

Beginners Game Programming with Naalaa



Introduction

NaaLaa (Not An Advanced Language At All) is a basic like programming language. And if you are interested in making retro style games it might be worth a shot. These are some of the features:

- Very easy to learn
- Create standalone programs for windows
- Generate java applets for the web
- Export and import libraries
- Use the NaaLaa standard libraries and tools to make advanced games in no time
- It's freeware

Web: <http://www.naalaa.com/>

Forum: <http://forum.retrogamecoding.org/>

Screenshots of Some games coded in NaaLaa



Jewel Crusher



Juan Hakholt



Blastemroids



Alien Bomber

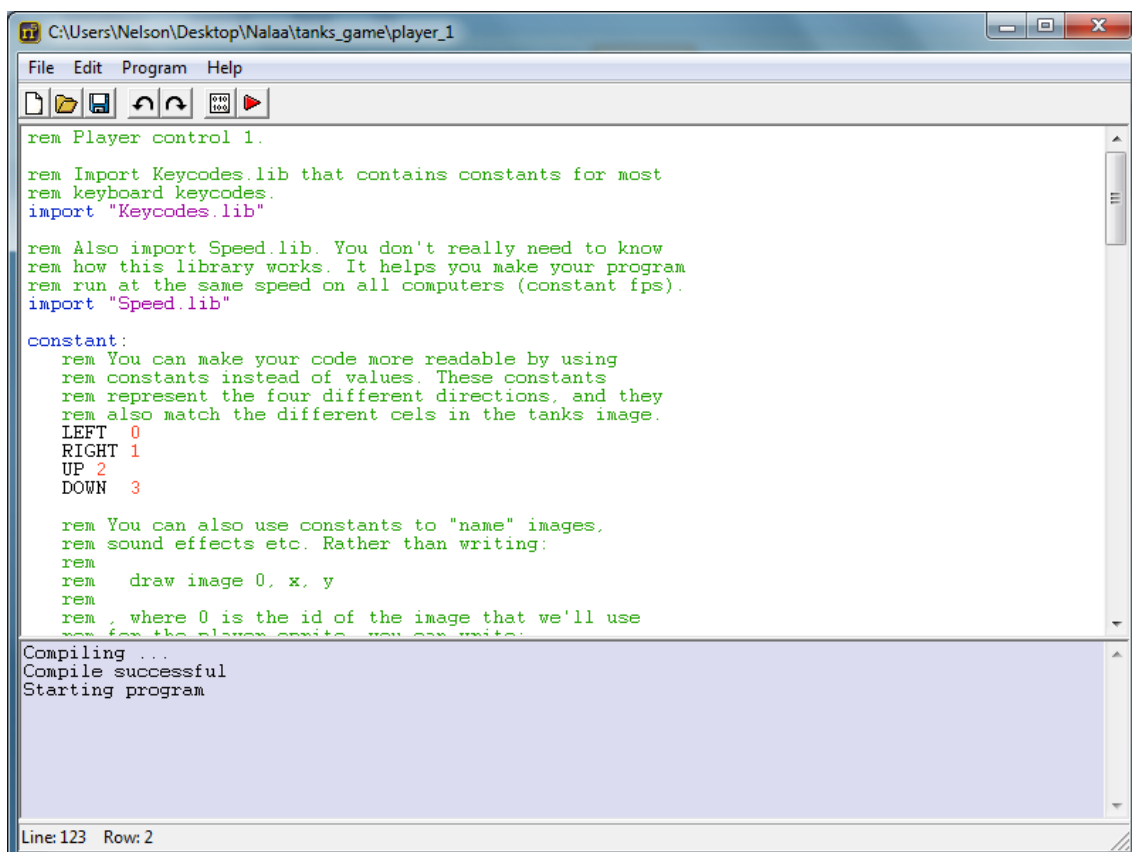
It can be found here: <http://games.naalaa.com/>

Beginners start your engines

In this tutorial we are going to learn how to code a simple and funny game that was one of my favorites back in the days.

This code and graphics were provided by Mopz, The Genius behind NaaLaa programming language.

We are going to use NED (NaaLaa Developer Environment), but there are other called NED2 that can also be used, below is a screenshot of NED.



```
rem Player control 1.

rem Import Keycodes.lib that contains constants for most
rem keyboard keycodes.
import "Keycodes.lib"

rem Also import Speed.lib. You don't really need to know
rem how this library works. It helps you make your program
rem run at the same speed on all computers (constant fps).
import "Speed.lib"

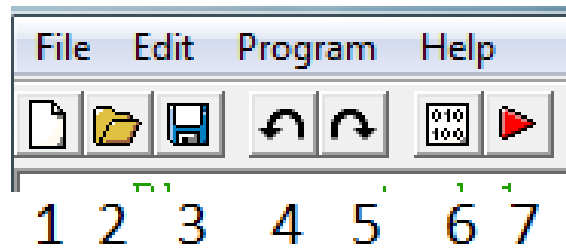
constant:
  rem You can make your code more readable by using
  rem constants instead of values. These constants
  rem represent the four different directions, and they
  rem also match the different cells in the tanks image.
  LEFT 0
  RIGHT 1
  UP 2
  DOWN 3

  rem You can also use constants to "name" images,
  rem sound effects etc. Rather than writing:
  rem
  rem   draw image 0, x, y
  rem
  rem , where 0 is the id of the image that we'll use
  rem for the player sprite, you can write:
```

Compiling ...
Compile successful
Starting program

Line: 123 Row: 2

As you can see is a simple editor and very efficient, the menu is simple and nice for beginners to understand, here is a screenshot of the NED menu.



1. New File
2. Open a File
3. Save
4. Undo
5. Undo
6. Compile
7. Run

Now that we are introduced let's start coding, but first we have to know some programming basics like variables, loops and other stuff that I will try to explain as we go ahead in the tutorial.

One of the things that we have to understand is a good programming practice, and for that we must indent the code for a better reading and comment the same code, this is a good thing because latter on if the game needs to be improved we know what that piece of code does.

For Naalaa comments instead of using (//) or ('), the author used a word called *Rem*, this is an old school basic programming word to comment the code.

rem You can make your code more readable by using rem constants instead of values.

For beginning the tutorial I will put the entire code and then I will explain it block by block and word by word, so let's start.

```
rem Player control 1.

rem Import Keycodes.lib that contains constants for most
rem keyboard keycodes.
import "Keycodes.lib"

rem Also import Speed.lib. You don't really need to know
rem how this library works. It helps you make your program
rem run at the same speed on all computers (constant fps).
import "Speed.lib"

constant:
rem You can make your code more readable by using
rem constants instead of values. These constants
rem represent the four different directions, and they
rem also match the different cels in the tanks image.
LEFT 0
RIGHT 1
UP 2
DOWN 3

rem You can also use constants to "name" images,
rem sound effects etc. Rather than writing:
rem
rem draw image 0, x, y
rem
rem , where 0 is the id of the image that we'll use
rem for the player sprite, you can write:
```

```
rem
rem draw image PLAYER_IMAGE, x, y
rem
rem , if you've declared a constant named
rem PLAYER_IMAGE, as below.
PLAYER_IMAGE 0
```

hidden:

```
rem Put window at 16, 16 and resize it to 640x480.
set window 16, 16, 640, 480
```

```
rem Turn off automatic redraw for flicker free graphics.
set redraw off
```

```
rem Load player image.
load image PLAYER_IMAGE, "data/yellow_tank.bmp"
```

```
rem If you look at yellow_tank.bmp, you'll see that the image
rem contains four images of a tank. Each image is 24x24 pixels
rem in size. With the command "set image grid", you can
rem virtually split an image into several equally sized sub
rem images.
```

```
rem In this case we want the image split into 4 columns and
rem 1 row.
set image grid PLAYER_IMAGE, 4, 1
```

```
rem We also want to make the black color (RGB: 0, 0, 0)
rem transparent. That can be done with "set image colorkey".
set image colorkey PLAYER_IMAGE, 0, 0, 0
```

```
rem Now we try to be a little modern and create an object
```

rem (called record or struct in other languages) for the
rem player. You create an object by adding a ? character after
rem the variable name.

player?

rem The player object should contain the player's current
rem direction (any of the constants LEFT, RIGHT, UP or DOWN),
rem and the position (x, y). We can also store the player's
rem speed.

player.direction = RIGHT

player.x = 320 - width(PPLAYER_IMAGE)/2

player.y = 240 - height(PPLAYER_IMAGE)/2

player.speed = 2

do

rem Move player tank with cursor keys.

rem Left.

if keydown(VK_LEFT)

rem If the player is already turned left,

rem we move it by decreasing the x field.

rem This condition may seem unimportant,

rem but the player might want to change

rem direction without moving.

if player.direction = LEFT

player.x = player.x - player.speed

endif

player.direction = LEFT

rem Right.

elseif keydown(VK_RIGHT)

if player.direction = RIGHT

player.x = player.x + player.speed

endif

```

player.direction = RIGHT
rem Up.
elseif keydown(VK_UP)
if player.direction = UP
player.y = player.y - player.speed
endif
player.direction = UP
rem Down.
elseif keydown(VK_DOWN)
if player.direction = DOWN
player.y = player.y + player.speed
endif
player.direction = DOWN
endif

rem Keep player within window. The min function returns
rem the smallest of two values, and the max function
rem returns the largest.
player.x = max(player.x, 0)
player.x = min(player.x, 640 - width(PAYER_IMAGE))
player.y = max(player.y, 0)
player.y = min(player.y, 480 - height(PAYER_IMAGE))

rem Draw.

rem Draw background.
set color 0, 16, 32
cls

rem Draw player.
set color 255, 255, 255
draw image PAYER_IMAGE, player.x, player.y, player.direction

```

```
rem Copy graphics to window and call SPD_HoldFrame to
rem maintain 50 fps.
redraw
proc SPD_HoldFrame 50
rem Loop until player presses the ESC key.
until keydown(VK_ESC)
```

As I said earlier let's break the code in several pieces:

```
rem Player control 1.
```

The first line is a comment which starts with the word *rem*.

```
import "Keycodes.lib"
```

This line imports a library that is called key codes, this library is very important because it contains constants for most keyboards key codes, without this library we couldn't move the player to the left, right, up and down, imagine that this library as all your keyboard keys. The word import is used by Naalaa to tell that it is needed to use that for the program can work.

```
import "Speed.lib"
```

The speed library is used to keep the same speed in all kind of computer, imagine that you have a computer slow and other faster, the game will run with different speed in each one of them, but if this library is imported that will not happen because she is the responsible to control that.

`constant:`

This line will declare all constant variables, there are two types of variables, the constants and the variables, the first will have always thru the program the same value and is not possible to change, unless you go and change by hand off course, the others can assume any value inside the program.

```
LEFT 0  
RIGHT 1  
UP 2  
DOWN 3  
PLAYER_IMAGE 0
```

This are constants variables that we will use in the program and assume the value 0, 1, 2, 3, so instead of calling the numbers you call the constant by its name, in this case is LEFT, RIGHT and so on. Later in the program this will be more detailed.

`hidden:`

This line is called to hide the variables that we declare so they can be accessible to the other sub routines (Procedures and Functions). Next is an example of an visible variables.

EX:

```
visible:  
i = 0  
j = 0
```

```
for i = 0 to 10
  proc Dummy
next

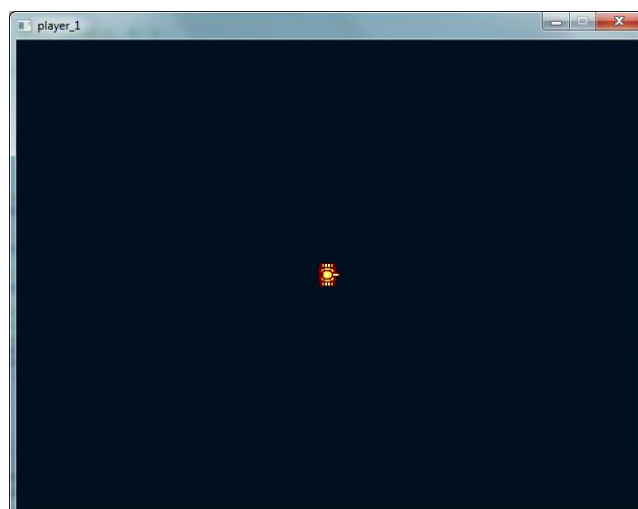
procedure Dummy()
  for i = 1 to 5
    write "hello"
  next
endproc
```

This program would get stuck in an infinite loop. Because every time the procedure Dummy runs, the `VISIBLE` variable `i` is changed. So the loop (for/to/next) in the main program will never end.

Next thing we are going to work on is to build a simple window with `640` height by `480` width, and for that we need the following code:

```
set window 16, 16, 640, 480
```

The `set window` will create a window and is going to be placed in your screen at `16` from the top and `16` from the left side. See the next screen.



This will be our screen, and our tank image, don't forget this is just the beginning of our game, there are more things that will be explained in detail more ahead, as said that lets continue.

```
set redraw off
```

To set off automatic redraw you need to use the line of code above, this will prevent the flickering of the images and the need of automatic redraw, this way it works better and let the `speed.lib` do her job. If you don't understand yet don't worry about it, you will, and if you don't remember when coding a game set always redraw of at the beginning unless you are coding a console program, if you are don't worry about setting the redraw off, let it be.

```
load image PLAYER_IMAGE, "data/yellow_tank.bmp"
```

Next thing to do is loading the images with the `load image` command and if you read since the beginning you should get the `PLAYER_IMAGE`, next the var is the absolute path of the image, in this case the image is inside a folder called data, after this done we have the image loaded and ready to be displayed.

Next is an example how to load an image without the constants variables, this also allowed in Naalaa, but to be more readable we advise you to use constants.

EX:

```
load image 0, "data/yellow_tank.bmp"
```

Get it now why does the constant `PLAYER_IMAGE` has the value `0`?

This is the image that we will use for the game; it's a 24x24 pixels image with 4 columns and one row, this is also known as Tile images.



At this point we have in memory our image, but we didn't tell NaaLaa how to treat this image so next thing we will set a grid to hold and split one image in 4, as you can see in the image above there is only one image and we need 4 to control the tank, LEFT, RIGHT, UP and DOWN.

```
set image grid PLAYER_IMAGE, 4, 1
```

As you can see this line of code creates a `grid` and uses the constant that holds the image and split it in 4 images with only one row.

```
set image colorkey PLAYER_IMAGE, 0, 0, 0
```

Ok, now what we are going to do? Well we could just display the image in the screen, but don't forget that the background of the image it's not transparent if you display it it will display a square with a tank on it, so to clear all this and to set the transparency to the image we going to use the `colorkey`, and don't forget that the `colorkey` have 3 values these values are `RGB` (`RED`, `GREEN`, `BLUE`), and to set the background of the image to black we will use `(0,0,0)`, if the background was white then it would be `(255,255,255)`.

Now we are ready to the next stage, this will be more difficult to understand, but with some trial and error and read over and over again this tutorial and the problem disappear, or not 😊.

Let's start then, in Naalaa to create an object is easier than most of languages I have the pleasure to know, so just to do it we only need the next code.

```
player?
```

That it, voila, and you have declare an object called player, now to see the difference between other language here is a small example.

```
struct {  
    int dia;  
    int mes;  
    int ano;  
} x;
```

See the difference? In Naalaa the only thing that we need is just a question mark. Good, but is not all to make that object useful for our game we need to put something in it, said this go ahead and put the next line of code.

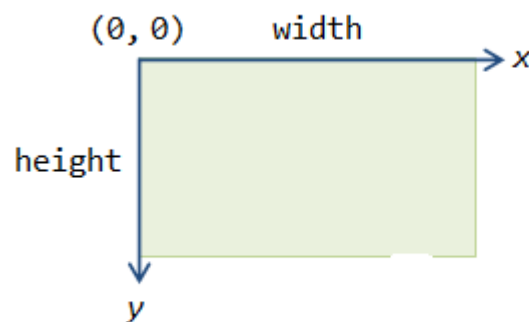
```
player.direction = RIGHT  
player.x = 320 - width(PAYER_IMAGE)/2  
player.y = 240 - height(PAYER_IMAGE)/2  
player.speed = 2
```

We start to add the player object a direction, this will display the first image of the tank in the screen turned to the right position,

next thing is to tell the program where in the screen it will be displayed, so we tell the program to set in the x coordinates the position 320 ($640/2$) less the image with that is $24/2$, and the same to the y coordinates, $(480/2) = 240$ and $24/2$ this is the same as the next example.

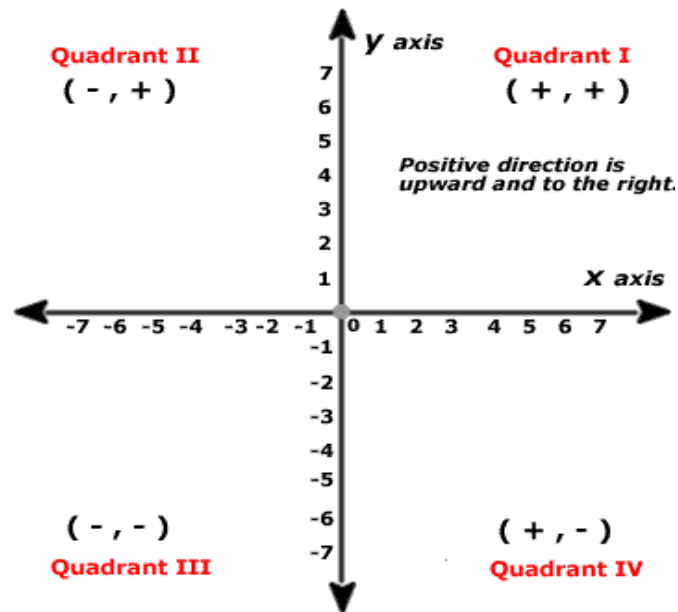
$320 - 12 = 308 \rightarrow X$ coordinates $240 - 12 = 228 \rightarrow Y$ coordinates
--

So the tank is positioned in the $308 X$ and $228 Y$. To understand better the coordinates system see the next image.



This graphic shows the coordinates of your monitor, so the $(0,0)$ is the upper corner from your left, so if you want to increase the y you must go down, instead of going up, get it?

The next image shows a normal coordinates system also known as Cartesian.



See the differences between the two of them? Good, now that we know this let's go to the next line of code.

```
player.speed = 2
```

This line as all we know by now is to set the speed that the tank will move in the screen, increase and decrease it and you will understand it better, go ahead don't be shied, nothing will be lost, don't forget that the better way of learning is by trial and error.

The next step is to set up the game loop, and you ask, what is the game loop? Well the game loop is where all is showed in the screen and where it handles the input from the user, and where all the math is done, so to have a proper game loop in NaaLaa, we must declare the next line.

```
Do  
until keydown(VK_ESC)
```

This is very important in every game, if the loop isn't declared there will be no game, no update, no display no text no nothing, so go ahead and type those two lines of code, this is a continue loop until the user or gamer as you wish, pressed the key down escape, and when that happens then the window will close and the game ends. Of course with the time passing you will understand better and notice that you can change the key or even if the player loses all the lives and... *GAME OVER*.

Now the next thing we must do is checking if the keys were pressed or not and change the position of the player in the *X* and *Y*, so to do that we will write the next line of code inside the loop until something is pressed or happen, so put the fingers in the keyboard and write.

```
if keydown(VK_LEFT)
    if player.direction = LEFT
```

In this case the code is telling the program if the key pressed was the left arrow and the player direction is set to the left then we must move the player; don't forget that the *LEFT* is a constant that is holding the *0* value. Well if these sets of instructions are correct then the player is going to move in the *X*.

To do that we need the next instruction, so type it.

```
player.x = player.x - player.speed
```

Remember the X and Y positions of the tank in the screen?

That's it when the user enters the key left then we going to decrease it, and to do that we use the player X position and subtract to her the player speed, this way the speed and the movement are the same in pixies, and don't forget to the left side of the screen we decrease the X coordinates to the right we increase it and if go up we must subtract (Different right? If you studied the graphics you will get it).

Said this, the if statement must end because we have all we need to move left the player, so let's put the end of the if.

```
endif
```

Simple as that, don't forget that this must come after every end of *//* statement, if you don't put it there then an error will come up, but don't think that is all over because it's not,

As we move everything to the left is obvious that is needed to say that left is the position we set the player, so to finish this all we going to type is:

```
player.direction = LEFT
```

This instruction is already in the first part of the *//*. So if it was checked and the position is left we say to the program stay left.

The next block of code will do the same and the only difference is the increase and decrease parte, so here is it:

```
elseif keydown(VK_RIGHT)
    if player.direction = RIGHT
        player.x = player.x + player.speed
    endif
    player.direction = RIGHT
rem Up.
elseif keydown(VK_UP)
    if player.direction = UP
        player.y = player.y - player.speed
    endif
    player.direction = UP
rem Down.
elseif keydown(VK_DOWN)
    if player.direction = DOWN
        player.y = player.y + player.speed
    endif
    player.direction = DOWN
endif
```

The only difference between the first piece of code and this one is the *elseif* part, this belongs to the first if and not to repeat all ifs, in NaaLaa and other languages uses this command, this is like saying:

Open that door and if that door doesn't open try another until one of those doors opens.

Don't forget to close the first *IF* with *endif*.

The next part is a bit difficult, what we are going to do is check if the player is out of the screen and if that is happening then we must put him back in and to do that we are going to use some built in function called *min* and *max*.

```
player.x = max(player.x, 0)
player.x = min(player.x, 640 - width(PAYER_IMAGE))
player.y = max(player.y, 0)
player.y = min(player.y, 480 - height(PAYER_IMAGE))
```

To understand better this two functions I advice to use two values on it to get what it does, Basically the max function returns the largest number on it, so player.x will have always the biggest number and when it reaches 0 then the value on the player.x stays inside the screen, got it?

```
player.x = min(player.x, 640 - width(PAYER_IMAGE))
```

This line is a bit different because if the image reaches the 640 and the 480 we must subtract the width and height of the image inside the functions, well if you try in the console mode with this two functions you will get it better, but this could be done like this:

```
If player.x > 640 then
  Player.x = 640-24
endif
```

Don't forget that the 24 is the height and width of the image that we strip.

Now that the math is done we are going to give the background a color and to do that we use this:

```
set color 0, 16, 32
```

This doesn't need to be explained but don't forget that the values *0,16,32* are *RGB*.

Finally this is what you guys and I were waiting for, the display of our tank in the screen, finally right!! So here is it.

```
set color 255, 255, 255  
draw image PLAYER_IMAGE, player.x, player.y, player.direction
```

Before we draw the image we set the color to white (*255,255,255*), and then voila, the magic command from NaaLaa to display an image in the screen, aka as *draw image*, the rest is self explanatory, because we already talk about them, the image, the *X* position the *Y* position and the direction that he is facing off.

Now for all this to be copied to the screen we must manually turn on redraw, easy right? To do that write only this line:

```
redraw
```

If you have followed until now everything I have written, and it was a lot and hard, you don't need more explanation for this, but you will need to know what's next because we need to set up an *FPS* (Frames per Second), if you know something about movies and things like that you know what this is, so once more we will set a built in procedure to set our frame in 50 (*FPS*). So the procedure is:

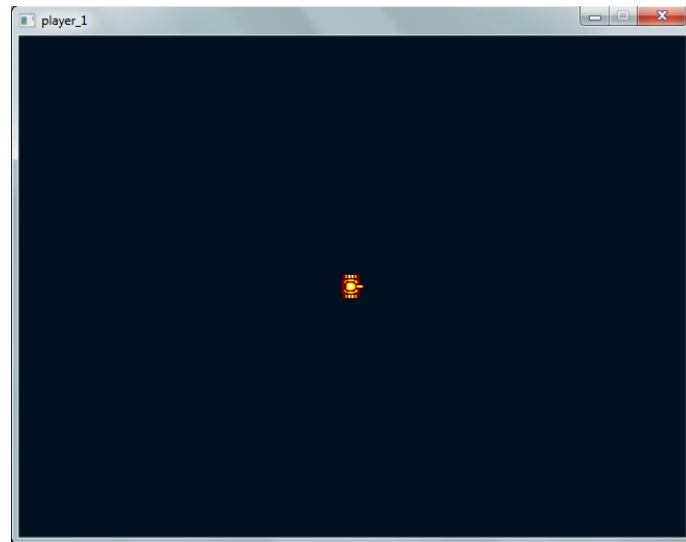
```
proc SPD_HoldFrame 50
```

The name says it all, this is a procedure to hold a 50 frames per second, this way everything goes smooth, we hope.... :). Don't worry about this if you don't follow because this will be used and explained more ahead in this series of tutorials.

And at this point if you did everything right your code should end with this line:

```
until keydown(VK_ESC)
```

This was already explained, this is to be certain that you put the code in the right place, so to end up this tutorial you should have the next image in your screen.



You have the same image then you do it all right and you are in the right track, congratulations. You are now a Naalaa Coder.

If you have any questions please put them in the forum and they will be answered.

This Tutorial is provided by Mopz, he is the owner and he can do whatever he wants with him.