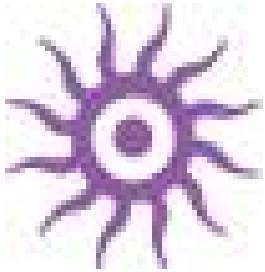


Code Ninja Competition

The Shiva Game Engine

(Multiplatform 3D Game Engine)

www.stonetrip.com

Sam Redfern

www.it.nuigalway.ie/~sredfern

www.psychicsoftware.com

Shiva: General Overview

The Shiva IDE

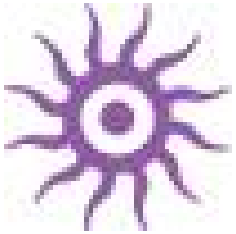
Overview of the Shiva API

Flow of Control

Key game/API entities

Miscellaneous useful/important topics

Shiva Game Engine (www.stonetrip.com)



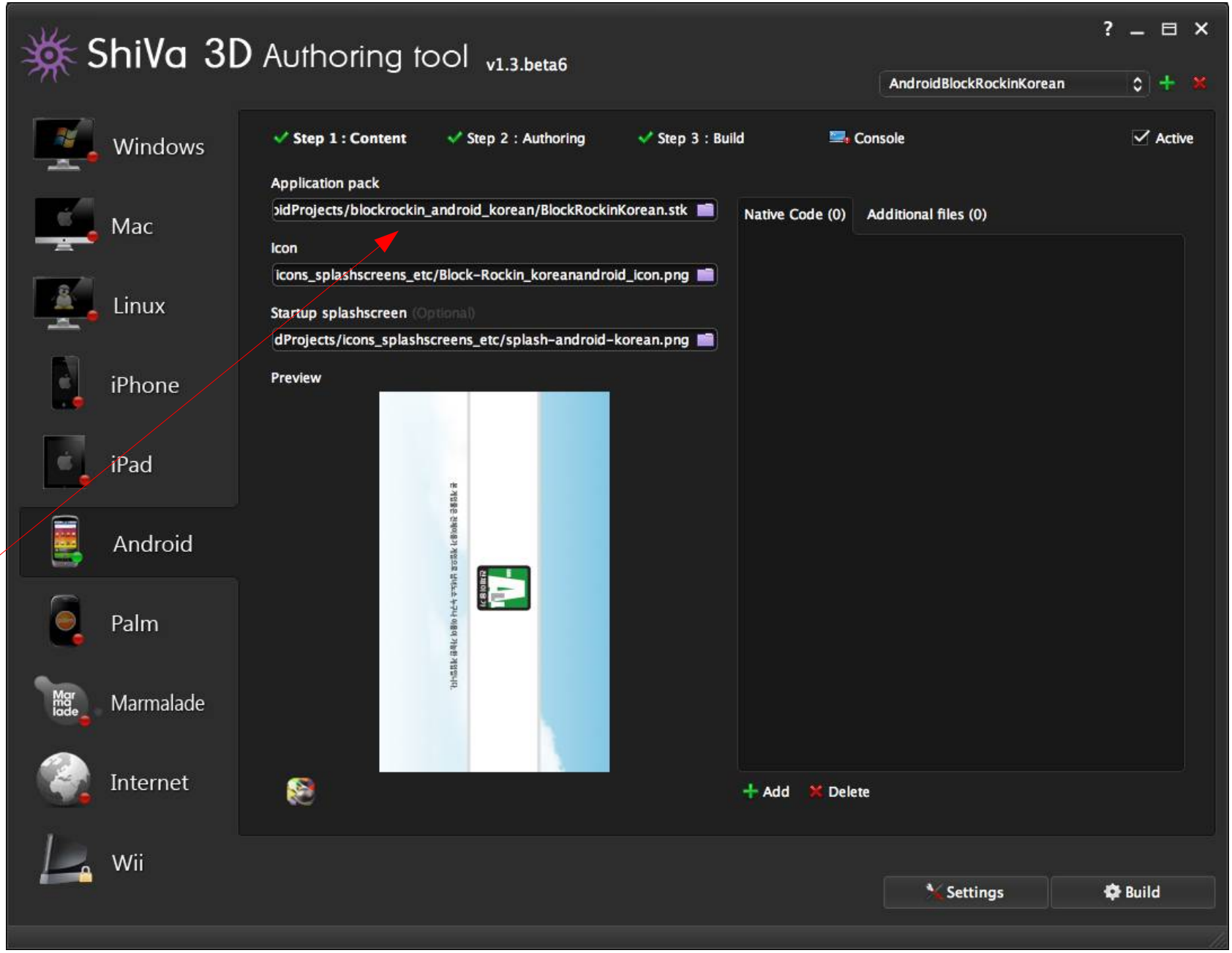
A closed-source games engine

- well designed and elegant to use
- excellent GUI/HUD editing and animation system
- exposes 'just enough' thru its API (*this is a good thing*)
- reasonable price full licence (~€200)
- deploys to: iOS, Android, Web plugin, Windows, Mac OSX, Linux, Wii, and more
- free version can do Web deployment + watermarked other versions for testing only

Shiva: platforms/"authoring tool"

Export executable app. immediately or produce project for further integration with 3rd party SDKs etc.

.stk file = shiva package exported from the main development environment



iOS/OSX: Xcode projects

mass.cpp – MarsDefender

4.2 | Deployment | MarsDefender | MarsDe...

Overview | Action | Breakpoints | Build and Run | Tasks | Info | Search

Groups & Files

- S3DEngine_iOS_Main.m
- S3DClient_iPhone.a
- Classes
  - Store.h
  - maslib.h
  - maslib.m
  - mass.cpp
  - mass.h
  - Store.m
  - StoreObserver.h
  - StoreObserver.m
  - S3DClient_Wrapper.h
  - S3DEngine_Wrapper.h
  - S3DEngine_Wrapper.cpp
  - S3DEngine_AppDelegate.h
  - S3DEngine_AppDelegate.m
  - S3DEngine_EAGLView.h
  - S3DEngine_EAGLView.m
  - S3DEngine_ViewController.h
  - S3DEngine_ViewController.m
  - S3DEngine_WWAN.m
- AIModels
- Plugins
- Additional
- S3DX
  - S3DXAIConstant.h
  - S3DXAIEngineAPI.h
  - S3DXAIFunction.h
  - S3DXAIModel.h
  - S3DXAIPackage.h
  - S3DXAIVariable.h
  - S3DXAIVariables.h
  - S3DXConfig.h
  - S3DXMacros.h
  - S3DXPlatform.h
  - S3DXPlugin.h
  - S3DXTypes.h
- AdMod
- TouchJSON
- Resources
  - Default@2x.png
  - Icon_114x114.png
  - Icon_57x57.png
  - S3DMain.stk
  - Default.png

File Name	Code
mass.cpp	57K

mass.cpp:1:1 <No selected symbol>

```
        if ( variables[0].GetType ( ) == S3DX::AIVariable::eTypeString )
        {
            //ObjectiveC_LogMessage(variables[0].GetStringValue());
            S3DX::AIVariable variable;

            int handle = maslib_GetProductByString(variables[0].GetStringValue());
            variable.SetNumberValue(handle);

            //ObjectiveC_LogMessage("buying item with handle:");
            //ObjectiveC_LogMessage(variable.GetStringValue());

            maslib_buyProduct(handle);
            //S3DClient_SendEventToCurrentUser("MASS", "onDeliverProductHandle", 1, &variable);
        }
    }
}

void productInfoCallback(ProductDescriptor_t *arg)
    //ObjectiveC_LogMessage("productInfoCallback");
    S3DX::AIVariable variables[5];

    // nHandle, sId, sName, nPrice
    variables[0].SetNumberValue(arg->handle);
    variables[1].SetStringValue(arg->sId);
    variables[2].SetStringValue(arg->name);
    variables[3].SetStringValue(arg->price);
    variables[4].SetStringValue(arg->description);
    S3DClient_SendEventToCurrentUser("MASS", "onDeliverProductInfo", 5, &variables);
}

void productBoughtCallback(ProductDescriptor_t *arg)
//int productHandle
{
    /*ObjectiveC_LogMessage("productBoughtCallback");
    S3DX::AIVariable variable;

    Don't do this at home, especially not with a c-style cast . . .
    variable.SetHandleValue((void*) productHandle);

    S3DClient_SendEventToCurrentUser("MASS", "onPerformProductPurchase", 1, &variable);
     */
    //ObjectiveC_LogMessage("productBoughtCallback");
    S3DX::AIVariable variables[5];

    // nHandle, sId, sName, nPrice
```

The precompiled shiva library

Engine-to-script bridge

ObjectiveC-to-C++ bridge

Your Lua game code as C++ classes (optional)

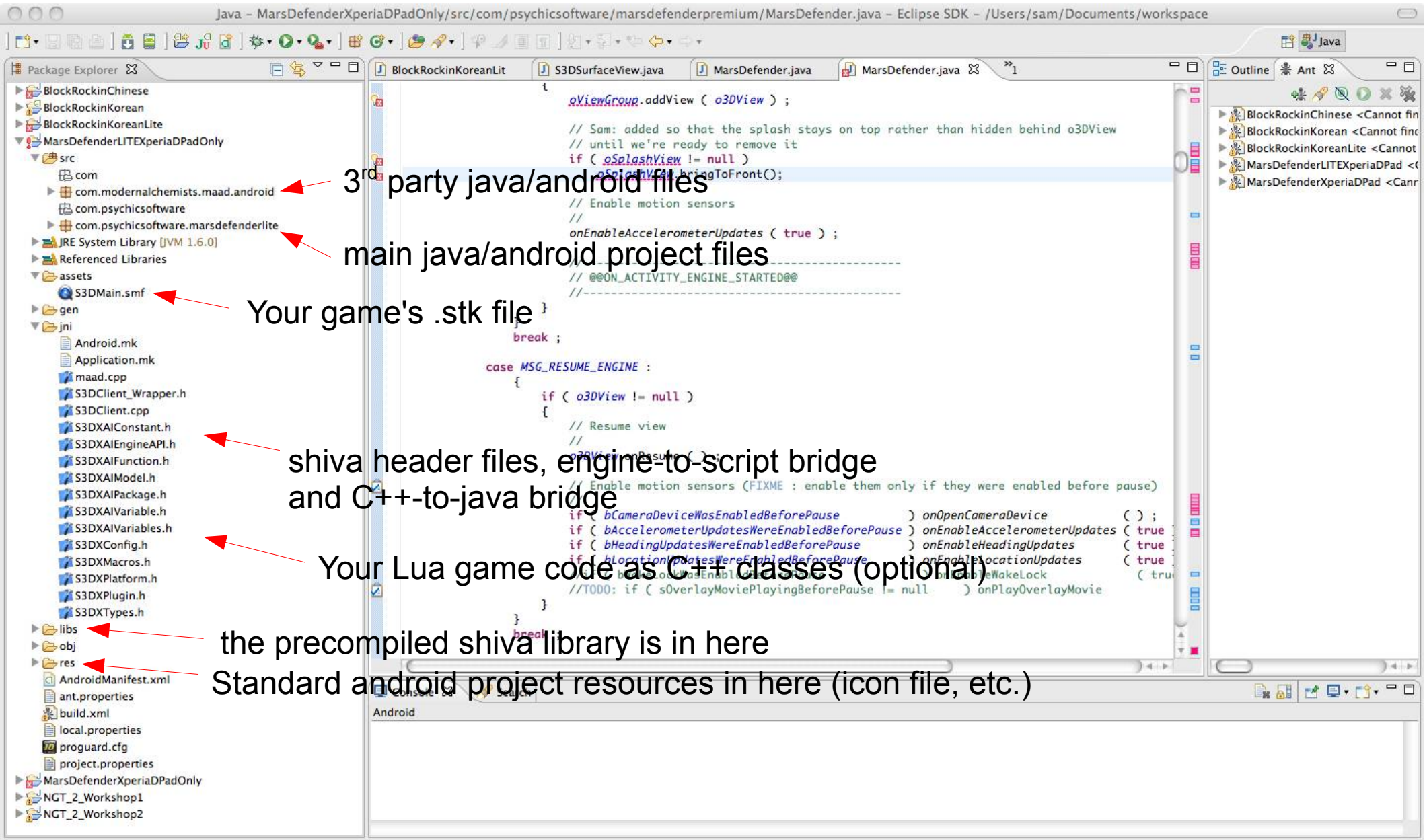
Shiva header files

3rd party library headers

Standard resources for Xcode project (icon file etc.)

Your game's .stk file

Android: Eclipse projects



3rd party java/android files

main java/android project files

Your game's .stk file

shiva header files, engine-to-script bridge and C++-to-java bridge

Your Lua game code as C++ classes (optional)

the precompiled shiva library is in here

Standard android project resources in here (icon file, etc.)

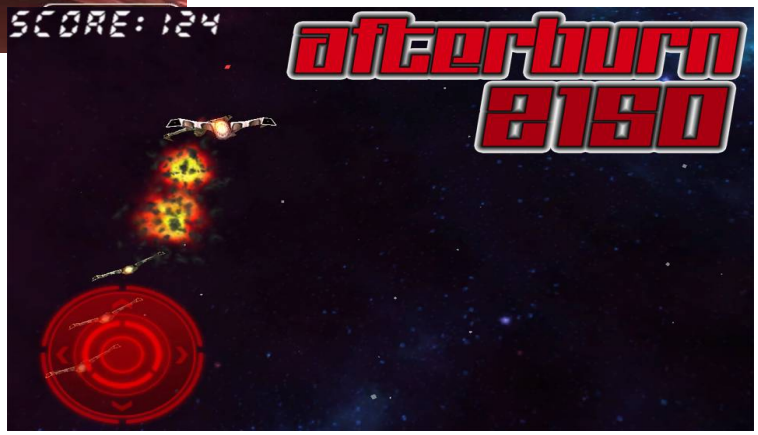
Some of my own Shiva games

- Since Feb 2011

Story-driven arcade-adventure

Turn-based car combat (early development)

3D tilt-controlled block-smasher



Space shooter (mid development)

Shiva: General Overview

The Shiva IDE

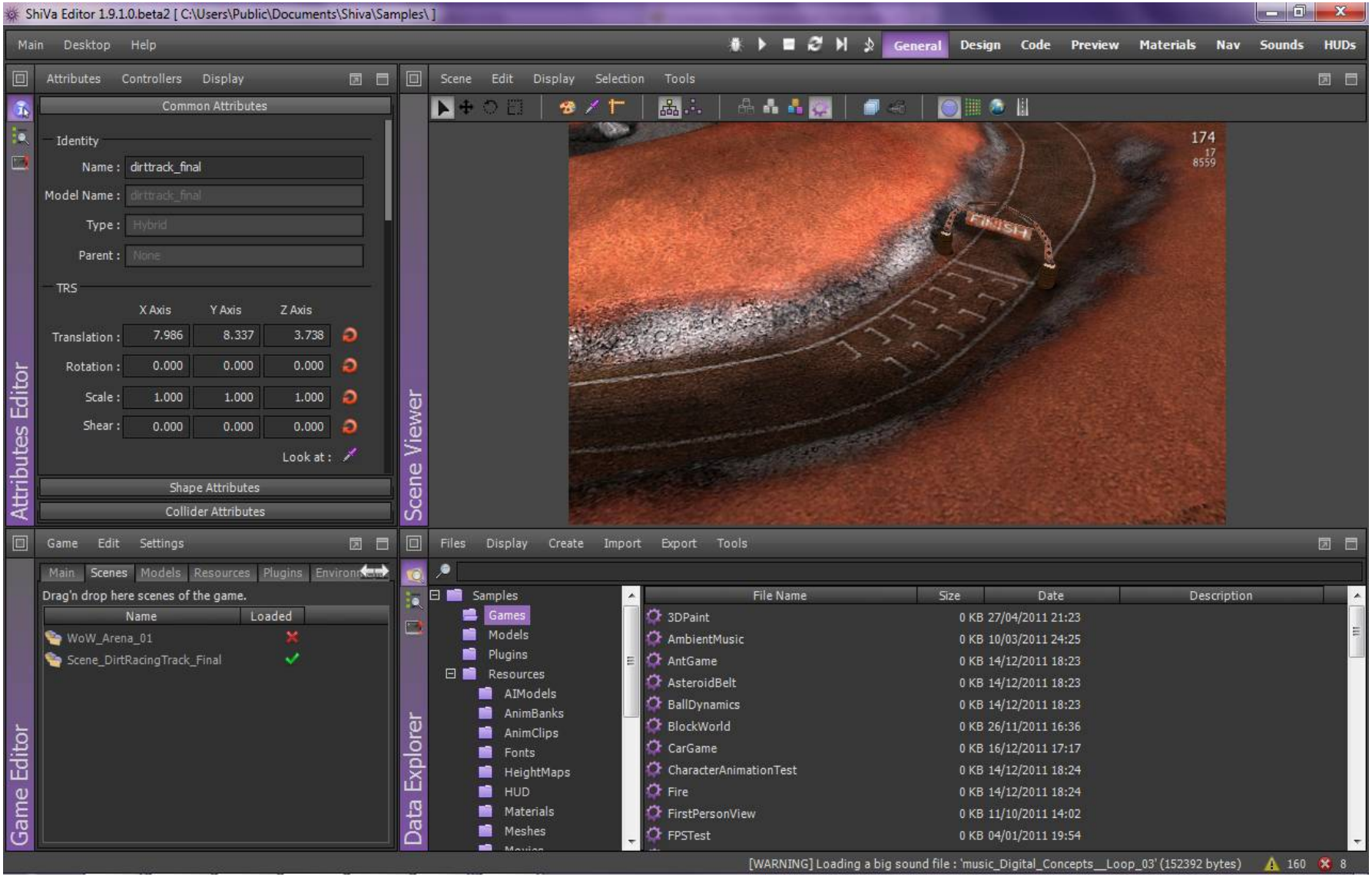
Overview of the Shiva API

Flow of Control

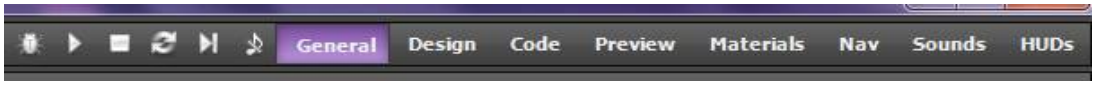
Key game/API entities

Miscellaneous useful/important topics

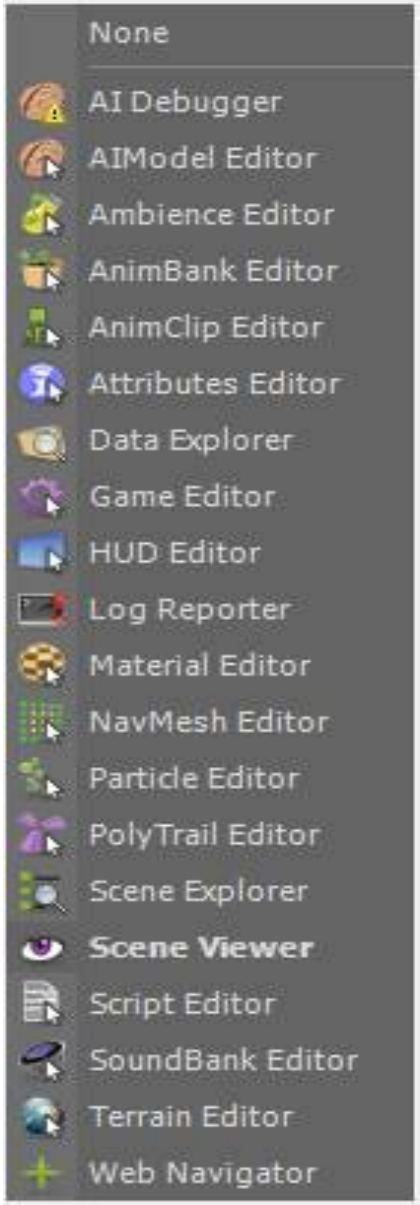
The Shiva IDE



The Shiva IDE



- Desktops
- Changing the panels in each desktop
- Important panels:
  - Data Explorer
  - Game Editor
  - Attributes Editor
  - AI Model Editor
  - HUD Editor
  - Log Reporter
  - Script Editor
  - Particle Editor
- More panels (slightly less important, maybe):
  - Ambience editor (must have a scene open)
  - SoundBank Editor
  - Material Editor
  - NavMesh Editor
  - Scene Explorer



Shiva: General Overview

The Shiva IDE

Overview of the Shiva API

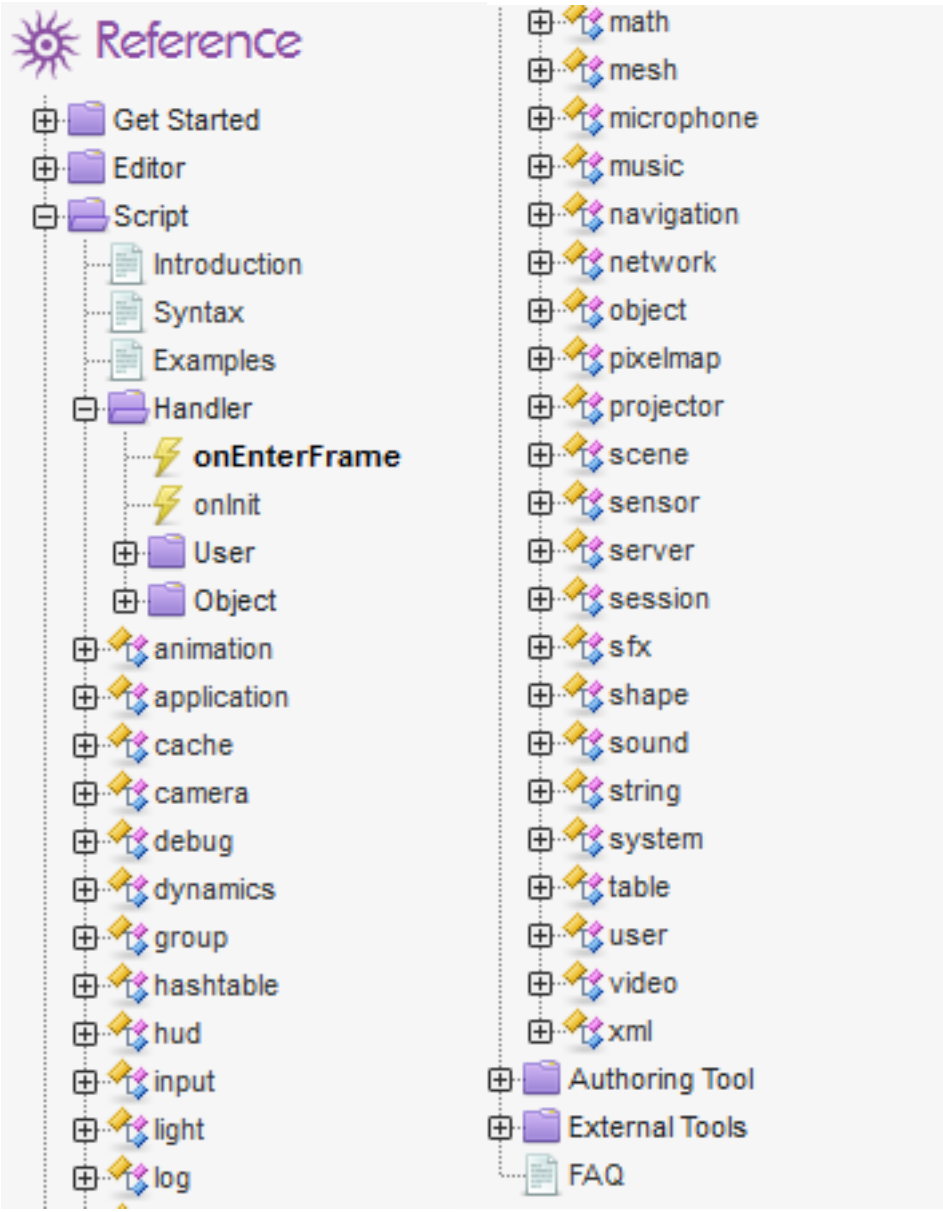
Flow of Control

Key game/API entities

Miscellaneous useful/important topics

The Shiva API - overview

- Log into stonetrip.com (create a free account)
- then select Developer > Documentation
-
- Also see the Shiva Wiki and Tutorials
- Available when logged into stonetrip.com
- Lots of targeted examples and explanations of specific topics



Key Game Entities

A single-user Shiva game consists of:

- One application object with various SDK methods
- One user object, with various SDK methods and one or more 'User Main AIs' which provide programmer-defined methods and data
- One or more scenes, with only one active at a time, with various SDK methods
- 3D game objects, live in the current scene, with zero or more attached AIs which provide programmer-defined methods and data
- Various assets (3D models, textures, AIs, HUDs) which may be loaded/deleted/attached at runtime

AIs

AIs are more or less equivalent to classes in languages like Java. They contain:

- Member variables
- Member functions (=private methods)
- Handler functions (=public methods plus automatically-invoked event handlers for keyboard, mouse, accelerometer etc.)
- States

AIs may be attached to:

- the user object (in which case they are single-instance)
- 3D game objects (in which case they may be instantiated multiple times, with each object having distinct copies of the member variables)

Shiva: General Overview

The Shiva IDE

Overview of the Shiva API

Flow of Control

Key game/API entities

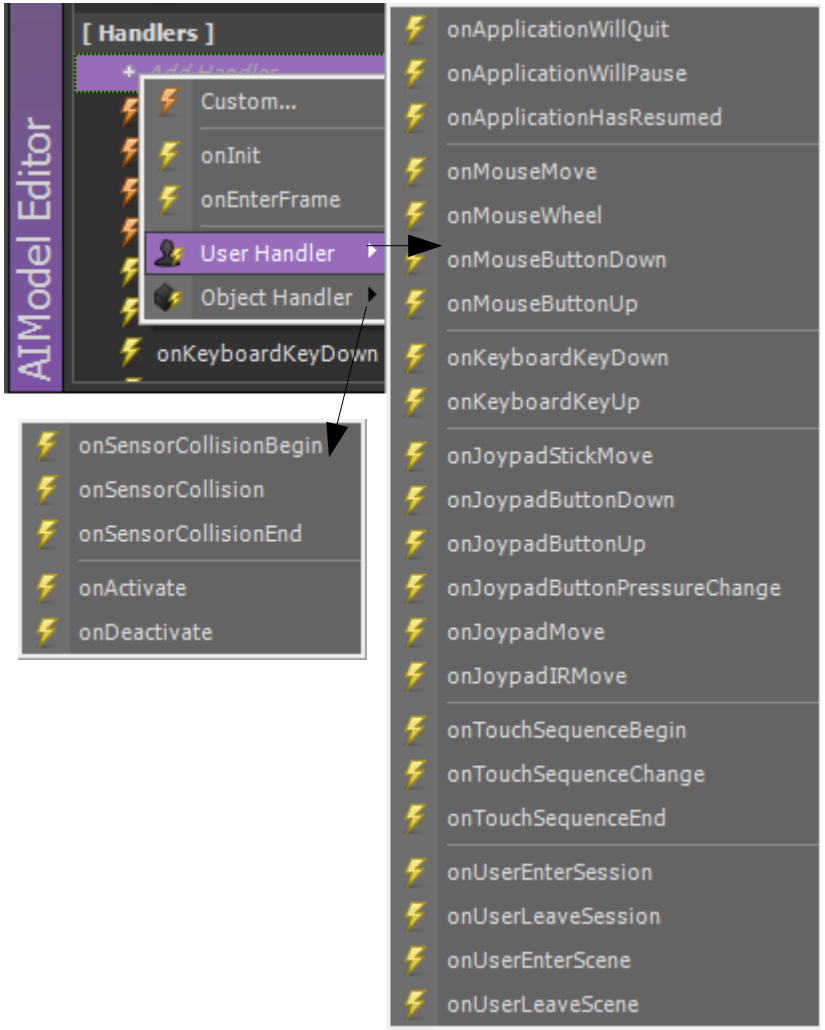
Miscellaneous useful/important topics

What happens when? (flow of control)

- Automatic flow of control
  - Predefined Handlers
  - States
- Programmed flow of control
  - Calling member functions
  - Calling custom handlers on the user or on a game object – with or without a delay

Automatic Flow-of-Control: Predefined Handlers

- onInit happens when the AI is first loaded (if this is a User Main AI, this means as soon as game starts)
- onEnterFrame happens once per frame update (i.e. every time the game is redrawn, many times per second)
- User handlers respond to system messages, the mouse, the keyboard, the joypad/accelerometer, multi-touch events, and multi-user enter/exit events
- Object handlers respond mostly to sensors (see later)



Programmed Flow of Control

- Calling member functions of the current AI
- Calling custom handlers on a game object
  - object.sendEvent
  - (See Joypad/Accelerometer example next slide)
- Calling custom handlers on the user's AI
  - user.sendEvent
- Delayed calling of object/user handlers
  - Use postEvent in place of sendEvent, and add an extra parameter after the object handle – this defines the delay in seconds (see Shrapnel example soon)

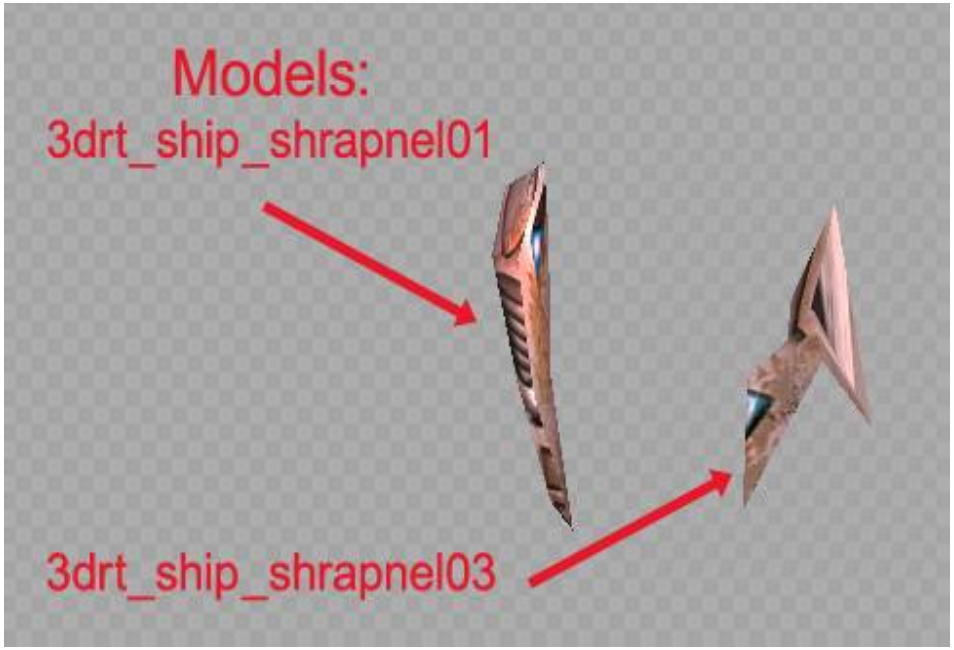
Joypad/Accelerometer example

- nControlOption is a member variable of SpaceShooterMainAI which defines the player's control preference (virtual joystick, accelerometer, etc.)

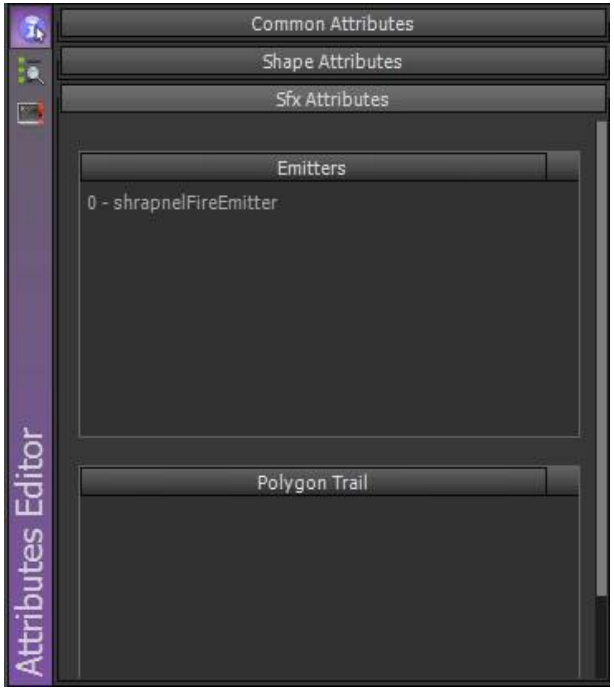
```
--------------------------------------------------------------------------------
function SpaceShooterMainAI.onJoypadMove ( nJoypad, nPart, nAxisX, nAxisY, nAxisZ )
--------------------------------------------------------------------------------

    if ( this.nControlOption ( ) == 2 ) then
        local s = this.hPlayerShip ( )
        if ( s ) then
            if ( nAxisX<0 ) then
                local d = math.clamp ( -50*nAxisX, 1, 10 )
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetXVelocity", d )
            elseif ( nAxisX>0 ) then
                local d = math.clamp ( -50*nAxisX, -10, -1 )
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetXVelocity", d )
            else
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetXVelocity", 0 )
            end
            if ( nAxisY<-0.6 ) then
                local d = -math.clamp ( 30*(nAxisY+0.6), -10, -1 )
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetYVelocity", d )
            elseif ( nAxisY>-0.6 ) then
                local d = -math.clamp ( 60*(nAxisY+0.6), 1, 10 )
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetYVelocity", d )
            else
                object.sendEvent ( s, "SpaceShooterSpaceshipAI", "onSetYVelocity", 0 )
            end
        end
    end

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

Example with AIs: adding shrapnel to an explosion in the Afterburn game

- Create a new AI called 'ShrapnelAI' to manage the setup, lifetime and destruction automatically – attach this to the game object models
- Code: see next slide



ParticleEmitters have been attached to these models at design time (drag emitter onto model) to create firey trails as they spin across the screen



```
-----------------------------------------------------------------------------
function SpaceShooterSpaceshipAI.explode ( )
-----------------------------------------------------------------------------
        local hUser = application.getCurrentUser ( )
        local s = this.getObject ( )
        local x, y, z = object.getTranslation ( s, object.kGlobalSpace )
        local vx = application.getCurrentUserAIVariable ( "SpaceShooterMainAI", "nCamVelX" )
        local vy = application.getCurrentUserAIVariable ( "SpaceShooterMainAI", "nCamVelY" )
        local vz = application.getCurrentUserAIVariable ( "SpaceShooterMainAI", "nCamVelZ" )
        local scale = 1
        local t = application.getLastFrameTime ( ) -- used to decide how fancy to make explosion

        if ( this.bHudArrowVisible ( ) ) then
            local hComp = hud.getComponent ( hUser, this.sHudArrow ( ) )
            hud.setComponentVisible ( hComp, false )
        end

        user.sendEvent ( hUser, "SpaceShooterMainAI", "onShipDestroyed", this.bIsPlayerShip ( ),
                                        this.nShipNum ( ), this.sHudArrow ( ) )

        user.sendEvent ( hUser, "SpaceShooterMainAI", "onMakeExplosion",
                        "SmallFireExplosionEmitter", x, y, z, vx, vy, vz, 3, 1.15, scale )
        user.sendEvent ( hUser, "SpaceShooterMainAI", "onMakeExplosion",
                        "SmallDustExplosion3DEmitter", x, y, z, vx, vy, vz, 1, 1, scale )
        if ( t < 0.0333 ) then -- 1/30 sec
            user.sendEvent ( hUser, "SpaceShooterMainAI", "onMakeExplosion",
                       "SmallFireExplosionEmitter", x, y, z, vx, vy, vz, 3.5, 1, scale )
        end

        -- shrapnel
        if ( t < 0.0333 ) then -- 1/30 sec
            local numPieces = scale * 3
            if ( t < 0.02 ) then -- 1/50 sec
                numPieces = scale * 5
            end
            local nScaleMin = scale/numPieces
            local nScaleMax = nScaleMin*3
            local hScene = application.getCurrentUserScene ( )
            for i=1, numPieces do
                local sModel
                if ( math.mod ( i,2 ) == 0 ) then
                    sModel = "3drt_ship_shrapnel01"
                else
                    sModel = "3drt_ship_shrapnel03"
                end
                local p = scene.createRuntimeObject ( hScene, sModel )
                object.setTranslation ( p, x, y, z, object.kGlobalSpace )
                object.addAIModel ( p, "ShrapnelAI" )
                object.sendEvent ( p, "ShrapnelAI", "onSetup", nScaleMin, nScaleMax,
                    vx-6, vx+6, vy-6, vy+6, vz-6, vz+6, 1.0, 3.0, false )
            end
        end
```

SpaceShooterSpaceshipAI is attached to each spaceship object after it is spawned into the game

Calling user-AI handlers to spawn particle emitters for explosion (flame, dust)

Only do this if we have a good framerate!

Spawn a number of 3D shrapnel objects, attach ShrapnelAI to each, and call the onSetup handler on it (onSetup is a custom handler I wrote)

ShrapnelAI: see next slide

ShrapnelAI - 'fire and forget' approach

```
-----------------------------------------------------------------------------
function ShrapnelAI.onSetup ( nScaleMin, nScaleMax, nVelXMin, nVelXMax, nVelYMin, nVelYMax,
            nVelZMin, nVelZMax, nLifetimeMin, nLifetimeMax, bGravity )
-----------------------------------------------------------------------------

    local o = this.getObject ( )
    object.setScale ( o, math.random ( nScaleMin, nScaleMax ), math.random ( nScaleMin, nScaleMax )
                                    , math.random ( nScaleMin, nScaleMax ) )
    dynamics.createBoxBody ( o, 1, 1, 1 )
    dynamics.enableGravity ( o, bGravity )
    dynamics.enableDynamics ( o, true )
    dynamics.setAngularDamping ( o, 0 )
    dynamics.setLinearDamping ( o, 0 )

    local vx = math.random ( nVelXMin, nVelXMax )
    local vy = math.random ( nVelYMin, nVelYMax )
    local vz = math.random ( nVelZMin, nVelZMax )
    dynamics.setLinearVelocity ( o, vx, vy, vz, object.kGlobalSpace )
    object.translate ( o, vx/5, vy/5, vz/5, object.kLocalSpace )

    local rx = math.random ( -7, 7 )
    local ry = math.random ( -7, 7 )
    local rz = math.random ( -7, 7 )
    dynamics.setAngularVelocity ( o, rx, ry, rz, object.kGlobalSpace )
    object.rotate ( o, rx, ry, rz, object.kLocalSpace )

    local nLifetime = math.random ( nLifetimeMin, nLifetimeMax )
    object.postEvent ( o, nLifetime*0.4, "ShrapnelAI", "onFade", 0.5 )
    object.postEvent ( o, nLifetime*0.8, "ShrapnelAI", "onFade", 0.25 )
    object.postEvent ( o, nLifetime, "ShrapnelAI", "onDestroy" )

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

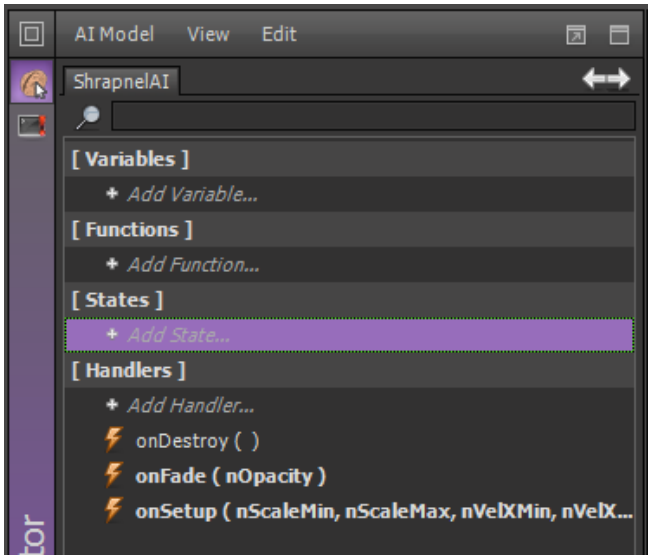
Randomise its scale

Set it under physics control

Randomise its linear velocity

Randomise its angular velocity

Schedule its self-destruction as a delayed call to onDestroy (also fade it out before that)




```
function ShrapnelAI.onFade ( nOpacity )
-----------------------------------------------------------------------------

    shape.setMeshOpacity ( this.getObject ( ), nOpacity )

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

```
function ShrapnelAI.onDestroy ( )
-----------------------------------------------------------------------------

    scene.destroyRuntimeObject ( application.getCurrentUserScene ( ), this.getObject ( ) )

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

Shiva: General Overview

The Shiva IDE

Overview of the Shiva API

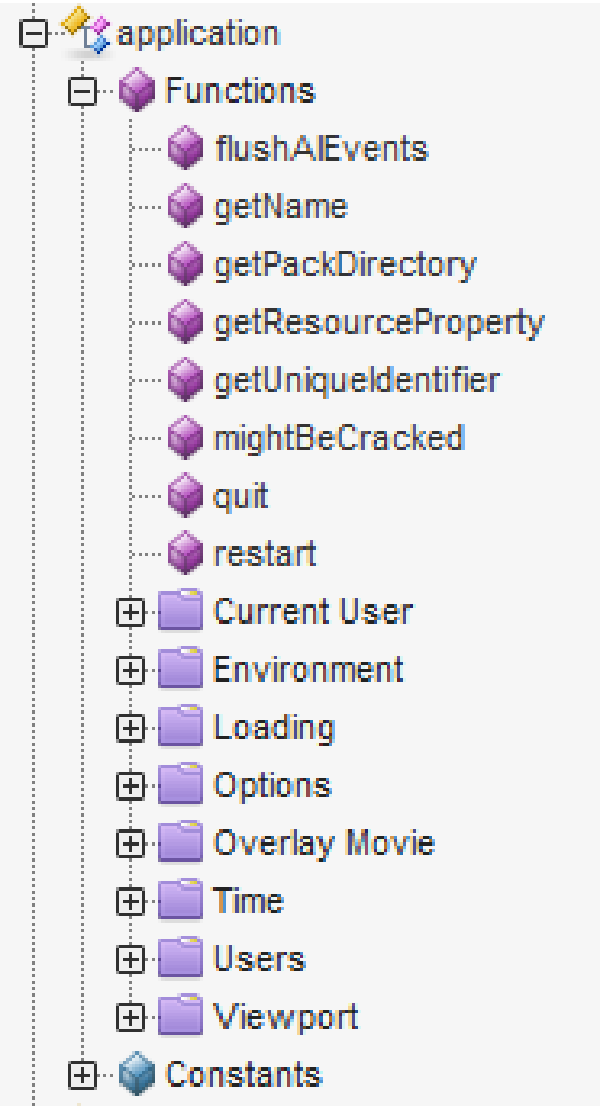
Flow of Control

Key game/API entities

Miscellaneous useful/important topics

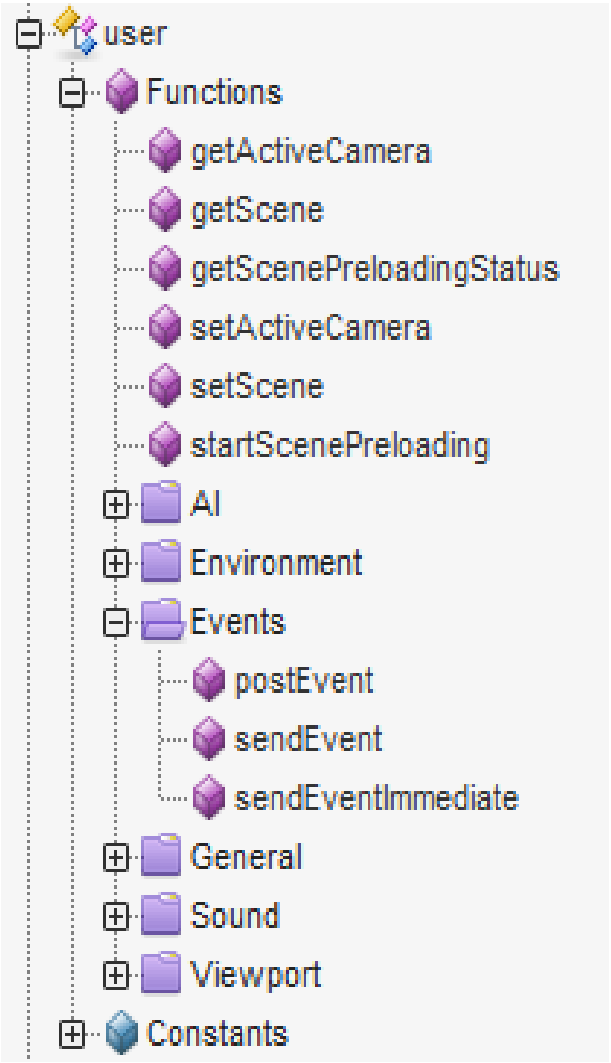
The Application Object

- There is always one application object
- Primary uses:
  - Obtaining the user object
  - Obtaining the user's AI state
  - Reading/writing variables belonging to 'user main' AIs
  - Obtaining/switching the user's current camera object
  - Obtaining the user's current scene object
  - Reading/writing persistent variables (stored on disk/flashdrive/webserver)
  - Reading/writing various Shiva options (e.g. screen orientation)



The User Object

- In a single-player game, there is always one user object
- Many of the user object's SDK methods are duplicated in the application object
- Additionally, the user object can call event handlers on a User Main AI, either immediately (asynchronously with 'send', synchronously with 'sendImmediate') or at a predefined delay ('post')

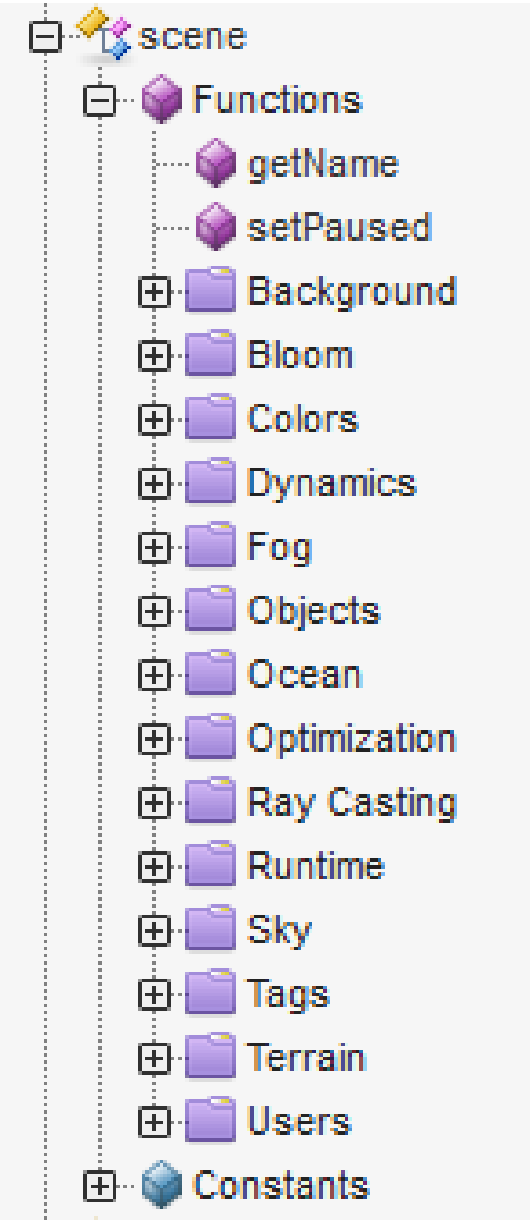


The Scene

- The scene represents everything that's currently 'live' in the visible game (mostly renderable 3D objects, though also some invisble dummy/helper objects)
- There is always one current scene while the game is running
- Scenes can generally be set up at design time using drag-drop, or at runtime by instantiating/deleting objects programmatically
- Which approach is appropriate probably depends on the type of game, i.e. are the levels in the game unique and content-rich, or are they generic and algorithmically produced

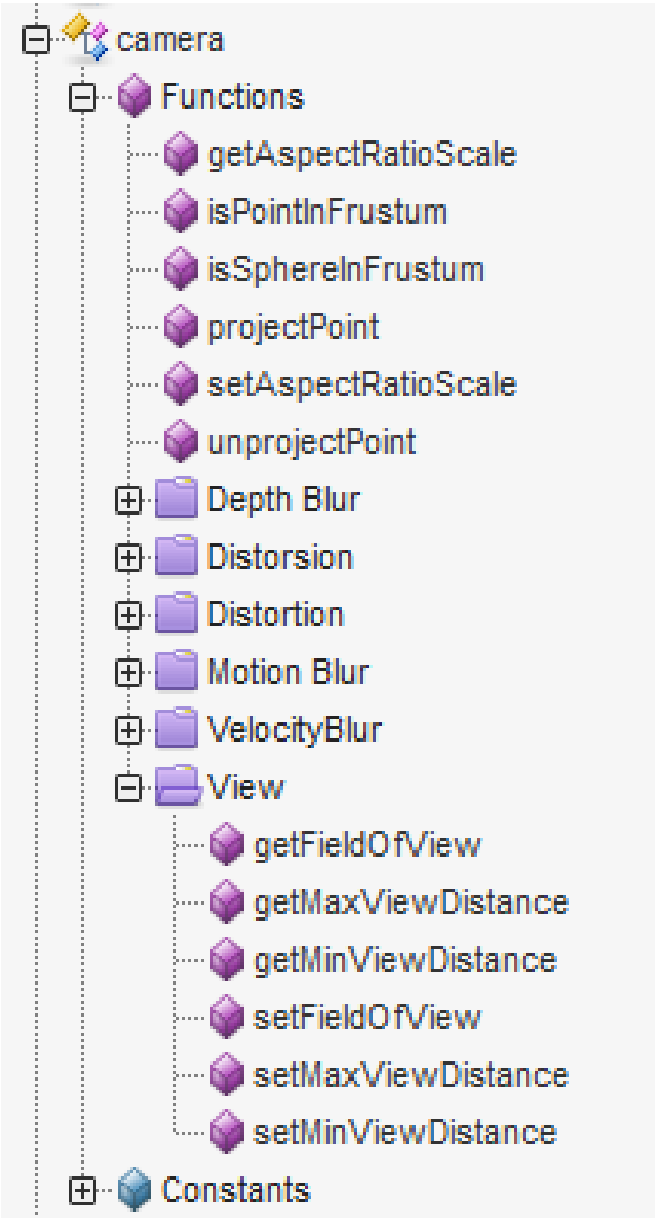
The Scene

- Primary uses:
  - Controlling the background image and/or skybox images
  - Controlling various rendering options (bloom, fog, ambient colors, optimizations)
  - Setting options in the physics (Dynamics) system
  - Iterating through 3D game objects
  - Ray casting (colliders/sensors)
  - Runtime mesh combining
  - Tagging 3D game objects and finding them again
  - Iterating through users in the scene (only for multiuser games..)



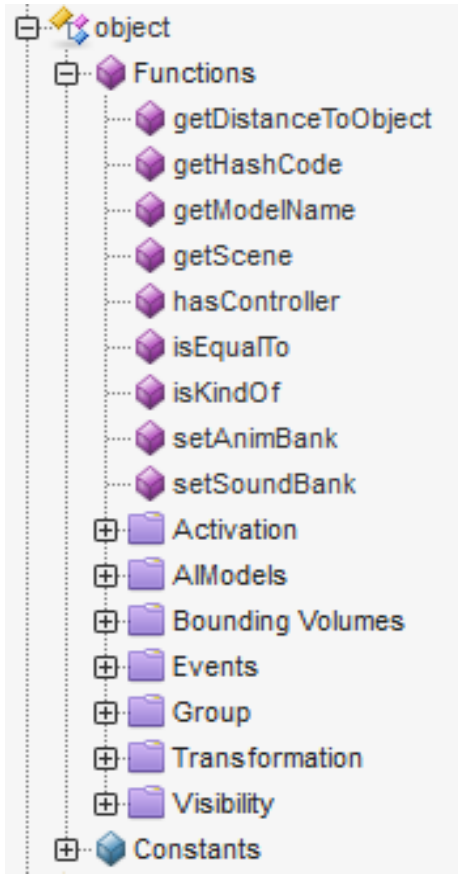
The Camera

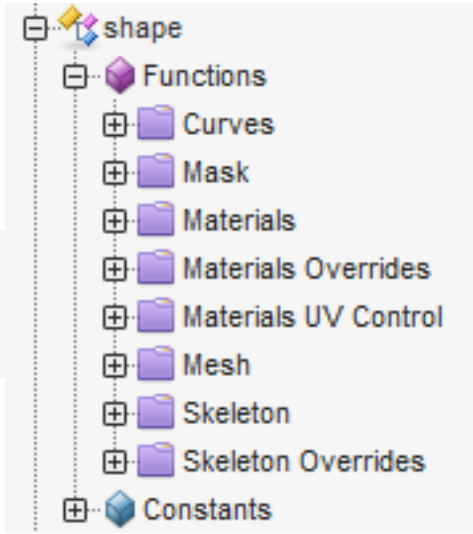
- There can be one or more cameras in a scene, though typically only one is active at a time (defining what's rendered into the viewport)
- Since cameras (conceptually) exist in the 3D world, many of the same things can be done to them that can be done to regular 3D objects, e.g. move them, rotate them etc. These things are done via the 'object' part of the SDK rather than the 'camera' part (see next slide)
- Useful camera-SDK methods enable:
  - Testing visibility of a 3D position
  - Projecting of a 3D position onto the 2D viewport (and vice versa)
  - Setting camera parameters (FoV etc.)
  - Special effects (depth blur etc.)



3D Game Objects

- 3D game objects include (mostly) visible 3D objects (e.g. imported from Blender), and also cameras and invisible 'dummy' or 'helper' objects (which are used for various purposes)
- The 'object' part of the SDK applies to all of these, but the 'shape' SDK and 'mesh' SDK are only for those which have a 3D model associated with them
- Useful methods of the object SDK enable:
  - Attaching AIs to objects
  - Reading/writing variables belonging to attached AIs
  - Determing bounding boxes/spheres
  - Calling event handlers (with or without a delay) on attached AIs
  - Parenting/grouping objects (with or without separate coordinate systems)
  - 3D world transformations (translation, rotation, scale, look at, x/y/z axis calculation etc.)
- The shape SDK lets you change the textures and materials of objects, as well as obtain their meshes
- The mesh SDK is mostly about adding/removing/moving the vertices of 3D objects





Shiva: General Overview

The Shiva IDE

Overview of the Shiva API

Flow of Control

Key game/API entities

Miscellaneous useful/important topics

Physics Vs. Direct Movement

Co-ordinate Systems

HUDs (=GUIs)

Importing Media Assets

Particle Emitters

Data: Runtime & Persistent

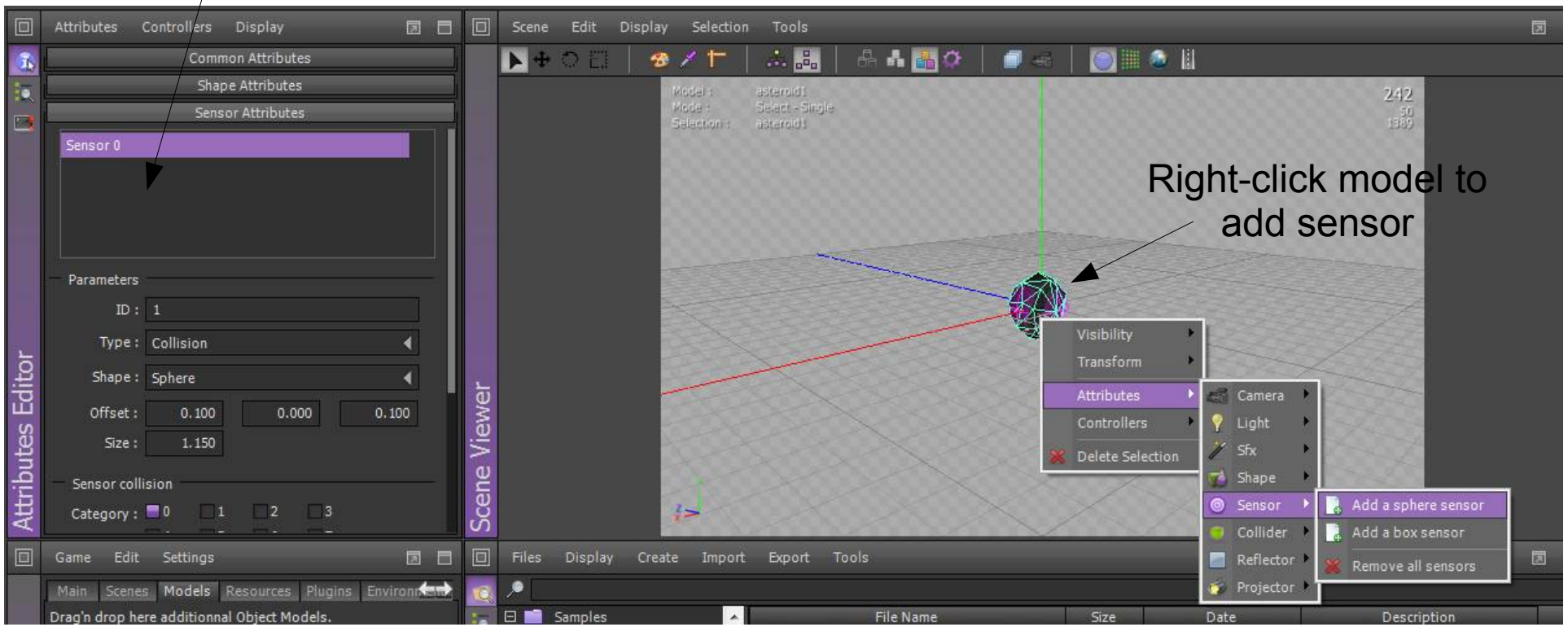
Device Limitations

Physics Vs. Direct Movement

- Direct control of your game objects can be done, using the object API
  - object.translate, object.rotate, object.lookAt, etc.
  - Coded e.g. in the onEnterFrame handler
- In many cases, letting the physics engine control them is better – more CPU efficient, integrated rigid-body collisions
  - dynamics.createBoxBody, dynamics.setMass, dynamics.setLinearVelocity, etc.
  - 'Fire & Forget' (slightly strange at first..)
  - Use with sensors & colliders
  - Shiva uses the ODE physics engine, by the way..

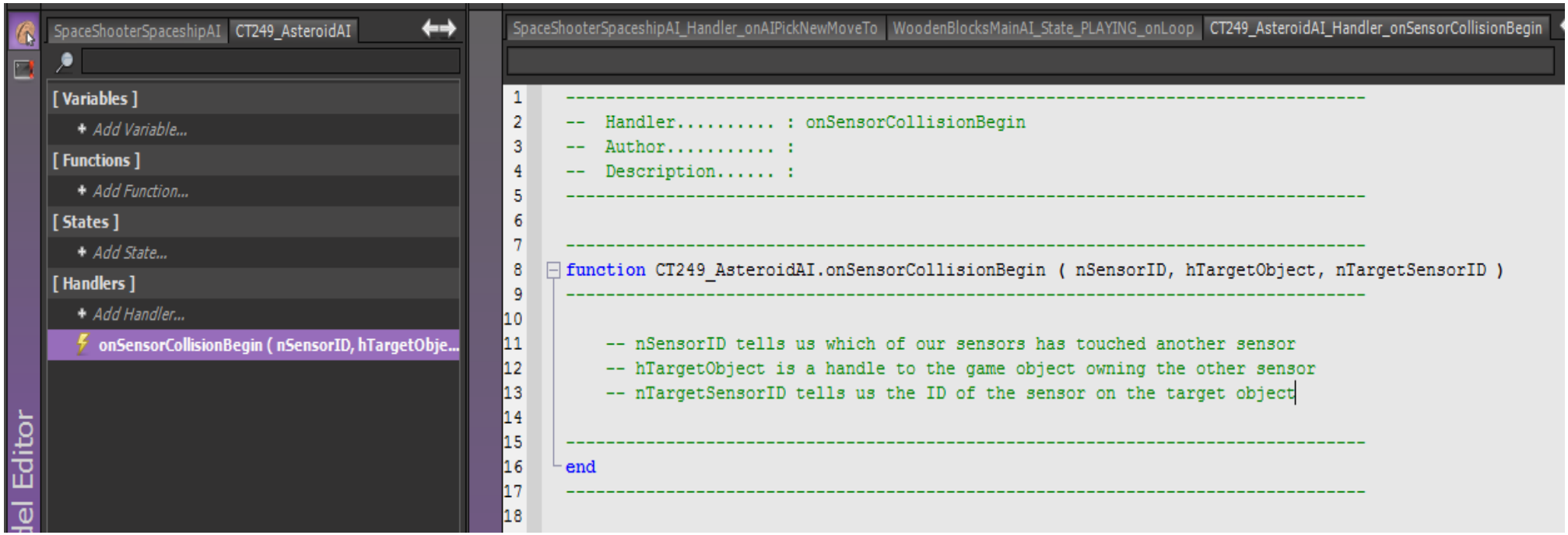
Adding a sensor

Sensor properties here



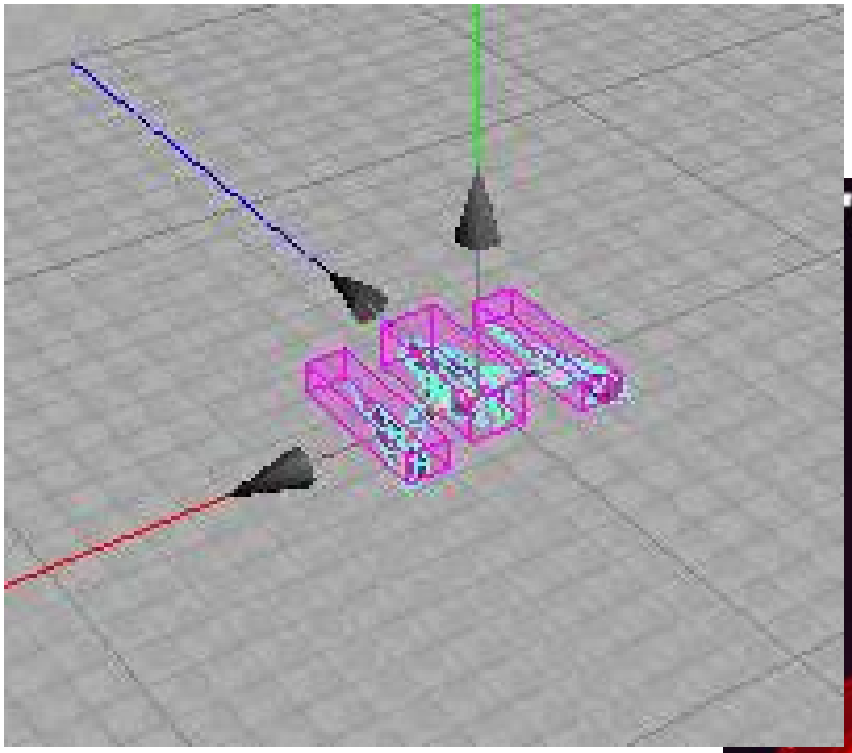
When an object passes inside a sensor, if the object has an AI attached, then the onSensorCollisionBegin handler is executed (see next slide)

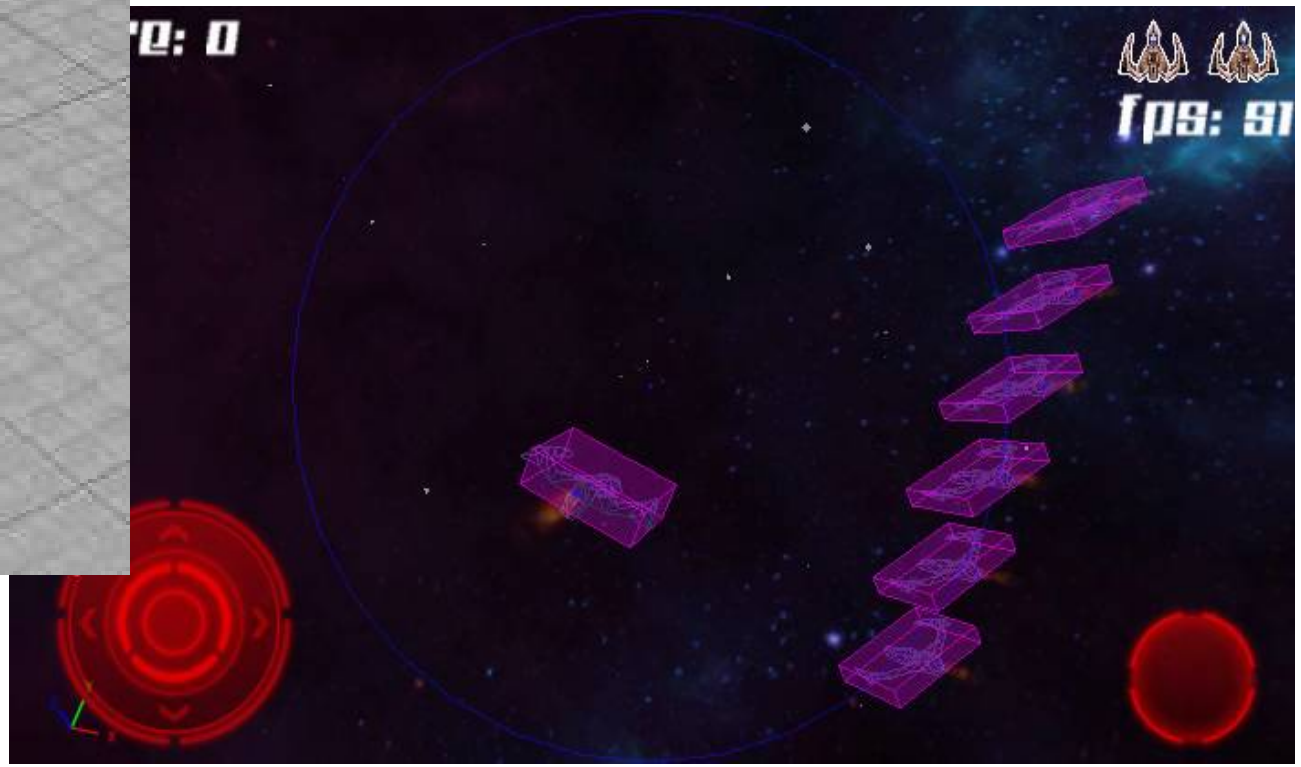
Handling a sensor collision



Co-ordinate Systems

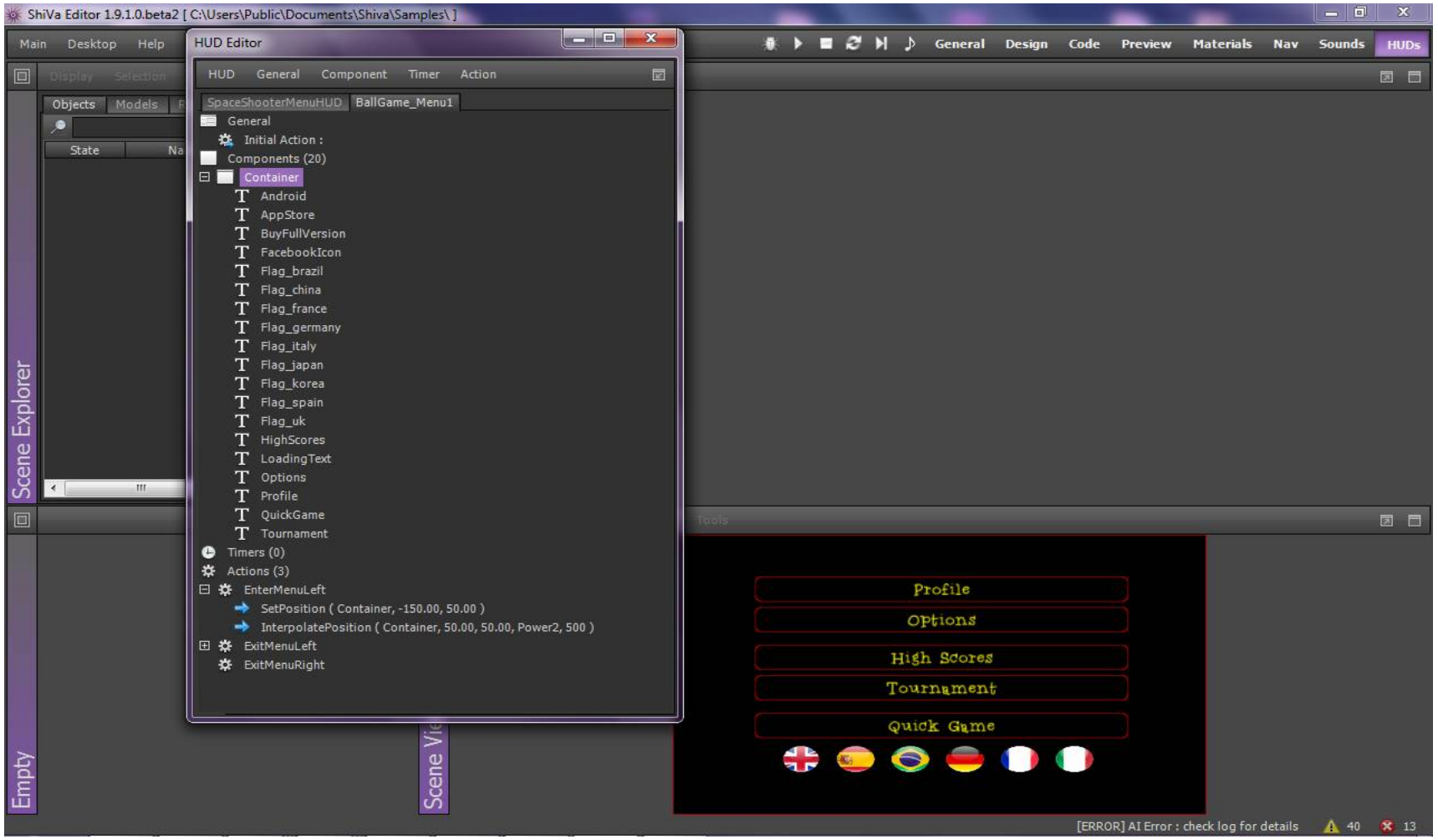
- object.kGlobalSpace .. the world's root co-ordinate system
- object.kLocalSpace.. each object's local co-ordinate system





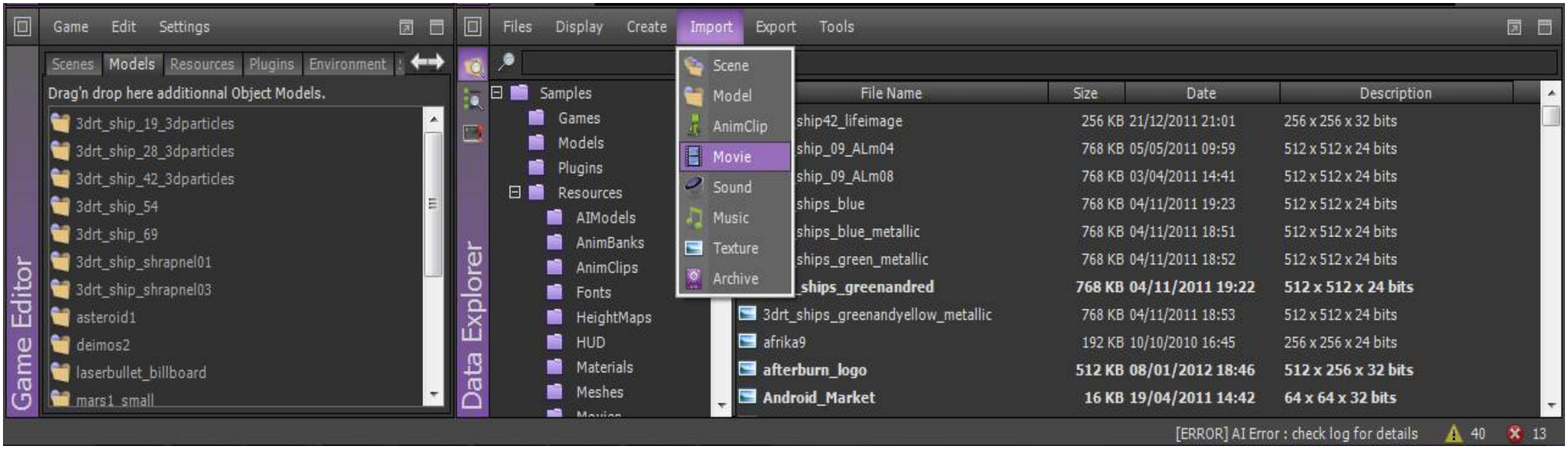
HUDs (=GUIs)

- Drag-drop GUI builder with integrated animation system
- Use hud.callAction to initiate an animation



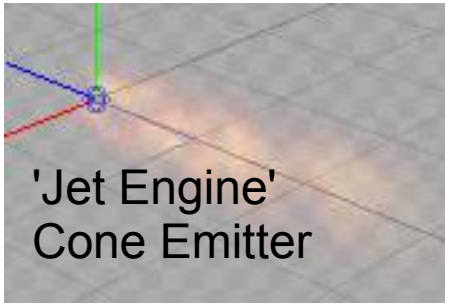
Importing Media Assets

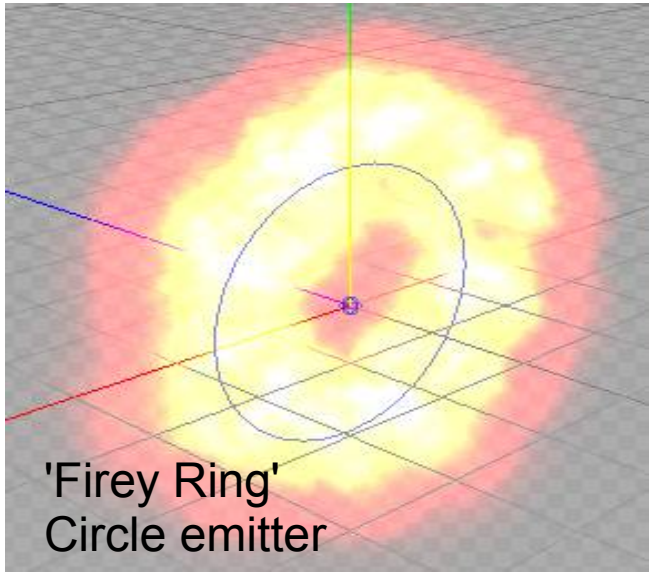
- The Data Explorer module maintains a library of media assets that you can drag-drop into your games via the Game Editor module
- Import menu to add 3d models, animations (collada format), textures/images (jpg, png), sounds (ogg, wav, mp3) etc.

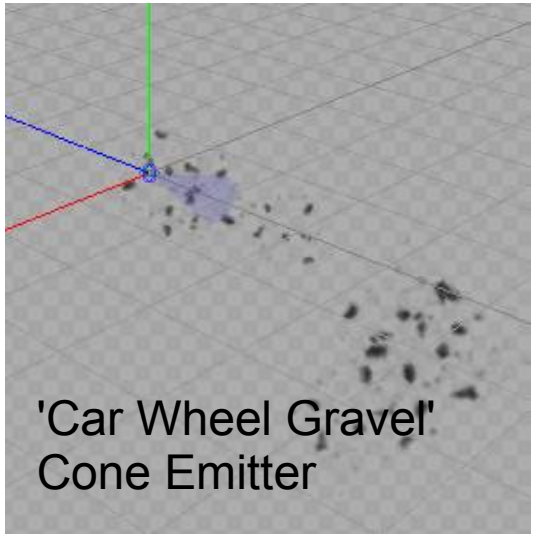


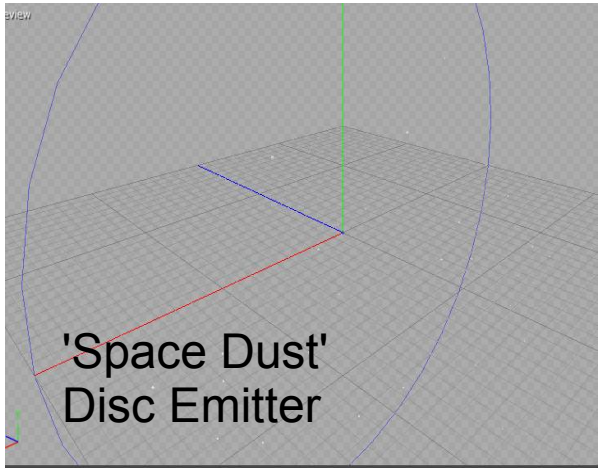
Particle Emitters

- Very computationally efficient way to do special effects – flames, raindrops, explosions, etc.
- Particles are textured billboards emitted from a source, with physical characteristics and size/movement/colour/opacity changes over their lifetime
- Very flexible with some thought..









Data: runtime & persistent

- Runtime data is attached to the user or to game objects as member variables of their attached AIs
- There are two ways of persisting data to disk/webserver
- CurrentUserEnvironment variables, controlled by the Application object:
  - application.getCurrentUserEnvironmentVariable
  - application.setCurrentUserEnvironmentVariable
  - application.loadCurrentUserEnvironment
  - application.saveCurrentUserEnvironment
  - http://www.stonetrip.com/developer/520-local-environment
- xml objects, loaded/saved as xml files, either locally or via HTTP POST to webserver; searchable/editable via the xml API in Shiva
  - xml.send, xml.receive
  - http://www.stonetrip.com/developer/377-xml-manipulation

Device Limitations

- A phone is a limited device, not as powerful as a PC
- OpenGLES is limited in specific ways versus its big brother OpenGL
  - You must minimise draw calls, even more than polygon count
- Lua script is interpreted, hence inefficient, so you must limit the amount of work you're doing per-frame
  - Clever use of postEvent delayed calls can help (interleave operations on objects over different frames)
  - Let the physics system control as much as possible (since it's compiled and optimised)
- A good idea is to scale back special effects based on frame-rate (therefore suits multiple devices simultaneously)

Thank you.. :-)

The Shiva Game Engine

www.stonetrip.com

and web-player demos
of some of my games!

I have posted
today's slides
here!

Sam Redfern

www.it.nuigalway.ie/~sredfern

www.psychicsoftware.com

NUI Galway
OÉ Gaillimh

<<psychic software>>