LP2B – Multimedia: Digital Representation

- LP2B Tutorial Class n°4 -

UI Canvas and Mouse Interactions

And a Glimpse into Event-based programming in Unity

This class will give you an overview of :

- Creating a 2D UI Canvas -
- Using the existing components for buttons -
- Event System and its mouse-related events -
- Sliders, drop down lists, and other UI elements -
- Detecting clicks on GameObjects that are NOT in canvas -
- Pros and Cons of using Canvas. Should you do it? -

Lets go back to our Video Player...

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

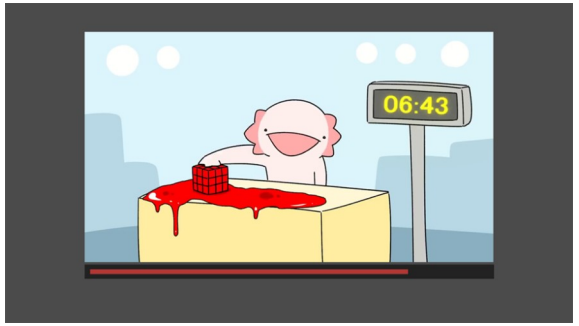
- Remember? We had a video player with a few controls programmed

- However, all those controls are bound to the keyboard!
  Memorizing tons of keyboard shortcuts is kinda inconvenient, don't you think?

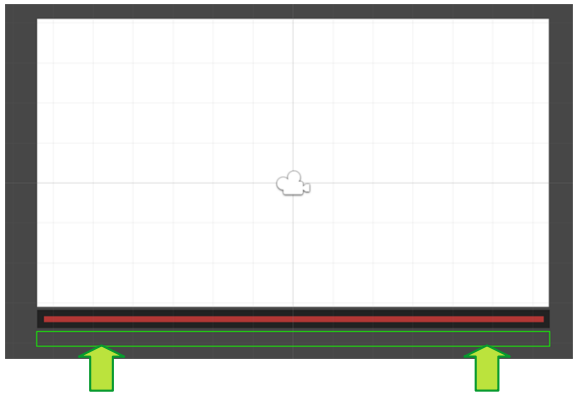
- To feel like a true video player, our project could really use some buttons!
  Those buttons can even remind the user what the key bindings are. Like in a pro app!

- In order to have buttons and other user interface (UI) elements, we need to care about an entity that we neglected until now : the mouse

- So, how can we detect when we click on things ?
  As a video game engine, Unity probably has components to do this. Right?



The state we left the project in (roughly)



Having some buttons down there would make our player easier to use, and feel more professional as well!

Introduction to Canvas

- Unity does indeed come with a lot of pre-made UI components.
  Buttons, sliders, drop-down lists… you name it! But there's a catch :

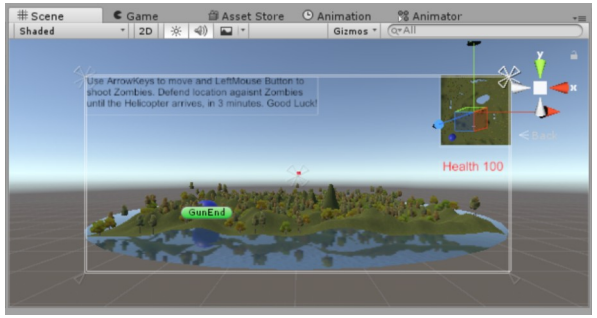
- Because of Unity's history as a 3D engine, its native UI components do not work alone : they must be put inside a specific space called a Canvas

- The world of Unity is, in fact, always 3D. Even with the "2D" template
  The illusion of 2D is created by the use of an "orthographic camera", which has no perspective. You will learn more about it if you pick classes about 3D imaging!

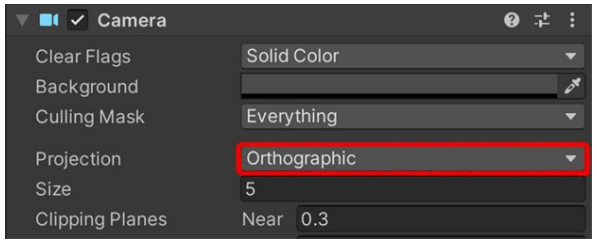
- Unlike the general world, a Canvas is true 2D space. Meant for UI.
  The original reason for this system being : the need for a 2D UI over a 3D world

- In theory, a 2D app could be done entirely within a Canvas
  A Canvas being true 2D space, it should be able to handle images and everything

- Wait, What ?! So why didn't we use Canvas from the beginning?!



A Canvas in use in a 3D game (and in an older version of Unity)



If you look at the camera that is automatically set for the scene, you can see it is indeed "orthographic". This is because we used the 2D template.

All 2D doesn't belong in a Canvas

- Canvas comes with restrictions and quirks. It's not good at everything

Some of the Pros

Gives Access to pre-made UI Components
(that's the really big one!)

Special position system which allows perfect management of changes in aspect ratio

Adapts automatically to the Camera

Will handle mouse events almost automatically

Can be used for a 2D overlay over a 3D world
(we're not doing it, but it's the original reason it exists)

Some of the Cons

Many Features do not work within Canvas
(including our friend the Physics Engine!)

Displaying elements requires more processing. Especially if there are a LOT of entities in Canvas

Harder to understand and configure

Sometimes tricky to synchronize with the "world"

Cuts a part of your app's rendering from the rest
(can be a problem for some graphic effects)

Creating a Canvas

- The best way to realize those pros and cons, is to start using Canvas!

- <u>To create one :</u> *(right click on hierarchy => UI => Canvas)*

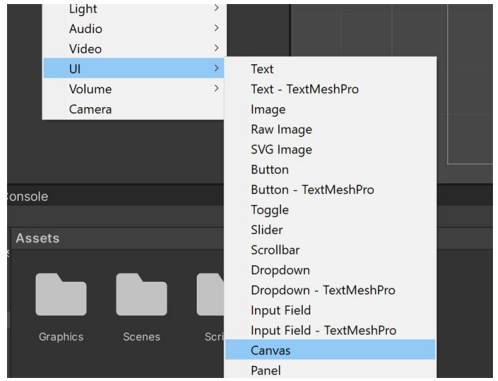
- As soon as you create a Canvas, you also create an “Event System”
  Both are GameObjects that have SEVERAL components attached

- <u>In Short :</u> “Event System” is what handles mouse events. “Canvas” displays.
  “Event system” can also handle the mouse in the “world”. But we won’t talk about that

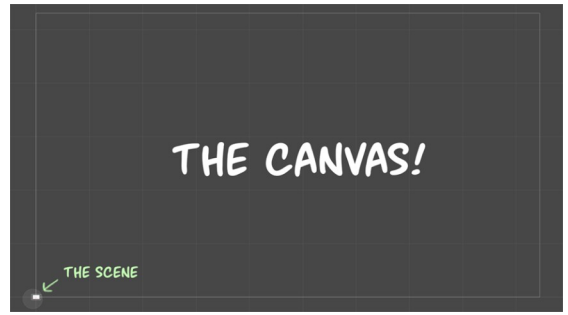
- But… where is our Canvas? Is it meant to be invisible?!
  No. it’s quite visible. Try zooming with the mouse wheel and you’ll see…

- See that GIANT white rectangle? That’s our Canvas!
  Yes. It’s that big. Because it uses pixels as its units, rather than normal coordinates!

- Fortunately, we can configure the Canvas to make it less cumbersome!



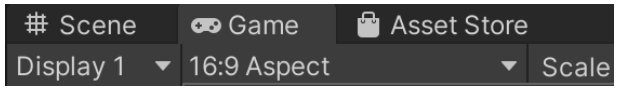
The Path to create a Canvas



On this screenshot, you can see how big the Canvas is relative to the scene! At runtime, it will still adapt to the Camera, but it’s inconvenient to have it being like that!

Setting the Canvas to match the Camera

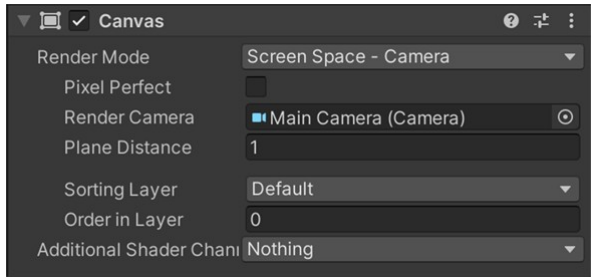
- If it wasn't done already, set the Camera/Game aspect ratio to 16:9



Reminder : the setting for the aspect ratio
Is in the "Game" tab (near the "Scene" tab)

- Select the Canvas GameObject in Hierarchy, so we can see its components

- In the Canvas Component, there is a "render mode" option
  Set it to "Screen Space – Camera", this will cause the canvas to match a Camera

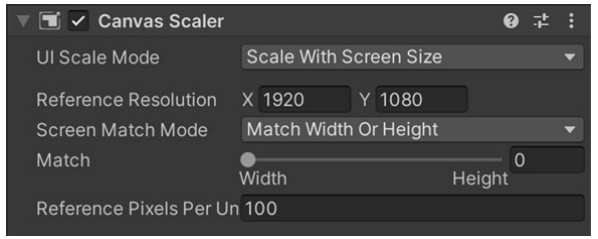


The settings you should use, this time,
for the "Canvas" component

- This mode requires a reference to a Camera in order to work :
  Drag and Drop the "Main Camera" GameObject into the field "Render Camera"

- In "Canvas Scaler", ensure the Canvas is set to match the screen size
  There are a lot of settings. When in doubt, copy what you can see on the side here

- The Canvas always stays in front of the Camera it is assigned to
  The distance between the Camera and Canvas can be set. Choose a short distance to ensure the Canvas is in front of your other objects (but not 0, or the camera culling will cut it!)



The settings you should use, this time,
for the "Canvas Scaler" component

- Now that the Canvas matches the Camera, it will be MUCH easier to use!

Creating our First Button

- Now that we have a Canvas, we can finally use UI components, like Button!

- We are going to use the “TextMeshPro Button” :
(right click on hierarchy => Create => UI => Button Text Mesh Pro)

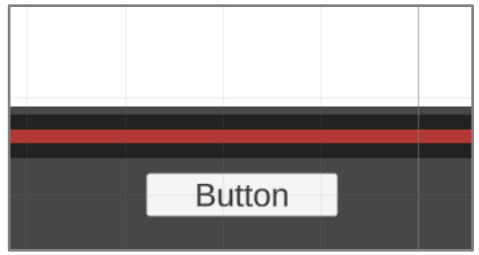
- The button should automatically become a Child of the Canvas we already have.
That’s because it simply can not function in another context

- You can show/hide the children of a GameObject with the little triangle
The triangle only shows up if a GameObject has at least one child, of course

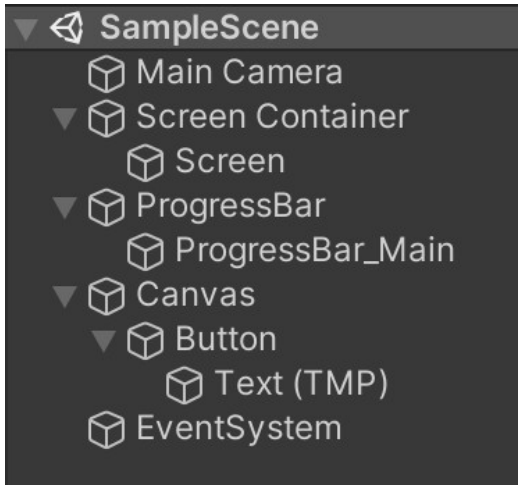
- Notice that the button we just created has a Child… the text it contains!
Considering that a Child is “inside” its parent is a good summary of what it means

- You get why the Tab showing the GameObjects is called “Hierarchy”, now?
It shows GameObjects, but ALSO how they are organized with parent-child relations

- But let’s go back to our Button…



The way the button looks just after its creation. On our Scene



The Hierarchy of my version, completely unfolded. You can see the button is child of the Canvas, and the Text(TMP) child of the button.

Connecting our Button to our Code

- It is possible to directly set the method called by the button with the UI!

- If you look inside the Button Component, you can see an OnClick() area
  It is a customizable event callback. You can add methods to call there through the GUI

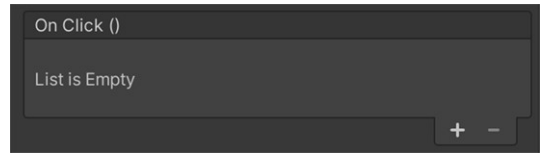
- Click the “+” button to add a field. It’s possible to have more than one
  A Unity UI Button can call several methods from different objects when clicked

- Drag_and-Drop the object carrying the script with the method you want to call.
  Here, we are going to call a method of our video screen. So drag the screen there.

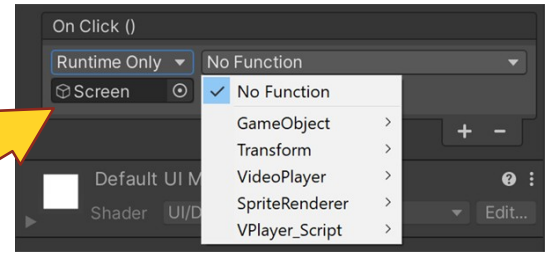
- Once the object is set, you can choose the method to call
  Click the right field, and all the methods (sorted by component) will be shown

- Choose a method you created yourself in “vplayer_script”
  Warning : only public methods are available this way. You might have to edit your code!

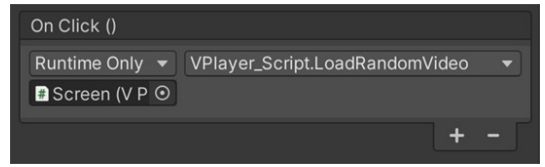
- Test the button. When clicked, it should launch the chosen method



The OnClick Area. Empty.



Once the GameObject is put in the left field, you can choose a component, and then a method or attribute inside that component



Setup done! Clicking the button will now call the “LoadRandomVideo” In the “Vplayer_Script” component (my custom script!)

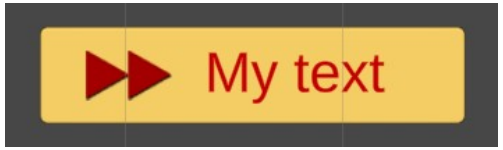
Customizing our Buttons

- Now that the buttons actually work, we can start caring about something less critical, but that our user will enjoy : their appearance

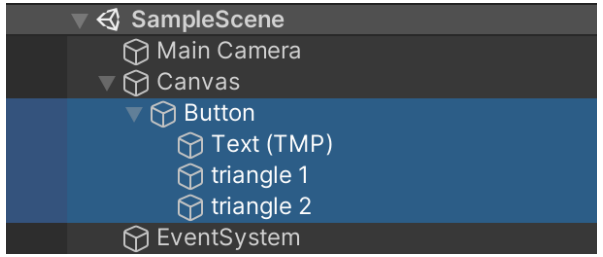
- First : you can edit (or even remove!) the TextMeshPro inside the button
  This text may be put inside automatically, but it's not actually mandatory

- Second : you can add other elements as child of the button
  The Children of the button do not affect its scripting. So you can add anything!

- Third : you can change the Image used for the background of the button
  The Button GameObject also has an "Image" Component. This is where it's at



A customized button!



Here is the Hierarchy, with the button and its contents highlighted. You see that I added stuff and modified the text box, and yet my button works!

Important :

- In a Canvas, 2D sprites are not managed by "SpriteRenderer", but by "Image". This allows the use of the Canvas special positioning system.
- If you look at the Inspector of a Canvas Object, you can see it has a "rect transform" for its position, which is very different from the regular "Transform"

Sprite Slicing and why you should care

- In Unity, a Sprite isn't just raw image data : there are settings added to that

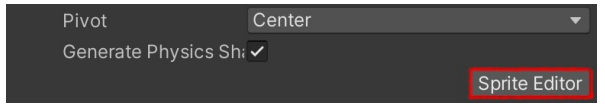
- Remember when we changed the pivot of a sprite? That's an example!
  The editing of most of this sprite meta-data is done through the Sprite Editor

- Resize a default button : notice how the corners do not deform?
  It is because the default sprite used for button is set to use Sprite 9-slicing

- Sprite 9-slicing allows us to define what area of your sprite will be deformed
  Once a sprite is sliced, only the middle zone will be distorted when resizing

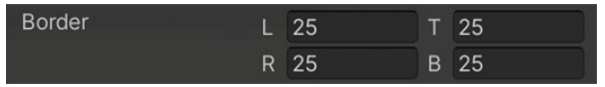
- In the Sprite Editor, you can edit the slicing by moving the green handles
  Doing so will move lines that will delimit 9 areas. Hence the name of the slicing!

- For precision setting, you can edit the position of the lines with numbers
  The numbers are distances in pixels from the top(T), bottom(B), left(L) and right(R)



To access the Sprite Editor, select an image asset in the Project tab, and search for this Button in the inspector

The area at the center will be the only one that gets distorted when the sprite is resized (the component that displays it has to use the "sliced" Image type for this to work)



This area allows you to define the position of the lines through numbers. Useful when several sprites need to be sliced the exact same way!

Now, it's your turn!

- Import the 3 button backgrounds and slice them in EXACTLY the same way

Button Transitions : Color Tint

- To understand that an area of the screen is a button, the user needs FEEDBACK
  If nothing happens when the cursor goes over a button, the user will not get it's a button!

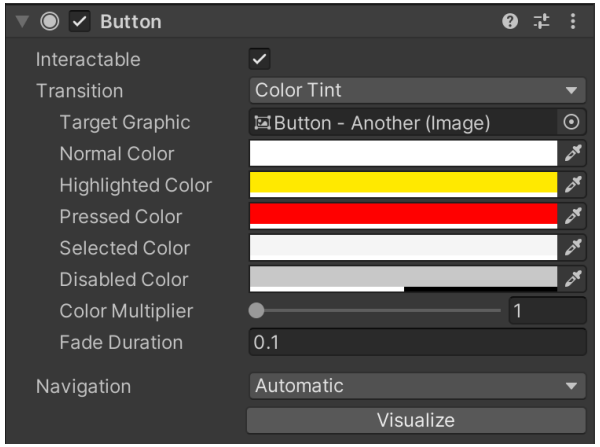
- The Button component comes with 3 premade settings to easily create feedback
  There's "Sprite Swap", "Animation", and the default we care about here : "Color Tint"

- Color Tint allows you to apply a color MODIFIER for each of the different states
  The colors chosen ALTER the original color depending on state, they do not replace it

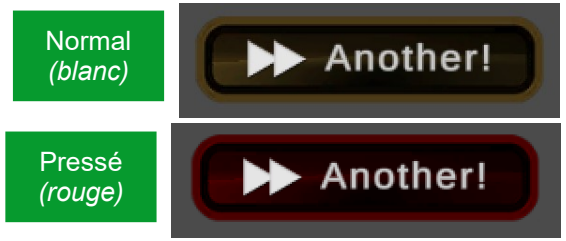
- Because the modifier is a "multiply", it works well for buttons with light colors
  A Unity default button is white, so it works wonders on this button, for instance

- For darker Buttons, however, "color tint" often gives very poor results
  The "color tint" mode is the simplest to use, but it also has the most limitations

- Because we want to use a dark interface, we are going to rely on another mode...



A Button set in "Color Tint" Mode
Because the colors are MULTIPLICATIVE,
it's not always as obvious to use at it seems



This button is the one matching the inspector above. You can see that it does become slightly red when clicked, but not that much.
It's because my sprite is very dark, so a multiplicative operation does not do much

Button Transitions : Sprite Swap

- "Sprite Swap" is based on changing the sprite used as background of the button
  To do that, you will need several similar sprites. ...Like those we imported before!

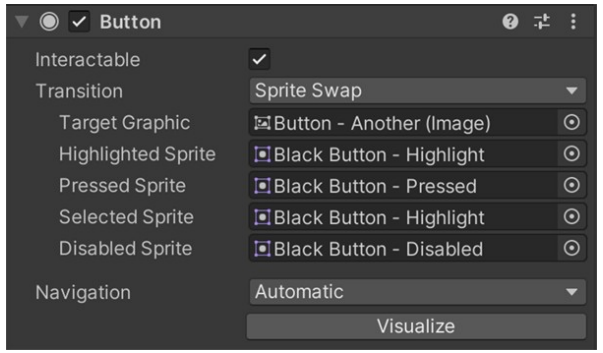
- When you change into this mode, the Inspector requests the Sprites to use
  We are going to use the sprites that we "9-sliced" before. They are meant for this

- The Image modified by default is the one attached to the Button GameObject
  Image being a component, it can coexist with other components on the same GameObject
  The Image component altered by Sprite Swap can, if you want, be modified

- Drag and drop the sprites in the fields in order to set up the appearance for the various interactions : Highlighted, Pressed, Selected, and Disabled
  A Sprite can be used for 2 different states : use the "highlighted" image for "selected" state too

- Once everything is setup, test the app. What do you notice after clicking a button?
  Wait... wasn't the button supposed to go back to default state after I move my mouse away?

- You can see the button stays "selected" after use. What if we don't want this?



A Button set in "Sprite Swap" Mode
This mode allows better control and more options than "Color tint". Its only weakness is that you need to create several sprites

This is how the button looks when Highlighted/Selected
...And it stays that way after I click and move my mouse away! What's going on?

Unity's Auto-Navigation : good or bad?

- The reason our buttons "stay selected after use" is because of auto-navigation

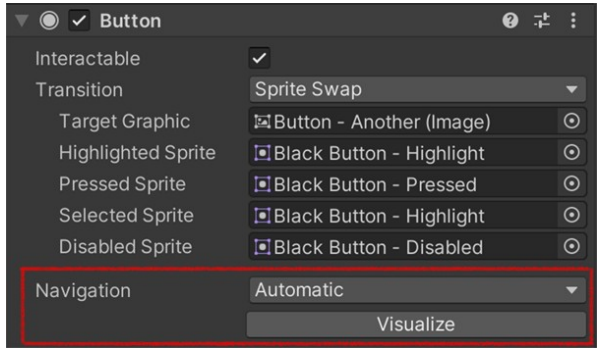
- It's a feature that makes our UI usable with all inputs, not just the mouse!
  This means the keyboard, but also gamepads, joysticks, and almost all inputs available for a PC

- "Auto-nav" creates invisible paths between UI elements, allowing key-based control
  This can function because the Paths are associated to a direction (up, down, left…)

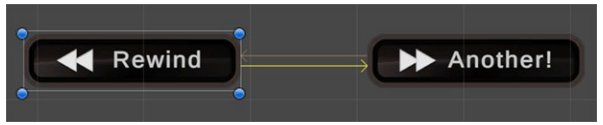
- In many cases, that's actually pretty neat. …But what if we want to disable this?
  You can keep "auto-nav" if you want, but it might conflict with some of our Keyboard shortcuts!

- To disable this, search the "Navigation" field of your Buttons. Set it to "none".
  You have to do this on all your buttons, or some will keep "auto-nav" behavior

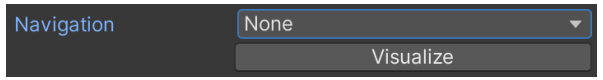
- Knowing auto-navigation exists is pretty important for debugging
  Without knowing, it's pretty easy to mistake a normal behavior for a bug



At the bottom of the Button Component, there is the "navigation" area. You can see that it is indeed set to "automatic"



If you click "Visualize", the invisible paths will be shown as colored arrows (you need at least 2 UI elements)



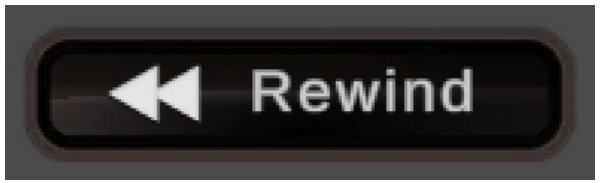
To disable this, set all your button's navigation to "none"

Now, make those Buttons glamorous!

- You now know everything you need to create all the buttons we want!

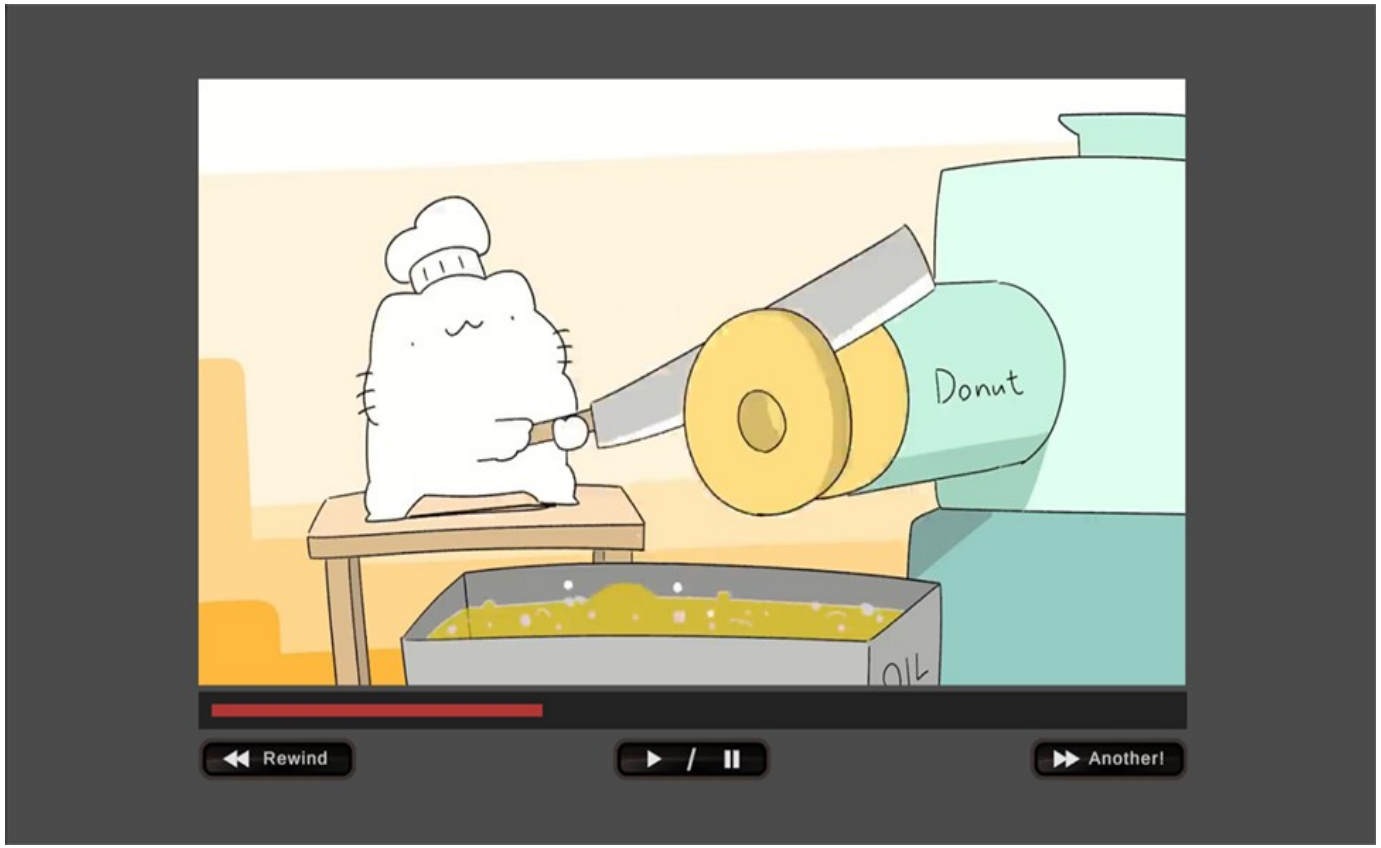
Now, it's your turn!

- Create 3 buttons : “Rewind”, “Play/Pause” and “Another!”
- Skin them with the provided sprites and the “Sprite Swap” transitions
- Connect them to methods in your code so they do the following things when clicked :
  1) “Rewind” returns to the beginning of the video
  2) “Play/Pause” toggles between pause and play
  3) “Another!” loads another random video from your array



- Note : we are not going to talk about “Animation” mode this time. Animations require the use of the Unity Animator component, which is very powerful, but also so complex, it would require a full tutorial devoted to it!

Finally! Our own, custom Buttons!



You now are able to customize your Buttons!

The fanciest option, animation, will be discussed in the next tutorial, which is all about the Animator component

But if you are able to create some images with graphic software, there already is a lot you can do with Sprite Swap and Color!

(If graphic creation isn't your thing, the Internet has a lot of free button templates you can use, too!)

The Drop-Down List

- The other UI component we are going to talk about is the drop-down list
  You can explore the other UI components on your own if you want to. For evaluation, we will only consider button, drop-down, and the notion of drag-and-drop

- We are going to use the "TextMeshPro Dropdown" :
  (right click on hierarchy => UI => Dropdown - Text Mesh Pro)

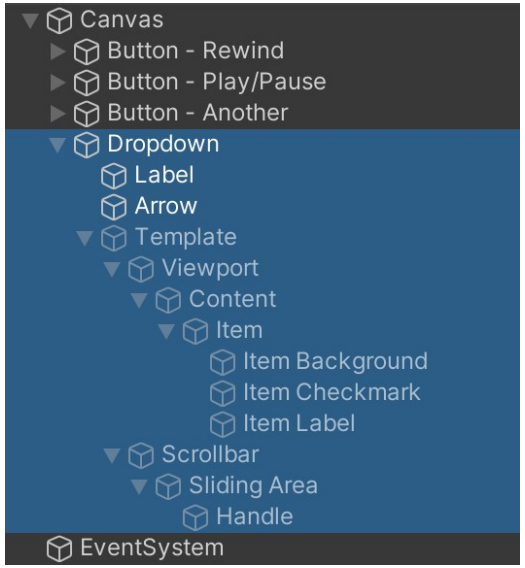
- Just like the button, our Dropdown will automatically become child of Canvas
  As a GameObject carrying UI components, it simply can not exist outside that context

- Like the Button, the hardest part with a dropdown is customizing appearance!
  Because in term of coding, it isn't actually much harder than the button we just saw!

- The fundamental logic of a Dropdown is that its various choices are numbered
  First choice is number 0, second choice is number 1, third choice is number 2… etc, etc…

- Unlike Buttons, dropdowns often need their choices to be generated through code
  We are going to see that in our coming example!

A Drop-down list just after its creation, with default appearance



As you can see, a drop-down is made of quite a bunch of hierarchy!
(that's why customizing its graphics is quite a bit more effort than for buttons)

Modifying the Choices of a Drop-down

- In our case, we want the dropdown list to show the videos in the array
  We want our user to freely be able to choose a video, and see it load as a response

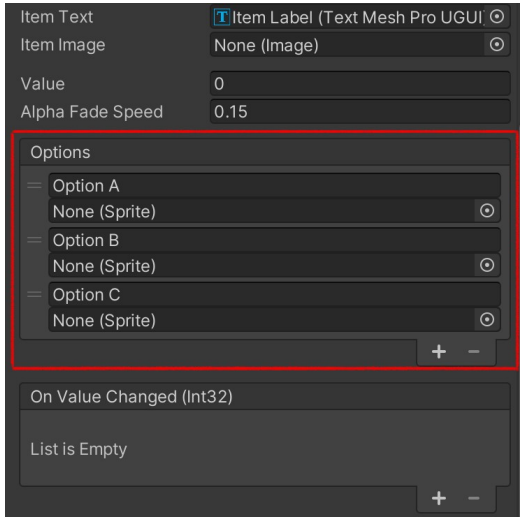
- To do that, we need to create the options of the dropdown through code
  Because our Array is in "Vplayer_Script', we are going to code this in that script!

- We need to give a reference to the Dropdown to our Script
  Type of the dropdown is TMP_Dropdown. To have access to it, you need to import TMPro

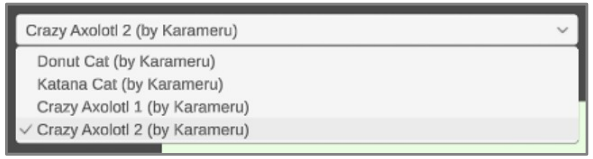
- AddOptions() allows you to add several options at once, as its parameter is a list
  Lists are fairly similar to arrays : they contain several items of the same type

- Because the Dropdown relies on numbering, choices in it are just text strings!
  However, the method AddOptions() takes a LIST of strings. So we have to talk a bit about that

- Another useful method is ClearOptions(), which deletes all choices in a dropdown
  Removing it all, and THEN redefine a new set of choices is often the right way to do things



This area allows you to define the choices/options of the dropdown manually. But doing so is only interesting if the options never change



But the big strength of drop-down is being able to modify the options dynamically, so we will look into that!

A few Things about Lists

- A list of elements of type "TypeToUse" is defined with List<TypeToUse>
  Any type can be used. However all items of a list must be of that type!

- The list must be initialized after declaration, with new List<TypeToUse>()
  This can often be done on the same line as the declaration, as you can see on the side!

- To add something to the list, use : listname.Add(*object of type TypeToUse*)
  The object will be added at the END of the list. There are other methods for inserting

- This is enough for our first use. But you can learn more about C# lists here :
  https://docs.microsoft.com/fr-fr/dotnet/api/system.collections.generic.list-1

```
List<int> integer_list = new List<int>();
List<string> string_list = new List<string>();
```

Example declarations of two lists : a list of integers and a list of strings

```
integer_list.Add(10);
string_list.Add("A string");
```

Examples of adding new elements to those strings defined above.

```
//Begin by building the dropdown from the array
ref_dropdown.ClearOptions();

//Build the list of strings to create new options
List<string> newOptions = new List<string>();
for( [redacted] )
{
    newOptions.Add( [redacted] );
}

//Use AddOptions to add the strings as choices
ref_dropdown.AddOptions(newOptions);
```

The code you will need to implement, with some blurred areas. The place where this code needs to be set is also up to you to figure out

Now, it's your turn!

- Using the information from this slide and the previous one, have your code add the names of all the videos to your drop-down list
- The names of the videos should be read from the videoclip's name

Drop-Down : Reacting to User Interaction

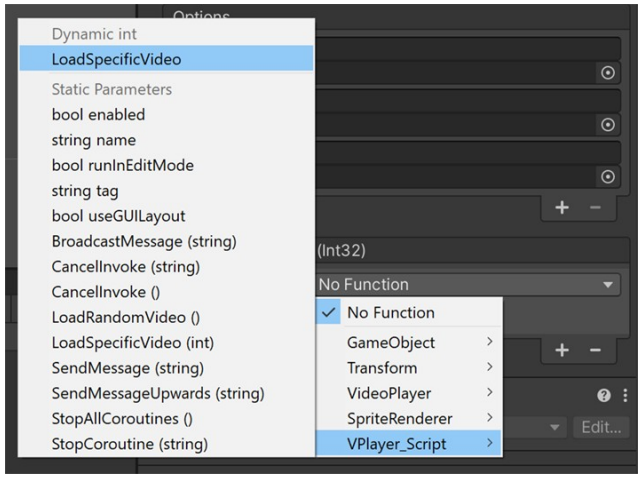
- Our drop-down now contains the right options… But they do nothing!
  We have not defined a method to be called when the value changes, yet

- Good news : the method called by the dropdown is setup easily. Like with Button!
  Look in the Inspector for the "Dropdown – TextMeshPro" component, and search this area

- The difference : the method to call needs to have an int parameter
  This parameter will be used to transmit the number of the choice made by the user

- Since we built the choices of our Dropdown from the video array, their numbers matches the index of the videos
  The first choice, choice number 0 is for video 0. then choice 1 for number 1, etc, etc….

Looks familiar? It should!



WARNING : when picking the method, make absolutely sure you pick it in the "dynamic int" area!

Otherwise, the integer passed to the method will not change when the drop-down is used, making the whole thing useless!

Now, it's your turn!

- Code a method LoadSpecificVideo(int index) which launches the video with the index provided, when it is called
- After making sure it works, connect it to the Drop-down and enjoy!

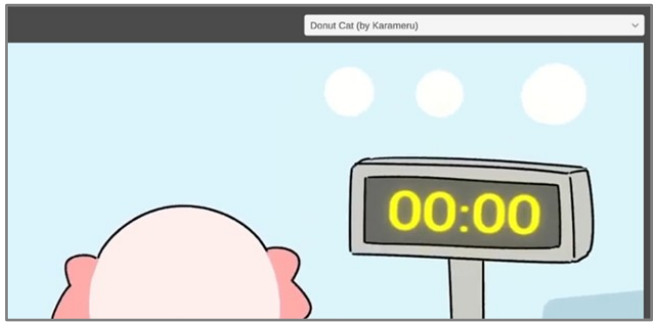
The Last Touch (in the code)

- Only one problem remains with our Drop-Down… Try using “random video”
  When the video changes from an outside source, the drop-down does not update!

- Fortunately, it is possible to manipulate the selected option through code!
  If we do that after a random change, we can keep the dropdown and video matched!

- The method to do that is dropdown.SetValueWithoutNotify(int i)
  This will force the dropdown to select option number “i”, and won’t count as an event

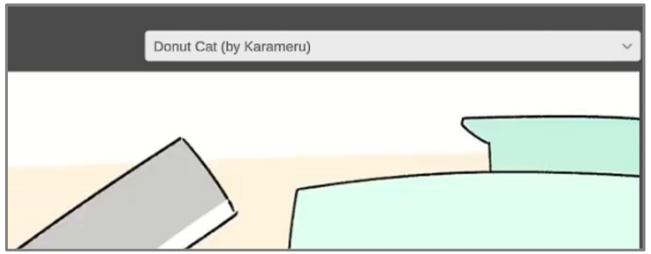
- Changing the value (choice) that way does NOT call the interaction from before
  That’s why the method is called “without Notify” : no event, no call!



The text in my Drop-down says “Donut Cat”,
But that’s not the video playing now!
This problem occurred after a random change

```
ref_dropdown.SetValueWithoutNotify(currentVideo);
```

This code will change the option displayed by the dropdown, but will not notify the event system of the change (no other method call will occur)



After the change, the video and the title displayed in the drop-down should always match!

Now, it's your turn!

- Modify the method that chooses a random video so it updates the Dropdown too
- Manipulate your player a little to check there are no big problems

Drop-Down : Customizing Appearance

- Drop-downs are much more complex than Buttons when it comes to appearance

- Explaining it in full detail would be way too long. But the general idea is to go through its hierarchy, and edit the various graphic elements there

  We are not going to spend 30 minutes on making a super-pretty drop-down, as you might not even use any drop-down in your final project!

- If you want to learn about making a pretty drop-down, I recommend you create a test scene and experiment over there when you have some time!

  Test scenes can be completely ignored when building our app. You can try stuff safely there

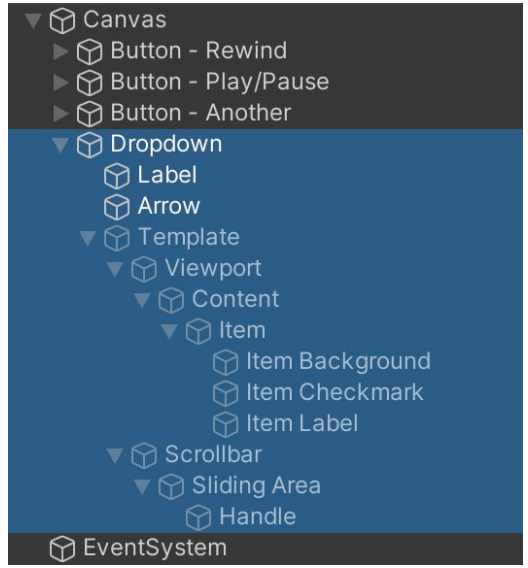
- The easy part is to customize the way the list looks while closed

  Dropdown also has the “Color Tint” and “Sprite Swap” modes from Button. But it applies only to the top. You can also change the “dropdown”, “label” and “arrow” elements.

- If your drop-downs are not customized, this will not count as a demerit in the project

  Since I didn’t really explain how it works to save time. It’s only fair!

An example of customization



Fully customizing the appearance of
A drop-down means customizing all of this!
So, sorry, we simply do not have the time!

Using UI Slider Component

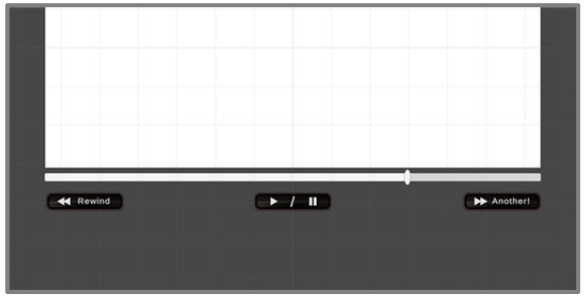
- A classic feature of video players is being able to click on the progress bar to navigate in the video, but we didn't implement that
  We didn't because our progress bar was made with sprites outside Canvas

- To remedy that, we are going to remove our progress bar, and replace it with a Canvas component adapted for this : Sliders!
  Sliders are progress bars on which the user can click to manipulate their "position"

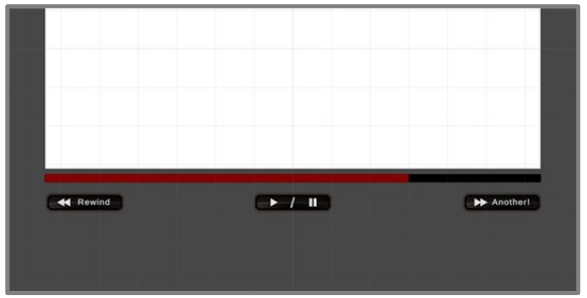
- To create one, it's pretty much the usual : *(right click on hierarchy => UI => Slider)*

- It will automatically be put in Canvas. You can then place it and change its size
  Sliders can not exist outside Canvas. Just like the other UI elements we saw before

- To Customize appearance, you have to edit the children of your Slider GameObject
  Just like the other UI stuff! This system is not the most intuitive, but it allows total control



My slider before Customization
Default color and appearance. I did set its size so it matches my screen, though



My slider after Customization
You can even make the "Handle" invisible by setting its size to 0 in RectTransform

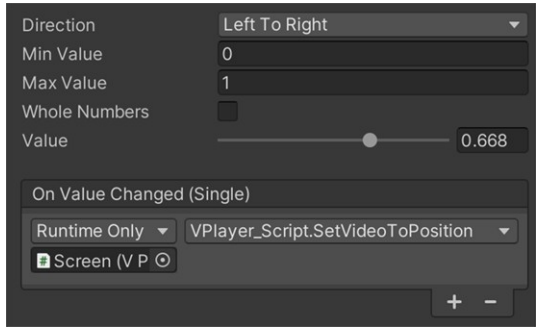
Connecting our Slider to Code

- Sliders can have a method connected to them through the graphic interface
  Just like the other UI elements we talked about before. Pretty convenient!

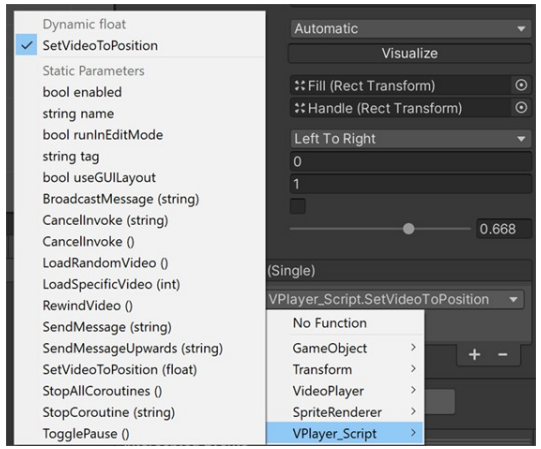
- The method will be called whenever the user clicks to change the slider position
  Test scenes can be completely ignored when building our app. You can try stuff safely there

- The method connected to a slider needs to have a float as parameter
  By default, a slider is “entirely on the left” if value is 0, and “entirely on the right” if value is 1
  This could be changed by editing the properties of the slider component

- In this case, having a value in [0,1] interval is good : with a little math, we can find the frame to teleport to whenever the user changes the slider value!



Some of the parameters of Slider



AGAIN : when picking the method, make absolutely sure you pick it in the “dynamic float” area!

Now, it's your turn!

- Code a method SetVideoToPosition(float percent) which will set the current frame of the video to the one matching the value provided
- Because VideoPlayer.frame is a “long” number type, you will have to do a cast

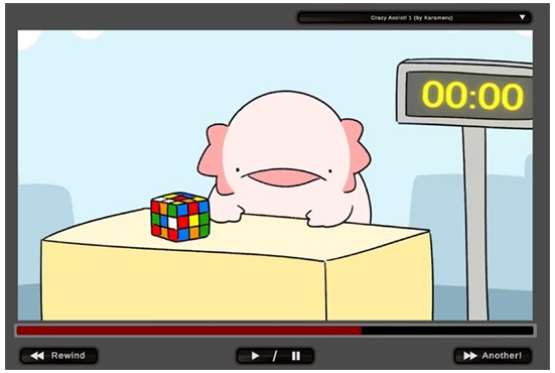
The Same Problem again!

- The slider reacts perfectly to user clicks, but it doesn't move on its own
  And that kind of defeats the point of a video navigation bar, don't you think?

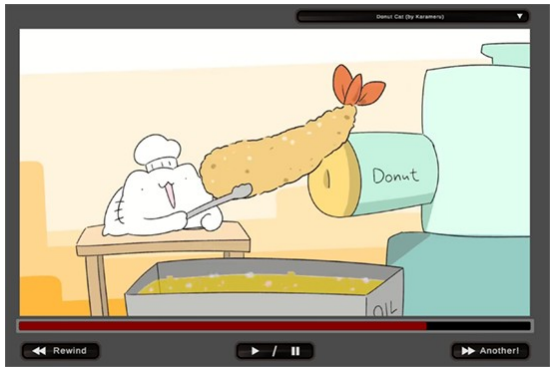
- This is actually the exact same problem than what we had with our dropdown!
  In both cases, we want the value of a UI component to change on its own when needed

- And the solution is the same : SetValueWithoutNotify. Which will change the value of our slider without counting as a click/interaction
  All the Unity UI components were designed to be as similar as possible. Which helps learning!

- As you might suspect, this version of SetValueWithoutNotify expects a float
  Dropdowns used integers as value. Sliders use floats. It's all logical!



I can click the slider to move in my video...
But the slider doesn't move on its own as the video progresses!

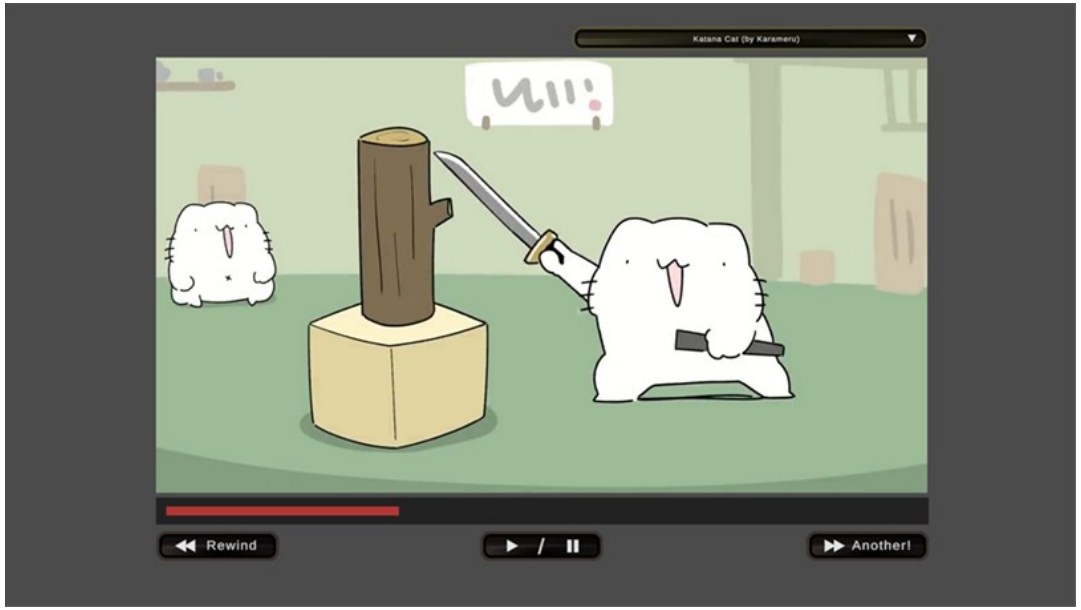


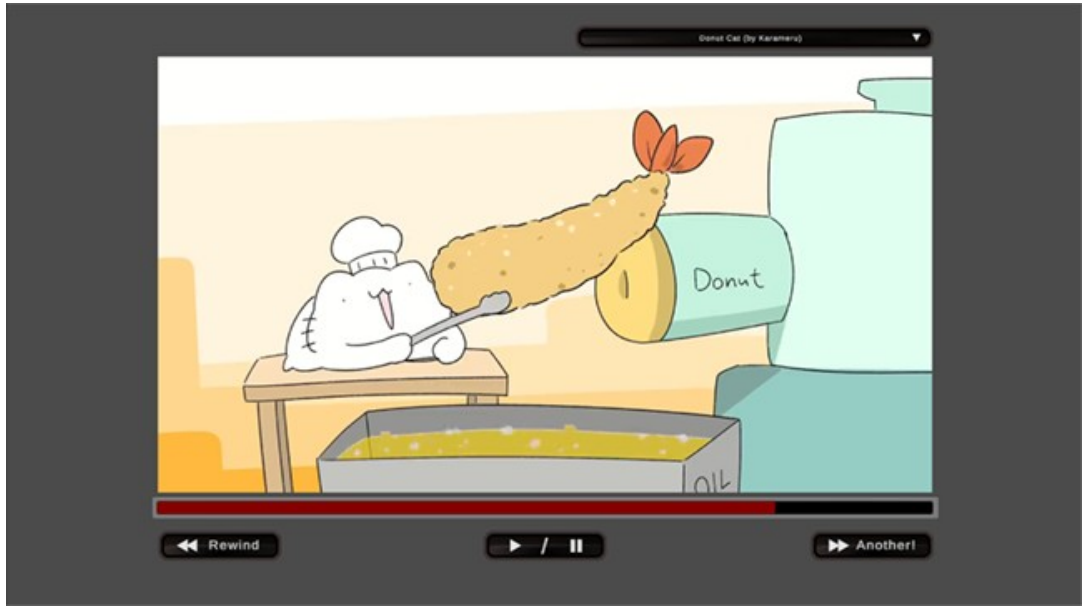
One good line of code... and we're done!

Now, it's your turn!

- Modify the Update() of your script so it changes the value of the slider correctly on each frame
- Remember the casting issue between "float" and "long"

Yay! Our First Canvas Interface!





You can now use 3 of the most common UI concepts

Buttons, Drop-downs, and Slider : with those 3 things, you can already do a lot!

A Stepping Stone

- Good work, you have successfully made a mouse interface in Unity!
  If you use what you learned cleverly, you already have LOTS of options. Try it and Practice!

- However, we only scratched the surface of what is possible with UI and the Canvas!
  Unity is a professional, feature-rich tool. A semester isn't enough to see everything!

- I encourage you to experiment on your own, and try some online tutorials!
  If you can, search in English. There are much more tutorials available than in French!

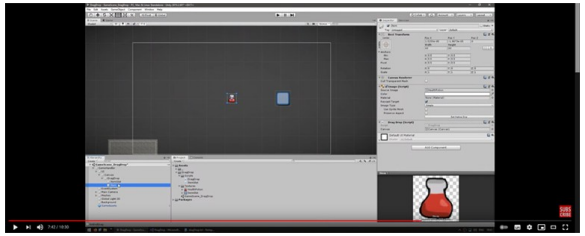
- Consider watching the following videos on the same topic :

Videos by "Code Monkey" (a channel I recommend for unity tutorials) :

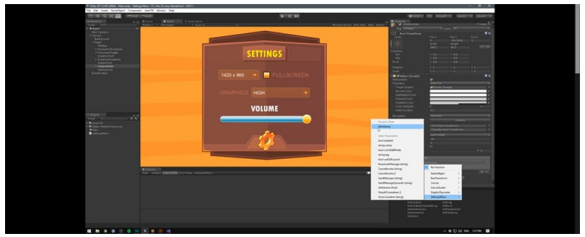
"Simple UI Setup" : https://www.youtube.com/watch?v=VHFJgQraVUs
"Simple Drag Drop" : https://www.youtube.com/watch?v=BGr-7GZJNXg

Videos by "Brackeys" (warning : older videos. Some details may be off) :

START MENU in Unity : https://www.youtube.com/watch?v=HwdweCX5aMI
SETTINGS MENU in Unity : https://www.youtube.com/watch?v=YOaYQrN1oYQ



The Videos by "Code monkey"re-explain the Canvas and the drag-and-drop we saw today in a different way



The Videos by "Brackeys" talk about other UI elements, so consider those if you want to go a bit further

Remember : the best way to learn Unity (and computers in general!) is to TRY STUFF!

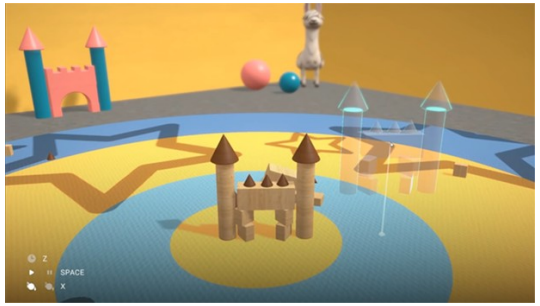
...And if I want to click in the “World”?

- The Raycast is an general way to handle clicks. It also works outside Canvas
  It is possible to detect click on SpriteRenderers, 3D objects, or anything else with a graphic

- But because of Unity class structure, doing it outside of Canvas is quite different!
  The core principles are similar, but the methods and classes/interfaces to use are not

- Remember that outside Canvas, you can NOT use the UI Components, which usually means you have to CODE THEM YOURSELF!!!
  Convenient, pre-programmed stuff like “Sprite Swap” will NOT be there to help you!
  You’ll have to remake that kind of feature on your own!

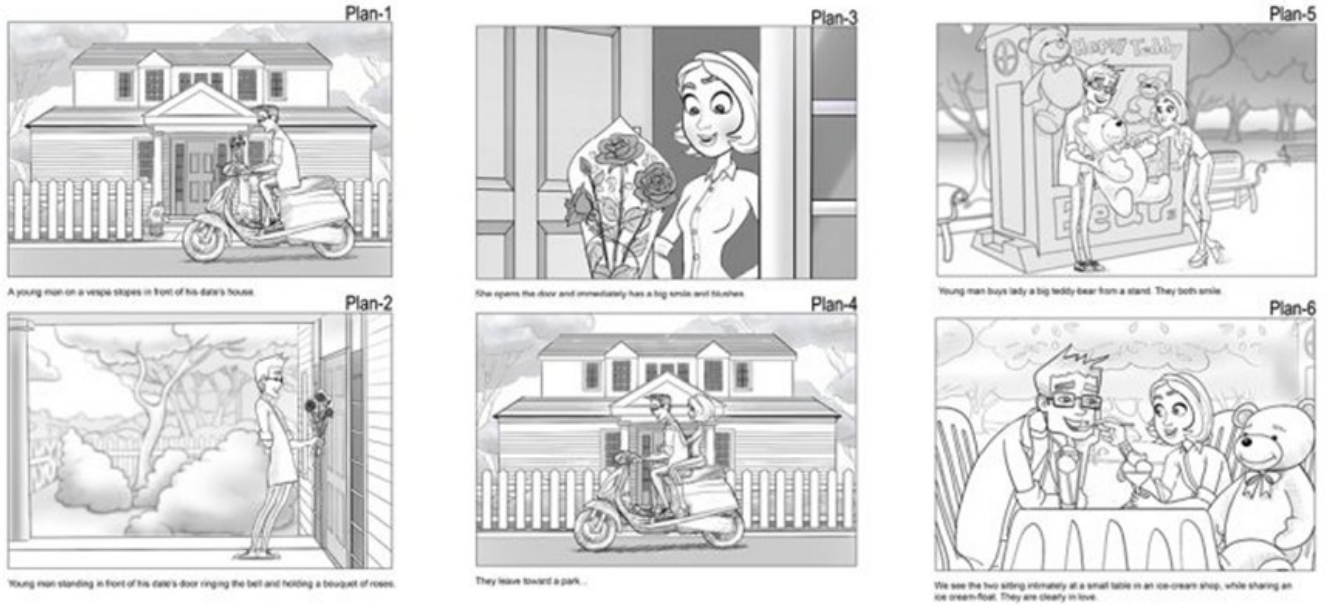
- That’s why, in your projects, I want you to use Canvas for the mouse interactions!
  Detecting clicks and drag on elements outside a Canvas is NOT part of this course

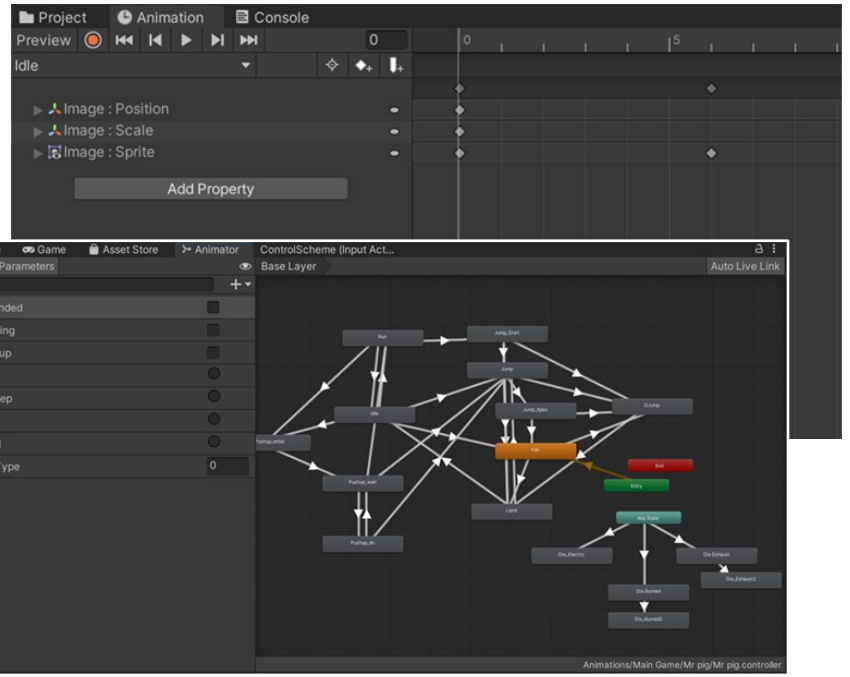


For your curiosity :

You can use the following video if you want to learn how to achieve drag-and-drop without a Canvas.
Just so you can see that it is possible, but quite different : https://www.youtube.com/watch?v=55TBhlOt_U8

Next Time...





Animation in Unity : Controllers and Clips

An intricate system to control animations and transitions