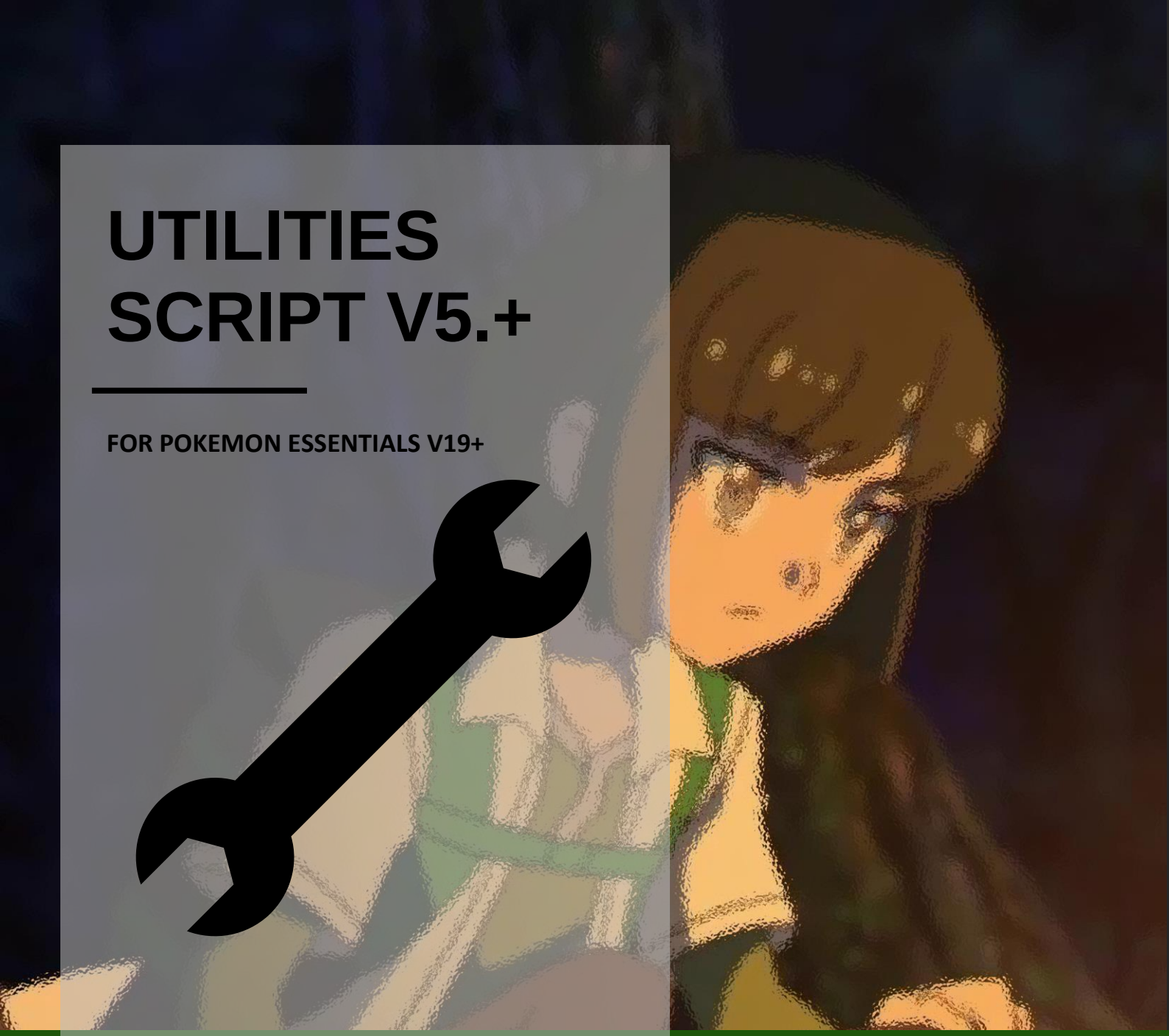
UTILITIES SCRIPT V5.+

FOR POKEMON ESSENTIALS V19+



Made by: S. A. Somersault

TABLE OF CONTENTS

FIRST OF ALL…

WHAT'S THIS?

This set of scripts are now the "brain" of all my other plugins, and hence they all use this kit as the main source of methods and classes. But that does not mean that you cannot use it as a standalone plugin as well. Just the opposite! This kit comes with quite a bunch of useful methods and utilities that you may want to use in your own fangames. SO! Without any further ado, let's get into detail.

INSTALLATION

THE MOST BASIC PART IN ANY SCRIPT

I guess you might be wondering “How on earth do I install this in my own game?”. So before moving on to how to control this script, let’s see how to implement it. It’s just one single step!

1. **SELECT ALL THE FOLDERS IN THE ZIP BUT THE READMEs**
   And uncompress them directly into your game root folder.

AND THAT’S IT!

USAGE

THE MOMENT YOU'VE BEEN WAITING FOR

This plugin will automatically handle all of my other plugins, so you don't have to worry about it at all. Nevertheless, you can still customise some of the behaviours of them, so let's check them one by one. First, let's check the contents of **SUSc.rb**:

XTRANSCEIVERADDED: A simple boolean that lets my other plugins know whether you have installed Klein Studio's Xtransceiver plugin and thus activating their respective compatibility with it.

SUSC_PATH: A string containing the relative path to the folder where all my plugins have their graphics stored. You can modify it, but I do not suggest you to do it unless you know what you are doing.

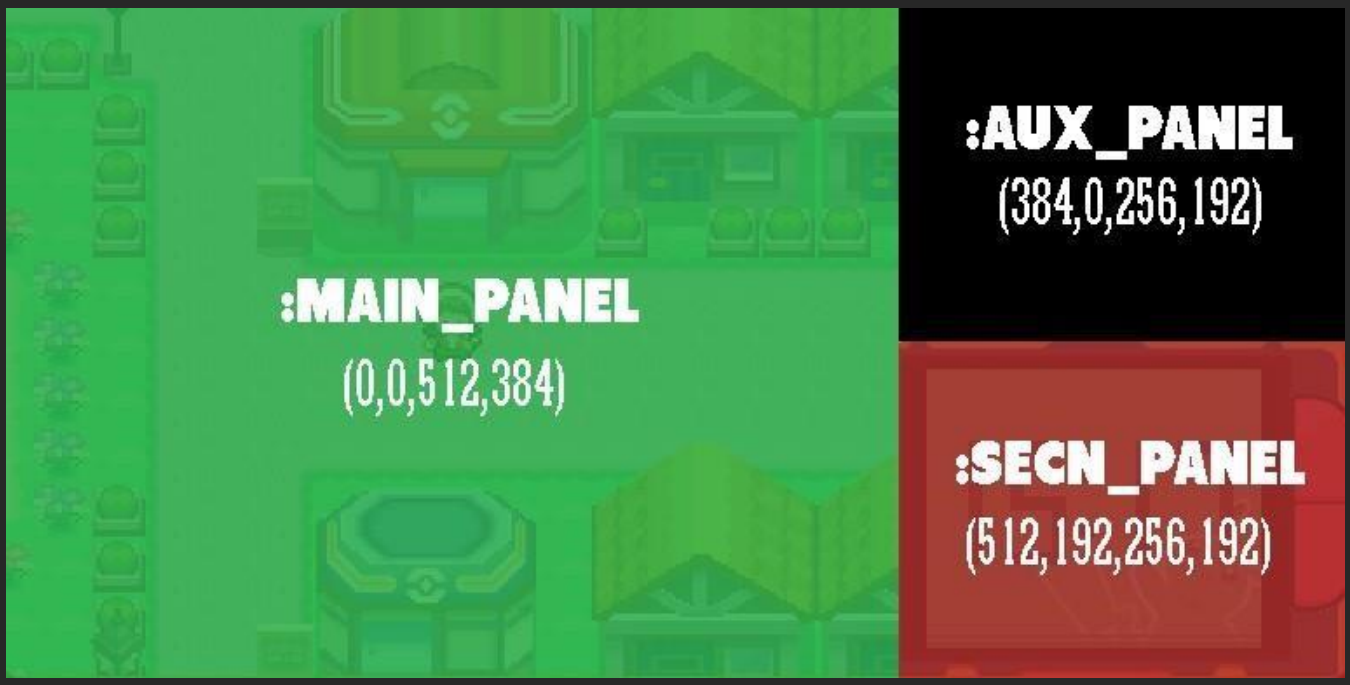
MULTI_SCREEN: This is another boolean that will activate the multi-screen mode for all my plugins! We will get further into detail later, but in short words, it will make all my other plugins to display the graphics in a double-screen system. In order to correctly see how this behaves, you should be aware of your game screen dimensions (more of this in a bit, too).

SCR_SETTINGS: Next up, we have a hash map with the settings for each panel to be displayed in multi-screen mode. For each entry, you have:

- The key of the entry (in order to know which one we are referring to).

- An array with four values containing respectively the position x,y on screen where this panel is placed and the size of it (width and height).

For instance, :MAIN_PANEL, :SECN_PANEL and :AUX_PANEL will have the following arrangement on screen:



Speaking of which, you can freely modify all of their four component attributes, but you must not delete neither :MAIN_PANEL nor :SECN_PANEL entries (if you are using any other of my plugins, that is). On the other hand, you can freely add new panels wherever you want and with the dimensions of your choice. Just add a new entry with the format **“:KEY => [X,Y,WIDTH,HEIGHT],”** and that’s it!

[FROM THIS ALL THE REMAINING METHODS AND CONSTANTS ARE FOR GENERIC USE]

DEBUG_INTRO: Whether the intro of the game should pop up or not during debug mode. Look for **"def pbCallTitle"** and add **"&& !SUSC::DEBUG_INTRO"** at the end of the line **"return Scene_DebugIntro.new if $DEBUG"** to use this feature.

USE_CUSTOM_SUG_NAME: Whether to use custom suggested names or not when aborting the name selection. You can customise the actual suggested names for respectively male and female with the constants **MALE_CHAR** and **FEMALE_CHAR**.

METHODS

SUSc also includes a few standalone methods that you can freely call whenever you want. (Note: the format is SUSC.whatever_method, unless you use inside a class and add the line **"include SUSC"** at the beginning of it). Let's see them as well:

pbSetCharName(name_m,name_f,var): Simple function to name a character one way or another depending on the player's gender. The value will be stored in $game_variables[var].

pbChDigitsAmount(num, n): A simple function to change the number of digits (and zeroes at the left) that a variable has. It returns a string with the number and additional zeroes at the left if 10^n > num.

pbDisplayNumberLCD(number, digits, Width, Height): A simple function to represent a number with graphics; where **number** is the actual number to be represented, **digits** is an array of Sprites which Bitmap is supposed to contain all the numbers from 0 to 9 (aka a sprite looking like this: "0123456789"), **width** and **height** are the respective dimensions of each digit in that sprite. Returns by parameter the same array with each sprite showing the specific digit they are representing.

U.A.R.M. FUNCTIONS FOR SPRITES!

pbNext_mrua_Pos(sprite,speed,dt,g): Applies a 2d vectorial force to a **sprite** with **speed=[sx,sy]**, and gravity **g**. Places the sprite in the position that it reaches on delta time **dt**.

pbMrua(sprite,speed,frames,dt,g): applies (and shows) next **frames** uarm positions to a given sprite, with **speed** [sx,sy], a delta time between frames of **dt**, and gravity **g**. Note: gravitational force is much weaker in the sprite world and hence it's now $1m/s^2$ by default; also notice that since y axis grows downwards, gravity is now positive.

Similar methods for U.R.M can be used by calling respectively **pbNext_mru_Pos(sprite,speed,dt)** and **pbMru(sprite,speed,frames,dt)**.

pbFrameAnimation(sprite,frames,se,row): An easy function to handle pressed button animations, where **sprite** is the sprite to play the animation, **frames** is the number of frames the graphic has, **se** is the sound effect to be played (optional), and **row** is the actual row of the sprite to which apply the animation (in case there is more than one sequence in it); optional as well.

pbAmbSound(soundFile,eventId,maxVolume,radius): a function to add ambient sounds! Create an event with parallel trigger at the source point and call the script. Notes: The sounfile will be placed at SE folder; if at some point the radius is bigger than the distance to the map border, then the SE will still be noticeable when the player goes out of the map; only one ambient sound per map (for the moment). For example, by calling **SUSC.pbAmbSound("Crowd.mp3",24,80,15)**, the file "Crowd.mp3" will be played and you will start hearing at a distance of 15 tiles. The volume will be at 80 when you are on the source point. The event id that calls the script (aka the source point event) is 24.

For bigger objects, like for example a coast or a shore, you can use **pbLAmbSound(soundFile,eventId,maxVolume,range,mode,side)** instead, where mode and side are both optional booleans, mode is for either horizontal (true) or vertical (false), and side is the side of the sound source where the sounfile won't be played lower as you get away in that direction (false for left or up, true for right or down). Once again, this is for sounds like the sea waves, for example.

The rest of the methods are mere auxiliar, so you don’t have to even use them.

SM-SPRITEWRAPPER

This is the essential element of the kit, and essential as well for all my other plugins. This class implements what is called **“Composite Pattern”**, which consists in that an object of this class contains within it another instance of its own class, or in this case, a hashmap of instances of its own class. But what is this useful for?

Imagine for example that you have an object, say, a watch, which in turn contains the graphics for the buttons, the background, the digits, etc. And you want to move the watch around the screen, tweak it, scale it and so on. But as you would find out soon, it would be a pain in the neck going sprite by sprite rescaling, repositioning, etc. So instead, in this case whenever a SMSpriteWrapper is requested to be scaled, repositioned etc, it will first apply that function to itself, and then it will automatically propagate the call for all its children in the hashmap. If these children have in turn more sprites in their own hash (for

example, we could have a SMSpriteWrapper for the entire object, that could contain the screen, the border sprite and the buttons. The screen, in turn, could have in its own hashmap the background, the digits, etc), they will include them as well, since at the end of the day, the children are called to use the very same method as the parents.

This way, with just one call, you can move lots of objects at once!

SM-UTILITY MODULES

In this file, there are several classes and other useful structures defined, but the only important ones for you as a developer are three, and they all in turn inherit from **SMSpriteWrapper**: **ScrManager** which is in charge of handling the different panels or “screens” in multiscreen mode, and **SMModule** and **SMModuleView**, which are the main classes that my scripts inherit from. The latter two work in tandem to respectively in the data and the graphics. This is useful because if wanted to change the looks of a script while

keeping the base functionality (for example, having different models of a Poketch) we just need to change the SMModuleView and add a new one. Both of them have three main methods to be filled (or not) by the subclasses: “init”, “execute” and “finish”.

CONCLUSION

JUST FOR THE SAKE OF GIVING THANKS (and a small spam with my sites if I may)

And this is it for this version. If you have any problem don’t hesitate to mail me at **somersault0023@gmail.com**, or contact me via discord: **Somersault**#9770 (copy and paste it because you are not going to find me otherwise), and I’ll be really glad to help you solve any problem with this or any other script.

I also invite you to check my **Deviantart** account as well as the site of ahuge **Sinnoh fangame** my team and I are developing (in which this script features, by the way!)

With all being said, I would also like to give huge thanks to you. Exactly! Infinite thanks for downloading and playing with this. I really appreciate it. I really hope you like it and you enjoy it. Stay safe and best regards!