Overview

This tutorial aims to explain the process of designating the player character and setting up a handful of animations: idle, walk – or, in this tutorial, run – turning left and right, and jumping.

The Mannequin Editor will be used to loop and fine-tune the transitions between the different animations.

In this tutorial, a third person game will be made, but these same techniques can also be applied to a first person game design.

> This tutorial follows the previous tutorial, Tutorial - Replacing the Player Character. For instructions on how to import a character and make it the player character, please read that tutorial.

> If you prefer to watch a video tutorial rather than read a written one, check out the video tutorial at the bottom of this page.

> Please be advised that GameSDK is NOT used for this tutorial. It is based on the templates that are provided with the Engine, so create a new project using the Third (or First) Person Shooter C++ template.
>
> It is not possible to continue on from the previous tutorial, as that was made in an isometric template.

On This Page

Related Pages

Tutorial - Replacing the Player Character

Tools Used

- Mannequin Editor
- Microsoft Visual Studio® (a free Community version is available)
- Any .xml editor, like Notepad++

Files

Download the following file and extract into your project folder:

deer_tutorial_part2.zip

> To see where the project folder is, hover over your project in the Launcher, click the cog i con and choose Reveal in Explorer.

Of course, any self-made character with animations can also be used.

> If a character definition file (.cdf) was already created in another project, this can be copied into a folder within the assets\objects\characters folder. Otherwise, follow the steps outlined in the previous tutorial on importing a character and animations using the Character Tool first.

Generating and Opening a Solution

Like in the previous tutorial, a C++ solution needs to be generated before it is possible to change the code. Of course, when following directly from the previous tutorial, this is not necessary as long as the first- or third-person template was used.

1. Right-click on the .cryproject file and choose Generate Solution.
2. When finished, open the solutions\game.sln file in Visual Studio.

In Visual Studio

Once Visual Studio has been opened, two files need to be edited: player.h and player.cpp.

Player.h

1. In Visual Studio, open the Project folder, then Components and player.h.
2. Lines 32 – 35 store flags for the left, right, forward and backward movements, and are mapped elsewhere to the W, A, S and D keys on a PC. Because a new jump animation will be added, we need a flag to capture the Jump action here as well:

```
MoveLeft = 1 << 0,
MoveRight = 1 << 1,
MoveForward = 1 << 2,
MoveBack = 1 << 3,
Jump = 1 << 4
```

> ⚠ Don't forget to add the comma at the end of line 35 to indicate that this code block continues onto the next line, and do not put a comma at the end of our new line.

3. A FragmentID for the jump animation needs to be added on what is now line 129:

```
FragmentID m_jumpFragmentId;
```

> ⓘ The m_ prefix is used by Crytek staff to easily distinguish between member variables and non-member variables.

That's it for the player.h. file.

Player.cpp

Now open the player.cpp file.

1. Make sure that line 24 points to the .cdf file that was created for the player character. In this case:

```
m_pAnimationComponent->SetCharacterFile("Objects/Characters/hazmat/hamza_final.cdf");
```

2. On line 41, retrieve the value of the jump fragment from the animation component and assign it to the member that was just created:

```
  // Acquire fragment and tag identifiers to avoid doing so each update
    m_idleFragmentId = m_pAnimationComponent->GetFragmentId("Idle");
    m_walkFragmentId = m_pAnimationComponent->GetFragmentId("Walk");
    m_rotateTagId = m_pAnimationComponent->GetTagId("Rotate");
    m_jumpFragmentId = m_pAnimationComponent->GetFragmentId("Jump");
```

> ⚠ Remember that the name of the animation, "jump" in this case, has to match what you named your animation when you created the Character Definition File.

Best practice is to treat all code as case-sensitive, because Windows is the only platform that ignores case. If the cases don't match, a message that something "can't be found" will be displayed. If this is the case (no pun intended), check the cases in the file paths.

3. Bind the Spacebar key on the PC platform to our jump animation starting on line 66:

```
m_pInputComponent->RegisterAction("player", "jump", [this](int
activationMode, float value) {
HandleInputFlagChange((TInputFlags)EInputFlag::Jump,
activationMode); });
  m_pInputComponent->BindAction("player", "jump",
eAID_KeyboardMouse, EKeyId::eKI_Space)
```

4. Define the physical force for the jump action on line 154:

```
const float jumpSpeed = 5.f;
```

This velocity is in m/s. This will need to be tested in your game and adjusted to the desired value; this value is just a starting point.

5. Tell the Engine in which direction the character should jump (which is Z for the Z axis, i.e. up):

```
    if ((m_inputFlags & (TInputFlags)EInputFlag::Jump) &&
m_pCharacterController->IsOnGround())
    {
       velocity.z += jumpSpeed;
    }
```

The bit of code reading && m_pCharacterController->IsOnGround()) checks whether the player character is on the ground.

Make sure the velocity is positive, otherwise the character will "jump downwards" through the terrain.

6. Lastly, the code on line 230 needs to be replaced to trigger the correct animation based on the player's current status. For example, if the player presses the jump key, they should only jump if they're currently on the ground, but not if they're already in the air.

   Old line 230:

```
const auto& desiredFragmentId =
m_pCharacterController->IsWalking() ? m_walkFragmentId :
m_idleFragmentId;
```

   If the player is on the ground, they should be idling or walking. From line 230 onwards, it should now read:

```
FragmentID desiredFragmentId;

if (!m_pCharacterController->IsOnGround()) // check if player is on
the ground before jumping
{
  desiredFragmentId = m_jumpFragmentId;
}
else
{
  if
(m_pCharacterController->IsWalking())
//
  {
    desiredFragmentId = m_walkFragmentId;
  }
  else
  {
    desiredFragmentId = m_idleFragmentId;
  }
}
if
(m_activeFragmentId != desiredFragmentId)
{
    m_activeFragmentId = desiredFragmentId;

m_pAnimationComponent->QueueFragmentWithId(m_activeFragme
ntId);
}
```

7. Compile the project (Build -> Build Project) and exit Visual Studio, presuming no errors have presented themselves.

> If the Editor is launched directly from Visual Studio, the Mannequin setup file can not be loaded.

In Mannequin

The following steps need to be executed in Mannequin, which includes editing .xml files as well as setting things up in the Mannequin Editor.

Pointing Mannequin to Third Person Character

1. Reveal the project from the Launcher and open Assets\Animations\Mannequin\Preview\Player.xml in any plain text editor.
2. This tutorial is focused only on a third person project, so remove references to the first person in the player.xml and update the location of the player character. The player.xml file should then read something like this:

```
<MannequinPreview>

 <controllerDef
filename="Animations/Mannequin/ADB/FirstPersonControllerDefiniti
on.xml"/>

 <contexts>

  <contextData name="ThirdPerson" enabled="1"
database="animations/mannequin/adb/thirdperson.adb"
context="ThirdPersonCharacter"
model="objects/characters/hazmat/hamza_final.cdf"/>

 </contexts>

 <History StartTime="0" EndTime="0"/>

</MannequinPreview>
```

They key here is to point the third person context to the exact path and filename of the player character definition file that was already created. It’s a good practice to copy and paste this path and filename from the File Explorer to avoid typos.

Up to CRYENGINE 5.5.2, the paths and filenames need to be kept all lowercase or the Mannequin will say that it can’t find the .cdf file even when it’s there.

Some readers may have noticed that the FirstPersonControllerDefinition.XML file wasn't renamed. Although given that this is a third person game it would make more sense to call it "ThirdPersonControllerDefinition.xml," if the name is changed, all the references to it in player component C++ code will also need to be changed, so it’s just easier to leave it as-is.

3. Save and close this file.

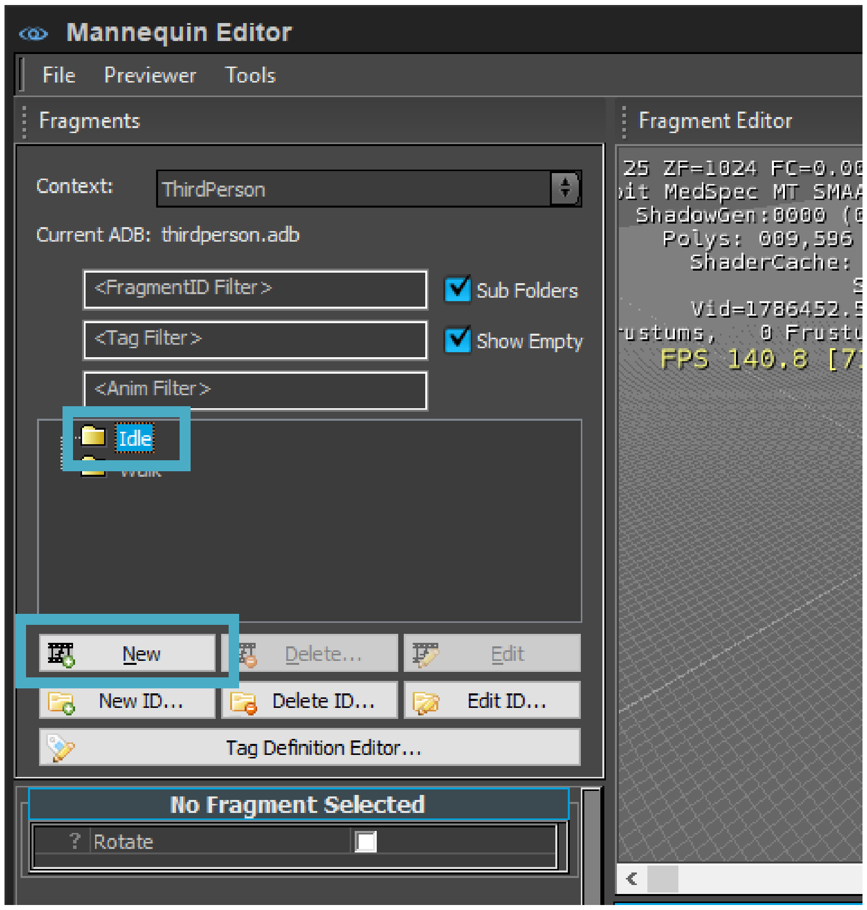
In Mannequin Editor

In the Mannequin Editor, the animations will be assigned to fragmentIDs so that they will be played on the character model when it stands idle, walks or jumps. Also

1. Open the project in the Sandbox. Opening a level is not necessary yet.
2. Open the Mannequin Editor tool (Tools -> Animation -> Mannequin Editor).
3. In the Mannequin Editor menu, choose File -> Load Preview Setup and open the player.xml file from the Assets\Animations\Mannequin\Preview folder.

At the bottom of the Fragments panel in the top left, there are buttons to create, modify and delete all of the things that manage animations. There are two fragment IDs defined in code: idle and walk. A third, jump, has been added in the code, but that has not been added in the Mannequin Editor yet, and the fragment IDs have no animation associated to them yet.

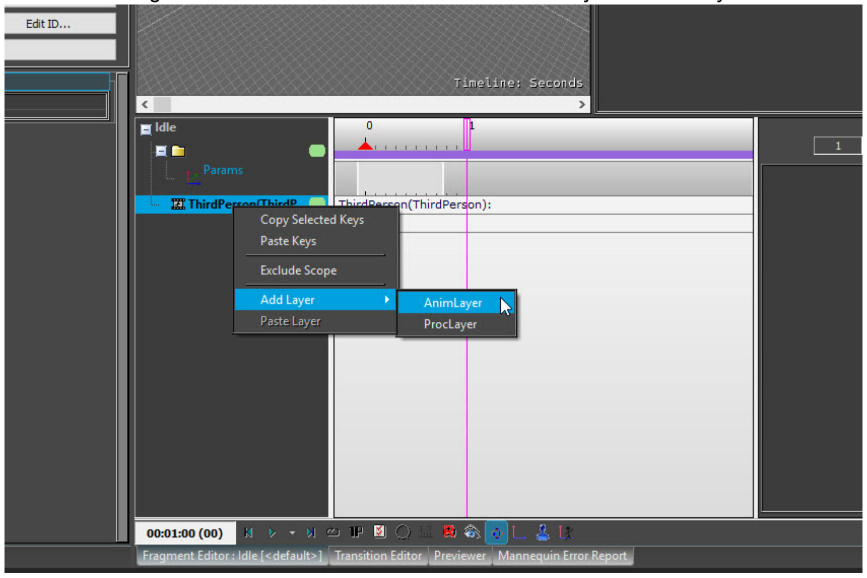
1. Select the Idle fragment ID and click on the New button to create a new Fragment. The Fragment will be called Option 1 and be put into what looks like a yellow folder called <default>:



Creating a new FragmentID

<Default> refers to the context – the conditions in which we'd even consider playing these animations. These are called tags. These context tags won't be discussed in this tutorial yet, so don't worry about them for now. For now, just use the default tag.

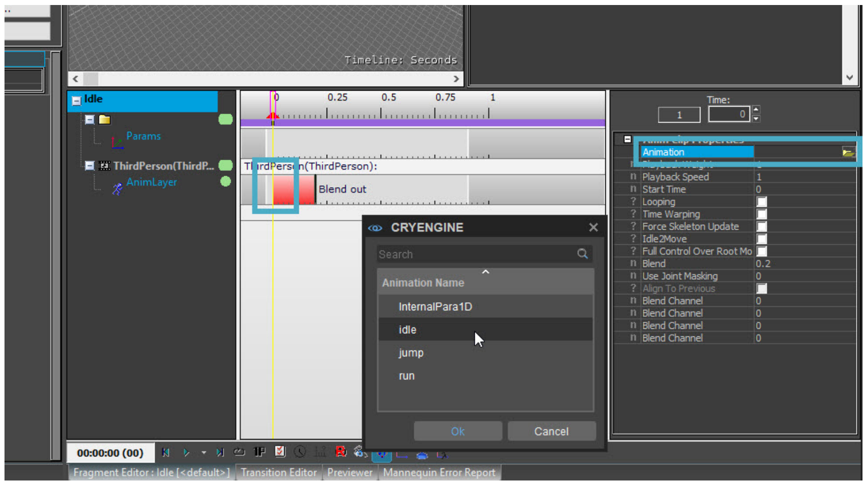
2. Double click on the new Option 1 fragment. Make sure the Fragment Editor is active by clicking on the corresponding tab at the bottom of the Mannequin Editor. Right click on the ThirdPerson fragment next to the timeline and choose Add Layer -> AnimLayer:



Adding an Animation Layer

3. Double click on the first frame of the ThirdPerson timeline to create a new animation clip.

Its properties appear at the right as Anim Clip Properties. Click on the value box next to Animation and browse to the Idle animation that you imported using the Character Tool:



Animation	
Playback Weight	1
Playback Speed	1
Start Time	0
Looping	☐
Time Warping	☐
Force Skeleton Update	☐
Idle2Move	☐
Full Control Over Root Mo	☐
Blend	0.2
Use Joint Masking	0
Align To Previous	☐
Blend Channel	0
Blend Channel	0
Blend Channel	0
Blend Channel	0

4. Enable looping to make sure that when standing still, the Idle animation will keep repeating itself.

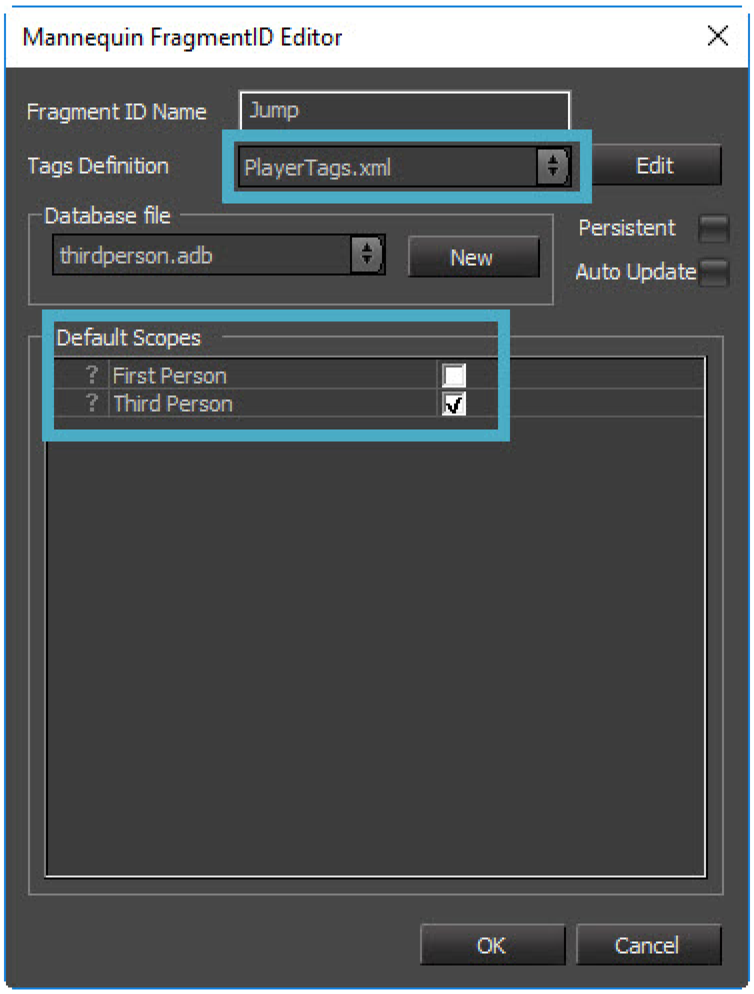
> Make sure that the animation clip starts at frame 0. If it starts somewhere else on the timeline because of a misclick, drag your animation over to start at frame 0.

> The purple animation clip is the actual .caf file that was created when importing the animation in the Character Tool. These animation clips are located in the animations folder inside the folder where the character was copied to at the beginning of this tutorial.

> It’s a good idea to preview how the animation looks. On the playback controls at the bottom of the screen, there's a loop button. This just affects this preview, not anything in game.

> If the animation preview pauses at the end, creating a hiccup in the loop, this has nothing to do with the animation clip; it will only appear in the Preview panel in the Mannequin Editor. However, it may be preferable to see a smooth animation loop. To do this, just right click just above the end of the animation clip on the timeline above the animation layers. A little red triangle will appear, which will mark where the preview should end. Again, none of this has any effect on what happens in game; that’s all determined by the Anim Clip Properties and where the clip was placed on the timeline.

5. Follow the same steps to set up the Walk (choose the Run animation for this), and Jump a nimations. Jump is not part of the default code, so make an entirely new FragmentID. Tell it to keep my tag definitions in the PlayerTags.xml file, and limit the scope of this to third person for now:



Fragment ID Name: Jump
Tags Definition: PlayerTags.xml
Database file: thirdperson.adb

Default Scopes

?	First Person	☐
?	Third Person	✓

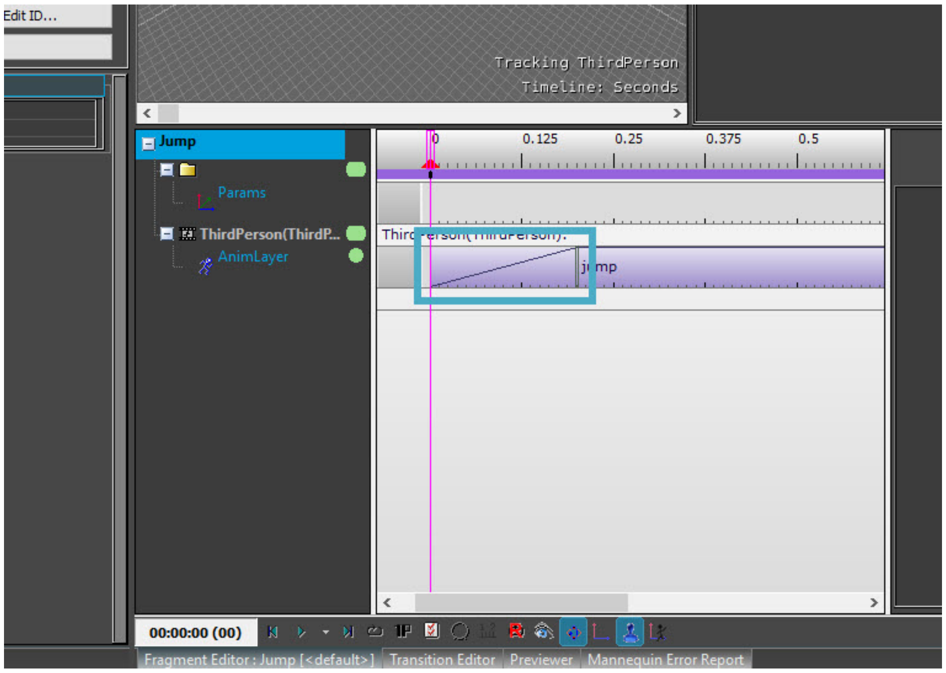
New Jump FragmentID

If it was an animation that worked fine for a first person perspective, and that needs to be supported as well, then leave that scope checked.

Blend Transitions

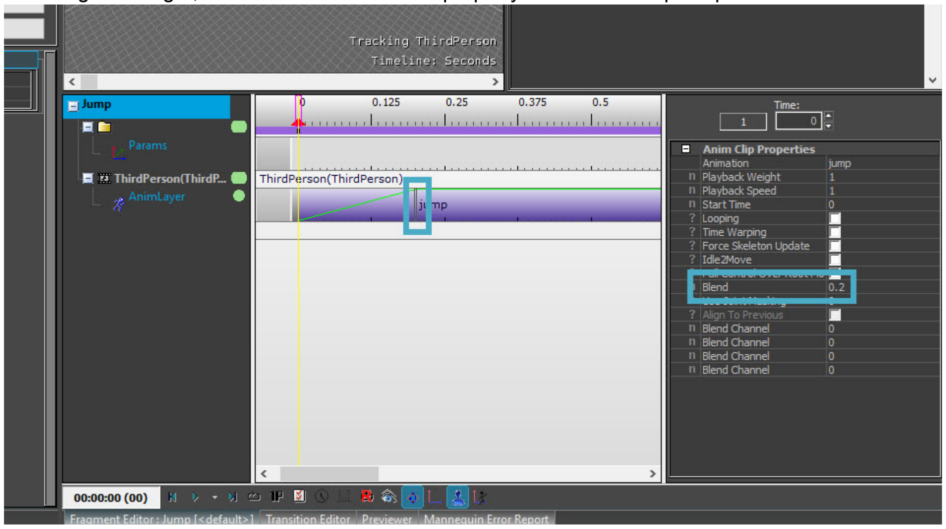
Mannequin provides something called a Blend Space to interpolate between animation parameters (thus “parametric”), which is outside the scope of this tutorial. The slanted line at the beginning of this animation clip is called a Blend Transition, which interpolates between animations over time.

For now, the only transition being discussed is a more simple version called a Blend Transition. This is represented by this slanted line at the beginning or end of an animation clip:



Blend Transition

The Blend Transition is the period of time over which the animation system transitions the skeleton between the current animation and this one as it is requested. You can drag this vertical double bar to change its length, or edit it under the Blend property in the Anim Clip Properties:



Anim Clip Properties	
Animation	jump
Playback Weight	1
Playback Speed	1
Start Time	0
Looping	☐
Time Warping	☐
Force Skeleton Update	☐
Idle2Move	☐
Full Control Over Root Mo	☐
Blend	0.2
Use Joint Masking	0
Align To Previous	☐
Blend Channel	0
Blend Channel	0
Blend Channel	0
Blend Channel	0

Adjusting Blend Transition

The default is 0.2 seconds.

> ⓘ In order to see this working, more than one animation needs to be set up.

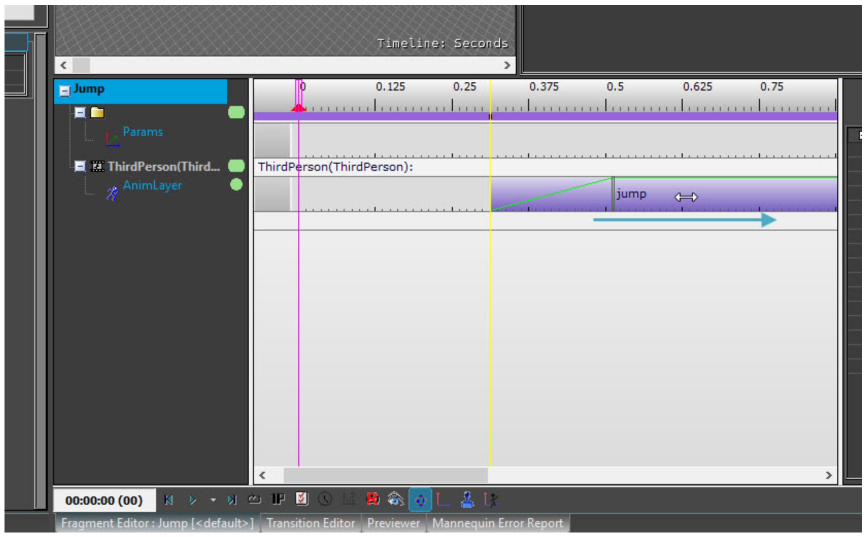
Now that all animations have been set up, go to File -> Save Changes.

To see this in action, create or open a level and go into game mode (Ctrl + G).

The Idle animation will be played right away as the character is not moving in any direction yet. Use the W, A, S or D key to trigger the walking fragment (or running, in this case), and Spacebar to jump into the air. The force that was applied in C++ on the Z axis is clearly too strong for a human being under Earth gravity, so go back and lower that value and test until it looks as desired.

Two examples of how the animations can be tweaked in the Mannequin Editor:

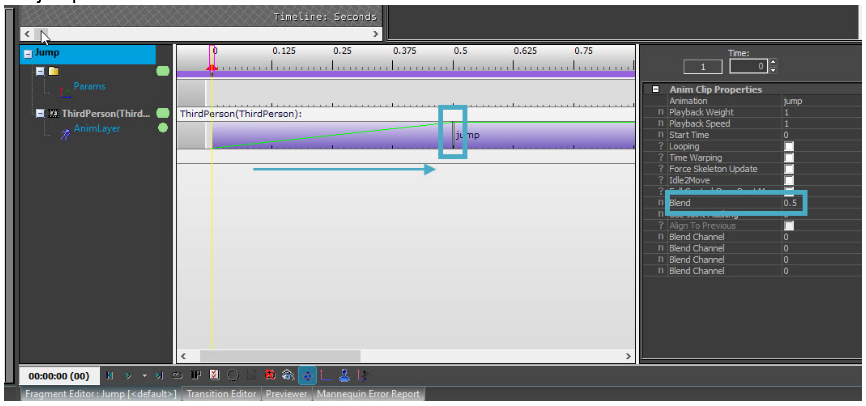
- Press Escape to go out of game mode, open the Jump fragment and just drag the animation so it doesn't start at frame 0:



Delaying Jump Animation

Go back into game with Ctrl + G, and observe that when pressing the Space bar to jump, the physical force pushes the player upwards, but the jump animation doesn't start playing right away because it was delayed.

- Drag the animation back to start at frame 0, but increase the blend transition at the start of the jump animation to 0.5 seconds:



Time: 1 0

Anim Clip Properties	
Animation	jump
Playback Weight	1
Playback Speed	1
Start Time	0
Looping	☐
Time Warping	☐
Force Skeleton Update	☐
Idle2Move	☐
Blend	0.5
Align To Previous	☐
Blend Channel	0
Blend Channel	0
Blend Channel	0
Blend Channel	0

Go into game, and when jumping, the body will be coiling from idle into the jump position over that half second interval.

This completes the introduction to setting up a new animation FragmentID and the basic use of Mannequin.

Combined with the assets from the previous tutorial, the level could now look similar to this:

In the next tutorial, more animations will be added and tags will be used to create contexts to select animations based on game state and the character’s current position.

Example Level

An example level can be found on the Marketplace that combines the results from the previous tutorial and this one. You can find it on the Marketplace.

✓ In this example level, it is possible to switch between the deer and the hazmat character by pressing F1.

Video Tutorial

To see this tutorial in video format, and to get a bit more background information, see the video below: