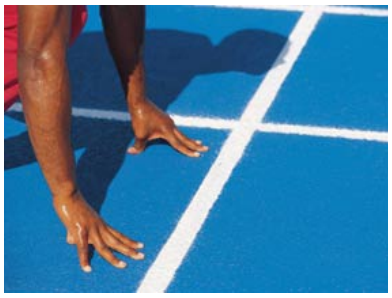
~~maXbox~~



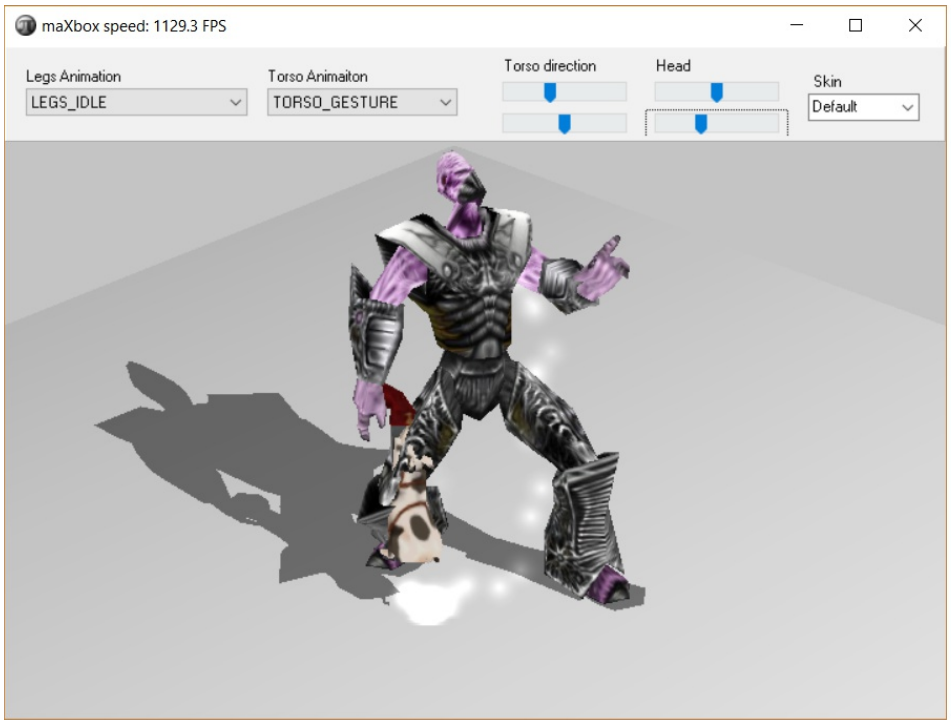
maXbox Starter 49

Work with Refactoring

1.1 Simplification

Learning how to refactor code, has another big advantage beside quality assurance it teaches you how to write good code at first try. Moreover because forecasting which features will be added in future some project mistakes are often unavoidable, refactoring gives you a tool and principle how to deal with it.

Each refactoring in the following is shown in detailed small steps so it is easily reproduced in your own system behaviour. At first I show refactoring steps with so called simplification rule steps and all functions have the same functionality so only substance changed. But in real projects, many methods not only way to refactoring, so it's not easy to rule real projects.



The most important aspect that is emphasized in this example is that you should apply refactoring when you need to, that means, if you don't understand your own code or when you add a feature or fix a bug:

```
procedure getUTCTime;
var
  s: string;
  lHTTP: TIdHTTP;
  lReader: string; //TStringReader;
begin
  lHTTP := TIdHTTP.Create(nil);
  try
    lReader := lHTTP.Get('http://tycho.usno.navy.mil/cgi-bin/timer.pl');
    //while lReader.Peek > 0 do begin
      //s := lReader.ReadLine;
      if Pos('UTC', lreader) > 0 then
      begin
        Writeln(lreader);
        //Break;
      end;
    //end;
  finally
    //lhttp.close;
    lHTTP.Free;
    //lReader.Free;
  end;
 end;
```

You see ugly code with console output in a procedure and nasty comments like dead code, second refactor is needed:

```
function getUTCTime2x: string;
var
  lHTTP: TIdHTTP;
  lReader: string; //TStringReader;
  apos: integer;
begin
  lHTTP := TIdHTTP.Create(Nil);
  try
    lReader:= lHTTP.Get('http://tycho.usno.navy.mil/cgi-bin/timer.pl');
      //s := lReader.ReadLine;
      apos:= Pos('UTC', lreader);
      if aPos > 0 then begin
        result:= copy(lreader,apos-10,15);
        //Break;
      end;
      //function getMatchString(arex, atext: string): string;
      //result:= getMatchString('<BR>(.+ UTC)', lReader);
      //ReplaceRegExpr('<BR>(.+ UTC)',lReader,' ', True)
  finally
```

```
    lHTTP.Free;
    lhttp:= Nil;
  end;
 end;
```

A bit better, now a function with a result but with more comments and hard coded literals, we say a no go, third try:

```
function getUTCTime3: string;
var
  lHTTP: TIdHTTP;
begin
  lHTTP := TIdHTTP.Create(Nil);
  try
     result:=
        copy(lHTTP.Get('http://tycho.usno.navy.mil/cgi-
bin/timer.pl'),Pos('UTC', lHTTP.Get('http://tycho.usno.navy.mil/cgi-
bin/timer.pl'))-10,15);
   finally
    lHTTP.Free;
    lhttp:= Nil;
  end;
 end;
```

Now shorter but still the literals hard coded:

```
function getUTCTime4: string;
begin
  with TIdHTTP.Create(Nil) do
  try
    result:=
      copy(Get('http://tycho.usno.navy.mil/cgi-bin/timer.pl'),
         Pos('UTC', Get('http://tycho.usno.navy.mil/cgi-
bin/timer.pl'))-10,15);
  finally
    Free;
   end;
 end;
```

As I said not just going blindly and refactoring everything. What we have in our team as a good rule of thumb is basically what is promoted in this steps - once you start working on a new feature or fixing a bug and you see any issues with the code you started working on, you start refactoring and then implement the feature or fix the bug.
n some cases you can fix the bug and refactor later, but the important thing is doing only ONE thing a time.

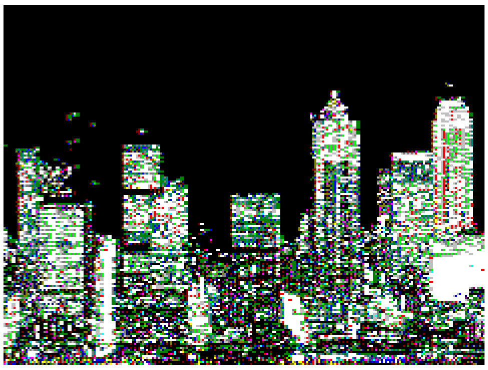
```
Const TIMEURL= 'http://tycho.usno.navy.mil/cgi-bin/timer.pl';

Function getUTCTime5: string;
begin
```

```
  with TIdHTTP.Create(Nil) do
  try
    result:=
      copy(Get(TIMEURL),Pos('UTC',Get(TIMEURL))-10,15);
   finally
    Free;
   end;
 end;
```

OK. step 5 looks affirmative, we have a const, we wrap or map the object to a function and free the resources too.
These instructions can then be sent to a machine that will interpret these lines and execute them one by one, in the end we get sort of internet time from an external service. But the copy function to extract the time of the html stream is a bit strange, what about to set a better known pattern like a match mask or regex function for reuse:

```
Function getUTCTime6: string;
begin
  with TIdHTTP.Create(Nil) do
   try
     result:= getMatchString('<BR>(.+ UTC)',Get(TIMEURL));
   finally
     Free;
   end;
 end;
```

So that's the last function we get.
Remember that refactoring is a process, not an event.



On the first hand you should have a robust test suite in order to be certain that refactoring didn’t change the behaviour of the software. Furthermore, you should never start writing new features on top of code that looks like it needs refactoring.
First refactor and then write new code. Lastly, refactoring should not stop the progress of your project, but be continuously integrated with the development phase.

As you may know we script it in maXbox4. This tool is great for fast coding but also provides mechanism for extending your functions and quality with checks and tests.
The script in this tutor 49 is called:

```
     examples\745_Web_scraping_UTCTime_DNSQuery3txt.txt
```

The call of the function in the end is:

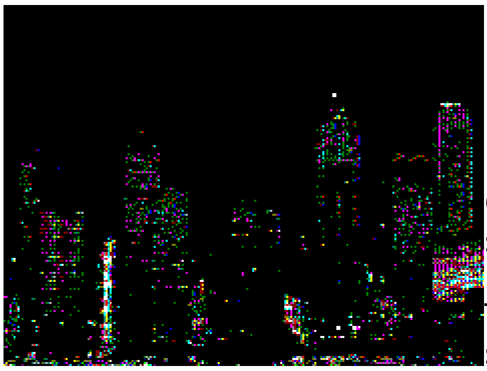
```
if IsInternet then begin
    getUTCTime;
    writeln('*************filtered:')
    writeln(getUTCTime6)
  end;
```

In step 6 there's still a literal, the regex itself. If empty then default (hard-coded) message is used. Default value is "" (empty). But avoid magic numbers or hard coded string literals. Better is:

```
Const REX_IN = '<BR>(.+ UTC)';

Function getUTCTime7: string;
begin
  with TIdHTTP.Create(Nil) do
   try
     result:= getMatchString(REX_IN,Get(TIMEURL));
   finally
     Free;
   end;
 end;
```

Comments about Regex is almost every time useful!

1.2 Beyond Refactoring



֍ & ֎ ҉

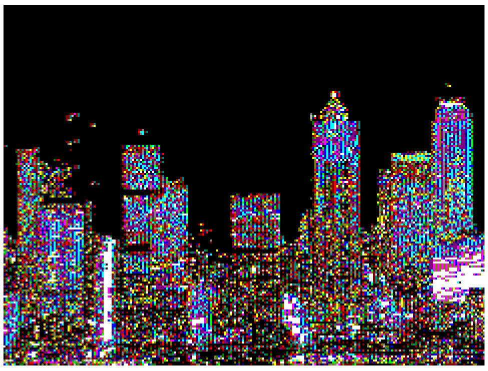
Challenging domains such as robot-assisted search and rescue, operations in space require humans to interact with robots. These interactions may be in the form of supervisory control, which connotes a high human involvement with limited robot automation (e.g., semi-autonomy, where the robot is truly autonomous for portions of the task or mixedinitiative systems, where the robot and human are largely interchangeable.

The interactions between humans and robots are often interchangeably referred to as “coordination” or “collaboration”.

The application of a script to interleave human and robot coordination is both logical and natural. Scripts simplify the relationship between human and robot making the task comprehensible to both novice and expert system users.

```
procedure StartBtnClick(Sender: TObject);
{User clicked start}
var  i:integer;
  drawtime:integer;
  startcount,stopcount:int64;
begin
  for i:= 2 to count do begin
    {put center of robo on the point}
    Robo.left:= saved[i].x-Robo.width div 2;
    Robo.top:= saved[i].y-Robo.height div 2;
    queryperformanceCounter(startcount);{Get time before we repaint}
    application.processmessages;
    queryPerformanceCounter(stopcount);{Get time after repaint}
    drawtime:= (stopcount-startcount) div freq;  {Compute ms to repaint}
    writeln('test freq ms: '+itoa(drawtime))
    sleep(max(0,sleepms-drawtime));   {wait whatever time is left, if any}
  end;
end;
```

The ability to simplify a task as a series of simple steps is necessary for this comprehension. The available or possible actions for the human operator at any particular time are clear in the script because of the GUI. This approach can be applied to any task, with any level of human-robot coordination.



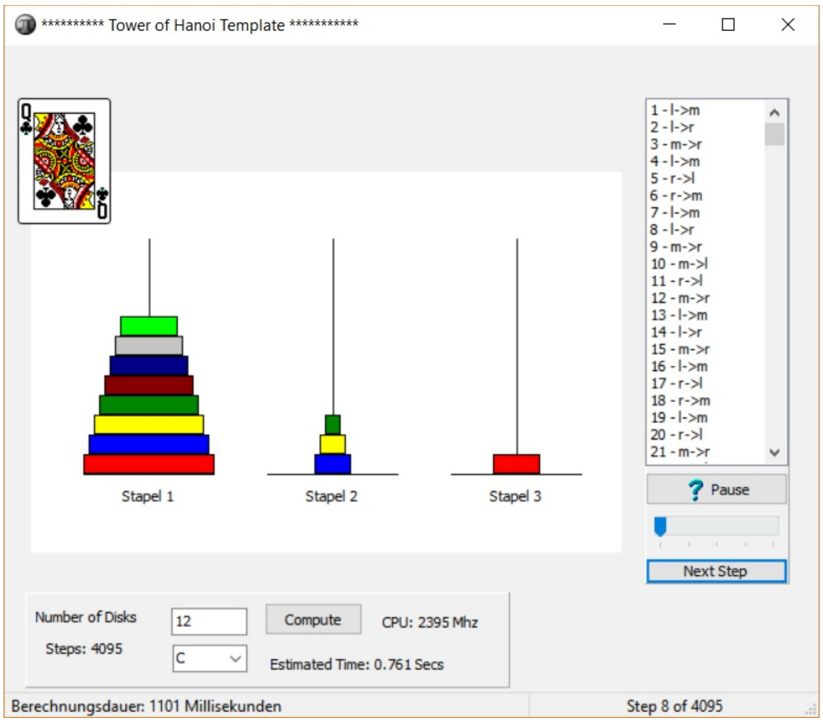
Due to the highly proprietary nature of robot software, most producers of robot hardware also provide their own software. While this is not unusual

in other automated control systems, the lack of standards of programming methods for robots does pose certain challenges.
For example, there are over 30 different manufacturers of industrial robots, so there are also 30 different robot programming languages required.
Fortunately, there are enough similarities between the different robots that it is possible to gain a broad-based understanding of robot programming without having to learn each manufacturer's proprietary language.

Another interesting approach is worthy of mention. All robotic applications need or explore parallelism and event-based programming.
Robot Operating System is an open-source platform for robot programming using Python, CPascal and C++. Java, Lisp, Lua, Pascal and Pharo are supported but still in experimental stage.

https://en.wikipedia.org/wiki/Robot_Operating_System

Another example is the tower of hanoi which can be solved for example by Lego mindstorm or other arduino frameworks:



********** Tower of Hanoi Template ***********

```
1 - l->m
2 - l->r
3 - m->r
4 - l->m
5 - r->l
6 - r->m
7 - l->m
8 - l->r
9 - m->r
10 - m->l
11 - r->l
12 - m->r
13 - l->m
14 - l->r
15 - m->r
16 - l->m
17 - r->l
18 - r->m
19 - l->m
20 - r->l
21 - m->r
```

Number of Disks: 12
Steps: 4095
C
CPU: 2395 Mhz
Estimated Time: 0.761 Secs
Berechnungsdauer: 1101 Millisekunden
Step 8 of 4095

The script is called: `examples\712_towerofhanoi_animation.pas`

Programming errors represent a serious safety consideration, particularly in large industrial robots. The power and size of industrial robots mean they are capable of inflicting severe injury if programmed incorrectly or used in an unsafe manner. Due to the mass and high-speeds of industrial robots, it is always unsafe for a human to remain in the work area of the robot during automatic operation.
A few following concepts should improve this safety:

- Console Capture
- Exception Handling
- Logfiles

1.3 Console Capture DOS

I'm trying to move a part of SysTools to Win64. There is a certain class `TStDecimal` which is a fixed-point value with a total of 38 significant digits. The class itself uses a lot of ASM code.

```
function BigDecimal(aone: float; atwo: integer): string;
begin
 with TStDecimal.create do begin
  try
   //assignfromint(aone)
   assignfromfloat(aone) //2
   RaiseToPower(atwo)  //23
   result:= asstring
  finally
   free
  end;
 end;
end;
```

But then I want to test some Shell Functions on a DOS Shell or command line output. The code below allows to perform a command in a DOS Shell and capture it's output to the maXbox console. The captured output is sent “real-time” to the Memo2 parameter as console output in maXbox:

```
    srlist:= TStringlist.create;
      ConsoleCapture('C:\', 'cmd.exe', '/c dir *.*',srlist);
      writeln(srlist.text)
    srlist.Free;
```

But you can redirect the output `srlist.text` anywhere you want.
For example you can capture the output of a DOS console and input into a textbox, or you want to capture the command start of demo app and input into your app that will do further things.

```
 ConsoleCapture('C:\', 'cmd.exe', '/c ipconfig',srlist);
 ConsoleCapture('C:\', 'cmd.exe', '/c ping 127.0.0.1',srlist);
```

☝ It is important to note that some special events like `/c java -version` must be captured with different parameters like /k or in combination.

Here's the solution with `GetDosOutput()`:

```
 writeln('GetDosOut: '+GetDosOutput('java -version','c:\'));
```

or like powercfg or the man-pages in Linux

```
 writeln('GetDosOut: '+GetDosOutput('help dir','c:\'));
GetDosOutput('powercfg energy -output
                          c:\maxbox\osenergy.htm','c:\')
```

1.4 Exception Handling

A few words how to handle Exceptions within maXbox:
Prototype:

```
procedure RaiseException(Ex: TIFException; const Msg: String);
```

Description:
Raises an exception with the specified message.
Example:

```
begin
  RaiseException(erCustomError,'Your message goes here');
// The following line will not be executed because of the
exception!
  MsgBox('You will not see this.', 'mbInformation', MB_OK);
end;
```

This is a simple example of a actual script that shows how to do try except with raising a exception and doing something with the exception message.

procedure Exceptions_On_maXbox;

```
var filename,emsg:string;
begin
   filename:= '';
   try
     if filename = '' then
       RaiseException(erCustomError,
                    'Exception: File name cannot be blank');
   except
     emsg:= ExceptionToString(ExceptionType, ExceptionParam);

       //do act with the exception message i.e. email it or
       //save to a log etc

     writeln(emsg)
   end;
end;
```

☝ The `ExceptionToString()` returns a message associated with the current exception. This function with parameters should only be called from within an except section.

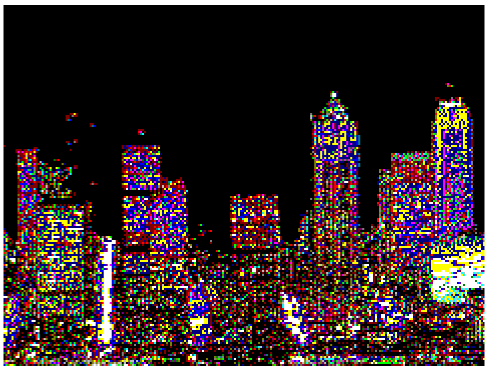
1.5 The Log Files

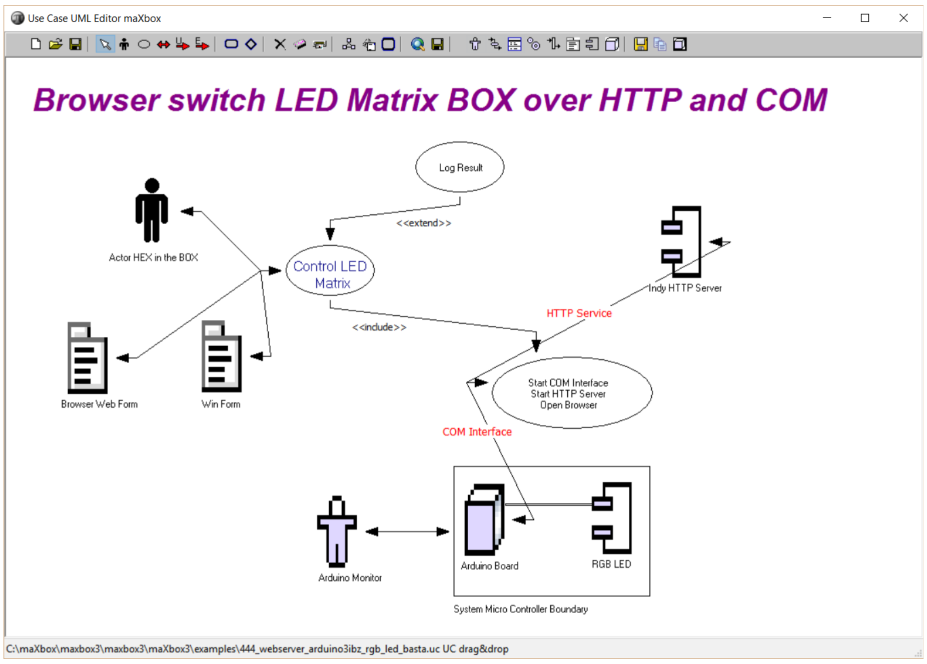
There are 2 log files a runtime log and an exception log:
using Logfile: `maxboxlog.log` , Exceptionlogfile: `maxboxerrorlog.txt`

```
New Session Exe Start C:\maXbox\tested
>>>> Start Exe: maXbox4.exe v4.0.2.80 2016-02-03 14:37:18
>>>> Start [RAM  monitor] : Total=2147483647,
Avail=2147483647, Load=30% ; [Disk monitor] : Available to
user=671317413888, Total on disk=972076589056, Free total on
disk=671317413888 ; 2016-02-03 14:37:18

>>>> Start Script: C:\Program Files
(x86)\Import\maxbox4\examples\640_weather_cockpit6_1.TXT
2016-02-03 14:37:33 From Host: maXbox4.exe of
C:\maXbox\maxbox3\work2015\Sparx\
>>>> Stop Script: 640_weather_cockpit6_1.TXT
[RAM  monitor] : (2147483647, 2147483647, 30%) Compiled+Run
Success! Runtime: 14:37:33.267

New Session Exe Start C:\Program Files(x86)\Import\maxbox4\
examples\640_weather_cockpit6_1.TXT
>>>> Start Exe: maXbox4.exe v4.2.2.95 2016-05-19 09:15:17
>>>> Start [RAM  monitor] : Total=2147483647,
Avail=2147483647, Load=25% ; [Disk monitor] : Available to
user=675888001024, Total on disk=972076589056
```

After completing the tasks as directed, the compiler proceeds to its second step where it checks for syntax errors (violations of the rules of the language) and converts the source code into an object code that contains machine language instructions, a data area, and a list of items to be resolved when he object file is linked to other object files.

At least there are two ways to install and configure your box into a directory you want. The first way is to use the unzip command-line tool or IDE, which is discussed above.
That means no installation needed. Another way is to copy all the files to navigate to a folder you like, and then simply drag and drop another scripts into the /examples directory.
The only thing you need to backup is the ini file **`maxboxdef.ini`** with your history or another root files with settings that have changed.





You simply put the line above on the boot script and make sure the ini file has it set to Yes. BOOTSCRIPT=Y //enabling load a boot script

“Wise men speak: Hand made by robots.

Feedback @ max@kleiner.com

Literature: Kleiner et al., Patterns konkret, 2003, Software & Support

https://github.com/maxkleiner/maXbox3/releases

https://en.wikipedia.org/wiki/Robot_software

EKON 21 Cologne 2017 Input for Robots & Components

1. AsyncPRO,	2. BigInteger,
3. Hotlog,	4. WMILib,
5. StBarCode,	6. XMLUtils,
7. LockBox,	8. cX509Certificate,
9. OpenGL,	10. WaveUnit,
11. OpenSSL,	12. Kronos,

13. HiResTimer,	14. Kmemo,
15. BigDecimals,	16. SynEdit,
17. SFTP,	18. Sensors,
19. PasScript,	20. ALJSON

1.6 Appendix External links External links

- "*The Basics - Robot Software*". Seattle Robotics Society.
- G.W. Lucas, "*Rossum Project*".
- "*Mobile Autonomous Robot Software* (MARS)". Georgia Tech Research Corporation.
- "*Tech Database*". robot.spawar.navy.mil.
- Adaptive Robotics Software at the Idaho National Laboratory
- A review of robotics software platforms Linux Devices.
- ANSI/RIA R15.06-1999 American National Standard for Industrial Robots and Robot Systems - Safety Requirements (revision of ANSI/RIA R15.06-1992)

1.7 References

- *O. Nnaji, Bartholomew. Theory of Automatic Robot Assembly and Programming (1993 ed.). Springer. p. 5. ISBN 978-0412393105. Retrieved 8 February 2015.*
- *"Robot programming languages". Fabryka robotów. Retrieved 8 February 2015.*

```
             _od#HMM6&*MMMH::-_
          _dHMMMR??MMM? ""| `"'-?Hb_
       .~HMMMMMMMMMHMMM#M?       `*HMb.
     ./?HMMMMMMMMMMM"*"""           &MHb.
    /'|MMMMMMMMMMM'             -   `*MHM\
   /  |MMMMMHHM''                    .MMMHb
  |   9HMMP   .Hq,                   TMMMMMH
 /     |MM\,H-""&&6\__               `MMMMMMb
|       `""HH#,       \             - MMMMMMM|
|           `HoodHMM###.              `9MMMMMH
|              .MMMMMMMM##\             `*"?HM
|         ..  ,HMMMMMMMMMMMMo\.             |M
|             |M'MMMMMMMM'MMMMMHo           |M
|              ?MMM'MMMMMMM'MMMM*           |H
|.              `#MMMM'MM'MMMMM'           .M|
 \                `MMMMMMMMMMM*            |P
 `\                MMMMMMMMT"'            ,H
  `\              `MMMMMMH?              ./
    \.            |MMMH#"               ,/
     `\.          |MMP'               ./'
       `~\        `HM:.-    .       ,/'
          "-\_       '_\ .      _.-"
              "-\-#odMM\_,oo==-"
```