

Minkoff's Animated Battlers - Enhanced

v 10.0

The Graphic Overlay system that converts
front-view battlesystems into sideview systems.

by M. B. Randolph / DerVVulfman

based on the original
Animated Battlers
by Minkoff

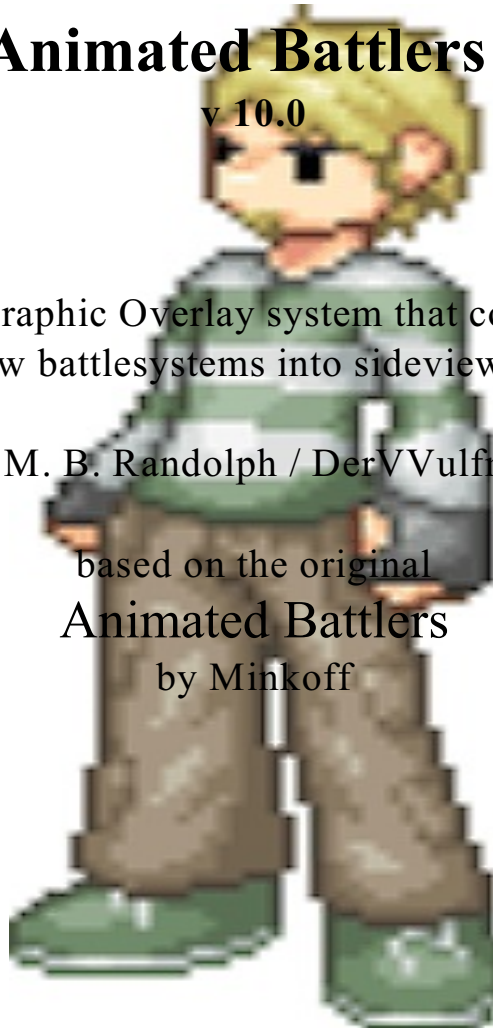


TABLE OF CONTENTS

1.) Introduction	3
2.) Configuration	4
• Arrow Controls	5
• General Controls	6
• Pose Control Center	11
• Expanded Pose Control Center	15
• Frame Control Center	20
• Movement Control Center	21
• Stationary Control Center	23
• Transparency Control Center	25
• Custom Feature Center	27
3.) Outside Calls	28
• Sideview System	28
• Formations	29
• Formation Expansion	30
4.) Other Systems	31
• Party Changing Systems	31
• Rtab's Connected Attacking	32
• Fomar0153's Large Party	33
• Advantages!	34
5.) Battler Templates	35

1.) INTRODUCTION

Welcome, and thank you for using Minkoff's Animated Battlers - Enhanced. This is an RGSS system for RPGMaker XP that is able to convert your standard front-view battlesystem into a simulated sideview battlesystem. This system was originally designed for use with a specifically designed graphic file known as a spritesheet battler, but further enhancements to this system enables Animated Battlers (or AnimBat) to use spritesheet battlers of various designs and the ability to use the default battlers that come with RPGMaker XP itself.

Is Animated Battlers a battlesystem?

No. It is not. This system is an add-on script that can convert many varieties of strategic battlesystems from a standard frontview system into a sideview system. Doing so allows you to create spectacular combat effects as you have seen in games such as Final Fantasy. All it needs is patience, hard work, and lots of graphics.

Is this an easy system to use?

Nope. Okay... well, sort of. There is a massive configuration section in this system that this manual goes into details about. Once you learn what this system can accomplish and how you can bring it about with this system, the sky's the limit. All that is asked in this case is accuracy.

What are these... 'battlers'... that I can use?

Well originally, Minkoff designed the system to use a graphic image known as a spritesheet that held multiple images. This image file was holding forty smaller pictures of the same character. It was ten rows down and four columns wide. A later version of this system added an eleventh row, which meant that the newer spritesheet held forty-four pictures. And while all this was going on, the layout of every battler had to follow the same guidelines. The very first row of images were to show the hero or monster in a battle-ready pose, the second row showed the battler being wounded or struck by an attack, the third... and so on.

This new system has been redesigned so the end user (you) can change this layout and allow you to use other spritesheets, from Cybersam's 7-pose battlers... to old Characterset Battlers. It even allows the use of the default RTP battlers that come with RPGMaker XP, and the ability to mix all these battlers files into the same game.

Will Animated Battlers interfere with my other scripts?

Um, it may. Nearly every system in Animated Battlers is an aliased system. That means that it doesn't overwrite any of the existing code. And while this system is not an SDK (Standard Development Kit) system, this script has been known to work with a number of SDK systems. But... I cannot guarantee it will work with every battlesystem script or add-on out there. In fact, I know it won't. But I have taken great lengths to keep Animated Battlers relatively compatible with as many scripts as possible.

Thank you, and enjoy this system.

2.) CONFIGURATION

The configuration of Animated Battlers is a very delicate thing. This system deals with boolean (true/false) values, flat rate numeric values (both integer and floating point), and various arrays and hash arrays. And while you may be knowledgeable on how to edit some of these values, you may not be aware how to edit them all.

Boolean, or `true/false` values are exactly that. They can be true... or false. These values are mere switches used to turn on certain features. Just remember that the entry of these values must be lowercase.

Many of the systems contain flat-rate values, typically pointing to an image in a spritesheet or used to set the number of stance poses or animation frames used by the system. These are numeric values, plain and simple. In a few cases, a flat-rate value may be optional. In these cases, a value of `nil` is acceptable.

A very few systems require floating point values. These are used for mathematical purposes... either speed rates or calculating percentages. Floating point values require that a number be on the left side of a visible decimal point, even if that number is a '0' zero. As such, setting a value to .25 must really be entered as `0.25`.

Arrays are values that can hold multiple numbers used by the system, mostly the ID numbers of ... something... in the RMXP database. Let's take a look at one right now.

```
DEFAULT_ENEMY_ID      = [ 2, 5 ]
```

This array holds two values, a '2' and a '5'. Values within arrays are separated by commas, assuming that there is more than 1 value within the array. Emptying an array is simple. Just leave it completely empty within the brackets like so:

```
DEFAULT_ENEMY_ID      = []
```

Hashes, or hash arrays are a little more complex. They can hold values that are being referenced by other values. Hmm... hard to explain... Okay, good old example coming up.

```
MNK_APOSE1           = { 2 => 6 }
```

You see, a hash requires at least two values, one key and one returned value. The returned value is a number (mostly) that you are trying to retrieve, and the key is a number you are using to get that value. In the above case, when the 2nd battler tries to use `MNK_APOSE1`, it calls on the 6th pose in his spritesheet. Or, in this hash, the number 6 is returned when a '2' is used as a key.

Advanced hash arrays are even possible, as below shows us:

```
MNK_POSES_FR_ACTOR = { 1 => { 2 => 7, 3 => 4 } }
```

Used in some cases, this hash actually calls on values from another hash within. Be mindful of the braces `{}` though.

2.) CONFIGURATION

ARROW CONTROLS

This little featurette is merely used to move where you want the targeting arrow to appear. By default, the arrow shows below the actor and enemy battlers. However, some battlesystems such as the Final Fantasy games use a ‘hand’ cursor that shows at the left or right of the target instead. As such, you can now position where you want your cursor to show.

```
MNK_ARROW_X           = 14      # The x position for your target cursor  
MNK_ARROW_Y           = 10      # The y position for your target cursor
```

MNK_ARROW_X

This value adjusts the height of the arrow in the battlesystem’s viewport. Higher values lower the cursor (in pixels) while lower values raise the cursor. Negative values are acceptable. The default system has this value set to 14.

MNK_ARROW_Y

This value adjusts the centering of the arrow in the battlesystem’s viewport. Higher values moves the cursor (in pixels) to the right while lower values move the cursor to the left. Like the previously described value, negatives are permitted. The default system has this value set to 10.

2.) CONFIGURATION

GENERAL CONTROLS

The General Controls center of the system adjusts features from the type of battler you are using, through the framerate/speed of the system, into the level of damage battlers sustain before showing an injured appearance.

Default Battler Style Switches

This subset to the general controls center turns on or off the ability to use spritesheets. The original system only used spritesheet battlers, but here you can permit the mixture of spritesheet battlers and default/RTP battlers.

```
DEFAULT_ENEMY           = false  # If true, these switches allows the use
DEFAULT_ACTOR           = false  # of default battlers for actors/enemies
DEFAULT_ENEMY_ID        = []     # Ids of enemies using default battlers
DEFAULT_ACTOR_ID        = [1]    # Ids of actors using default battlers
DEFAULT_COLLAPSE_ACTOR  = false  # If true, restores the old 'red fade'
DEFAULT_COLLAPSE_ENEMY  = false  # collapse effect (using spritesheets)
```

By design, all 'default' battlers use the 'default' red-out death. The pose-styled death is only available for spritesheets.

DEFAULT_ENEMY

When this value is set to true, this informs the system to only use default/RTP-styled battlers for your enemies. Having this value set to false allows you to use spritesheet battlers as the system was originally designed to use.

DEFAULT_ACTOR

Just like the above value, this value has a true/false setting. When it is set to true, it informs the system to only use default/RTP-styled battlers for your heroes. And keeping it set to false allows spritesheet use as normal.

DEFAULT_ENEMY_ID

Assuming that the DEFAULT_ENEMY value is set to 'false', thus informing the system that you are using spritesheet battlers, this value allows you to single out certain enemies that use default/RTP-styled battlers. This way, you can use both default and spritesheets at the same time.

It is relatively simple. Enter the ID numbers of your enemies within the [] brackets and the system will know that these enemies are using RTP battlers. When entering the IDs of your enemies, their IDs are separated by commas. Not using this feature, you leave this array empty... no spaces within the brackets.

DEFAULT_ACTOR_ID

This system mirrors the DEFAULT_ENEMY_ID value, save for the fact that it controls the use of default/RTP-styled battlers for the Actors in your database. Other than that, there really isn't any more to say.

DEFAULT_COLLAPSE_ACTOR

Under normal circumstances, spritesheet battlers use a 'pose' to show whether a battler is dead. This value allows you to configure the system to use the default collapse system to fade the actor battlers away when they die. This assumes that the DEFAULT_ACTOR value is set to false as default/RTP-styled battlers automatically use the default collapse system.

2.) CONFIGURATION

GENERAL CONTROLS

Default Battler Style Switches

DEFAULT_COLLAPSE_ENEMY

Again, this is a mirrored value and it affects enemy battlers just like the previous one affects actor battlers. This value allows you to configure the system to use the default collapse system to fade the enemy battlers away when they die. This assumes that the DEFAULT_ENEMY value is set to false as default/RTP-styled battlers automatically use the default collapse system.

2.) CONFIGURATION

GENERAL CONTROLS

Animation Frames and Animation Speed

This subset to the general controls system determines the speed of this system's animation and the maximum size of the spritesheets being used. When I refer to the maximum size of a spritesheet, I am not referring to pixels but the number of poses and frames. For example, if you're using a spritesheet with 12 frames and 15 poses and another spritesheet with 18 poses and 4 frames of animation, you would set the system to recognize 18 poses and 12 frames as the apparent maximum.

MNK_SPEED	= 4	# Framerate speed of the battlers
MNK_RUSH_SPEED	= 1.5	# Melee/Skill/Item motion speed of the battlers
MNK_POSES	= 11	# Maximum # of poses (stances) in the template
MNK_FRAMES	= 4	# Maximum # of frames in each pose
MNK_FRAMES_STANDARD	= 4	# Standard # of frames played in each pose.

MNK_SPEED

This value controls the frame rate of the battler... not how fast they actually run across the screen but how fast the animation cycles through the frames for each pose. A value of 7 is decent enough for 4 frame animations... 2 is dead slow.

MNK_RUSH_SPEED

This value controls the attack-moving speed of the battler... you know... how fast they actually run across the screen when attacking. 1.0 is the default speed, 0.5 is half the speed and 2.0 is pretty speedy.

* NOTE: A speed of 1.0 to 1.5 is considered the best speed ranges. Anything above 1.5 could cause strange visual behavior between battlers, possibly causing them to 'stick' or 'freeze' in the middle of the screen.

MNK_POSES

This value refers to the battler image as well. By default, it informs the system that your battler file holds eleven separate battle stances, or poses. Increasing this value to 12 informs the system that the battlers have 12 poses... decreasing it to 4 informs the system that all your battlers use a mere four poses. This value must apply to 'all' your battlers.

MNK_FRAMES

This value is in reference to the battler image you're using. By default, it informs the system that your battler file has been designed with four frames of animation for each of the battle poses it holds. If you design your battler with MORE frames of animation, you tell the script how many frames you're using here. These frames are placed left to right in the image file. This value must apply to 'all' your battlers.

MNK_FRAMES_STANDARD

Now this informs the system just how many frames of animation are 'typically' used in each battler pose. It cannot be GREATER than the total number of poses in the spritesheet, but it can be smaller. This way you can have different frames of animation in different poses.

2.) CONFIGURATION

GENERAL CONTROLS

Individual Spritesheet Control Center

This subset is useful if the system is mixing spritesheet battlers that have a different number of poses and/or frames from the norm. By the use of this system, you can define a system that can use Minkoff 10-pose or 11-pose battlers, Cybersam 7-pose battlers, Characterset battlers and the like. You can even specify the size of battlers that have custom frames for unique attacks, skills or... well... your choice.

<code>MNK_POSES_ENEMY</code>	<code>= {1 => 4}</code>	<code># ID and # of poses for each enemy</code>
<code>MNK_FRAMES_ENEMY</code>	<code>= nil</code>	<code># ID and # of frames for each enemy</code>
<code>MNK_POSES_ACTOR</code>	<code>= {2 => 4}</code>	<code># ID and # of poses for each actor</code>
<code>MNK_FRAMES_ACTOR</code>	<code>= nil</code>	<code># ID and # of frames for each actor.</code>

MNK_POSES_ENEMY

This hash refers to the poses available for individual enemy battler images. It allows you to use spritesheets that have fewer poses than the number of poses (set in the `MNK_POSES` value). You cannot have 'more' poses than the maximum set, but you can have fewer.

MNK_FRAMES_ENEMY

This hash refers to the frames available for individual enemy battler images. It allows you to use spritesheets that have fewer frames than the standard number of frames (set in the `MNK_FRAMES` value). Currently, you still have to use the same `MNK_FRAMES_STANDARD` and like values and such...

MNK_POSES_ACTOR

This hash refers to the poses available for individual actor battler images. It allows you to use spritesheets that have fewer poses than the number of poses (set in the `MNK_POSES` value). You cannot have 'more' poses than the maximum set, but you can have fewer.

MNK_FRAMES_ACTOR

This hash refers to the frames available for individual actor battler images. It allows you to use spritesheets that have fewer frames than the standard number of frames (set in the `MNK_FRAMES` value). Currently, you still have to use the same `MNK_FRAMES_STANDARD` and like values and such...

2.) CONFIGURATION

GENERAL CONTROLS

Wooziness Rates

This subset to the general controls center determines how injured a battler need be before it shows whether it is injured or not. Please note that this only works with spritesheet battlers.

```
MNK_LOW_HP_PERCENTAGE = 0.25 # Health% for WOOZY pose.
MNK_LOW_HP_ACTOR = {7 => 0.50, 8 => 0.75} # Ind. health% for actors.
MNK_LOW_HP_ENEMY = {1 => 0.50} # Ind. health% for enemies.
MNK_LOW_HP_FLAT = true # If true, flat rate hp
```

MNK_LOW_HP_PERCENTAGE

This value adjusts how low a battle's health must be before the battler switches to a 'WOOZY' pose. Originally this was hardwired to a value of 25%, but now you can adjust the value to your liking (default is still set at 0.25).

MNK_LOW_HP_ACTOR

This hash adjusts how low each hero's health must be before the battler switches to a 'WOOZY' pose. Unlike the `@low_hp_percentage` value, you can set each value for each hero.

MNK_LOW_HP_ENEMY

This hash adjusts how low each enemy's health must be before the battler switches to a 'WOOZY' pose. Unlike the `@low_hp_percentage` value, you can set each value for each enemy.

MNK_LOW_HP_FLAT

This is a true/false value that changes the hp 'wooziness' system from using percentage rates to flatrate hp values. As long as this value is kept false, the system continues to go by percentage values. But setting this value to true allows the system to go by flat rate values.

If, for example, the system is set to use the flat rate system and the `MNK_LOW_HP_PERCENTAGE` value is set to 23, then the 'WOOZY' pose will occur when the battlers health drops below 23hp. This affects the `MNK_LOW_HP_ACTOR` and `MNK_LOW_HP_ENEMY` values as well.

2.) CONFIGURATION

POSE CONTROL CENTER

The Poses Control Center allows you to set up the system to use whatever style of spritesheet you are employing. The original version of Animated Battlers did not have this feature and the system was only able to afford a specifically designed spritesheet.

Now, it is designed with a system where you can change the location of the stances/poses it retrieves from your battler(s). And as this is based on the original Minkoff system, there are eleven poses which you can employ. As such, you can now employ spritesheets other than the Minkoff-Styled battlers.

If you are using default/RTP-styled battlers, this entire system is ignored. ^_^

Primary Template

The primary template is the main template used by the spritesheet system. Each value set informs the system where the actual pose/stance is within your spritesheet battler.

```
MNK_POSE1 = 1 # Sets the 'Ready Pose' (MNK_POSE1) #1 in your template
MNK_POSE2 = 2 # Sets the 'Struck Pose' (MNK_POSE2) #2 in your template
MNK_POSE3 = 3 # Sets the 'Woozy Pose' (MNK_POSE3) #3 in your template
MNK_POSE4 = 4 # Sets the 'Block Pose' (MNK_POSE4) #4 in your template
MNK_POSE5 = 5 # Sets the 'Charge Pose' (MNK_POSE5) #5 in your template
MNK_POSE6 = 6 # Sets the 'Retreat Pose' (MNK_POSE6) #6 in your template
MNK_POSE7 = 7 # Sets the 'Attack Pose' (MNK_POSE7) #7 in your template
MNK_POSE8 = 8 # Sets the 'Item Pose' (MNK_POSE8) #8 in your template
MNK_POSE9 = 9 # Sets the 'Skill Pose' (MNK_POSE9) #9 in your template
MNK_POSE10 = 10 # Sets the 'Victory Pose' (MNK_POSE10) #10 in your template
MNK_POSE11 = 11 # Sets the 'Defeat Pose' (MNK_POSE11) #11 in your template
```

MNK_POSE1 to MNK_POSE11

These values are used to tell the system where each pose is located on a battler image file. Typically, most anyone leaves this alone. They just create their battlers to match THIS system's template (1st pose for ready, 2nd pose for a struck battler, etc). But you could rework the poses to your personal tastes. It can be reformatted to work with Cybersam battlers, Characterset Battlers, RM2K3 battlers, or your own custom format.

2.) CONFIGURATION

POSE CONTROL CENTER

Primary Template

Example (Cybersam layout):

```
MNK_POSE1 = 2 # Sets the 'Ready Pose' (MNK_POSE1) #2 in your template
MNK_POSE2 = 4 # Sets the 'Struck Pose' (MNK_POSE2) #4 in your template
MNK_POSE3 = 7 # Sets the 'Woozy Pose' (MNK_POSE3) #7 in your template
MNK_POSE4 = 3 # Sets the 'Block Pose' (MNK_POSE4) #3 in your template
MNK_POSE5 = 1 # Sets the 'Charge Pose' (MNK_POSE5) #1 in your template
MNK_POSE6 = 4 # Sets the 'Retreat Pose' (MNK_POSE6) #4 in your template
MNK_POSE7 = 5 # Sets the 'Attack Pose' (MNK_POSE7) #5 in your template
MNK_POSE8 = 6 # Sets the 'Item Pose' (MNK_POSE8) #6 in your template
MNK_POSE9 = 6 # Sets the 'Skill Pose' (MNK_POSE9) #6 in your template
MNK_POSE10 = 2 # Sets the 'Victory Pose' (MNK_POSE10) #2 in your template
MNK_POSE11 = 7 # Sets the 'Defeat Pose' (MNK_POSE11) #7 in your template
```

As you can see, the numbers applied have changed. Each one points out a different row in your battler's spritesheet (their pose) which is applied to the system. In this example, the value of MNK_POSE2 which identifies the 'STRUCK' pose looks for the 4th pose in the Cybersam spritesheet.

2.) CONFIGURATION

POSE CONTROL CENTER

Custom Actor Spritesheets

The Custom Actor template is used if you plan to use more than one spritesheet design. Like the primary template, it contains the basic 11 pose/stance types which can be entered for your system. But unlike the primary template, these are hashes which can affect 1 or 'more' actor battlers at a time and these values are optional.

```
MNK_APOSE1 = {2 => 2}
MNK_APOSE2 = {2 => 2} # Basil is using a Charset graphic as a battler.
MNK_APOSE3 = {2 => 2} # The battler was copied into the Battler folder.
MNK_APOSE4 = {2 => 2} # This setup allows you to use Charactersets for
MNK_APOSE5 = {2 => 2} # battlers battlers.
MNK_APOSE6 = {2 => 3}
MNK_APOSE7 = {2 => 2}
MNK_APOSE8 = {2 => 2}
MNK_APOSE9 = {2 => 2}
MNK_APOSE10 = {2 => 1}
MNK_APOSE11 = {2 => 4}
```

MNK_APOSE1 to MNK_APOSE11

This is basically the same thing as the primary editable template (MNK_POSE1 to MNK_POSE11) with the exception that these are editable hashes that allow you to use different spritesheets or layouts for individual actor battlers. As an example, you can see the layout for a characterset spritesheet (just a charset) for the 2nd actor in your database.

You can apply more than one custom layout at a time for your actors, though only one layout per battler. Thus... Basil can have one custom layout and Hilda can have another, but no single actor can have two layouts. Here's an example for the ready pose:

```
MNK_APOSE1 = {2 => 2, 8 => 6}
```

The above sample sets battler #2 to use the 2nd pose in the spritesheet for the READY pose while battler #8 uses the 6th pose. It isn't required to have any values in these hashes and any values that are empty are bypassed. If there are no values for a given pose, the system uses those set in the Primary Template (above).

However, if no actor uses a custom spritesheet layout (everyone uses the same battler design), then you are to empty all the values within the braces {}.

2.) CONFIGURATION

POSE CONTROL CENTER

Custom Enemy Spritesheets

The Custom Enemy template mirrors the Custom Actor Template in function and design with two main differences. The first... it applies to enemy battlers (duh). And the second... there is no MNK_APOSE10 for an enemy 'Victory' pose as virtually every battlesystem exits or goes to the gameover screen when the badguys win.

```
MNK_EPOSE1 = {1 => 2}
MNK_EPOSE2 = {1 => 2} # Did the same to the ghosts. Note that enemies have
MNK_EPOSE3 = {1 => 2} # no victory pose.
MNK_EPOSE4 = {1 => 2}
MNK_EPOSE5 = {1 => 2}
MNK_EPOSE6 = {1 => 3}
MNK_EPOSE7 = {1 => 2}
MNK_EPOSE8 = {1 => 2}
MNK_EPOSE9 = {1 => 2}
MNK_EPOSE11 = {1 => 12} # Setting the ghost to an invalid pose erases it.
```

MNK_EPOSE1 to MNK_EPOSE11

Again, this is basically the same thing as the primary editable template (MNK_POSE1 to MNK_POSE11) with the exception that these are editable hashes that allow you to use different spritesheets or layouts for individual enemy battlers. As an example, you can see the layout for a character set spritesheet (just a charset) for the 1st enemy in your database.

And again, it doesn't have an MNK_EPOSE10 as there is no victory pose for enemy battlers.

NOTE: Entering a value for a non-existent pose does not cause an error. Instead, this will merely hide the battler from the screen until a valid pose is made available. In the Enemy Template example, a pose was set to 12. With the assumption that the battler has no more than 4 poses, a 12th pose doesn't exist and will erase the battler from the screen.

2.) CONFIGURATION

EXPANDED POSE CONTROL CENTER

The Expanded Pose Control Center allows you to go beyond the basic eleven poses that Minkoff originally employed. Mostly hashes or expanded hashes, it contains a few flat rate values that can be nil-ed out.

Non-Default Poses

Non Default poses are optional values. They add a number of additional poses for your battlers to display and subsets if a pose is only applied by an individual actor or enemy. These values, again are optional and can be emptied. Flat rate values can be set to nil, while hash values are emptied by emptying the braces {}.

```
MNK_POSES_SETUP           = 7           # Choose animation pose for 'preparation'
MNK_POSES_SETUP_A        = {2 => 4}
MNK_POSES_SETUP_E        = {1 => 4}
MNK_POSES_CASTPREP       = 4           # Set 'casting' pose for skill preparation
MNK_POSES_CASTPREP_A     = {}
MNK_POSES_CASTPREP_E     = {9 => 3}
MNK_POSES_DYING          = 6           # Choose animation pose for dying throws.
MNK_POSES_DYING_a        = {}
MNK_POSES_DYING_E        = {9 => 5}
MNK_POSES_ESCAPE         = 2           # Set 'coward' pose for fleeing monsters)
MNK_POSES_ESCAPE_A       = {}
MNK_POSES_ESCAPE_E       = {9 => 5}
MNK_POSES_CRITICAL        = nil        # Set pose for BIG hits
MNK_POSES_CRIT_A         = {}
MNK_POSES_CRIT_E         = {9 =>5}
MNK_POSES_WINNING        = 4           # Set winning (Victory Dance before pose)
MNK_POSES_WINNING_A      = {}
MNK_POSES_WINNING_E      = {}

MNK_LOOPS_WINNING        = [7]        # Actor IDs if their victory pose loops
MNK_LOOPS_DEFEATED_ACTOR = []         # Actor IDs if their defeat pose loops
MNK_LOOPS_DEFEATED_ENEMY = []         # Enemy IDs if their defeat pose loops
```

MNK_POSES_SETUP

This value is used to tell the system what pose in your battler image file is used to show the battlers actually readying themselves for battle. Warriors can be seen drawing their swords, archers can notch their arrows, mages can read spellbooks... all before the battle starts.

MNK_POSES_SETUP_A

This is a hash value that can specify an exact pose for an individual actor battler. As an example, if you were to add '4 =>8' into this hash, the 4th actor in your database would use the 8th pose to perform his specific setup pose.

MNK_POSES_SETUP_E

This is a hash value that can specify an exact pose for an individual enemy battler. As an example, if you were to add '9 =>4' into this hash, the 9th enemy in your database would use the 4th pose to perform his specific setup pose.

(The similar pattern of _A and _E hashes are used for the below values too...)

2.) CONFIGURATION

EXPANDED POSE CONTROL CENTER

Non-Default Poses

MNK_POSES_CASTPREP

This value is used to tell the system what pose in your battler image file is used for 'casting'. This assumes that you're using a recognized system that delays a skill's use.

MNK_POSES_DYING

This value is used to tell the system what pose in your battler image file is used to show your battler collapsing before actually shown defeated. It cycles once and could be used to show heads lopped off, skeletons falling apart into a pile of bones, ghosts popping like balloons... or whatever you design.

MNK_POSES_ESCAPE

This value is used to tell the system what pose in your battler image file is used when escaping from battle. Only for enemy battlers, as the system merely fades to the game map when heroes flee from battle. By default (when this value is set to nil), the escaping monsters use the attack pose before fading out.

MNK_POSES_CRITICAL

This value is used to tell the system what pose in your battler image file is using when he/she suffers a critical hit. Normally suffering just a normal hit, this value allows you to use a separate animation for massive skull-pounding blows.

MNK_POSES_WINNING

This value is used to tell the system what pose in your battler image file is used to show the battler beginning his victory cheer. While the normal victory pose is a steady (or frozen) single pose, this winning pose can show the battler performing a dance before shooting off his victory stance.

MNK_LOOPS_WINNING

This value is used to tell the system what hero (by their ID) uses a victory pose that loops indefinitely. This allows certain heroes to show victory stances with flowing capes fluttering in the breeze... or whatnot. There is no enemy battler equivalent.

It is relatively simple. Enter the ID numbers of your actors within the [] brackets and the system will know that these actors have looping poses. When entering the IDs of your actors, their IDs are separated by commas. Not using this feature, you leave this array empty... no spaces within the brackets.

MNK_LOOPS_DEFEATED_ACTOR

This value is used to tell the system what hero (by index) uses a defeated pose that loops indefinitely. This allows certain heroes to show poses where a battler is lying in the dirt, wind blowing their cape or ... or whatnot.

MNK_LOOPS_DEFEATED_ENEMY

This value is used to tell the system what enemy (by index) uses a defeated pose that loops indefinitely. This allows certain villains to show poses where a battler is lying in the dirt, wind blowing their cape or ... or whatnot.

2.) CONFIGURATION

EXPANDED POSE CONTROL CENTER

Non-Default Pose Hashes

Much more detailed, the Pose Hashes details anything from custom poses based on the item used by an actor or skill used by an enemy, to the critical hit pose used by an actor hit by a specific weapon or skill. All values herein are hash values, however the ones with either an _A or _E suffix call hashes themselves.

```
MNK_POSES_CASTED           = {61 => 6} # Set a specific skill to use a pose
MNK_POSES_CASTED_A        = {}
MNK_POSES_CASTED_E        = {}
MNK_POSES_STATUS           = {3 => 3} # Set status values to poses here
MNK_POSES_STAT_A          = {}
MNK_POSES_STAT_E          = {}
MNK_POSES_SKILLS           = {57 => 7} # Default: #57(Cross Cut) does 'Attack'
MNK_POSES_SKILLS_A        = {}
MNK_POSES_SKILLS_E        = {}
MNK_POSES_ITEMS           = {13 => 4} # Default: #13(Sharp Stone) does 'Block'
MNK_POSES_ITEMS_A         = {}
MNK_POSES_ITEMS_E         = {}
MNK_POSES_WEAPONS         = {} # Didn't set any weapons to any poses
MNK_POSES_WEAPS_A         = {}
MNK_POSES_WEAPS_E         = {} # Non-functional (Enemies don't use 'em.)

MNK_STRUCK_WEAPS          = {} # Set a specific 'Struck' to a weapon attack
MNK_STRUCK_WEAPS_A        = {}
MNK_STRUCK_WEAPS_E        = {}
MNK_STRUCK_SKILLS         = {} # Set a specific 'Struck' to a skill
MNK_STRUCK_SKILLS_A       = { 7 => { 7 => 4 }}
MNK_STRUCK_SKILLS_E       = {}
MNK_STRUCK_ITEMS          = {} # Set a specific 'Struck' to an item attack
MNK_STRUCK_ITEMS_A        = {}
MNK_STRUCK_ITEMS_E        = {}
MNK_CRIT_WEAPS            = {} # Set a specific 'Critical Hit' to a weapon
MNK_CRIT_WEAPS_A          = {}
MNK_CRIT_WEAPS_E          = {}
MNK_CRIT_SKILLS           = {} # Set a specific 'Critical Hit' to a skill
MNK_CRIT_SKILLS_A         = {7 => {7 => 10 }, 5 => {7 => 7}}
MNK_CRIT_SKILLS_E         = {}
MNK_CRIT_ITEMS            = {} # Set a specific 'Critical Hit' to an item
MNK_CRIT_ITEMS_A          = {}
MNK_CRIT_ITEMS_E          = {}
```

MNK_POSES_CASTED

This array holds the ID number of skills that forces a battler to perform a casting pose, and assigns an individual pose for each of 'em. As an example: using 61 => 5 will assign to skill #61 to the 5th pose.

MNK_POSES_CASTED_A

This is a hash value that can specify an exact casting pose for an individual actor battler based on EACH spell being cast. As an example, if you were to add '2 => {4 => 8}' into this hash, then when the 2nd actor in your database uses the 4th spell in the database, he/she would use the 8th pose to perform his specific casting pose

2.) CONFIGURATION

EXPANDED POSE CONTROL CENTER

Non-Default Pose Hashes

MNK_POSES_CASTED_E

This is a hash value that can specify an exact casting pose for an individual enemy battler based on EACH spell being cast. As an example, if you were to add '1 =>{3 =>12}' into this hash, then when the 1st actor in your database uses the 3rd spell in the database, he/she would use the 12th pose to perform his specific casting pose (assuming the spritesheet has 12 poses).

(The similar pattern of _A and _E hashes are used for the below values too...)

MNK_POSES_STATUS

This array holds the ID number(s) of status effects (like dazzle or poison) and assign separate poses for each of 'em. As an example: using 3 => 2 will assign to status #3 (venom) to the 2nd pose.

MNK_POSES_SKILLS

This array holds the ID number(s) of skills and the 'pose' in your battler image that the skill uses. As an example: using 57 => 6 will assign to skill #57 to the 6th pose.

MNK_POSES_ITEMS

Similar to MNK_POSES_SKILLS, this array holds the ID number(s) of items and the 'pose' in your battler image that the skill uses. As an example: using 13 => 3 will assign to item #13 to the 3rd pose.

MNK_POSES_WEAPONS

This array holds the ID number(s) of weapons and the 'pose' in your battler image that the weapons use. As an example: using 2 => 13 will assign to weapon #2 (the Iron Sword) to the 13th pose (assuming your spritesheet has 13 poses).

MNK_STRUCK_WEAPS

This array holds the ID number(s) of weapons and the 'pose' in your battler image that is shown when struck BY that weapon. As an example: using 2 => 13 will assign to weapon #2 (the Iron Sword) to force the battler to show the 13th pose when hit.

MNK_STRUCK_SKILLS

This array holds the ID number(s) of skills and the 'pose' in your battler image that is shown when hit BY that skill. As an example: using 57 => 6 will assign skill #57 to force the battler to show the 6th pose when struck.

MNK_STRUCK_ITEMS

This array holds the ID number(s) of items and the 'pose' in your battler image that is shown when affected BY that item. As an example: using 13 => 3 will assign to item #13 to force the battler to show the 3rd pose when applied.

MNK_CRIT_WEAPS

This array holds the ID number(s) of weapons and the 'pose' in your battler image that is shown when a Critical Hit was scored BY that weapon. As an example: using 2 => 13 will assign to weapon #2 (the Iron Sword) to force the battler to show the 13th pose when hit.

2.) CONFIGURATION

EXPANDED POSE CONTROL CENTER

Non-Default Pose Hashes

MNK_CRIT_SKILLS

This array holds the ID number(s) of skills and the 'pose' in your battler image that is shown when a Critical Hit is scored BY that skill. As an example: using 57 => 6 will assign skill #57 to force the battler to show the 6th pose when struck.

MNK_CRIT_ITEMS

This array holds the ID number(s) of items and the 'pose' in your battler image that is shown when a Critical Hit is scored BY that item. As an example: using 13 => 3 will assign to item #13 to force the battler to show the 3rd pose when applied.

2.) CONFIGURATION

FRAME CONTROL CENTER

There is only one section to the Frames Control Center... so let's get on with it...

Frames Control

This one section has only three values to edit, yet does so much. It allows you to set a single pose to a certain number of animation frames. Even more, you can set a single actor or enemy to use a specific number of frames for a skill.

```
MNK_FRAMES_PER_POSE = {}  
MNK_POSES_FR_ACTOR = {}  
MNK_POSES_FR_ENEMY = {}
```

MNK_FRAMES_PER_POSE

This array is what identifies the number of animation frames in 'specified' battler poses. As an example: using 1 => 2 will assign to the 'ready' pose, a mere two frames of animation, assuming that the 'ready' pose is pose #1 in your spritesheets.

MNK_POSES_FR_ACTOR and MNK_POSES_FR_ENEMY

These arrays are for 'advanced' editing and manipulation of the frames for each battler. With it, you can control the number of frames used for each pose in a spriteset. But even more than that, you can choose a completely different combination of frames per pose for EACH INDIVIDUAL BATTLER, whether it be a hero or an enemy battler.

Example: MNK_POSES_FR_ACTOR = {7 => {1 => 4}}

This example only holds a set for just one hero... hero #7 (or Gloria if you're using the Default system). Linked to #7 is a smaller hash {1 => 4}. In that, the 1st pose you are using (the 'ready' pose by default) is set to use 4 frames.

2.) CONFIGURATION

MOVEMENT CONTROL CENTER

This section doesn't change the poses or the frames at all. It moves the battler around.

Forward Step System (Final Fantasy-Style)

This system controls how the battler moves, whether they take a single step forward before they deliver their attack (be it skill or whatever), or... not.

```
MNK_RUSH_OFFSET      = 0          # How much additional space between battlers
MNK_RUSH_ATTACK      = false       # If true, battler steps forward to attack
MNK_RUSH_SKILL       = true        # If true, battler steps forward to use skill
MNK_RUSH_ITEM        = true        # If true, battler steps forward to use item
```

MNK_RUSH_OFFSET

This value controls how much extra overlap there is between actor and enemy battlers when they attack each other. The larger the number, the more they overlap. This value is measured in pixels.

MNK_RUSH_ATTACK

As long as this value is set to true, the battlers will take a courtesy step forward before performing a attack. This way you can see them in action, otherwise the battler will charge straight to the target (the only 'RUSH' switch that operates this way). It will be overridden if using a weapon in MNK_STATIONARY_WEAPONS or if MNK_MOVE2CENTER_ATK is enabled

MNK_RUSH_SKILL

As long as this value is set to true, the battlers will take a courtesy step forward before using a skill. This way, you can see them in action, otherwise they just stand there. It will be overridden if MNK_MOVING_SKILL or MNK_MOVE2CENTER_SKILL is triggered.

MNK_RUSH_ITEM

As long as this value is set to true, the battlers will take a courtesy step forward before using an item. This way, you can see them in action, otherwise they just stand there. It will be overridden if MNK_MOVING_ITEM or MNK_MOVE2CENTER_ITEM are triggered.

2.) CONFIGURATION

MOVEMENT CONTROL CENTER

Movement Arrays

This system controls other movement systems. It permits certain skills and/or items to make the battler go towards their target, or just move to the approximate center of the screen.

```
MNK_MOVING_ITEM      = [1]      # Examples are items that need to be applied.
MNK_MOVING_SKILL     = [61]     # Examples are martial-arts and sneak attacks
MNK_MOVE2CENTER_ATK  = []       # Moves battler to center based on weapon id!
MNK_MOVE2CENTER_ITEM = [5]      # Moves battler to center for a big item atk!
MNK_MOVE2CENTER_SKILL = [7]     # Moves battler to center for a big skill atk!
```

MNK_MOVING_ITEM

This array holds the id numbers of any item that the designer feels would require the hero to rush the enemy before use. Something like a poison dagger that needs to hit, a splash of holy water. Whatever. Rarely used, this system is available none the less. Note that this overrides the MNK_RUSH_ITEM switch.

MNK_MOVING_SKILL

Yet another array. This array holds the id numbers of any skill that the designer feels would require the hero to rush the enemy before use. More like combat skills... martial arts... wrestling attacks or whatnot. It's more useful than the \$moving_item_atk array. Note that this overrides the MNK_RUSH_SKILL switch.

MNK_MOVE2CENTER_ATK

Another array. This array holds the id numbers of any weapon that the designer feels would require the hero to rush to the center of the screen. Something like swinging a massive halberd or super-long claymore would use something like this. Note that this overrides the MNK_RUSH_ATTACK array.

MNK_MOVE2CENTER_ITEM

Another array. This array holds the id numbers of any item that the designer feels would require the hero to rush to the center of the screen. Something like throwing a fragmentary grenade that would hit all the enemies would use something like this. Note that this overrides the MNK_MOVING_ITEM array and MNK_RUSH_ITEM switch.

MNK_MOVE2CENTER_SKILL

Yet another array. This array holds the id numbers of any skill that the designer feels would require the hero to rush to the center of the screen. Something like a massive all-encompassing spell that would hit the enemies would use something like this. Note that this overrides the MNK_MOVING_SKILL array and MNK_RUSH_SKILL switch.

2.) CONFIGURATION

STATIONARY CONTROL CENTER

The opposite of the movement control center, this system is useful to FREEZE the battler from moving during an attack. Yes, even animated battlers must stay put too...

Stationary Battlers

These are simple true/false values that determine if a battler moves in battle or not. Pretty much a full override system.

```
MNK_STATIONARY_ENEMIES = false    # If the enemies don't move while attacking
MNK_STATIONARY_ACTORS   = false    # If the actors don't move while attacking
```

MNK_STATIONARY_ENEMIES

This merely controls whether the enemies move to attack.

MNK_STATIONARY_ACTORS

This merely controls whether the actors move to attack

2.) CONFIGURATION

STATIONARY CONTROL CENTER

Stationary Arrays

These are specialized values that make certain battlers 'stay' in place depending on certain conditions, whether by their ID, their weapons, skills cast or items used.

```
MNK_STATIONARY_ENEMY_IDS = []          # Enemies that don't RUN during melee attacks
MNK_STATIONARY_WEAPONS = [17,18,19,20,21,22,23,24] # (examples are bows & guns)
MNK_STATIONARY_SKILLS = []           # (examples are bows & guns)
MNK_STATIONARY_ITEMS = []           # (examples are bows & guns)
```

MNK_STATIONARY_ENEMY_IDS

Not a value, but an array. This array holds the id numbers of any enemy that the designer feels would be prevented from moving. These would typically be enemies armed with guns, bows, and the lot.

MNK_STATIONARY_WEAPONS

Not a value, but an array. This array holds the ID numbers of any weapon that the designer feels would prevent the hero from moving. These would typically be guns, bows, and the lot. The demo and the script already has the RTP database's guns and bows in the array as an example.

MNK_STATIONARY_SKILLS

Not a value, but an array. This array holds the id numbers of any skill that the designer feels would prevent the hero from moving.

MNK_STATIONARY_ITEMS

Not a value, but an array. This array holds the id numbers of any item that the designer feels would prevent the hero from moving.

2.) CONFIGURATION

TRANSPARENCY CONTROL CENTER

One of the more gimmicky sections of this system. It controls the battler's transparency or opacity during battle.

Transparency Settings

The Transparency settings control just how transparent a battler can be, and allows you to set certain actors or enemies 'as' transparent battlers. It also controls the battler phasing system which allows a battler to fade out and in when attacking.

```
MNK_TRANSLUCENCY      = 127      # Degree of transparency
MNK_TRANSLUCENT_ACTOR = []       # ID of actor at translucency settings
MNK_TRANSLUCENT_ENEMY = [1, 9]   # ID of enemy at translucency settings
MNK_PHASING           = true     # If battlers fade in/out while charging
MNK_PHASING_ACTOR     = [1, 2]   # IDs of actors that fade in/out if charging
MNK_PHASING_ENEMY     = [9]     # IDs of enemies that fade in/out if charging
MNK_FADE_IN           = true     # Battler fades in if replaced or transparent
```

MNK_TRANSLUCENCY

This value sets the degree of transparency or opacity is used for translucent battlers. The higher the number, the more 'solid' a battler is while lower values make the battlers harder to see. A Range of 0 to 255.

MNK_TRANSLUCENT_ACTOR

Not a value, but an array. This array holds the id numbers of any actor that is transparent or translucent. The degree of the actors' translucency is based on the MNK_TRANSLUCENCY value described above.

MNK_TRANSLUCENT_ENEMY

Not a value, but an array. This array holds the id numbers of any enemy that is transparent or translucent. The degree of the enemies' translucency is based on the MNK_TRANSLUCENCY value described above.

MNK_PHASING

When set to true, this creates a cool effect where the battler fades out when he/she starts to attack an enemy, and fades in on the strike (ooh... ninja-style).

MNK_PHASING_ACTOR

Not a value, but an array. Similar in function as the MNK_PHASING value, this array holds the id numbers of any actor that disappears from sight only to reappear before performing a hit against his target.

MNK_PHASING_ENEMY

Not a value, but an array. Similar in function as the MNK_PHASING value, this array holds the id numbers of any enemy that disappears from sight only to reappear before performing a hit against his target.

2.) CONFIGURATION

TRANSPARENCY CONTROL CENTER

MNK_FADE_IN

When set to true, this allows battlers that change their battler-graphics to fade into the battlescreen instead of abruptly 'popping' into view. NOTE: If using Transparent battlers, a false value will show any transparent battler immediately on startup of combat instead of 'fading into view' with all the other non-transparent battlers. It works much better kept 'true'.

2.) CONFIGURATION

CUSTOM FEATURE CENTER

Specialized features that are not covered by any other section

Custom Features

Just a number of features like mirrored enemies, delay systems and a few other nuances

```
MNK_MIRROR_ENEMIES = true      # Enemy battlers use reversed image
MNK_CALC_SPEED     = false     # System calculates a mean/average speed
MNK_AT_DELAY       = true      # Pauses battlesystem until animation done.
MNK_ADV_OFF_TURN   = 1        # Number of turns before enemies turn around
```

MNK_MIRROR_ENEMIES

Really, a designer's tweak. Hero battlers are required (by design) to face the left side of the screen. If you design ALL your battlers (heroes AND enemies) to face the left side of the screen, then set this switch to true. Then, the enemy battlers will be drawn facing the RIGHT side of the screen... in essence, mirrored. However, if the switch is set to false, then the enemy battlers must be CREATED with them facing the right side of the screen.

MNK_CALC_SPEED

A math system that controls the speed of the animation script in relation to the speed of the battle system (default or RTAB) that it adapts. Usually, this is set to true, but if you find a battler or two slowing down (acting sluggishly), then turn this switch to false. It should be set to false when in use with Trickster's Stealing, Mugging & Scan v6 script.

MNK_AT_DELAY

A tweak for actual Active Timer battlesystems. When this value is set to false, the system will allow any and all battlers to move or perform attacks regardless of the actual 'state' of any other battler... moving or otherwise. Good if you want rapid action in your system... bad if you notice that attack moves target where the target 'was' and not 'is'. However... if you set this value to true, then the actions of the battlesystem are halted (including the AT bars) until the battler in action finishes his move.

The RTAB patch will be required if you are using this feature with RTAB.

MNK_ADV_OFF_TURN

A special feature is used with the 'Advantages!' battle add-on. When used with the Advantages add-on, actors or enemies in Animated Battlers may find themselves facing the wrong way. This value sets how many turns the battlers are 'turned around' before they return to normal.

3.) OUTSIDE CALLS

This section describes system routines that are called from map events or other scripts.

SIDEVIEW MIRROR

By default, when the battle system is invoked, the heroes are positioned on the right side of the screen while the battlers are stationed on the left. But by calling the following RGSS code:

```
$sideview_mirror = 1
```

You can reverse the position of the heroes and enemies.

Here's another example (a map event):

```
@>Script:$sideview_mirror = 1  
@>Battle Processing: Ghost*3  
@>Script: $sideview_mirror = 0
```

This map event (like in my upcoming demo) tells us that

- 1) The heroes and enemies are reversed... heroes on left, enemies on right
- 2) The battle begins (the RTP's 2nd Troop... 3 ghosts)
and finally...
- 3) I reset the heroes and battler positions to their original position.

Technically speaking, **1** tells us that it's reversing their position. Any other value (not necessarily **0**) will return it to normal.

Just remember that when you set up your **TROOPS** in the database, that you move 'em all to the left when you set up your game.

3.) OUTSIDE CALLS

FORMATIONS

By default, the heroes line up in a diagonal pattern across the screen, but by changing the `$formation_style` value in an event or script call, you can change their battle formation.

```
$formation_style = 3
```

You have 9 different options... 8 formations (0-7):

0=Diagonal (default) #1	4= Column Pattern #1
1=Diagonal Pattern #2	5= Column Pattern #2
2=Slanted Two-Column #1	6=Row Pattern #1
3=Slanted Two-Column #2	7=Row Pattern #2

... or you can enter a value of '8' to generate a random formation.

Others can be added by you if you are a scripter (in the 'screen_x' and 'screen_y' defs)... but that's for you to do. I can't add every possible formation. You may also have to edit the configuration section's ...

3.) OUTSIDE CALLS

FORMATION EXPANSION

Your battle formations need not have only teams up to 4 players. Now, the system is adaptable to handle larger parties (if using a large party script like the one designed by KGC).

By default, the heroes can only go up to 4 members on the battlefield, but by changing the **\$formation_max_member** value in an event or script call, you can change the number that can appear.

```
$formation_max_member = 6
```

Note: The number does have to be divisible by two. No odd values are acceptable (sorry).

Also, you can change the height and width of the party's formation on the screen with the following values, also editable in an event or script call:

```
| $formation_max_height |  
| $formation_max_width |  
| $battlestatus_height |
```

The **\$formation_max_height** value tells it how close to the top of the screen (in pixels) the party will be shown. It's defaulted at 220.

The **\$formation_max_width** value tells it how wide the party is spread out on the screen. It's default value is 128.

The **\$battlestatus_height** value is SUPPOSED to hold the height of the battlestatus window, so the system can judge how far down the battlers can go. Normal battlestatus windows are 160 in height.

4.) OTHER SYSTEMS

PARTY-CHANGING SYSTEMS

Though the system can now allow for the changing of spritesheet battlers, it will not remember placement of your party members in the 'current' battle formation.

What does that mean?

If you use a compatible in-battle party changing script, and remove a member or replace a member from your party, the system may fadein/out and reshuffle your members in combat. It will look clunky.

Best bet it to use an in-battle changer that can replace a party member and keep the actual party formation...

Before	Replace	After
Aluxes		Aluxes
Basil		Basil
Hilda	-> -> ->	Felix
Gloria		Gloria

Now if it can keep the party order... no problem, otherwise the battlers will disappear and fade back in, or will just suddenly change position without moving.

Can't be helped.

4.) OTHER SYSTEMS

RTAB'S CONNECTED ATTACKING

The Connected Attacking script used to interfere with the Animated Battler's script big-time. It interfered with a couple of skill definitions in the code, and 'Hung' the script whenever the heroes actually moved from their spot during an attack. Now the system autodetects Connected Attacking, but you must still comment out (or remove) its **action_phase** definition (starts at line 29).

4.) OTHER SYSTEMS

FOMAR0153'S LARGE PARTY

This system is compatible with Fomar0153's Large Party system (version 2+), so you can increase or decrease the maximum number of actors in your party.

If you are using Fomar0153's Large Party system (version 2+) and intend to limit your party to have 'fewer' than four partymembers, it is important for you to edit your battlesystem's 'Spriteset_Battle' class. To be specific, you will have to edit the def update section and remove the following lines of code:

```
@actor_sprites[0].battler = $game_party.actors[0]
@actor_sprites[1].battler = $game_party.actors[1]
@actor_sprites[2].battler = $game_party.actors[2]
@actor_sprites[3].battler = $game_party.actors[3]
```

Do not worry as these lines are replaced by both Fomar0153's system and Animated Battlers. It is THIS code that allows increase and decrease of party members in battle.

Please note that you will have to post the 'BattleStatus' page of Fomar0153's system below Animated Battlers for it to work properly.

4.) OTHER SYSTEMS

ADVANTAGES!

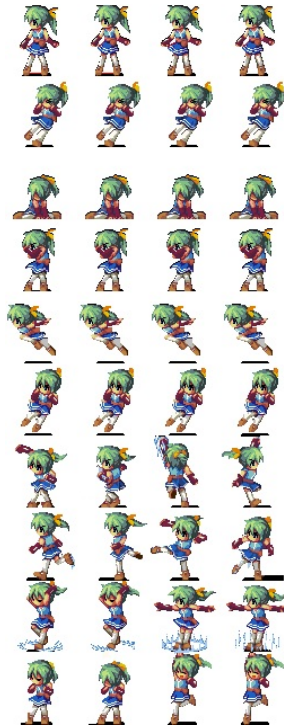
This system is compatible with my Advantages! System, so battles not only allow enemy troops or player parties have an advantage in combat, but now allows a party to have an advantage that shows one side literally behind the other... backstabbing them essentially.

The Advantages! script must be properly configured and pasted above Animated Battlers for it to work.

5.) BATTLER TEMPLATES

This system can use a number of templates, though it was primarily designed for use with the Minkoff 11-pose battler. It can use Cybersam battlers, character set battlers, and a variety of other spritesheet templates. Currently, the only type of battler that I haven't modified the system to use are Ccoa's spritesheet battlers. Be that as it may, let me give you two battler examples and their templates (the values you enter into the the template system):

Minkoff's 11-pose



- #1, Ready
- #2, Struck
- #3, Woozy
- #4, Block
- #5, Charge
- #6, Retreat
- #7, Attack
- #8, Item
- #9, Skill
- #10, Victory
- #11, Defeat

Cybersam's 7-pose



- #1, Charge
- #2, Ready
- #3, Block
- #4, Struck
- #5, Attack
- #6, Skill
- #7, Defeat