App game kit mobile

Getting started/tutorial (8 april 2018)

By Rudy van Etten (pakz001)

I am using the ios version on a ipad 2018.

***Note**: currently there are two onscreen keyboards. The native ios keyboard does not work correctly. When in the editor on the right bottom side of the screen there is a icon for the custom screen keyboard. Use this one instead.*

***Note**: there might be some bugs with agk mobile for ios. Sometimes it may get a messed up screen. Closing the app and restarting it fixes it.*

Hello World

Choose: create new project.

Between the lines. **Do** and **loop** below the line **print(screenFPS())** add the following line.

Print("Hello World")

Press the run button and you should see the hello world line been printed/drawn to the screen.

Note: *to exit a program hold your finger in the top right corner of the screen for around 3 seconds.*

Drawing a line

To draw a line to the screen you can use the **Drawline** command. It takes a number of inputs.

***Drawline**(x1,y1,x2,y2,red(0..255),green(0..255),blue(0,255*))

Drawline(0,0,50,0,255,0,0)

The above line draws a line from coordinates 0,0 to coordinates 50,0 using the red color.

Create a new project and below the line **Print(screenFPS())** add the **Drawline** command and experiment with it.

Different resolutions

You might notice when you draw something that you have not told app game kit what resolution you want. Let us learn how to do this now.

SetVirtualResolution(Width,Height)

Width for example could be 320 and Height could be 480. You can place this command somewhere in your code when you set up your project.(top of the code)
When you have set the resolution you will be able to use drawing commands like **Drawline** with the resolution you have given.

For loops

A **For** loop is a way to do a lot of useful things with. You probably wil be using it a lot.

```
For i = 0 to 10
    Print(i)
Next i
```

Above we create our **for** loop. After the **for** command we see the character i. This is a variable that wil be used to store and keep track of the numbers used by the **for** loop. In this case we tell it to count from 0 **to** 10. We print the number (i) on the screen 11 times(0 including) The **next** command followed by the i variable is used to tell the program back the next **for**. When the i is in its last value the code continues.

For loops and integer arrays

A array is a piece of memory that can store information. Here I wil show how to create a Integer array and then print the data to the screen.

Note*: an Integer is a number. 123 is a integer value. “Abc” is not an integer value. 123.0 is also not an integer value.*

```
Dim test[10] As Integer

Test[0] = 10
Test[10] = 15
```

Place the lines above in an new project above the **Do** command. (The part where we update and draw things on the screen.

Place the lines below in the main part of the program. (After the Do command and below the print screenFPS part)

```
For i = 0 to 10
   Print(Test[i])
```

Next

When you now run the code you should see a series of numbers on the screen. One should be a 10 and a series of 0's and a 15.

Globals

A global command is used to tell agk that a variable can be written and read from anywhere in your code.

Global abc **As Integer**

You usually create your global things at the top of your code.

abc can now be used to store a number anywhere and be read anywhere.

If and **then**

An **if** command is one of the most valuable commands. It can be used to test situations like if an variable is of a certain value. The **then** command is used to do something after an condition like **if**.

If a = 10 **then** a = 5

Above we tell agk to check if the variable a is 10 and if so change it to 5.

Note*: Agk is short for App Game Kit*

Random

The command Random is used to create a random number.

Local a **As Integer**
a = **Random**(0,10)
Print(a)

The code above can be put in the **do loop** section of a new project. We create a **Integer** variable named a and we put a number between 0 and 10 in it and **print** it on the screen.

User touching the screen (touch up)

The command **GetPointerPressed()** is used to see if the user touched the screen. 1 finger on the screen and the off again.

```
If GetPointerPressed() = 1
   Print("finger released")
Endif
```

If you put the code above in a empty project below the **print(screenFPS**) line and run it then on the screen every time you touched the screen a message wil be printed,

User finger on screen and coordinates (touch down x and y)

The command **GetPointerState**() is used to see when the user his finger is on the screen. The commands **GetPointerX**() and **GetPointerY**() contain the last x and y coordinates where the finger was located on the screen.

```
Print(GetPointerState())
Print(GetPointerX())
Print(GetPointerY())
```

Place the code above in a new project below the **print(screenFPS**) line and run and touch the screen to see the touch commands function.

Gosub (goto subroutine)

Gosub is a command that tells the program to jump to a particular part in you code. This part in the code needs a label and a return part. When the return command word is read then the program returns to the code where the gosub was called.

```
Gosub test
```

Place the line above in a new project below the **print(screenFPS)** line.

```
Test:
Print("test")
Return
```

Place the 3 lines above at the bottom of the empty project. Make sure it is not placed inside other parts of code like inside **do loop** sections. Notice the label Test has a : character behind it.

Run the code and you should see the word test printed on the screen.

Create and display text on the screen.

Createtext(number,text) is a command used to create text that wil be drawn on the screen. Each number(1..x) is a text label and the text is a string$. You need the **settextposition**(number,x,y) to place it on the screen. The **settextsize**(number,size) is used to set its size.

Create a new project and in the code above where the main loop begins(the **do** command) put the following lines.

```
Createtext(1,"game over man")
```

```
SetTextSize(1,20)
SetTextPosition(1,0,50)
```

Press run and you should see the text on the screen.

Detecting finger on text

The text commands come with a feature that let it detect when the finger touched the text. See below.

```
CreateText(1,"press me")
SetTextSize(1,20)
SetTextPosition(1,0,50)
```

Place the 3 lines above above the main loop in a new project. Place the following lines inside the main loop(below the **print(screenFPS**)) line.

```
If GetPointerPressed() = 1
  Hit = GetTextHitTest(1,GetPointerX(),GetPointerY())
  If Hit = 1 then SetClearColor(Random(0,255),0,0)
  Print Hit
EndIf
```

When run you can press your finger on the text label and the you should see the screen color change.

Creating a sprite and moving it

We can create sprites with agkmobile. Here I wil show how you can create a empty sprite, put it on the middle of the screen and move it continuously from the left side of the screen to the right side.

```
Sp = CreateSprite(0)
SetSpriteY(Sp,50)
Local x As Integer
```

Put the 3 lines above here above the main loop of a empty new project. Put the 3 lines below here below the **print(screenFPS())** in the main loop.

```
SetSpriteX(Sp,x)
x=x+1
If x>100 then x=0
```

CreateSprite creates a new sprite object. The Sp variable wil contain the sprite number.
SetSpriteX and **SetSpriteY** are used to position a sprite on the screen.

Converting integer to string (**Str**)

The command **Str**() can be used to convert a integer variable into a string.

```
Local test As Integer
test = 150
Print("The number is : " + Str(test))
```

Put the 3 lines above inside the main loop of a new empty project. Below the **Print(screenFPS**()) line. When run you wil see the number printed to the screen.

Comments in the code

We can put all sorts of text inside our code file without it causing errors. This to explain things or even to disable certain code. We can add a **Rem** or **//** or ‘ on the most left side of a line to tell agk that this line is comment.

The 3 lines below here are comments.

```
Rem this is a comment
// this is a comment
‘ this is a comment
```

Types (creating and printing to the screen)

Types are something we can use that can hold data. Somewhat like arrays but with types you can hold different data types as integers and strings and floats at the same time.

Put the code below in a new empty project. Above the main loop that starts at the **do** command.

```
// this is our type. It starts with the Type keyword and ends with the EndType keyword.
Type test
  X As Integer
  Y As Integer
EndType

// here we create an array as test with 5 types.
test as test[5]

// here we put random values inside the x and y variables.
For i = 1 to 5
  test[i].x = random(0,10)
  test[i].y = random(0,10)
Next i
```

Put the following lines inside the main loop. Below the **print(screenFPS**()) line.

```
// here we print the type variables contents on the screen
For i=1 to 5
  Local a As String
  a = test[i].x+","+test[i].y
  Print(a)
Next i
```

When you run the project you should see a series of numbers on the screen.

Functions

Functions are sections of code that can take optional input and optionally return data.

```
Function test()
  Print("test")
Endfunction
```

The 3 lines above show how a **function** is set up. The code above can be called or executed from anywhere in you program by using its name test()

```
Function test(a As Integer)
  Print(a)
EndFunction
```

The code above shows a **function** that takes 1 input. In this case a number which is used to **print** to the screen when the **function** is executed.

```
Function test()
EndFunction "hello"

Print(test())
```

The code above shows you how to create a **function** that returns data. In this case a **string**. We could put a variable there in stead also. The **print** line below that function shows you how a **function** like that could be used.

Drawing on a image

When we look at a game like space invaders then we can see that when a alien bullet hits a bunker this bunker gets damaged. In agk we could create a image with the drawing of a bunker and slowly erase it piece by piece when it gets hit.

We can use the following code to create and render to an image.

```
// create image 1 with width and height of 32
Createrenderimage(1,32,32,0,0)
// here we tell agk to use the drawing commands on image 1
Setrendertoimage(1,0)
// here we draw two lines.
Drawline(0,0,32,32,255,0,0)
Drawline(0,32,32,0,255,0,0)
// tell agk to draw to the regular screen
Setrendertoscreen()
// create our new sprite with our new image
Createsprite(1,1)
```

The section of code above should be placed in a new project above the main loop. The next few lines below should be placed inside the loop below the print(screenFPS()) line.

```
// put sprite 1 on a random location of the screen.
Setspritex(1,random(0,100))
Setspritey(1,random(0,100))
```

When everything is right then after you press run you should see a sprite being drawn on the screen which is jumping around.

Creating a 3d box and rotating it

To create a 3d box we can use the command **createobjectbox**(width,height,depth)

The line below should be placed above and outside the main loop. This line creates a 3d box. The object can be found/adressed with its name Box.

Box = **createobjectbox**(2,2,2)

The line below is used to rotate a object locally around its y axis. Place this line in the empty new project where the above lines also should be below the **print** fps line.

Rotateobjectlocaly(Box,2)

The number 2 is the amount rotated.

If all went wel and when pressed run you should see a box rotating on the screen.

Shooting a sprite after touch

Shooting is something that in videogames happens a lot. Here we are going to show how to shoot a bullet or laser from the bottom of the screen. This sprite we create for that travels from the bottom to the top of the screen after a screen touch.

Create a new empty project and look for the line in the code that says **do**. Above this line add the 5 lines below here.

```
Rem Create our bullet or laser sprite.
Sp = createsprite(0)
Setspritesize(sp,4,8)
Setspriteposition(sp,48,100)
Shot = 0
```

Inside the main loop(below the **do** command and below the **print(screenFPS**()) line) add these following lines.

```
Rem if not shot yet and touched the screen
If getpointerpressed()=1 and Shot=0
  Shot = 1
Endif

Rem if bullet is active(traveling)
If Shot = 1
  Setspritey(sp,getspritey(sp)-3)
Endif

Rem if sprite if above top of screen
```

```
If getspritey(sp) < -10
  Rem disable moving the sprite and restore default position
  Shot = 0
  Setspritey(sp,100)
Endif
```

If you are here and have pressed the run button and if everything went right you should be able to shoot 1 sprite at a time with a touch.

Sprite collisions

Finding when a sprite collides with another sprite is not difficult at all. We can use the command **getspritecollision**(sprite1,sprite2) for this.

Create a new empty project and type or copy the following 6 lines below here above the **do** command and below the **usenewdefaultfonts**(1) line.

```
Sp1 = createsprite(0)
Sp2 = createsprite(0)
Setspritesize(Sp1,10,10)
Setspritesize(Sp2,15,10)
Setspriteposition(Sp2,50,50)
A as string
```

Type or copy and paste the following 9 lines in the main loop of the project. Below the **print(screenFPS**()) line.

```
Print(“move finger on screen to move sprite”)
Setspritex(Sp1,getpointerx())
Setspritey(Sp1,getpointery())
If getspritecollision(Sp1,Sp2) = 1
  A = “sprites colliding”
Else
  A = “no sprites colliding”
Endif
Print(A)
```

When you run the program you can move you finger on the screen and 1 sprite wil stay under your finger. When the 2 sprites that are on the screen touch there wil be a collision.

Arrays (one and multidimensional) and length

Place the following 3 lines in a new empty project below the **usenewdefaultfonts**(1) line.

```
Dim test[] as integer = [1,2,3,4]
Dim test2[1,2] as integer
Dim test3[1,2,3] as integer
```

Place the following **print** lines in the main loop. Below the **print(screenFPS**()) line.

```
Print(“array test length”)
Print(test.length)
```

Print(“array test2 length”)
Print(test2.length)
Print(“array test2[0] length”]
Print(test2[0].length)
Print(“array test3 length”)
Print(test3.length)
Print(“array test3[0] length”)
Print(test3[0].length)
Print(“array test3[0,0] length”)
Print(test3[0,0].length)

When you run the program then you wil see the **length** information of the created arrays on the screen. Change the numbers in the **dim** lines to see them change when run.

Note*: only single dimension arrays can have default values.*

Array **insert** and **remove**

For algorithms like the a* (astar) pathfinding or floodfilling it is useful to be able to remove and insert data into a list. Agk makes this possible with the **insert** and **remove** commands for arrays.

Type or copy and paste the following 9 lines in a new empty project below the **usenewdefaultfonts**(1) line.

Dim test[] **as integer** = [1,2,3,4,5]
Rem insert a new item with value 6 at end of array
test.**insert**(6)
Rem remove the first item from the array
test.**remove**(0)
Rem remove the last item from the list
test.**remove**()
Rem insert value 10 at array position 1
test.**insert**(10,1)

Type or place the following line in the main loop below the **print(screenFPS**()) line.

For i = 0 **to** test.**length**
  Print(test[i])
Next i

If you run the code you should see a series of numbers. 2 10 3 4 5

Insert and types and arrays

Place the following lines (to and with **endfunction** a) in a new empty project. Place these below the **usenewdefaultfonts**(1)

Rem *create a type called test*
Type test
  X **as integer**
  Y **as integer**

```
Endtype

Rem make a empty array called test2 containing the test type
Test2 as test[]

Rem insert 2 new test types into the test2 array
Test2.insert(newtest())
Test2.insert(newtest())
Test2[0].X = 10
Test2[1].X = 20

Rem this function creates a new test type and returns it.
Function newtest()
   A as test
Endfunction A
```

Place the following 3 lines in the main loop below the **print(screenFPS**()) line.

```
For i = 0 to Test2.length
   Print(Test2[i].X)
Next i
```

If you run the code then you should see two numbers below the frames per second number. These numbers, the x we set should be 10 and 20.

Modifying a array using a **function** (by reference)

Create a new empty project and below the **usenewdefaultfonts**(1) line add the following lines.

```
Dim test[] as integer = [1,2,3,4,5]

Rem here we create a function that can modify a array. Note the ref word.
Function inc_array(a ref as integer[])
   For i = 0 to a.length
      a[i] = a[i] + 1
   Next i
Endfunction

Rem here we move the array into the function to have it be modified.
inc_array(test)
```

Add the following 4 lines to the main loop. Below the **do** and **print(screenFPS**()) lines.

```
Rem print the contents of our test array to the screen
For i = 0 to test.length
   Print(test[i])
Next i
```

If everything went right then the default values with which the test array were created should be 1 value higher then before.

Arrays and tilemap using default values.

A tilemap is a piece of memory that contains information about the makeup of our screen. With it we draw tiles on the screen. It has a width and a height. Each location has a number telling which tile we should draw. Location 0,0 in the tilemap could be the value of 1 and when we have a tilemap drawn on the screen from top left going to the bottom right in typewriter style means the top most tile is tile number 1. Each tile we have could be a part of a drawing that together form a image of for instance a millitary base.

Below here I am going to show you how to draw a small map on the screen.

Create a new empty project and type or paste the lines below here until the **endfunction** underneath the **usenewdefaultfonts**(1) line.

```
Rem we will create a map with width and height of 5
Global mapwidth=5
Global mapheight=5
Rem the width and height of the crosses here
Global tilewidth=16
Global tileheight=10
Rem here we create our map array. Note the y is first
Dim map[mapheight,mapwidth] as integer
Rem here we create our map tiles. 1 is a cross, 0 is nothing.
map[0] = [0,1,1,1,1,0]
map[1] = [1,1,0,0,1,1]
map[2] = [1,0,0,0,0,1]
map[3] = [1,0,0,0,0,1]
map[4] = [1,1,0,0,1,1]
map[5] = [0,1,1,1,1,0]
Rem our drawmap function
Function drawmap()
   for y=0 to mapheight
   for x=0 to mapwidth
      Rem if inside our array we read a 1 value the. Draw our tile.
      If map[y,x] = 1
         Local x2 as integer
         Local y2 as integer
         x2 = x * tilewidth
         y2 = y * tileheight
         Drawline(x2,y2,x2+tilewidth,y2+tileheight,255,0,0)
         Drawline(x2+tilewidth,y2,x2,y2+tileheight,255,0,0)
      Endif
   Next x
   Next y
Endfunction
```

Add the line below here into the main loop of the project. Below the **print(screenFPS**())

```
drawmap()
```

If everything went right then you should see a series of crosses drawn to the screen. If you modify the array(0 and 1's) then the crosses will change also.

Bouncing sprite

Gravity in games is something you might see a lot in the games you play. A simple way of creating gravity for 2d games is shown below here.

Create a new empty project and below the line **usenewdefaultfonts**(1) place the following lines.

```
Rem how fast we bounce upwards
Force# = 6.0
Y = 80 // location of the sprite
My# = Force# // set the increment
Direction = 1 // 1=upwards, 2=downwards
Sp = createsprite(0)
```

Place the following lines in the main loop. Below the **do** command and below the **print(screenFPS())**

```
Rem position the sprite
Setspriteposition(Sp,50,Y)

Rem here we bounce the sprite
If Direction=1 // going up
   Y=Y-My#
   My#=My#-1
   Rem if there is no more upward force then change direction
   If My#<0 then Direction=0
Else // going down
   Y=Y+My#
   My#=My#+1
   Rem if we hit the ground
   If Y>80
      Y=80 // align on the ground
      My#=Force# // set new upward force
      Direction=1 // change direction
   Endif
Endif
```

If everything went right then you will see a sprite bouncing on the screen. It goes from the bottom of the screen to the top and back again. It keeps bouncing forever until you close the program by holding your finger on the top right location of the screen for a couple of seconds.

Drawsprite and tilemap

Drawsprite draws the image of a sprite underneath its current location. You can draw a tilemap with this method. Create a sprite for each unique tile and while you are drawing your map tile. Position the relevant sprite on that location and draw it. This way you do not need a sprite for each tile in the map.

Place the lines below here in a new empty project. Place these lines below the **usenewdefaultfonts**(1) line.

```
Sp1=createsprite(0)
Sp2=createsprite(0)
```

```
Setspritesize(sp1,16,16)
Setspritesize(sp2,16,16)
Setspritecolor(sp1,255,0,0,255)
Setspritecolor(sp2,0,255,0,255)

Rem create our array with the tile contents
Dim Map[5,5] as integer
Map[0]=[1,1,1,1,1,1]
Map[1]=[0,0,0,0,0,0]
Map[2]=[1,1,0,0,1,1]
Map[3]=[0,0,0,0,0,0]
Map[4]=[1,1,1,1,1,1]
Map[5]=[1,0,1,0,1,1]
```

Place these lines in the main loop of our new empty project. The loop starts at the line that has the **do** command on it. Place the code below here below the **print(screenFPS**()) line.

```
Rem make the sprites visible
Setspritevisible(Sp1,1)
Setspritevisible(Sp2,1)
Rem draw our tilemap
For y=0 to Map.length
For x=0 to Map[0].length
   If Map[y,x] = 1
      Rem set sprite to tile position
      Setspriteposition(Sp2,x*16,y*16)
      Drawsprite(Sp2) // draw the sprite to the screen
   Endif
   If Map[y,x] = 0
      Rem set sprite to tile position
      Setspriteposition(Sp1,x*16,y*16)
      Drawsprite(Sp1) // draw the sprite to the screen
   Endif
Next x
Next y
Rem make the sprites invisible
Setspritevisible(Sp1,0)
Setspritevisible(Sp2,0)
```

If everything went allright then you will see a tilemap on the screen after you pressed the run button.

Moving a sprite towards the last touched position - angle

If you want to make a game where you control a verhicle on the screen then take a look at this. Here the code creates a sprite that moves towards the last touched position on the screen in a straight line. The sprite rotates into the direction he is headed.

***Note**: the **atanfull**() command needs a value of 90 to be taken off to get it to move into the right direction.*

Place the next 7 lines in a new empty project underneath the **usenewdefaultfonts**(1) line.

```
Rem sprite x and y position
X# = 50
Y# = 50
P = createsprite(0) // create our sprite
Setspritesize(P,10,5)
Rem our variable that holds the angle
A# = 0
```

Place the next block of code in the main loop. This is below the **do** command and below the **print(screenFPS**()) line.

```
Print("touch the screen to move the sprite towards it")

Rem here we move the sprite
Rem first we get the coordinates of the center of the sprite
X2# = X#+getspritewidth(P)/2
Y2# = Y#+getspriteheight(P)/2
Rem if the distance between the center and destination is great enough
If (abs(X2#-Getpointerx())+abs(Y2#-getpointery())) > 10
   Rem get the angle to head towards
   A# = atanfull(getpointerx()-X2#,getpointery()-Y2#) - 90
   Rem update the sprite coordinates
   X# = X# + cos(A#)
   Y# = Y# + sin(A#)
   Rem update our sprite
   Setspriteposition(P,A#)
   Setspriteangle(P,A#)
Endif
```

If everything went allright and when you then press run you wil be able to move a sprite around the screen. Press anywhere to move him.

Double tapping the screen

Using the command **getmilliseconds**() we can get the time passed since the start of the program. With this command we can also detect if we pressed the screen within a certain time. Double tapping could be used for initiating a jump or shooting something.

Create a new empty project and in the top of the code velow the line usenewdefaultfonts(1) place the following lines.

```
Rem with this variable we store the time after a press.
Global taptime as integer
Rem this variable if 1 means we had a double jump.
Global doubletap as integer
Rem how long between two presses should be a double tap
Global tapdelay as integer
tapdelay = 300
```

Place the following lines inside the main loop. Below the **do** and **print(screenFPS**()) lines.

```
Print(“Touch the screen twice fast”)
Rem if we had or had no double tap
If doubletap = 1
   Print(“double press detected”)
Else
   Print(“no double touch detected”)
Endif

Rem here we check if the user touched the screen
If getpointerpressed()
   Rem if the touch is within a certain time of last touch
   If getmilliseconds() < taptime
      doubletap = 1// we detected a double tap
   Else
      Rem if there was no double tap then store time plus time
      Rem in within a double press can occur.
      taptime = getmilliseconds() + tapdelay
      doubletap = 0// only one press detected
   Endif
Endif
```

If everything went right and when you run the program then double pressing the screen gets detected.

Getspritehit() and **setspritecoloralpha**()

You probably wil want to know how to find the sprite you touched on the screen. Also making a sprite transparent is useful.

Note: *Sprites can be used as buttons and other gui and hud related imagery.*

Place the 3 lines below in a new empty project below the **usenewdefaultfonts**(1) line

```
S = createsprite(0)
Setspritesize(S,100,100)
Setspritecoloralpha(s,125) // 0(invisible)....255(not transparent)
```

Place the 5 lines below here in the main loop below the **do** and **print(screenFPS**()) lines.

```
Print(“touch the sprite(top of screen)”)
Print(hit)

If getpointerpressed()
   hit=getspritehit(getpointerx(),getpointery())
Endif
```

Jumpgame

App game kit can be used to create games. Let us make a simple game. Here we have a game where there are objects coming from the right of the screen. We can jump with our player who is on the left side of the screen. When the player hits a object the score is reset to 0.

Place the code below in a new empty project below the **usenewdefaultfonts**(1) line.

```
Dim s[1]
For i=0 to 1
   s[i]=createsprite(0)
   Setspriteposition(s[i],200,60)
Next

Dim time[1]
time[0]=getmilliseconds()+100
time[1]=getmilliseconds()+200

P=createsprite(0)
Setspritey(P,60)
Pjumptime=0
Pjumpwaittime=0
```

The line below here should be placed in the main loop. This is below the **do** and below the **print(screenFPS**()) lines.

```
Print("touch to jump")
Print("score: "+str(Score))

Score=Score+1// increase our score

Rem handle the player jump
If Pjumptime>0
   Pjumptime=Pjumptime-1
Else
   Setspritey(p,60)
   If Pjumpwaittime>0 then Pjumpwaittime=Pjumpwaitime-1
Endif

Rem if touch the screen then jump the player
If getpointerpressed() and Pjumptime=0 and Pjumpwaittime=0
   Setspritey(p,40)
   Pjumptime=30
   Pjumpwaittime=10
Endif

Rem move the sprites to the left and check collision
For i=0 to 1
   If getspritecollision(p,s[i]) then score = 0
   If getmilliseconds() > time[i]
      Setspritex(s[i],getspritex(s[i])-3)
      If getspritex(s[i]) < -10
         time[i] = getmilliseconds() + random(200,1500)
         Setspritex(s[i],200)
      Endif
```

```
    Endif
Next i
```

If everything went allright then you should have a little game. Press the screen to jump over the object that are coming your way. You could add new features to this game like graphics and music and other things.

Highest number in array.

For certain algorithms amongst others you need to know if a certain index in a array contains the highest value. Here I show you a way to find the index number containing the highest value.

***Note**: in the astar(a*) algorithm code like here is used.*

Place the following 5 lines in a new empty project below the **usenewdefaultfonts**(1) line.

```
Dim number[] as integer
Rem insert a series of numbers into array.
For i = 0 to 10
    number.insert(random(0,500))
Next
```

Place the lines below here inside the main loop. This is below the **do** command and the **print(screenFPS**()) lines.

```
Rem print the numbers from the array
For i = 0 to number.length
    Print("index: "+str(i)+" = "+ str(number[i]))
Next

Rem find the highest number index
Highest=0
Highestindex=0
For i=0 to number.length
    If number[i] > Highest
        Highest=number[i]
        Highestindex=i
    Endif
Next

Print("index: "+str(Highestindex)+" has the highest number.")
```

If everything went allright then when you run the code you should see a list in numbers on the screen. The last line tells which index has the highest number.

Screen transition effect using sprites.

This transition effect is real easy to make. You create 10x10 spites and place them on the screen so they cover everything and make them invisible. Then one by one you make them visible.

Place the lines below here in a new empty project underneath the line that reads **usenewdefaultfonts**(1)

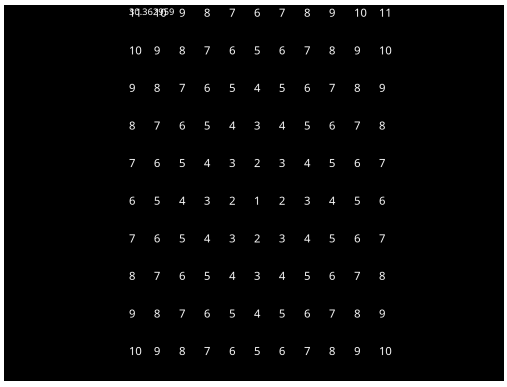
```
Rem create our sprites
Dim Spr[10,10]
For y = 0 to 10
For x = 0 to 10
   Spr[x,y] = createsprite(0)
   Setspritesize(Spr[x,y],10,10)
   Setspriteposition(Spr[x,y],x*10,y*10)
   Setspritevisible(Spr[x,y],0)
Next x
Next y
```

Place the lines below here in the main loop. Below the **do** and **print(screenFPS**()) lines.

```
Rem here we do the transition effect
Exitloop = 0
Cnt = 0
Repeat
   Rem get random position
   X = random(0,10)
   Y = random(0,10)
   Rem if position sprite is invisible then..
   If getspritevisible(Spr[X,Y]) = 0
      Setspritevisible(Spr[X,Y],1)
      Exitloop=1
   Endif
   Cnt = Cnt + 1
Until Exitloop = 1 or Cnt > 100
```

If everything went right we should see the screen transition effect as soon as we press the run in agk.

Flooding a map with distance from point (pathfinding)



Letting a enemy player find the player is pretty simple to do. We create a map and put a value of 1 at the location where the enemy is supposed to go. We then flood the map. Below is code that shows how this is done.

Place the lines below here in a new empty project below the **usenewdefaultfonts**(1) line. Include the **function** code block(ends at **endfunction**)

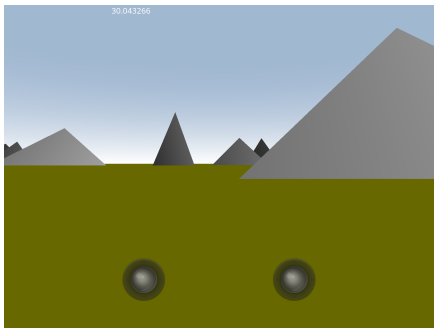
```
Dim map[10,10]
map[5,5]=1 // our destination is of value 1 ( change 5,5 for diff.)
Floodmap() // here we flood the map with distances

Rem here we create text labels with the distance values that
Rem we display on the screen.
Dim t[10,10]
For y=0 to 10
For x=0 to 10
   t[x,y]=createtext(str(map[x,y]))
   Settextposition(t[x,y],x*10,y*10)
Next x
Next y


Function Floodmap()
   Exitloop = 0
   Rem we look for a number 1 to start flooding
   Num = 1
   Rem we check above,right,bottom and left of position
   Dim mx[] as integer=[0,1,0,-1]
   Dim my[] as integer=[-1,0,1,0]
   Rem loop until the entire map has been filled with distances
   While Exitloop = 0
      Exitloop=1
      For y=0 to 10// 10 is the size of the array
      For x=0 to 10
         If map[x,y] = Num
            For ii=0 to mx.length
               x2 = x+mx[ii]
               y2 = y+my[ii]
               Rem if coordinates are outside map bounds then skip loop
               If x2<0 or y2<0 or x2>10 or y2>10 then continue
               If map[x2,y2] <> 0 then continue // skip if no 0 here
               Rem if we get here then loop once more
               Exitloop = 0
               map[x2,y2] = Num+1
            Next ii
         Endif
      Next x
      Next y
      Num=Num+1
   Endwhile
Endfunction
```

If you run the program above then you should see the screen filled with numbers. Around number 1 a higher number is shown and this goes on until the numbers reach the bounds of the screen.

Plane ground, 3d touch movement, cone mountains.



Setting up a simple 3d world with agk is not that hard. Here I created a large plane, which is basically a large flat surface. I then created a number of cones which act like mountains. These together form a classic 3d world. I added 2 buttons for moving the camera through this world.

Put the code below here in a new empty project below the **usenewdefaultfonts**(1) line.

```
Rem create a large ground
P = createobjectplane(5000,5000)
Setobjectcolor(P,100,100,0,255)
Rotateobjectlocalx(P,90)

Rem create a number of cone objects and place then on the
Rem plane at random locations.
For i = 0 to 50
   Rem height of cone
   H = random(90,190)
   Rem depth of cone
   D = random(120,500)
   Createobjectplane(i,H,D,3)
   Setobjectposition(i,random(0,5000)-2500,H/2,random(0,5000)-2500)
   Local g as integer
   g = random(0,100)+50
   Setobjectcolor(i,g,g,g,255)
Next i

Rem add touchscreen joystick(left side) look around
Addvirtualjoystick(1,15,85,20)
Rem add touchscreen joystick(right side) movement
Addvirtualjoystick(2,85,85,20)
```

Setskyboxvisible(1)
Setcamerarange(1,1,4000)

Put the code below here in the main loop. This is below the **do** command and below the **print(screenFPS**()) line.

Rem *look around with the left virtual joystick*
Rotatecameralocalx(1,**getvirtualjoystick**(1))

Rem *keep x axis within certain bounds*
If getcameraanglex(1) > 50 **then rotatecameralocalx**(1,0-**getvirtualjoysticky**(1))
If getcameraanglex(1) < 50 **then rotatecameralocalx**(1,0-**getvirtualjoysticky**(1))

Rem *look left and right.*
Rotatecameralocaly(1,**getvirtualjoystickx**(1))

Rem *keep the camera upright*
Setcamerarotation(1,**getcameraanglex**(1),**getcameraangley**(1),0)

Rem *move around with the right virtual joystick*
Movecameralocalz(1,0-**getvirtualjoysticky**(2))
Movecameralocalx(1,0-**getvirtualjoystickx**(2))
Rem *keep the camera on the same height above the ground.*
Setcameraposition(1,**getcamerax**(1),**getcameraz**(1))

If everything went right and if you run the program then you wil be able to move through a primitive 3d world using onscreen controls. Experiment!

A chunk system example

In the game minecraft there are really large worlds. There is no way that a computer can draw the entire map at once. So people talk about chunks that solves this. A chunk is like a piece of a puzzle. All the pieces connect to form a picture. In a game you would have those chunks around you forming the game map. When you move chunks that get to far away from you get removed and new chunks that get in range get created.
Below here there is a example of a crude chunk system. The example is 2d but it might show you enough for you to learn how something like this works. I based it on code that I use in my voxel world that I was working on.

First lets start with adding the first code. Place this in a new empty project below the line **usenewdefaultfonts**(1)

```
Type chunk
    Rem chunk x and y
    x as integer
    y as integer
    Rem location on the screen
    sx as integer
```

```
    sy as integer
Endtype
Rem create a list where we are going to
Rem put the types(chunks) in.
Global chunklist as chunk[]

Rem width of the chunk and height of the chunk
Global chunkwidth = 16
Global chunkheight = 16
Rem position on the map we are on.
Global px = 50
Global py = 50

Rem here we create text that we use to display the
Rem chunk locations on the screen.
Global t as integer[4,4]
For y=0 to 4
For x=0 to 4
    t[x,y] = createtext("0")
    Rem place in the center of the screen
    Settextposition(t[4,4],x*chunkwidth+20,y*chunkheight+20)
Next x
Next y
```

Next we are going to have to add code into the main loop of our default empty project. This is below the **do** command and below the **print(screenfps**()) line.

```
Print("touch the screen to scroll map")

Rem here we read if the use touches the screen.
Rem we change the position of the player.
If getpointerstate()
    If getpointerx()<50// if screen touched on the left side
        px=px-chunkwidth
    Else
        px=px+chunkwidth
    Endif
    If getpointery()<50
            py=py-chunkheight
        Else
            py=py+chunkheight
    Endif
Endif

Rem these are function calls.
updatechunks()
drawchunks()
```

Below here we are going to place the last of the code. You can place functions on diffent locations but I tend to place them at the bottom of the file. Place the following lines below the line that has the **loop** command.

```
Rem this function has code that recreates every chunk every time it is called.
Rem you could for instance also remove chunks and insert chunks. But here
```

```
Rem we just erase the chunkarray.
Function updatechunks()
   Rem get our current chunk location
   cx=(px/chunkwidth)
   cy=(py/chunkheight)
   Rem erase every chunk
   chunklist.lenght=0
   For y=-2 to 2
   For x=-2 to 2
      chunklist.insert(newchunk(cx+x,cy+y,x,y))
   Next x
   Next y
Endfunction


Rem this function reads from the chunklist and
Rem puts the information in the onscreen text.
Function drawchunks()
   For i=0 to chunklist.length
      Rem in the onscreen text put the chunk tile locations. The player would be
      Rem in the center one.
      Settextstring(t[chunklist[i].sx+2,chunklist[i].sy+2],str(chunklist[i].x)+","+str(chunklist[i].y))
   Next i
Endfunction


Rem this function creates a new instance of the chunk type
Rem that we can insert into the chunk array list.
Function newchunk(x1,y1,x2,y2)
   a as chunk
   Rem tile number, the printed text on the screen.
   a.x= x1
   a.y= y1
   Rem text location
   a.sx= x2
   a.sy= y2
Endfunction a //note the a being returned this way.
```

If you are here and have put the code above inside agk then it should run. You can see numbers on the screen. Imagine the player being in the center on a part of the world and around him being more parts of the world. You can see only as far as the parts of the world numbered around you. When you move the world gets updated and the world gets changed around you.

Distance equation manhattan

Sometimes you need to know the distance between two points. The manhattan method is one way to get the distance. Do not that there are more precise methods but this one is pretty easy and short.

```
Function distance(x1,y1,x2,y2)
   a = abs(x2-x1) + abs(y2-y1)
Endfunction a
```

The Random Bag

Sometimes the random command is not good enough. You might want more of a certain number to be returned. Maybe because your sword has a magic effect to hit with max damage more often. To solve this you could create a array with a series of numbers and randomly select a value from this. You could have 3 values of 3 and 6 values of 9(max damage) You could also then remove this value from the array for other purposes.

Note : *that below the code might not work*
If you type it into agk straight away. Read more
Of this book to see where to place certain lines.

```
Rem setup the array
Dim bag[10] As Integer

Rem this code would create random numbers
Rem in the array.
For i=0 to bag.Length
  bag[i] = Random(0,10)
Next

Rem code that prints a value from the bag array on the screen.
Print(bag[Random(0,bag.Length)])
```

Selection Lists

When you program things you will need to know techniques to get things done quick and easy. One technique here is something I learned to get a value from a list. I had the situations where I had a monster traveling through the woods using the random obstacle avoidance method. This method lets a ai agent(other word for monster etc.) step in a random direction if there is an obstacle in his way. Not every position around him would be reachable so you can use the following technique. Basically I loaded the free positions around the player in a list and selected 1 position to move to.

```
Rem Here we use a single array to simplify the example
Local sel as integer[0]

Rem Insert a couple of values into our list.
sel.Insert(10)
sel.Insert(20)
sel.Insert(30)

Rem Print out one value from the list(array)
Print(sel[Random(0,sel.Length)])
```

Note : *We could use 2 arrays(one for x and one for y)*
that contain the movement difference, so our monster
could step either left, up, down or right. Maybe you
could create some code yourself that does this.

Pattern movement

In video games we move stuff around. One technique of moving things around is called pattern movement. We for instance have a array with instructions telling a dog in the game what to do. The instructions we give the dog could be “Move left, move left, sit”

Below is a simple example of how we can move a sprite around on the screen.

```
Rem Just above the main loop in a new empty project place these following
Rem lines. This would be above the Do and Print(ScreenFPS()) lines.

Rem here is our array that wil contain the instructions.
Dim instruct[] as String
Rem here we insert instructions.
instruct.Insert(“down”)
instruct.Insert(“down”)
instruct.Insert(“left”)
instruct.Insert(“left”)
instruct.Insert(“up”)
instruct.Insert(“up”)
instruct.Insert(”right”)
instruct.Insert(”right”)
Rem Here we create our sprite and set its size and position
dog = CreateSprite(0)
SetSpriteSize(dog,10,10)
SetSpritePosition(dog,30,30)
Rem this variable contains the position in the instruction
Rem array. 0 is the first command(“down”)
position = 0

Rem In the main loop place this following code. This would be in a new
Rem empty project below the Do and Print(ScreenFPS()) lines.

Rem we wil read from the array at the current position.
Select instruct[position]
  Case “up” // if in the array here is written “up”
    Rem move the sprite up.
    Setspritey(dog,getspritey(dog)-10)
  Endcase
  Case “down”
    Setspritey(dog,getspritey(dog)+10)
  Endcase
  Case “left”
    Setspritex(dog,getspritex(dog)-10)
  Endcase
  Case “right”
    Setspritex(dog,getspritex(dog)+10)
  Endcase
Endselect

Rem Increase our position in the instruction array
position = position + 1
Rem If we have no more instructions left the start back at 0.
If position > instruct.length then position = 0
```

Rem *end of code*

Tip : patterns like used here could be used in
more complex code like Genetic Algorithms.
Where for instance patterns are randomly created
and the most successful one(closest to destination)
Would be used for creating new patterns with
added random instructions and several mutations
(Replacing instructions)