LP2B – Multimedia: Digital Representation

- LP2B Tutorial Class n°5 -

Animation in Unity : Controllers and Clips

An intricate system to control animations and transitions

This class will give you an overview of :

- The Button's "Animation" mode -
- Creating and Editing Animation Clips -
- Animator component and Animator Controller -
- Animating any entity from scratch -
- Transitions in the Animator Controller -
- A powerful tool requires lots of experience! -

Lets go back to our Video Player...

- Open the hub and open the “Test Video” project again.

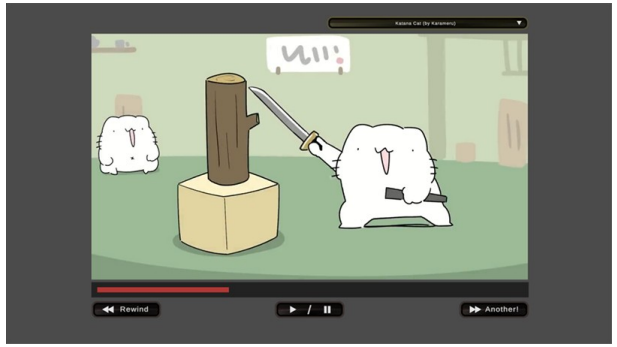
- Open the scene with the Video Player, and the nice buttons we created last time!
  We used the “Sprite Swap” setting to have them change appearance, remember?

- But what if I told you that we can make this buttons EVEN BETTER?
  For that, we are going to use the most complex but most powerful button mode : ANIMATION

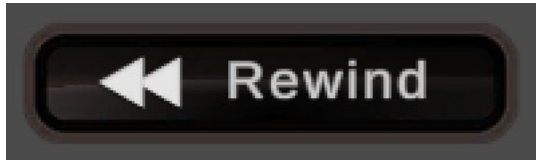
- Animation can be applied to any GameObject in Unity. But we are going to start with the Button, as using its “Animation mode” will do some of the work for us!
  The partial automation is why buttons are the best way to get into this topic

- Once we get this simplified case of the Button, we’ll generalize to everything else
  Be aware, though, that good animation is very much a matter of experience
  Even once you get the principle, you will still need practice to make endearing animations

- Anyways, let’s begin our dive into the world of animation!



The state we left the project in (roughly)



Our buttons made with SpriteSwap mode
That mode isn’t bad, but it’s still fairly limited.

The only way to gain TOTAL control of the button’s appearance is to use Animation mode!

Setting Animation Mode on a Button

- Pick on of your buttons and set it to “Animation mode” for transitions
  In this mode, Animation clips will be used to manage the button states

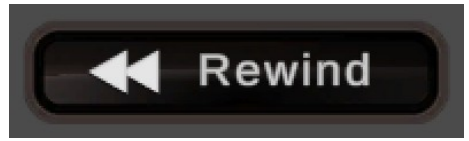
- Once the mode is selected, it is possible to “Auto-Generate Animation”
  Doing that will automatically create an Animation Controller and Animation Clips for our button

- When you “Auto-generate”, Unity asks you to name the controller it will create
  Name it “Button” or “ButtonAnim”, as we are going to use it for several of our buttons

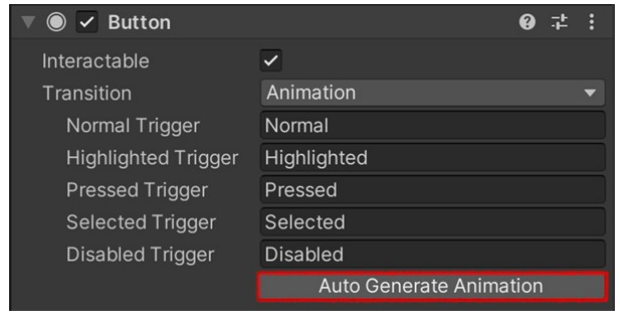
- The controller comes with 5 animation clips “folded” into it
  Those are the 5 animations for the 5 states of the button. We’ll have to edit those to animate!

- Please sort your controller into an “Animation” folder
  This folder can be inside “graphics”, or separate. Both make sense as animation is visual

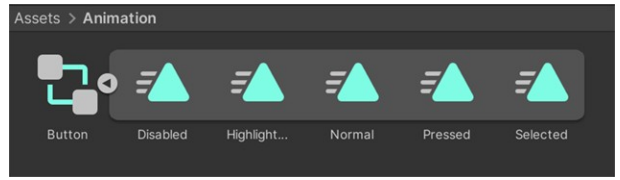
- The controller is a customizable entity. But for now we’ll focus on the clips
  The auto-generation of controllers for buttons is why I used them as a gateway to this topic!



Let’s pick one of our buttons…



Once the Button is set to animation mode, a button “auto Generate animation” is available



My animation controller (named “Button”) with its 5 animation Clips inside

The icons allow me to identify them as controllers and clips, which is why names like “Button” are OK!

The Animation Tab

- The editing of Animation Clips takes places in the Animation Tab
  But this tab is hidden in the default Unity interface. We will have to activate it

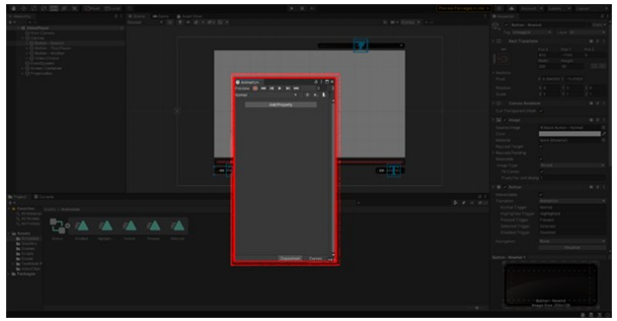
- To activate the tab : *(Window=> Animation => Animation)*
  You can see that there are other tabs in the “animation” category. We’ll talk about them later

- I recommend moving the tab into the bottom area, near “project” and “Console”
  But you can also keep it in a sub-window if you want. Just consider that we will need lots of horizontal space, which the bottom area provides perfectly

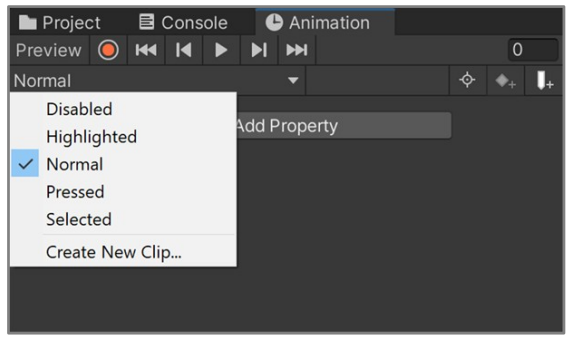
- Make sure the Button we set to “animation mode” is still selected in hierarchy

- If it is, you should be able to see the 5 animations from before in the list of Clips
  The list of clips is in the top-left corner. Use the images on the right to help you find it

- If you change the clip, nothing seems to change. It’s because all clips are EMPTY!
  The auto-generation of controllers for buttons is why I used them as a gateway to this topic!



Once you make the animation tab visible, it might pop up like this, in a big sub-window



Don’t get confused by the names! “Disabled”, “Normal” and all the others are Indeed our animation clips!

The Animation Timeline

- The part highlighted in red on this screenshot of the animation Tab is the Timeline

The Timeline will allow us to define Keyframes, which will modify an Object's status and appearance at a specific time.
Whenever it is possible, the Animator Component will INTERPOLATE between the Keyframes, creating smooth transitions

Lost? Don't worry. This will become clearer as I guide you through your first animation!

Some Things to Grasp First

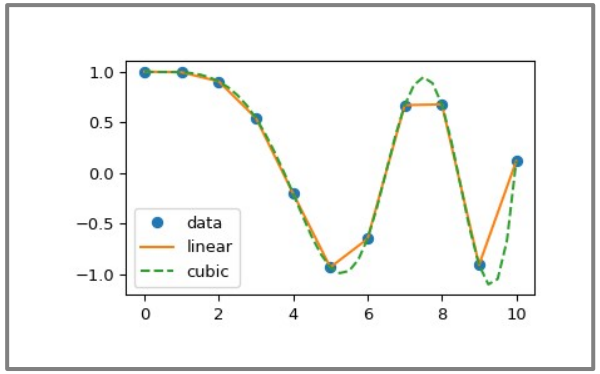
- For the Animation window to display information, you need to select a GameObject carrying an animator in the hierarchy *(or one of its children)*
  Children may carry their own animator. But we'll avoid this case for now

- A computer animates through math, not through art. The key mathematical concept here is INTERPOLATION between steps called KEYFRAMES.
  Animator grants total control over these interpolations. But as beginners we'll use default ones

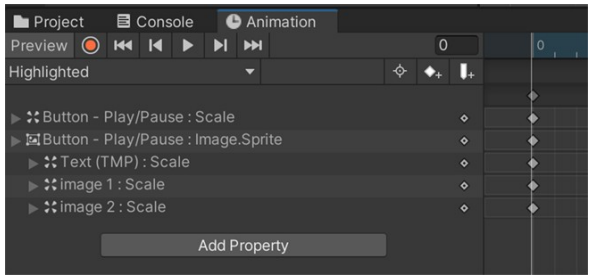
- Animations actually manipulate the values of our GameObjects and components : position, rotation, color… all those are numbers, and therefore can be interpolated over time!
  You can switch between animations managed by the same controller on the left of the window

- The Controller is the entity which regulates transitions between the animations clips it contains. Animation clips are the animations themselves
  Properties that are not numbers can be modified by Animator, but there will be no interpolating

- Animation creation occurs in a "separate time" : the changes you do to your GameObjects to create your animation will not affect their initial state in the scene
  Those are the 5 animations for the 5 states of the button. We'll have to edit those to animate!



Unity Interpolates through Bezier Curves, meaning the values are interpolated like the green dotted line.



On this completed animation, you can see the list of properties that the animator will modify and interpolate

Using Record Mode to Create Keyframes

- Using the list, change the current animation to “Highlighted”

- Enter “record mode” by clicking the “red circle button” of the Animation Tab
  When you are in record mode, the **timeline becomes red** as a reminder you are in this mode

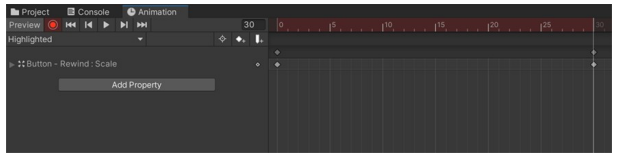
- You can choose the time to edit by clicking in the upper area (now red)
  The numbers indicate the time, either in frames or seconds (can be set with a menu on the right)
  Warning : even if the time is in seconds, a second is still split into 60 parts rather than 100

- Set the time at 30 frames, and start modifying the button directly
  Because we are in “record mode”, the changes will happen AT THE SELECTED TIME

- For instance, change the Button’s scale to make it bigger. Keyframes appear!
  As long as you’re not moving the white line, the changes keep happening on frame 30

- If the button moves when scaled, it means its pivot of the RectTransform is wrong!
  EXIT RECORD MODE, then set it to X=0.5 Y=0.5. Otherwise it will only be correct on frame 30

When you edit any property in record mode,
The field changes color to become red
It shows the changes only affect the object
at specific time in the timeline



After I changed the Scale on frame 30, a line
with the “Scale” property appeared.
That line has 2 keyframes : on 0 and 30

Confused? Let's review what we did here...

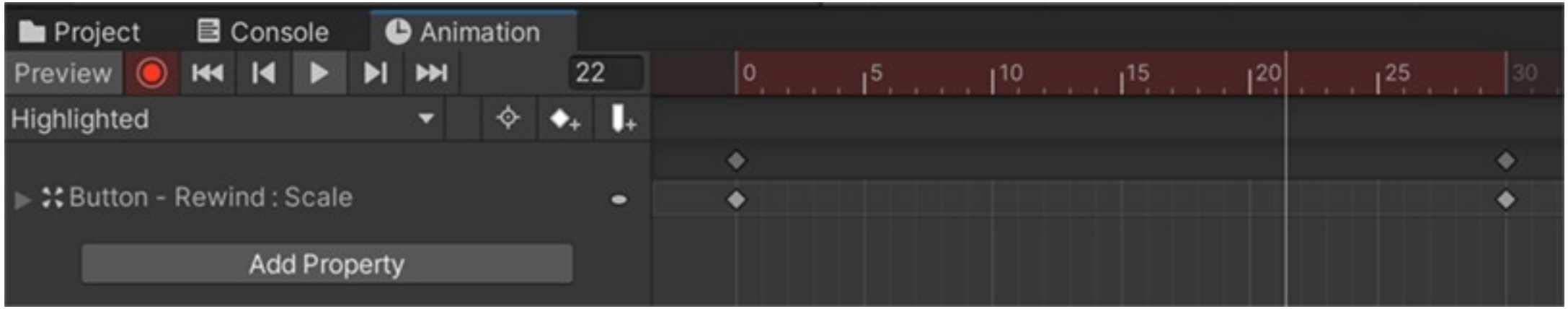
- Question : We only created a different state on frame 30. So why are there 2 Keyframes?

  Answer : The Keyframe that was automatically created at frame 0 has the original state of the object

- Question : Why was a line with *"<GameObjectName> : Scale"* as title created?

  Answer : We edited the Scale property in Record Mode. This made it part of the data affected by the animation

Animations are, in summary : A set of GameObjects Properties that are altered by Keyframes over Time



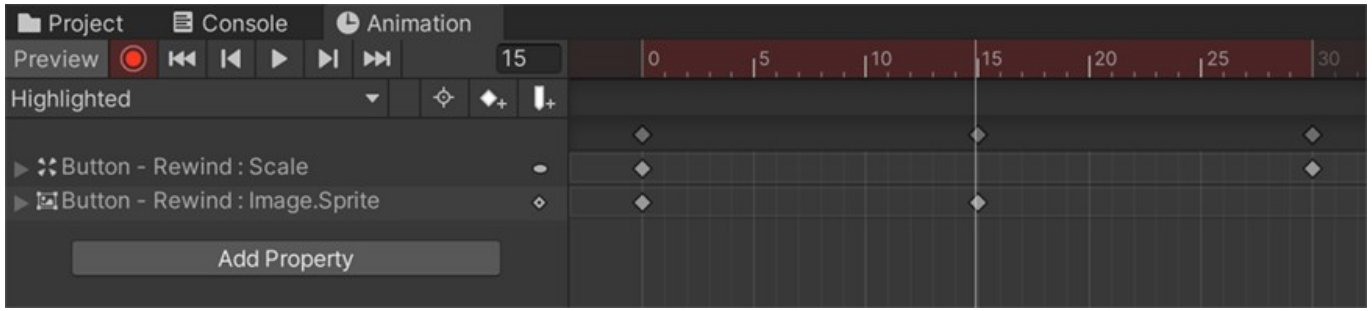
Try testing your Animation by clicking the "Play" button in the upper left corner (near Record Mode)

Interpolation, or how Numbers animate things!

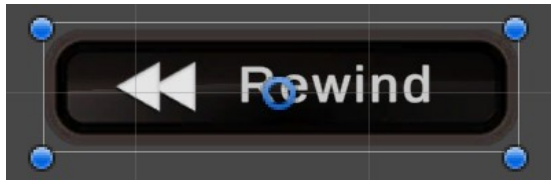
- The test will show your button become bigger over 30 frames, then loop back
  You might wonder : why does it become bigger slowly, but reverts abruptly?

- Scale is a numerical value. This is why interpolation occurs between frames 0-30
  However, when the clip loops back, it does not interpolate, which is why it's abrupt!

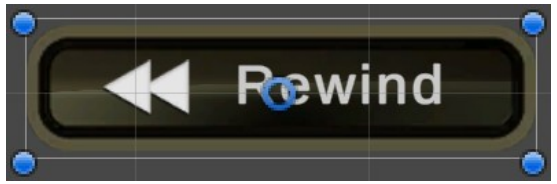
- Return to Record Mode, set yourself on frame 15, and edit the source image
  This adds another line to our animation, as it now affects another property : Image.Sprite



- If you test the animation again, you can see that the sprite change is abrupt, even though it's not on the last frame *(where the clip loops)*
  That is because a sprite is not a numerical value : it can not be interpolated. So it's not!



The way my button is displayed when the white line (current time) is over frame 0

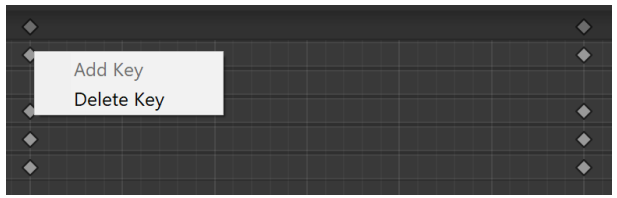


The way my button is displayed when the white line (current time) is over frame 16

While testing the animation, you can see the scale being interpolated in real time!

Manipulating Keyframes

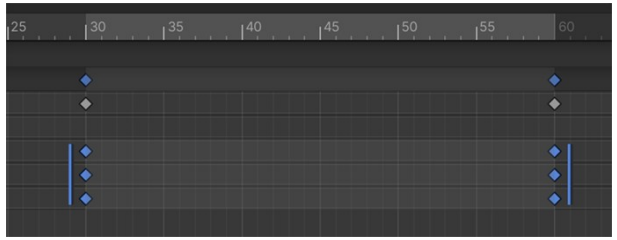
- Keyframes can be created through Record Mode relatively easy
  ...But once they are created, they become entities you can manipulate individually



- Keyframes can be drag-and-dropped to change the frame they occur on
  This will affect the timing of the animation. The Interpolations will adapt to the change

- Right-click will open a small menu to either add a Keyframe or delete it

  Adding Keyframe : the right click must be on an empty frame belonging to a property line
  Deleting Keyframe : simply right click on a keyframe on you will have the option to delete



- You can also select several frames at the same time!
  Either by drawing a selection box by holding left-click, or by holding CTRL and clicking many

- There is no graphic prompt, but it is perfectly possible to copy-paste frame(s)!

  Copying Keyframe(s) : press CTRL + C with one or many keyframes selected
  Pasting Keyframe(s) : after moving the white line to a different time, press CTRL + V

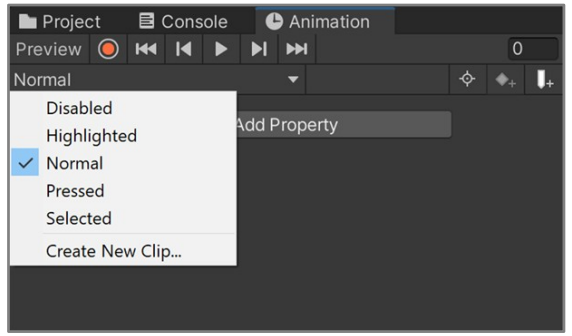
Now, it's your turn!

- Edit your animation so it loops properly, with no “abrupt jump-cuts”. *(Copy-Pasting will help you)*

Creating the 5 Animations for our Button

- Animation can be pretty confusing for newcomers. One has to think “through time”

- The only way to grasp those concepts is to practice using them!
  …And we are going to take some time to do that by creating our 5 animations!



Thanks to the auto-generation, the 5 clips are there. Ready to be edited!

Now, it's your turn!

- Fill the pre-existing 5 animations, for the 5 states of the button
- I recommend you manipulate at least Scale and the Image Source
- But you can try to animate other properties too!
- Once you have your 5 animations, test the app to see them in action!

Animations with 3 “steps” are good enough
No need to make more complex ones!

- This should take you a bunch of time, but it’s because you’re new to this!

But, Good News! : if two GameObjects have the same structure, they can share animations!
(and we’re going to use that to quickly animate the other ones!)

Recycling Animation : It has Conditions!

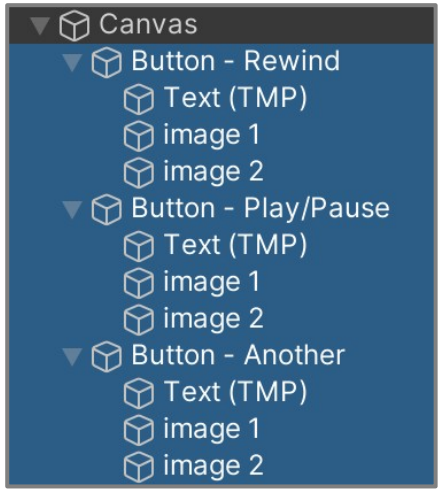
- Good job! You now have a fully animated button! With all 5 states.
  But we have at least 2 other similar buttons in our app… do we have to redo it all?

- The good news is that animations can work on entities that may look entirely different, as long as they share similar internal structure!
  This ability to "recycle" is one of the main reasons computer animation so powerful

- "Similar Internal Structure" actually means :
  1) Components manipulated by the animation exist in both objects
  2) Children accessed by the animation must have the same name
  3) Components manipulated on the children must also exist in both objects

- It's possible to have objects that respect those conditions, and yet look different
  Look to the side for an example of this. The buttons have "similar structure", but look different

Those buttons look different, but
They actually have "similar structure"!
(as you can see below)



Now, it's your turn!

- IF NEEDED, modify your buttons so they have "similar structure" with the one you animated

Applying our Animations to another Button

- Once our buttons have similar structure, we can use the power of recycling!
  Make sure the children and the components, which are part of your animations, are the same

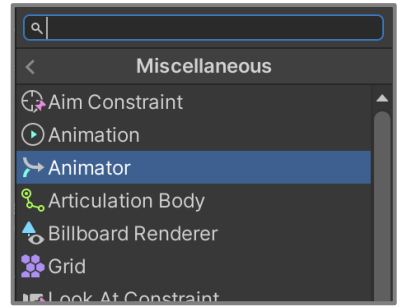
- Once you're good, set the other buttons to "Animation" transition mode
  However, DO NOT CLICK "Auto Generate Animation" after that!

- We'll do it manually : first, add an Animator Component to the GameObject
  Just like with any Component, you can do this with the "Add Component" button

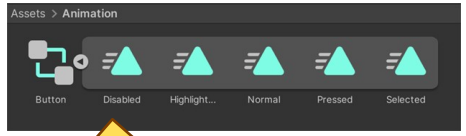
- The animator expects something called a "Controller". But we do have one!
  Remember when we had to save a file when creating our first animation? It WAS the controller!

- The controller can be seen in our Project Tab, if you're looking at the right folder
  If you listened to my advice, it should be an "Animation" folder. Sort your projects! ;)

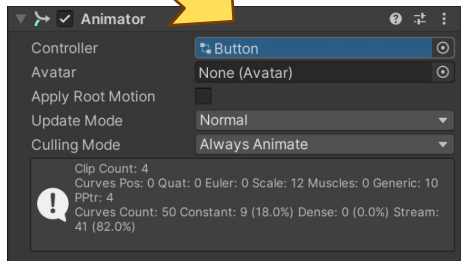
- Drag-and-Drop the controller right into the controller field of Animator
  And, there we go, we have transferred the animations of our first button to another one!



Animator is in the "Miscellaneous" Category, which can be confusing ...you can also find it with search!



Drag and Drop!



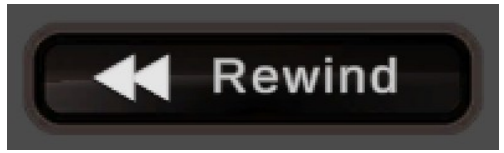
Any Problems?

- Test the app. Try the button we just edited. Is the animation working as expected?

- If no, it can mean one of two things :

  1) There is a disparity between the internal structures of the buttons
     Ex: A component manipulated by the animation doesn't exist in the new button

  2) Your animation edits a property that is different between the buttons
     Ex: Your animation edits the scale, but the initial scale of the 2 buttons were different



- Once it works as expected, you have made another big step in understanding Unity!
  Don't rest on your laurels, though. Animations can get much more complex than this!

- Becoming a really good animator requires lot of experience. It's a real job!
  In the industry, there are dedicated animation specialists for 2D and 3D. And even them often use the help of Motion Capture or other tools to help them!

- But our Buttons are simple enough, you can definitely do it!

Now, it's your turn!

- Apply the animation controller to all the buttons, and check this causes no unexpected, weird visuals

What's this "Animator Controller", anyways?

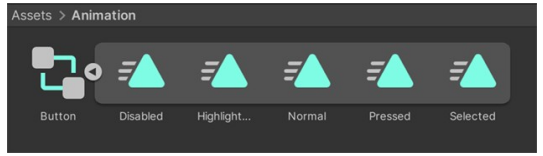
- By assigning the same "Animation Controller" to all buttons, we have given them the same set of animations. ...But what is an "Animation Controller", exactly?

- You probably think it is a "box" that contains a set of animations?
  You guessed right, it is! ...But that's also only about a third of the answer!

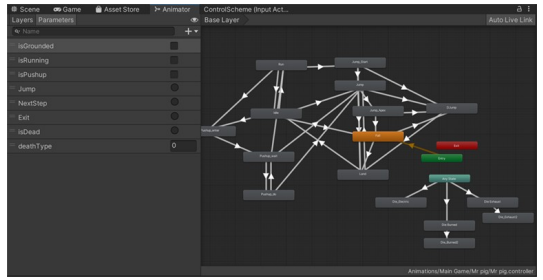
An Animator Controller is a "state machine". Each state having an assigned animation. The Controller also stores the possible transitions between those states, and the conditions required to trigger those transitions.

- That's a lot to swallow at once. Don't worry, we will see all that through examples!

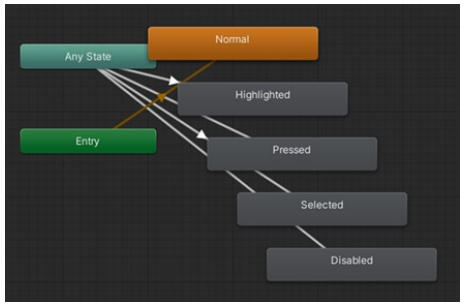
- To do that, we are going to reverse-engineer the controller that was created automatically for the button, and that we used until now!
  The fact that "auto-generate" can create a controller suited for buttons, automatically, that's the very reason we used buttons as our gateway to the world of animation!



When we look at that, we sure see a "box" containing animations inside



States, connected by transitions. That's an animator Controller!



Let's Analyze the Button's Controller

- To analyze a controller, you have to display the "Animator" Tab
  It's not the same as the "Animation" tab we've been using. Don't confuse them.

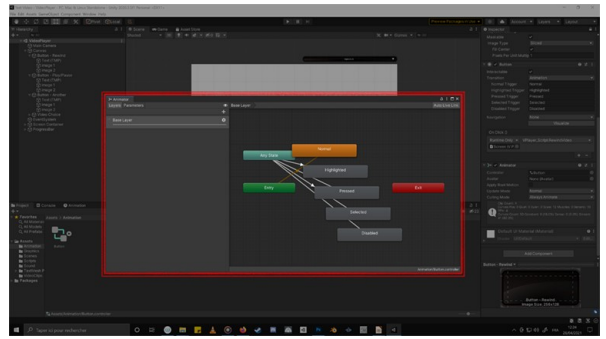
- To activate the tab : *(Window=> Animation => Animator)*
  Personally, I recommend placing this tab near the "Scene" and "Game" ones. But do as you wish!

- This Tab will always show the last controller you interacted with
  This includes "indirect" interactions such as selecting a GameObject carrying an Animator

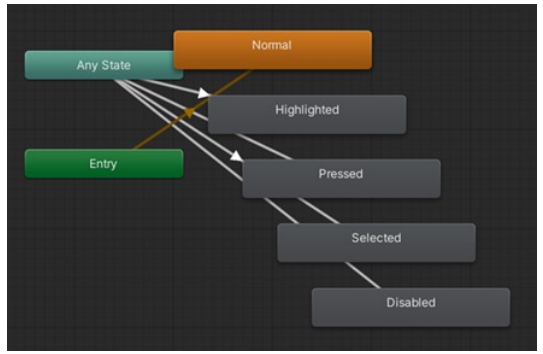
- Since your project should have only one controller, it will be on display
  You should see something like the pictures on the side here =>

- You can see there are 5 states, which have the same name as the Animations
  "Normal", "Highlighted", "Pressed", "Selected" and "Disabled".

- There are also three special states : "Entry", "Any State", and "Exit"
  Those states do not have animations associated to them. They exist for logic purposes



When you open it for the first time, the tab might appear in a separate window
Like any tab, you can move it and stick it In an existing window as well



The Animation Controller of our buttons should look like this

The States of a Controller

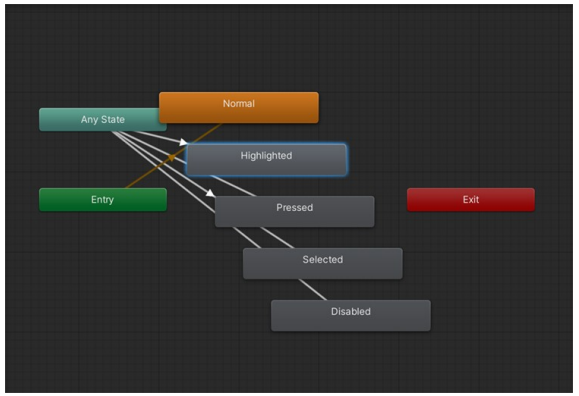
- Controller States are represented by grey rectangles with a name on it
  Some have special colors, but they are not normal states with animation on them. Except...

- The Orange State is the default state, in which the animator starts by default
  It will always be connected to the special "Entry" state, which is where the animator initializes

- "Any State" is used to create transitions that will trigger from ALL the states
  You can notice that the controller we're looking at uses it as a start for all its connections!

- The "Exit" state will cause the Animator to reset once reached
  It is almost useless, at it will cause the animator to loop back to "entry" status

- States contain a bunch of properties. Select a state to see them in Inspector
  For now, we will only care about three of those settings:
  1. The name of the state, on top. By default, it's the same as the animation
  2. "Motion" contains a reference to the animation that the state uses
  3. "Speed" controls the speed at which the animation will be read by the state (1 = 100%)



Among all the "colored" states, only one is a "normal" state with an animation : the Default state, displayed in Orange

The properties we care about in the Inspector. The others are meant for more advanced users and we'll ignore them

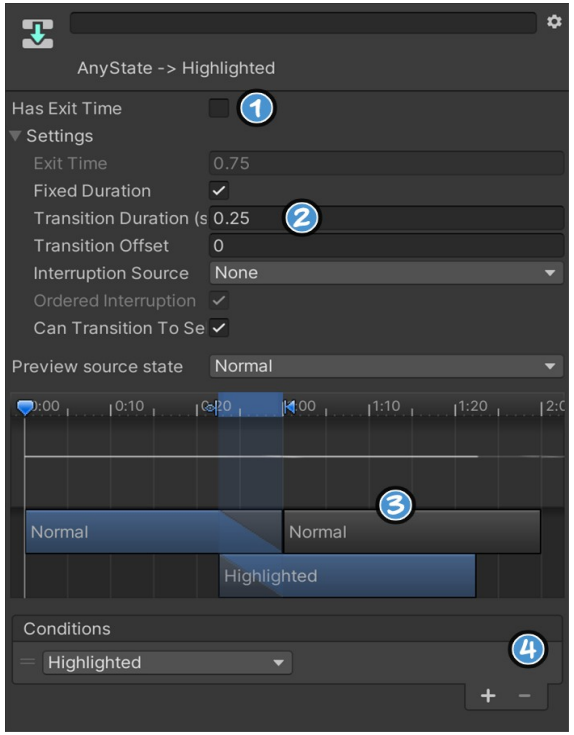
Transitions between States

- The Main reason the Animation controller exists is for the Transitions
  Transitions are “paths” between states, which can be controlled very accurately

- The Transitions are represented by arrows between the states
  A transition only goes in 1 direction. So if a transition has an “inverse”, there will be 2 arrows

- Select an arrow so we can see its properties in the inspector
  And you should get the pretty daunting display on the right! It’s a bit scary at first…

- Trying to understand all the subtleties of this system today would be crazy. So we will again focus on a few simple properties you can understand now:
  1. ”Has Exit Time” determines if the start animation has to complete at least X% of its duration before being able to transition out *(its disabled on all transitions of our example)*
  2. ”Transition Duration” determines the duration of the transition, in seconds. It works differently if “fixed duration” is unchecked. But please keep it checked for now
  3. This area gives a timeline with the start and end animation, and the interpolation between
  4. The Conditions determine when this transition will trigger. They use specific variables.



So many properties, hidden in a single arrow! As beginners, we will only care about a few of them, as usual!

Wait… Animator has its own Variables?

- Yes. They are called Parameters, and are used as basis for the Condition system.
  There are only 4 types available for parameters : **float, int, bool** and **Trigger**

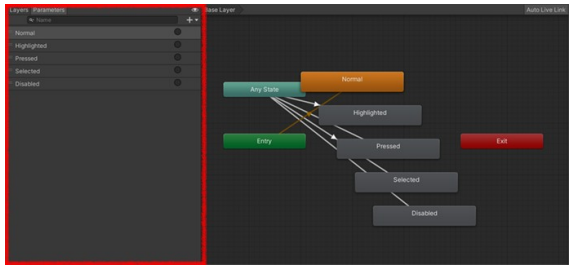
- The parameters can be seen in the left part of the animator/window tab
  You **have to click the "parameters" button on top** to switch to this display, of course

- In the case of the Anim. Controller of a button, all the parameters are TRIGGERS
  Triggers can be seen as **"booleans which are only true for one frame, and then revert"**

- Select a transition once again and look at the Condition : it's a trigger!
  The button Component "fires" those triggers automatically as we interact with the button
  This is what causes the animator to change state, and play our various animations!

- Trigger lasting only one frame, it is useless to store a permanent status
  This is why the other type of parameters exist. Parameters are checked every frame
  so any change to a parameter might trigger a transition

- The parameters can be controlled by code relatively easily
  They are the best bridge between the animator controller (GUI) and our Scripts (code)



Parameters are in this area on the left
(This area can show "parameters" or "layers", so change if needed)

The Controller of our button has 5 parameters, and they're all triggers!
(the circle on the right is a visual indicator)

Triggers and Conditions for Transitions

- The Parameters are used for creating the conditions of a transition
  A transition can have several conditions. In that case, they must ALL be fullfilled

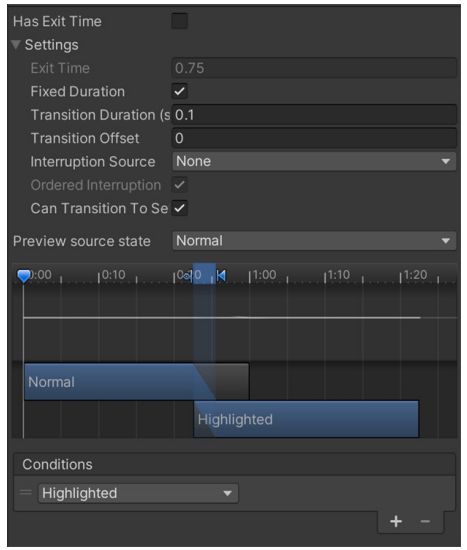
- A transition can also have zero conditions. It will then trigger as soon as possible
  Useful if a transition has an "exit time" : in that case, the transition will trigger as soon as the exit time is reached *(exit time can be seen as a "special" condition)*

- To create a parameter, click the "+" in the top right of the parameter subwindow
  When you do, you will have to choose the type of the parameter, and then name it

- To create a condition, click the "+" at the bottom the condition list
  You can only create conditions with existing parameters. So create the parameters first!

- Try creating parameters and conditions now, but make sure to revert afterwards!
  You can delete a parameter with right-click, and a selected condition with the "-"
  If you mess up anything, remember you can cancel any action with CTRL+Z!

The Parameter Subwindow for a button, with 5 triggers



The settings of a transition. The condition list is at the bottom

Why is it all so Complicated !?

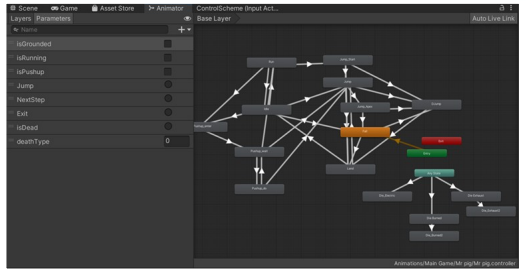
- Despite all this information, we have barely scratched the surface of Controllers!
  Animations! States! Transitions! Parameters! There's so, so much stuff!

- The system is complex because it was designed to give the user COMPLETE control over advanced features such as animation blending
  The system is very powerful and feature-rich, but that's what makes it complex!

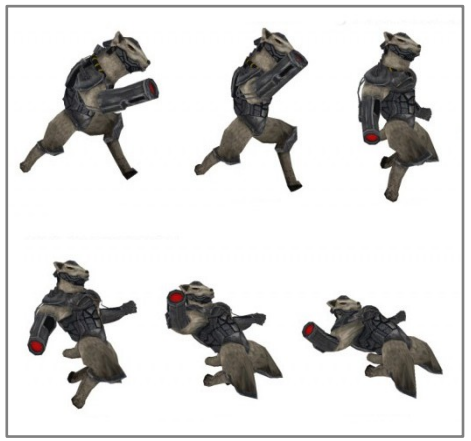
- It's a rule of computer software *(and reality in general!)* :
  the more options and control a system offers, the harder it is to understand!

- Fortunately, like often in Unity, it is possible to ignore a lot of the settings of Animator and focus entirely on its core principles : it's enough to be able to do things!
  But there is no escaping states, transitions, and parameters. Which is why I wanted to talk about them a bit before we begin making our own Anim. Controller

- Because, yes, we are going to make our own Animation Controller now!



An advanced animation controller can look pretty intimidating, BUT...



...It can allow neat stuff like this!
Here, the computer seamlessly interpolates between a run and a slide animation, Making the movement continuous and look much more natural and awesome!

Let's learn by doing our own Controller!

- We are going to create a custom controller, for something else than a button!

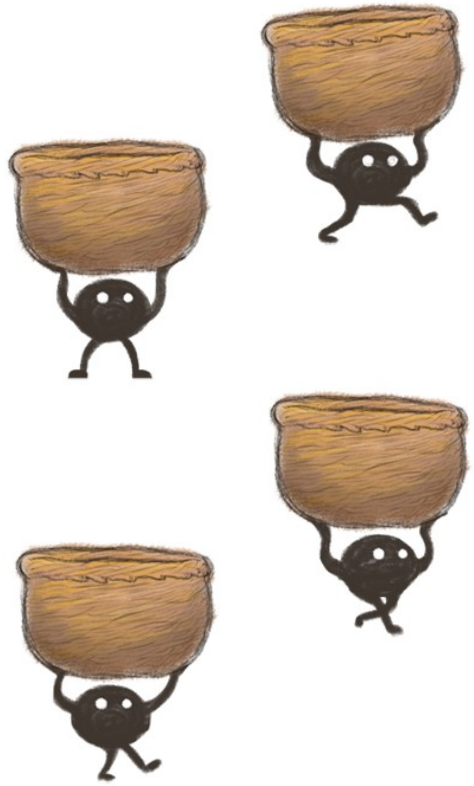
- The Video project isn't really suited for that… so let's go back to "Apple Catcher"
  Make sure you save everything before closing the Video project. Then use the hub

- Check Moodle : you have a zip file of pictures for this session
  Drawings of a character holding a basket… You can probably guess where this is going

- Yup. We are going to replace the static basket from before with this character!
  OK. That character isn't high art, but it's still a major upgrade from a static basket!

- We are going to animate this character so it matches the input of the player
  To do that, our code will influence the parameters, which will change the animation state

- There are 3 animations in the pack : "Idle", "Walk" *(walk right)*, and "Rear" *(walk left)*
  From there, we have to think what transitions and parameters we need!



Analyzing what we will need...

- We have 3 animations : “Idle”, “Walk”, “Rear”
  => We will need at least 3 states. One for each animation

- When the game starts, we can consider the player is not moving
  => The default state in which the Animator will start should be “Idle”

- From “Idle”, the Player can start moving left or right as he wants
  => We need two transitions from idle, one for each “Run” animation

- The player can quickly shift from moving left to moving right
  => We need a transition between “Move Left” and “Move Right”

- The Player can stop moving, returning to the “Idle” status
  => We need to be able to return to “Idle” from any other status

The Walk animation is a 7-frame cycle
Play the frames in order, then loop back to 1

The other two use a “1-2-3-2” cycle
In a cycle, the drawings are played forwards, then backwards

Now, it's your turn!

- Can you see the parameters/triggers this animation controller will need?
- Is there only one possible design that can fulfill those objectives ?

Abusing “Any State” VS True Structuring

- Some of you might have suggested a design inspired by the button. Like this :

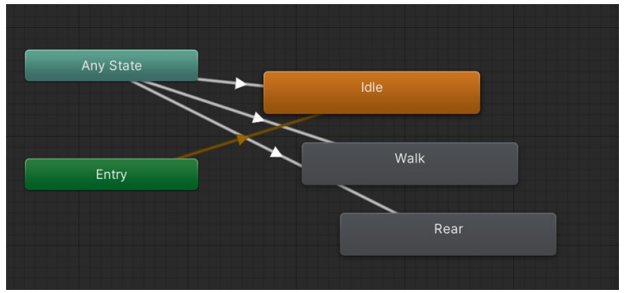
- And it will work! …But the thing is, it might give you a bad habit for later!
  Making all the transitions start from “any state” only works well for simple Controllers
  If the controller gets truly complex, with 15 or more states, it is a bad idea

- Our controller only has 3 states… But I want to give you good habits right away!
  So we are going to place transitions logically, with as few as possible using “Any state”

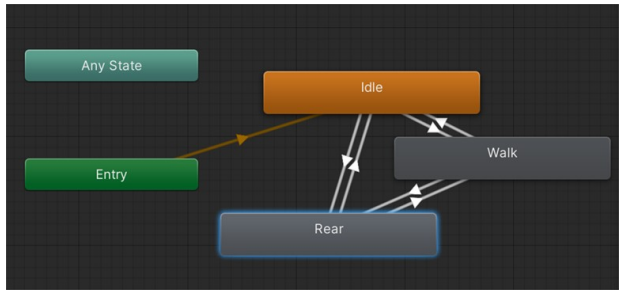
- Doing so will result in something that looks more like the bottom version :
  In the Unity community, using “Any State” as little as possible is considered “the right way”

- Regarding parameters, there are several solutions as well
  We could use a “IsMoving” bool, an integer that codes for the direction of movement…

- But since Triggers is the only parameter type you have some familiarity with…
  …We will use them!



Using transitions like this is what the button does. It’s fine when the animator has few states, but quickly becomes unmanageable on complex controllers



Which is why I would rather have you connect the states directly between them, using “Any state” as little as possible

Creating an Animator and a blank Controller

- Choose the Basket GameObject, change its default sprite and name to reflect that it is now our character : Catchboy!

  You will likely need to adjust position, scale, and colliders too

- Use “Add component” to add an Animator Component to Catchboy

  Animator is in “Miscellaneous” category, but you can also make a search to find it

- We are going to create a blank animation Controller to use. To do that :

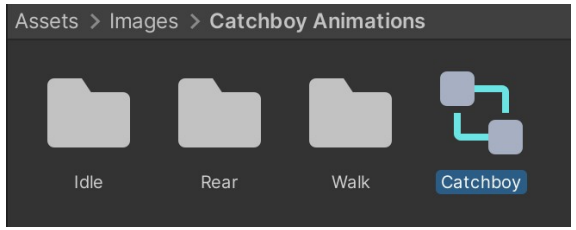
  Right click in “project” tab => Create => Animation Controller

- Assign the controller we just created to the Animator Component

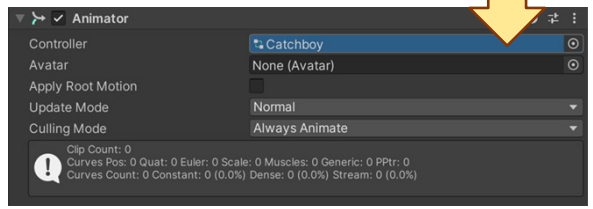
  As usual, this can be done by drag-and-dropping the animator in the field

- Now that the controller is assigned, we can create Animations on the GameObject!

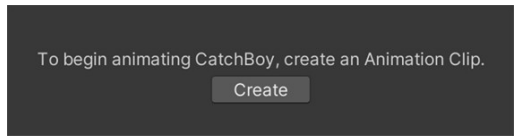
  If you attempt to use the animation window before you made this connection, Unity will offer to create a controller automatically



The controller just after its creation. Which you drag...



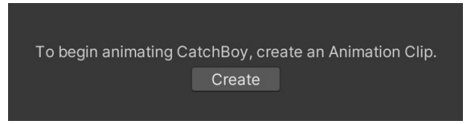
And drop in the “controller” field of the Animator Component!



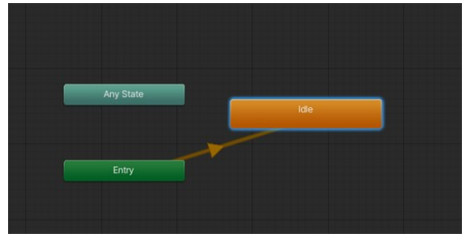
You will see this in the animation tab, as we have a controller, but no animation!

Adding new Animations and States

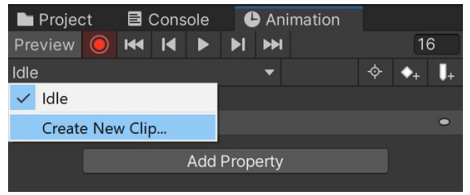
- Make sure that both the Animation tab and the Animator tab are enabled
  We will have to use both, and we'll also see how they interact together



- In Animation Tab, Unity prompt you to create a first animation clip. Do it and create and Animation named "Idle"
  The animation is a file. Unity will ask you to name it and save it somewhere in your Asset Folder
  Remark : we can not put the animations "inside the controller". That only works for buttons

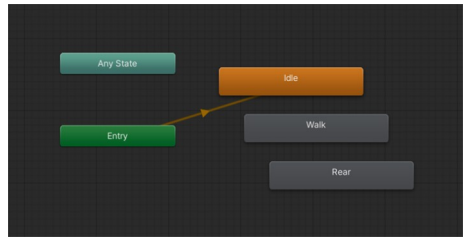


- Notice that when you create that first animation, a state is added in Animator tab
  And since it is the first state available, it's also selected as the default state



- Edit the idle animation so it has the speed and feel you want

- To add another animation, use the animation list in the top left of the animation window, and pick "Create new clip"



- Creating a new animation Clip will add another new state, and so on

Adding the Transitions we Want

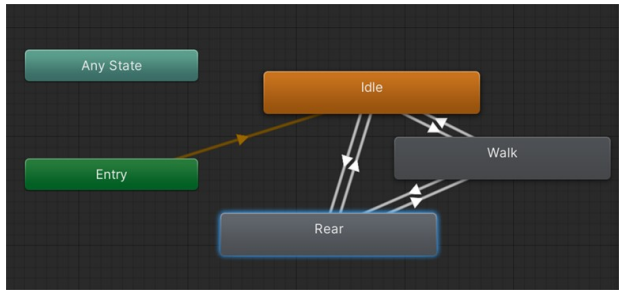
- Now, we work within the “Animator” tab. Begin by creating 3 triggers
  Those are created in the “parameters” subwindow, by clicking the “+” you have there

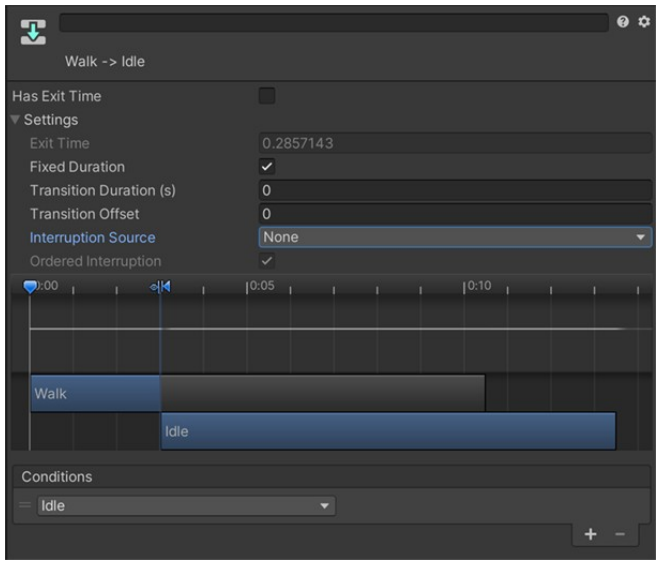
- Name the triggers “Idle”, “Forwards”, “Backward”.

- To create a transition, right click the STARTING state and choose “Make Transition”.
  An arrow will then follow your mouse. Click on the ARRIVAL state to connect

- Use this to create the network of transitions you can see on the side

- Edit ALL transitions in the following way :
  1) Disable “has exit time” so the transition may trigger without waiting
  2) Set “Transition duration” to 0 with “fixed duration enabled”
  3) Put the trigger matching the transition as a condition for its activation





Manipulating the Animator's Triggers in C#

- For that we need to access the Animator Component in the code
  Using GetComponent<Animator>() is what makes the most sense here

- To fire a Trigger, use the SetTrigger(name) method on your reference to Animator
  The name of the trigger is provided as a string. As usual, it's case-sensitive!

- When the triggers are fired, they will launch transitions, changing the state, which will change the animation accordingly!
  Phew! As we said, this is pretty complex. But power comes at this price!

- It's not a good idea to fire Triggers every frame, so you'll have to improve your code…
  A good solution is to remember the movement speed of the previous frame, and only fire the trigger if the movement speed of the current frame is different!

- This is where we finally test a lot of previous work. So if you have a bug, remember that it might come from your transitions and/or animation clips!
  This is the toughest tutorial yet. So don't hesitate to ask questions!

```
ref_animator.SetTrigger("Backwards");
```

Example of use for SetTrigger.
ref_animator is obviously where I stored the result of GetComponent<Animator>

```
void Update()
{
    float newSpeed = 0;

    if (Input.GetKey(KeyCode.LeftArrow))
    {
        newSpeed = -10f;
        if (newSpeed != previousSpeed) { ref_animator.SetTrigger("Backwards"); }
    }
    else if ( Input.GetKey(KeyCode.RightArrow) )
    {
        newSpeed = 10f;
        if (newSpeed != previousSpeed) { ref_animator.SetTrigger("Forwards"); }
    }
    else
    {
        if (newSpeed != previousSpeed) { ref_animator.SetTrigger("Idle"); }
    }

    //Move with the speed found
    transform.Translate(newSpeed * Time.deltaTime, 0, 0);

    //As update ends, the new speed becomes the previous one
    previousSpeed = newSpeed;
}
```

Here is my version of the Update() method used on Catchboy.
You can use it as inspiration, especially if you're running out of time!

Good job, you can now Animate !

You can now bring any sprite to life, with a little bit of effort!

(As usual, we actually only scratched the surface of this incredibly powerful tool. But, as usual as well, you already know enough to create really neat things!)

Next time...

Coroutines and other Upgrades

We'll do things we already know how to do, but better!