Convenience Functions for the OHR

Volume 1

Complete Edition

Thank you for downloading this convenience pack for the OHR. Its goal is to provide more flexible movement, character, and essential functions to speed up the scripting process during complex cutscenes and layouts. Most of the functions simplified here are based on commonly designed features, such as moving heroes and NPCs around, but a few special tricks are also listed for all-purpose scripting.

Please familiarize yourself with each function before plugging them into your plotscript. The process will go much smoother when you know what you have to work with. Doing so may also provide you with ideas for additional convenience functions.

This document will explain what each function is and how to use it in your script properly.

Please remember to include "convenience functions updated 1.txt" at the top of your plotscript. Consult "movement shortcuts updated 1.txt" for an example.

Note: The global variable "fpstimer" *must* be defined at the start of the game with a value equaling the game's default frame rate for "wait equivalent" to work as intended. Example: To simulate classic OHR games, set this value to 18. If plotscripting ever allows flexible frame rates, then "fpstimer will need updating when the script calls for an FPS change.

Important Global Variables and Constants:

Remember to include these global variables and constants in your game's plotscript before you compile. The "convenience functions" depend on them to work. (See "Movement Shortcuts updated 1.txt" for usage example.)

```
# num, fpstimer
# num, waitsecs start
# num, waitsecs counter
# num, npc original picture
# num, npc original palette
# num, npc original direction
# num, npc original frame
# num, store npc original
# num, store speed
# num, speed is set
# num, px plus
# num, py plus
# num, npc behavior
# num, generaltileid
```

num, generalzoneid
num, npcneighbor
num, fixedx
num, fixedy
num, fixeddirection
num, northx
num, northy
num, eastx
num, easty
num, southx
num, southy
num, westx
num, westy
num, forwardx
num, forwardy
num, forwardx1
num, forwardx2
num, forwardy1
num, forwardy2
num, fixednpcx
num, fixednpcy
num, fixednpcdirection
num, northnpcx
num, northnpcy
num, eastnpcx
num, eastnpcy
num, southnpcx
num, southnpcy
num, westnpcx
num, westnpcy
num, forwardnpcx
num, forwardnpcy
num, forwardnpcx1
num, forwardnpcx2
num, forwardnpcy1
num, forwardnpcy2
num, compareparticipants
num, valcompare
num, comparecondition
num, generalfunctionvar
num, generalfunctionarr
num, symbol string
num, money value
num, sound value
num, polar string
num, exp string

num, exp value
num, exp sound

And be sure to add these constants.

num, axis:xline
num, axis:yline
num, condition:match
num, condition:greater than
num, condition:less than
num, condition:greater or equal
num, condition:less or equal

Category Contents:

A Note about Syntax: Optional arguments are italicized. “Convenience functions updated 1.txt” reveals each function’s default values for additional information.

Remember that you can edit “movement shortcuts update 1.txt” to experiment with each function before testing it in your own game.

To All Users: If any particular function is broken, please mention it in the “Introducing Convenience Functions for the OHR: Volume 1” thread on Slime Salad so that I may fix it (and so others don’t use it).

Finally, each section begins on a new page, so keep scrolling for the next section, or use the table of contents to quickly jump to the section of your choice.

Hope this makes your scripting adventures go a lot faster and smoother. If I make Volume 2, let me know what you’d like to see made more convenient for that release. Thanks.

--Pepsi Ranger

General Shortcuts:

These shortcuts include text, range, and other basic functions.

--Wait Functions--

This series of functions is designed for precision waiting when dealing with real time or frame rate adjustments.

wait equivalent (ticks)
-This function is a stand-in for "wait." Substituting all basic "wait (ticks)" with "wait equivalent (ticks)" will futureproof your script for any changes to frame rate without speeding up your animations or other carefully choreographed elements. Note that all functions in "convenience functions updated 1.txt" uses "wait equivalent" instead of "wait." If you'd rather affect all actions with your frame rate adjustments, then convert "wait equivalent" back to "wait."

waitsecs (ticks)
-A substitute for "wait (ticks)." This function uses precise timing to determine waits and is based on real time, not system time. This function is ideal for syncing with music. Note: A change in frame rate *may* affect precision. Adjust values in "waitsecs" script if syncing isn't exact.

waitsecs2 (ticks)
-An alternative value for extra precision if "waitsecs" isn't perfect.

waitsecs start
-Essential function for triggering "waitsecs" capability. Should be used at the top of any script that uses "waitsecs."

waitsecs end
-Essential function for ending "waitsecs" functionality. Should be used at the end of any script that uses "waitsecs."

--Text Functions--

These functions are designed to simplify basic text display options. Text shortcuts involving NPC movements can be found in the "Bonus Movement Shortcuts" section.

basic text (text)
-A substitute for "show text box" that includes "wait for text box" functionality.

basic text box (text)
-Another name for "basic text" for those who like consistency.

show text box and wait (text)
-Another syntax-specific name for "basic text." Displays the text and waits for input.

delayed text (text, time)
-A substitute for "show text box" that displays text for a set length of time before self-advancing.

text with sound (text, sound, *time*, *duration*)
-A substitute for "show text box" that waits for a set length of time (defaults to zero) before playing a sound effect. To play a sound immediately, just leave the "time" argument blank. To play a sound repeatedly, set "duration" to "true." Note: This function does not end the sound when the textbox advances.

random text (start, finish)
-Displays a random textbox within the defined range. "Start" and "finish" values should be textbox ID numbers denoting the desired range.

textbox menu (text, menu)
-Opens the defined menu ID after the textbox closes.

basic conversation (*text1*, *tick1*, *text2*, *tick2*, …, *text8*, *tick8*)
-Simple textbox series template that integrates pauses between continuous blocks of text when simulating a dramatic conversation between characters. Supports up to eight textbox series.

--Slice Functions--

This section covers functions dealing with slices.

slice liberation (which slice)
-This function checks that a slice is valid before freeing it. Useful for avoiding "invalid slice" warnings when freeing a slice.

--Sound Functions--

This section covers functions related to using sound effects.

reset sound (sfx)
-Stops a sound-in-progress before playing it again. Useful for keypresses and other relevant areas for repeating sounds.

stop sounds (*sfx*, *sfx2*, …, *sfx12*)
-Permits stopping a bank of up to 12 sound effects at once.

--Tag Functions--

This section covers functions related to using tags.

set tags (state, *tag1*, *tag2*, …, *tag12*)
-Permits setting a bank of tags to the same "on" or "off" value state at once. Supports up to 12 tags.

set tag by state (*tag1*, *state1*, *tag2*, *state2*, …, *tag12*, *state12*)
-Permits setting a bank of up to 12 tags at once. "State" values default to ON.

--Timer Functions--

This section covers functions related to timers.

stop timers (*timer1*, *timer2*, …, *timer12*)
-Stops up to 12 timer IDs at once.

--Frame Rate Functions--

This section is for advanced FPS functionality.

target frame rate
-Determines the game's current frame rate.
Note: This is a function that TMC posted on Slime Salad.

--Range and Value Functions--

This section simplifies value ranges, item checks, and other conveniences involving numbers.

make decimal string (string id, value, *digits*)
-This will ideally place a decimal or float behind two number values in a string. "Digits" is optional and defaults to two places.
Note: This is another TMC special.

in range (v, min, max)
-Checks whether the value of a variable (*v*) sits within a specific range of numbers (*min*, *max*). Must be used as a condition.
Note: Another TMC special.

in range reverse (v, max, min)
-Like "in range" but better for reading when using reverse ranges. Regular "in range" accomplishes the same thing, so this is only necessary for ease of reading.

match search (v, *rangevalue1*, *rangevalue2*, …., *rangevalue8*)
-Returns true if the value of *v* (used as a variable) matches *exactly* one of up to eight values listed in the range. Note: This function has a much more sophisticated cousin called "look for one." See below for details.

set compare condition (num, value, condition)
-A function used to store three global variables needed for the "look for one" and "look for all" functions. "Num" determines how many items will be included in the search. "Value" determines the number that the participating items will be compared to. For example, if the script requires three specific items equaling 2 or more, then the value of *value* should be "2." "Condition" sets the comparison constant (*condition:match*, *condition:greater than*, *condition:less than*, *condition:greater or equal*, *condition:less or equal*). Make sure these constants are defined before using. This function also requires the global variables "compareparticipants," "valcompare," and "comparecondition" to work properly. Make sure these are also defined.

look for one (*obj1*, *obj2*, …, *obj16*)
-This function uses the values defined in "set compare condition" to pass a *true* or *false* statement if the conditions are met. This is useful for puzzles or any script that requires a specific item be present among a list of possible items for the condition to be true. In other words, if the script is checking for five items, but only one of those items needs to be present to pass the test, then this function should be used as the checkpoint. "Objects" can be predefined in a variable, and that variable can equal whatever value is necessary to permit the script to continue, up to and including a combination of values. Function must be used as a condition, and it should use variables as part of its range. Conditions are set in "set compare condition" as *condition:match*, *condition:greater than*, *condition:less than*, *condition:greater or equal*, and *condition:less or equal*. Supports searching up to 16 objects.

look for all (*obj1*, *obj2*, …, *obj16*)
-Works exactly like "look for one," but requires all conditions met to pass the checkpoint.

Example for using "look for one":

```
script, find me some pants, begin
 variable (found)
 set compare condition (3, 1, condition:greater or equal)
 if (look for one (jeans, shorts, pantaloons)) then(
  found += 1
 )
 if (found >= 1) then(
  show text box (1)  #I have pants!
  wait for text box
 )
end
```

In the above example, “set compare condition” is searching through three pants items (jeans, shorts, and pantaloons), all of which are defined as global variables and have stored counts attached. The script is searching all three items to see if at least one has a count or a value of 1. Maybe jeans and shorts both have a value of 0, but if the script finds pantaloons with a value of 2, meaning the player has picked up two pantaloons somewhere, then the script will check the required match condition to see if it passes. Because the condition is set to “greater or equal,” and because the minimum match value is 1, the function will pass a “true” value, which means it will run the condition inside (increase the value of “found”) and run the corresponding condition that matches the “found” result, which is a text confirming that “I have pants.”

inc global (global id, *amount*)
-A quick way to increase or increment the value of a global variable by its ID number rather than its reference. “Amount” defaults to 1. Note: If you’d rather use the global variable’s reference, then write it as “@global.”

dec global (global id, *amount*)
-A quick way to decrease or decrement the value of a global variable by its ID number rather than its reference.

same global (global id, *value*)
-Uses the same syntax as “inc global” and “dec global,” but works exactly like “write global.” Useful only for keeping it in the family.

fair random (*value*)
-A scale for keeping random numbers fairly balanced. It rolls two numbers between 1 and the *value* and attempts to match them, then returns *true* or *false* based on the result. Works as a condition and is best used within a *for/do* or *while* block that counts each failure to match and breaks on success, using the final count as the designated “random value.” The lower the *value*, the better the results. Defaults to 10.

Example:

```
script, how many pizzas I have, begin
  variable (f)
  global match := 0
  for (f, 1, 50) do(
    if (fair random (20)) then(
      global match := f
      show text box (1)  #I have 12 pizzas
      wait for text box
      break
    )
  )
  if (global match == 0) then(
    show text box (2)  #I have no pizzas
    wait for text box
  )
```

```
end
```

In the above example, "global match" is a global variable embedded in textbox (1) that reports how many pizzas I have. Instead of picking any number at random between 1 and 20 to represent my pizza count, the script runs a loop as many as fifty times to attempt to match two random values between 1 and 20, and it manages to make that match on the 12th loop. So I have 12 pizzas. Alternatively, if I run through 50 loops without a match, then the script will run the second textbox, declaring that I have no pizzas.

Note that this is just one way to use "fair random." It can just as easily be used in a nuanced pass/fail situation. It can also be used in a double for/do loop where the outer loop records the count and the inner loop (of say 10 cycles) will count only if a match is made before the loop expires. So, if the *g* loop (outer loop) runs 50 times, but the *f* loop (inner loop) runs only 10 times and fails on most loops, then the *g* loop may end up counting only three or four *f* loop successes, meaning that I have only three or four pizzas. So, "fair random" should be used in whichever way best suits the "fairness" of the script's value determination.

smart balance (global id, low, high)
-Takes the value of a variable and "nudges" it to just inside the range defined by *low* and *high*. Best used for capping values at either end of the scale. *Global ID* should be written as "@variable" if using reference instead of ID number.

--Group Everything Functions--

This section covers the all-purpose function used to store any value into a group for quick reference.

Notes: "Mass function" should be used with a for/do script to cycle through all of the variables.
Use "generalfunctionvar" as the value for the intended function within the for/do block.
This is an all-purpose way to shortcut all group functions into a single command.
Always "clear mass functions" before running a for/do block containing "mass functions."

mass function (*val1*, *val2*, …, *val24*)
-Uses the global variable "generalfunctionvar" to store the value listed in the first argument. When run a second time, it will store the value listed in the second argument. This can be done up to 24 times before it needs to be reset. Best used in a *for/do* block.

Note: The purpose of this function is to loop through up to 24 values that a single function can use, like forcing (up to) 24 NPCs to stop walking, or choosing (up to) 24 unique unconnected textboxes to display in a row, for example. The function that uses the stored value must be run before the loop cycles, or else the next argument will overwrite it before it runs. The global variable "generalfunctionvar" must be written into the subsequent function to use the values defined in "mass function." Example:

```
script, all purpose example, begin
  variable (f)
```

```
  clear mass functions
  for (f, 1, 3) do(
    mass function (4, 7, 9)
    show text box (generalfunctionvar)
    wait for text box
  )
end
```

The above script will display text boxes 4, 7, and 9 in a row before the loop breaks. Make sure the loop count matches the number of arguments defined in “mass function” for maximum effect.

clear mass functions
-Function needed to clear the values associated with “mass function.” Should be run *before* the loop that uses “mass function.”

Movement Shortcuts:

This section combines simple movements with waits. These functions minimize the complexity in scripts where every line is a move command or a wait command.

--Hero Movements--

This section focuses on short hero movements. It assumes that heroes are either solo or part of a "suspend caterpillar" order in some cases.

basic hero walk (who, direction, distance, *frame*)
-Substitute for "walk hero." Includes the "wait for hero" command. Allows for an optional end frame if needed.

basic hero walk and turn (who, direction, distance, stop direction, *frame*)
-Same as "basic hero walk" but turns the hero in a new direction when stopping. "Stop direction" should be defined as the end direction (north, east, south, west).

basic hero walk and 180 turn (who, direction, distance, *spin*, *time*, *frame*)
-Same as "basic hero walk" but turns the hero in the opposite direction when stopping. "Spin" determines which way the hero turns, "left" or "right" (defaults right). "Time" represents how many ticks pass before the hero makes a turn (defaults to two ticks). Note: Left turns counterclockwise; right turns clockwise.

walk hero and wait (who, direction, distance, *frame*)
-Syntax-friendly version of "basic hero walk." Works exactly as it implies.

walk hero and turn (who, direction, distance, stop direction, *frame*)
-Same.

walk hero and 180 turn (who, direction, distance, *spin*, *time*, *frame*)
-Same.

walk hero and smart turn (who, direction, distance, stop direction, *ticks*, *frame*)
-Walks hero in the designated direction at the designated distance, then turns to the designated stop direction using the lowest required frames. Defaults to two ticks per turn frame.

walk hero to x and wait (who, x, *frame*)
-Does exactly what it says.

walk hero to y and wait (who, y, *frame*)

walk hero to x and turn (who, x, direction, *frame*)

walk hero to y and turn (who, y, direction, *frame*)

walk hero to x and smart turn (who, x, direction, *ticks*, *frame*)

walk hero to y and smart turn (who, y, direction, *ticks*, *frame*)

walk hero to x and 180 turn (who, x, *spin*, *time*, *frame*)

walk hero to y and 180 turn (who, y, *spin*, *time*, *frame*)

hero spin 180 (who, *spin*, *time*)
-Faces the selected hero in the opposite direction through a timed rotation.

walk hero to x and y (who, x, y, *no wait*)
-Moves hero directly to the x and y-coordinate, creating a potential diagonal path. "No wait" determines whether the script waits for the hero to stop before calling the next function. Setting it to "true" means the function will *not* wait for hero to stop.

walk hero to y and x (who, y, x, *no wait*)
-Identical to "walk hero to x and y." May move the hero north or south on launch instead of east or west, which the former function may do.

walk hero to x then y (who, x, y)
-Moves hero to x-coordinate before moving to y. Triggers "wait for hero" after each move.

walk hero to y then x (who, y, x)

walk hero to x then y then turn (who, x, y, direction, *smart*, *ticks*, *frame*)
-Ends an x/y walk cycle with a finishing turn. "Smart" determines whether to point the hero in the ending direction directly or "spin" it into positing using "smart turn." Defaults to *false* (standard turn). Set "smart" to *true* to enable "smart turn." "Ticks" determines how much time should pass between turn directions (defaults to 2 ticks). "Frame" determines which frame the image should end on (0 or 1, defaults to 0).

walk hero to y then x then turn (who, y, x, direction, *smart*, *ticks*, *frame*)

walk hero with camera (who, direction, distance, *camera direction*, *camera distance*, *camera speed*)
-Pans the camera with the hero's basic walk movement. Useful for syncing camera movement with the hero without necessarily attaching it to the hero's position. Leave the last three arguments blank to keep the camera synced with the hero (for example, tracking the hero's movement from several tiles above). Add camera direction, distance, and speed if you want the camera to move elsewhere as the hero moves. Note: Use this function and its siblings at your own risk. It has the potential for shenanigans.

walk hero to x with camera (who, x, *camera direction*, *camera distance*, *camera speed*)
-Similar to "walk hero with camera" but calculates distance between the hero's starting point and destination point to maintain tracking functionality when camera controls are left alone. Has the potential for an epic fail.

walk hero to y with camera (who, y, *camera direction*, *camera distance*, *camera speed*)

walk hero to x and y with camera (who, x, y, *camera direction*, *camera distance*, *camera speed*)

walk hero to y and x with camera (who, y, x, *camera direction*, *camera distance*, *camera speed*)

hero turn (who, direction, *spin*, *time*, *frame*)
-Shows hero rotation when turning from one direction to another. "Spin" defines which way the hero turns toward the target direction (right = clockwise; left = counterclockwise). "Time" determines how many ticks pass before the hero turns to the next position. "Frame" determines the end frame.

hero smart turn (who, direction, *time*, *frame*)
-Same as "hero turn," but uses the shortest distance between source and destination direction to complete the turn.

--NPC Movements--

These functions are identical to their "hero" counterparts.

basic npc walk (who, direction, distance, *frame*)

basic npc walk and turn (who, direction, distance, stop, *frame*)

basic npc walk and 180 turn (who, direction, distance, *spin*, *time*, *frame*)

walk npc and wait (who, direction, distance, *frame*)

walk npc and turn (who, direction, distance, stop direction, *frame*)

walk npc and smart turn (who, direction, distance, *tick*, *frame*)

walk npc and 180 turn (who, direction, distance, *spin*, *time*, *frame*)

walk npc to x and wait (who, x, *frame*)

walk npc to y and wait (who, y, *frame*)

walk npc to x and turn (who, x, direction, *frame*)

walk npc to y and turn (who, y, direction, *frame*)

walk npc to x and smart turn (who, x, direction, *ticks*, *frame*)

walk npc to y and smart turn (who, y, direction, *ticks*, *frame*)

walk npc to x and 180 turn (who, x, *spin*, *time*, *frame*)

walk npc to y and 180 turn (who, y, *spin*, *time*, *frame*)

npc spin 180 (who, *spin*, *time*)

walk npc to x and y (who, x, y, *no wait*)

walk npc to y and x (who, y, x, *no wait*)

walk npc to x then y (who, x, y)

walk npc to y then x (who, y, x)

walk npc to x then y then turn (who, x, y, direction, *smart*, *ticks*, *frame*)

walk npc to y then x then turn (who, y, x, direction, *smart*, *ticks*, *frame*)

walk npc with camera (who, direction, distance, *camera direction*, *camera distance*, *camera speed*)

walk npc to x with camera (who, x, *camera direction*, *camera distance*, *camera speed*)

walk npc to y with camera (who, y, *camera direction*, *camera distance*, *camera speed*)

walk npc to x and y with camera (who, x, y, *camera direction*, *camera distance*, *camera speed*)

walk npc to y and x with camera (who, y, x, *camera direction*, *camera distance*, *camera speed*)

npc turn (who, direction, *spin*, *time*, *frame*)

npc smart turn (who, direction, *time*, *frame*)

--Advanced Hero Movements--

This section focuses on advanced hero movements for more specialized actions.

reset hero speed (who)
-Returns the hero to its previously recorded speed. Note: This command works only if the global variable "store speed" has been used to store the hero's original speed before changing it to something else. **Special Note:** This function is embedded in all "blow hero to" commands and does not need to be called twice.

speed walk hero (who, direction, distance, speed)
-Temporarily changes a hero's speed before sending him walking, then sets it back to normal at the end of the walk.

speed walk hero to x (who, x, speed)
-Temporarily changes a hero's speed before walking him to an x-coordinate, then changes it back to normal when he reaches his destination.

speed walk hero to y (who, y, speed)
-Ditto, but to a y-coordinate.

ghost walk hero (who, direction, distance)
-Allows a hero to walk through any wall or obstruction before returning it to its normal, non-passable state.

ghost walk hero to x (who, x)
-Removes a hero's passability rules before walking him to an x-coordinate.

ghost walk hero to y (who, y)
-Ditto, but to a y-coordinate.

hero spin (who, *rotations*, *spin*, *ticks*, *frame*)
-Spins a hero a full 360-degree circle. "Rotations" determines how many spins; defaults to 1. "Spin" determines whether the hero turns right or left, defaults to *right*. "Ticks" sets the amount of time it takes for the hero to change direction (defaults to 2). "Frame" determines which frame the hero ends on when he's done spinning.

hero opposite direction (who)
-Points the hero in the opposite direction from where he's facing.

moonwalk hero (who, direction, distance)
-Points the hero away from the direction he's walking.

moonwalk hero to x (who, x)

moonwalk hero to y (who, y)

moonwalk hero to x and y (who, x, y)

moonwalk hero to y and x (who, y, x)

moonwalk hero to x then y (who, x, y)

moonwalk hero to y then x (who, y, x)

hero staircase (who, distance, *direction1*, *direction2*, *step*)
-Walks a hero in a "staircase" or zigzagging pattern. "Distance" determines how far from the origin point the hero will walk. "Direction1" is the starting direction up the step; defaults to *up*. "Direction2" is the crossing direction to the next step; defaults to *right*. "Step" determines the "roughness" or "smoothness" of the step. A value of 0 creates a "smooth" or diagonal movement. A value of 1 or higher changes the size of the step. Defaults to 1.

smooth staircase (who, direction1, direction2, distance)
-A direct command to move a hero up a smooth diagonal line.

rough staircase (who, direction1, direction2, distance, step)
-A direct command to move a hero up a jagged diagonal line.

rough staircase plus (who, direction1, direction2, distance, step)
-A direct command to move a hero up a jagged diagonal line and finish with one extra "direction1" movement.

hero square (who, direction1, direction2, distance)
-Walks a hero in a square-shaped pattern. "Direction1" and "direction2" determine the first two movements, and "distance" determines the size of the square. Hero will return to the origin after completing a square movement.

hero diamond (who, direction1, direction2, distance)
-Same as "hero square" but moves the hero in a diamond pattern.

walk hero in-line (who, direction, distance, face direction)
-Walking script designed for "strafing." Can also be used to "moonwalk" if you don't want to use "moonwalk hero." "Face direction" is set after the hero starts walking.

walk hero in-line to x (who, x, face direction)

walk hero in-line to y (who, y, face direction)

walk hero in-line to x and wait (who, x, face direction)

walk hero in-line to y and wait (who, y, face direction)

walk hero in-line to x and turn (who, x, face direction, end direction)

walk hero in-line to y and turn (who, y, face direction, end direction)

walk hero in-line to x and y (who, x, y, face direction)

walk hero in-line to y and x (who, y, x, face direction)

walk hero in-line to x then y (who, x, y, face direction 1, *face direction 2*)
-This walking series allows for two different facing directions for x and y. Leave "face direction 2" blank if you want the hero to maintain the same facing direction as the first.

walk hero in-line to y then x (who, y, x, face direction 1, *face direction 2*)

hero wind walk (who, direction, distance, *speed*, *face direction*)
-Moves a hero without animation, as if floating on wind.

blow hero to x (who, x, *speed*, *face direction*)
-Same as "hero wind walk," but sends hero to a set x position.

blow hero to y (who, y, *speed*, *face direction*)
-Same as "hero wind walk," but sends hero to a set y position.

blow hero to x and wait (who, x, *speed*, *face direction*)

blow hero to y and wait (who, y, *speed*, *face direction*)

blow hero to x and turn (who, x, direction, *speed*, *face direction*)

blow hero to y and turn (who, y, direction, *speed*, *face direction*)

blow hero to x and y (who, x, y, *speed*, *face direction*)

blow hero to y and x (who, y, x, *speed*, *face direction*)

blow hero to x then y (who, x, y, *speed*, *face direction 1*, *face direction 2*)

blow hero to y then x (who, y, x, *speed*, *face direction 1*, *face direction 2*)

hero float (who)
-Sets the hero's z-axis one complete tile, two pixels at a time.

hero drop (who)
-Returns the hero to earth.

--Advanced NPC Movements--

Unless otherwise noted, these commands are the NPC equivalents to "Advanced Hero Movements."

reset npc speed (who)

speed walk npc (who, direction, distance, speed)

speed walk npc to x (who, x, speed)

speed walk npc to y (who, y, speed)

ghost walk npc (who, direction, distance)

ghost walk npc to x (who, x)

ghost walk npc to y (who, y)

npc spin (who, *rotations*, *spin*, *ticks*, *frame*)

moonwalk npc (who, direction, distance)

moonwalk npc to x (who, x)

moonwalk npc to y (who, y)

moonwalk npc to x and y (who, x, y)

moonwalk npc to y and x (who, y, x)

moonwalk npc to x then y (who, x, y)

moonwalk npc to y then x (who, y, x)

npc staircase (who, distance, *direction1*, *direction2*, *step*)

npc smooth staircase (who, direction1, direction2, distance)
-Note: This function should defer to "npc staircase."

npc rough staircase (who, direction1, direction2, distance, step)
-Note: This function should defer to "npc staircase."

npc rough staircase plus (who, direction1, direction2, distance, step)

npc square (who, direction1, direction2, distance)

npc diamond (who, direction1, direction2, distance)

walk npc in-line (who, direction, distance, face direction)

walk npc in-line to x (who, x, face direction)

walk npc in-line to y (who, y, face direction)

walk npc in-line to x and wait (who, x, face direction)

walk npc in-line to y and wait (who, y, face direction)

walk npc in-line to x and turn (who, x, face direction, end direction)

walk npc in-line to y and turn (who, y, face direction, end direction)

walk npc in-line to x and y (who, x, y, face direction)

walk npc in-line to y and x (who, y, x, face direction)

walk npc in-line to x then y (who, x, y, face direction 1, *face direction 2*)

walk npc in-line to y then x (who, y, x, face direction 1, *face direction 2*)

npc wind walk (who, direction, distance, *speed*, *face*)

blow npc to x (who, x, *speed*, *face direction*)

blow npc to y (who, y, *speed*, *face direction*)

blow npc to x and wait (who, x, *speed*, *face direction*)

blow npc to y and wait (who, y, *speed*, *face direction*)

blow npc to x and turn (who, x, direction, *speed*, *face direction*)

blow npc to y and turn (who, y, direction, *speed*, *face direction*)

blow npc to x and y (who, x, y, *speed*, *face direction*)

blow npc to y and x (who, y, x, *speed*, *face direction*)

blow npc to x then y (who, x, y, *speed*, *face direction 1*, *face direction 2*)

blow npc to y then x (who, y, x, *speed*, *face direction 1*, *face direction 2*)

npc float (who)

npc drop (who)

door to door (who, x, y, direction, *movetype1*, *movetype2*)
-Simulates walking an NPC through one door and emerging in another. X and Y determine the coordinates for the exit door. "Direction" determines which way the NPC is facing on exit. "Movetypes" default to *standstill* and *wander* respectively.

Note: "Door to door" works best in a timer-triggered script. It checks the position of a moving NPC, and when the NPC walks on a specific tile or zone (or whatever the script checks), it converts the NPC to a plotscript-controlled NPC, moves it toward the door (assuming the door sits one tile away in the direction called), and teleports the NPC to the door at x/y and walks it out onto the floor (using the same direction). For best results, create connecting doors that adhere to the same directional standards (example: moving up into one door means having to move up to exit the adjoining door). Connected doors should be placed on the same map.

--Group Movements--

This section provides information on functions designed to move entire groups of heroes. Some functions may depend on a "suspend caterpillar" order to function properly; be advised. There is currently no NPC counterpart. Feel free to make your own! Most hero arguments are optional. Plugging in custom values (0-3) will change hero order. Leaving the hero arguments alone will tell the script to start with the first position in the party (0) and end with the last (3).

Note: Group functions assume all available party heroes apply to the function. **Selective group** functions assume that only those party heroes assigned by "group size" apply to the function. You'll want to use "selective group" for any command that requires flexibility in group manipulation. Remember to define the group's size in the first argument before you issue directions.

group walk (direction 1, distance 1, direction 2, distance 2, direction 3, distance 3, direction 4, distance 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-A single line function that determines direction and travel distance for all four heroes (or however many are found in the active party). Defaults to moving each hero according to position in party. Last four arguments should list preferred move order if default order is not desired. Best used under a "suspend caterpillar" order. Includes wait order. Note: Undefined directions default to "north," and undefined distances default to zero.

group wait (*group size*, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Determines which party members should wait after a group move order. "Group size" determines how many party members should obey the wait order (defaults to 4). The remaining arguments should respect the size value (up to 4). Leave all arguments blank if you want the entire party to wait at the same time.

group same walk (direction, distance, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Similar to "group walk" but sends all available heroes in the same direction and distance.

group separate walk (direction, distance 1, distance 2, distance 3, distance 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends all party members in the same direction but at different distances.

group split walk (distance, direction 1, direction 2, direction 3, direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends all party members the same distance but in different directions.

group walk to x (x1, x2, x3, x4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Same as "group walk" but uses an x location as the destination. Each x-coordinate corresponds to the hero for that slot.

group walk to y (y1, y2, y3, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)

group walk to same x (x, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Same as "group walk to x" but sends all available party members to the same x-coordinate. Useful for moving columns of heroes east or west.

group walk to same y (y, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Useful for moving rows of heroes north or south.

group turn (hero 1, direction 1, hero 2, direction 2, hero 3, direction 3, hero 4, direction 4)
-Turns all available party members simultaneously in their own individual directions.

group turn and wait (turn speed, hero 1, direction 1, hero 2, direction 2, hero 3, direction 3, hero 4, direction 4)
-Exactly like "group turn" but forces each party member to wait until the previous one turns. "Turn speed" determines how much time (in ticks) passes before the next hero turns.

group same turn (direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Simultaneously turns all available heroes in the same direction.

group same turn and wait (turn speed, direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)

selective group walk (group size, *hero 1*, *direction 1*, *distance 1*, *hero 2*, *direction 2*, *distance 2*, *hero 3*, *direction 3*, *distance 3*, *hero 4*, *direction 4*, *distance 4*, *no wait*)
-All "selective" group functions work like their non-selective counterparts, but they focus on chosen party members, not available party members. Selective groups *must* be built by "group size" before their actions can be defined. Excluded members will not follow the order. All "directions" default to *north*. All "no waits" default to *false*; set to "true" if you want active walking orders to continue during the next command.

selective group same walk (group size, direction, distance, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group separate walk (group size, direction, *hero 1*, *distance 1*, *hero 2*, *distance 2*, *hero 3*, *distance 3*, *hero 4*, *distance 4*, *no wait*)

selective group split walk (group size, distance, *hero 1*, *direction 1*, *hero 2*, *direction 2*, *hero 3*, *direction 3*, *hero 4*, *direction 4*, *no wait*)

selective group walk to x (group size, *hero 1*, *x1*, *hero 2*, *x2*, *hero 3*, *x3*, *hero 4*, *x4*, *no wait*)

selective group walk to y (group size, *hero 1*, y*1*, *hero 2*, y*2*, *hero 3*, y*3*, *hero 4*, y*4*, *no wait*)

selective group walk to same x (group size, x, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group walk to same y (group size, y, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group turn (group size, *hero 1*, *direction 1*, *hero 2*, *direction 2*, *hero 3*, *direction 3*, *hero 4*, *direction 4*)

selective group turn and wait (group size, turn speed, *hero 1*, *direction 1*, *hero 2*, *direction 2*, *hero 3*, *direction 3*, *hero 4*, *direction 4*)

selective group same turn (group size, direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)

selective group same turn and wait (group size, turn speed, direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)

--Advanced Group Movements--

This section provides all of the advanced movement commands issued to groups and selective groups. Note: All group movements imply a "group wait." Selective group movements allow for group wait overrides.

group reset hero speed
-Assuming the global variable "store speed" has been set previously, this function will reset all heroes to the previously defined speed. **Caution:** Do not use this command without first setting "store speed" or using a function that automatically sets "store speed." For safety, you can assign "store speed" to the default hero speed at the start of each game to ensure this command will always return heroes to their proper speeds.

group spin (*rotations*, *spin*, *ticks*, *frame*)
-Causes every member in the party to spin. "Rotations" defaults to 1; "spin" defaults to "right"; "ticks" defaults to 2; "frame" defaults to 0 (or left).

group opposite direction
-Automatically flips all heroes in the active party to the opposite direction.

group moonwalk (direction 1, distance 1, direction 2, distance 2, direction 3, distance 3, direction 4, distance 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Causes all heroes in the active party to moonwalk (or walk backwards) the designated distance in the designated direction, according the hero's slot. Directions default to "north."

group same moonwalk (direction, distance, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Works like "group moonwalk" but sends all members of the active party the same distance and in the same direction.

group separate moonwalk (direction, distance 1, distance 2, distance 3, distance 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends all members of the active party moonwalking in the same direction but at different distances.

group moonwalk to x (x1, x2, x3, x4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends all members of the active party moonwalking to a variety of x-coordinates. Only those in the active party need a position declaration. Empty positions can be left blank.

group moonwalk to y (y1, y2, y3, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Same as "group moonwalk to x," but moonwalks everyone in the active party to a y-coordinate.

group moonwalk to same x (x, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends everyone in the active party moonwalking to the same x-coordinate. This is useful if you want to create a vertical line of heroes. Make sure everyone is on a different y-coordinate first.

group moonwalk to same y (y, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-The same as "group moonwalk to same x," but on the opposite axis.

group in-line walk (move direction 1, distance 1, face direction 1, move direction 2, distance 2, face direction 2, move direction 3, distance 3, face direction 3, move direction 4, distance 4, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Allows for group "strafing." All movement directions default to "north." All facing directions also default to north.

group same in-line walk (move direction, distance, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends the group strafing in the same direction and distance.

group same in-line walk different face (move direction, distance, face direction 1, face direction 2, face direction 3, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Sends the group strafing in the same direction and distance, but with each one facing in different directions.

group in-line walk to x (face direction, x1, x2, x3, x4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Based on the hero's position in the party, sends the group strafing to an x-coordinate while facing in the same direction.

group in-line walk to y (face direction, y1, y2, y3, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Based on the hero's position in the party, sends the group strafing to a y-coordinate while facing in the same direction.

group in-line walk to same x (x, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to the same x-coordinate, facing the same direction.

group in-line walk to same y (y, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to the same y-coordinate, facing the same direction.

group in-line walk to x different face (x1, face direction 1, x2, face direction 2, x3, face direction 3, x4, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to slot-based x-coordinate while facing in different directions.

group in-line walk to y different face (y1, face direction 1, y2, face direction 2, y3, face direction 3, y4, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to slot-based y-coordinate while facing in different directions.

group in-line walk to same x different face (x, face direction 1, face direction 2, face direction 3, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to same x-coordinate while facing in different directions.

group in-line walk to same y different face (y, face direction 1, face direction 2, face direction 3, face direction 4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Group strafing to same y-coordinate while facing in different directions.

selective group opposite direction (group size, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-All "selective" group functions work like their non-selective counterparts, but they focus on chosen party members, not available party members. Selective groups *must* be built by "group size" before their actions can be defined. Excluded members will not follow the order. All "directions" default to *north*. All "no waits" default to *false*; set to "true" if you want active walking orders to continue during the next command.

selective group moonwalk (group size, *hero 1*, *direction 1*, *distance 1*, *hero 2*, *direction 2*, *distance 2*, *hero 3*, *direction 3*, *distance 3*, *hero 4*, *direction 4*, *distance 4*, *no wait*)

selective group same moonwalk (group size, direction, distance, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group separate moonwalk (group size, direction, *hero 1*, *distance 1*, *hero 2*, *distance 2*, *hero 3*, *distance 3*, *hero 4*, *distance 4*, *no wait*)

selective group moonwalk to x (group size, *hero 1*, *x1*, *hero 2*, *x2*, *hero 3*, *x3*, *hero 4*, *x4*, *no wait*)

selective group moonwalk to y (group size, *hero 1*, *y1*, *hero 2*, *y2*, *hero 3*, *y3*, *hero 4*, *y4*, *no wait*)

selective group moonwalk to same x (group size, x, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group moonwalk to same y (group size, y, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group in-line walk (group size, face direction, *hero 1*, *move direction 1*, *distance 1*, *hero 2*, *move direction 2*, *distance 2*, *hero 3*, *move direction 3*, *distance 3*, *hero 4*, *move direction 4*, *distance 4*, *no wait*)

selective group same in-line walk (group size, move direction, distance, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group in-line walk to x (group size, face direction, *hero 1*, *x1*, *hero 2*, *x2*, *hero 3*, *x3*, *hero 4*, *x4*, *no wait*)

selective group in-line walk to y (group size, face direction, *hero 1*, *y1*, *hero 2*, *y2*, *hero 3*, *y3*, *hero 4*, *y4*, *no wait*)

selective group in-line walk to same x (group size, x, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group in-line walk to same y (group size, y, face direction, *hero 1*, *hero 2*, *hero 3*, *hero 4*, *no wait*)

selective group in-line walk different face (group size, *hero 1*, *move direction 1*, *distance 1*, *face direction 1*, *hero 2*, *move direction 2*, *distance 2*, *face direction 2*, *hero 3*, *move direction 3*, *distance 3*, *face direction 3*, *hero 4*, *move direction 4*, *distance 4*, *face direction 4*, *no wait*)

selective group same in-line walk different face (group size, move direction, distance, *hero 1*, *face direction 1*, *hero 2*, *face direction 2*, *hero 3*, *face direction 3*, *hero 4*, *face direction 4*, *no wait*)

selective group in-line walk to x different face (group size, *hero 1*, *x1*, *face direction 1*, *hero 2*, *x2*, *face direction 2*, *hero 3*, *x3*, *face direction 3*, *hero 4*, *x4*, *face direction 4*, *no wait*)

selective group in-line walk to y different face (group size, *hero 1*, *y1*, *face direction 1*, *hero 2*, *y2*, *face direction 2*, *hero 3*, *y3*, *face direction 3*, *hero 4*, *y4*, *face direction 4*, *no wait*)

selective group in-line walk to same x different face (group size, x, *hero 1*, *face direction 1*, *hero 2*, *face direction 2*, *hero 3*, *face direction 3*, *hero 4*, *face direction 4*, *no wait*)

selective group in-line walk to same y different face (group size, y, *hero 1*, *face direction 1*, *hero 2*, *face direction 2*, *hero 3*, *face direction 3*, *hero 4*, *face direction 4*, *no wait*)

--Position Shortcuts--

This section introduces shortcuts for advance character placement.

basic hero position (who, x, y, direction, *frame*)
-Works like "set hero position" but also allows for direction and frame settings on placement.

basic npc position (who, x, y, direction, *frame*)

basic hero offscreen (who, *position*)
-Quick function that places heroes in the upper left corner of the screen, based on position, up to 15 slots per column. Position zero will place the hero at coordinate (0, 0). Any position over 15 will place the hero on a block corresponding to the remainder of 15 in the adjacent column. Note: This function works best on maps that do not allow access to the left edge of the screen and those that can hide heroes under overhead tiles. Undefined positions will default to zero. Leave the "position" argument blank if you want all offscreen heroes to hide at the (0, 0) position. For maps where the top-left corner of the screen is visible to players, it *might* be useful to use negative position values instead.

basic npc offscreen (who, *position*)

set hero direction and frame (who, direction, frame)
-Advanced "set hero direction" by adding a condition for hero's frame. Good for using single frames to end a movement or cue up an animation.

set npc direction and frame (who, direction, frame)

hero look opposite (who)
-Alternative command to "hero opposite direction." Tells the hero to reverse direction.

npc look opposite (who)

hero look clockwise (who)
-Turns the hero to the next clockwise position.

npc look clockwise (who)

hero look counterclockwise (who)
-Turns the hero to next counterclockwise position.

npc look counterclockwise (who)

position hero clockwise (who)
-Sets the hero on an adjacent tile in the clockwise position (based on diagonal movement). Note: This does not walk the hero to the adjacent spot; it drops him there. Ideal for turn-based strategies or chess simulations.

position npc clockwise (who)

position hero counterclockwise (who)

position npc counterclockwise (who)

group line x position (direction, x, y1, y2, y3, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-This function allows for group positioning based on a single x-coordinate. Useful for creating a column of heroes.

group line y position (direction, y, x1, x2, x3, x4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-This function allows for group positioning based on a single y-coordinate. Useful for creating a row of heroes.

group position (direction, x1, y1, x2, y2, x3, y3, x4, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-This function allows for flexible group positioning.

group offscreen (*position 1*, *position 2*, *position 3*, *position 4*, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Positions all members of the active party along the top-left edge of the map. Defaults to (0, 0). Note: Using negative positions *may* be useful for maps where the top-left edge is visible to the player.

selective group line x position (group size, direction, x, y1, y2, y3, y4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Works like "group line x position" but moves only those heroes defined in the function. "Group size" must be defined before assigning heroes to positions. Any vacant position after the group size cap is met will be ignored.

selective group line y position (group size, direction, y, x1, x2, x3, x4, *hero 1*, *hero 2*, *hero 3*, *hero 4*)

selective group position (group size, direction, *hero 1*, *x1*, *y1*, *hero 2*, *x2*, *y2*, *hero 3*, *x3*, *y3*, *hero 4*, *x4*, *y4*)

selective group offscreen (group size, *hero 1*, *position 1*, *hero 2*, *position 2*, *hero 3*, *position 3*, *hero 4*, *position 4*)

--Hero and NPC Pixel Movements--

This section provides movement shortcuts based on pixels, using "px plus" and "py plus" global variables to function. Remember to clear these values at the end of each use to prevent corruption. Because these functions share the same global variables, it's advisable to use each function one at a time. If you need to move multiple heroes or NPCs at the same time, you'll want to define alternative "store" functions using alternative global variables to prevent chaos.

store hero pixel position (who)
-Stores the hero's position by pixel coordinates. Run this function before moving heroes by pixel. Can only handle one hero at a time. Define a new "store" function with alternative global variables to adjust multiple heroes at once.

store npc pixel position (who)
-Stores the NPC's position by pixel coordinates. Run this function before moving NPCs by pixel. Can only handle one NPC at a time. Define a new "store" function with alternative global variables to adjust multiple NPCs at once. Note: This function uses the same global variables as "store hero pixel position," so they should not be used at the same time.

reset pixel position
-Resets "px plus" and "py plus" to zero, making them fresh for new use.

hero pixel move (who, px, py, value, distance, direction)
-Moves the hero by pixel. "Value" determines how many pixels the hero moves across. "Distance" determines how far the hero moves each frame. A greater distance simulates faster speeds. "Direction" should be defined according to north, east, south, west constants.

npc pixel move (who, px, py, value, distance, direction)

diagonal hero pixel move (who, px, py, value, distance 1, direction 1, distance 2, direction 2)

diagonal npc pixel move (who, px, py, value, distance 1, direction 1, distance 2, direction 2)

hero pixel equalize (who, px, py)
-Sets the hero back on the tile, using "px plus" and "py plus."

npc pixel equalize (who, px, py)

--NPC Passage Shortcuts--

This section simplifies NPC movement and passage rules.

npc free movement (who)
-Allows NPCs to move through walls and obstacles.

npc default movement (who)
-Resets NPC movement rules to the default. Note: This function should be called when the NPC is no longer needed if "npc free movement" is called previously.

npc free movement wrap (*npc 1*, *npc 2*, *npc 3*, *npc 4*, *npc 5*, *npc 6*, *npc 7*, *npc 8*)
-Set movement rules to "free" for up to eight NPCs at a time.

npc default movement wrap (*npc 1*, *npc 2*, *npc 3*, *npc 4*, *npc 5*, *npc 6*, *npc 7*, *npc 8*)

mass npc manipulation (stattype, value, *npc 1*, *npc 2*, *npc 3*, *npc 4*, *npc 5*, *npc 6*, *npc 7*, *npc 8*)
-Implements a mass "alter npc" command for up to eight NPCs. "Stattype" refers to which stat (example: *npcstat:movetype*). "Value" refers to the item to change (example: *NPCmovetype:standstill* or 0).

wait for npcs (*npc 1*, *npc 2*, *npc 3*, *npc 4*, *npc 5*, *npc 6*, *npc 7*, *npc 8*)
-Waits for up to eight NPCs to stop moving.

--Camera Movement Shortcuts--

This section simplifies camera movements to include "wait" commands.

focus camera and wait (x, y, speed)
-Explanatory.

pan camera and wait (direction, distance, speed)
-Ditto.

--Bonus Movement Shortcuts (for Fine-tuning)--

This section features functions that "polish" movement.

hero wait (who)
-Advanced wait function that realigns hero to the tile. Best used where tile misalignment is likely.

npc wait (who)

hero align (who)
-Identical to "hero pixel equalize" but does not need global variables to work. Can also be used on multiple heroes. Probably the more flexible option.

npc align (who)

npc opposite direction (who)
-Quick function for setting the NPC in the opposite direction. Use twice to revert the NPC to its starting position.

npc talk (who)
-Quality of life function that forces the NPC to stop and face the hero when engaged in a conversation. Prevents the NPC from wandering off or looking into the distance during conversation. Depends on the global variable "npc behavior" to work properly.

npc finish (who)
-Allows the NPC to resume movement behavior after a conversation ends. Best used *after* the conversation is over.

npc conversation (who, *text 1*, *tick 1*, *text 2*, *tick 2*, *text 3*, *tick 3*, *text 4*, *tick 4*, *text 5*, *tick 5*, *text 6*, *tick 6*, *text 7*, *tick 7*, *text 8*, *tick 8*)
-Inserts timed pauses between blocks of text for more nuanced conversational beats. Note: This does not affect the delivery of chained text, just those that are called via the script.

stop npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "standstill."

stand still npc (who)
-Same as "stop npc" but more consistent with "movetype" syntax.

wander npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "wander."

pace npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "pace."

right turn npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "right turns."

left turn npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "left turns."

random turn npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "random turns."

chase you meander npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "chase you (meander)."

chase you direct npc (who)
-Implements a movement behavior change to NPC. Changes behavior to "chase you (direct)."

chase you pathfinding npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “chase you (pathfinding).”

avoid you meander npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “avoid you (meander).”

avoid you direct npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “avoid you (direct).”

walk in place npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “walk in place.”

follow walls right npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “follow walls right.”

follow walls left npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “follow walls left.”

follow walls right and stop npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “follow walls right and stop for others.”

follow walls left and stop npc (who)
-Implements a movement behavior change to NPC. Changes behavior to “follow walls left and stop for others.”

Character Display and Animation Shortcuts:

This section covers functions that simplify repetitious image display orders.

hero short frame (who, direction, time 1, *time 2*, *set frame*)
-A single switch between two character frames (from the same direction). Displays both frame 0 and frame 1 (or vice versa) and allows for a set time (in ticks) between them. "Time 1" references the amount of time that should pass (in ticks) before the second frame is displayed. "Time 2" references the amount of time that should pass (in ticks) before the function expires (leave blank or "-1" if identical to "time 1"). Change "set frame" to "false" if you want to display frame 1 before frame 0.

hero short frame cycle (range, who, start, finish, time, *exact*, *extra start time*, *extra end time*)
-Allows for a repetitive cycle of hero frame flips. "Range" determines how many cycles pass before the function expires. "Start" designates the starting frame (0 or 1); "finish" designates the final frame (1 or 0). "Time" determines the common time between frame flips. Setting the optional argument "exact" to "true" tells the function to use "waitsecs" instead of "wait," allowing for music-timed flips if needed (leave "false" to use standard waits). "Extra start time" and "extra end time" are optional and allow for flexible times between the first and second frames. These values should determine the *extra* time added to the base time; they do not substitute the base time.

npc short frame (who, direction, time 1, *time 2*, *set frame*)

npc short frame cycle (range, who, start, finish, time, *exact*, *extra start time*, *extra end time*)

hero exact frame (who, picture, *direction*, *frame*)
-Quick function to display an exact hero image onscreen, using picture, direction, and frame arguments. Defaults to "north, 0."

hero exact frame and revert (who, picture, *direction*, *frame*, *ticks*, *palette*)
-Displays an updated "exact frame" for hero, then reverts back to the previous image after a set time. "Direction" and "frame" default to the hero's current direction and frame. "Ticks" defaults to 2 ticks. "Palette" defaults to the default palette.

two npc direction and frame (npc 1, npc 2, direction, *frame*, *ticks*)
-Sets two NPCs by direction and frame at once. "Ticks" references how long to wait before next command is issued.

two npc direction and frame cycle (npc 1, npc 2, direction, *frame*, *ticks*, *reverse*)
-Cycles through a complete left, right cycle of the given direction for two NPCs. "Ticks" references how much time passes between frame changes. "Reverse" determines whether the animation goes from left to right or right to left. Defaults to "false" (left to right). Note: If you set "frame" to 1, then you'll want to set "reverse" to *true*.

npc cosmetic switch (who, picture, *direction*, *frame*, *palette*)
-Quick function to display an exact NPC image onscreen, using picture, direction, and frame arguments. Unlike "hero exact frame," this function stores the NPC's original image settings to allow for a simple reset to the previous image when finished using the new image. "Direction" defaults to the current direction.

npc cosmetic reset (who)
-Switches the NPC's image to its default image.

hero animation (who, picture, palette, cut frame, *ticks*, *hold*, *start frame*, *reverse*, *break frame*, *launch script*)
-Shortcut for displaying a range of hero frames in order to simulate character animation. Unlike "hero short frame cycle," this function runs through an entire range of frames (including those belonging to subsequent pictures) until it reaches the final designated frame. "Picture" determines the starting image set. "Palette" should be set to -1 to maintain the picture's default palette. "Cut frame" determines how many frames must be displayed before the animation ends. "Ticks" (defaulting to 2) determines the length of time between frame flips. "Hold" determines whether the final frame remains onscreen after the animation ends or reverts back to the original frame (set to "true" if final image should remain). "Start frame" determines which frame should begin the animation if not equal to the picture set's "north / frame 0" position (1-8). "Reverse," when set to true, will cycle backward from the "cut frame" to the "start frame." "Break frame" determines which frame permits a script interruption before finishing the animation. "Launch script" will interrupt the animation at the "break frame" with a new script, then resume the animation when the script expires. Script name should follow the @ symbol. Note: This function can be used only one character at a time. There is currently no group function for character animation.

npc animation (who, picture, palette, cut frame, *ticks*, *hold*, *start frame*, *reverse*, *break frame*, *launch script*)

Tile Animation Shortcuts:

Shortcuts for simplifying non-default or complex tile animations.

write animation block (x, y, first tile, last tile, layer, wait time, *reverse*)
-Should be obvious. First and last tiles must be defined from low value to high value for normal use. If you want to display animation from high value to low value, set “reverse” to “true.”

General Animation Shortcuts:

Animation shortcuts for non-tile or NPC elements.

backdrop cutscene (start, finish, ticks, *skip*)
-Allows you to define the first and last frames in a backdrop animation series and fills in the rest. "Ticks" determines how long each frame should remain onscreen before cycling to the next one. "Skip" determines how many backdrop IDs to skip past to get to the next preferred displayed frame. Defaults to 1 for consecutive backdrops. Use a negative number for "skip" if you want the animation to run backward.

Tile Reading and Writing Shortcuts:

These shortcuts will make using and manipulating tiles and coordinates a little easier.

delete map block (x, y, *layer*)
-Sets the tile at the designated (x, y) coordinates to the default tile (tile 0). "Layer" is optional and defaults to zero or the bottom layer. Note: Remember that this function will replace the tile with whatever is defined as the first tile in the tileset.

delete pass block (x, y)
-Removes all passability at the designated (x, y) coordinates.

check tile forward (*layer*, *which hero*)
-Returns the tile ID of the position immediately ahead of the designated hero, depending on what direction he's facing. Both arguments default to zero. Must be returned to a reference.

check tile backward (*layer*, *which hero*)
-Returns the tile ID of the position immediately behind the designated hero, depending on what direction he's facing.

check tile left (*layer*, *which hero*)
-Returns the tile ID of the position diagonally counterclockwise to the direction the hero is facing.

check tile right (*layer*, *which hero*)
-Returns the tile ID of the position diagonally clockwise to the direction the hero is facing.

check tile north (*layer*, *which hero*)
-Returns the tile ID of the position directly north of the hero, regardless which direction he's facing.

check tile east (*layer*, *which hero*)
-Returns the tile ID of the position directly east of the hero, regardless which direction he's facing.

check tile south (*layer*, *which hero*)
-Returns the tile ID of the position directly south of the hero, regardless which direction he's facing.

check tile west (*layer*, *which hero*)
-Returns the tile ID of the position directly west of the hero, regardless which direction he's facing.

check tile underfoot (*layer*, *which hero*)
-Returns the tile ID of the position directly beneath the hero's feet.

write tile underfoot (*tile*, *layer*, *who*)
-Writes a new tile ID at the position the hero is standing. Leaving the "tile" argument blank accomplishes the same function as "delete map block." Do this when you want to delete a map tile at the hero's position.

write wall underfoot (*wallbit*, *who*)
-Writes a new pass block value at the hero's position. "Wallbit" requires a value that corresponds with wall map data. The Plotscript Dictionary has a list of acceptable values and practices under the heading "write map block." (0 = delete, 1 = "north wall," 2 = "east wall," 4 = "south wall," 8 = "west wall," 16 = "vehicle A," 32 = "vehicle B," 64 = "harm tile," 128 = "overhead tile"; variable := northwall + eastwall + southwall + westwall; etc.)

full write block (x, y, tile, layer, wall)
-Writes a tile at the selected (x, y) coordinates with passability bits enabled.

check npc tile forward (*npc*, *layer*)
-Returns the tile ID of the position immediately ahead of the designated NPC, depending on what direction he's facing. Note that the NPC ID comes before the layer ID, which is the reverse of the previous "check tile forward." This is because "check tile forward" assumes you're checking the tile in relation to the default hero. With NPCs, there are no assumptions, so you must define that first. Must be returned to a reference.

check npc tile backward (*npc*, *layer*)

check npc tile left (*npc*, *layer*)

check npc tile right (*npc*, *layer*)

check npc tile north (*npc*, *layer*)

check npc tile east (*npc*, *layer*)

check npc tile south (*npc*, *layer*)

check npc tile west (*npc*, *layer*)

check npc tile underfoot (*npc*, *layer*)

write npc tile underfoot (*npc*, *tile*, *layer*)

write npc wall underfoot (*npc*, *wallbit*)

Zone Reading and Writing Shortcuts:

These shortcuts will make using zone actions and coordinates a little easier.

Note: Most of these functions read a zone. Only the last one writes to the zone.

validate zone position (zone, *which hero*)
-Checks whether the hero is standing in the designated zone. It's basically a faster way of writing "read zone (zone, hero x (which hero), hero y (which hero))." Returns "true" or "false."

validate zone underfoot (zone, *which hero*)
-Same as "validate zone position." Just another way to remember it.

validate zone forward (zone, *which hero*)
-Checks whether the designated zone applies to the position ahead of the hero. Returns "true" or "false."

validate zone backward (zone, *which hero*)
-Checks whether the designated zone applies to the position behind the hero. Returns "true" or "false."

validate zone left (zone, *which hero*)
-Remember, "left" and "right" are synonyms for "counterclockwise" and "clockwise." Unless the hero is facing north, this will not necessarily return the east and west positions.

validate zone right (zone, *which hero*)
-Ditto.

validate zone north (zone, *which hero*)
-Checks whether the designated zone applies to the position north of the hero. Returns "true" or "false."

validate zone east (zone, *which hero*)
-Checks whether the designated zone applies to the position east of the hero. Returns "true" or "false."

validate zone south (zone, *which hero*)
-Checks whether the designated zone applies to the position south of the hero. Returns "true" or "false."

validate zone west (zone, *which hero*)
-Checks whether the designated zone applies to the position west of the hero. Returns "true" or "false."

what zone underfoot (*count*, *which hero*)
-Checks and returns the zone ID(s) the hero is standing in, based on the "zone at spot" function. "Count" refers to the position of the zone on the tile. Defaults to zero. Passing "get count" as the argument will return the number of zones the hero is standing on. Using this function in a loop can return more than one result, up to 16 unique IDs. Refer to "zone at spot" in the Plotscript Dictionary for more information on how zone counts work.

what zone forward (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate ahead of the hero, depending on which direction he's facing.

what zone backward (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate behind the hero, depending on which direction he's facing.

what zone left (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate left of the hero, or counterclockwise, depending on which direction he's facing.

what zone right (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate right of the hero, or clockwise, depending on which direction he's facing.

what zone north (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate north of the hero, regardless of which direction he's facing.

what zone east (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate east of the hero, regardless of which direction he's facing.

what zone south (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate south of the hero, regardless of which direction he's facing.

what zone west (*count*, *which hero*)
-Uses "zone at spot" on the (x, y) coordinate west of the hero, regardless of which direction he's facing.

check zone position (zone, x, y, *which hero*)
-Checks whether the hero is standing at the designated (x, y) coordinates in the designated zone ID. Returns "true" or "false." Why would you want to use a function like this when "validate zone position" is more to the point? If you have a map that uses zones to alter or record data depending on a value state (like changing the appearance of trees if the wind is exceptionally strong wherever a particular zone is outlined), then the decision to make that change may depend on which door coordinates the hero emerges from, assuming that the door is also tied to that

zone. In this case, (x, y) would correspond with the door coordinates and "check zone position" would check whether the hero is standing in that door and return the *true/false* condition based on what it finds. Using this function in a loop that cycles through multiple "status zones" is the most effective way to achieve on-the-fly data collection about certain sectors or places defined within zones.

Here's how *Entrepreneur: The Beginning* uses this function to determine the land values of shops that the hero enters or exits:

```
script, scan land values, begin
 variable (f, val, lv)
 val := 1
 for (f, 1000, 1004) do(
  if (check zone position (f, generalfrontdoorx,
generalfrontdoory)) then(
   lv := val
  )
  val += 1
 )
 convert to land value constant (lv)
end

script, convert to land value constant, lv, begin
 switch (lv) do(
  case (1) generalshoplandvalue := land value:poor
  case (2) generalshoplandvalue := land value:lower class
  case (3) generalshoplandvalue := land value:middle class
  case (4) generalshoplandvalue := land value:upper class
  case (5) generalshoplandvalue := land value:rich
 )
end

script, check security need, begin
 variable (val)
 switch (generalshoplandvalue) do(
  case (land value:poor) val := random (7, 10)
  case (land value:lower class) val := random (5, 8)
  case (land value:middle class) val := random (3, 6)
  case (land value:upper class) val := random (1, 4)
  case (land value:rich) val := random (1, 2)
 )
 generalshopsecurityneed := (val * crimeaura) / 100
end
```

validate npc zone position (npc, zone)

validate npc zone underfoot (npc, zone)

validate npc zone forward (npc, zone)

validate npc zone backward (npc, zone)

validate npc zone left (npc, zone)

validate npc zone right (npc, zone)

validate npc zone north (npc, zone)

validate npc zone east (npc, zone)

validate npc zone south (npc, zone)

validate npc zone west (npc, zone)

what npc zone underfoot (npc, *count*)

what npc zone forward (npc, *count*)

what npc zone backward (npc, *count*)

what npc zone left (npc, *count*)

what npc zone right (npc, *count*)

what npc zone north (*npc*, *count*)

what npc zone east (*npc*, *count*)

what npc zone south (*npc*, *count*)

what npc zone west (*npc*, *count*)

check npc zone position (npc, zone, x, y)

check zone (zone, x, y)
-Works like "check zone position" but doesn't depend on the hero's position when he enters the map. In this case, using a loop to search the values of zone, x, and y would allow for greater flexibility in making general changes to the map based on zone status.

Example: To turn desert (zone 1000) to grass (zone 1001) on the fly, I might write:

```
script, search desert and replace, begin
 variable (f, g)
 for (g, 0, map height) do(
```

```
  for (f, 0, map length) do(
   if (check zone (1000, f, g)) then(
    write zone (1001, f, g, true)
   )
  )
 )
 replace desert tiles
end

script, replace desert tiles, begin
 variable (f, g)
 for (g, 0, map height) do(
  for (f, 0, map length) do(
   if (check zone (1001, f, g)) then(
    write map block (f, g, grass tile, layer)
   )
  )
 )
end
```

Note: Writing the script this way ensures that the tiles stay grass when the hero exits and reenters the map, assuming "Tile Data State" is set to "saved when leaving" in the General Map Data settings. Also keep in mind that "check zone" can be adjusted for size and doesn't have to scan the entire map.

Hint: You can also use the command "swap with zone" to accomplish the same function as above. See function(s) below.

scan for zone (zone)
-Scans the entire map for the requested zone. Returns "true" if the zone is found, "false" if it isn't. Using the above desert example, this function can be used to check whether any desert section (1000) still exists on the map.

swap with zone (current zone, new zone)
-Replaces every instance of the "current zone" on the map with the "new zone" ID.

Advanced Positioning Shortcuts:

These scripts are used for advanced positioning and tracking of characters and coordinates.

check forward (x, y, *which hero*)
-Checks whether the tile ahead of the hero (depending on how he's facing) matches the position or coordinates for x and y and returns "true" or "false." Must be used as a conditional.

check backward (x, y, *which hero*)
-Checks the position of the tile behind the hero to see if it matches with the x- and y-coordinates and returns "true" or "false."

check left (x, y, *which hero*)
-Checks the position of the tile left of the hero's facing position (or the next counterclockwise direction) to see if it matches the x- and y-coordinates and returns "true" or "false."

check right (x, y, *which hero*)
-Checks the position of the tile right of the hero's facing position (or clockwise) to see if it matches the x- and y-coordinates and returns "true" or "false."

check position (x, y, *which hero*)
-Checks the position underneath the hero to see if it matches the x- and y-coordinates and returns "true" or "false."

check npc position (npc, x, y)
-Checks the position under the NPC and compares it to the x- and y-coordinates and returns "true" or "false."

who is npc forward (*which hero*, *face*)
-Returns the ID of whichever NPC is standing face-forward from the hero. Stores the information in the global variable "npcneighbor." "Face" refers to the hero's compass (defaults to north or zero) and should not be changed unless the function needs to accomplish an alternate result (like satisfying a "confusion" state, for example).

Note: This function accounts for only one NPC at a time.

Note 2: All "who is" functions are returnable as values, but they also store the value in the global variable "npcneighbor" for delayed use if that action is desired.

who is npc forward to npc (who, *face*)
-Just like above, but checks the position face-forward from the designated NPC.

who is npc backward (*who*, *face*)
-Returns the ID of the NPC standing behind the hero and stores it in the global variable "npcneighbor." "Face" represents compass north and should not be changed without good reason.

who is npc backward to npc (who, *face*)

who is npc left (*who*, *face*)

who is npc left to npc (who, *face*)

who is npc right (*who*, *face*)

who is npc right to npc (who, *face*)

who is npc north (*who*)
-Returns the ID of the NPC standing due north of the hero to the global variable “npcneighbor.”

who is npc east (*who*)

who is npc south (*who*)

who is npc west (*who*)

who is npc north to npc (who)

who is npc east to npc (who)

who is npc south to npc (who)

who is npc west to npc (who)

hero blocks npc (who, *which hero*)
-Checks the tile ahead of the NPC (determined by NPC direction) to see if the hero is standing in its way. Useful for organically determining whether an NPC must be moved to an unobstructed and adjacent tile before marching it across the room past the hero's position.

check blocking character (who, *which hero*)
-Checks to see if a hero or another NPC is standing in front of the designated NPC. “Who” should be the NPC ID or reference, not the hero. “Which hero” can be designated to check for caterpillar party obstruction, but will otherwise default to the main hero. Good for prepping smoother navigation around maps and obstacles.

store current position (*target hero*, *face*)
-Stores the coordinates of a hero and his current direction to a series of global variables referencing positions, directions, and distances. Refer to the script for this function to see which variables are affected.

Note: This function is essential to other position-based functions on this list and is already included in their scripts, so you probably won't need to use it on its own. But it's here in case you want to do something fancy.

store fixed position
-Records the east, west, north, south (x/y) coordinates of the tiles around the hero, based on the values returned to "store current position."

Note: All auxiliary "store position" functions depend on calling "store current position" first to work.

store north position
-Records the coordinates to the first three tiles north of the hero.

Note: All direction positions are saved to the global variables: forwardx, forwardy, forwardx1, forwardx2, forwardy1, and forwardy2. X/Y stores the closest tile to the hero, x1/y1 stores the second closest, and x2/y2 stores the most distant of the three.

store east position

store south position

store west position

store current npc position (target npc, *face*)
-For NPC storage references, check the scripts.

store fixed npc position

store npc north position

store npc east position

store npc south position

store npc west position

set difference to npc (npc, *which hero*)
-Checks how many spaces stand between the hero and NPC.

npc find nearest empty space (who, *destination*, *axis*, *spin*)
-This script walks the NPC to the nearest empty space. "Destination" is a reference tile to determine a starting direction (less than the NPC's position will turn him west or north, more will turn him east or south); "axis" is defined by "axis:xline" or "axis:yline" and determines which axis the "destination" references; "spin" determines if the NPC looks clockwise or counterclockwise for an empty spot. All three are optional. The script will essentially cause the

NPC to search all four directions for an empty tile, rotating clockwise (spin = “right,” default) or counterclockwise (“left”), and then move to that tile when an accessible tile is found. The optional arguments merely determine the first direction. Considers all obstructions when making a final decision.

Team Building Shortcuts:

These scripts are used for converting NPCs to heroes and vice versa. They work best when caterpillar parties are enabled. If your game requires just one hero be on display at a time, then this won't work as well.

hero to npc conversion (who, new id)
-This script takes a member of your caterpillar party and converts him into an NPC. "Who" represents which hero in the caterpillar party, and "new id" determines which NPC the hero will become. Note that the hero should have an NPC version of itself already defined in the editor before running this function, as the function will call that NPC ID from the library and place it where the hero was standing.

npc to hero conversion (old id, who)
-Opposite of "hero to npc conversion."

multiple swap out (*hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Allows you to swap up to four heroes out of the caterpillar party at once. Just fill in arguments for the ones you want out. The rest can stay blank.

multiple swap in (*hero 1*, *hero 2*, *hero 3*, *hero 4*)
-Reverse of "multiple swap out."

special join (npc)
-Walks an NPC to the back of the caterpillar line (it knows which position represents the back) and faces him toward the lead hero. This is best used in conjunction with "npc to hero conversion," which actually turns the NPC into a member of the hero party.

join and convert (npc, which hero)
-Combines the actions of "special join" and "npc to hero conversion."

convert and join (npc, which hero)
-Duplicate of above action (in case you forget the syntax order).

Stat Manipulation Shortcuts:

These scripts are shortcuts for affecting various stats.

Special Note: The two core functions the others are based on (the first two on this list) were written long before "gain hero stat" was introduced to the Plotscript Dictionary. You may find that these functions merely duplicate that one.

maximum stat bonus (who, amount, which stat)
-Just a quick and easy function to increase or decrease a permanent stat bonus with a single line. Note: Buffs should use positive values, debuffs should use negative values. "Which stat" depends on your HSI definitions for a particular stat. Check your HSI file if you're not sure.

current stat bonus (who, amount, which stat)
-Same as above, but for temporary gains.

permanent stat bonus (who, amount, which stat)
-Syntax friendly alternative to "maximum stat bonus."

temporary stat bonus (who, amount, which stat)
-Syntax friendly alternative to "current stat bonus."

permanent stat buff (who, amount, which stat)
-Most accurate naming of permanent stat buffs. Note: This function automatically increases values, hence the "buff."

temporary stat buff (who, amount, which stat)
-Most accurate naming of temporary stat buffs. Note: This function automatically increases values, hence the "buff."

permanent stat debuff (who, amount, which stat)
-Most accurate naming of permanent stat debuffs. Note: This function automatically decreases values, hence the "debuff."

temporary stat debuff (who, amount, which stat)
-Most accurate naming of temporary stat debuffs. Note: This function automatically decreases values, hence the "debuff."

Special Effects Shortcuts:

These scripts are shortcuts for various special effects.

shake screen (*amount*, *width*, *speed*, *exact*, *variance*, *contained*)
-Simulates side-to-side earthquake function. “Amount” determines the number of shakes (defaults to 2). “Width” determines how far from center the screen should shake (defaults to 2). “Speed” determines how fast the camera should move (defaults to random). “Exact” is a true/false declaration that determines whether the shake distance should be exactly the defined width. Note: Defaults to “false,” allowing the width to have a variant distance (positive, negative, or neutral) from the defined value, which adds to the random nature of the shake. “Variance” determines how much extra is added to the width (positive or negative) for each shake (defaults to 2, but is canceled if “exact” is set to “true”). “Contained” is a true/false declaration that determines whether the shake distance happens within the defined width area, or if it spreads double the distance on the way back from its previous source. Note: Defaults to “false,” but if “exact” is also set to false, then the shake could get out of control. Setting “exact” to “false” and “contained” to “true” creates a more jittery effect, while leaving both set to “false” will shake the screen more violently while keeping the hero closer to the center of the action.

rumble screen (*amount*, *height*, *speed*, *exact*, *variance*, *contained*)
-Simulates up-and-down earthquake function.

swirl screen (*amount*, *width*, *height*, *spin 1*, *spin 2*, *speed*, *exact*, *variance*, *contained*)
-Spins the screen around like a tornado. Note: “Spin 1” represents the starter direction and defaults to “right.” “Spin 2” is the follow-up direction and defaults to “up.” For best results, set the first spin to one axis and the second spin to the other.

String Display Shortcuts:

This is an advanced series of string display scripts that puts a positive or negative numeric value (money or experience) over your main character's head and "floats" it upward. This is the type of string function that *Entrepreneur: The Beginning* uses to display money transactions directly on the map.

Note: This series is based on hard-coded functions that have adaptable alternatives. You'll still need to edit the string fields inside of "convenience functions updated 1.txt," or create a cloned version of these scripts, if your game uses symbols different than the dollar sign or "EXP" to communicate money or experience changes, or if you have some other function you want to achieve with floating string effects. It also depends on exact strings. If you need to call a symbol from within the editor, you'll need to swap the string display with an ASCII code call. (Consult the Plotscript Dictionary to learn how to do this.)

Note 2: This script sequence may not respect resolutions outside of the default. If you need something more specialized, then wait for the updated version that I hope to release with my "Scenery Sounds" extra functions pack sometime in the future, which will hopefully also respect ASCII code features.

money change designations (string ID, value, *sound*)
The "string ID" argument should represent the string ID you want to use for the floating symbol that pairs with the money text. "Value" is the amount of money that changes. "Sound" is the sound effect that plays when the string displays. Defaults to -1 (or off). Best used with a coin exchange sound effect (defined in the editor).

Note: This function must be called before running "money change display."

Note 2: The default symbol within the string is "+$" for positive change and "-$" for negative change. If you need a different symbol, you'll need to change it inside the "convenience functions updated 1.txt" script file (under the script "fill in change values").

experience change designations (string ID 1, string ID 2, value, *sound*)
-Works like "money change designations" but represents a display of experience rather than money. "String ID 1" and "string ID 2" designate which string IDs will display "+/-" and "EXP" respectively.

money change display (amount, direction)
-Determines the amount of money the hero receives following an action and displays it onscreen. "Direction" identifies whether the money is added ("up") or subtracted ("down").

experience change display (amount, direction)
-Ditto for experience.

Note: Please remember that these functions only *display* the changes. They don't actually make the changes. You'll still need to write a corresponding function (like "give experience") to make the change.

Conclusion:

Remember that these functions are plug & play, and I haven't tested them all. If anything doesn't work, let me know on the forums and I'll attempt to fix them. But these should hopefully reduce the amount of time it takes to script impressive scenes and moments within your game.

Stay tuned for a "Scenery Sounds" bonus function in the coming weeks. Because this script sequence is far more complicated than everything else on this list, I've left it out of the official Volume 1, but I do think it's convenient enough to add as a separate item. Keep a lookout for it.

Thanks for using the function pack. Let me know what you'd like to see added for Volume 2.