

HEROES V

OF MIGHT AND MAGIC



The Basics of Heroes V Scripting

Version 1.0 © Celestial Heavens, 2007.

This Scripting Guide was written for Celestial Heavens by



Pitsu

Proofreading and constructive criticism by
Grumpy Old Wizard, Gaidal Cain and Robenhagen

Layout
Robenhagen

If you have feedback, questions or comments, please contact us at:
staff@celestialheavens.com

or post at the relevant topic on The Heroes Round Table:
<http://www.celestialheavens.com/forums/viewforum.php?f=8>

Foreword

Scripting your maps is a way to make them more interesting, personal and control the game course better. The Heroes of Might and Magic V Editor (game patch 1.3 or later required) allows you to build such scripts. In your game folder you even have documentations for editor handling and the script commands. Yet, it is easy to get lost when opening the script editor window and seeing the blank white page. What and how do write in there?

This little pdf is meant as a supplementary material for the official editor documentations. It does not describe all the individual commands for instance, since they already are quite well covered in official materials. It is just about the trivial of SCRIPT

writing: the syntax, how to build a function and set triggers properly. But even with this guide and other materials at hand, be ready to lose some nerves when doing you first scripts? Luckily on online forums other people can help you in your darkest hours. I must admit that without people asking for help and without other mapmakers demonstrating their neat solutions, this piece of manual would not have been written. My thanks to all the regulars of the Round Table Mapmaking Guild!

Oh, and an important warning before you start to work with scripts: be aware that Heroes V scripts works only on single player maps!

*Wishing you good luck with scripts,
Pitsu*

Table of Contents

1. Script file layout	2
2. Functions	3
2.1. Defining functions	3
2.2. If ... then, while ... do etc. blocks	3
2.3. How many “end”s does there have to be?	3
2.4. Function fname() or function fname(parameter)	3
2.4.1. Function fname(hero_name)	3
2.5. Threads	4
3. Variables	5
3.1. Variable types	5
3.1.1. Global variables	5
3.1.2. Local variables	5
3.1.3. Game variables	5
4. Triggers	6
4.1. Instant triggering of functions	6
4.2. Binding functions to map events and objects	6
4.3. Triggering fname(parameter) type of functions	6
4.4. Removing a trigger	7
5. Syntax	8
5.1. Are empty lines, semicolons and number of spaces important?	8
5.2. Case sensitivity	8
5.3. Quotation marks and their use	8
5.3.1. USE NO QUOTES WHEN	8
5.3.2. USE QUOTES WHEN	8
5.4. How to refer text message files, map objects and heroes	9
5.5. Meaning of some symbols	10
6. Hints for Debugging	11
6.1. Enabling console	11
6.2. Using the print command	11
6.3. My script does not run!	11

Script File Layout

1. Script File Layout

When you create a new map and open its script file (Script tab under Map Properties) you'll see only a blank white sheet. When you open a heavily scripted map script file you'll see quite a mess of text. Is there any particular layout to follow? The answer is no, no particular layout has to be followed.

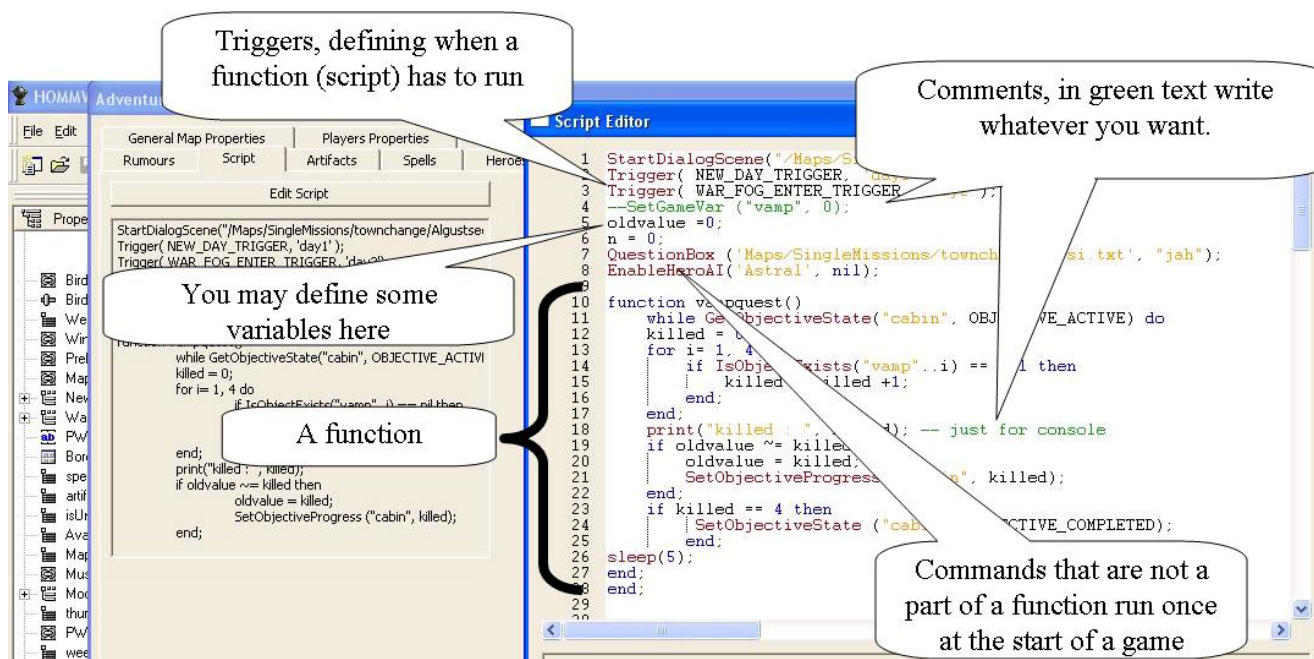
Thus, depending on the style of the script writer, script file can have the following things in chaotic or well-organized mess:

- **functions** – groups of commands which will be the main part of the script file. Each script that you want your map to have, gives you one or more functions. Simpler scripts need one function, more complicated may need many. Often it is

advised to split one script into several functions just for better overview of the different parts of the script.

- **triggers** – defines when a function(script or part of script) is triggered
- **commands** – commands that are NOT a part of any function are triggered when the map is started and the script file is read the first time. That is the early morning of day 1 week 1 month 1.
- **comments** – comments are ignored by the game and are meant for script writer only. In case you are afraid of forgetting the meaning of the script, add a comment to it. Comments start with double minus (double hyphen) -- and end with a line break. Adding double minus in front of a command is a simple way to test how your script runs without that line. Sometimes this is useful for hunting bugs.

Example



Note the text color code. It helps a bit to avoid spelling and other mistakes. When it is **green** (comment), you can write anything, it is ignored by the game. When it is **blue**, it is a number or has a certain meaning for lua programming language. When it is **red**, it is a properly spelled Heroes V specific command. Text between quotation marks (parameter names) is **orange**. The rest is **black**.

2. Functions

2.1. Defining functions

Functions are the main part of a script file. They consist of commands that you can find in one of the official pdfs. Simpler functions look like:

```
function give_it_a_nice_name()
    commands_of_your_choice ;
end;
```

More complicated functions can have different blocks. For instance:

```
function give_it_a_nice_name()
    commands_that_run_every_time_when_the_
    function_is_triggered ;
    if boolean_test then
        commands_that_run_only_if_boolean_test_
        returns_true ;
    end;
    again_commands_that_run_every_time_when_the_
    function_is_triggered ;
end;
```

You can have blocks within other blocks:

```
function function_name()
    commands_that_run_always_when_the_
    function_is_triggered ;
    if boolean_test then
        commands_that_run_only_if_boolean_test_
        returns_true ;
        if boolean_test2 then
            commands_that_run_only_if_boolean_test_and_
            boolean_test2_BOTH_return_true ;
        end;
    end;
end;
```

NOTE: Clicking on the script editor <Check> button gives you an ERROR “function xxx not defined” for every function. This is meant only to confuse and scare you ... Ignore it.

2.2. If ... then, while ... do etc. blocks

The basic most common types of blocks are briefly described below. But in fact lua, the language that Heroes 5 uses, is more powerful than that, but in order to learn other possibilities of the same commands, head over to any lua manual (<http://www.lua.org/pil/>).

```
if boolean_test then
    commands;
end;
```

```
if boolean test then
    commands;
else
    commands;
end;

while boolean test do
    commands;
end;

for variable name = start_value, end_value do
    commands;
end;

repeat
    commands;
until boolean test
```

2.3. How many “end”s does there have to be?

An “end” is required for each “block” that is started with:

- function
- if ... then
- if ... then ... else
- while ... do
- for ... do

Everything that is between an above listed command and respective “end” statements forms a block and are run together or none is run. See examples under general description of functions.

One block that does not require “end” is:

```
repeat
    commands;
until boolean test;
```

2.4. Function fname() or function fname(parameter)

Imagine situation where you have several objects which trigger similar scripts. For example several towns that upon capture are converted to another town type. You could write a separate script for each town, but you can use the possibility to write a general function and specify the town that has to be changed in the trigger. For that is the possibility to define variables in parenthesis after function name. You can have more than one such variable there, separated by commas, if needed.

Example

The general function looks like:

```
function transform(townname)
    TransformTown(townname, TOWN_ACADEMY);
end;
```


When calling it up, you can specify the variable “townname”. Lets say the town names are town1 and town2. When you trigger this function by

```
transform("town1");
```

then variable townname is given value “town1” and the original function is understood as:

```
function transform()
  TransformTown("town1", TOWN_ACADEMY);
end;
```

```
transform("town2"); -- would result the very same
                      original function read as:
function transform()
  TransformTown("town2", TOWN_ACADEMY);
end;
```

In the following example “townname” variable is not used for anything. The variable is defined, but always town2 is converted. Thus “function transform()” is more appropriate to use than “function transform(townname)”.

Bad example

```
function transform(townname)
  TransformTown("town2", TOWN_ACADEMY);
end;
```

2.4.1. Function fname(hero_name)

Heroes have a special relationship with functionname(parameter) type of functions. Namely, one does not need to specify the hero to which the generic function applies. If not specified, the currently selected hero, who triggered the script is chosen.

Example

```
function message2(1011u) -- without defining
  the parameter here, HasArtefact returns an error
  if HasArtefact(1011u, 2) then -- tests whether
    selected hero has artifact number 2
    SetRegionBlocked("passage", 0, 3);
  end;
end;
```

This function can be called up without specifying the hero name like :

```
message2();
```

or

```
Trigger(REGION_ENTER_WITHOUT_STOP_TRIGGER,
  "areal", "message2");
```

See also “Triggering fname(parameter) type of functions” chapter.

2.5. Threads

Threads are similar to continuous events in Heroes IV. They can kick in at any time and they happily use up your CPU resources. A thread, once triggered, can run forever and in parallel to other functions. For becoming a successful thread a function must have the following structure:

```
function funcname()
  while boolean test do
    commands;
    sleep(10);
  end;
end;
```

As long as the boolean test returns true, commands are run. Most often you do need the opposite: commands are hold back until a certain requirement is fulfilled. For that use the following:

```
function funcname()
  while boolean test do
    if boolean test2 then
      commands;
    end;
    sleep(10);
  end;
end;
```

If you have problems defining the first boolean test which determines how long the thread lives, give it eternal life with “while 1 do”. Official maps use this statement quite often.

sleep(10) – why that? This is to save your CPU. If you forget it or misplace it so that it is not effective, you force the thread to run without pauses, causing game lags. The higher the number the more time you give for other processes. 10 is a good value, but anything between 5 and 20 is quite normal in my opinion. You can experiment with other values at your wish.

Triggering threads goes via *startThread(functionname)* command. Triggering them in other ways basically blocks access to other scripts as long as the (infinite) thread runs.

Example script: You want a script that gives the player an instant win when he accumulates 100.000 gold.

```
function checkgold()
  while 1 do
    if GetPlayerResources(PLAYER_1, 6) >= 100000
    then
      win();
    end;
    sleep(10);
  end;
end;

startThread(checkgold);
```


3. Variables

A variable can have different values while the script is run. Do not confuse variable names with function names and names of adventure map objects, scenario objectives, file names etc. They are used and handled differently. When in doubt whether a name refers to a variable or not, check if there is a function, adventure map object or scenario objective with this name. If not, it is most likely a variable.

3.1. Variable types

There are 3 types of variables based on their accessibility. Local are used inside one function only, global in an entire script file and game variables for transferring variables from one script file to another (in cases when a map has more than one script file). You can transfer the values from one type to another. For example:

```
local m = GetGameVar("temp.gamevar1");
```

3.1.1. Global variables

These are the most common ones. Usually you can live with only them, without using any other type of variables. It is not a must to define them prior use. The first time you use a name and give it a value defines the variable and it can be used at any place in the script file. For keeping better track of them and avoid potential errors where using a variable before giving it a value you may still define them at the start of a script file like shown on the image on page 4.

Example

```
heronm = "Aberrar"; -- with this line you define
a text string type variable called
heronm, with current value Aberrar.
```

3.1.2. Local variables

Local variables are meant to work only within the function where they are defined. There are specific cases when this may be desired, but usually one does fine with global variables only.

Example

```
function day2()
local day =2; -- defines a local variable with
               value 2
if GetDate(DAY) == day then -- if statement re-
                             turns true on second day.

    blabla();
end;
end;

function day3() -- another function in the same
                 script file
if GetDate(DAY) == day then -- error, since
                             variable "day" does not exist in
                             this function.

    blabla();
end;
end;
```

3.1.3. Game variables

Game variables are required for transporting variables between script files. For example you have a separate combat script file beside the general map scripts and you want to use the main fail variables in combat script. Then you have to define and use game variables with the following commands:

```
SetGameVar("variablename", value) and
GetGameVar("variablename")
```

4. Triggers

Another very important part of a script is to set the triggers so that each function is run at the moment when you wish it. You can call a function up instantly by using its name or you can define the conditions when a function has to run by the *Trigger* command.

4.1. Instant triggering of functions

First, a function can be called up within other function or as a map starting event. That goes simply by typing the function name.

Example

```
function blabla()
    MessageBox("Maps/SingleMissions/test/blah.txt");
end;

function day2()
    if GetDate(DAY) == 2 then
        blabla(); -- here function blabla is triggered
                    and the messagebox is shown at day 2.
    end;
end;
```

4.2. Binding functions to map events and objects

How do define that a function has to run when a certain event happens on your map? That is what you have the *Trigger* command for. It is quite nicely described in the official editor documentation pdfs, consult it for more information. Such triggers look like:

```
Trigger(EVENT_TYPE, "object to which the event
is bound", "function name that has to be
triggered");
```

When the *Trigger* command is not a part of a function, the function specified among its parameters is bound to the specified event at the start of the map. Being a part of another function means that the link between specified function and event is created when the original function runs.

Note that only one function can be bound to each object and event combination. For example you cannot trigger two functions simultaneously by defining:

```
Trigger(OBJECT_CAPTURE_TRIGGER, "town1",
"function1");
Trigger(OBJECT_CAPTURE_TRIGGER, "town1",
"function2");
```

or

```
Trigger(NEW_DAY_TRIGGER, "day1");
Trigger(NEW_DAY_TRIGGER, "day2");
```

In both cases only the latter would work, since it overwrites the first. Luckily you can do an intermediate function that triggers as many other functions as you wish.

Example

```
Trigger(NEW_DAY_TRIGGER, "days");

function days()
    day1();
    day2();
end;

function day1()
    commands;
end;

function day2()
    commands;
end;
```

Of course different objects can have the same type triggers without problems and the same object can have different type of triggers without problems.

4.3. Triggering fname(parameter) type of functions

What if the function has some parameter that you need to specify? As explained above, in cases where the parameter refers to currently selected hero, there is no complication. In other cases, avoid using parameters for parameters (long way) or use different sets of quotation marks

Example of the long way

```
Trigger(NEW_DAY_TRIGGER, "transform1");

function transform1()
    transform("town1");
end;

function transform(townname)
    transformtown(townname, 2);
end;
```

Example of the short way

```
Trigger(NEW_DAY_TRIGGER, "transform('town1')");
-- NB! Different types of quotation marks are
used. Check that "transform('town1')" is entirely
orange colored in your editor!

function transform(townname)
    TransformTown(townname, 2);
end;
```

4.4. Removing a trigger

For removing a trigger use “nil” instead of any function name:

```
trigger(event_type, "object to which the event is  
bound", nil);
```

Example script: At day 7 a town called town1 turns into Academy and thereafter the trigger is removed.

```
Trigger(NEW_DAY_TRIGGER, "transform");  
  
function transform()  
    if GetDate(DAY) == 7 then  
        TransformTown("town1", 2);  
        Trigger(NEW_DAY_TRIGGER, nil);  
    end;  
end;
```

5. Syntax

5.1 Are empty lines, semicolons and number of spaces important?

Each “end;” has to have a semicolon, and, although it is not a must, I recommended to have them in the end of each line except the ones that start with *function*, *if*, *else*, *while*, *for* or *repeat*.

You could write all the script in one row without changing the line, but for better tracking it is advised to change line after every individual command.

Having more than one space instead of one or adding empty lines between two functions does not affect the script functionality.

5.2. Case sensitivity

Yes, commands as well as variables and map object names are all case sensitive. Spelling mistakes are among the most common errors when a script does not run.

5.3. Quotation marks and their use

Proper using of quotation marks with functions, variables and map object names is quite confusing, but it follows a system nevertheless. Additionally, note that there are two types of quotes that can be used “ and ‘. Generally it does not matter which you use as long as start and ending marks are the same. A specific case is given under “Triggering funcname(parameter) type of functions”.

5.3.1. USE NO QUOTES

- When declaring a function:

Example

```
function blabla() -- blabla is without quotes.
    MessageBox("Maps/SingleMissions/test/blah.txt");
end;
```

- Numbers and the hard-coded “all capital letters” parameters (like NEW_DAY_TRIGGER or PLAYER_1) are always without quotes. These capitalized words are actually names of certain numbers (see official file [IDs_for_scripts.pdf](#)).
- The function name acts as a command. Namely, each function that you write can be called up in the same way as the hard-coded commands.

Example

```
function blabla()
    MessageBox("Maps/SingleMissions/test/blah.txt");
end;

function day2()
    if GetDate(DAY) == 2 then
        blabla(); -- no quotes around blabla.
    end;
end;
```

- Names of local and global variables are always without quotes. Note that “game variables” which are accessible via SetGameVar and GetGameVar are different.

Example

```
function transform(townname) -- townname without
                                quotes since it
                                is a variable.

    local towntype = 2; -- defining a local variable called towntype. No quotes anywhere.
    TransformTown(townname, towntype); -- townname and towntype are both variables. The value of the first is defined when the function is called up (see examples below), the value of the second was set to 2 right before.

end;
```

5.3.2. USE QUOTES WHEN

- the name of a function, adventure map object, scenario objective etc. is a parameter for a command. For extra dummies: use quotes for everything that is inside parenthesis and is a name of a function, messagetext file, hero, adventure map object or scenario objective.

Example

```
function blabla()
    if GetDate(DAY) == 2 then
        MessageBox("Maps/SingleMissions/test/blah.txt"); -- quotes required for the text file name.
    end;
end;

Trigger(NEW_DAY_TRIGGER, "blabla"); -- quotes required for function blabla name
```

- quotes are required when calling up or saving game variables. See official manuals for examples.

Example script: you have a map with towns named (in respective town properties window) town1 and town2. A (unnecessarily long, but illustrative) script that changes town1 to academy upon capture:

```
function transform (townname) -- townname without
                                quotes since it is a variable
    TransformTown(townname, TOWN_ACADEMY); -- townname is a variable name and TOWN_ACADEMY is, well, “all capital letters” parameter.

end;
```

```
function town1capture()
    transform("town1") -- transform, which is the
                        name of another function acts
                        as a command. It uses param-
                        eter town1, which is the name
                        of one of the towns.

end;

Trigger(OBJECT_CAPTURE_TRIGGER, "town1",
"town1capture"); -- town1 and town1capture are
names of map object and function,
respectively. They are parameters
for Trigger command and not vari-
ables, thus must have quotes.
```

5.4. How to refer text message files, map objects and heroes

For beginners it can be very hard to get the references to external names correctly.

Referring to text files requires full path starting from Maps directory.

For example

```
MessageBox("Maps/SingleMissions/MyMap/
textfilename.txt");
-- MyMap is the name of your map
-- textfilename.txt is the text file name you
created. (Open map properties tree and chose
resources → SavesFileNames → right click and
"add" → make a new file and add message text
into it)
```

Referring to **map objects** other than heroes goes like that: Select the object and hit spacebar (or open object properties tree). Find the field called "name" and type a good name there. In script you can simply use the name and the object is found. When giving identical name to two or more objects, you do not get an error message right away, but the script may crash when run.

Referring to **heroes** goes via their in-game ID names. You can give them a name by yourself, but scripts do not recognize them. Not even the standard game names are not understood.

Here is the list of standard names and ID names (the unbolded names):

Agrael Agrael	Galib Tan	<i>Laszlo Laszlo</i>	Segref Segref
Alaron Ildar	Giar Giar	Lethos Dalom	Shadya Kelodin
Alastor Efion	Gilraen Gillion	<i>Lorenzo RedHeavenHero02</i>	Sinitar Inagost
<i>Andreas RedHeavenHero01</i>	<i>Giovanni Giovanni</i>	Lucretia Tamika	Sorgal Ferigl
Anwen Metlirn	Glen Glen	Maahir Maahir	<i>Svea Vegeyr</i>
Biara Biara	Godric Godric	Maeve Maeve	Talanar Nadaur
*Brand Brand	Grawl Calid	Marbas Marder	Temkhan Timerkhan
Cyrus Cyrus	Grok Grok	Markal Berein	<i>Thralsai Thralsai</i>
Deirdre Nemor	<i>Guarg Guarg</i>	Naadir Muscip	<i>Valeria RedHeavenHero03</i>
Deleb Deleb	Havez Havez	Narxes Razzak	Vayshan Ohtarig
Dirael Diraya	<i>Helmar Ottar</i>	Nathir Nur	Vinrael Elleshar
Dougal Orrin	<i>Inga Una</i>	Nebiros Jazaz	Vittorio Christian
<i>Duncan Duncan</i>	<i>Ingvar Ingvar</i>	Nicolai Nicolai	Vladimir Pelt
<i>Ebba Bersy</i>	Irina Ving	Nur Astral	<i>Wulfstan Wulfstan</i>
Ellaine Nathaniel	Isabel Isabell	Nymus Nymus	Wyngaal Linaas
Erasial Erasial	Jezebeth Oddrema	<i>Ornella Ornella</i>	<i>Ylaya Shadwyn</i>
Ergar Ergar	Jhora Sufi	Orson Straker	Ylthin Itil
<i>Erling Egil</i>	<i>Karli Skeggy</i>	Ossir Ossir	Yrbeth Almegir
Eruina Eruina	Kaspar Gles	Raelag Raelag	Yrwanna Urunir
Faiz Faiz	<i>King Tolghar KingTolghar</i>	Raven Effig	Zehir Zehir
Findan Heam	Klaus Sarge	Razzak Isher	Zoltan Aberrar
Freyda Axel	Kythra Menel	<i>Rolf Rolf</i>	
<i>Freyda Freyda</i>	Laszlo Mardigo	Rutger Brem	

Italic indicates names from the Hammers of Fate expansion pack.

(Names are from Curios cheat guide http://www.heroesofmightandmagic.com/heroes5/heroes5_cheats_names.shtml)

5.5. Meaning of some symbols

<code>=</code>	means give value to. <code>m = 2</code> means that henceforth variable <code>m</code> has value 2.	<code>AND</code>	between two boolean tests means that both tests have to be true in order to proceed
<code>==</code>	compares whether the left side is equal to the right side. (returns true or false)	<code>OR</code>	between two boolean tests means that one of the tests has to be true in order to proceed
<code><=</code>	compares whether the left side is smaller than the right side. (returns true or false)	<code>random(number)</code>	returns a random number between 1 and specified number. Both inclusive.
<code>>=</code>	compares whether the left side is larger than the right side. (returns true or false)	<code>mod</code>	the left-over of dividing. <code>11 mod 2</code> equals to 1; <code>10 mod 2</code> equals 0.
<code>~=</code>	compares whether the left side is NOT equal to the right side. (returns true or false)	<code>nil</code>	– means “false” or nil or “none”. Used for disabling.

6. Hints for debugging

6.1. Enabling console

Adding string

```
setvar dev_console_password = schwing-des-todes
```

as the last line of `autoexec.cfg` file.

Under

My Documents/My Games/Heroes of Might and Magic V/
Profiles

find `input.cfg` file and add

```
bind show_console ""
```

right after the line saying

```
bind enter_pressed 'NUM_ENTER'.
```

In case of having **Hammers of Fate** expansion you need to edit `input_a1.cfg`. (A different key might be needed for non-english keyboards).

In game you should now be able to enter the console by pressing ` (the key above **TAB**).

Console shows you many things about the game, including error messages for faulty scripts.

6.2. Using the print command

With console activated, you can track you scripts by using the `print` command. It prints specified message or variable into the console window. You can put it pretty much anywhere between other commands and see whether the script runs that far or the problem occurs before.

```
print('thus far the script works');
print(variable1); -- shows the value of this
                    variable in the console window.
```

6.3. My script does not run!

- Enable console and read the error messages that are displayed there.
- Double-check spelling in the script.
- Double-check tiggers. Maybe the trigger is accidentally set to run under different conditions than you wish.
- Check that there are enough “end;” commands. A common problem is that the program does not understand properly where a function ends and therefore ignores (a part of) the script file.
- Make sure that the map is `SingleMission` and not `MultiPlayer` (open the `*.h5m` file with zip or rar archiver and see the names of subdirectories).

