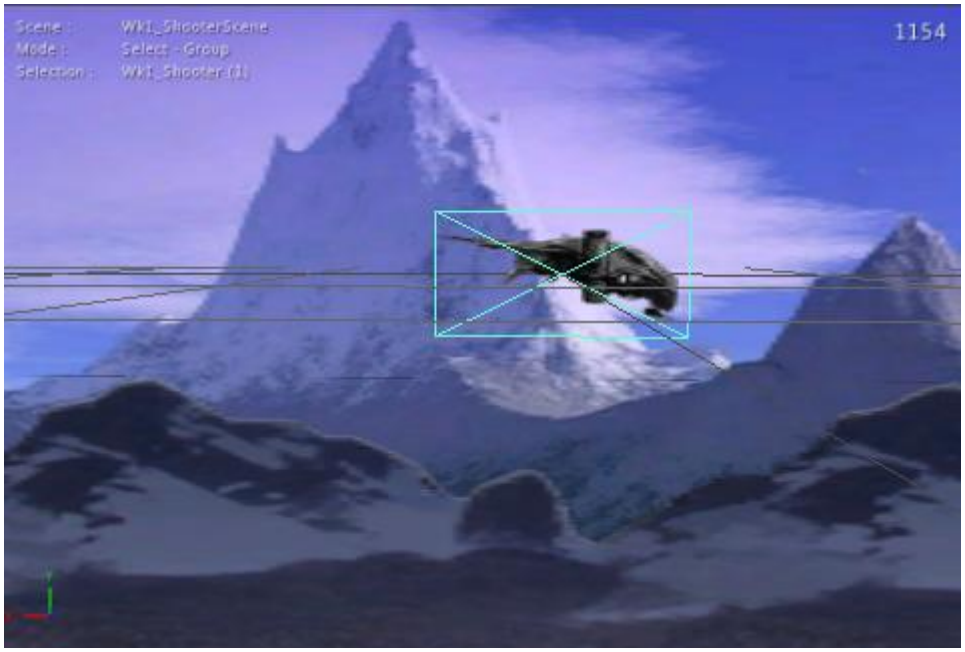
Week2 Lecture: iPhone Games

Contents

Design Discussion.

Okay onto the next bit. We will do a bit of coding. Scared yet? Don't be, it's incredibly easy. We need to code the background and foreground layers to scroll as the player moves left and right to give the impression of speed and motion. We also need to handle scrolling and looping the layers so they loop seamlessly.



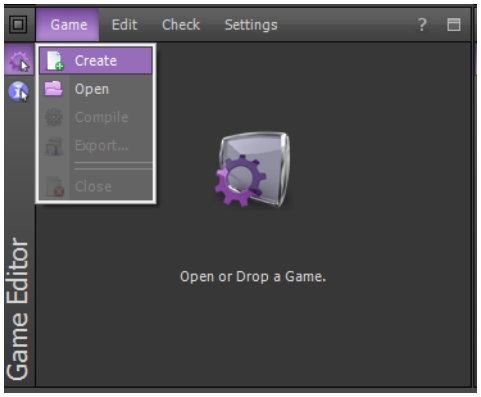
Here's the scene as we had it from last week. We have two layered back grounds and one space ship.

In order to start controlling this we need to add LUA code.

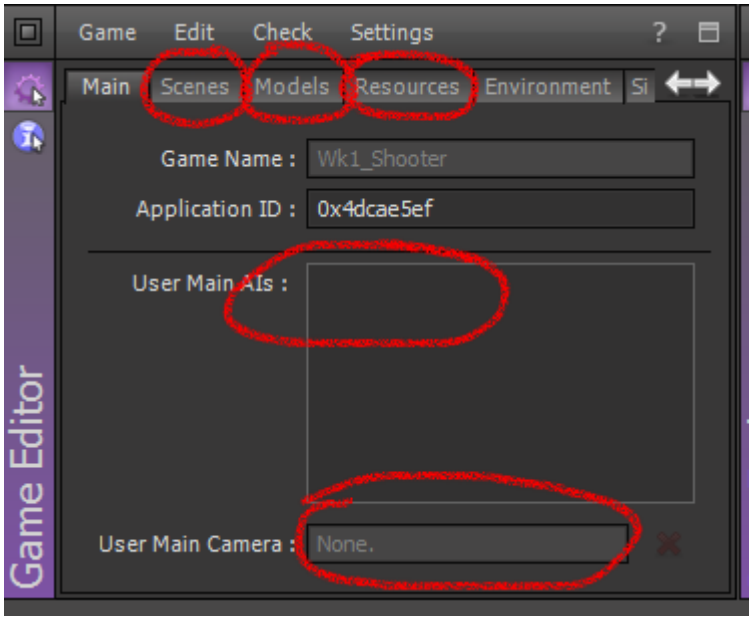
Game Creation.

Firstly we need to create a Game. So load up ShiVa where you were before (click on Main->Settings) and change to the correct folder before you do this.

In the Game Editor click on Game->Create



So now have done this, let's have a look at what we need to fill out next.

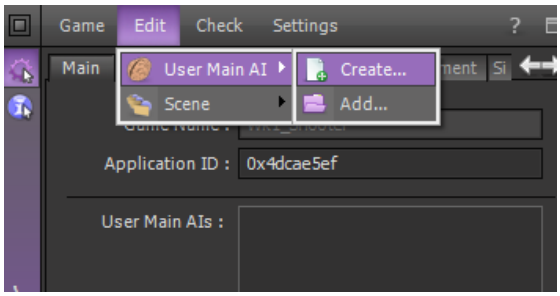


The red circled items are as follows.

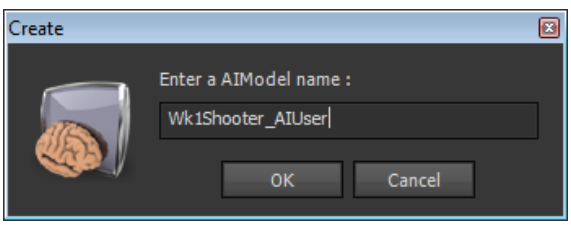
- Scenes – we drop our scene(s) into this section. Because it can take more than one scene, we can jump from scene to scene per level if we want to.
- Models – all the models required for all scenes added to the Game
- Resources – everything else, HUD's, graphics, materials.
- User Main AIs – all the LUA modules we will create that need to have access for the user. In general we only have to put one in here.
- User Main Camera – of all the cameras created, drop on in here for control during the game.

Let's start by making a User Main AI.

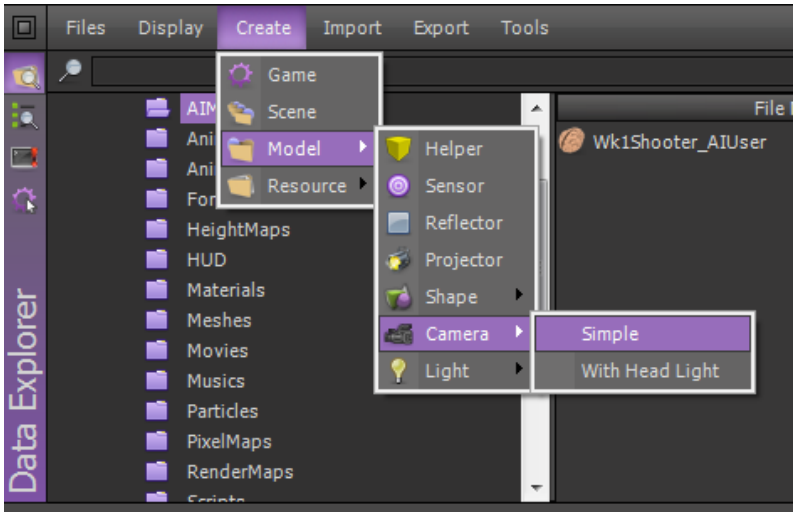
In the Game Editor, click on Edit->User Main AI->Create



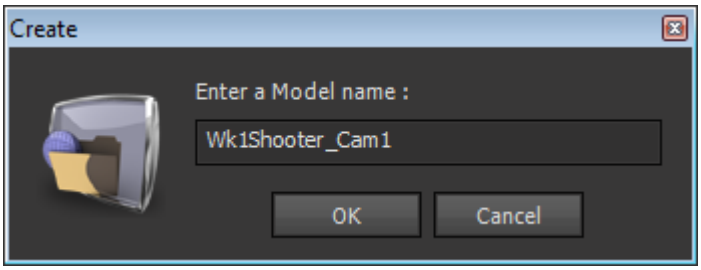
Call it Wk1Shooter_AIUser



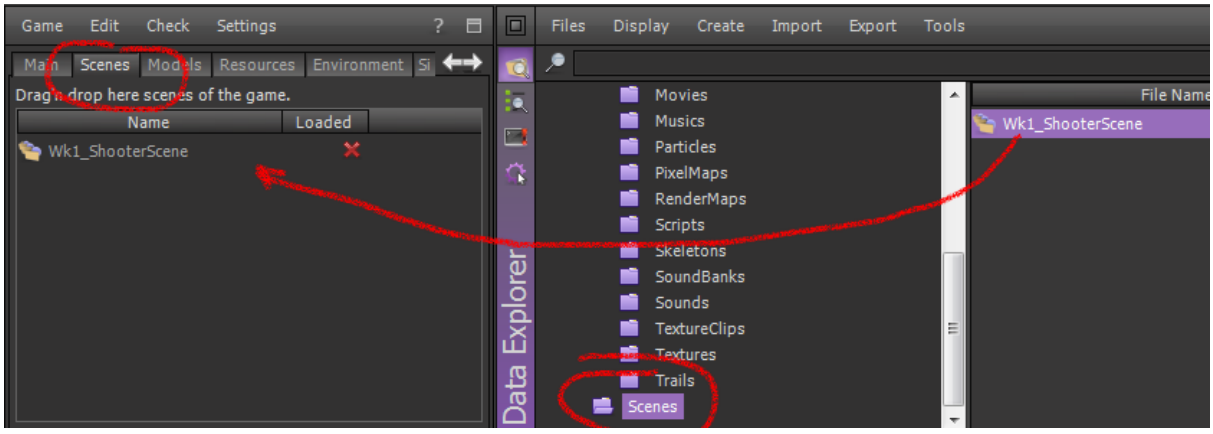
Now let's add a camera by going to the Data Explorer and clicking on Create->Model->Camera->Simple.



Call it Wk1Shooter_Cam1

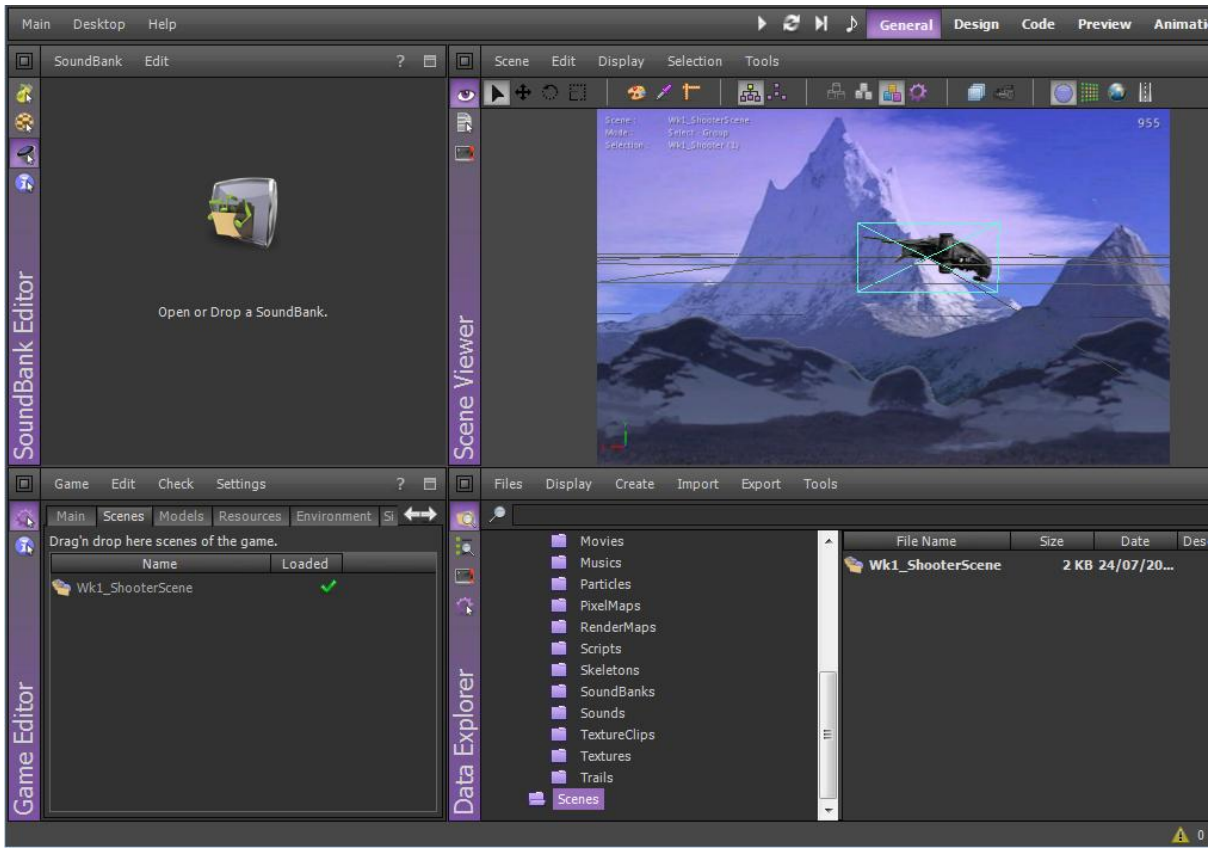


Scroll down to Scenes in the Data Explorer to display the Scene (don't click on it yet). Go over to the Game Editor and change the top tab to Scene, then drag that scene over to the Scene tab in the Game Editor now open.

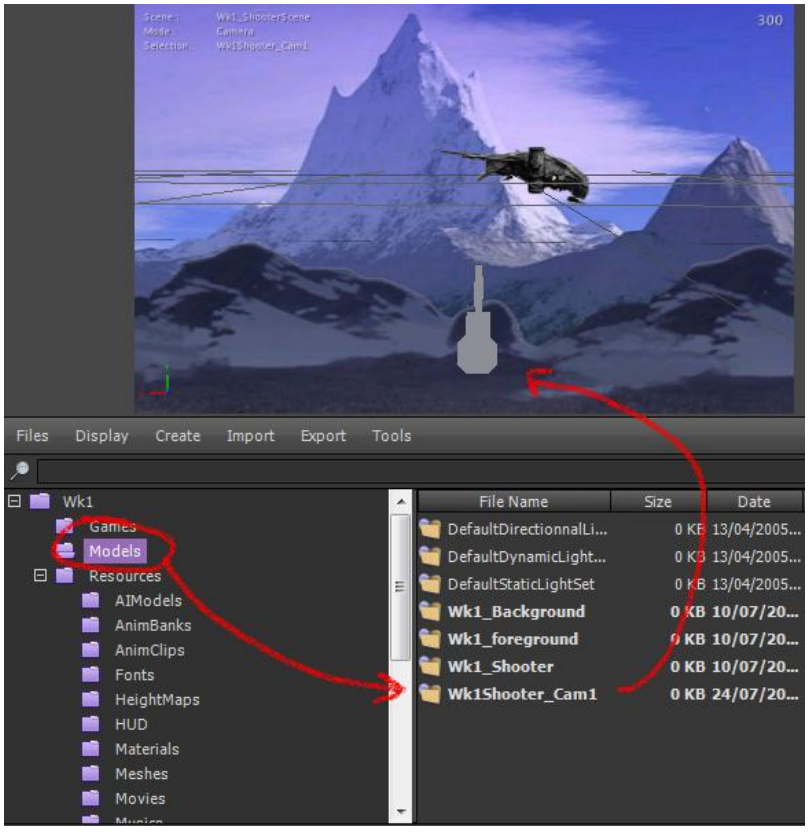


Now double click on the Scene in the Game Editor side...

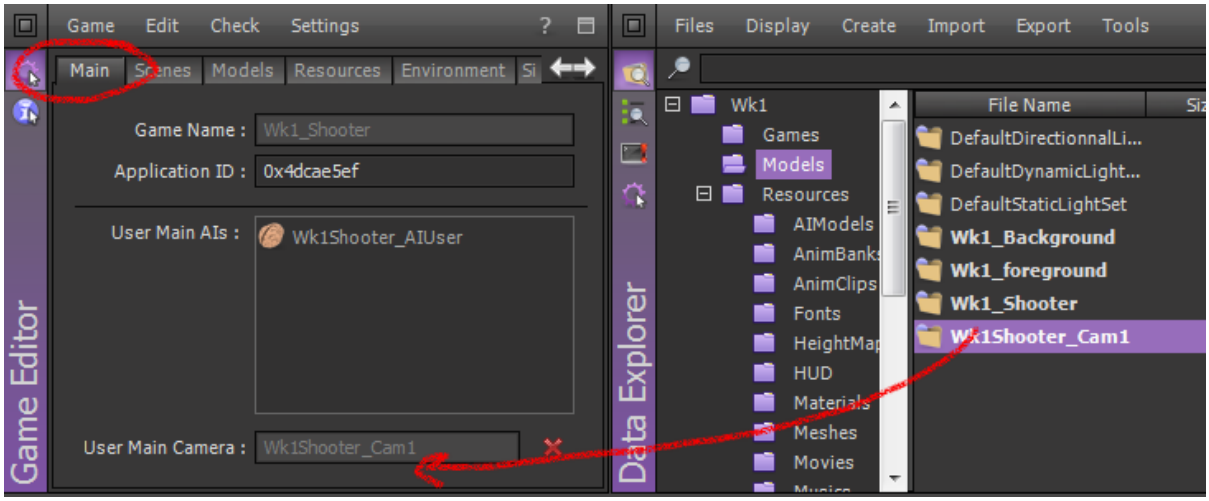
You should have a set up like this now.



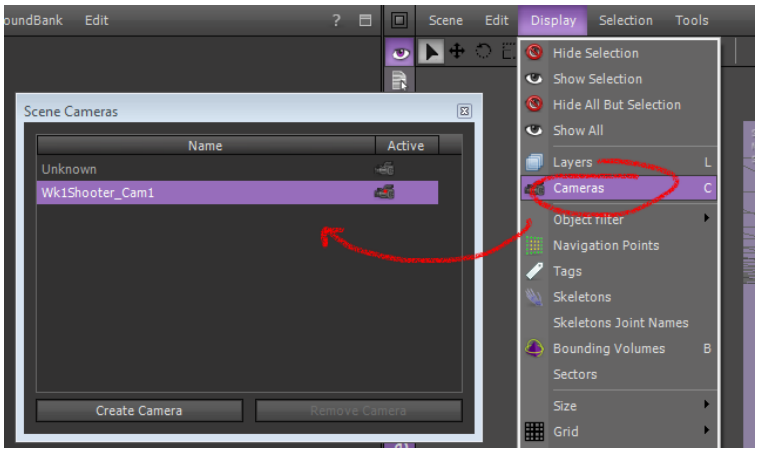
In the Data Explorer, scroll up and click on Models. Then drag the Wk1Shooter_Cam into the scene.

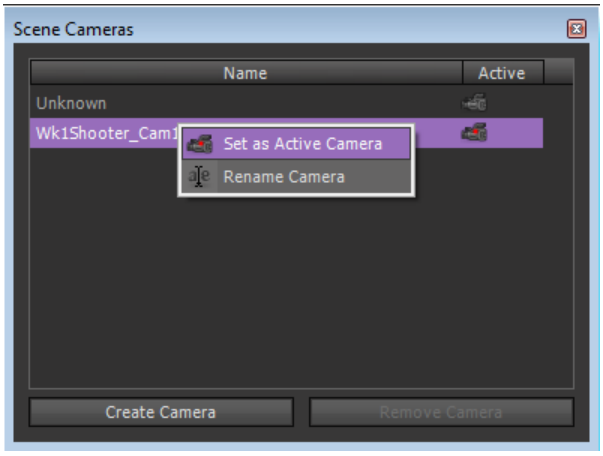


We may as well assign this camera as the main camera for the Game, but you don't always want to do this. Let's do this by clicking on the Main tab in the Game Editor and dragging the Wk1Shooter_Cam we just dragged into the scene, but this time drag it over to the edit box at the bottom that says User Main Camera.



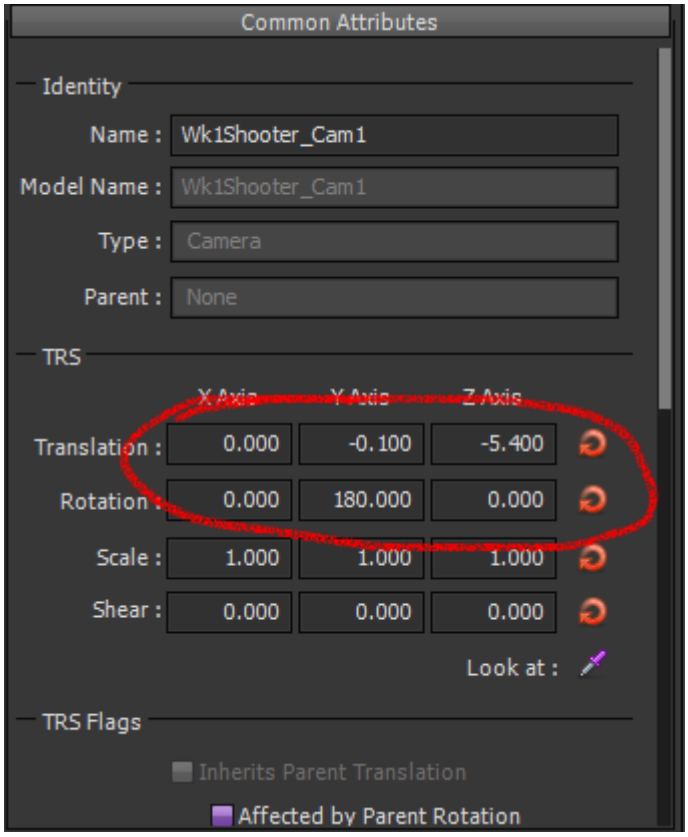
Now click in Scene Viewer, click on Display->Cameras and right click to select Wk1Shooter_Cam1 as the active camera. The scene will flick around, so you'll need to set up the camera.



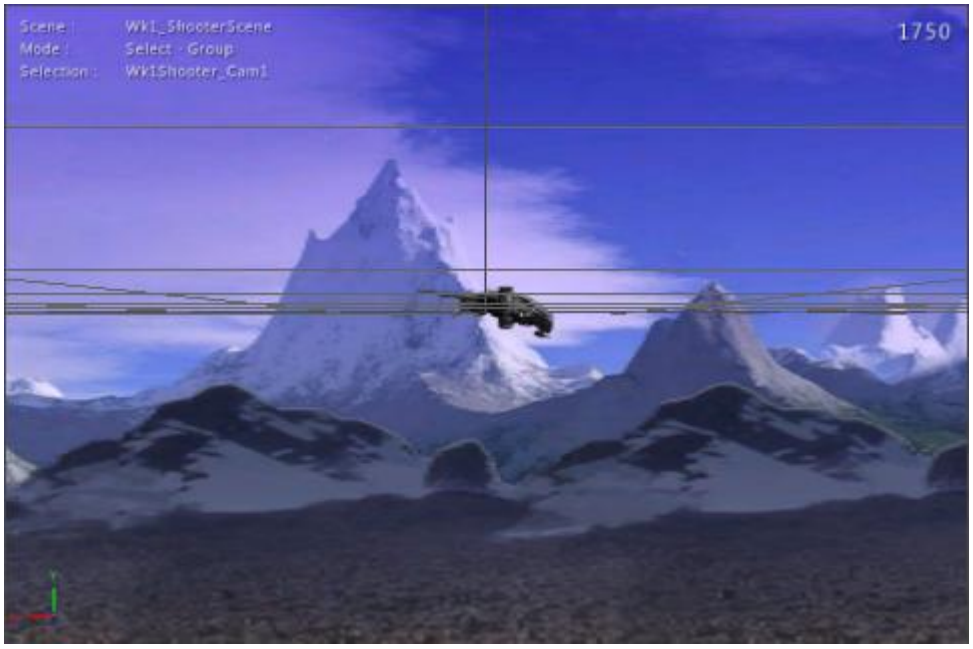


Close that screen and select the Attribute Editor on the left hand side.

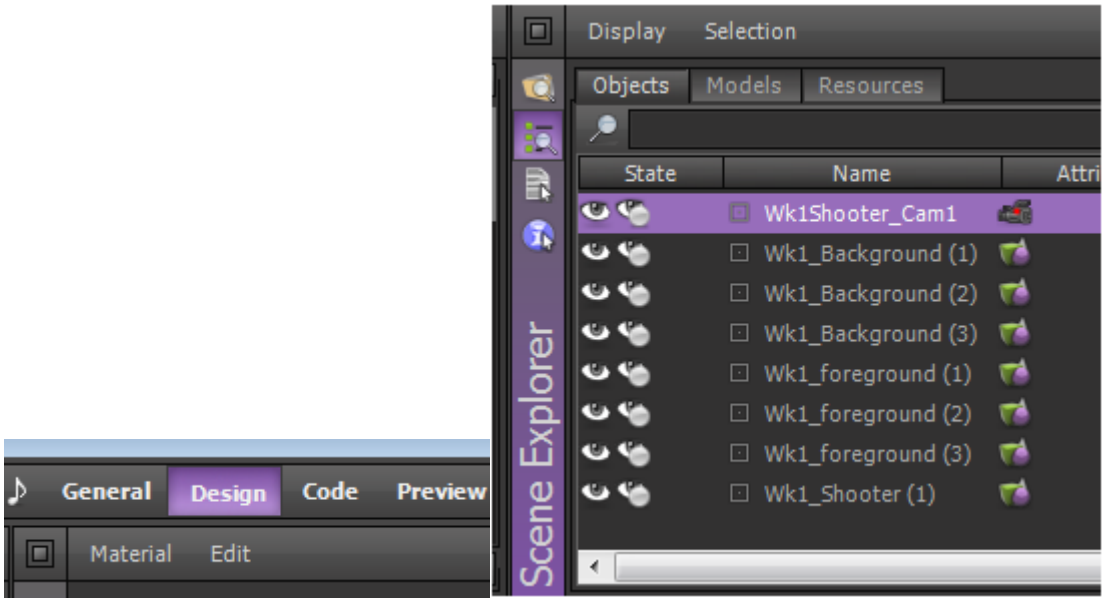
Now change the camera translation and rotation to this.



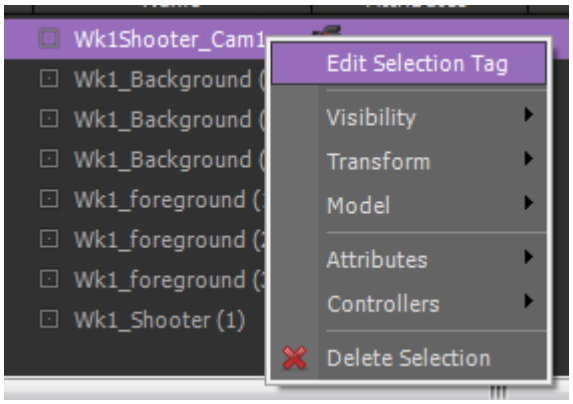
You should have a scene like this now.



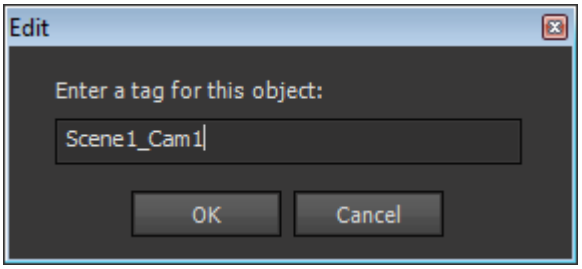
Change to Design view and in the Scene Explorer delete DefaultCam and Unknown cam if they are there.



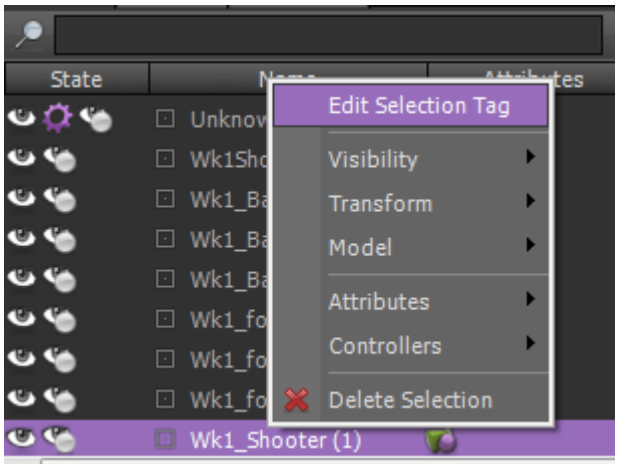
Right click on the Wk1Shooter_Cam1 in the Scene Explorer and click on Edit Tag Selection



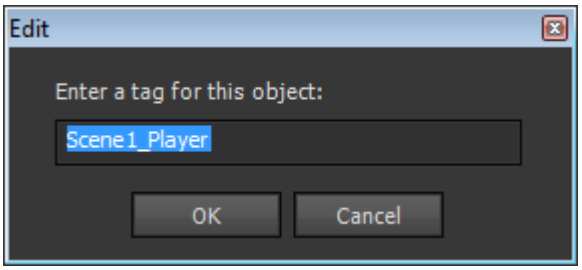
Name it Scene1_Cam1.



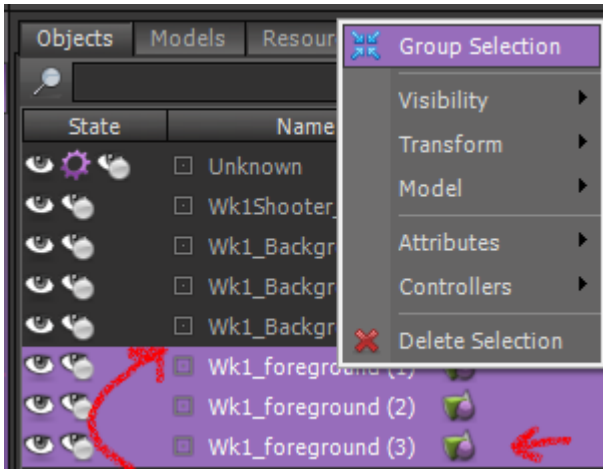
Then right click on the Wk1_Shooter(1) and select Edit Selection Tag.



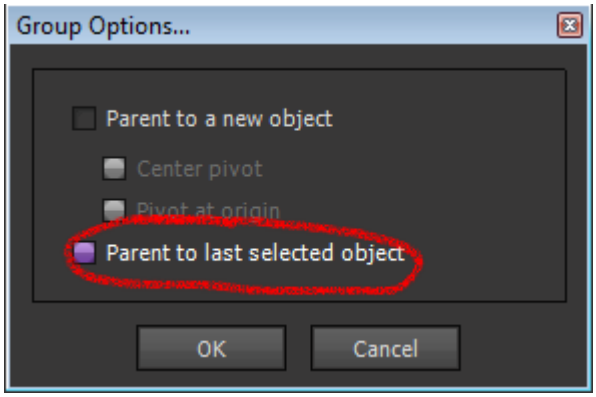
Then name it Scene1_Player



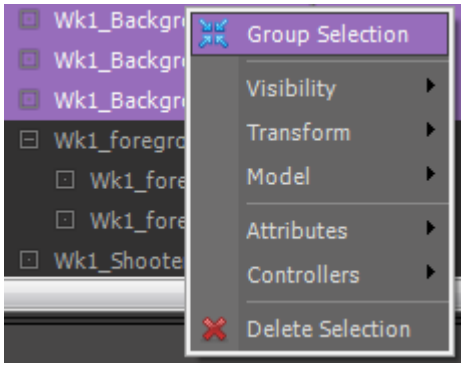
Left click on the Wk1_foreground(3) first then SHIFT+Left click on Wk1_foreground(1) last, then right click and Group Selection.

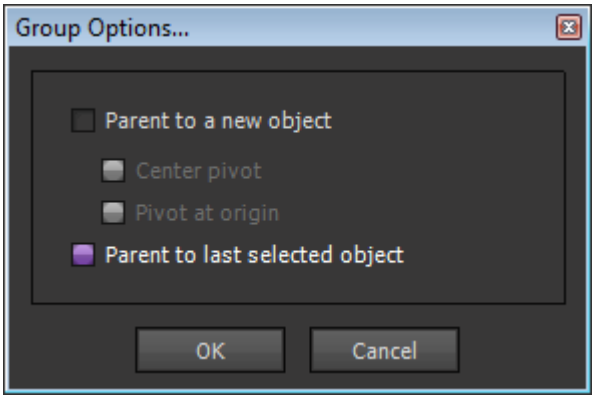


Change the settings to Parent to last Selected Object

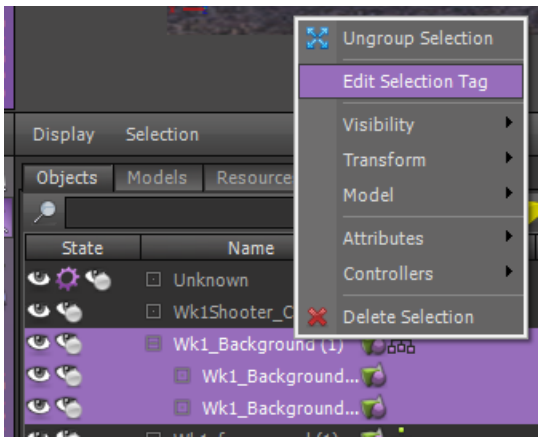


Do the same for Wk1_background(3) to (1)

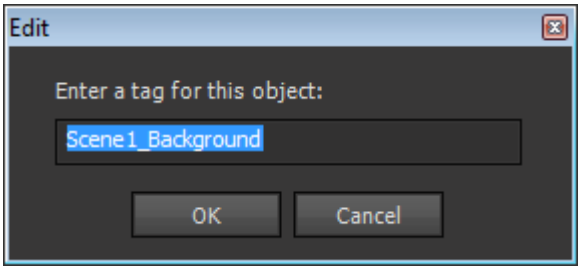




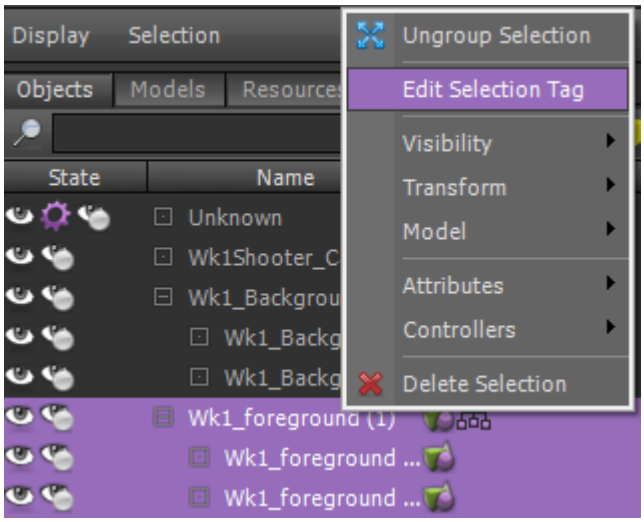
Now right click on Wk1_Background(1) and select Edit Selection Tag.



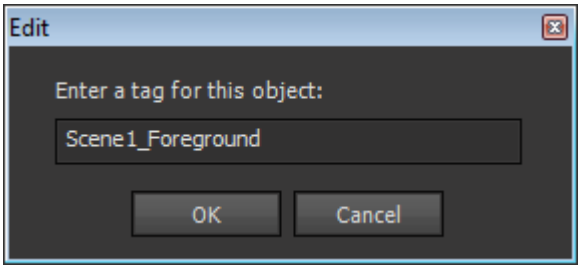
Name it Scene1_Background



Then right click on Wk1_Foreground(1) and select Edit Tag Selection.



Name it Scene1_Foreground.



Okay all set to code.

AI Models

AI Models are pretty much a module that is a collection of LUA scripts batched together for one purpose. But don't even think of it like that. Think of an AI Model as the object code for your game. I tend to make a main one like we have as seen above in the User Main AI (Game Editor) to drive the game and separate ones for each object type such as the main character, alien generator, bullet generator etc.

This is where we want to focus on Object Oriented coding.

Factory Methods.

Even though the design pattern (ways to rethink coding to make it more efficient) for Factories is as WIkiPedia explains it below I take a slightly different approach.

The **factory method pattern** is an object-oriented design pattern to implement the concept of factories.

Like other creational patterns, it deals with the problem of creating objects (products) without specifying the exact class of object that will be created. The factory method design pattern handles this problem by defining a separate method for creating the objects, which subclasses can then override to specify the derived type of product that will be created.

Outside the scope of design patterns, the term *factory method* can also refer to a method of a factory whose main purpose is creation of objects.

(http://en.wikipedia.org/wiki/Factory_method_pattern)

We will use it more like the final statement. For example we will want to create aliens at some point, and we want to do it simply in the final abstracted code. Something like...

```
generateAlien( x, y, z, type, image )
```

And that's as complex as we want to get. Of course we have to write the core code below that, the factory method and the object instance code but we only do that once.

Abstraction.

I've introduced another concept of object oriented coding here called abstraction. From WikiPedia...

In computer science, the mechanism and practice of **abstraction** reduces and factors out details so that one can focus on a few concepts at a time.

(http://en.wikipedia.org/wiki/Abstraction_%28computer_science%29)

So in the instance of the alien generation problem, we know at the end of the day in our game code we want to just "generate an alien". How we make this a possibility is to create a function called something similar to that to abstract the internal complexity away from the problem at hand. We don't care that it takes a sprite, some internal registers, a bunch of drawing calls, and perhaps some sound control too – we JUST WANT AN ALIEN.

Encapsulation.

Factories also introduce the object oriented coding term Encapsulation. Factory methods encapsulate the creation of objects. This can be useful if the creation process is very complex, for example if it depends on settings in configuration files or on user input.

From WikiPedia...

- A language mechanism for restricting access to some of the object's components.[3][4]

- A language construct that facilitates the bundling of data with the methods operating on that data.[5][6]

Programming language researchers and academics generally use the first meaning alone or in combination with the second as a distinguishing feature of object oriented programming. The second definition is motivated by the fact that in many OOP languages hiding of components is not automatic or can be overridden; thus information hiding is defined as a separate notion by those who prefer the second definition.

As information hiding mechanism

Under this definition, encapsulation means that the internal representation of an object is generally hidden from view outside of the object's definition.

(http://en.wikipedia.org/wiki/Encapsulation_%28object-oriented_programming%29,

http://en.wikipedia.org/wiki/Information_hiding)

So what does this mean to us? It means we generate don't hack around with the internal registers manually; we use the class public methods to do this.

For an example of information hiding, I have an object ALIEN. It has an internal register (or attribute – object oriented version of variable) called xPos. I could quite easily go...

```
ALIEN.xPos = 100
```

But as a responsible programmer, what I would do to stop people hacking this around is make the xPos attribute private so it couldn't be accessed externally. This generates an exception (or error) pretty much telling the programmer "hey back off man, don't be messing with my internal data like that".

We should be nice though and provide a public (available to anyone) method (object oriented version of function)

```
function GetPositionX( )
     return this.xPos()
end
```

And to be fair we should also add a set method...

```
function SetPositionX(  position )
     this.xPos( position )
end
```

So to now legally access the object we can do this..

```
ALIEN.SetPositionX( 100 )
```

In ShiVa however we don't really write code like that (except for the get and set methods)

We would write something like...

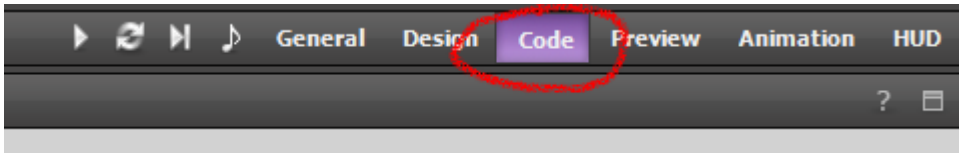
```
object.sendEvent(application.getCurrentUser, "AlienAI",
"setPosition", 100)
```

It's also to future-proof the object code, so if I decide to change its name to xPosition, the calling code still works because I just change all references to xPos manually and the outside code does care, ie SetPosition will still work.

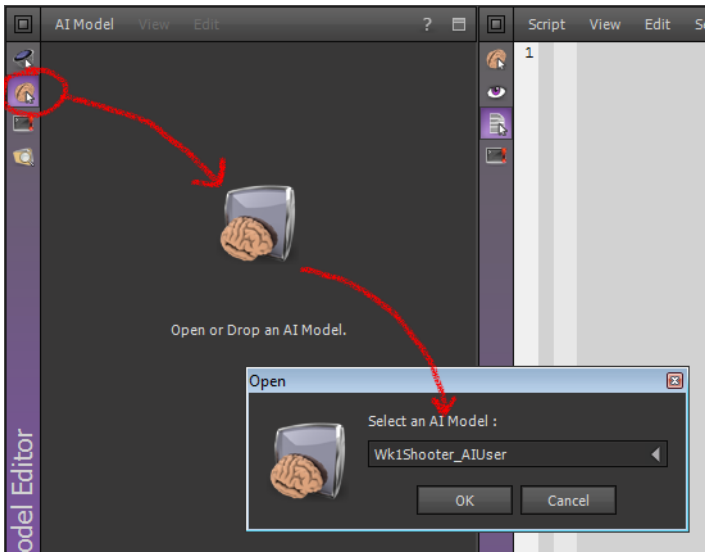
Coding User Main

Now that we know all that, we can be reasonable certain about the way we should design our code. We'll start with the Wk1Shooter_AIUser module.

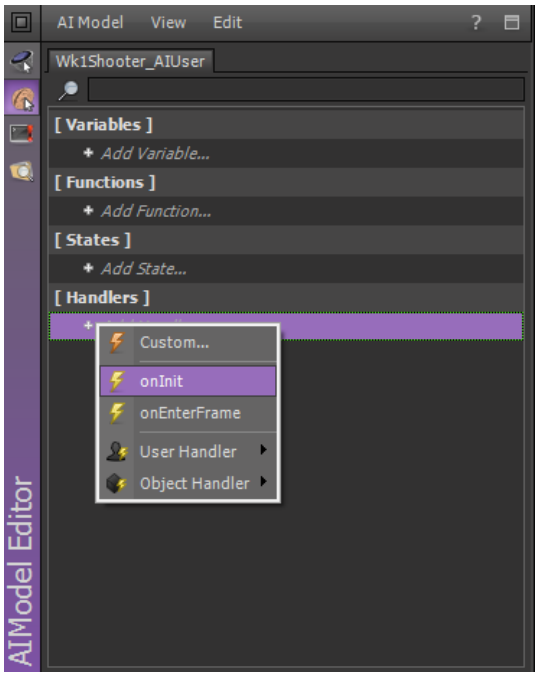
At the top right of the screen click on Code.



On the left hand side of the screen we can now see the AI Model Editor. Click on the big icon to Open and AIModel, and select Wk1Shooter_AIUser...

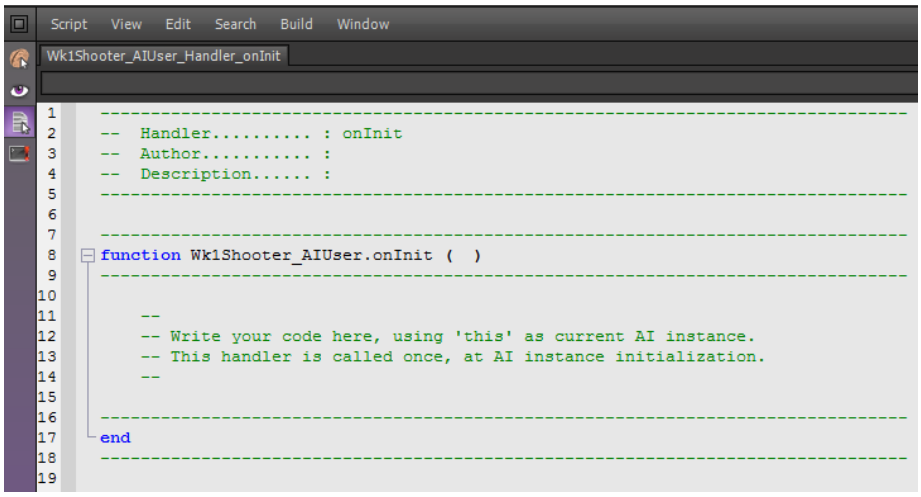


Now, to have some action occur at the loading of the AI Model, you need to amend the "onInit" function. To do this you need to access the "Handler" for the AI Model, by left-clicking on the "+" next to "*Add Handler…*" under the "[Handlers]" heading, and navigating to the "onInit" Handler:



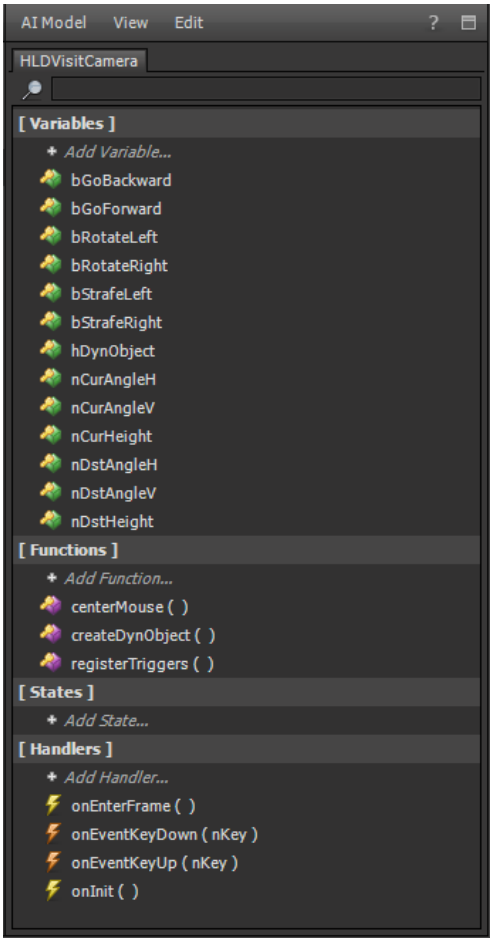
"Handlers" are just another term for a specific type of Script that "Handles" some Event, in this case the "onInit", or "On Initialisation" Event. Once you've clicked on the "onInit" Handler, you will see that the Handler has been added below the "[Handlers]" heading in the AIModel Editor, and also that the Script has been opened in the Script Editor. If you can't see the Script Editor, open it up now.

In the Script Editor, you will notice that a dummy Script has been opened up for you. This Script has a header that allows you to put in your name and the description of the Script, and also a preformed body that has one Function – Wk1Shooter_AIUser.onInit. Note that the name of this Function is taken from the AI Model's name (Wk1Shooter_AIUser) and the selected Handler (onInit).



It is **VERY IMPORTANT** that you do not change anything other than lines beginning with "--" or blank lines. If you do, the chances are that your Script will not work.

AI Model Editor



The AIModel Editor module allows you to edit AIModel resources. An AIModel is a behavioural model that can be attached to an Object or to a User. Once attached to an Object, the initial values of the Variables of the AIModel can be overridden in the AI Attributes section of the Attributes Editor. As a result, different behaviours can be created using the same AIModel.

Basically, the AIModel Editor allows you to define Variables, Functions and Handlers, configure Variables for the designer view, search and compile.

onInit Code

What do we want in onInit? Well anything basically that happens once because it will never come back in here unless we explicitly call it. I have a bunch of things I also put in there so you may as well too. Grab this code from Blackboard under (week 2)

Create games for mobile phones and PDAs [22893|-]

```
-------------------------------------------------------------------------------
function Wk1Shooter_AIUser.onInit ( )
-------------------------------------------------------------------------------

    ---------------------------------------
    -- set up the screen orientation
    ---------------------------------------
    application.setOption ( application.kOptionViewportRotation, 0 )        -- default handset view
    if (system.getOSType ( ) == system.kOSTypeIPhone) then
        application.setOption ( application.kOptionViewportRotation, 3 )    -- turn device counterclockwise (common)
    end

    ---------------------------------------
    -- load our first scene
    ---------------------------------------
    application.setCurrentUserScene ('Wk1_ShooterScene')
    -- set up the camera
    local cam1 = scene.getTaggedObject ( application.getCurrentUserScene ( ), 'Scene1_Cam1' )
    application.setCurrentUserActiveCamera ( cam1 )
    -- adjust the viewing distance
    camera.setFieldOfView ( cam1, 22 )
    -- load assets and save data
    this.loadEverything ( )
    this.loadEnvironment ( )
    -- refresh network
    network.disconnect ( )

-------------------------------------------------------------------------------
end
-------------------------------------------------------------------------------
```

First of all it sets up the screen, and then if we are running this on an iPhone (iPod/iPad) it rotates the screen for us. This kind of adaptive programming is important when we are developing on PC and deploying to iPhone. It means we can see the screen the right way on both devices.

The second set of code sets the scene used to the be the one we created before called "Wk1_ShooterScene".

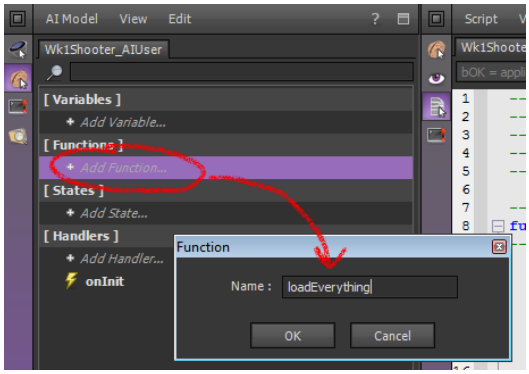
Next it sets up a local pointer to the scene camera and sets it's field of view, also making it the active camera.

Then it runs two functions `loadEverthing` and `loadEnvironment` – we will have to write these. They should be purple when the interpreter finds them in the module.

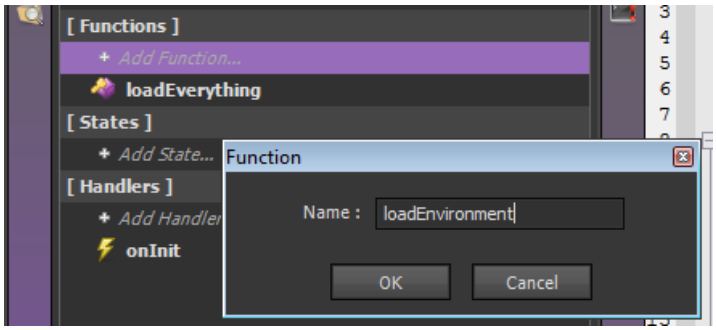
Finally we force a network disconnect because we may have been connected online last time, had a crash and we need to re-establish a link again.

For now we will create code stubs for `loadEverthing` and `loadEnvironment`.

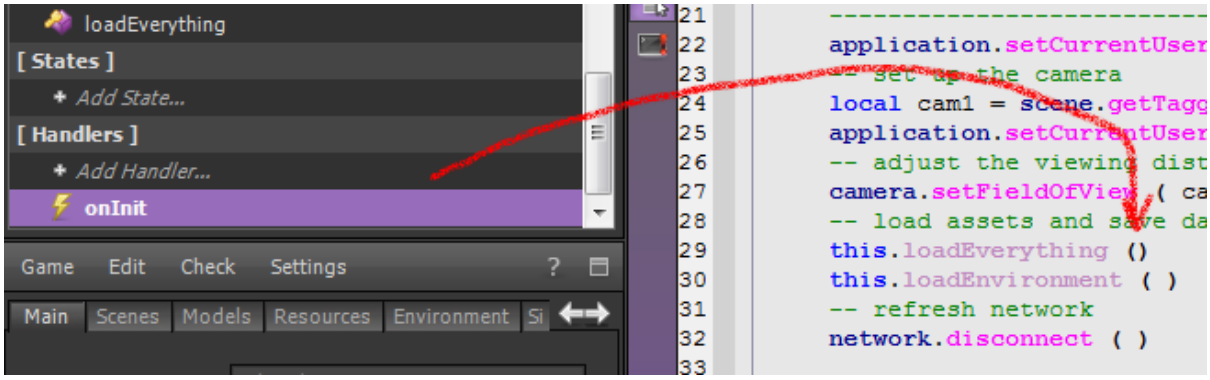
Click on +Add Function under [Functions] in the AIModel Editor, and create a function called loadEverything...



Next click on +Add Function under [Functions] in the AIModel Editor, and create a function called loadEnvironment...



Little trick – now open onInit again, remove the space between the first bracket after loadEverything and the two function will turn purple – they are now value methods (they were before the trick – just sometimes the script buffers don't show it).

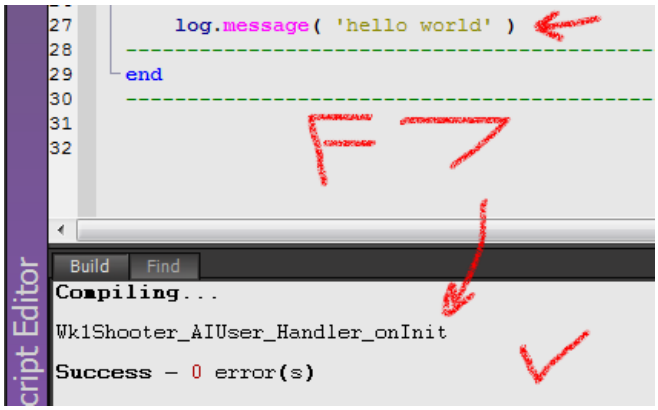


Hit space again to neaten it up.

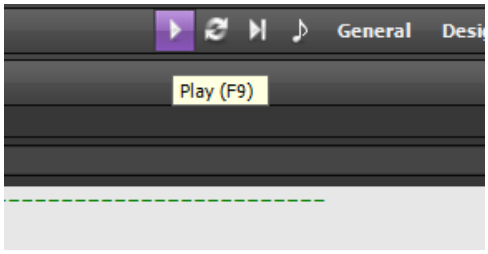
Let's just run it to make sure it all works. Just for fun put this at the bottom of the onInit function – we will remove this soon.

```
log.message( "hello world" )
```

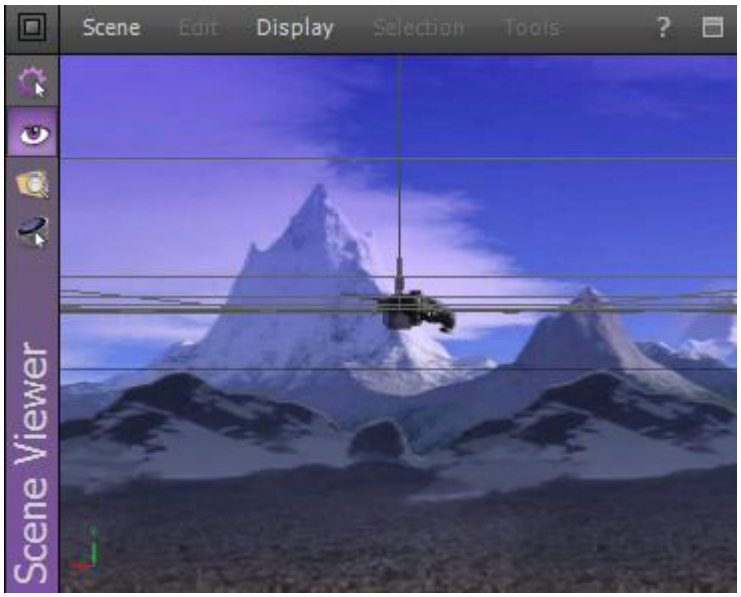
Now you must compile all scripts by pressing F7. If all goes well you should see something in the Output window.



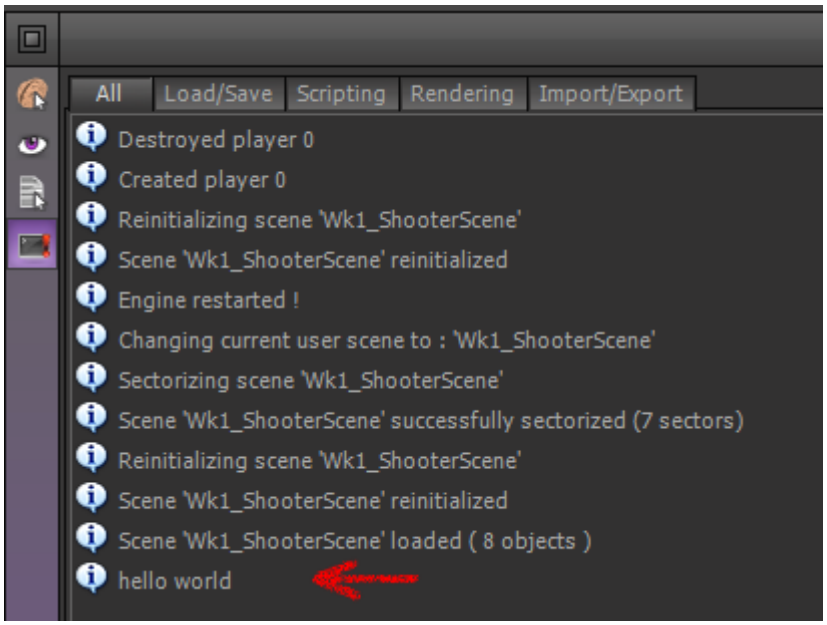
So now we are ready to run it. At the top of the screen is a play button. Hit it.



I got this view first run.



Change the Script Editor to Log Reporter and you will see the hello world printed to the log.

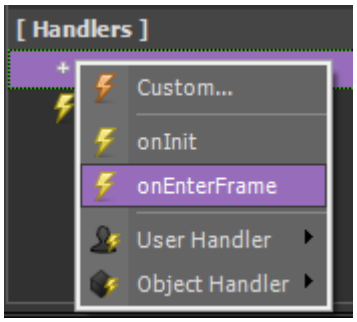


It all seems to work so ready to move the ship?

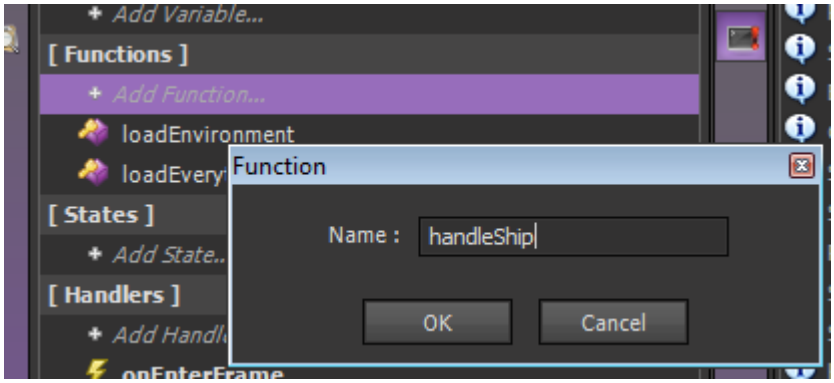
Moving The Ship

The plan is to move the ship, and move everything else as a function of the ship.

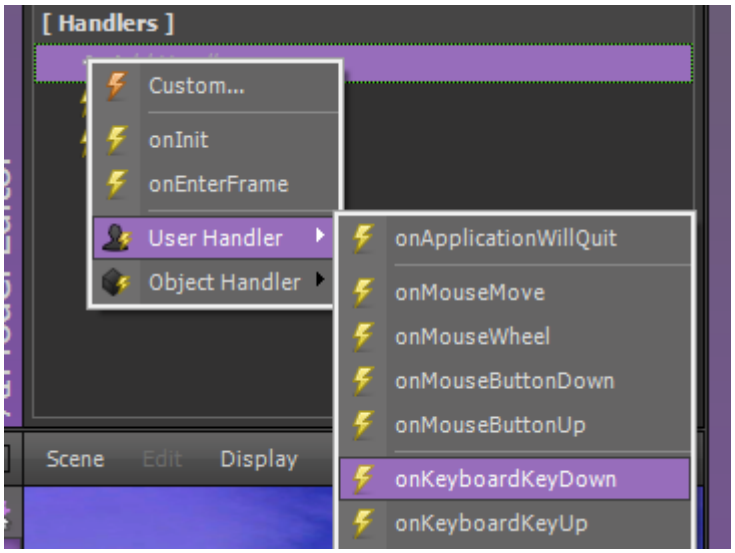
Eventually it will be cool to introduce camera lag so the camera has to catch up to the ship but we'll do that later. For now, add a new event called onEnterFrame.



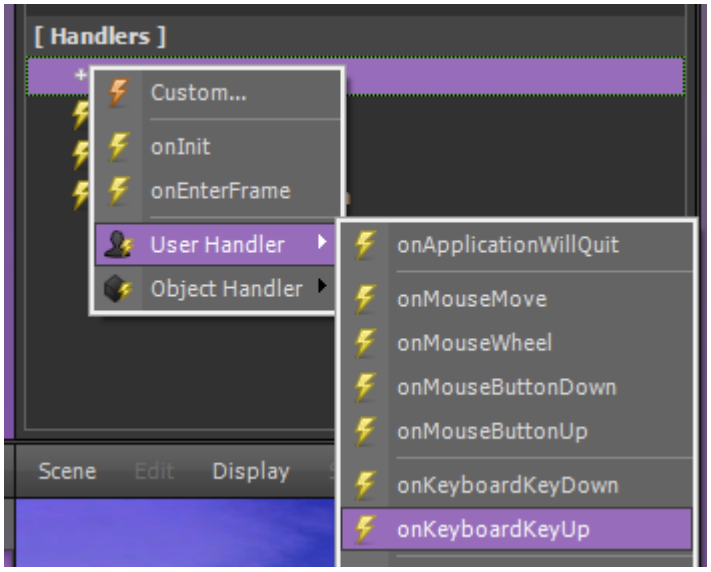
Add a function called handleShip



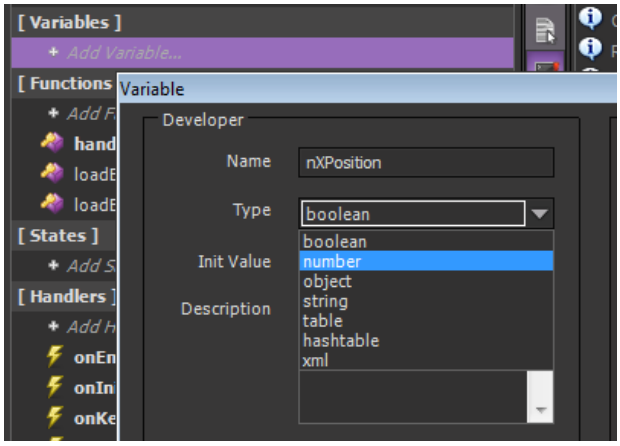
Add a handler called onKeyBoardDown



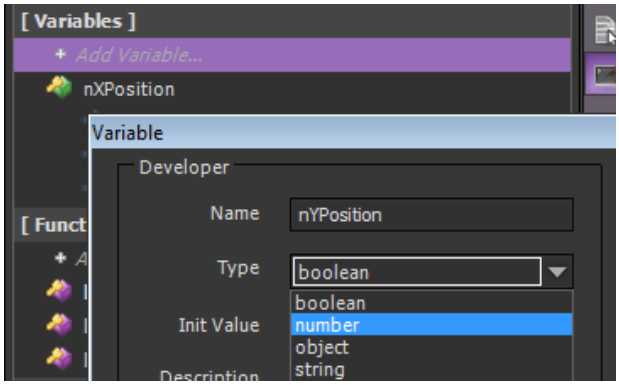
Add another handler called onKeyboardUp



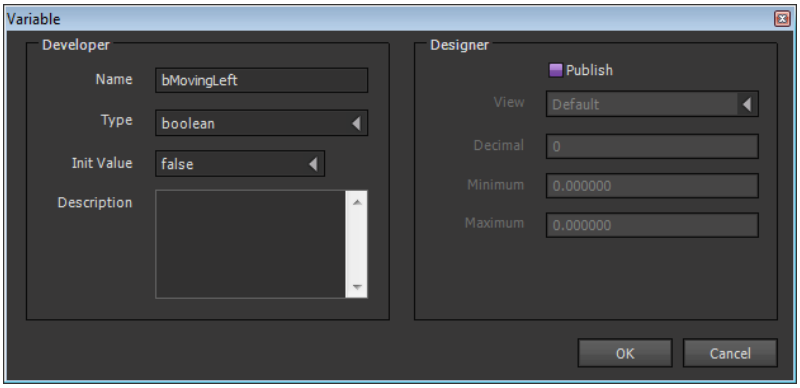
Add a variable called nXPosition, and make it a number type.



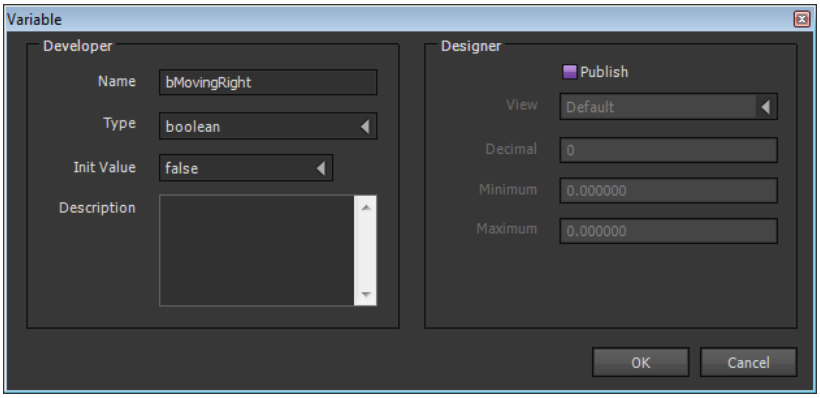
Then add nYPosition as well.



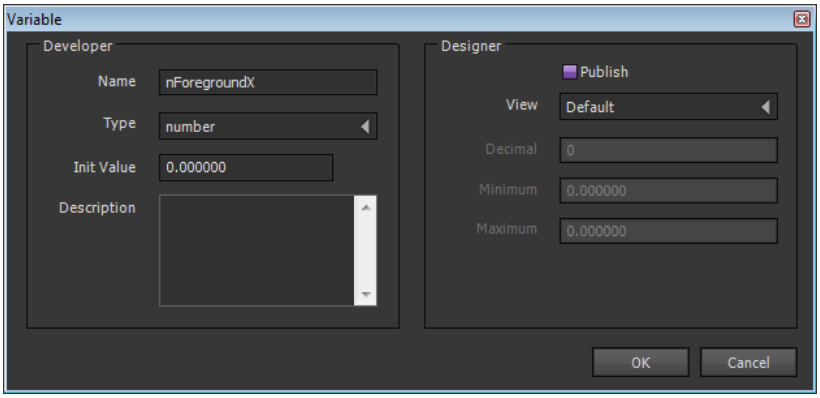
Add a variable called bMovingLeft, make it a Boolean and set it to false.



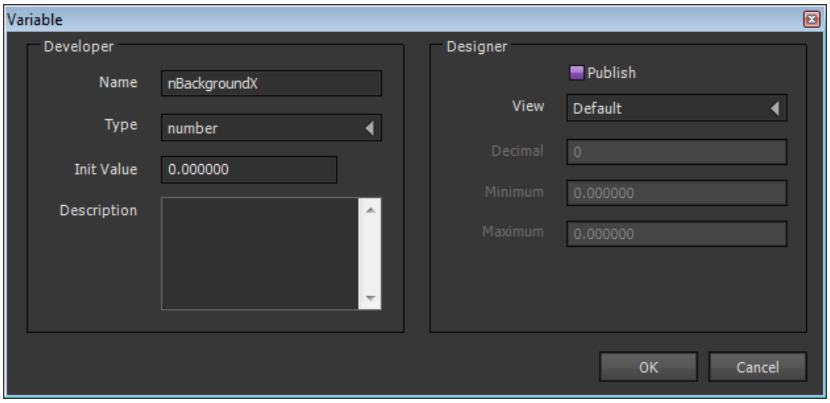
Then make a variable called bMovingRight, make it a Boolean and set it to false.



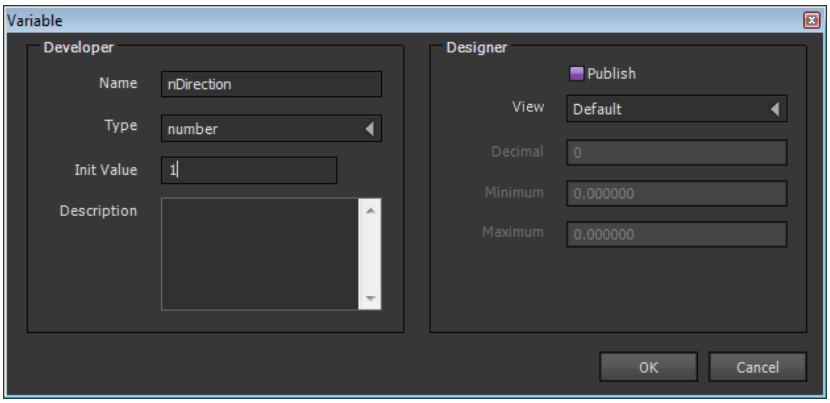
Add a variable called nForegroundX...



Add a variable called nBackground X



Finally add a variable called nDirection and set it to 1



Double click on onEnterFrame in the AI Model Editor and enter handleShip() as below

```
1  ----------------------------------------------------------------------------
2  --  Handler.......... : onEnterFrame
3  --  Author........... :
4  --  Description...... :
5  ----------------------------------------------------------------------------
6
7  ----------------------------------------------------------------------------
8  function Wk1Shooter_AIUser.onEnterFrame (  )
9  ----------------------------------------------------------------------------
10
11     this.handleShip ( )
12
13 ----------------------------------------------------------------------------
14 end
15 ----------------------------------------------------------------------------
16
```

Now double click on the handleShip function and paste the code from this file “handleShip”

Grab this code from Blackboard under (week 2)

Create games for mobile phones and PDAs [22893|-]

```
if (this.bMovingLeft ( ))
then
    this.nXPosition ( this.nXPosition ( ) + 0.05)
    this.nForegroundX( this.nForegroundX ( ) - (0.05 * 0.75))
    this.nBackgroundX( this.nBackgroundX ( ) - (0.05 * 0.05))

end
if (this.bMovingRight ( ))
then
    this.nXPosition ( this.nXPosition ( ) - 0.05)
    this.nForegroundX( this.nForegroundX ( ) + (0.05 * 0.75))
    this.nBackgroundX( this.nBackgroundX ( ) + (0.05 * 0.05))
end

local scn = application.getCurrentUserScene ( )
local player = scene.getTaggedObject ( scn, 'Scene1_Player' )

local sX, sY, sZ = object.getScale ( player )
local pX, pY, pZ = object.getTranslation ( player, object.kGlobalSpace )

-- adjust scale based on direction
if (this.nDirection ( )==1)
then
    sX = -1
else
    sX = 1
end
object.setScale ( player, sX, sY, sZ )

-- move player
object.setTranslation ( player, this.nXPosition ( ), pY, pZ, object.kGlobalSpace )

-- then move camera to player position
local cam1 = scene.getTaggedObject ( scn, 'Scene1_Cam1' )
pX, pY, pZ = object.getTranslation ( cam1, object.kGlobalSpace )
object.setTranslation ( cam1, this.nXPosition ( ), pY, pZ, object.kGlobalSpace )


-- adjust other layer using a paralax effect
local background = scene.getTaggedObject ( scn, 'Scene1_Background' )
local foreground = scene.getTaggedObject ( scn, 'Scene1_Foreground' )

pX, pY, pZ = object.getTranslation ( background, object.kGlobalSpace )
object.setTranslation ( background, this.nBackgroundX(), pY, pZ, object.kGlobalSpace )

pX, pY, pZ = object.getTranslation ( foreground, object.kGlobalSpace )
object.setTranslation ( foreground, this.nForegroundX ( ), pY, pZ, object.kGlobalSpace )
```

Do the same for onKeyBoardDown and onKeyBoardUp (they are on Blackboard in the same location)

```
--------------------------------------------------------------------------------
function Wk1Shooter_AIUser.onKeyboardKeyDown ( kKeyCode )
--------------------------------------------------------------------------------

    if (kKeyCode == input.kKeyD and this.bMovingRight ( ) == false)
    then
        this.bMovingLeft ( false )
        this.bMovingRight ( true )
        this.nDirection ( 1 )
    end

    if (kKeyCode == input.kKeyA and this.bMovingLeft ( ) == false)
    then
        this.bMovingLeft ( true )
        this.bMovingRight ( false )
        this.nDirection ( -1 )
    end

--------------------------------------------------------------------------------
end
--------------------------------------------------------------------------------
```

```
--------------------------------------------------------------------------------
function Wk1Shooter_AIUser.onKeyboardKeyUp ( kKeyCode )
--------------------------------------------------------------------------------

    if (kKeyCode == input.kKeyD )
    then
        this.bMovingRight ( false )
    end

    if (kKeyCode == input.kKeyA )
    then
        this.bMovingLeft ( false )
    end

--------------------------------------------------------------------------------
end
--------------------------------------------------------------------------------
```

Click in the Script, press F7 to compile and run the game.

Now you should be able to move left and right and the background and foreground scroll along. If you go far left or right you outrun the scene boundary. We will work on that next week.