ARES Slideshow
user manual V1.3

ARES Video offers the possibility to use self-created "intelligent" slideshows for laser shooting training. In these not only different images can be displayed, but also the transition between these images can be defined in many ways and even made interactive. That means the slideshows can react to the shots. The shows are written in a simple scripting language, which does not require any programming knowledge, but also offers the possibility for complex sequences. To facilitate the creation of the slideshows, there is also an editor that allows, among other things, to play the slideshows step by step. In addition, it offers color highlighting of keywords and displays error messages during execution if necessary.

BASIC STRUCTURE AND FUNCTIONALITY OF THE SLIDESHOWS

The slideshow files have the extension ".dia". They are simple text files that can be opened with any text editor. When played, the file is processed line by line. Empty lines are allowed and are simply skipped. It is also possible to write comments. These begin with a hash symbol (#) which ensures that all text up to the end of the respective line is ignored. Each line contains exactly one command, which can be of any length. The end of a line therefore also marks the end of a command. An explicit character, as it is usual in some other programming languages, is not necessary. The slide show stops automatically when the last line has been processed. It can also be stopped at any time by pressing the escape key.

BASIC ELEMENTS OF THE SCRIPTS: COMMANDS, VARIABLES, UND LOGIC

The capabilities of the slideshow scripts can be roughly divided into three areas. There are commands, variables and logic.

COMMANDS determine the actual appearance of the slideshow. They can be used to display images, output text, play sounds or define waiting times.

VARIABLES are used to store values and to calculate with them or to make decisions depending on the values. You can define and use as many variables as you like. The system itself also provides a lot of information about variables. For example the hit position of the last shot, the size of different elements or the time elapsed since the beginning.

LOGIC is the part that determines the flow of the slideshows and thus provides the "intelligence". There are branches, loops, jumps and the logical links AND and OR. With this it is possible to run through different parts of the slideshow depending on variable values.

OVERVIEW OF COMMANDS

All commands should be placed at the beginning of a line, because they cause an action, but cannot be processed by other commands. The only exception is the RAND command, which can be used anywhere where a variable or a number could be placed.

Command	Function
`OPEN "%File%"`	Opens an image file and displays it on the screen. The file must be in the same folder as the slideshow. Only the following formats are supported (jpg, png, bmp). The file name is enclosed in quotation marks. The filename can also be generated from a combination of text and variables. Example: `OPEN "Image" + $Var + ".jpg"`

	If the variable $Var has the value 1 the image file "Image1.jpg" is opened, if it has the value 2 the file "Image2.jpg" is opened.
OPEN_SCORE "%File%"	Similar to OPEN but the image file is not displayed but opened in the background as a "dot template". Ideally, the template should have the same size as the displayed image. After a shot, the color of the dot template is available at the hit spot via 3 variables. Thus, any hit zones can be defined, but they do not have to be visible to the shooter.
OPEN_DECAL "%File%"	Selects the file for the decal object. This is an image that can be displayed over the slideshow, with the position and size freely adjustable via variables. Movements can also be realized by continuously changing the position.
PRINT "%Text%"	Sets the text of the text object to the specified text. The text object can be used, for example, to display messages, but also scores and times. As with the OPEN commands, the text can be composed of a combination of text pieces and values. Example: **PRINT "Time: " + $RUNTIME +" seconds"** Sets the text to "Time: 31.415 seconds" where the number is the current value of the $RUNTIME variable, which contains the runtime of the slideshow so far.
SOUND "%File.wav%"	Plays a sound file in the wav-format. This command should not be used too often in succession, but wait until the previous sound has been played before playing the next one.
WAIT %Time%	Waits for the specified time in seconds before executing the next line. Example: **WAIT 1,5**
RAND %Number%	Generates a random number between 0 and the entered number minus 1. Only integers without comma part are generated. If you want decimal numbers, you have to divide the result accordingly. Example: **RAND 5** Returns a random number that can be 0,1,2,3 or 4. **(RAND 100)/10** Returns a random number between 0,0 and 9,9.

VARIABLES AND CALCULATIONS

A variable has a name and a value can be assigned to it. The name of the variable must start with a dollar sign ($) and may then consist of any number of upper and lower case letters or digits. Umlauts are not allowed and the only special character allowed is the underscore (_). Each variable is created automatically as soon as its name appears for the first time. If it is not already assigned a value, the value 0 is assumed. The value is a fixed point number with 3 decimal places. The smallest value greater than 0 is 0.001. The largest possible value is 2,000,000 (2 million). If a variable is assigned a larger value, it is automatically limited to this 2 million. Negative values are also possible, so that the entire value range goes from -2,000,000 to 2,000,000. For values, both the period and the comma are allowed as decimal separators.

A variable retains its value until the end of the slideshow, unless a new value is explicitly assigned to it. Assigning a value is done using a simple equal sign (=), where the value to the right of the equal sign is assigned to the variable to the left of it. The value to the right can also be a variable itself. Chain assignments also work.

```
$varA = 5            #Variable $varA now has the value 5
$varB = $varA        #Variable $varB now also has the value 5
$varA = $varB = 7    #Variable $varA and $varB both have the value 7
```

The four basic arithmetic operations can be performed with variables and fixed values. The results of these calculations can be used as input for commands or they can be assigned to variables in order to store them. For these calculations, the dot-before-dash rule is observed, whereby round brackets can also be used to influence the order of processing.

```
$varA = 2 + 1,5                 #Variable $varA is 3,5
$varB = $varA * 2               #Variable $varB is 7
$varC = $varB/3                 #Variable $varC is 2,333
$varD = ($varA - $varC)*2       #Variable $varD is 3,334
```

It often happens that you want to calculate the value of a variable with another value and assign the result to the same variable again. For this purpose, there is a shorthand notation in which the symbol for the arithmetic operation is followed by an equal sign.

```
$varA = $varA + 5   #Variable $varA is increased by 5
$varA += 5          #Variable $varA is also increased by 5
$varB -= $varA      #$varB is decreased by the value of $varA
```

LOGIC

Logic commands can be used to influence the flow of the slideshow by linking the execution of commands to conditions.

CONDITIONS

sind Ausdrücke bei denen Variablen oder Werte miteinander verglichen werden. Das Ergebnis dieses Vergleichs kann wahr oder falsch sein. Folgende Vergleiche sind möglich:

`$varA == $varB`	equal
`$varA != $varB`	not equal
`$varA < $varB`	lesser
`$varA > $varB`	bigger
`$varA <= $varB`	lesser or equal
`$varA >= $varB`	bigger or equal

Note that a double equal sign must be used to compare for equality. The single equal sign is used for the assignment.

Values or variables can also be a condition in themselves. If only one variable is specified at the position where a condition should be, the condition is evaluated as not fulfilled if this variable is exactly 0, otherwise it is considered as fulfilled.

Several individual comparisons can be linked with the keywords AND and OR, whereby round brackets can also be used to determine the order of processing (from inside to outside).

```
IF $varA < 5 OR $varA == 8
#...
ENDIF

IF ($varA >= 10 AND $varB != 7) OR $varC == $varD
#...
ENDIF
```

IF [CONDITION]...ELSE...ENDIF

With the **IF** construction parts of the script can be executed depending on whether a certain condition is met. The optional **ELSE** part can then be used to execute another part if the condition is not met. For better readability, the parts in the individual branches are indented in the example. This is permissible and recommended, but not necessary.

```
#Example IF without ELSE branch
IF $varA < 5
  PRINT "A ist lesser 5"      #happens only if A lesser 5
ENDIF

#Example IF with ELSE branch
IF $varB == 3
  PRINT "B is 3"
ELSE
  PRINT "B is not 3"
ENDIF
```

WHILE [CONDITION]...ENDWHILE

A While loop is used to run through a certain section again and again as long as a condition is met. When the program flow reaches the line with the **WHILE**, it is checked whether the condition is fulfilled. If this is the case, the following program section is executed, otherwise the program flow jumps to the line after the corresponding **ENDWHILE** command. When the program reaches the line with the **ENDWHILE** after executing the commands, it checks whether the condition is still fulfilled. If it is, the program jumps back up and repeats the entire part. Whenever the slideshow makes such a jump back to the beginning of a loop, it automatically makes a pause of 0.01 seconds. This makes it perfectly safe to create infinite loops, i.e. loops whose condition is always met. In many other "programming languages" such infinite loops would lead to serious errors.

```
#Countdown
$Countdown = 3
WHILE $Countdown > 0      #repeat until 0 is reached
  PRINT $Countdown        #output current number
  $Countdown -= 1         #decrease number by 1
  WAIT 1                  #wait 1 second
ENDWHILE
PRINT ""                  #clear text
```

BRAKE

Brake is not a construct of its own, but a command that allows a **WHILE** loop to exit even if the condition after its **WHILE** command is still true. When the program flow encounters a **BREAK** within a **WHILE** loop, it immediately jumps to the line after the corresponding **ENDWHILE**. Thus, loops with multiple termination conditions can be created. For example, you can create an infinite loop and still exit it. Depending on the preferred writing style, this can lead to short and clear programs.

```
WHILE 1                 #endless loop: "1" is always true
  IF $RUNTIME > 10
     BREAK              #exit loop after 10 seconds
  ENDIF
ENDWHILE
```

Loops can also be nested, whereby a **BREAK** command always leaves only the innermost of the currently active loops.

```
#Nested loops. A DECAL image is loaded and this
#then moved line by line from left to right. At the end of each line
#it jumps down a bit
OPEN_DECAL "Target.png"
$DECAL_X = $DECAL_Y = 0
WHILE $DECAL_Y < 100
     WHILE $DECAL_X < 100
          $DECAL_X += 10
          WAIT 0.02
     ENDWHILE
     $DECAL_X = 0
     $DECAL_Y += 10
ENDWHILE
```

GOTO :MARKER

With a GOTO command, the program flow jumps directly to the line with the specified marker. This marker must exist and it must be unique. A marker must stand alone in a line. The same naming rules apply to it as to variables, but it does not begin with a dollar sign but with a colon.

```
GOTO :END       #jump to Marker ":END"
$varA = 7       #this line is skipped
:END
```

There are no limits to the position of the target marker. A GOTO command can be used to jump out of loops or into them at any point. It is recommended to use the GOTO command only very sparingly, since it bears the danger of making the program flow very confusing.

SYSTEM VARIABLES, DECALS AND TEXT

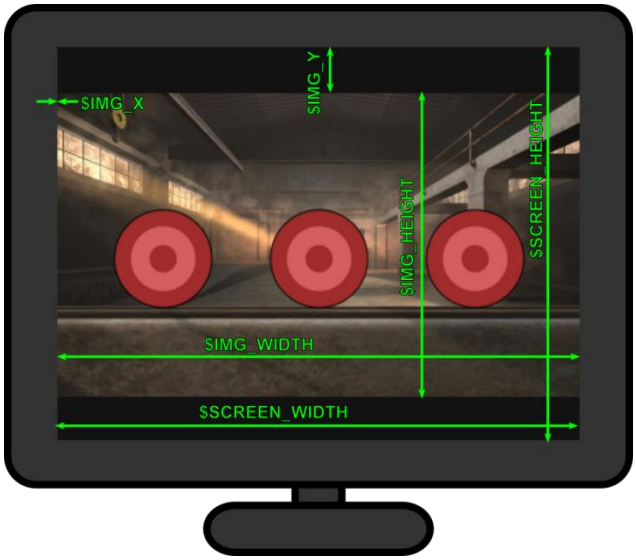
System variables are variables that the system uses to provide information or to manipulate objects. They can be used like normal variables, however, they do not necessarily keep their value permanently, but are always filled with the current values without the script's intervention. They can be roughly divided into 5 categories. Each of these categories can be shown or hidden in the slideshow editor to keep the list of displayed variables clear.

GENERAL SYSTEM VARIABLES contain information about the screen size and time

variable	description
$RUNTIME	Time that has elapsed since the start of the slideshow. Can also be changed if necessary after which the time is counted from the new value
$SCREEN_WIDTH	Width of the display area in pixels (screen width).
$SCREEN_HEIGHT	Height of the display area in pixels (screen height)
$IMG_WIDTH	Width of the displayed image in pixels
$IMG_HEIGHT	Height of the displayed image in pixels

$IMG_X	X-position of the displayed image in pixels related to the left edge of the screen
$IMG_Y	Y-position of the displayed image in pixels related to the top edge of the screen

When an image file is opened, it is automatically scaled so that it fills the display area as much as possible, but its aspect ratio is not distorted. So if the image is exactly the same size as the screen or at least has the same aspect ratio, it will be displayed full screen. The image height and width are then equal to the screen height and width, respectively, and the image positions are 0. If the aspect ratio does not fit, there will be black bars either on the left and right or on the top and bottom. The two position values then indicate the position of the top left corner of the image relative to the top left corner of the screen. The variables starting with "$IMG_" are automatically set to the values as shown in the following picture with every OPEN command.



These variables are especially useful if you want to write a slideshow so that it works at any screen resolution. In this case, objects should not be positioned according to absolute values, but according to the screen resolution. So an object whose X position is half of **$SCREEN_WIDTH** and whose Y position is half of **$SCREEN_HEIGHT** is always centered in the middle of the screen.

INPUT VARIABLES contain the current state of keyboard keys. The variables have the value 1 if the key is currently pressed and the value 0 if it is not. Thus it is possible to influence the slide show also by keyboard inputs. Available are the letter keys (without special characters), the number keys 0 to 9, the F keys F1 to F12 and the Enter key. The Escape and Space keys are already assigned special functions by the program. As with playing videos, a slide show can be paused at any time with the space bar and ended with the escape key.

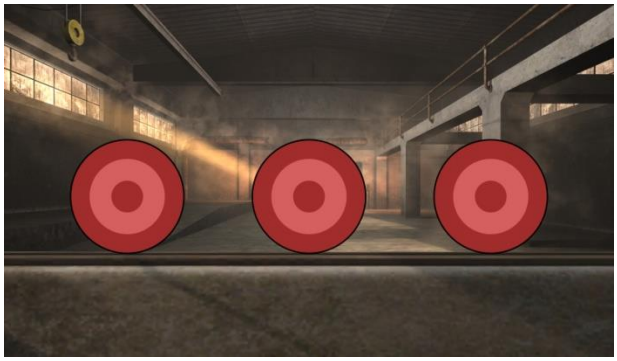
variable	description
$KEY_0 … **$KEY_9**	state of the number key (1 = pressed) / (0 = not pressed)
$KEY_A … **$KEY_Z**	state of the letter key (1 = pressed) / (0 = not pressed)
$KEY_F1 … **$KEY_F12**	state of the F- key (1 = pressed) / (0 = not pressed)
$KEY_ENTER	state of the enter key (1 = pressed) / (0 = not pressed)

HIT VARIABLES are updated with each new shot and contain information about it. Both the number of shots fired and the position of the last shot are counted. In addition, the color information of the hit pixel is read out, both in the normal, displayed image and in the score image. The latter thus offers a possibility to evaluate hit zones.

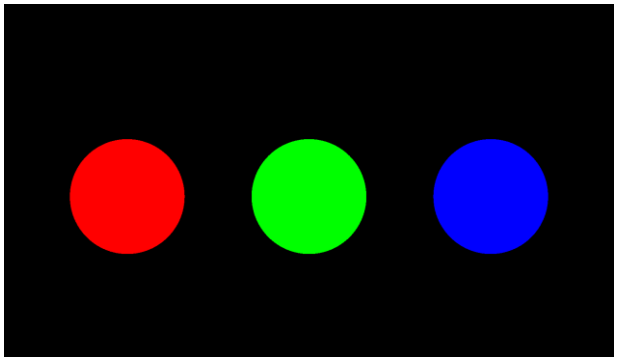
variable	description
$SHOT_COUNT	Number of shots fired so far. Can be changed if necessary, after which the time is counted from the new value.

$HIT_TIME	Width of the display area in pixels (screen width).
$HIT_X	X-position of the hit in pixels related to the left edge of the screen
$HIT_Y	Y-position of the hit in pixels related to the upper screen edge
$HIT_RED **$HIT_GREEN** **$HIT_BLUE**	Color values of the pixel hit in the image previously opened by "OPEN" command. The 3 variables stand for the 3 color channels red, green and blue. If the image has not been hit or no shot has been fired yet, all 3 values are 0.
$HIT_SCORE_RED **$HIT_SCORE_GREEN** **$HIT_SCORE_BLUE**	Color values of the pixel hit in the image previously opened by "OPEN_SCORE" command. The 3 variables stand for the 3 color channels red, green and blue. If the image has not been hit or no shot has been fired yet, all 3 values are 0.

The position in the score image is calculated from that in the displayed image. So if the hit in the displayed image is e.g. one third of the image width from the left edge and one quarter from the top edge, then this also applies to the evaluated position in the score image. This means that the displayed image and the score image do not have to be the same size or have the same aspect ratio. However, it is recommended to take images of the same size anyway, as this makes things easier. The pictures below show how a score image can be used to evaluate a hit. Using the three colored circles, you can not only tell if a target was hit, but even which one.



displayed image



score image

If you want to evaluate hit zones by colored areas, you should take care not to have unwanted antialiasing effects when creating the images. Antialiasing is a technique that many paint programs use to make edges look smoother. Instead of drawing a hard transition, a smooth color transition is created at the edge. This is of course annoying if you want to make a clear decision based on the color whether a hit is inside or outside the area. Therefore, you should be aware of the effect and find out how to disable it in the painting program you are using or use an appropriate program.

without antialiasing

with antialiasing

THE DECAL-OBJECT

It is possible to display a single image file freely positionable and scalable in front of the background. This is done with the so-called decal object (decal = sticker/decal). As long as no image file is loaded, the decal is not visible. The loading of the image file is done with the command "OPEN_DECAL" and works according to the same rules as the

other two Open commands. The image must be in the same folder as the slideshow and may have the formats *.png, *.jpg or *.bmp. PNG images also take the transparency channel into account, so they are the way to go if you want to display non-rectangular decals. The Decal object is configured with a set of variables. The $DECAL_X and $DECAL_Y variables determine the position relative to the upper left corner of the screen, and the $DECAL_WIDTH and $DECAL_HEIGHT variables determine the size. However, the latter two are also changed by the system. After each **OPEN_DECAL** command, they contain the height and width of the loaded image file, which is therefore displayed undistorted in its original size. However, if these variables are changed by the script after opening an image file, the decal image will be scaled to the corresponding size. This can also change the aspect ratio, which distorts the image. The values can of course also be calculated based on the screen resolution, so that the decal always looks the same regardless of this. When positioning the decal, the upper left corner of the decal is always the reference point which is positioned by setting **$DECAL_X** and **$DECAL_Y**. To make the decal object invisible again, you can simply call the **OPEN_DECAL** command with a non-existent (or empty) file name.

variable	description
$DECAL_X	X-position of the DECAL object in pixels related to the left screen edge.
$DECAL_Y	Y-position of the DECAL object in pixels related to the upper screen edge.
$DECAL_WIDTH **$DECAL_HEIGHT**	Width/height of the decal. Is set to the width of the image file at the OPEN_DECAL command and can be changed afterwards to scale the decal. If negative values are used here, the decal is mirrored around the corresponding axis.

THE TEXT OBJECT

In addition to the decal, there is also the possibility to create a text object and position it freely. The text is specified by the PRINT command. As with the other commands, the PRINT keyword can simply be followed by a text in quotation marks or by a combination of several pieces of text and variables that are joined together via plus signs. The values of the variables are inserted into the text. The following rules apply:

- If a value has no decimal places, only the pure value before the decimal point is output (e.g. "3").
- If a value has decimal places, it is always output with all 3 decimal places (e.g. "12.300").
- The separator is a dot
- If a value is negative, it is preceded by a minus sign. Positive values have no sign.

It is also permissible to write the PRINT command exclusively with a variable or a fixed value without linking it to a piece of text.

To make the text completely invisible, simply execute the PRINT command with an empty text.

Changing the text or the font size causes the size of the text object to change. Therefore, the corresponding variables are updated in the process.

The font is Arial and cannot be changed, but the text color can be set using 3 variables.

To create multi-line texts, a line break can be created with the character combination "\n". Nevertheless, the whole PRINT command must be in one line of the script.

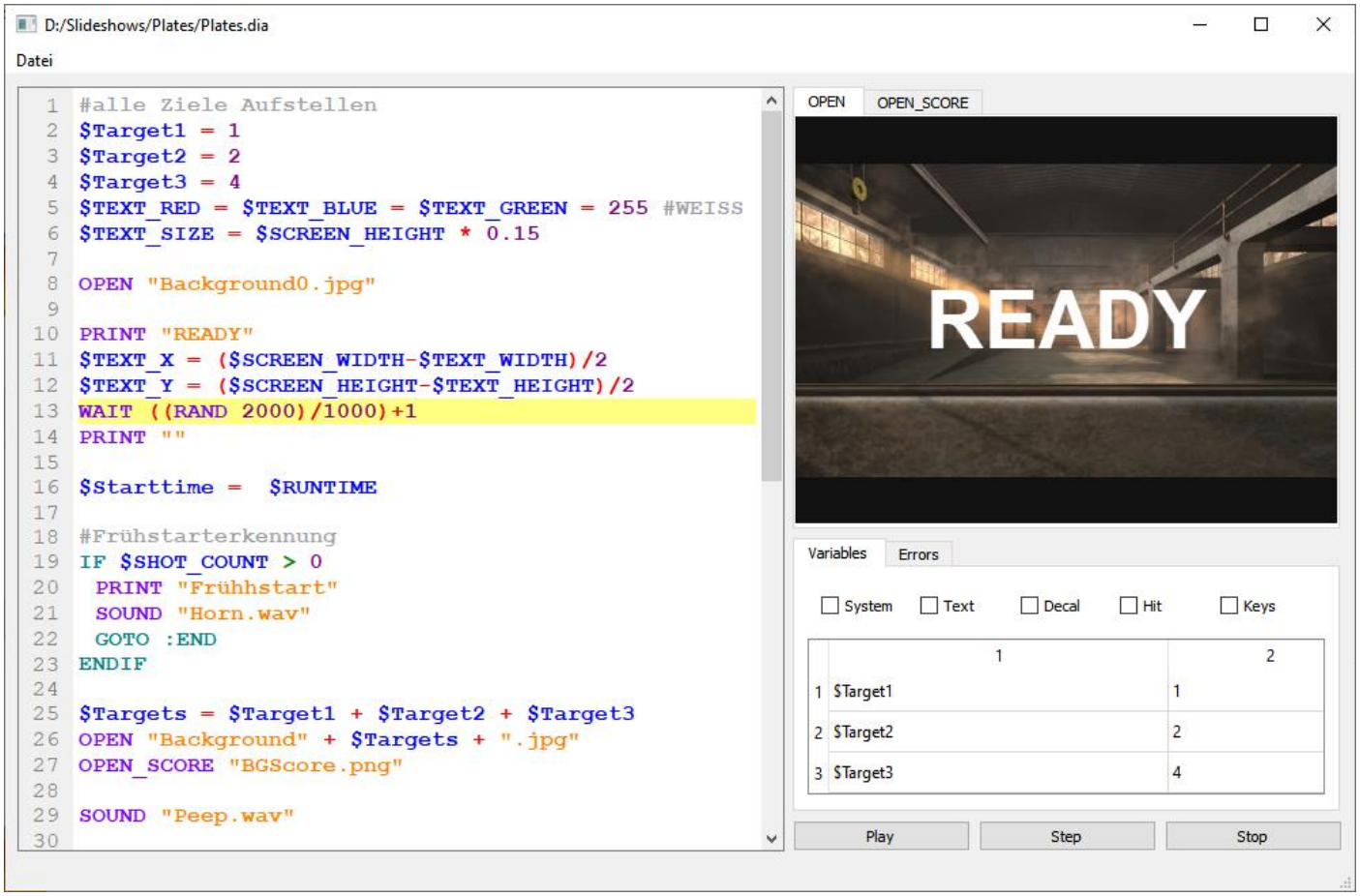
variable	description
$SHOT_COUNT	Time that has elapsed since the start of the slideshow. Can be changed if needed after which the time is counted from the new value.
$TEXT_X	X-position of the text object in pixels related to the left screen edge.
$TEXT_Y	Y-position of the text object in pixels relative to the top edge of the screen.
$TEXT_SIZE	Font size of the text object. The height of the font is approximately the value of this variable in pixels.

$TEXT_WIDTH **$TEXT_HEIGHT**	Width and height of the text object. They are automatically updated when the text or font size is changed. The values can be used, for example, to center the text object at a specific location.
$TEXT_RED **$TEXT_GREEN** **$TEXT_BLUE**	Determine together the color of the text object. Only values between 0 and 255 should be used for each color channel.

```
$TEXT_RED = $TEXT_BLUE = $TEXT_GREEN = 255   #Text colour whithe
$TEXT_SIZE = $SCREEN_HEIGHT*0.06             #Text size
PRINT $RUNTIME + " Sek.\n" + $SHOT_COUNT + " Schuss"
$TEXT_X = ($SCREEN_WIDTH-$TEXT_WIDTH)/2      #center vertical
$TEXT_Y = $SCREEN_HEIGHT*0.75                #horizontal 75%

#The above code writes a white text that represents in two
#lines representing the time and the shots fired
#
# 12.865 sec.
# 14 shot
```

Arbeiten mit dem Editor



The editor helps to write the slideshow scripts. Via the file menu .dia files can be loaded and saved. But you can also simply drag and drop them into the editor window. With the button "Play/Pause" the slideshow can be played in the preview window. By clicking into the preview window, shots are simulated. In addition to the background image visible to the player, which is loaded with an OPEN command, the point template (OPEN-SCORE) can also be displayed. It is also possible to run the slideshow line by line. In this case, a yellow marker indicates the next line to

be executed. During execution, all the variables used and their current values are displayed in the variable window, whereby the system variables of the various categories can be shown or hidden.

If an error occurs during execution, the slide show stops and the corresponding line is highlighted. In the "Errors" tab, a short description is also displayed to help find the cause of the problem.

error message	description
Unknown Token(s):"xyz"	The part "xyz" in the line cannot be assigned to a variable, a command or any other keyword.
Number Of Brackets	The number of opening and closing parentheses in the line does not match.
Wrong Paramters Types: "xyz"	In the "xyz" part, an attempt is made to perform an operation with components that do not match. For example, to assign a text to a variable ($VAR = "text")
OPEN - File Not Found: "xyz" **OPEN_SCORE - File Not Found: "xyz"**	The file "xyz" which should be loaded with an OPEN command or OPEN_SCORE command was not found. With this error the slideshow does not stop, but loads a default image instead.