5. Images for Events

Let's add some images to the event! c!pp

Adding images is very similar to adding values. First we must provide the image for that we have to define a Pattern in the Event definition. The Pattern is define with the Pattern class, which takes a key for identifying the correct pattern and the image pattern. The image pattern is just the path starting from the game folder or in your case starting from the root folder of the mod, with the root folder being the folder where your metadata file is located. The Event definition would then for example look like this:

```
init 1 python:
    set_current_mod('base')


    event_label_event = Event(3, "event_label",
        TimeCondition(day = "5-10", month = "2-", daytime = "f"),
        RandomListSelector("girl", "Aona Komuro", "Lin Kato", "Luna Clark"),
        Pattern("main", "/images/events/school building/sb_event_4 <school_level> <girl> <step>.webp"),
        Pattern("main2", "/images/events/school building/sb_event_5 <school_level> <girl>.webp", 'school_level',
'girl'),
    )
    courtyard_events["patrol"].add_event(event_label_event)
```

Here I have added two patterns. First let's look at the Pattern with the key "main".

In Pattern I put "main", which is the key used to identify the correct pattern when showing the images and after that I put in the path, or the pattern. Now I say pattern because you can see in the latter part of the path there are keys put in <>-brackets. **school_level**, **girl_name** and **step**. Now we know **girl**, since it is defined in the Selector directly above, and we know **school_level** since it is always present in the event data. **step** is a new key we haven't seen yet. This key is one of the main functions of these image patterns. The **step** key is used to create a series of images. For example you have 3 images in an event, so you can just name your images "**image 1.webp**", "**image 2.webp**" and "**image 3.webp**". To include all of these images you can use the pattern "**/images/.../image <step>.webp**" and the method later used to access the images will check what images are available and will then provide them for usage.

In a similar way will the method also replace the keys **school_level** and **girl** with the actual values, so the path could then look like this: "**/images/events/school building/sb_event_4 1 Aona Komuro <step>.webp**". Step is still step since it will then be replaced on demand.

Let's look at the pattern with the key "**main2**". It is almost identical to the "**main**"-Pattern except for a little different path, no step key and more keys behind the path. That is another feature these pattern provide. By defining these keys you can set the pattern to also check for images that don't have a value for the corresponding key. The image file would then just have a "**#**" at the keys position instead. In this example you could have an image named: "**sb_event_5 # Aona Komuro 0.webp**" or "**sb_event_5 1 #.webp**" or even "**sb_event_5 # #.webp**". This is particulary useful if you have images where for this example you don't have any girls in the picture, so instead of having the same image three times just with the different names to conform to all possible value combinations, you can just define one image with "**#**" for the key.

But please make sure to only use that feature when it is needed. If there is no image with "**#**", the keys should not be set for ignoring. The method checks for all possible combinations and selects the best one, so while being pretty optimized, it could still have impact on performance.

Pattern(name: str, pattern: str, *alternative_keys: str)

This class provides a way to insert an image patten into the event for it to be used.

name: str

This is the key used for identification.

pattern: str

The image path or pattern.

*alternative_keys: str

(Optional) A set of keys that can be ignored by the patterns.

Now let's use the images in our event. To do that we use the **convert_pattern** or the **show_pattern** method.

The difference between these two is that you use **convert_pattern** for series of images and **show_pattern** for a single image. In our example we have both, so lets add both. Tha would then look something like this:

```
label event_label (**kwargs):
    $ begin_event(**kwargs);


    $ image = convert_pattern("main", **kwargs)
```

```
$ show_pattern("main2", **kwargs)
person "Hi I'm a dialogue text."

$ image.show(0)
person "Oh another image."

$ image.show(1)
person "Wow such simple"

// stat changes

$ end_event('new_daytime', **kwargs)
```

Here you see I used **convert_pattern** to get the pattern with the key "**main**". It is a conversion because that method converts the pattern to an ImageSeries-class. That method comes with a lot of handy functions. One of those is the show method. First things first, the **convert_pattern** method converts the pattern into an ImageSeries-class and returns it, so I then store it in the image variable so I can use it later. Since it is a series of images defined by the step key inside the pattern, I can show the image with the corresponding step number just by calling "**$ image.show(n)**" where n is the number of the image. This class can also show videos, but that comes later.

show_pattern does a similar thing, but instead of creating an ImageSeries it just fills the pattern and directly shows the image.

Maybe you already noticed, but the **get_value** methods are missing here. That is because the **convert_pattern** and the **show_pattern** methods retrieve and register those values themselves, so no need to do that again, unless you want to do other things with these values like changing the dialogue.

Another key that can be used for the pattern I haven't talked about yet is the "**variant**" key. If you add the variant key, you are able to have multiple variants of the same image and the methods will just select one of those at random. You don't have to specify anything. You only have to add the "**variant**" key to the pattern and the methods check for themselves what variants are available. The only thing you have to make sure is, that you have a series of numbers for that starting with 1, since the method starts looking for the variant "1" and then just goes up until it doesn't find a higher variant number.

If you want to override the variant, you can call the show method with the variant argument like this: "**$image.show(0, variant = 1)**".

Be aware that the the order where **convert_pattern** and **show_pattern** occur in the code and the order in which the keys in the pattern occur is also a factor for the order of data stored in the gallery.

---

convert_pattern(pattern_key: str, **kwargs)

Gets a pattern from the event data based on the key and converts it to an ImageSeries class.

pattern_key: str

The key under which the pattern is defined.

**kwargs

This is the data provided for the event. It has to be the kwargs that comes with the event itself.

show_pattern(pattern_key: str, **kwargs)

Gets a pattern from the event data based on the key and converts it to an image and then displays that image.

pattern_key: str

The key under which the pattern is defined.

**kwargs

This is the data provided for the event. It has to be the kwargs that comes with the event itself.

On the last page of this quick manual on how to create an event, I will show you how to manipulate the stats for the different characters.

Revision #3
Created 1 October 2024 10:00:54 by SuIT-Ji
Updated 1 October 2024 11:45:05 by SuIT-Ji