How To Implement a Progress Bar

BigWorld Technology 1.9.1. Released 2008.

Software designed and built in Australia by BigWorld.

Level 3, 431 Glebe Point Road
Glebe NSW 2037, Australia
www.bigworldtech.com

Table of Contents

Chapter 1. Introduction

Progress bars are always useful in games and other applications, in order to indicate to the user how long they will need to wait until a long task has finished.

These tasks may be initialising the game, or loading the world around a player when they either log in the game or teleport to another part of the world.

Chapter 2. The Script and GUI Files

Creating a progress bar using BigWorld technology is a straightforward task. In this document, we will implement a progress bar using a GUI script (in Python), which in turn generates the GUI file (in XML format).

2.1. Script for generating the GUI file

For this example, we will be using the FantasyDemo scripts.

Open the `fantasydemo/res/scripts/client/PyGUI.py`, and include the following lines:

```
"""
This class is a simple Progress Bar.
   """
   class ProgressBar( PyGUIBase ):

      factoryString="PyGUI.ProgressBar"

      def __init__( self, component ):
         PyGUIBase.__init__( self, component )

      def setProgress( self, value ):
         pass
```

Progress bar script `PyGUI.py`

The example given in this document uses `PyGUIBase`, which is the base class for GUI components, and is described in file `fantasydemo/res/scripts/client/helpers/PyGUI.py`. The use of this class to implement the progress bar is optional.

The class attribute `factoryString` tells the GUI library which class to instantiate when we load our GUI from the XML file. The XML will contain a `<script>` entry with the value "`PyGUI.ProgressBar`"

The method `setProgress` is called by the user to set the current progress level. Our script will respond by moving the bar to the appropriate place.

2.2. Generating the GUI file

It is easier to create a GUI at runtime if you use the in-game Python console. So, first launch the FantasyDemo executable, then press `Caps Lock+P` to open the Python console. The code below defines the GUI for our progress bar:

```
import GUI
progress = GUI.Window( "maps/gui/gui_bar_back.dds" )
progress.bar = GUI.Simple( "maps/gui/gui_bar.dds" )
progress.bar.clipper = GUI.ClipShader()
progress.height = progress.bar.height = 0.1
progress.materialFX = progress.bar.materialFX = "BLEND"
progress.bar.colour = (128,128,255,255)
GUI.addRoot( progress )
```

Script for generating GUI file — Defining visual elements

The file above defines a bitmapped border component for the bar, and a bitmapped bar component hooked up to a Clip GUI Shader.

Given that the specified border component is the image below:

Border component

And given that the bar component is the image below:

Bar component

Note that we have created the progress bar as a Window. This enables us to reposition the whole progress bar simply by moving the root component.

For example, to move the entire progress bar to the top middle of the screen, we would write:

```
progress.position = (0,0.8,1)
```

Script for generating GUI file — Positioning bar on top middle screen

The final attribute we need to set on the progress bar component is its script. We do this simply by instantiating the appropriate Python class, as illustrated below:

```
from Helpers import PyGUI
progress.script = PyGUI.ProgressBar(None)
```

Script for generating GUI file — Defining bar's script

To generate the progress bar GUI file, hook up the script to the GUI component and save it to disk, as illustrated below:

```
progress.save( "guis/progress_bar.gui" )
```

Script for generating GUI file — Generating the bar's GUI file

2.3. The generated GUI file

The generated file `guis/progress_bar.gui` will look something like this:

```
<progress_bar.gui>

    <WindowGUIComponent>   301410224
        <position>         0.000000 0.000000 1.000000 </position>

        <widthInClip>      true                       </widthInClip>
        <width>            0.500000                   </width>
        <heightInClip>     true                       </heightInClip>
        <height>           0.100000                   </height>
        <colour>           255.0 255.0 255.0 255.0    </colour>
```

```
        <angle>             0                           </angle>
        <flip>              0                           </flip>
        <visible>           true                        </visible>
        <horizontalAnchor>  1                           </horizontalAnchor>
        <verticalAnchor>    1                           </verticalAnchor>
        <textureName>       maps/gui/gui_bar_back.dds   </textureName>
        <materialFX>        1                           </materialFX>
        <tiled>             false                       </tiled>
        <tileWidth>         16                          </tileWidth>
        <tileHeight>        16                          </tileHeight>
        <script>            "PyGUI.ProgressBar"         </script>
        <children>
            <bar>           301291568                   </bar>
        </children>
        <scroll>            0.000000 0.000000           </scroll>
        <minScroll>         0.000000 0.000000           </minScroll>
        <maxScroll>         0.000000 0.000000           </maxScroll>
    </WindowGUIComponent>

    <SimpleGUIComponent>    301291568

        <position>          0.000000 0.000000 1.000000 </position>
        <widthInClip>       true                        </widthInClip>
        <width>             0.500000                    </width>
        <heightInClip>      true                        </heightInClip>
        <height>            0.100000                    </height>
        <colour>            128.0 128.0 255.0 255.0     </colour>
        <angle>             0                           </angle>
        <flip>              0                           </flip>
        <visible>           true                        </visible>
        <horizontalAnchor>  1                           </horizontalAnchor>
        <verticalAnchor>    1                           </verticalAnchor>
        <textureName>       maps/gui/gui_bar.dds        </textureName>
        <materialFX>        1                           </materialFX>
        <tiled>             false                       </tiled>
        <tileWidth>         16                          </tileWidth>
        <tileHeight>        16                          </tileHeight>
        <shaders>
            <clipper>       301215592                   </clipper>
        </shaders>
    </SimpleGUIComponent>

    <ClipGUIShader>         301215592

        <mode>              0                           </mode>
        <value>             1.000000                    </value>
        <speed>             0.000000                    </speed>
        <delay>             0.000000                    </delay>
        <slant>             0.000000                    </slant>
    </ClipGUIShader>

</progress_bar.gui>
```

GUI file `progress_bar.gui`

Now that we know the names of the components, we can implement the GUI script. The script controls the progress bar via its method `setProgress`:

```
def setProgress( self, value ):
    clipper = self.frame.bar.clipper
    if value > clipper.value:
```

```
        self.frame.bar.clipper.value = value
```

Progress bar script file `PyGUI.py` — Controlling the progress

This method simply maps the incoming progress value (between 0.0 and 1.0) to the clip value on the GUI Clip shader. The Clip shader will smoothly clip the component to the new value over a set amount of time.

Chapter 3. Implementing Progress Bars

Once the bar's GUI file has been generated, you can incorporate it to your game, both at the loading screen or during the transition due to a teleport.

3.1. Using the progress bar

Once the GUI file has been created, it is a simple task to display it in your game, and update its value.

3.1.1. Displaying the progress bar

The first step to have your game display the bar is to associate it with its GUI file, as illustrated below:

```
import GUI
progressBar = GUI.load( "guis/progress_bar.gui" )
```

And display it via method `GUI.addRoot`:

```
GUI.addRoot( progressBar)
```

If you derived your script from the class `PyGUIBase`, then you can also display the bar via the script's active method, as illustrated below:

```
progressBar.script.active(1)
```

3.1.2. Updating the progress bar

In order to update the progress bar, all you have to do is call the script method `setProgress`:

```
progressBar.script.setProgress( 0.18 )
```

The script will perform the relevant actions on the underlying GUI component.

3.2. Creating a progress bar for the game startup

Built into the client engine is a progress bar for tracking start-up. This can be used directly in your game, or just sued as example code.

3.2.1. To enable the GUI-based application progress bar

Place the following entry in file `<res>/resources.xml` (where `<res>` is the first folder specified in environment variable `BW_RES_PATH`):

```
<system>
<loadingScreenGUI> folder/loading_screen.gui </loadingScreenGUI>
</system>
```

`<res>/resources.xml`—Defining file for the loading screen

3.2.2. What must my GUI script support?

The application will invoke your GUI script as it initialises. The loading screen/progress bar file must have the following interface:

```
def setProgress( self, value ):
    #change the progress bar here
    pass
def addMessage( self, str ):
    #display a loading message on-screen
    pass
```

Loading screen/progress bar file `folder/loading_screen.gui`

3.3. Creating a player teleport progress bar

There is a method called `spaceLoadStatus`, which returns the percentage of the world that has been loaded. This feature can be used to update a progress bar when the player is teleporting to another part of world.

The method's signature is illustrated below:

```
BigWorld.spaceLoadStatus( distance )
```

To link your progress bar up to the chunk loading, you will have to poll this method, as illustrated below:

```
def progressCheck( self, endTime ):
    finished = (endTime < BigWorld.time())
    if not finished:
        status = BigWorld.spaceLoadStatus( self.distance )
        finished = (status > 0.95)
    if finished:
        self.setProgress(1.0)
        self.component.fader.value = 0.0
        if self.callbackFn:
            BigWorld.callback( self.component.fader.speed, self.callbackFn )
        else:
            self.setProgress( status )
            BigWorld.callback( self.checkRate, Functor( self.progressCheck,
 endTime ) )
```

Updating the progress bar with chunk loading status

Chapter 4. Troubleshooting

4.1. My game's personality script takes a while to initialise

The application will call method `setProgress` until it reaches 100%. Once this value is reached, the personality script `init` method is called.

If you want to seamlessly incorporate the loading of the application with the loading of the personality script, then make your loading bar GUI script rescale the incoming values.

For example in `fantasydemo/res/scripts/client/Helpers/ProgressBar.py`, here are the code snippets that perform this task:

```
def setMinMax( self, min, max ):
    self.min = min
    self.max = max

def setProgress( self, value ):
    #remap value into the correct range
    range = self.max - self.min
    value = self.min + value*range
    self.component.bar.clipper.value = value

def onLoad( self, section ):
    self.phase1Ratio = section.readFloat( "phase1Ratio", self.phase1Ratio )

def startPhase( self, num ):
    if num == 1:
        self.setMinMax( 0.0, self.phase1Ratio )
    else:
        self.setMinMax( self.phase1Ratio, 1.0 )
```

Incorporating the loading of application with loading of personality script

FantasyDemo's loading bar loads a value from the `.gui` file called `phase1ratio`. This is set to 0.66, meaning that the application's initialisation goes up to 66% of the progress bar. Therefore 33% of the time will be used by the personality script to load chunks.