Squirrel Scripting Best Practices

# Global Access

Use the double colon operator (::) for every global you use, including global function calls, class intantiation, base class referencing and even class extension. This operator skips looking for that symbol in the local scope and looks for it in the global scope, eliminating several hash lookups. This is essential for any functions being called every frame.

function GlobalFunc( )

{

// ...

}

class SubClass extends ::SuperClass

{

canvas = null

constructor( )

{

::SuperClass.constructor( ::GlobalFunc( ) )

canvas = ::AnimatingCanvas( )

// ...

}

}

# Use locals!

Every time you use the dot operator (.) you're going to be doing a hash lookup. Look at the following lines of code:

::GameApp.HudRoot.Invisible = false

::GameApp.HudRoot.SetAlpha( 1 )

if( ::GameApp.GameMode.IsSinglePlayer )

rootMenu = ::SinglePlayerStandardRootMenu( )

else if( ::GameApp.GameMode.IsVersus )

rootMenu = ::HeadToHeadStandardRootMenu( )

else if( ::GameApp.GameMode.IsCoOp )

rootMenu = ::CoOpStandardRootMenu( )

if( !::GameApp.GameMode.IsFrontEnd && ExtraMode )

{

if( ::GameApp.GameMode.IsSplitScreen )

::GameApp.SecondaryPlayer.SpawnCharacter( "max.sigml", "p2" )

::GameApp.FrontEndPlayer.SpawnCharacter( "max.sigml", "p1" )

}

Notice the strings of symbol.symbol.symbol... ? These can be mostly eliminated using locals at the top of the function:

local gameApp = ::GameApp

local hudRoot = gameApp.HudRoot

local gameMode = gameApp.GameMode

hudRoot.Invisible = false

hudRoot.SetAlpha( 1 )

if( gameMode.IsSinglePlayer )

rootMenu = ::SinglePlayerStandardRootMenu( )

else if( gameMode.IsVersus )

rootMenu = ::HeadToHeadStandardRootMenu( )

else if( gameMode.IsCoOp )

rootMenu = ::CoOpStandardRootMenu( )

if( !gameMode.IsFrontEnd && ExtraMode )

{

if( gameMode.IsSplitScreen )

gameApp.SecondaryPlayer.SpawnCharacter( "max.sigml", "p2" )

gameApp.FrontEndPlayer.SpawnCharacter( "max.sigml", "p1" )

}

This reduces the hash lookups in this function by almost half. If this was an OnTick function or something called every frame, the benefit would be significant.

## Storing references

If you are using something using the player profile, the GameMode, the MapType of the current level or whatever, store the reference to it in the object and then use that variable instead. Looking up a class member is much faster than getting it from the global GameApp:

class A

{

profile = null

constructor( player )

{

profile = player.GetUserProfile( )

}

function OnTick( dt )

{

// Bad:

if( ::GameApp.FrontEndPlayer.GetUserProfile( ).GetSetting( ... ) )

// ...

// Bad:

local player = ::GameApp.FrontEndPlayer

local profile = player.GetUserProfile( )

if( profile.GetSetting( ... ) )

// ...

// Better (just storing the profile as part of the class)

if( profile.GetSetting( ... ) )

// ...

}

}

# Arrays

## Allocation & Initialization

The C++ way to create and initialize might be to allocate the whole array at once (if you know the starting size) and then fill in the values. In Squirrel, accessing the array to set a value during initialization is just as fast as pushing onto the end. What you shouldn't do is repeatedly access the array to set a variety of members.

// Bad

displays = array( displayCount, null )

for( local i = 0; i < displays.len( ); ++i )

{

displays[ i ] = ::Gui.Text( )

displays[ i ].SetFontById( FONT\_SIMPLE\_SMALL )

displays[ i ].SetRgba( COLOR\_DIRTY\_WHITE )

displays[ i ].BakeCString( i.tostring( ), TEXT\_ALIGN\_LEFT )

displays[ i ].SetYPos( i \* displays[ i ].LineHeight )

AddChild( displays[ i ] )

}

// Better

displays = [ ]

for( local i = 0; i < displayCount; ++i )

{

local text = ::Gui.Text( )

text.SetFontById( FONT\_SIMPLE\_SMALL )

text.SetRgba( COLOR\_DIRTY\_WHITE )

text.BakeCString( i.tostring( ), TEXT\_ALIGN\_LEFT )

text.SetYPos( i \* text.LineHeight )

displays.push( text )

AddChild( text )

}

## Iterating Arrays

If you didn't know, the easiest and most-easily-readable way to iterate over an array (or a table) is to use foreach:

foreach( i, value in myArray )

// ...

foreach( value in myArray )

// ...

foreach( key, value in myTable )

// ...

foreach( value in myTable )

// ...

This also helps because you automatically have a local reference to the item in the container, which means you don't need to index into the table/array. You really should only be doing a for( local i = 0...) type for-loop if you are actually counting to a specific number.

// Bad:

for( local i = 0; i < myArray.len( ); i += 1 )

{

myArray[ i ].someVar = 42

myArray[ i ].Foo( )

}

// Better:

for( local i = 0; i < myArray.len( ); i += 1 )

{

local obj = myArray[ i ]

obj.someVar = 42

obj.Foo( )

}

// Much Better:

foreach( obj in myArray )

{

obj.someVar = 42

obj.Foo( )

}

# Classes

## Constructors and Members

While it may seem tempting to assign an initial value when defining members of classes, don’t do it. Always set class members to null and assign a default value in the constructor. If you don’t, that value will have undefined behavior.

class SubClass extends ::SuperClass

{

// Kinda bad

someMemberValue = 27

// Very bad

someMemberTable = { }

// Better

someMemberValue = null

someMemberTable = null

constructor( )

{

::SuperClass.constructor( ::GlobalFunc( ) )

// Do this:

someMemberValue = 27

someMemberTable = { }

}

}