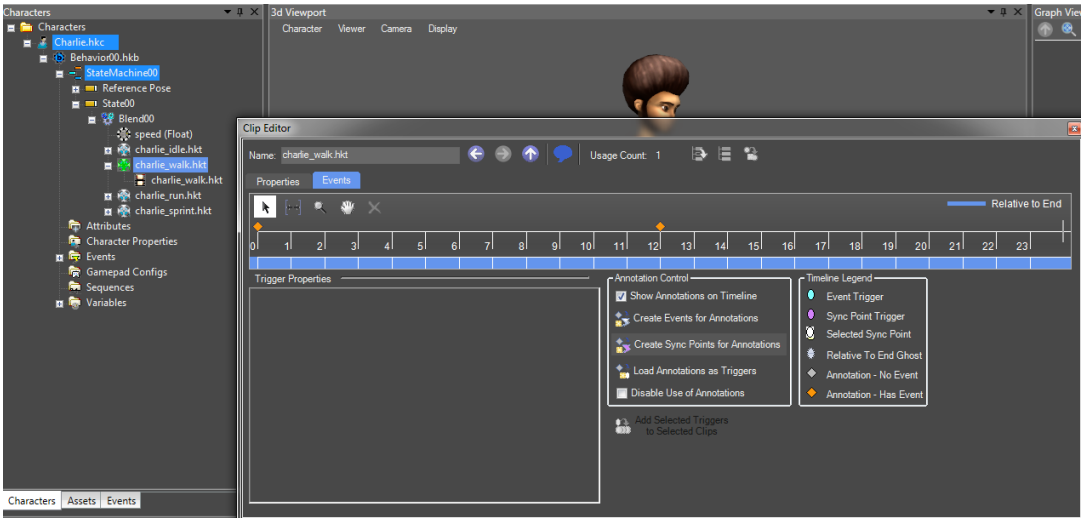
Dblclick either of these animations to open the **Animation Editor** then under the **Annotations / Times** fields there should be two labelled events: ***LeftDown*** & ***RightDown***.



These events were generated during the export process using the **Havok Content Tools** and signal the point of time in the animation when either the left or right foot is flat down on the ground and. *For a detailed explanation please see Appendix B.*

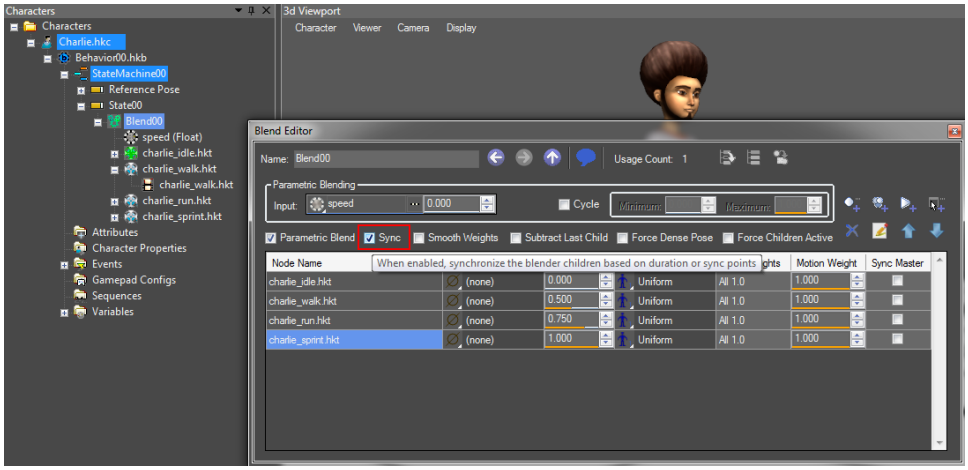
Return to the **Characters** tab and expanding the ***Blend00*** node *Dblclick* the ***charlie_walk.hkt*** node to open the Clip Editor. Select the **Events** tab and under the **Annotation Control** field click the **Create Sync Points for Annotations** button.

Note the two diamonds on the event timeline will change colour from grey to yellow indicating they are activated.



Repeat for ***charlie_run.hkt*** and ***charlie_sprint.hkt***.

Finally open the **Blend Editor** from ***Blend00*** node and check the **Sync** check box.

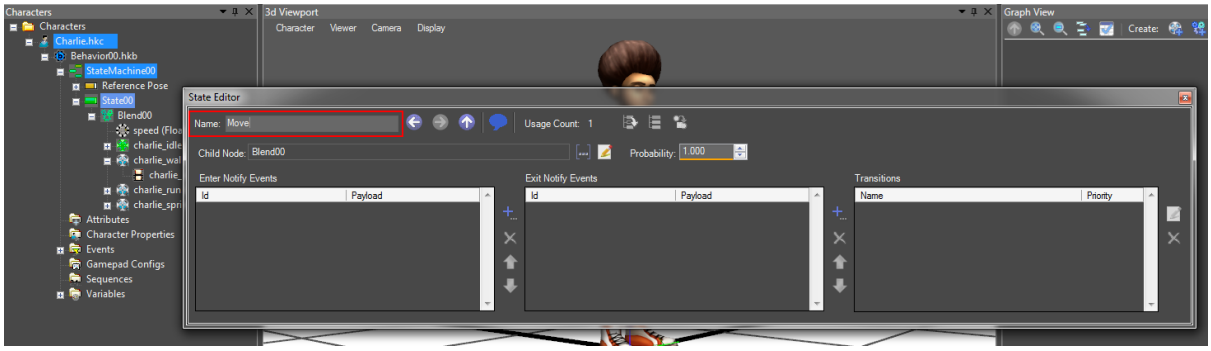


Again with the ***Blend00*** node selected run the behaviour (▶ or *SPACE*) and adjust the ***"speed"*** slider and note now that the feet are now synchronised and seamlessly blending.

Rename the State

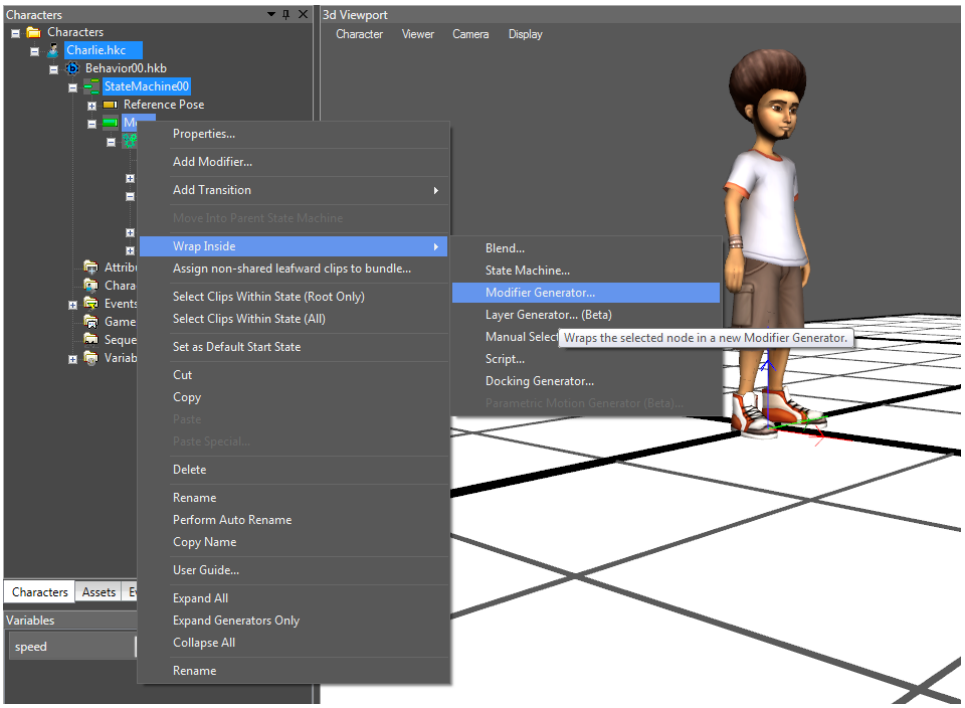
To aid with clarity and future tasks, now is a good time to give a meaningful name to the ***State00*** node.

Dblclick on the ***State00*** node to invoke the **State Editor** and change **Name:** to ***"Move".*** Ensure to press *RETURN* on the string entry field before closing the dialogue to register the change.



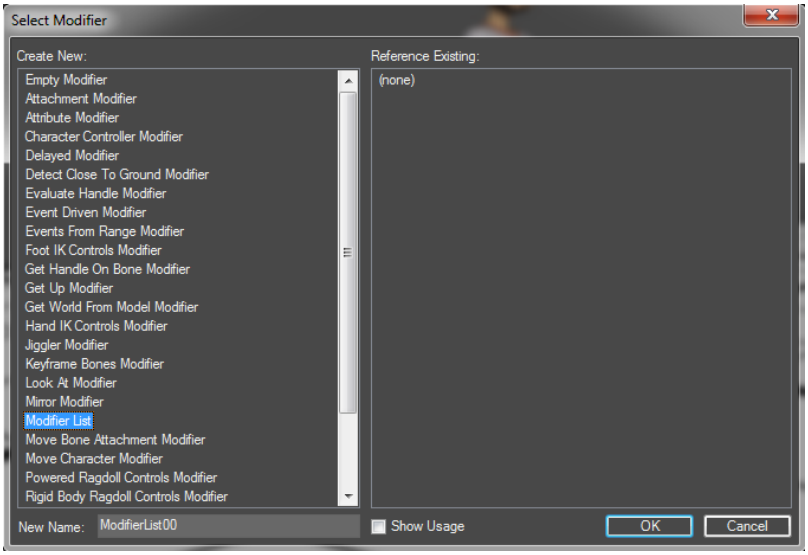
Adding further modifications

RMB on the ***Blend00*** node and from the pop-up menus select ***Wrap Inside -> Modifier Generator…***



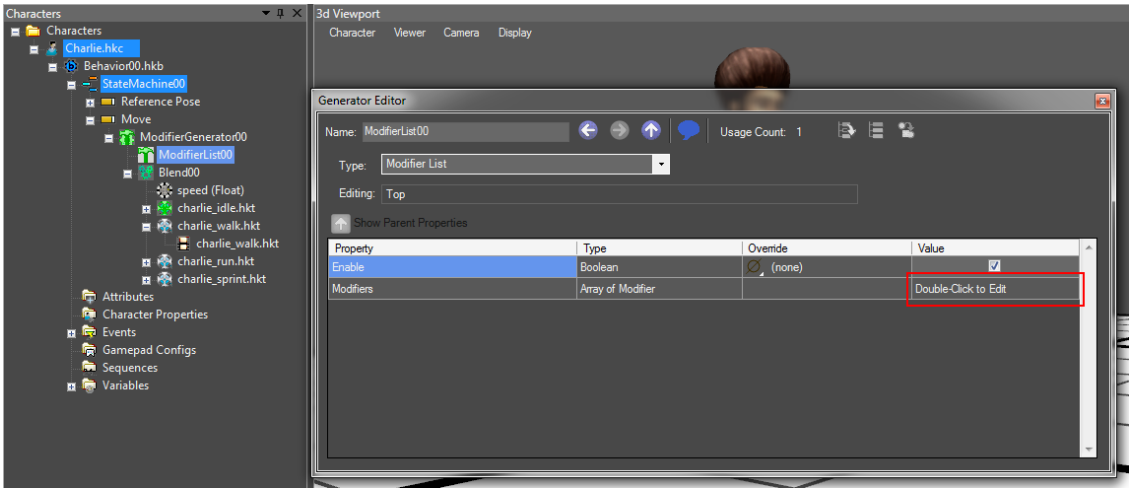
The **Select Modifier** dialogue will be displayed.

As we will be adding more than one modifier to our character we will first add a **Modifier List** by selecting it from the **Create New:** control and pressing **OK**.

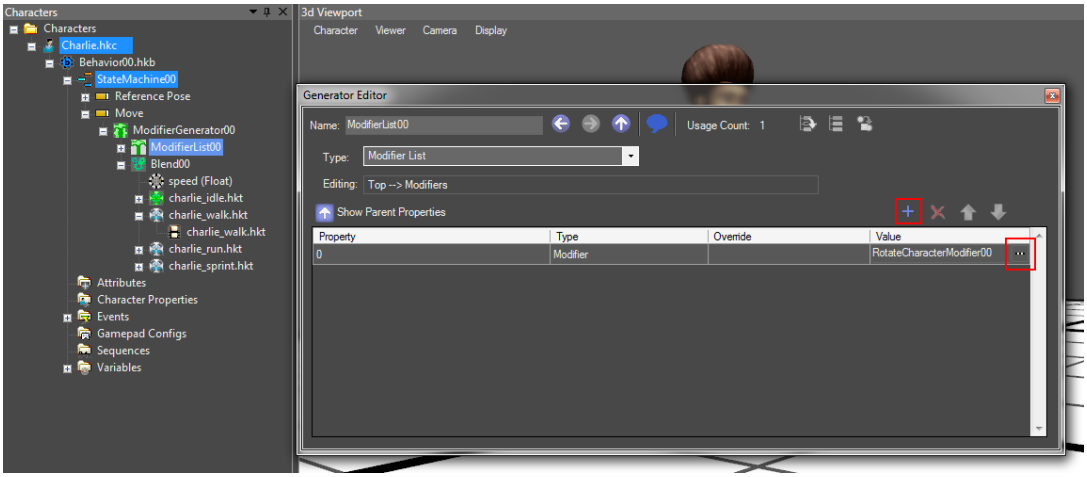


Adding a Rotate

In the **Characters** tab there will be a newly created ***ModifierGenerator00*** node with ***Blend00*** and the new ***ModifierList00*** as child nodes. *Dblclick* on the ***ModifierList00*** node to open the **Generator Editor** and *Dblclick* on the box labelled **Double-Click to Edit**.



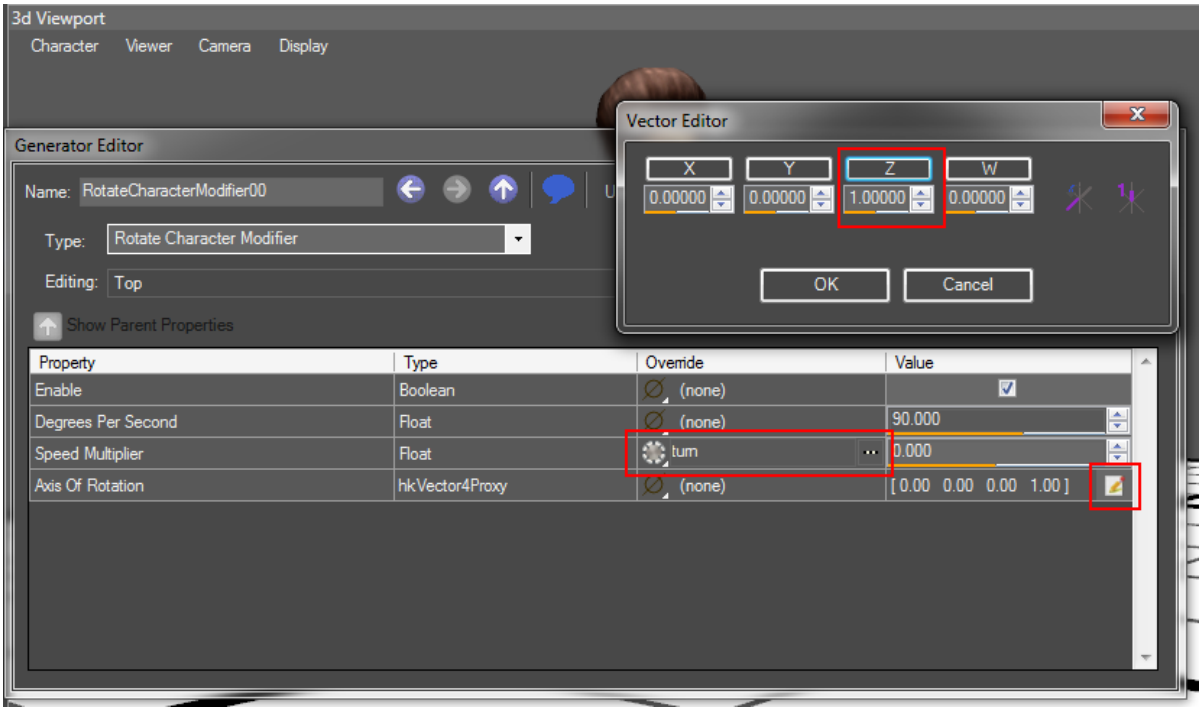
Click on Add + and open the selection list ▪▪ and select **Rotate Character Modifier.**



Dblclick on the newly created ***RotateCharacterModifier00*** to invoke the **Generator Editor** for this node.

Inside the **Speed Multiplier** field create an **Override** variable called ***"turn"***.

Inside the **Axis of Rotation** field use the **Vector Editor** (pen / pad icon) to set the rotation axis to be around **Z** (The up vector in our world).



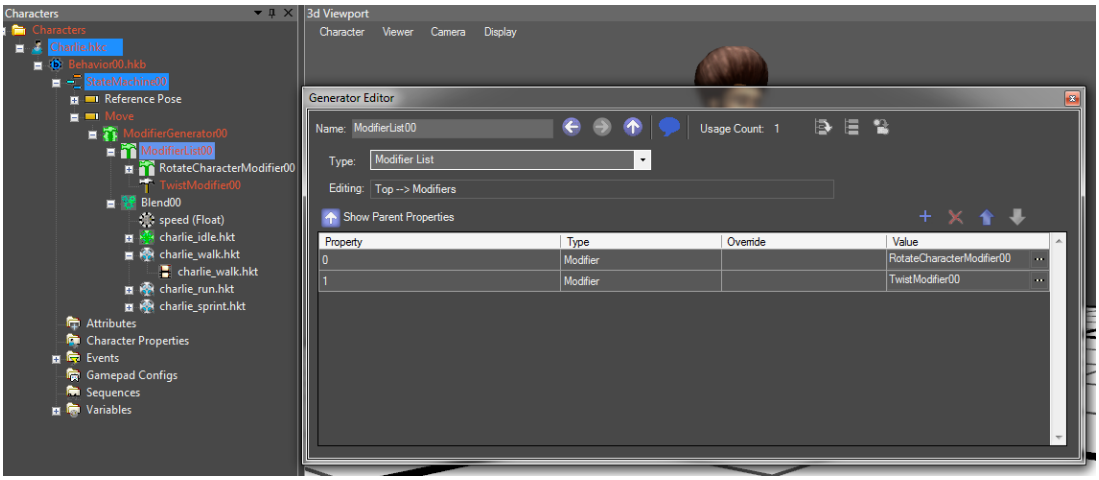
Use the **Variable Editor** to set the properties of ***"turn"*** such that:

Minimum		-1.00
Maximum		1.00
Default Value		0.00

Return to the **Character** tab and with the ***Move*** state node selected run the behaviour (► or *SPACE*) and adjust the ***"turn"*** slider in conjunction with the ***"speed"*** slider to control the characters direction and animation.

Adding a Twist

Re-open the ***ModifierList00*** **Generator Editor**, click on Add + and in the new field open the selection list ▪▪ and select **Twist Modifier.**



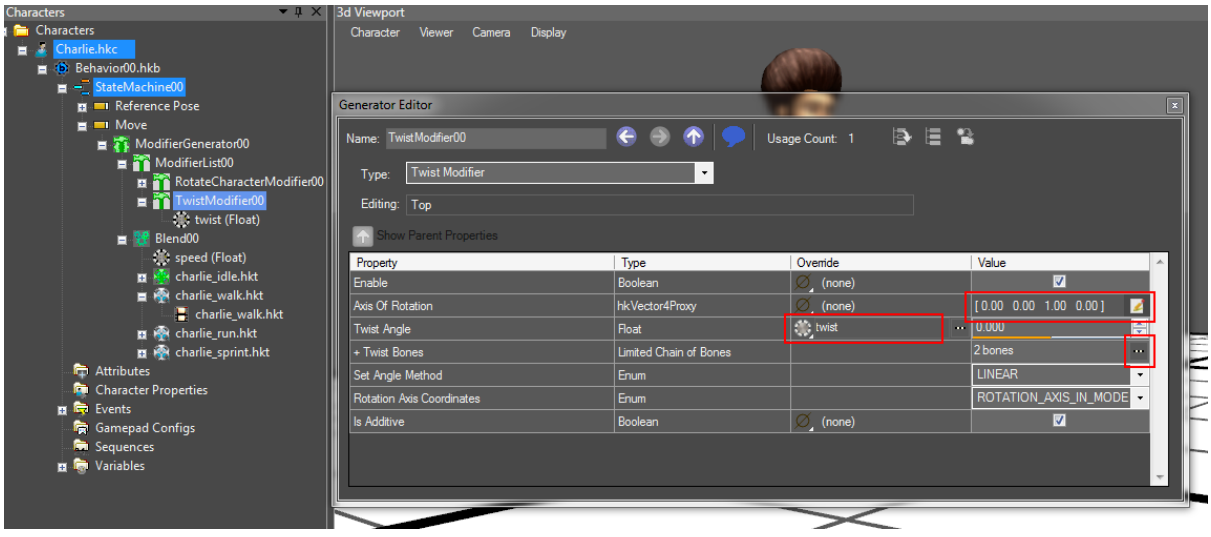
You will note some nodes within the ***Charlie.hkc*** tree structure have turned red. This is due to the **Twist Modifier** having property fields that must be completed before use, in this case the default settings are not sufficient on their own.

Dblclick on the newly created ***TwistModifier00*** to invoke the **Generator Editor** for this node.

As before ensure the **Axis of Rotation** is set to be around the **Z** axis.

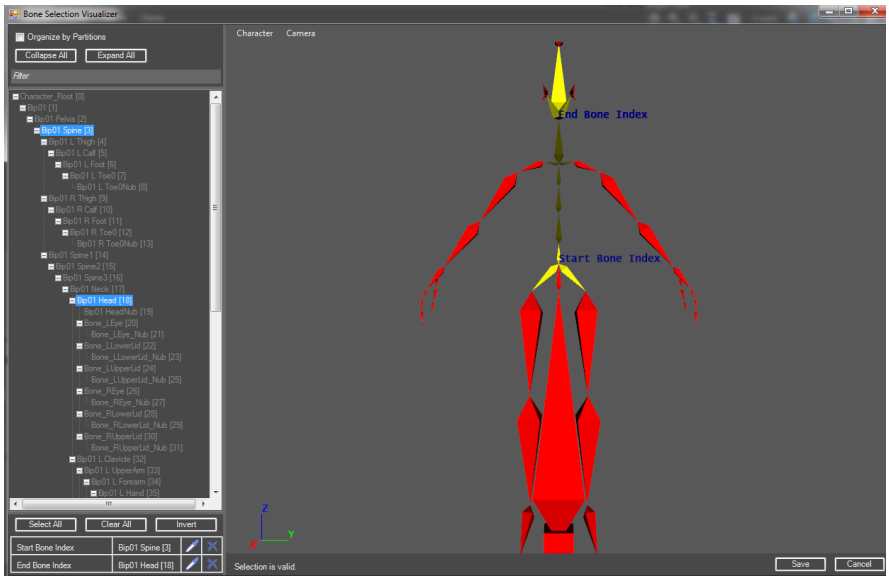
Within the **Twist Angle** field create an **Override** variable called ***"twist"*** the properties of which should be set to:

Minimum		-45.00
Maximum		45.00
Default Value		0.00



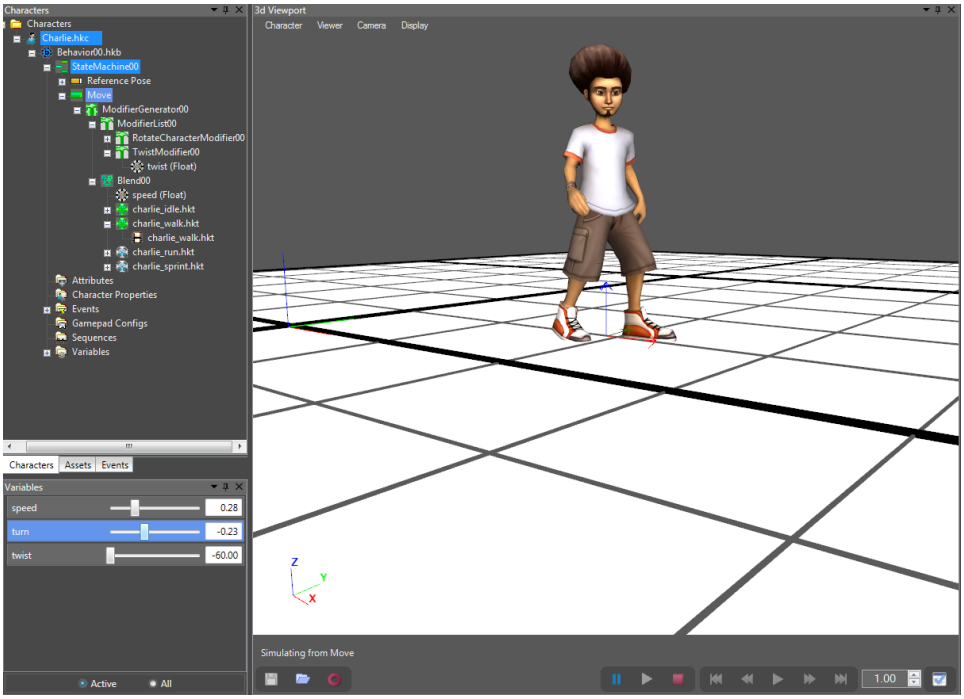
Finally, we need to specify the bones within the skeleton that are to be affected. This is the critical setting that has put the behaviour tree structure in a state of error.

In the **Twist Bones** field, use the ellipse selector ⋯ to display the Bone Selection Visualizer.



Select ***Bip01_Spine*** as the **Start Bone Index** and ***Bip01_Head*** as the **End Bone index**. Press **Save**.

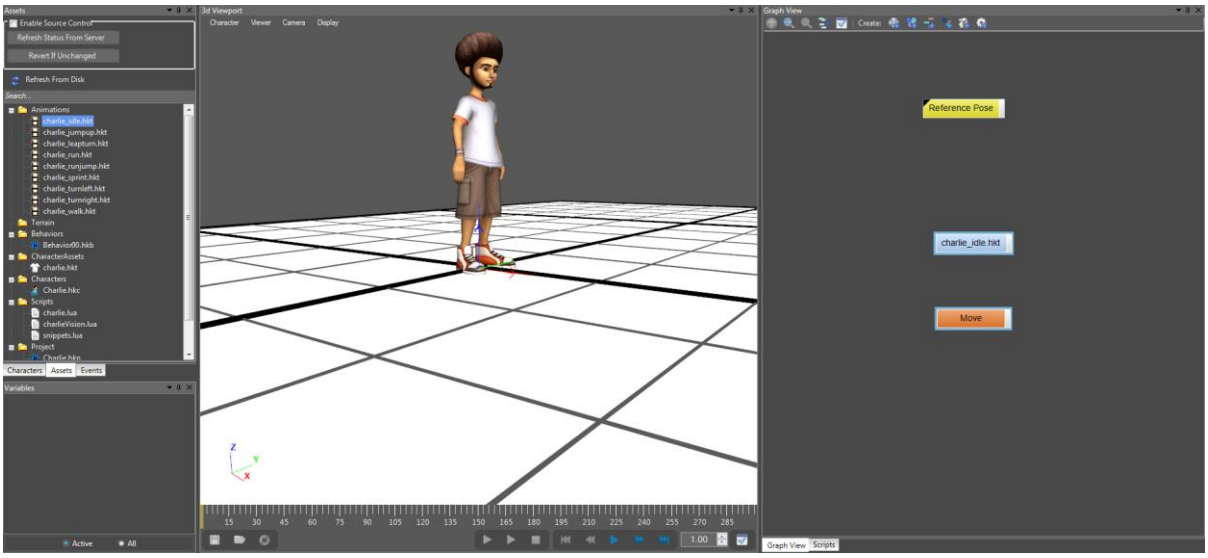
Return to the **Character** tab and if all is well, the tree structure will no longer display nodes in red. Select the ***Move*** state node and run the behaviour (▶ or *SPACE*). Test that manipulating the ***"twist"*** slider causes the character's upper torso to twist about the hip.



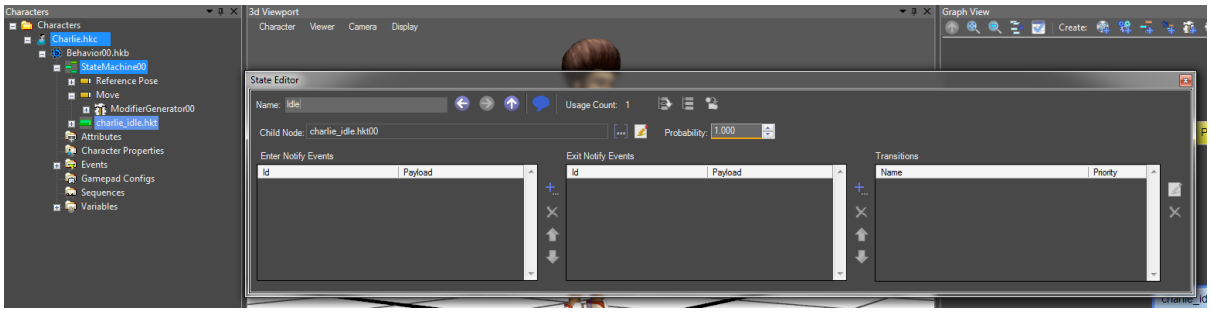
Using a combination of all three sliders we can manipulate and manoeuvre our character about the checked floor.

Adding the Idle State

Now that we have our Move state functioning, we next need to add an Idle state to our state machine. From the **Assets** tab drag the ***charlie_idle.hkt*** into the **Graph View**.

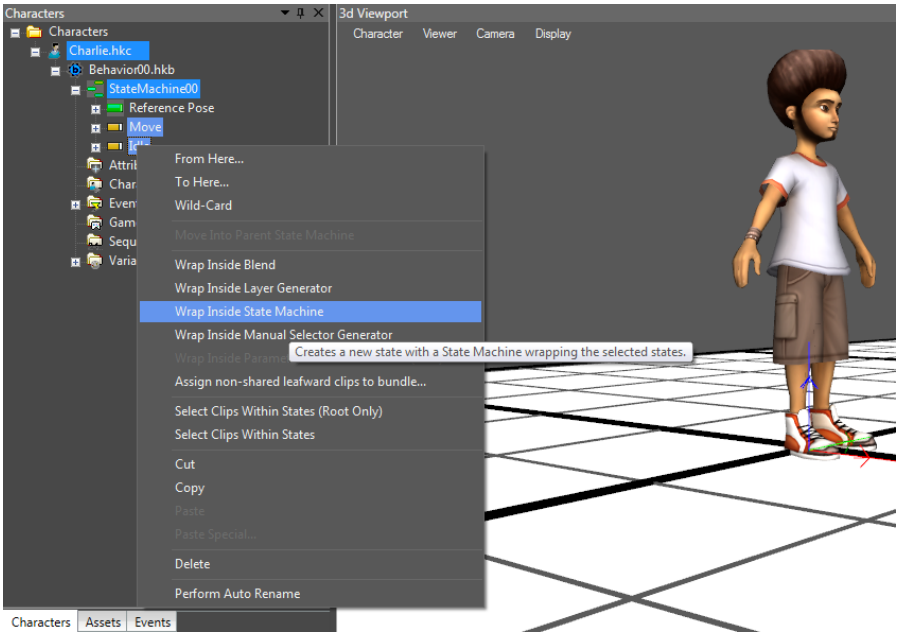


To maintain our naming convention, rename the newly created state to ***"Idle",*** remembering to *RETURN* on the **Name:** field before closing the dialogue.

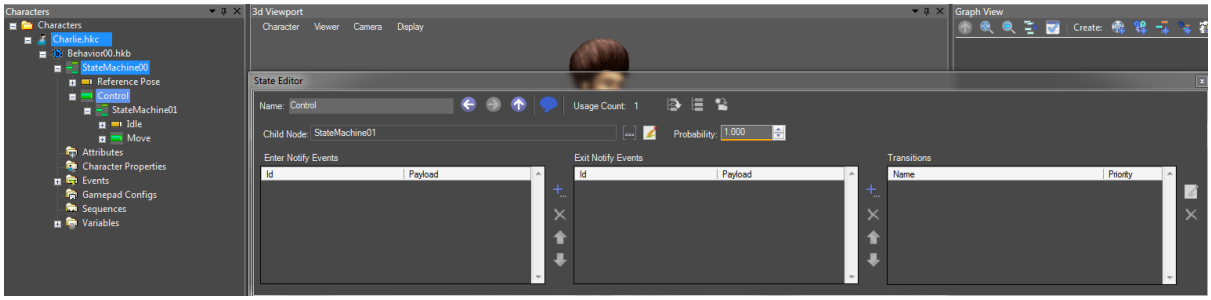


Creating a new State Machine

In order to manage our new states and to facilitate future functionality, we should wrap our new states inside a new State Machine. Select both the ***Idle*** and ***Move*** state nodes and from the *RMB* pop-up menu select **Wrap Inside State Machine**.

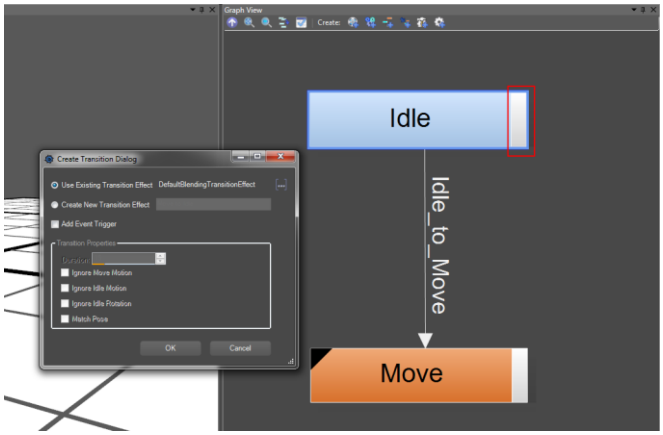


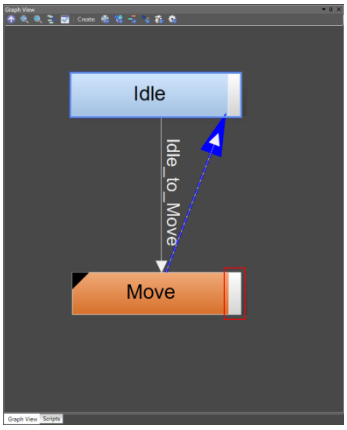
The new state machine also becomes a state node within the Behaviour hierarchy and should be renamed ***"Control"*** for clarity.



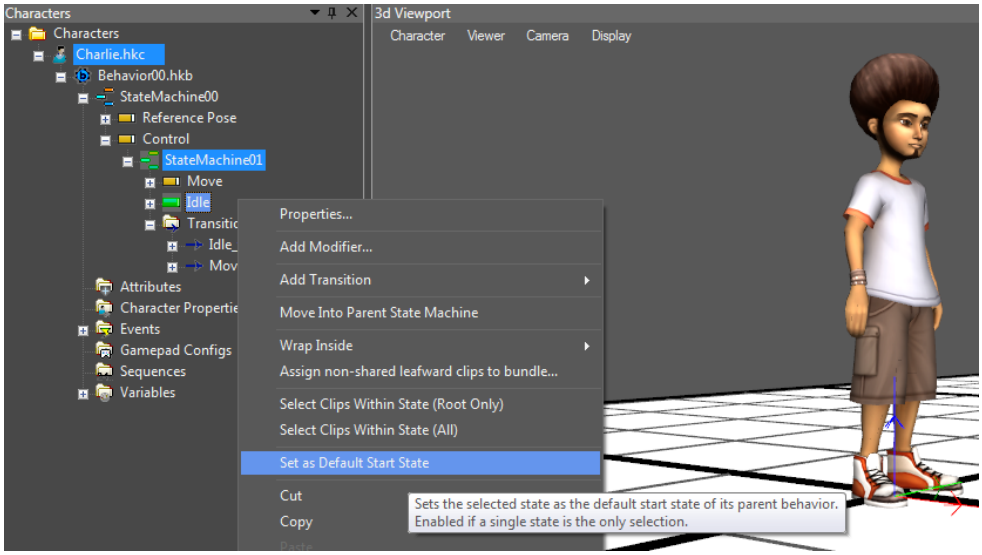
Adding State Transitions

From within the **Graph View**, *LMB* on the white box to the side of the ***Idle*** node and drag into the ***Move*** node. A new transition named ***Idle_to_Move*** will be created and the **Create Transition Dialog** will appear. The default settings will suffice so press OK.



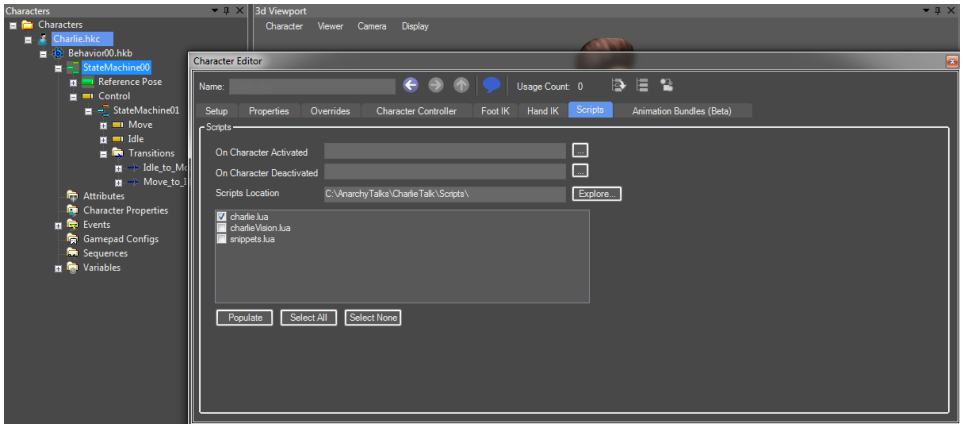


Repeat the process from the ***Move*** node to the ***Idle*** node. Finally we need to specify the default starting node for the State Machine so select the ***Idle*** node and from the *RMB* pop-up menu select **Set as Default Start State**.

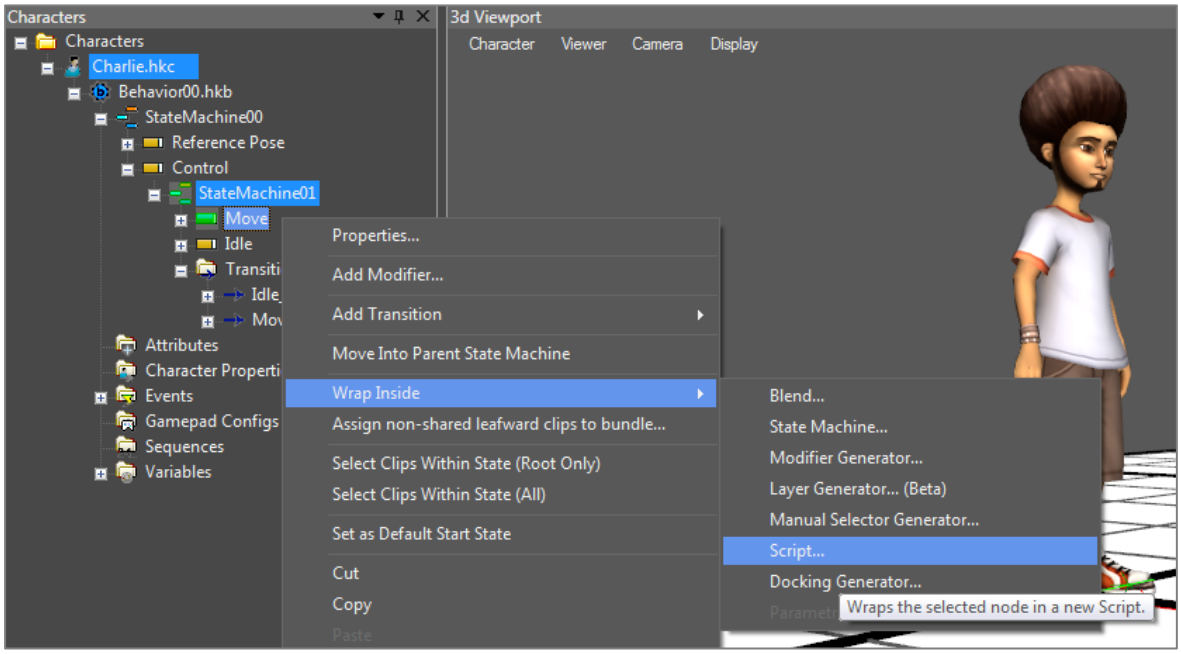


Adding a Control Script

Next we need to add a script to control the state machine and the motion variables. In the **Characters** tab, *Dblclick* on the ***Charlie.hkc*** root node to open the **Character Editor**. Go to the **Scripts** tabs and three *.lua* scripts should be listed, if not click the **Populate** button. Check the ***Charlie.lua*** check box as this is the only script we need.

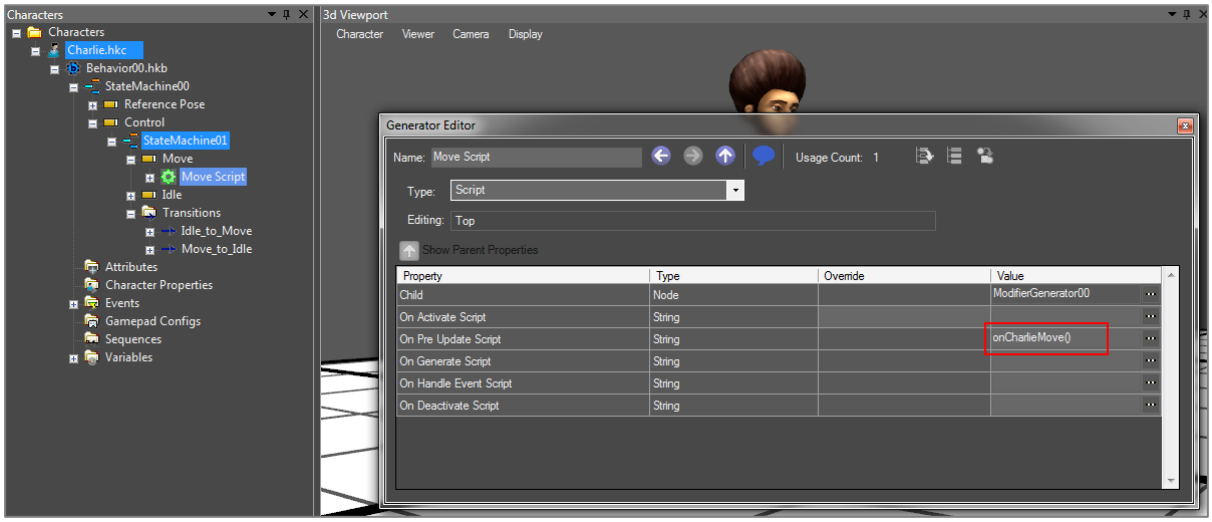


Select the ***Move*** state node and from the *RMB* pop-up menu select **Wrap Inside -> Script...**

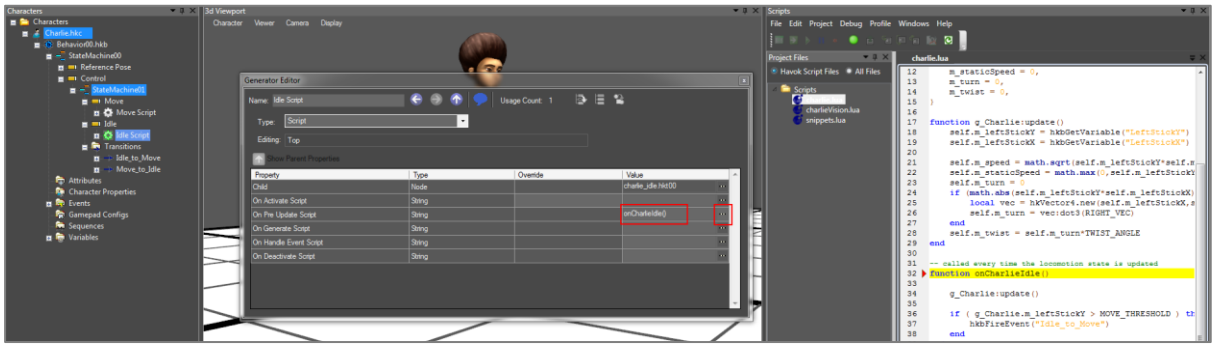


Attaching the Script

Dblclick on the newly created **Move Script** node to invoke the **Generator Editor**. In the **On Pre Update Script** field type ***"onCharlieMove()"*** into the **Value** box



Repeat the process for the ***Idle*** state node but this time set the function as ***"onCharlieIdle()"***.



Tip. To verify the function attachment click on the ellipse ▪▪ and the corresponding function should be highlighted in the **Scripts** view.

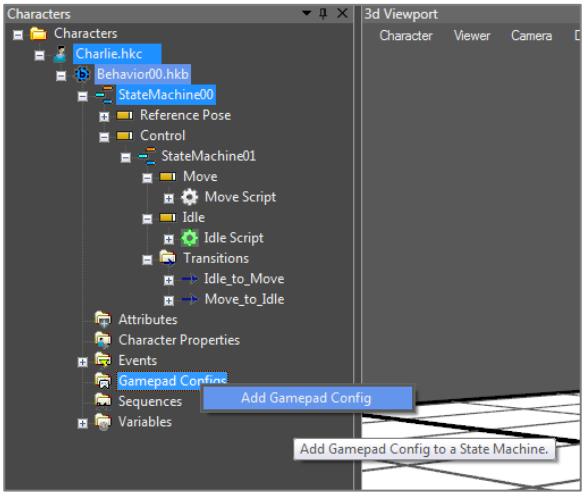
The scripts are now attached and functional, but we now need to define a user input to drive them..

Connecting a Control Pad

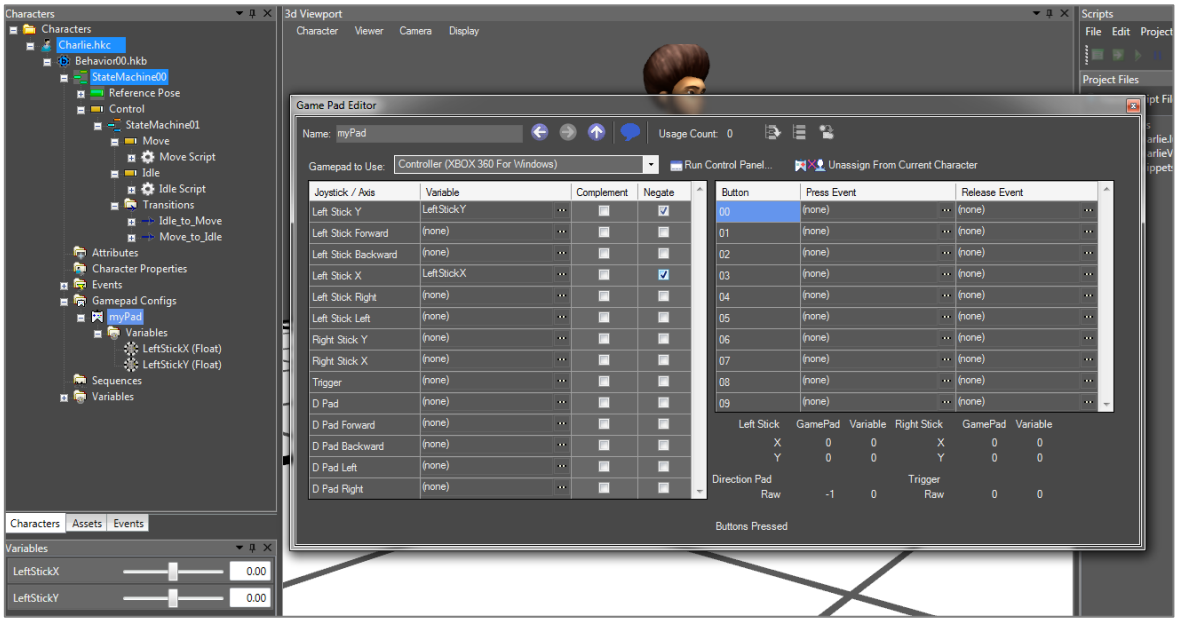
In the **Characters** tab *RMB* on the **Gamepad Configs** node and select **Add Gamepad Config**.

A new **Gamepad Config** node will be crated inviting a name to be given. In this case we can label it ***"myPad"***.

Dblclick on the new ***myPad*** node to open the **Game Pad Editor**.



In the **Left Stick Y** field create a new variable called ***"LeftStickY"*** and in the **Left Stick X** field a variable called ***"LeftStickX".*** For both fields check the **Negate** check box to invert the controls.

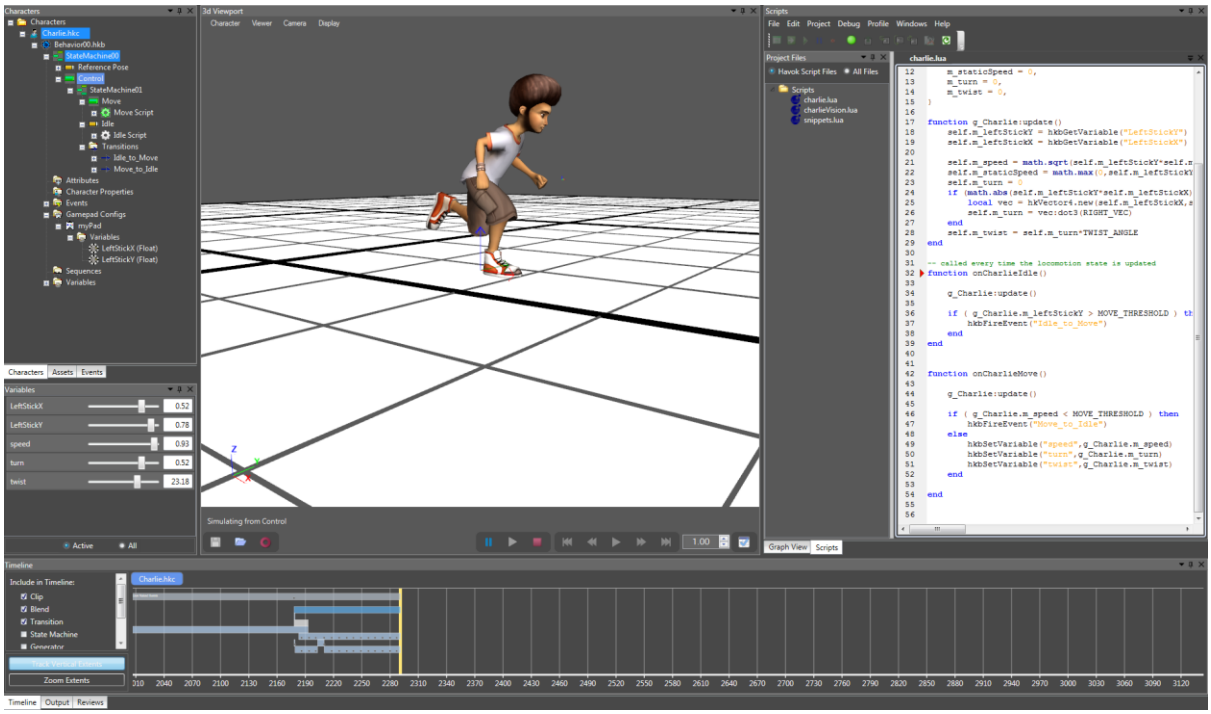


Using the **Variable Editor** to set the properties of ***"LeftStickY"*** and ***"LeftStickX"*** such that:

Minimum		-1.00
Maximum		1.00
Default Value		0.00

Select the ***Control*** state node and run the behaviour (▶ or *SPACE*) using the attached gamepad.

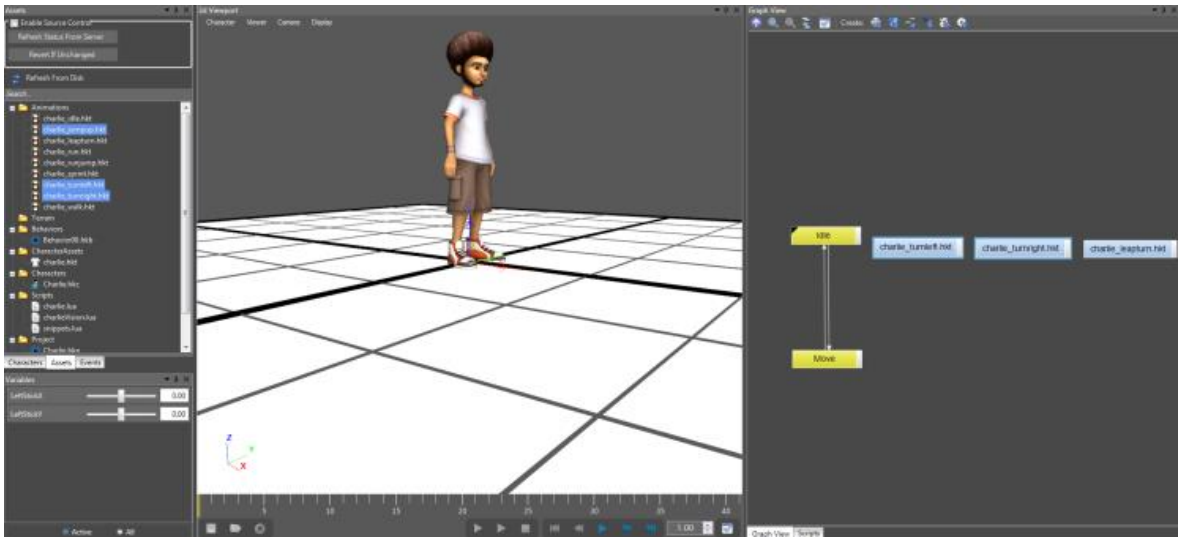
If all is well the character should switch back and forth between the **Idle** and **Move** states with the ***"speed"***, ***"turn"*** and ***"twist"*** variables dynamically changing value allowing full control of the character.



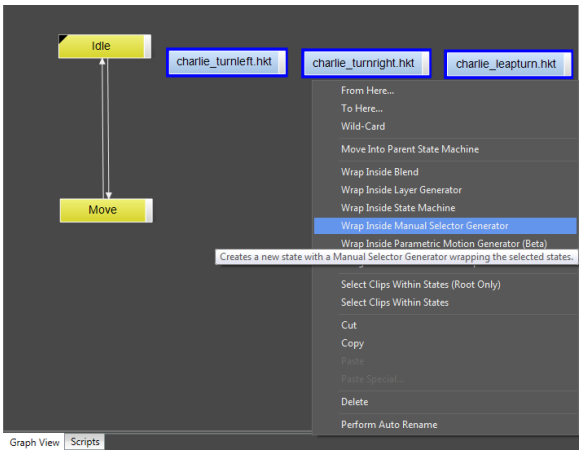
Adding Static Turns

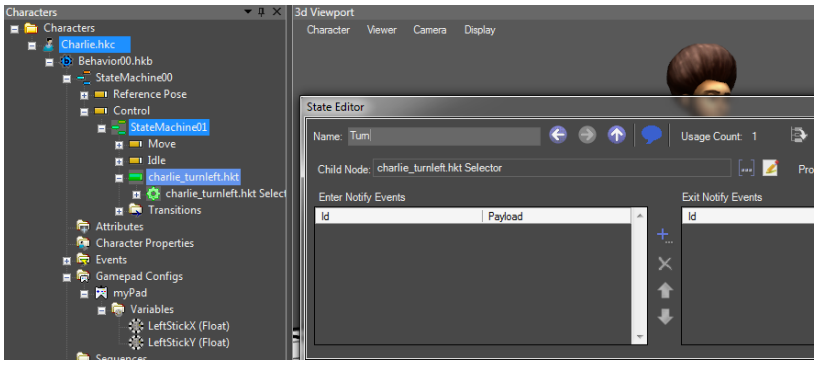
Next we want to create a new state that allows the character to turn around when not moving.

From the **Assets** tab select and drag ***charlie_turnleft.hkt***, ***charlie_turnright.hkt*** and ***charlie_leapturn.hkt*** into the **Graph View**.



Still in the **Graph View** select the three new animation clips and with the *RMB* pop-up menu select **Wrap Inside Manual Select Generator**.

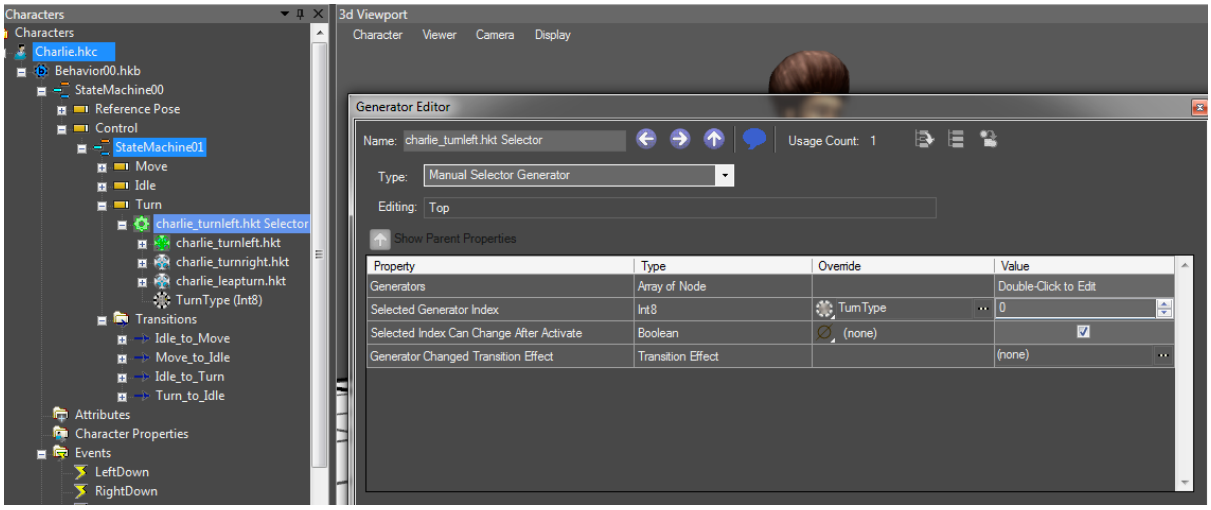




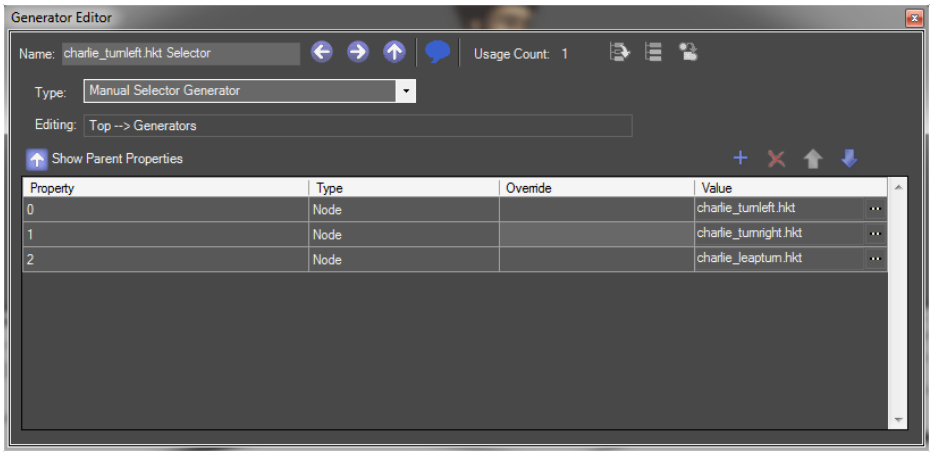
And in the **Character** tab use the **State Editor** to rename the newly created state to ***"Turn"***.

Configuring the Manual Selector

In the Character tab Dblclick the new **charlie_turnleft.hkt Selector** node to bring up the Generator Editor.



In the **Selected Generator Index** field create a new **Override** variable called ***"TurnType"***.

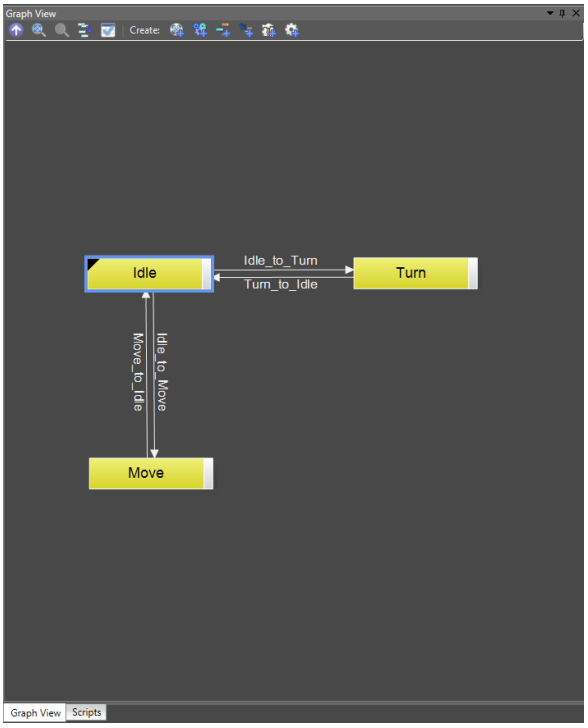


Dblclick on the **Double-Click to Edit** box and ensure the three animation clips are ordered as show above. If they are not use the arrow icons (⇧⇩) to reposition the entries in the list. It is important the correct value in the **Property** fields match the correct animation clips.

Use the Variable Editor to set the properties of ***"TurnType"*** as follows:

Minimum		0
Maximum		2
Default Value		0

Finally in the **Graph View**, LMB on the white box to the side of the ***Idle*** node and drag into the ***Turn*** node creating an ***Idle_to_Turn*** transition event….

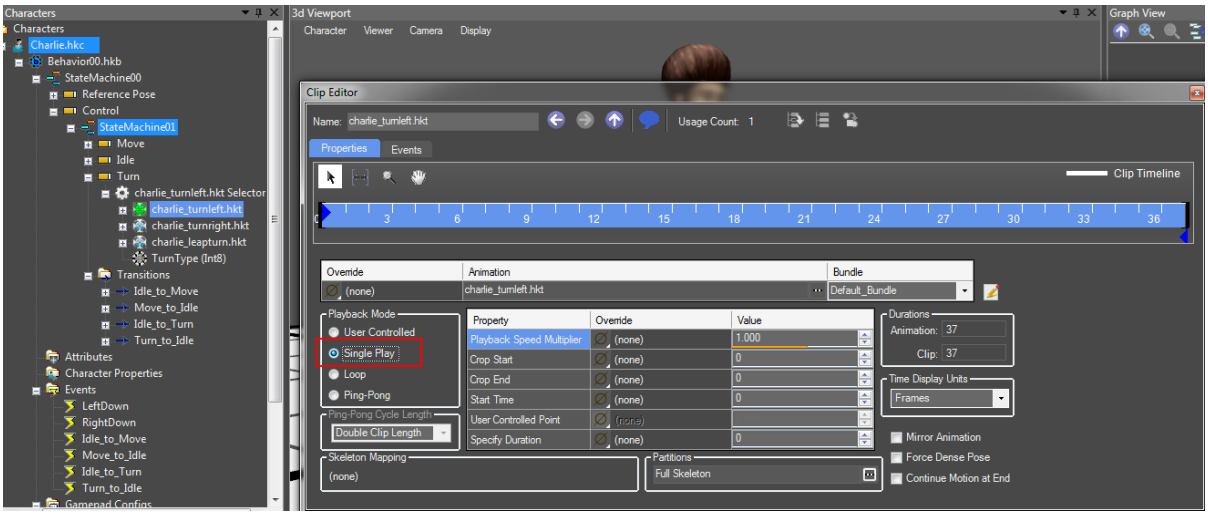


…and LMB on the white box to the side of the ***Turn*** node and drag into the ***Idle*** node creating an ***Turn_to_Idle*** transition event.

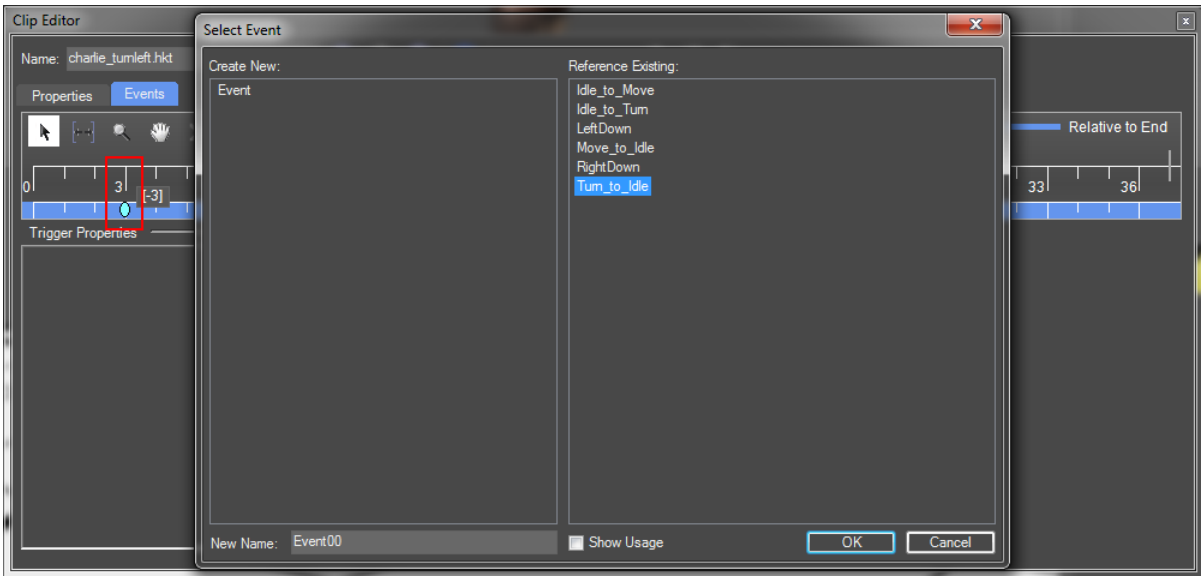
Configuring the Turn Animations

Unlike the motion and idling animations, we need to configure the turn animations to only play the once then return back to the ***Idle*** state.

Dblclick on the ***charlie_turnleft.hkt*** node to open the **Clip Editor**. In the **Playback** Mode field box, set **Single Play**.



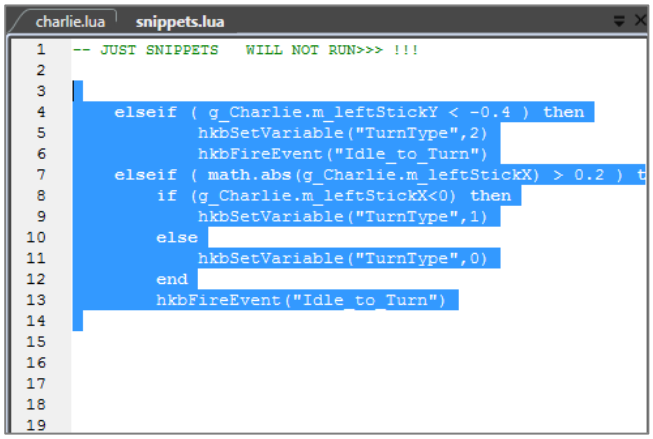
Still in the **Clip Editor**, select the **Events** tab and *LMB* on the blue **Relative to End** track just under the ***3***. An event marker will be placed in the track and the **Select Event** dialogue will open. Select the ***Turn_to_Idle*** event and press **OK**.



Repeat this process for the ***charlie_turnright.hkt*** and ***charlie_leapturn.hkt*** nodes.

Scripting the Static Turns

All that is needed now to finish implementing the Static Turns is a way of controlling when and how they are triggered.



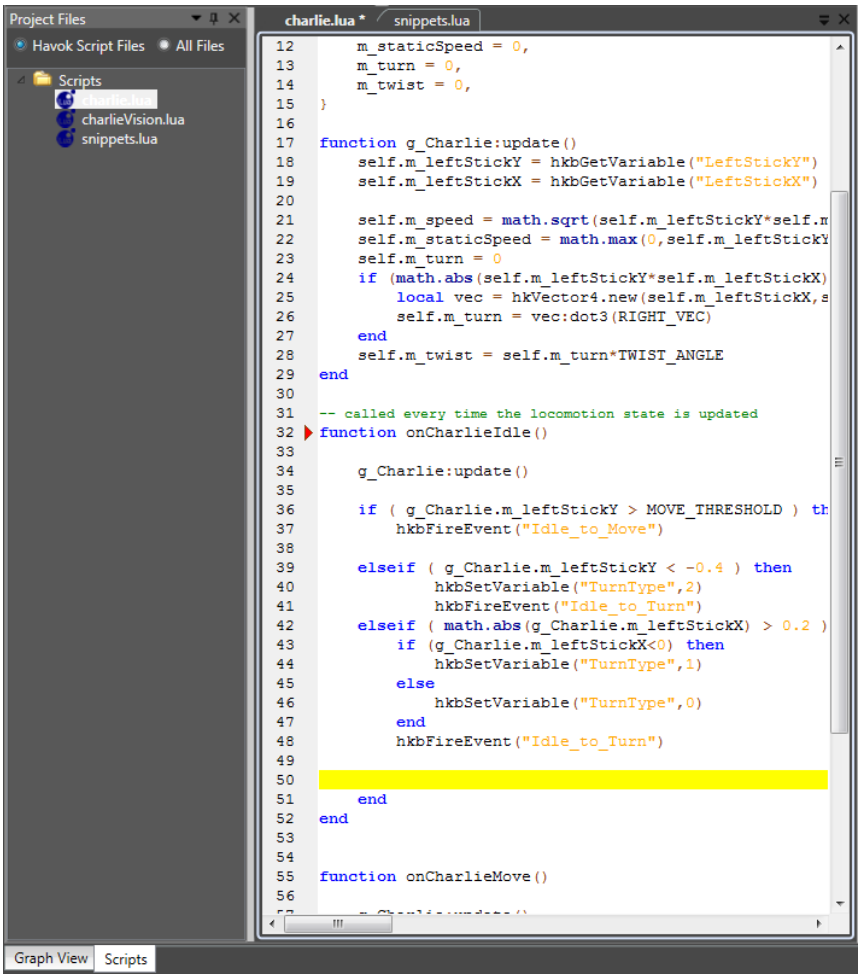
Go to the **Scripts View** and open the ***"snippets.lua"*** file and select and copy the first block of code.

Then paste into the ***"charlie.lua"*** file at line ***38*** after:

```
hkbFireEvent("Idle_to_Move")
```

Save the changes. (***File->Save File***)

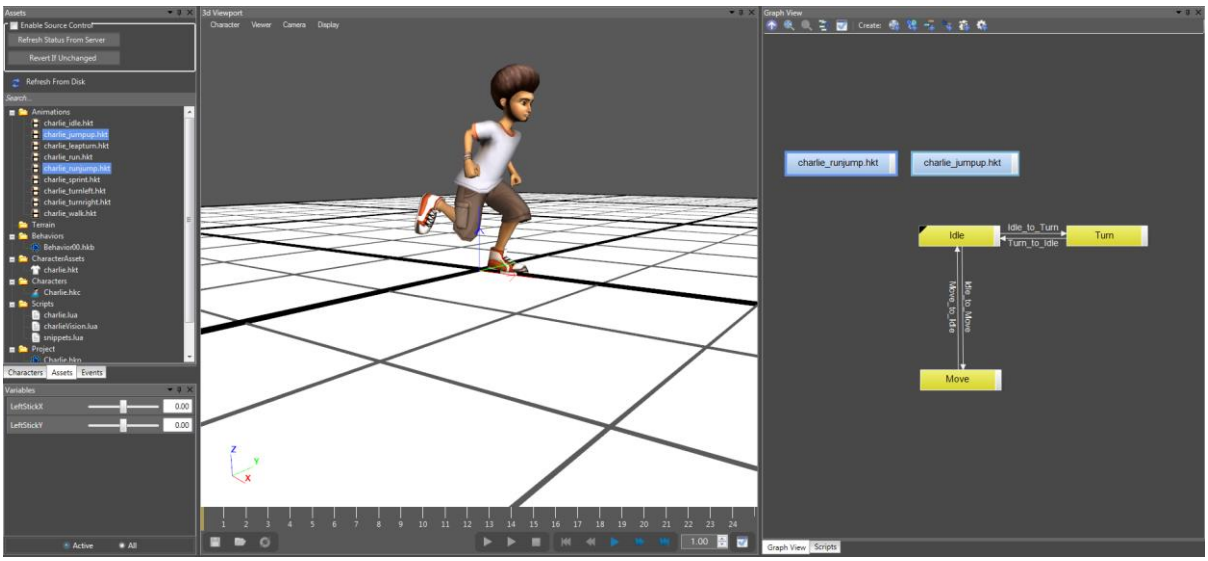
Select the ***Control*** state node and run the behaviour (▶ or *SPACE*) using the attached gamepad. Pushing the left thumb stick left or right should cause our character to perform a left or right turn on the spot. Pulling the stick back causes the character to jump and turn 180° in the air.

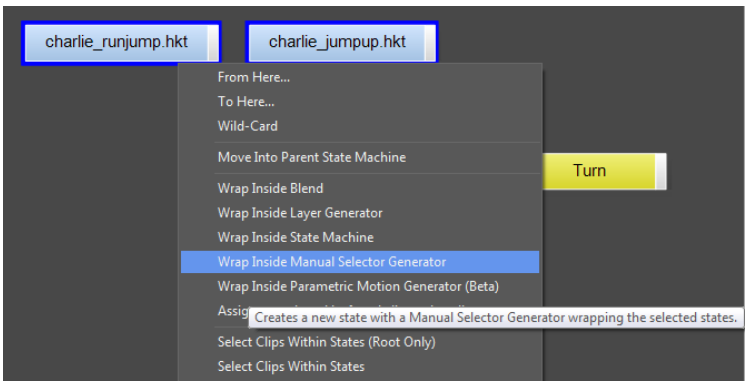


Adding the Jumps

The final state we want to add is a dynamic jump that can be triggered at any time and correctly play the appropriate animation and automatically return to the correct state.

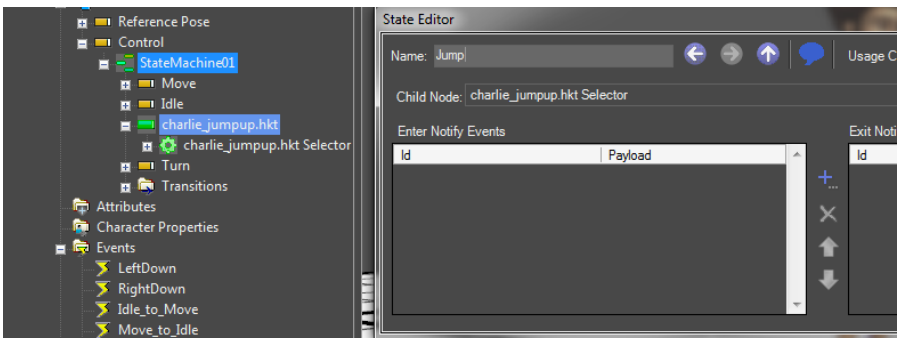
As we did for the turn animations, in the **Assets** tab select and drag ***charlie_jumpup.hkt*** and ***charlie_runjump.hkt*** into the **Graph View**.





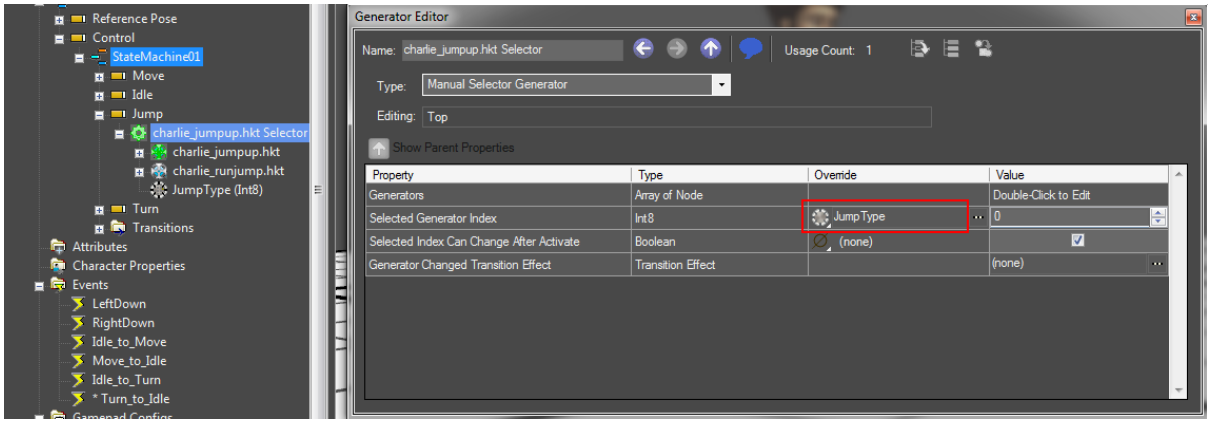
Still in the **Graph View** select the two new animation clips and with the *RMB* pop-up menu select **Wrap Inside Manual Select Generator**.

And in the **Character** tab use the **State Editor** to rename the newly created state to ***"Jump"***.

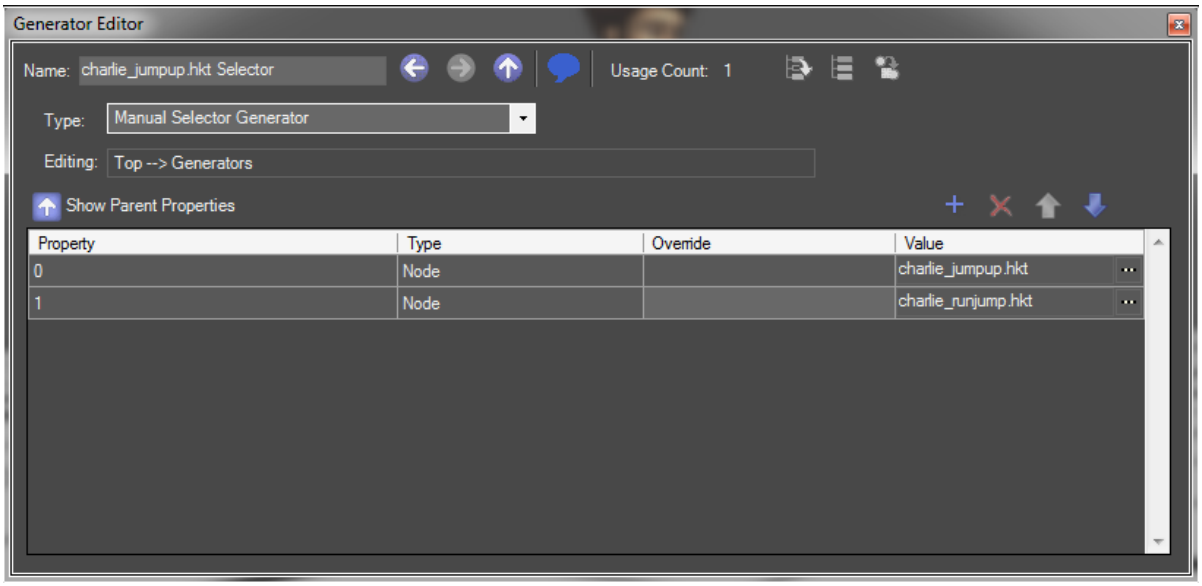


Configuring the Jump Selector

In the Character tab *Dblclick* the new **charlie_jump.hkt Selector** node to bring up the **Generator Editor**.



In the **Selected Generator Index** field create a new **Override** variable called ***"JumpType"***.

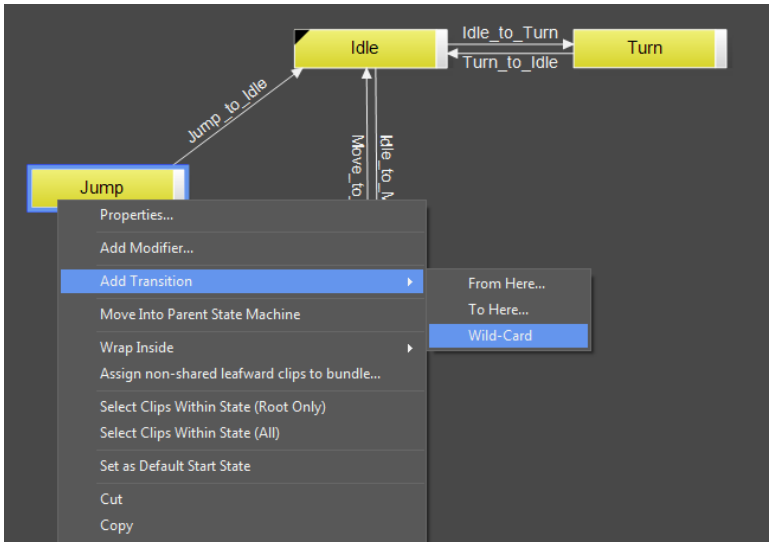


Dblclick on the **Double-Click to Edit** box and ensure the two animation clips are ordered as show above. If they are not use the arrow icons (⇧⇩) to reposition the entries in the list. It is important the correct value in the **Property** fields match the correct animation clips.

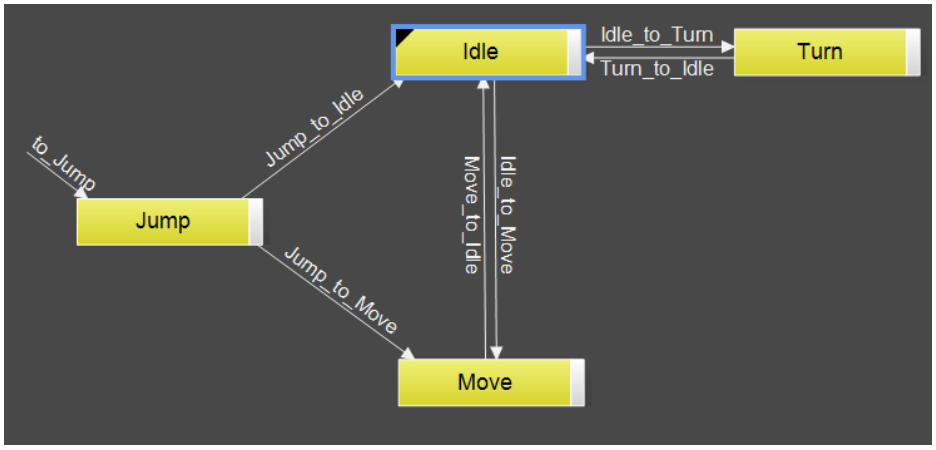
Use the **Variable Editor** to set the properties of ***"JumpType"*** as follows:

Minimum		0
Maximum		1
Default Value		0

Next in the **Graph View**, *LMB* on the white box on the side of the ***Jump*** node and drag into the ***Idle*** node creating a ***Jump_to_Idle*** transition event. Then again from the ***Jump*** node to the ***Move*** node creating a ***Jump_to_Move*** event.



And finally *RMB* on the ***Jump*** node and from the pop-up menu select **Add Transition -> Wild-Card**.

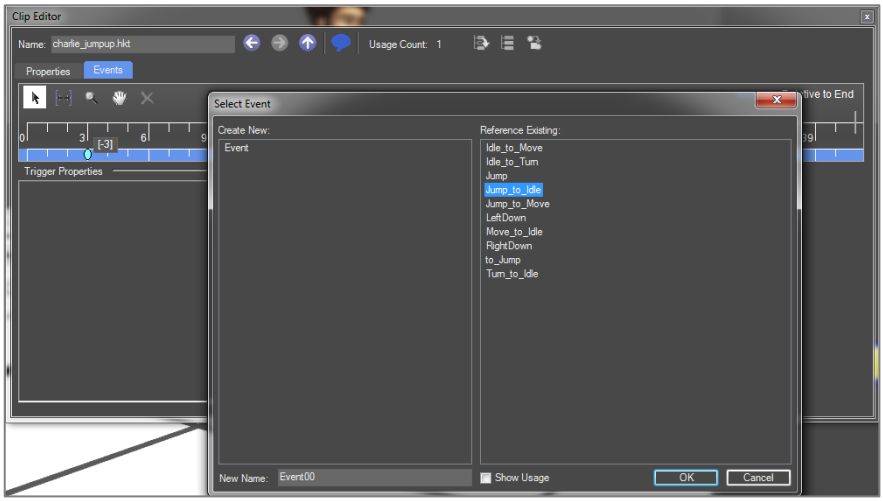


A new transition event called ***to_Jump*** will be automatically created.

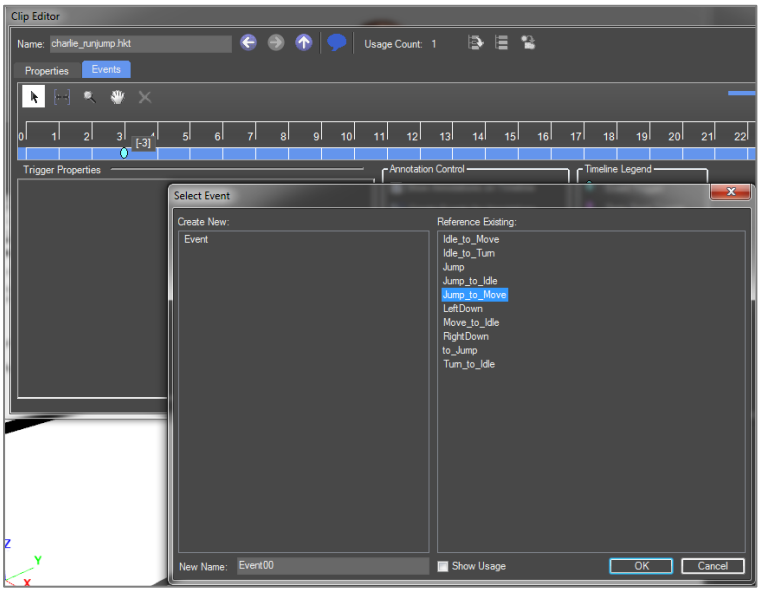
Configuring the Jump Animations

Just as we did for the turn animations we need to set the jump animations to play once and cause the return to the correct state.

Dblclick on the ***charlie_jumpup.hkt*** node to open the **Clip Editor**. In the **Playback** Mode field box, set **Single Play**. Then in the Events tab *LMB* on the blue **Relative to End** track just under the ***3***. An event marker will be placed in the track and the **Select Event** dialogue will open. Select the ***Jump_to_Idle*** event and press **OK**.

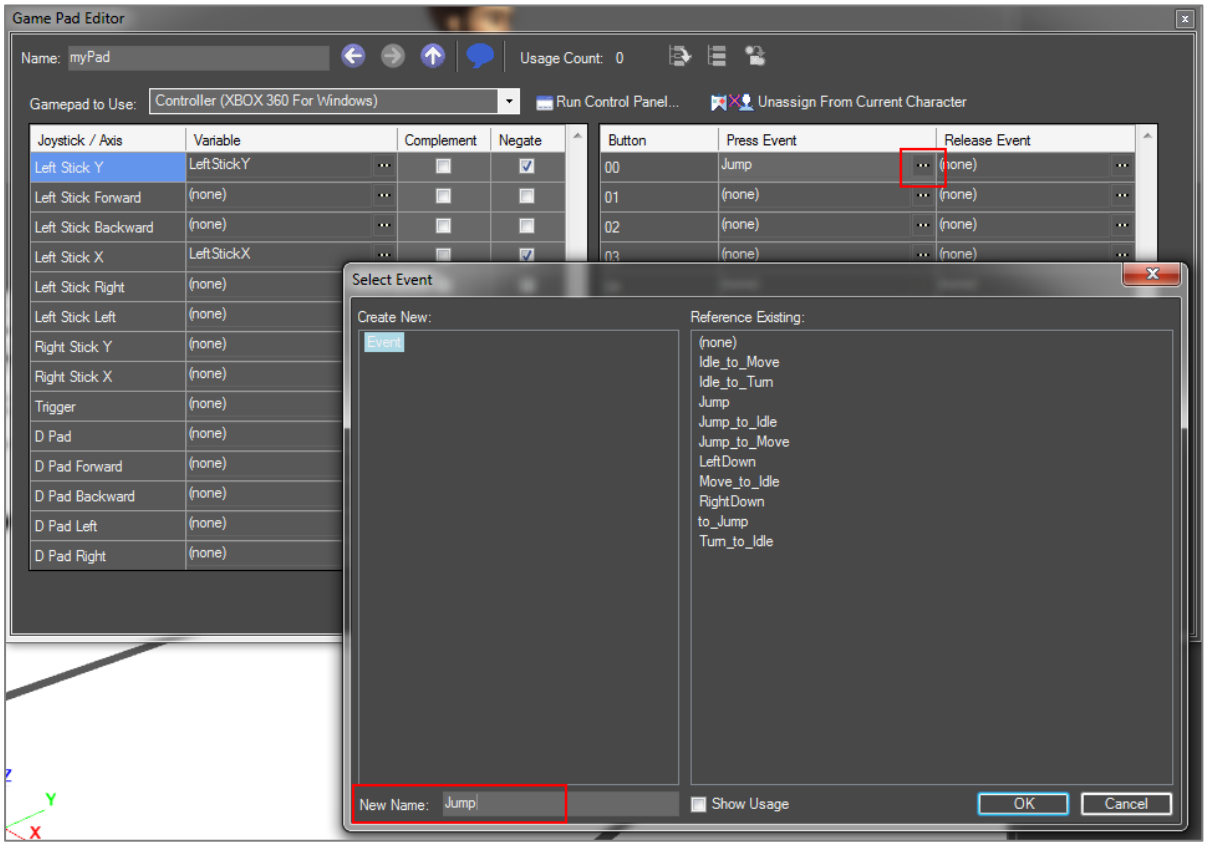


Repeat for the ***charlie_runjump.hkt*** node except this time set the event to ***Jump_to_Move***.



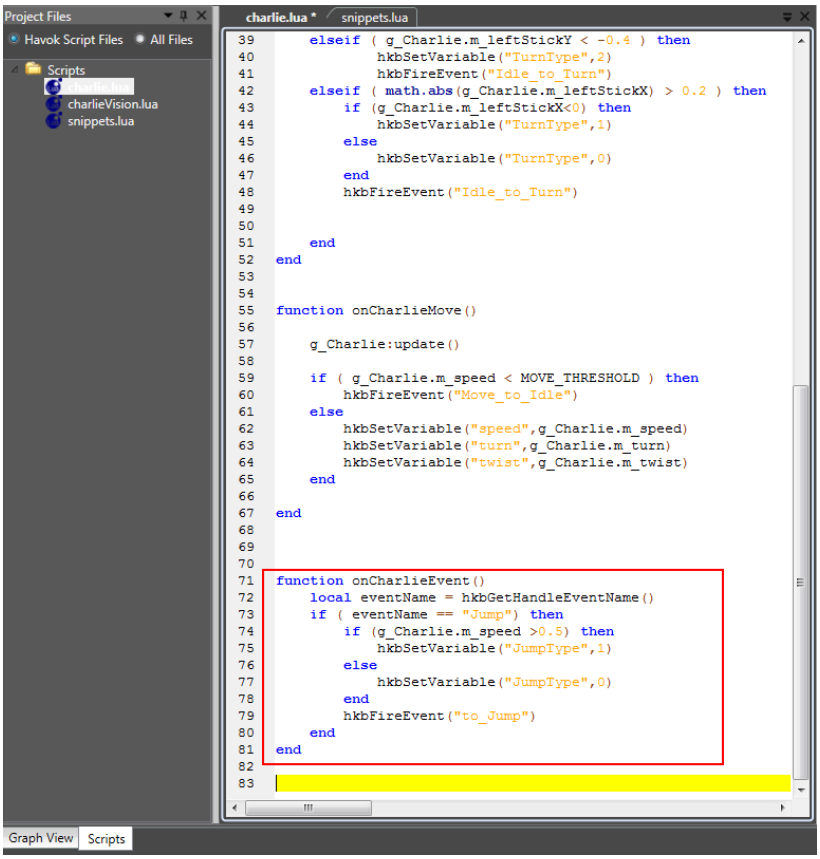
Adding a Jump Button

In the **Characters** tab *Dblclick* on the ***myPad*** gamepad configuration to bring up the Game Pad Editor. In the fields for **Button 00** create a new **Press Event** by clicking on the ellipse ·· to open the **Select Event** dialogue. In the **New Name:** field type ***"Jump"*** and press **OK**.



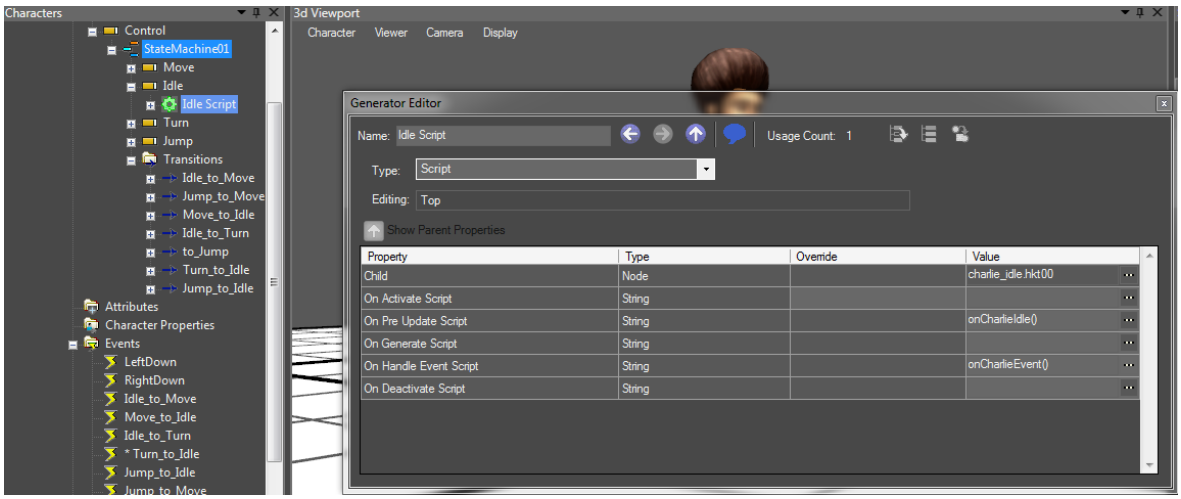
Scripting the Jump

In the **Scripts View** open the ***"snippets.lua"*** file and copy the whole of the `onCharlieEvent()` function and paste into the end of the ***"charlie.lua"*** file. Save the changes.




```
39      elseif ( g_Charlie.m_leftStickY < -0.4 ) then
40              hkbSetVariable("TurnType",2)
41              hkbFireEvent("Idle_to_Turn")
42      elseif ( math.abs(g_Charlie.m_leftStickX) > 0.2 ) then
43          if (g_Charlie.m_leftStickX<0) then
44              hkbSetVariable("TurnType",1)
45          else
46              hkbSetVariable("TurnType",0)
47          end
48          hkbFireEvent("Idle_to_Turn")
49
50
51      end
52  end
53
54
55  function onCharlieMove()
56
57      g_Charlie:update()
58
59      if ( g_Charlie.m_speed < MOVE_THRESHOLD ) then
60          hkbFireEvent("Move_to_Idle")
61      else
62          hkbSetVariable("speed",g_Charlie.m_speed)
63          hkbSetVariable("turn",g_Charlie.m_turn)
64          hkbSetVariable("twist",g_Charlie.m_twist)
65      end
66
67  end
68
69
70
71  function onCharlieEvent()
72      local eventName = hkbGetHandleEventName()
73      if ( eventName == "Jump") then
74          if (g_Charlie.m_speed >0.5) then
75              hkbSetVariable("JumpType",1)
76          else
77              hkbSetVariable("JumpType",0)
78          end
79          hkbFireEvent("to_Jump")
80      end
81  end
82
83
```

In the Characters tab *Dblclick* on the ***Idle Script*** node to invoke the **Generator Editor**. In the **On Handle Event Script** field type ***"onCharlieEvent()"*** into the **Value** box.



Repeat the process for the ***Move Script*** node.

To test the jump functionality select the ***Control*** state node and run the behaviour (▶ or *SPACE*) using the attached gamepad. Press the ***"Jump"*** button (using the Xbox 360 controller button **00** maps to A) when idling, walking or running and note the automatic selection of the two jump types and return to corresponding state on completion.

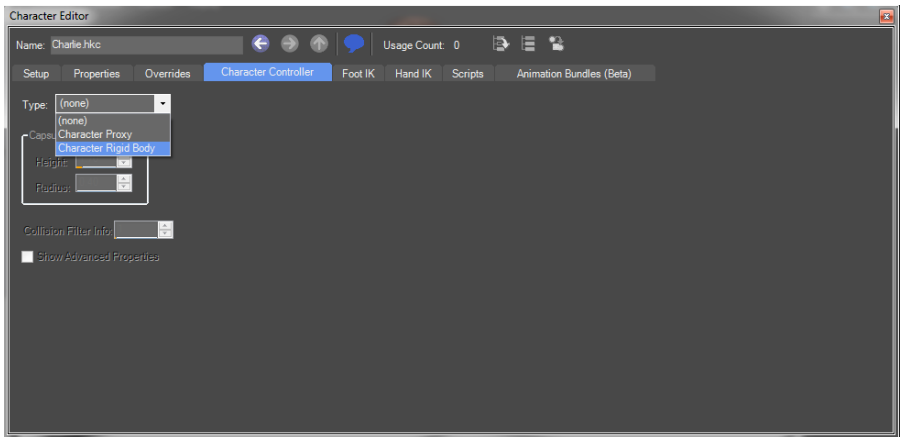
Giving the Character a Physical Presence

The Charlie character's behaviour should now be complete within the specifications of this demo. All that remains to do is to prepare for use outside of the Havok Animation Studio.

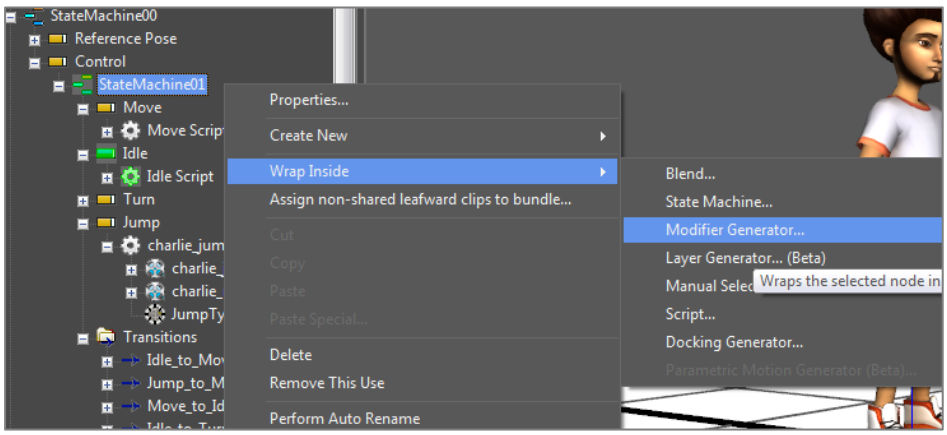
To simplify integration of our character into an actual 3d world, we will attach a physics object representation to our character and associate it to a basic controller that provides terrain following, collision and gravity….

Adding the Character Controller

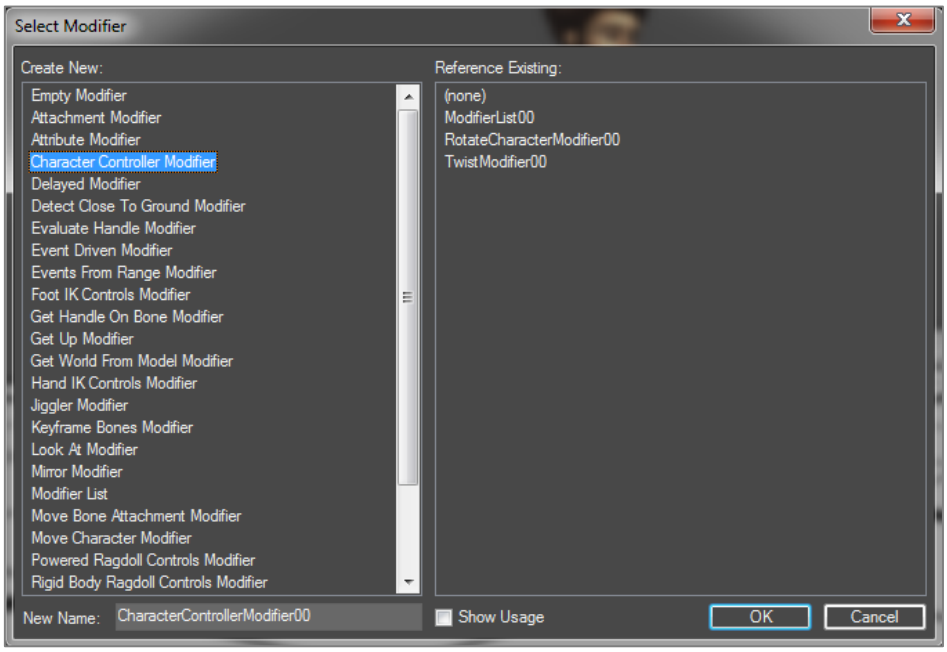
In the **Characters** tab, *Dblclick* on the ***Charlie.hkc*** root node to open the **Character Editor**. Go to the **Character Controller** tab and from the **Type** pull-down box select **Character Rigid Body**.



Return to the **Characters** tab *RMB* on the ***StateMachine01*** node and from the pop-up menu select **Wrap Inside -> Modifier Generator…**



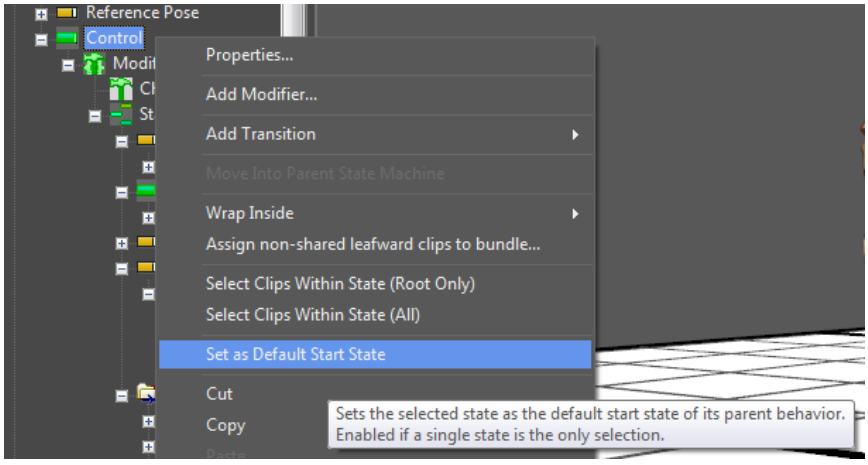
Then from the **Select Modifier** dialogue, select the **Character Controller Modifier**.



Last but not Least

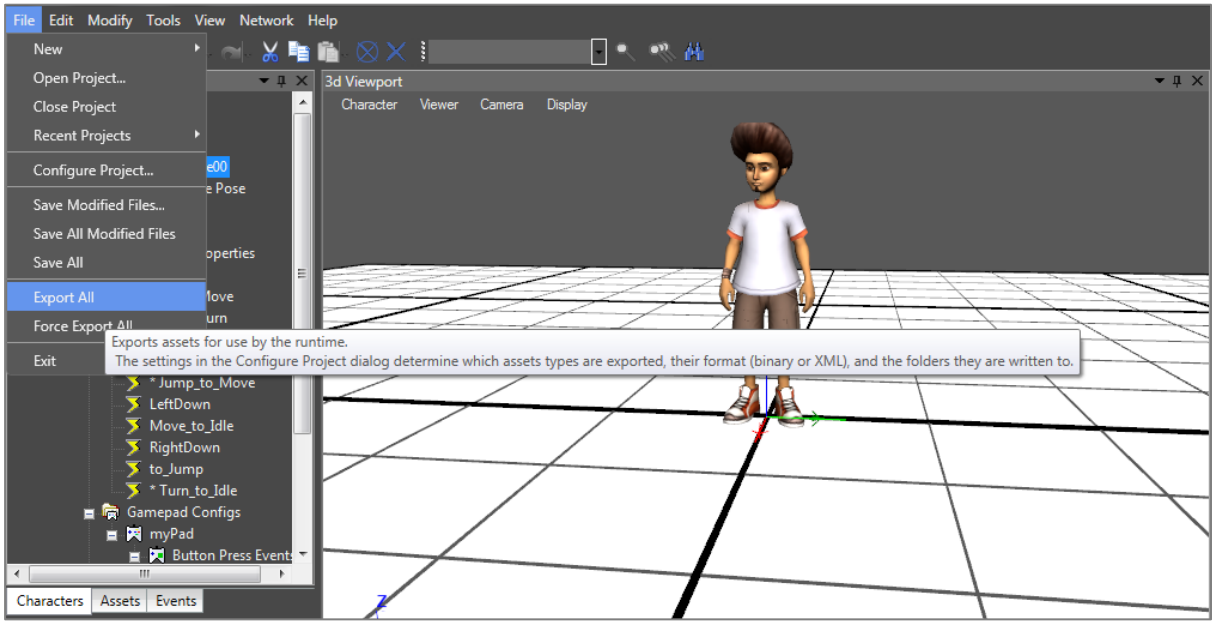
A critical but easily overlooked task is to specify how the behaviour is to start from outside of the *Havok Animation Studio* tool.

In the **Characters** tab select the ***Control*** state node and with the *RMB* pop-up menu select **Set as Default Start State**. Failure to do so results in the exported character reverting to the **Reference Pose** state.



Everything is now ready for the character to be exported from the tool for inclusion in other applications.

Exporting the Behaviour

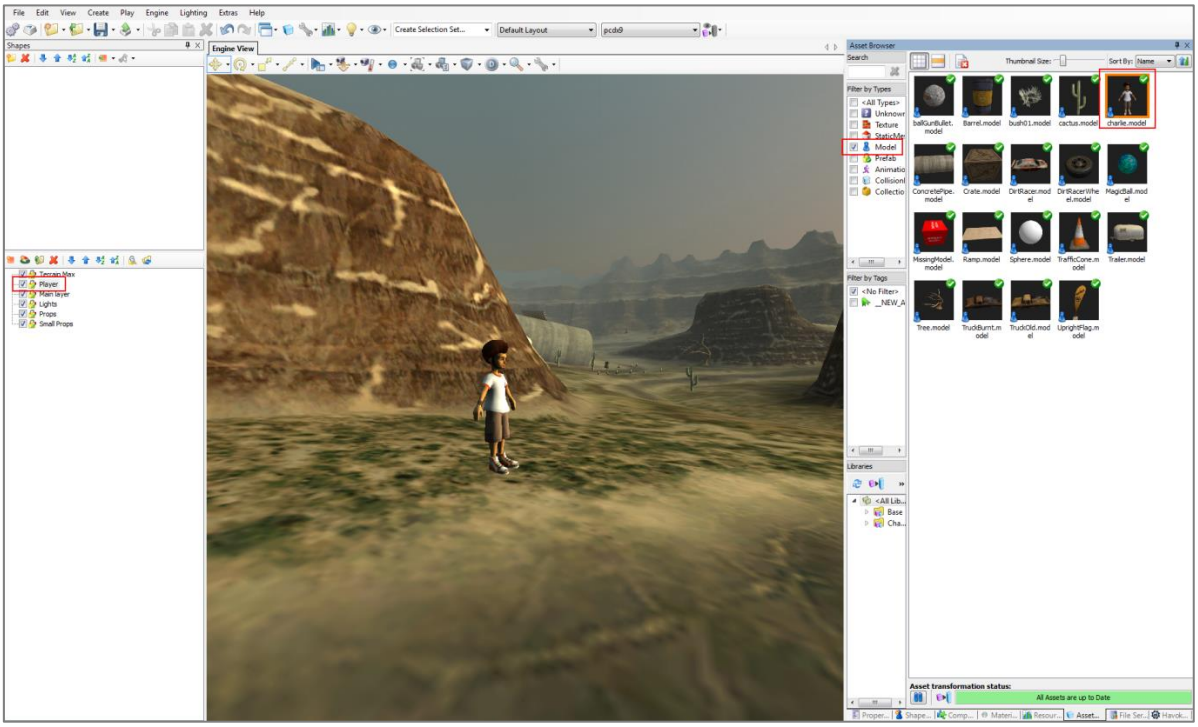


Exporting the character is as simple as selecting **File -> Export** All from the application menu bar. See the **Configure Project** dialogue for more detailed export settings.

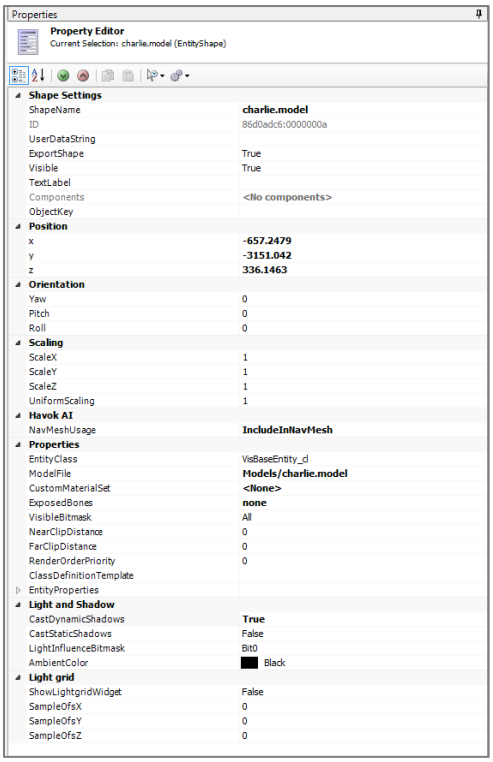
Adding the Playable Character into Vision

Open *vForge* and load the ***"CharliesWorld"*** scene from the ***"CharlieTalk"*** project.

Starting with the **Default Layout**, select the **Player** layer and open the **Asset Browser** tab with Model checked in the **Filter by Types**. Locate a suitable starting location for the Charlie character then drag the ***"Charlie.model"*** into that location.

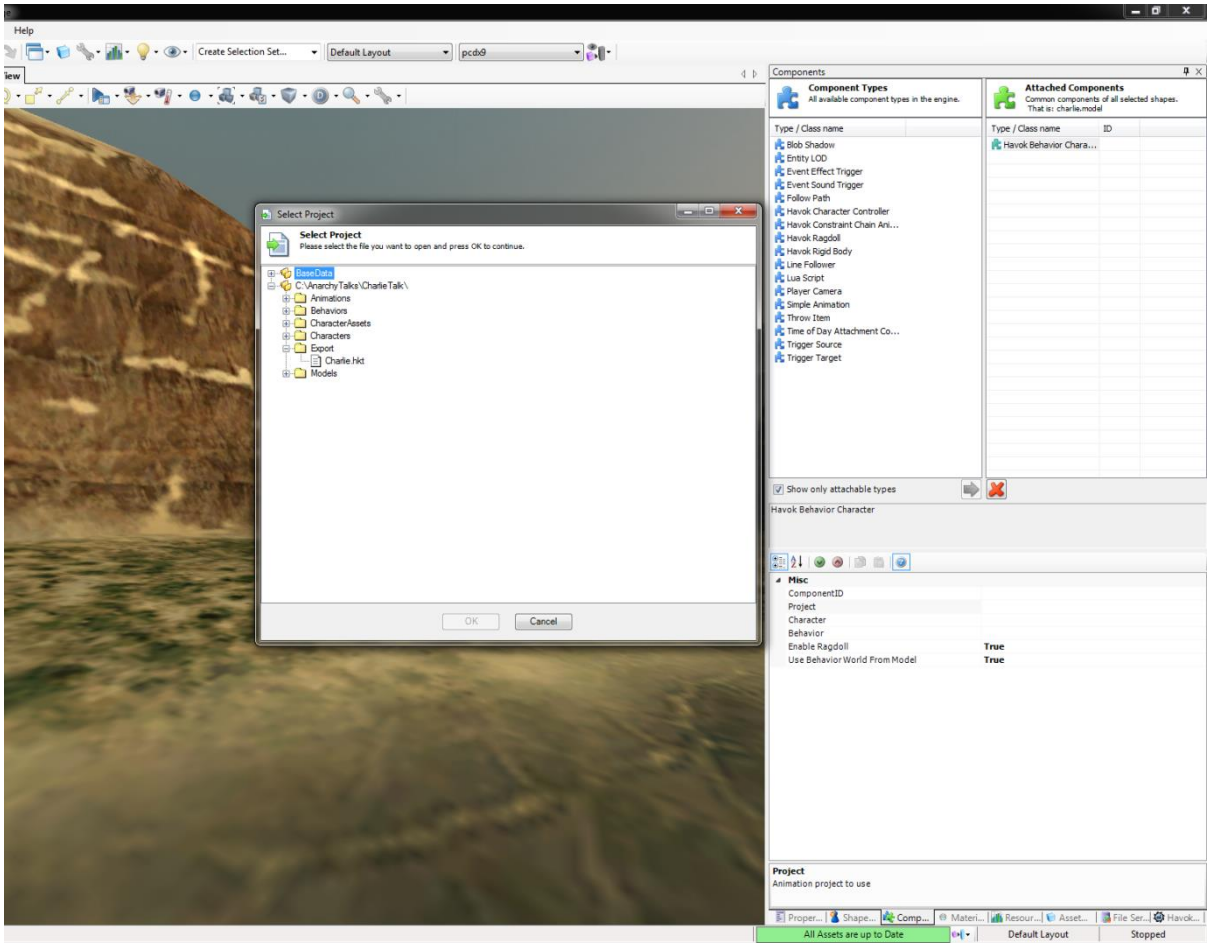


Before progressing check the model is *'grounded'* onto the world mesh and that dynamic shadowing is enabled by checking the **CastDynamicShadows** check box.

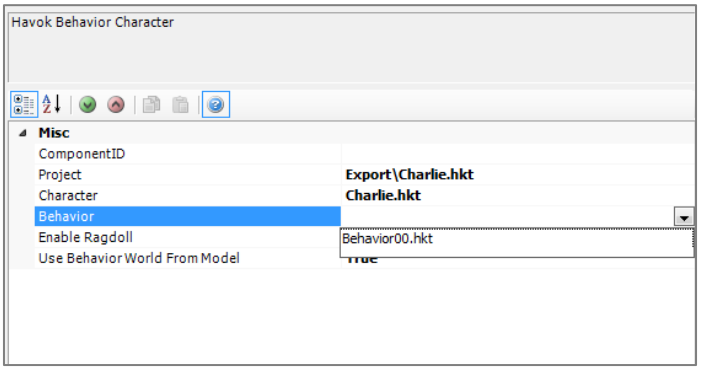


Importing the Behaviour

With the ***'Charlie'*** model still selected, open the **Components** tab and add a ***Havok Behaviour Character*** component by dragging it into the **Attached Components** window.



In the properties window for the ***Havok Behaviour Character*** click on the **Project** field and using the ellipse button to open the *Select Project* dialogue navigate to and then select ***"...\CharlieTalk\Export\Charlie.hkt".***

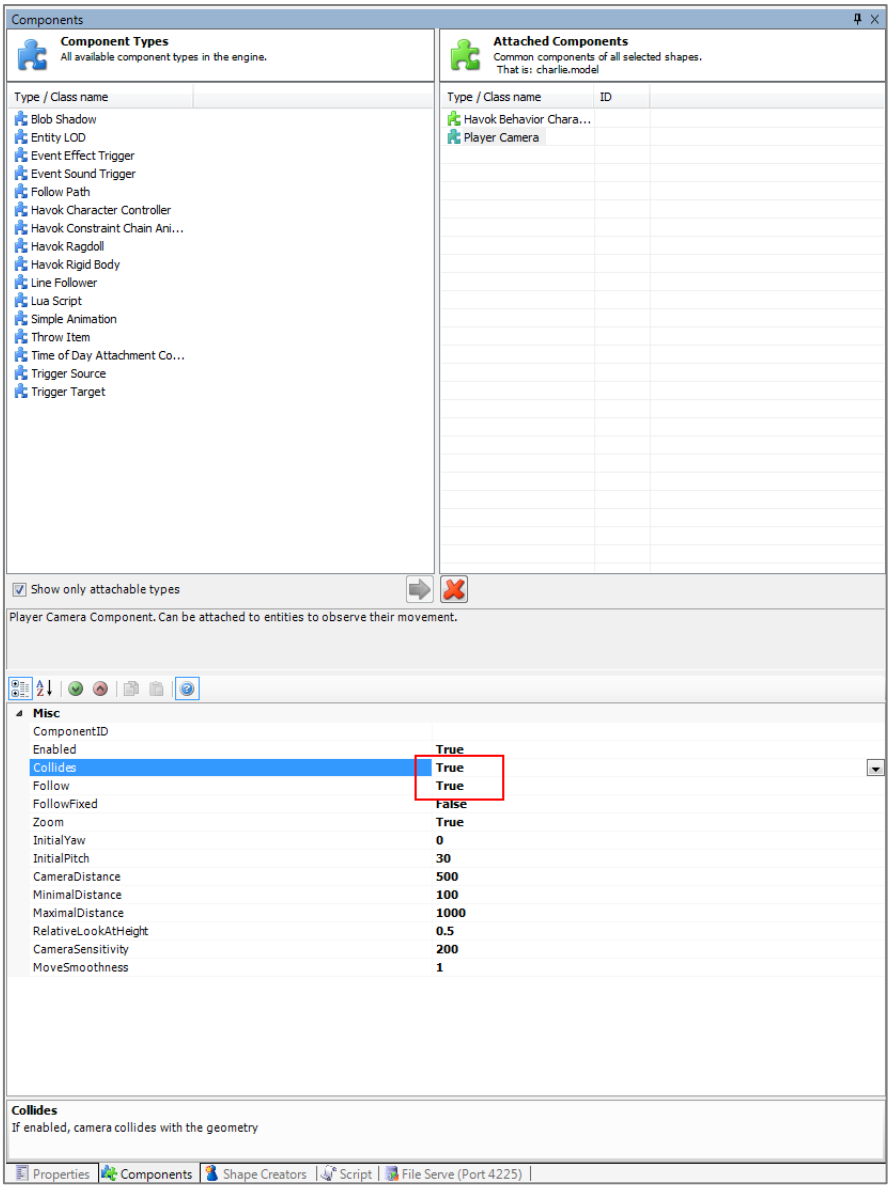


In the **Character** field, use the drop-down selector to select ***"Charlie.hkt"***

In the **Behavior** field use the drop-down selector to select ***"Behavior00.hkt"***

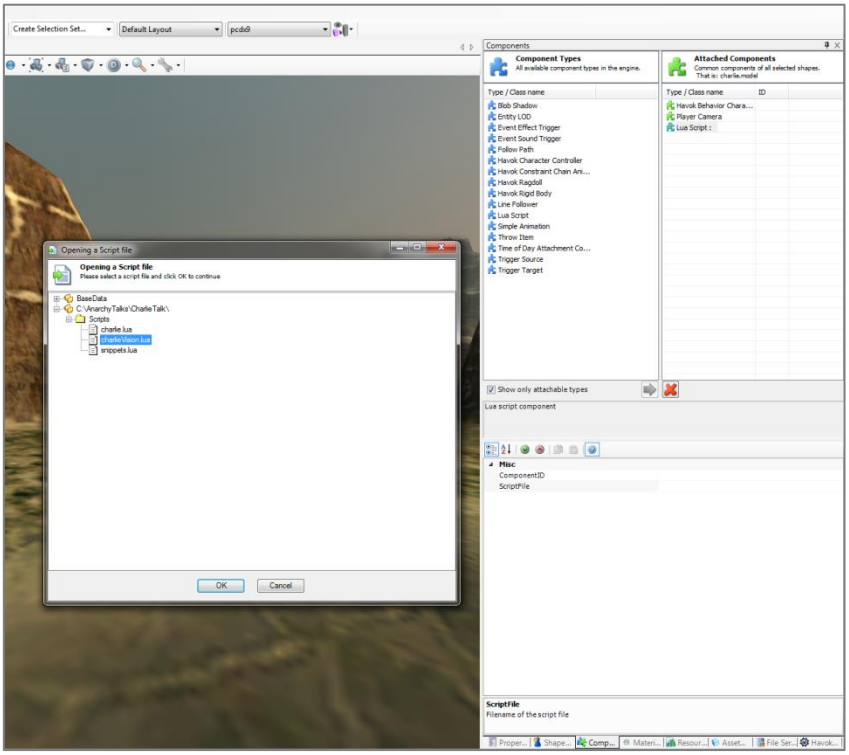
Adding the Camera

Still in the Components tab and with the ***'Charlie'*** model still selected, add a ***Player Camera*** component by dragging it into the **Attached Components** window.



Set the **Player Camera** properties ensuring **Collides** and **Follow** and both set **True**.

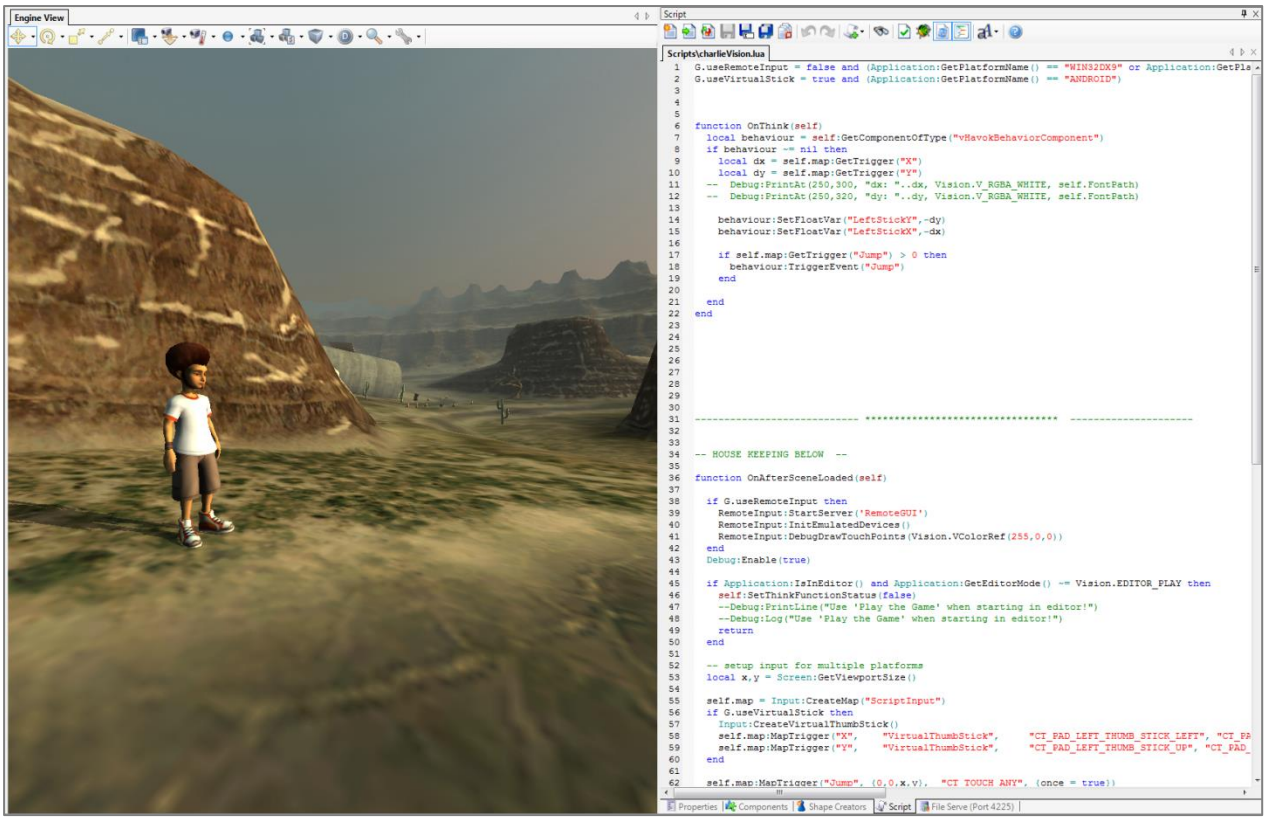
Adding a Control Script



Still in the Components tab and with the ***'Charlie'*** model still selected, add a ***Lua Script:*** component by dragging it into the **Attached Components** window.

From the **ScriptFile** field of the ***Lua Script:*** component properties open the **Opening a Script file** dialogue.

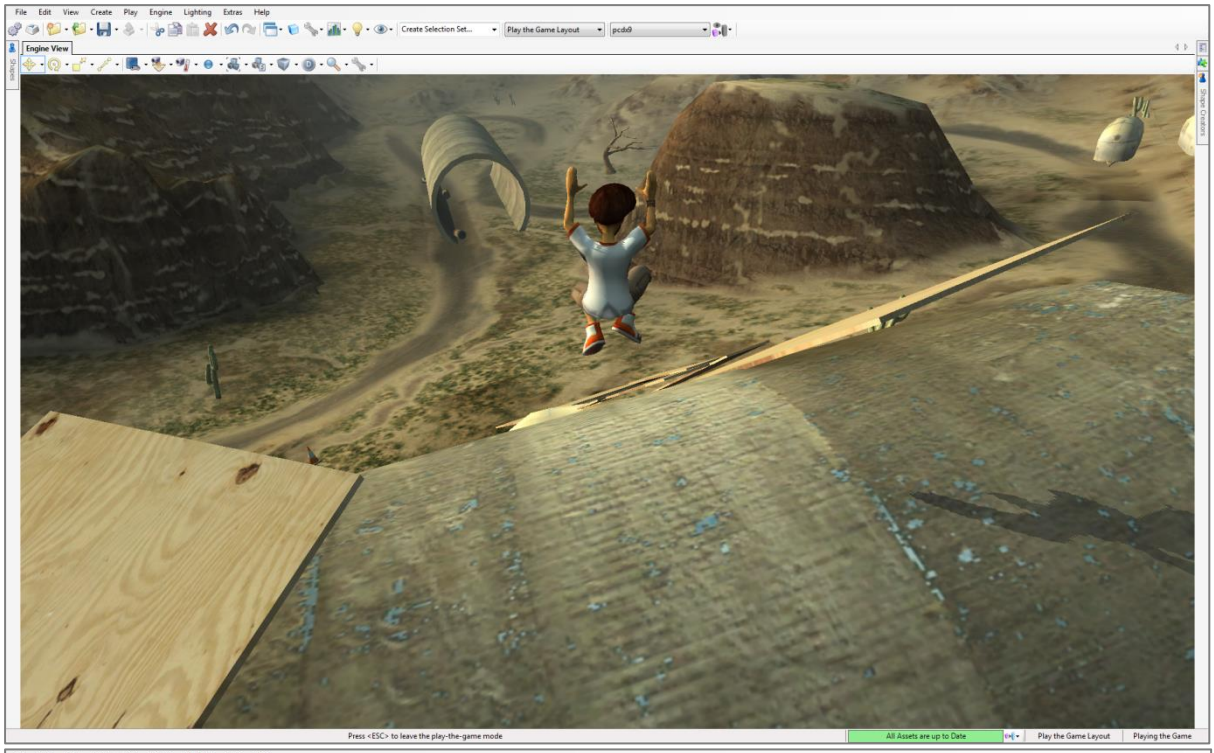
Locate and select "***...\CharlieTalk\Scripts\charlieVision.lua".***

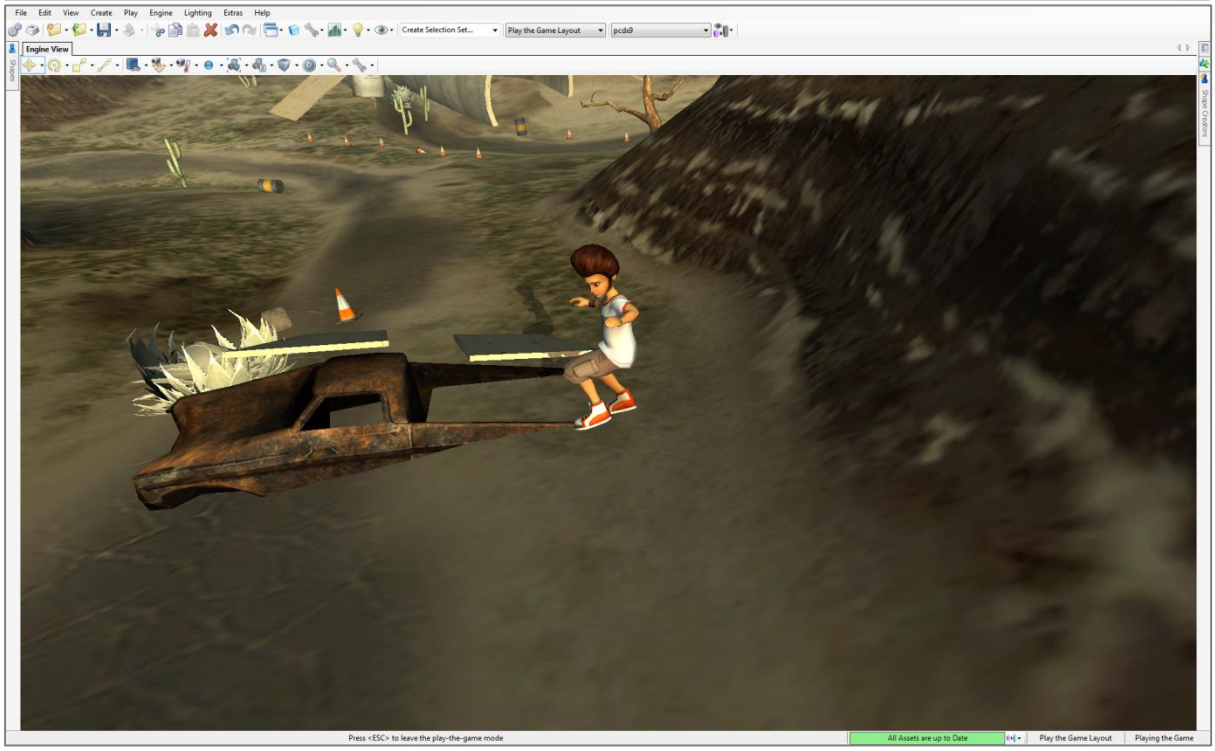


```
G.useRemoteInput = false and (Application:GetPlatformName() == "WIN32DX9" or Application:GetPla
G.useVirtualStick = true and (Application:GetPlatformName() == "ANDROID")



function OnThink(self)
  local behaviour = self:GetComponentOfType("vHavokBehaviorComponent")
  if behaviour ~= nil then
    local dx = self.map:GetTrigger("X")
    local dy = self.map:GetTrigger("Y")
  --  Debug:PrintAt(250,300, "dx: "..dx, Vision.V_RGBA_WHITE, self.FontPath)
  --  Debug:PrintAt(250,320, "dy: "..dy, Vision.V_RGBA_WHITE, self.FontPath)

    behaviour:SetFloatVar("LeftStickY",-dy)
    behaviour:SetFloatVar("LeftStickX",-dx)

    if self.map:GetTrigger("Jump") > 0 then
      behaviour:TriggerEvent("Jump")
    end

  end
end









---------------------------- ******************************** ----------------------



-- HOUSE KEEPING BELOW --

function OnAfterSceneLoaded(self)

  if G.useRemoteInput then
    RemoteInput:StartServer('RemoteGUI')
    RemoteInput:InitEmulatedDevices()
    RemoteInput:DebugDrawTouchPoints(Vision.VColorRef(255,0,0))
  end
  Debug:Enable(true)

  if Application:IsInEditor() and Application:GetEditorMode() ~= Vision.EDITOR_PLAY then
    self:SetThinkFunctionStatus(false)
    --Debug:PrintLine("Use 'Play the Game' when starting in editor!")
    --Debug:Log("Use 'Play the Game' when starting in editor!")
    return
  end

  -- setup input for multiple platforms
  local x,y = Screen:GetViewportSize()

  self.map = Input:CreateMap("ScriptInput")
  if G.useVirtualStick then
    Input:CreateVirtualThumbStick()
    self.map:MapTrigger("X",    "VirtualThumbStick",    "CT_PAD_LEFT_THUMB_STICK_LEFT", "CT_PA
    self.map:MapTrigger("Y",    "VirtualThumbStick",    "CT_PAD_LEFT_THUMB_STICK_UP", "CT_PAD_
  end

  self.map:MapTrigger("Jump", {0,0,x,y},  "CT_TOUCH_ANY", {once = true})
```

Playing the Game

All that is left to do now is open the **Play the Game Layout** (for best results) and select **Play the Game** from the tool bar and all being well Charlie will run, jump and turn around in our world.

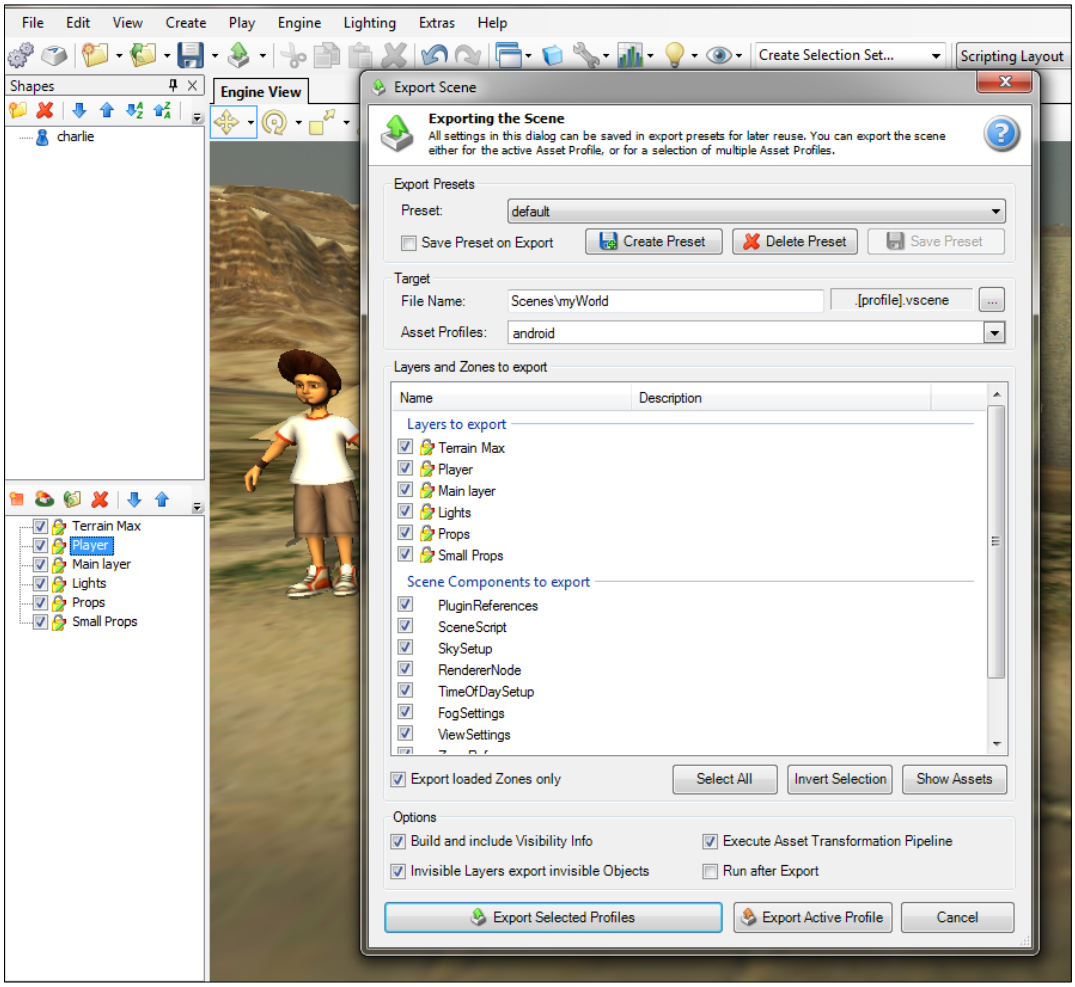




Exporting the scene

Click on the **Export** icon in the tool bar to open the **Export Scene** dialogue. From the **Assets Profile** drop-down un-check *pcdx9* so that only *android* is selected. Ensure the **File Name** is set to ***'Scenes\myWorld'*** and click on the **Export Selected Profiles** button.

The scene will be exported.



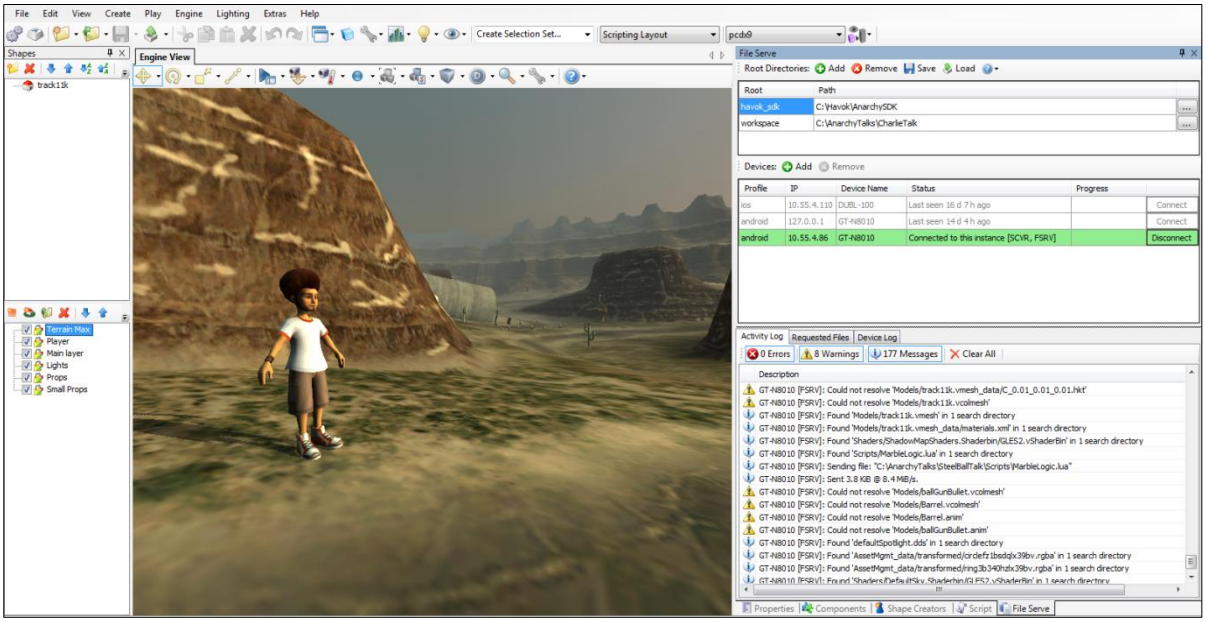
Deploying to device

Ensure the Android device has Wi-Fi enabled and is connected to the same network as the PC.

Open the **File Serve** tab and check the *havok_sdk* and *workspace* **Paths** are correct. They should point to the installed ***AnarchySDK*** folder and the location of the ***Charlietalk*** folder respectively.

Launch ***vPlayer*** on the Android device and in the **Device** pane **Connect** to the instance of the device. Within the ***vPlayer*** App select the **Exported** tab and tap on the ***myWorld(Exported scene)*** that should be displayed.

The scene will now load on the device. *(note: on first install this may take a few moments as the files have not yet been cached on the device.)*



The 'game' will now be running on the device. Tilt the device to steer the ball, tap to jump.

If establishing a network connection fails altogether, you can connect to an Android device by setting up a local port forward by running the following command with administrator privileges:

```
adb forward tcp:4223 tcp:4223
```

Afterwards, add `127.0.0.1` as a device entry. This entry will remain invisible, but you can connect to it anyway as described above. You will have to restart the port forwarding every time the ADB server exits.

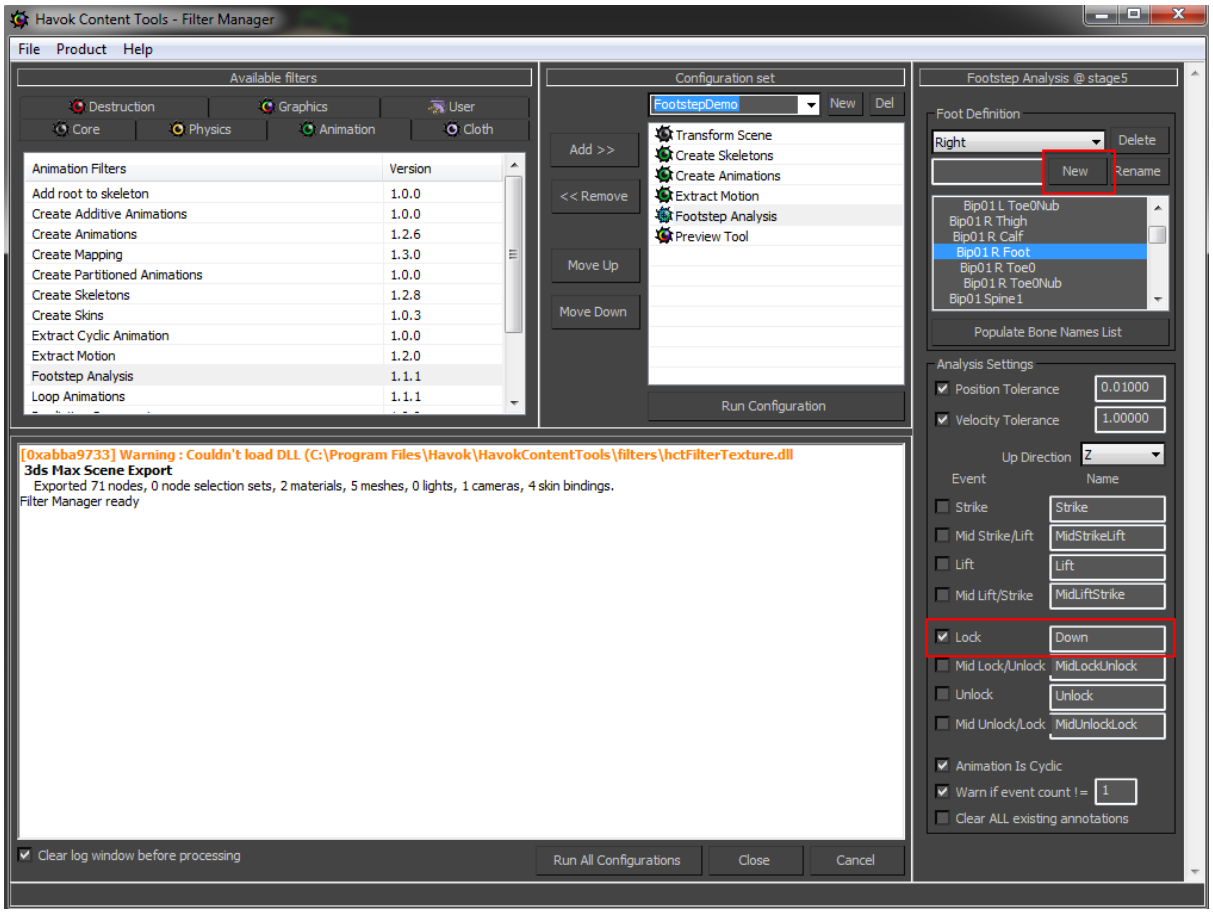
note: that this will cause any feature that relies on connecting from the device to the host PC to fail (e.g. automatically running a scene in the vPlayer after exporting it), so it is recommended to use this connection method if no alternative is available.

A: Footstep Analysis

Footstep analysis can easily be added to the animation export pipeline saving the need to annotate the feet manually. It is fully customisable and allows for fine adjustment to achieve the desired results.

From within *3dsMax* (Or *Maya*) launch the **Havok Content Tools - Filter Manager** and from the **Animation Filters** tab add the **Footsteps Analysis** filter to the **Configuration Set**. Also add the **Preview Tool** from the **Graphics** tab and optionally remove the **Write to Platform** filter.

Select the added **Footstep Analysis** filter to open the properties sheet.

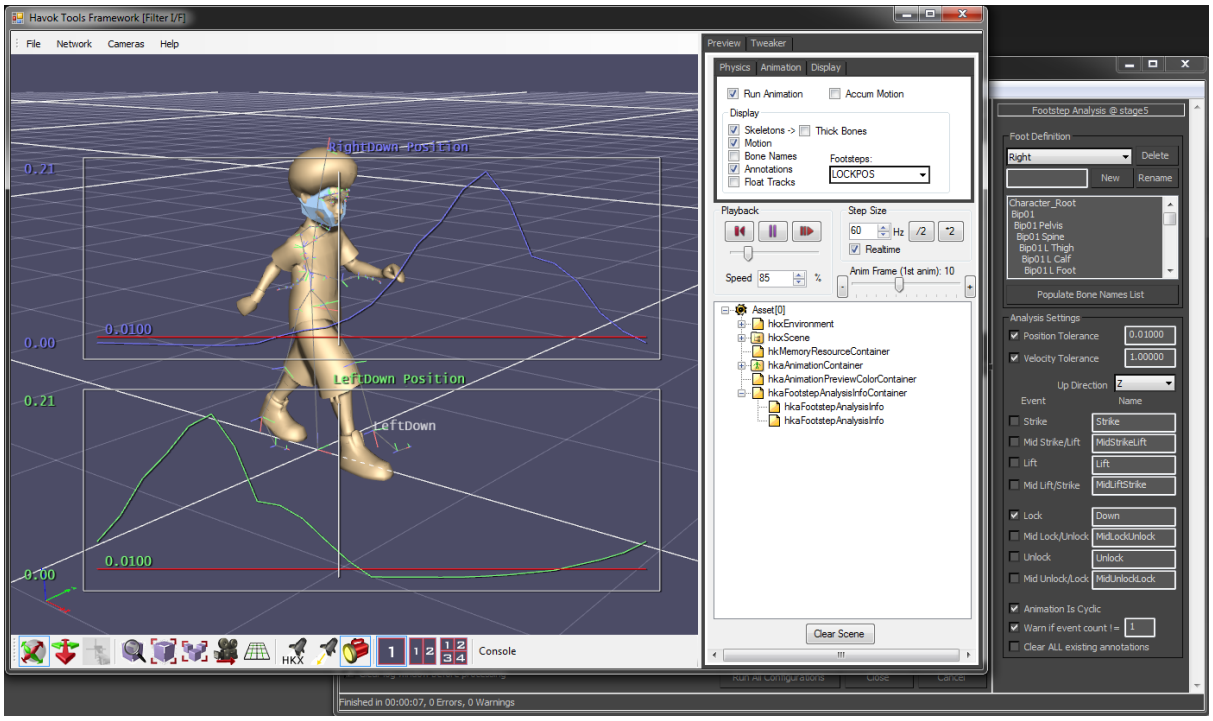


Click on **New** and create a definition called ***"Right".*** Navigate the Bone hierarchy and select ***Bip01 R Foot***.

Click on New again and create the ***"Left"*** definition and in the Bone hierarchy associate ***Bip01 L Foot***.

In **Analysis Settings** check the **Lock** check box, and change the event name to ***"Down".***

Click on **Run All Configurations**. The **Preview Tool** will be launched and display the analysis of the footsteps and the annotated events.



Once happy with the results, remove the **Preview Tool** and if necessary re-add the **Write to Platform** filter and re-export the animation data.