

Civilization 5 UI Tutorial

Part 5, Feedback

Version: 1.0

Status: Draft

Date: 18 May 2012

Author: William Howard

Contents

Overview	3
Create a Feedback Mod	3
Feedback 1 - MouseOvers and ToolTips	3
Feedback 2 - Progress Bars	8
Feedback 2b - Texture Progress Bars.....	Error! Bookmark not defined.
Feedback 3 - Meters	Error! Bookmark not defined.
Feedback 4 - Percent Meter.....	Error! Bookmark not defined.
Summary	Error! Bookmark not defined.

Overview

The objective of this tutorial is to cover the Civ 5 User Interface (UI) elements and controls for providing feedback - namely ToolTips, Bars and Meters.

The tutorial takes the practical approach - first we will write some "code", then we will look at what it does. At each step we will have working UI dialogs - which may or may not in themselves be useful - but which can be used as a working starting point for your own UI mods.

This tutorial assumes you have used ModBuddy and FireTuner to create your own simple non-UI mods, and also that you have a basic understanding of XML. Some Lua will be needed, but that will be explained in each step.

All the code in this tutorial can be downloaded from the in-game ModHub as "Test - UI Tutorial - 5 Feedback" found under the "Other" category.

So, to start we need a ModBuddy project ...

Create a Feedback Mod

Using ModBuddy, create a new mod called "Test - Feedback" (or some such).

To this mod add the two folders "UI" and "XML". In the UI folder create the two files "Feedback.xml" and "Feedback.lua" (delete the standard content added to these files). In the XML folder create the file "FeedbackText.xml" (you can leave the standard content in this file).

In the mod's properties, on the "Mod Info" tab, uncheck "Affects Saved Games". On the "Actions" tab, add an "On Mod Activated - Update Database" entry for "XML/FeedbackText.xml". On the "Content" tab, add an "InGameUIAddin" for "UI/Feedback.xml".

Save the project.

Feedback 1 - MouseOvers and ToolTips

Sometimes there just isn't enough space on the screen to display all the information you need to!

Add the following to the "UI/Feedback.xml" file

```
<?xml version="1.0" encoding="utf-8" ?>
<Context>
  <Instance Name="TabItem">
    <Container ID="Tab" ToolTipType="TooltipTypeTopPanel"
    ConsumeMouse="1" Anchor="R,T" Size="33,29">
      <HideOnMouseOver>
        <Image Anchor="L,T" Size="33,29" Texture="MeterBarFrame.dds">
          <Box Anchor="L,T" Size="20,21" Offset="13,4"
          Color="Black,255">
            <Label ID="TabIn" Anchor="L,C" Offset="-1,-3"
            ColorSet="Beige_Black_Alpha" Font="TwCenMT22"/>
          </Box>
        </Image>
      </HideOnMouseOver>
    </Container>
  </Instance>
</Context>
```

UI Tutorial - Feedback

```
        </Box>
    </Image>
</HideOnMouseOver>

    <ShowOnMouseOver>
        <Image Anchor="L,T" Size="163,29" Offset="-130,0"
Texture="MeterBarFrame.dds">
            <Box Anchor="L,T" Size="150,21" Offset="13,4"
Color="Black,255">
                <Label ID="TabOut" Anchor="L,C" Offset="10,-1"
ColorSet="Beige_Black_Alpha" Font="TwCenMT22"/>
            </Box>
        </Image>
    </ShowOnMouseOver>
</Container>
</Instance>

    <Stack ID="TabStack" Anchor="R,T" Offset="0,105"
StackGrowth="Bottom" Padding="4"/>
</Context>
```

Add the following to the "UI/Feedback.lua" file

```
local g_TopPanelToolTip = {}
TTManager:GetTypeControlTable("TooltipTypeTopPanel",
g_TopPanelToolTip)

function ByScore(iPlayerA, iPlayerB)
    local iValueA = Players[iPlayerA]:GetScore()
    local iValueB = Players[iPlayerB]:GetScore()

    if (iValueA ~= iValueB) then
        -- Player with highest score comes first
        return iValueA > iValueB
    else
        -- otherwise player with lowest id comes first
        return iPlayerA < iPlayerB
    end
end

function GetMetMajorCivs(fnSort)
    local players = {}

    local iActivePlayer = Game.GetActivePlayer()
    local pActiveTeam = Teams[Game.GetActivePlayer()]

    for iPlayer = 0, GameDefines.MAX_MAJOR_CIVS do
        local pPlayer = Players[iPlayer]
        if (pPlayer:IsEverAlive()) then
```

UI Tutorial - Feedback

```
        if (iPlayer == iActivePlayer or
pActiveTeam:IsHasMet(pPlayer:GetTeam())) then
            table.insert(players, iPlayer)
        end
    end
end

if (type(fnSort) == "function") then
    table.sort(players, fnSort)
end

return players
end

function GetScoreToolTip()
    local sToolTip = ""
    local sPrefix = ""

    for _, iPlayer in ipairs(GetMetMajorCivs(ByScore)) do
        local pPlayer = Players[iPlayer]
        local sCiv = Locale.ConvertTextKey(GameInfo.Civilizations[
pPlayer:GetCivilizationType()].ShortDescription)
        local sEra = Locale.ConvertTextKey(GameInfo.Eras[
pPlayer:GetCurrentEra()].Description)
        local iScore = pPlayer:GetScore()

        sToolTip = string.format("%s%i:
[COLOR_POSITIVE_TEXT]%s[ENDCOLOR] (%s)", sToolTip, sPrefix, iScore,
sCiv, sEra)
        sPrefix = "[NEWLINE]"
    end

    return sToolTip
end

function ScoreTipHandler()
    g_TopPanelToolTip.TooltipLabel:SetText(GetScoreToolTip())
    g_TopPanelToolTip.TopPanelMouseover:SetHide(false)
    g_TopPanelToolTip.TopPanelMouseover:DoAutoSize()
end

function Init()
    local tabScore = {}
    ContextPtr:BuildInstanceForControl("TabItem", tabScore,
Controls.TabStack)
    tabScore.Tab:SetToolTipCallback(ScoreTipHandler)
    tabScore.TabIn:SetText("[ICON_CAPITAL]")
    tabScore.TabOut:LocalizeAndSetText("TXT_KEY_TEST_FEEDBACK_SCORE")
end
```

```
Init()
```

Add the following to the "XML/FeedbackText.xml" file

```
<?xml version="1.0" encoding="utf-8"?>
<GameData>
  <Language_en_US>
    <Row Tag="TXT_KEY_TEST_FEEDBACK_SCORE">
      <Text>Score</Text>
    </Row>
  </Language_en_US>
</GameData>
```

Note: For brevity I've excluded the Culture and Science tabs, these are included in the downloadable code.

Save the files and build the mod. Start Civ 5, enable the mod, and load a saved game.



At the top right hand side of the screen you will have a small tab (or three if you've used the downloadable code). Moving the mouse over it will cause it to "fly-out" from the side and a summary of the scores of the Civs you've met so far is displayed. This uses two useful techniques - paired Show/Hide mouse-overs and dynamic tooltips - we'll look at the paired mouse-overs first.

UI Tutorial - Feedback

```
<Container ID="Tab" ToolTipType="ToolTipTypeTopPanel" ConsumeMouse="1"
Anchor="R,T" Size="33,29">
  <HideOnMouseOver>
    <Image Anchor="L,T" Size="33,29" Texture="MeterBarFrame.dds">
      <Box Anchor="L,T" Size="20,21" Offset="13,4" Color="Black,255">
        <Label ID="TabIn" Anchor="L,C" Offset="-1,-3"
ColorSet="Beige_Black_Alpha" Font="TwCenMT22"/>
      </Box>
    </Image>
  </HideOnMouseOver>

  <ShowOnMouseOver>
    <Image Anchor="L,T" Size="163,29" Offset="-130,0"
Texture="MeterBarFrame.dds">
      <Box Anchor="L,T" Size="150,21" Offset="13,4" Color="Black,255">
        <Label ID="TabOut" Anchor="L,C" Offset="10,-1"
ColorSet="Beige_Black_Alpha" Font="TwCenMT22"/>
      </Box>
    </Image>
  </ShowOnMouseOver>
</Container>
```

The Container contains the two versions of the tab - the short one with just the icon and the long one that displays the full text. There's nothing special about them, they are both just an image with a black inner area and some text. The HideOnMouseOver and the ShowOnMouseOver elements provide the "magic". Normally, the short tab is shown (as the mouse is not over the container, so the HideOnMouseOver element that contains it is not active) and the long tab is hidden (as the ShowOnMouseOver element that contains it is also not active). When the user moves the mouse over the container (which is the same size as the smaller tab), both mouse-over elements become active and the visibility of each tab changes - so the short tab is hidden and the long tab becomes visible. Simple! (There is no Lua needed.)

The Container provides the tooltip, but we can't use the ToolTip attribute on the element as the contents of the tooltip needs to change and the attribute can only provide the same fixed (static) text.

The core game engine provides for dynamic tooltips (where we can change the text to be displayed just before it is show to the user) and we need a combination of the ToolTipType="ToolTipTypeTopPanel" attribute and some Lua to achieve this. (There are several "tool tip types" but ToolTipTypeTopPanel is the generic (and hence most useful and common) one.)

In order to change the tooltip text before it is displayed we need to register a tooltip callback

```
function Init()
  local tabScore = {}
  ContextPtr:BuildInstanceForControl("TabItem", tabScore,
Controls.TabStack)
  tabScore.Tab:SetToolTipCallback(ScoreTipHandler)
  tabScore.TabIn:SetText("[ICON_CAPITAL]")
```

```
tabScore.TabOut:LocalizeAndSetText("TXT_KEY_TEST_FEEDBACK_SCORE")
end
```

Before we can set our variable text into the tooltip we need to retrieve the `ToolTipType` control, we only need to do this once, so the code is usually placed at the top of the Lua file

```
local g_TopPanelToolTip = {}
TTManager:GetTypeControlTable("ToolTipTypeTopPanel",
g_TopPanelToolTip)
```

the `ToolTipType` is a special type of Instance, so we retrieve it into a Lua set.

Our tooltip handler constructs the tooltip string to display to the user, and then resizes the tooltip area

```
function ScoreTipHandler()
    g_TopPanelToolTip.ToolTipLabel:SetText(GetScoreToolTip())
    g_TopPanelToolTip.TopPanelMouseover:SetHide(false)
    g_TopPanelToolTip.TopPanelMouseover:DoAutoSize()
end
```

keeping the code to construct the actual text to be displayed to the user in a separate function (`GetScoreToolTip()` in this case) makes it easier to change the text without worrying about messing up the required "tool-tip magic".

`GetScoreToolTip()` just loops over all major civilizations, works out which ones we've met, sorts them by score and then builds the tool-tip string - it's not that complex a piece of Lua, just long!

Feedback 2 - Progress Bars

Sometimes a picture is worth a thousand words!

Change the "UI/Feedback.xml" file to

```
<?xml version="1.0" encoding="utf-8" ?>
<Context>
    <Grid ID="ProgressGrid" Anchor="R,T" Offset="5,75" Size="140,150"
Style="Grid9DetailSix140" ConsumeMouse="1">
        <Label ID="CultureName" String="TXT_KEY_TEST_FEEDBACK_CULTURE"
Anchor="L,T" Offset="20,50" ColorSet="Beige_Black_Alpha"/>
        <Bar ID="CultureBar" Anchor="L,T" Offset="20,65" Size="100,8"
Direction="Right" FGColor="Culture,255" BGColor="Culture,128"/>

        <Label ID="ScienceName" String="TXT_KEY_TEST_FEEDBACK_SCIENCE"
Anchor="L,T" Offset="20,80" ColorSet="Beige_Black_Alpha"/>
        <Bar ID="ScienceBar" Anchor="L,T" Offset="20,95" Size="100,8"
Direction="Right" FGColor="Science,255" BGColor="Science,128"/>
    </Grid>
</Context>
```


and change the "UI/Feedback.lua" file to

```
function UpdateCulture(pPlayer)
    if (Game.IsOption(GameOptionTypes.GAMEOPTION_NO_POLICIES)) then
        Controls.CultureBar:SetHide(true)
    else
        Controls.CultureBar:SetHide(false)

        local fComplete
        if (pPlayer:GetNextPolicyCost() > 0) then
            fComplete = pPlayer:GetJONSCulture() /
pPlayer:GetNextPolicyCost()
        else
            fComplete = 1
        end

        Controls.CultureBar:SetPercent(fComplete)
    end
end

function UpdateScience(pPlayer)
    if (Game.IsOption(GameOptionTypes.GAMEOPTION_NO_SCIENCE)) then
        Controls.ScienceBar:SetHide(true)
    else
        Controls.ScienceBar:SetHide(false)

        local fComplete
        local iCurrentTech = pPlayer:GetCurrentResearch()

        if (iCurrentTech == -1) then
            fComplete = 0
        else
            fComplete = pPlayer:GetResearchProgress(iCurrentTech) /
pPlayer:GetResearchCost(iCurrentTech)
        end

        Controls.ScienceBar:SetPercent(fComplete)
    end
end

function OnChangeEvent()
    local pPlayer = Players[Game.GetActivePlayer()]

    UpdateCulture(pPlayer)
    UpdateScience(pPlayer)
end
Events.SerialEventGameDataDirty.Add(OnChangeEvent)
Events.GameplaySetActivePlayer.Add(OnChangeEvent)
```

save the changes, rebuild the mod, restart Civ 5, re-enable the mod and load a saved game.



At the top right, there is now a panel with two progress bars - one for Culture and one for Science.

I can work out what "18/30" for Culture or "42/75" for Science means in terms of how much I've "done" of the current policy or tech, but "1327/1850" and "2846/3600" takes a little more effort on my part - and I'd rather play the game than get a calculator out! Progress bars permit the same information to be displayed graphically (and if we hook up a dynamic tooltip we can still display "the numbers" if the user wants them.)

Each bar is a single UI element

```
<Bar ID="CultureBar" Anchor="L,T" Offset="20,65" Size="100,8"
Direction="Right" FGColor="Culture,255" BGColor="Culture,128"/>
```

Bars can either be horizontal (Direction="Right") or vertical (Direction="Up") and need to specify a "filled" (or foreground) colour (FGColor="") and an optional "un-filled" (or background) colour (BGColor="") I'd recommend giving both, it saves getting some weird effects when the default background lets the underlying map show through!

To set how much of the bar is filled, we need to use some Lua

```
Controls.CultureBar:SetPercent(fComplete)
```

where fComplete is a number between 0.0 (empty) and 1.0 (full). This is done in the UpdateCulture() method, which also takes into account if the user is playing a scenario where policies are off.

We need to update the bars whenever the user's culture or science changes - we could detect the start of the players turn, but that's not only when culture changes - they could wander into a Goody Hut that gives culture, adopt a policy that decreases the culture needed to complete the next policy, found a new city which will increase the culture needed, or any number of other events - some of which may come from other mods that we don't even know about!

Fortunately the core game engine provides an event (SerialEventGameDataDirty) that we can hook to be notified whenever any player data changes, no matter how it changed

```
function OnChangeEvent()  
    local pPlayer = Players[Game.GetActivePlayer()]  
  
    UpdateCulture(pPlayer)  
    UpdateScience(pPlayer)  
end  
Events.SerialEventGameDataDirty.Add(OnChangeEvent)  
Events.GameplaySetActivePlayer.Add(OnChangeEvent)
```

The bold line is us being "hot-seat friendly", and updating the progress bars when the current (human) user changes.

Continued in Part 2