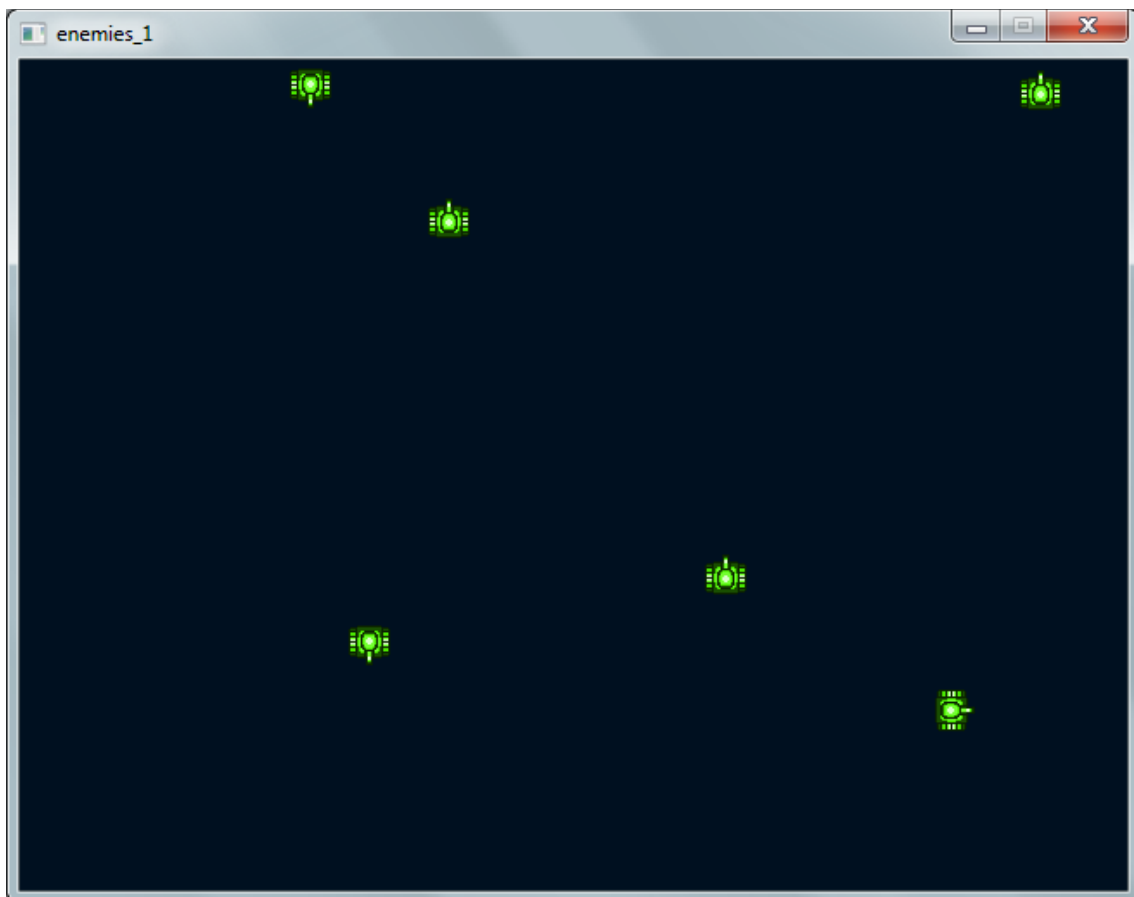


Beginners Game Programming With Naalaa

The Return Of The Enemies



Screen Of the enemies in the battle field

I hope that you guys liked the previous tutorial, and as promised this is the second part of it, in this part I will try to explain how to put enemies on the screen, the player you already know how to, this time is to put random enemies in the battle field, so as the previous tutorial first of all I will put all the code here and then I will explain it line by line even the code that already as been explained.

The code is all comment by the author, so ready it if you have any doubts, it's enough to get almost everything.

Let's start then:

```
rem Enemies with completely random movements.
```

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

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

```
constant:
```

```
LEFT      0
```

```
RIGHT     1
```

```
UP        2
```

```
DOWN      3
```

```
UNDEFINED 4
```

```
rem Max number of visible enemies.
```

```
MAX_ENEMIES 16
```

```
rem Enemy image id.
```

```
ENEMY_IMAGE 1
```

```
visible:
```

```
rem An array with enemy objects.
```

```
vEnemies?[MAX_ENEMIES]
```

```
rem A countdown timer. Every time it reaches 0, we add an enemy.
vEnemyTimer
```

hidden:

```
rem Set window and turn off automatic redraw.
```

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

```
set redraw off
```

```
rem Load enemy image, split it into 4 columns and make the black color
rem transparent.
```

```
load image ENEMY_IMAGE, "data/green_tank.bmp"
```

```
set image grid ENEMY_IMAGE, 4, 1
```

```
set image colorkey ENEMY_IMAGE, 0, 0, 0
```

```
rem Clear enemies.
```

```
proc InitEnemies
```

```
rem Game loop.
```

```
do
```

```
rem Update enemies.
```

```
proc UpdateEnemies
```

```
rem Clear screen.
```

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

```
cls
```

```
rem Draw enemies.
```

```
proc DrawEnemies
```

```
rem Copy graphics to window and call SPD_HoldFrame to keep a
```

```

rem constant fps.
redraw
proc SPD_HoldFrame 50
until keydown(VK_ESC)

rem
=====

rem Init/Clear all enemies.
rem
=====

procedure InitEnemies()
    rem Loop through all enemies and clear the fields.
    for i = 0 to MAX_ENEMIES - 1
        rem If 'active' is true, the enemy is active and moving around
        rem on the screen.
        vEnemies[i].active = false
        rem Direction LEFT/RIGHT/UP/DOWN of enemy.
        vEnemies[i].direction = UNDEFINED
        rem Position of enemy.
        vEnemies[i].x = 0
        vEnemies[i].y = 0
        rem Speed of enemy.
        vEnemies[i].speed = 2
        rem If this flag is true, the enemy is standing still.
        vEnemies[i].pause = false
        rem This timer is used to control for how the enemy should be
        rem moving in a certain direction or standing still.
        vEnemies[i].timer = 0
    next
    vEnemyTimer = 200

```

```

endproc

rem
=====
=====
rem Add a new enemy on a random side of the game field.
rem
=====
=====
procedure AddEnemy()
    rem Find an empty spot in the enemy array.
    index = -1
    for i = 0 to MAX_ENEMIES - 1
        rem Is this enemy free to use?
        if not vEnemies[i].active
            index = i
            break
        endif
    next

    rem If index hasn't been set, no empty spot could be found.
    if index < 0 then return

    rem Activate enemy.
    vEnemies[i].active = true

    rem Set enemy direction to LEFT, RIGHT, UP or DOWN by random.
    vEnemies[i].direction = rnd(4)

    rem Put enemy outside the visible window on the opposite side
    rem compared to its direction. This way, it will enter the game
    rem field nicely.

```

```

if vEnemies[i].direction = LEFT
    vEnemies[i].x = 640
    vEnemies[i].y = rnd(480 - height(ENEMY_IMAGE))
elseif vEnemies[i].direction = RIGHT
    vEnemies[i].x = -width(ENEMY_IMAGE)
    vEnemies[i].y = rnd(480 - height(ENEMY_IMAGE))
elseif vEnemies[i].direction = UP
    vEnemies[i].y = 480
    vEnemies[i].x = rnd(640 - width(ENEMY_IMAGE))
else
    vEnemies[i].y = -height(ENEMY_IMAGE)
    vEnemies[i].x = rnd(640 - width(ENEMY_IMAGE))
endif

```

```

rem The enemy will move 24 + 0.63 steps before doing anything
rem else.

```

```

vEnemies[i].timer = 24 + rnd(64)

```

```

vEnemies[i].pause = false

```

```

endproc

```

```

rem

```

```

=====

```

```

=====

```

```

rem Update enemies.

```

```

rem

```

```

=====

```

```

=====

```

```

procedure UpdateEnemies()

```

```

    rem Count down timer.

```

```

    vEnemyTimer = vEnemyTimer - 1

```

```

    rem Add a new enemy if timer is 0.

```

```

if vEnemyTimer = 0
  proc AddEnemy
    rem Increase the count down timer.
    vEnemyTimer = vEnemyTimer + 200 + rnd(100)
  endif

rem Update all active enemies.
for i = 0 to MAX_ENEMIES - 1
  rem Is enemy active?
  if vEnemies[i].active
    rem Decrease the enemy's count down timer.
    vEnemies[i].timer = vEnemies[i].timer - 1
    rem Keep moving (or standing still) as long as the timer is
    rem greater than 0.
    if vEnemies[i].timer > 0
      if not vEnemies[i].pause
        dir = vEnemies[i].direction
        rem Moving left?
        if dir = LEFT
          rem Move left.
          vEnemies[i].x = vEnemies[i].x - vEnemies[i].speed
          rem Turn enemy around if it has left the game field.
          if vEnemies[i].x < 0
            vEnemies[i].x = 0
            vEnemies[i].direction = RIGHT
          endif
          rem Moving right?
          elseif dir = RIGHT
            vEnemies[i].x = vEnemies[i].x + vEnemies[i].speed
            if vEnemies[i].x > 640 - width(ENEMY_IMAGE)
              vEnemies[i].x = 640 - width(ENEMY_IMAGE)
              vEnemies[i].direction = LEFT
            endif
          endif
        endif
      endif
    endif
  endif
endfor

```

```

        endif
    rem Moving up?
    elseif dir = UP
        vEnemies[i].y = vEnemies[i].y - vEnemies[i].speed
        if vEnemies[i].y < 0
            vEnemies[i].y = 0
            vEnemies[i].direction = DOWN
        endif
    rem Moving down.
    else
        vEnemies[i].y = vEnemies[i].y + vEnemies[i].speed
        if vEnemies[i].y > 480 - height(ENEMY_IMAGE)
            vEnemies[i].y = 480 - height(ENEMY_IMAGE)
            vEnemies[i].direction = UP
        endif
    endif
endif
endif
rem The timer is 0, and it's time for a new action.
else
    rem Should enemy stand still for a while?
    if rnd(4) = 0 and not vEnemies[i].pause
        vEnemies[i].pause = true
    rem Give the enemy a new direction.
    else
        vEnemies[i].pause = false
        vEnemies[i].direction = rnd(4)
    endif
    rem Increase the count down timer for the new action.
    vEnemies[i].timer = 64 + rnd(64)
endif
endif
next

```

```

endproc

rem
=====
=====
rem Draw enemies.
rem
=====
=====
procedure DrawEnemies()
    set color 255, 255, 255
    for i = 0 to MAX_ENEMIES - 1
        if vEnemies[i].active
            draw image ENEMY_IMAGE, vEnemies[i].x, vEnemies[i].y,
vEnemies[i].direction
        endif
    next
endproc

```

This code is provided by Mopz the Master Mind behind NaaLaa

The introduction are made by now so lets start with the first line of code that you guys already know what it means.

I will explain again all lines of code, but the next tutorial will focus only in the new lines of code and what they mean so don't get lazy and start learning and studying.

```
rem Enemies with completely random movements.
```

This line is a comment, in Naalaa to comment something you must start with the word *REM*, and then your comment, this is a good practice because if some time later if you want to update your code or other they can ready and know what that chunk of code means and do.

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

This is mandatory if you don't want to declare key constants, or know all the keys code, so to get rid of that extra work we import the *Keycodes.lib* to do the job for us, next thing is the *Speed.lib*, the speed library is the responsible to keep constants frame rate (FPS), this will ensure that the velocity is all the same in different machines (old or new).

```
constant:  
LEFT      0  
RIGHT     1  
UP        2  
DOWN      3  
UNDEFINED 4
```

This six lines are known as Variables also known as *Constants*, this means that this variables are declared at the beginning of any code and can't be changed by the code in the entire program or game as

you wish, the only way to change its value is by hand and by the programmer.

As all of you can see they are declaring *LEFT* as *0*, this way when you need to indicate the left you only have to use the word declared as *LEFT* and not the zero, if not declared we must use *0* which is more difficult to read than *LEFT*, or not.

The other variables are the same thing, so be curious and try them out, and don't forget that to learn you must try it out, we learn with the mistakes made.

```
MAX_ENEMIES 16
```

This is another constant variable that holds the number of maximum enemies that the game will have, this will hold *16*.

```
ENEMY_IMAGE 1
```

Another constant that will hold the number one, as you can see instead of calling *1* every time we need to show or load an image we call *ENEMY_IMAGE*, isn't that more intuitive?

```
visible:
```

In case that you guys don't remember this built-in word means that all variables that are declared after her will be accessible to all the functions sub-routines and stuff, thing that is a global variable, so if

we need a global variable then in NaaLaa we must declare this nifty word called *visible*.

```
vEnemies?[MAX_ENEMIES]
```

Remember in NaaLaa how to declare an Object?

Well it's easy, to do that just get a nice name and one that is related to the object in question and just add in front of it a question mark (?), and viola the object is declared, and since this is after visible it will be accessible to all program, don't forget that.

But this object has a strange thing in front of them, what is that? Where did I just see that capital letters name.....??? Hum...😊

Well it's the number of max enemies that we declare earlier as a constant, remember? And those blocks...hum or something, all together, what are those things man...?

Well don't worry they are no monsters that means that is an object array and will contain all enemies objects, in this case *16*, the other way of doing it is:

```
vEnemies?[16]
```

Continuing next we are going to declare a timer, not a watch to put in your wrist...lol, just a simple timer to get enemies on the screen when it reaches 0, let's do it.

```
vEnemyTimer
```

Don't forget that this is a global variable either.

But I'm tired of being seen to all people, how can I hide?

Well it was nice to see you, I will put you hidden so no one even me can see you.

And how do you do that?

With a magic trick?

No silly we are working with NaaLaa, and to do that just use the next word.

```
hidden:
```

Ok, now that the guy is hidden and don't do more silly questions let's start coding and build our self's a nice and pretty window, and to do that just type the follow words in NED.

But that guy wasn't already hidden?

No NED is not a guy is the Editor for NaaLaa programming language.

Ok, start coding the window.

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

As you already know we will set a window at the screen of yours computer with the height of 480 and a width with 640 pixels, this was already covered in the previous, so if you still don't know, go back and see it again, there is no shame on that.

```
set redraw off
```

Remember? Set redraw off if you are not in console mode, if you are don't bother with this line of code, if not set redraw off, we only need to set redraw on in the end of each game loop, that's when we need to show all the images and stuff on the screen.

So the next thing to do is loading the enemies, a green tank. Let's do it then.

```
load image ENEMY_IMAGE, "data/green_tank.bmp"
```

The name of the command says everything, what what's that? Loading an image that previous we called as *ENEMY_IMAGE*, and then the path of the image is self, in this case the image is located at data folder with the name *green_tank* with the extension *bmp* (Bitmap).

Now the image is loaded, what to do from here? Well is simple lets call the image grid.

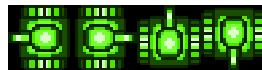
```
set image grid ENEMY_IMAGE, 4, 1
```

If you guys were with attention in the previous tutorial you now by now that this is a grid that have four images, one to the left, right, up and down, and all of this in one single .bmp file, so to have individual images we create a grid with one row and four columns to split the image in four separated images, get it?

```
set image colorkey ENEMY_IMAGE, 0, 0, 0
```

Now, the next line will tell Naalaa to get rid of the black back color of the Green tank, unless you like a green tank with a black background, it's your choice but I think that is a little bit weird.

So to get rid of that and have a transparent background to do so we put the same color as the background, and this is black, `RGB(0,0,0)`.



Tank Image

Get it now? Green tank with four images in a single file and a black background.

```
proc InitEnemies
```

This is the first time that appears on the code, no, no, no, this is the second time that we initialize a procedure, the first time was to initialize the 60 frames procedure, this one is to initialize the

enemies, we will talk about that more ahead, but every time you build a function or a procedure you have to initialize her first.

Next we must construct the main loop of our game, let's do it.

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

Now, that we have our main loop let's start coding for real, don't forget that this will run until the player hits the escape key,

Now inside this game loop will be our draw enemies in the screen, because we want them there and if don't hit the escape key, so we create a procedure named *UpdateEnemies* and *DrawEnemies*.

```
proc UpdateEnemies
```

This call to the procedure is inside the game loop, but the *InitEnemies* is not, this is because you need to initialize them once but is needed constant update them or else the changes in them won't work.

Let's put some color on the screen with the set color and clear the screen after that procedure inside the game loop.

```
rem Clear screen.  
set color 0, 16, 32  
cls
```

Simple hã!!!! *C/s* means *c*lear *s*creen, and the color is easy, and this inside the game loop why?

Well that's easy, this is needed inside the game loop because we want that color thru the all game and *c/s* to update the screen, this is, imagine that an enemy is killed, well to get rid of them we must clear the previous screen and update to a new one with the enemies less the one you killed.

Going on

```
rem Copy graphics to window and call SPD_HoldFrame to keep a  
rem constant fps.  
redraw  
proc SPD_HoldFrame 50
```

Need no introduction, the comment says it all, now that we put everything inside the game loop we need to show in the screen, and for that we call *redraw* and to maintain the constant frame rate we need the procedure *SPD_HoldFrame* which is a procedure that belongs to the *Speed.Lib*, remember. Ok, ok I know what about that 50 value? Well the 50 value is the frames per second that the game will run, try to move the value and see what happens.

Now that we build the main part of the game we will see some different things, some of them new some of them old, so if you think that this was hard, prepare because this will make your brain hurt.

This is the big difference between this and the last tutorial, the first tutorial was built to understand what's going on in the main loop, this one was built by parts, and these parts are procedures, this was meant to separate to conquer, more readable.

So to do that we must code this:

```
procedure InitEnemies()  
endproc
```

Piece of cake isn't? Well so far is, now this cake is empty let's cook it.

```
rem Loop through all enemies and clear the fields.  
for i = 0 to MAX_ENEMIES - 1
```

Another new command for you guys is a for loop, basically this loops from 0 to *MAX_ENEMIES* this means that the variable *i* will hold 16 values, because 0 to 16 are 17 enemies but if you do it less 1 then it will be 16 get it?

If you don't get it then I advise you to read some stuff about loops, and not only the *for*, read about the basics of programming.

Now inside the loop put the following lines of code.

```
rem If 'active' is true, the enemy is active and moving around  
rem on the screen.  
vEnemies[i].active = false
```

Remember the Object array? Well don't worry if you don't because now we are going to use it, and to start we will tell the computer that all of the enemies are not active, they are initialized but inactive, sleeping if you want

```
rem Direction LEFT/RIGHT/UP/DOWN of enemy.  
vEnemies[i].direction = UNDEFINED
```

.

Next we code the direction and set it as *UNDEFINED*, at this part the direction is not important because they are all inactive, but we have to initialize the object.

Next...

```
vEnemies[i].x = 0  
vEnemies[i].y = 0
```

How yes this I know how, how, it's the direction of the enemies,

You are absolutely right, those are the coordinates of the tank enemy, the *x* and *y* positions.

```
rem Speed of enemy.  
vEnemies[i].speed = 2
```

This line tells us that the tanks will have a speed of 2 pixels. No need to explain more.

```
rem If this flag is true, the enemy is standing still.  
vEnemies[i].pause = false
```

Next thing to do is tell the program that the enemy is not in pause, they are moving around, but when needed we pause them, but for now they are walking.

```
rem This timer is used to control for how the enemy should be  
rem moving in a certain direction or standing still.  
vEnemies[i].timer = 0
```

See the comments on green? They tell everything that is needed to know about our timer, but before we give him a counter we initialize with zero, so let us move forward and add a few more lines in this procedure.

```
next
```

This is easy, it tells the for loop to go to the next number until it reaches 16, then when it reaches it goes to the next line of code, and in this one we will give our timer some time.

```
vEnemyTimer = 200  
endproc
```

And there is it, the timer for our enemy, by now is all you need to know about, next thing to do is checking if the procedure is closed and to do that we write *endproc*.

```
rem Add a new enemy on a random side of the game field.  
rem  
=====
```

```
procedure AddEnemy()
```

Now that we have initialized the enemies we must add them to the screen with a random generated number, and we must put them on the screen coming from several places of the screen (Left, Right, Up and Down), this way is cooler than the tank coming always from the same place, isn't it?

```
rem Find an empty spot in the enemy array.  
index = -1  
for i = 0 to MAX_ENEMIES - 1  
  rem Is this enemy free to use?  
  if not vEnemies[i].active  
    index = i  
    break  
  endif  
next
```

Well in the first line we going to give our index the -1 position, we do it to find an empty place inside of the array, next line is the same

as the above, next we will ask if the enemies are not active, and if they are not then our index will be equal to the I variable that will have the values from 0 to 16, and then we break the code and jump, the other two lines you guys know what them mean by now, so moving on.

```
rem If index hasn't been set, no empty spot could be found.  
if index < 0 then return
```

And after jumping we found this line of code and if we find that the enemy index is less than zero we return it.

```
rem Activate enemy.  
vEnemies[i].active = true
```

In case of the enemies are all inactive we put him all active and ready to pump, easy right? Well if you guys have some troubles be patient and try it and try it, eventually you will get the stuff going.

```
rem Set enemy direction to LEFT, RIGHT, UP or DOWN by random.  
vEnemies[i].direction = rnd(4)
```

Well those guys don't know where to go so we must tell them to go where the generated random number tells him to go. So we have four parts like I said earlier we will `rnd(4)`. Next.

```
if vEnemies[i].direction = LEFT
    vEnemies[i].x = 640
    vEnemies[i].y = rnd(480 - height(ENEMY_IMAGE))
```

Ok, now that we gave them a place to go we will check what direction the tank is faced, first we check if he is left and if so then we will tell him that is position will be at the position **640**, and his y position will be somewhere in the **480** position less the height of our image.

```
elseif vEnemies[i].direction = RIGHT
    vEnemies[i].x = -width(ENEMY_IMAGE)
    vEnemies[i].y = rnd(480 - height(ENEMY_IMAGE))
elseif vEnemies[i].direction = UP
    vEnemies[i].y = 480
    vEnemies[i].x = rnd(640 -
width(ENEMY_IMAGE))
else
    vEnemies[i].y = -height(ENEMY_IMAGE)
    vEnemies[i].x = rnd(640 - width(ENEMY_IMAGE))
endif
```

The code above is the rest of the positions, Right, Up and down, the last 4 lines of code don't have the position, but if all the others are false then the down position is the last, and viola that it for the positions of all enemies in the screen.

```
rem The enemy will move 24 + 0..63 steps before doing
anything
rem else.
vEnemies[i].timer = 24 + rnd(64)

vEnemies[i].pause = false
endproc
```

Now to finish this procedure we only have to give our timer some numbers, so we will put them on moving 24 +0 to 63 steps until they do anything else, like in the comment, and of course they must be moving so no lazy tanks here just put pause false and close the procedure.

For this program to be ready to pump we need more two procedures, one that draws to the screen and one to update them, so we will start by coding the drawing part, this is because it's easier and few lines of code, and most of them you already know.

Do it..

```

procedure DrawEnemies()
  set color 255, 255, 255
  for i = 0 to MAX_ENEMIES - 1
    if vEnemies[i].active
      draw image ENEMY_IMAGE, vEnemies[i].x, vEnemies[i].y,
vEnemies[i].direction
    endif
  next
endproc

```

Here is it, great we are going to cut her now. First line blablabla, easy, next set the color to.....white that's right, the for sentence, easy, are they active???? Yes we make them active remember?

So if so draw them in the screen, attack.... And don't forget to put the direction of them and the coordinates too. See the *draw image* command if you don't understand it.

Now the ruff part, the update procedure, ruff because they will be not static, they will move back and forward and make a pause, well run it and see what they do, hand to the work.

```

procedure UpdateEnemies()
  rem Count down timer.
  vEnemyTimer = vEnemyTimer - 1
  rem Add a new enemy if timer is 0.
  if vEnemyTimer = 0
    proc AddEnemy
      rem Increase the count down timer.

```

```
vEnemyTimer = vEnemyTimer + 200 + rnd(100)
endif
```

The procedure is quite big so I will put chunks of code each time to try to explain them.

The first line tell the timer to decrease by one, remember that we tell him to store the value 20, so the first time the timer will decrease by one to 19 until reaches zero and every time it reaches 0 they a new enemy is added to the screen, one by one with the *procedure AddEnemy*, then when it adds one the timer increases again until reaches zero and so on.

```
rem Update all active enemies.
for i = 0 to MAX_ENEMIES - 1
  rem Is enemy active?
  if vEnemies[i].active
    rem Decrease the enemy's count down timer.
    vEnemies[i].timer = vEnemies[i].timer - 1
    rem Keep moving (or standing still) as long as the timer is
    rem greater than 0.
    if vEnemies[i].timer > 0
      if not vEnemies[i].pause
        dir = vEnemies[i].direction
```

Now that we have some in the screen we must update them again, and for 0 to 15 or 0 to 16-1, and if they are active on the screen

then the timer will decrease by one and if the timer is greater than 0 then they will continue to move.

```
rem Moving left?
    if dir = LEFT
        rem Move left.
        vEnemies[i].x = vEnemies[i].x - vEnemies[i].speed
        rem Turn enemy around if it has left the game
field.
        if vEnemies[i].x < 0
            vEnemies[i].x = 0
            vEnemies[i].direction = RIGHT
        endif
        rem Moving right?
    elseif dir = RIGHT
        vEnemies[i].x = vEnemies[i].x +
vEnemies[i].speed
        if vEnemies[i].x > 640 - width(ENEMY_IMAGE)
            vEnemies[i].x = 640 - width(ENEMY_IMAGE)
            vEnemies[i].direction = LEFT
        endif
        rem Moving up?
    elseif dir = UP
        vEnemies[i].y = vEnemies[i].y -
vEnemies[i].speed
        if vEnemies[i].y < 0
            vEnemies[i].y = 0
            vEnemies[i].direction = DOWN
        endif
```

```

        rem Moving down.
    else
        vEnemies[i].y = vEnemies[i].y +
vEnemies[i].speed
        if vEnemies[i].y > 480 - height(ENEMY_IMAGE)
            vEnemies[i].y = 480 - height(ENEMY_IMAGE)
            vEnemies[i].direction = UP
        endif
    endif
endif
rem The timer is 0, and it's time for a new action.
else
    rem Should enemy stand still for a while?
    if rnd(4) = 0 and not vEnemies[i].pause
        vEnemies[i].pause = true
        rem Give the enemy a new direction.
    else
        vEnemies[i].pause = false
        vEnemies[i].direction = rnd(4)
    endif
    rem Increase the count down timer for the new action.
    vEnemies[i].timer = 64 + rnd(64)
endif
endif
next
endproc

```

Well now that we know and verify that they are moving we must check what position they are faced, and to do so we create a variable with the name *dir*, and this will hold the object direction, and next we will check if his faced left, and if so then we move them to the left, and as you know when the x position decreases then we use $vEnemies[i].x = vEnemies[i].x - vEnemies[i].speed$, and it will move left. This should be clear for you guys, but if don't go back to the first and try it again, well let us check the other directions too, and when it is less than zero then the x will be zero.

Then move to other direction, in our case we move right and when it is less than zero then will move again and again, the code by now should be clear to read.

```
rem The timer is 0, and it's time for a new action.  
    else  
        rem Should enemy stand still for a while?  
        if rnd(4) = 0 and not vEnemies[i].pause  
            vEnemies[i].pause = true
```

To clarify these last pieces of code, what they do is if the random numbers will be equal to zero and the enemies are moving then we must pause them a bit.

Next thing to do is giving them a new direction and to do that we must add the following code to finish.

```
rem Give the enemy a new direction.
    else
        vEnemies[i].pause = false
        vEnemies[i].direction = rnd(4)
    endif
    rem Increase the count down timer for the new action.
    vEnemies[i].timer = 64 + rnd(64)
endif
endif
next
endproc
```

Well nothing new, there are lines of code alike thru the code, so to clear for the last time, what this do is:

Put the enemies off pause, them give a new direction thru random numbers, next gives the timer a new action, and viola the enemies will appear one by one in the screen until reaches **16** enemies tanks.

By now you should have a screen like the one I have in the first page, if so congratulations if not well nothing is lost and start all over again.

Please rate this if you like it.

This is property of Mopz and NaaLaa.