Options Menu

creationkit.com/index.php

Contents

Overview

Using Show - Message, it is possible to make an options menu with any number of buttons and/or levels. This is enabling as one can maintain but a single plugin with an options menu offering multiple configuration setting rather than necessitating multiple versions. In these examples, we'll use apparel items and a book, but a menu can be prompted and managed in a number of ways. First, create a message(s) form(s) and add/fill the buttons with the options you'd like to offer. Note that no more than ten buttons can be in a message box and that the button indices are offset by one such that the first option's index is 0 and not 1. If offering a lot of options, it's best to think ahead regarding how you want to organize your options, making the message forms first, then plugging them into the script.

Examples

For the first examples, we'll have only three options: "Mage", "Thief", and "Warrior". An options menu can be accessed many ways, a few of which will be demonstrated below.

- First, we'll attach the script to an unplayable armor item. When the item is added to the player, the menu will be prompted and will exit as soon as a button is selected, executing the appropriate code right after the token is silently removed.

```
ScriptName OptionsMenuScript Extends ObjectReference

Actor Property PlayerREF Auto
Armor Property MenuARMO Auto
Message Property OptionsMESG Auto

Event OnContainerChanged(ObjectReference akNewContainer, ObjectReference
akOldContainer)
 If akNewContainer == PlayerREF ; Only the player
  Int iButton = OptionsMESG.Show() ; Shows your menu.
  PlayerREF.RemoveItem(MenuARMO, 1, True) ; Silently remove token. 'Self' does not
work in this context, thus the property
  If iButton == 0  ; Mage
   Debug.Notification("Mage selected")
  ElseIf iButton == 1 ; Thief
   Debug.Notification("Thief selected")
  ElseIf iButton == 2 ; Warrior
   Debug.Notification("Warrior selected")
  EndIf
 EndIf
EndEvent
```

- Next, we'll show the same menu, but prompt it with a spell given to the player. Note the similarities of these examples and that code can be compartmentalized by creating custom functions. The Menu function below could be moved to another script of a different type, meaning you can offer access to your menu however you see fit.

```
ScriptName OptionsMenuScript Extends ActiveMagicEffect

Actor Property PlayerREF Auto
Message Property OptionsMESG Auto ; The Message form that configures the menu buttons

Event OnEffectStart(Actor akTarget, Actor akCaster)
 If akCaster == PlayerREF ; Only the player can open the menu
  Menu()
 EndIf
EndEvent

Function Menu(Int aiButton = 0) ; The menu will exit on its own after a selection is
made.
 aiButton = OptionsMESG.Show() ; Shows your menu.
 If aiButton == 0  ; Mage
  Debug.Notification("Mage selected")
 ElseIf aiButton == 1 ; Thief
  Debug.Notification("Thief selected")
 ElseIf aiButton == 2 ; Warrior
  Debug.Notification("Warrior selected")
 EndIf
EndFunction
```

For this example, we'll offer sub-options for each main selection. For a multilevel menu, a function works well. Keep in mind each button can have conditions, so you could hide "Lunch" and "Dinner" if it's time for breakfast or hide "Lobster" if it's not currently available. In this case, to make it repeatable, we'll use a book so the menu will show each time it is read. A book cannot be favorited or hotkeyed, unlike an apparel item. A potion can be hotkeyed, but it will be consumed when used and not remain hotkeyed even if immediately replaced. This example will let the user choose breakfast, lunch, or dinner, then close after one meal is selected. As your options become fleshed out, keep in mind that you can add and use arguments to store information temporarily rather than necessitating declarations of more variables or properties.

```
ScriptName OptionsMenuScript Extends ObjectReference

Message Property MainMenuMESG Auto
Message Property BreakfastMESG Auto
Message Property LunchMESG Auto
Message Property DinnerMESG Auto

Event OnRead()
 Game.DisablePlayerControls(False, False, False, False, False, True) ; Ensure
MessageBox is not on top of other menus & prevent book from opening normally.
 Game.EnablePlayerControls(False, False, False, False, False, True) ; Undo
DisablePlayerControls
 Menu()
EndEvent
```

```
Function Menu(Bool abMenu = True, Int aiButton = 0)
 While abMenu
  If aiButton != -1 ; Wait for input (this can prevent problems if recycling the
aiButton argument in submenus)
   aiButton = MainMenuMESG.Show() ; Main Menu
   abMenu = False ; End the function
   If aiButton == 0 ; Breakfast
    aiButton = BreakfastMESG.Show()
    If aiButton == 0 ; Sweet Roll & Coffee
    ElseIf aiButton == 1 ; Pancakes, Bacon & Eggs
    ElseIf aiButton == 2 ; Chicken Fried Pony Steak
    EndIf
   ElseIf aiButton == 1 ; Lunch
    aiButton = LunchMESG.Show()
    If aiButton == 0 ;  Glazed Turkey Sandwich
    ElseIf aiButton == 1 ; Grilled Ham Sandwich
    ElseIf aiButton == 2 ; Shredded Pony Sandwich
    EndIf
   ElseIf aiButton == 2 ; Dinner
    aiButton = DinnerMESG.Show()
    If aiButton == 0 ; Filet Mignon
    ElseIf aiButton == 1 ; Pony Fajitas
    ElseIf aiButton == 2 ; Lobster
    EndIf
   EndIf
  EndIf
 EndWhile
EndFunction
```

To make a multilevel, looping menu with thirty buttons that will not close until a "Done" button is pressed, use the above method but with an altered Menu() function. Note that you can jump to a given message by specifying the aiMessage argument when calling the function. Sub-options as described in the previous example can be added to the below in the same manner. Theoretically, any number of options can be added with the below structure. By making it conditional, we can check its property values with MessageBox buttons using GetVMScriptVariable and pointing to the placed instance of the item this script is attached to. To ensure the player gets said reference, make a property for the specific reference in another script, and add it to the player with AddItem by passing the reference as akItemToAdd.

```
ScriptName OptionsMenuScript Extends ObjectReference Conditional

Actor Property PlayerREF Auto
Armor Property MenuARMO Auto ; Playable apparel item
Bool Property bFeatureEnabled Auto Conditional ; Toggling of this demonstrated below.
GlobalVariable Property DragonsEnabled Auto ; Toggling of this demonstrated below.
Message Property OptionsMenu00MESG Auto
Message Property OptionsMenu01MESG Auto
Message Property OptionsMenu02MESG Auto
YourQuestScriptName Property QuestScript Auto ; Is Conditional with a Conditional
bQuickening property
```

```
Event OnEquipped(Actor akActor)
 If akActor == PlayerREF
  Game.DisablePlayerControls(False, False, False, False, False, True) ; Momentarily
disable other menus
  PlayerREF.EquipItem(MenuARMO, True, True) ; Prevent unequip/reequip in favorites
until the current menu is resolved
  Utility.Wait(0.01) ; This ensures equipping the token from the favorites menu works
  PlayerREF.UnequipItem(MenuARMO, False, True) ; Silently unequip item
  Game.EnablePlayerControls(False, False, False, False, False, True) ; Undo
DisablePlayerControls
  Menu()
 EndIf
EndEvent

Function Menu(Int aiMessage = 0, Int aiButton = 0, Bool abMenu = True)
 While abMenu
  If aiButton == -1 ; As above, can prevent problems if recycling aiButton
  ElseIf aiMessage == 0
   aiButton = OptionsMenu00MESG.Show()
   If aiButton < 2 ; Toggle script property. Buttons have opposite
GetVMScriptVariable conditions so only the applicable option is given.
    bFeatureEnabled = !bFeatureEnabled ; Set boolean to whatever it is not
    If bFeatureEnabled ; == True
     Debug.Trace("Featured enabled. Set things up.")
    Else ; If bFeatureEnabled == False
     Debug.Trace("Featured disabled. Stop doing stuff and clean up.")
    EndIf
   ElseIf aiButton < 4 ; Toggle quest property. Buttons have opposite conditions
checking the property value with GetVMQuestVariable.
    QuestScript.bQuickening = !QuestScript.bQuickening ; Set boolean to whatever it
is not
    If QuestScript.bQuickening ; == True
     Debug.Trace("Start polling.")
    Else ; If QuestScript.bQuickening == False
     Debug.Trace("Stop polling.")
    EndIf
   ElseIf aiButton < 6 ; Toggle DragonsEnabled. Buttons have opposite conditions as
above, but checking the global's value with GetGlobalValue.
    DragonsEnabled.SetValue((!DragonsEnabled.GetValue() As Bool) As Float) ; If 1,
set to 0. If 0, set to 1
    If DragonsEnabled.GetValue() ; != 0
     Debug.Trace("Dragons enabled.")
    Else ; If DragonsEnabled.GetValue()  == 0
     Debug.Trace("Dragons disabled.")
    EndIf
   ElseIf aiButton == 6
   ElseIf aiButton == 7
   ElseIf aiButton == 8 ; More
    aiMessage = 1
```

```
   ElseIf aiButton == 9 ; Done
    abMenu = False
   EndIf
  ElseIf aiMessage == 1
   aiButton = OptionsMenu01MESG.Show()
   If aiButton == 0
   ElseIf aiButton == 1
   ElseIf aiButton == 2
   ElseIf aiButton == 3
   ElseIf aiButton == 4
   ElseIf aiButton == 5
   ElseIf aiButton == 6
   ElseIf aiButton == 7 ; Back
    aiMessage = 0
   ElseIf aiButton == 8 ; More
    aiMessage = 2
   ElseIf aiButton == 9 ; Done
    abMenu = False
   EndIf
  ElseIf aiMessage == 2
   aiButton = OptionsMenu02MESG.Show()
   If aiButton == 0
   ElseIf aiButton == 1
   ElseIf aiButton == 2
   ElseIf aiButton == 3
   ElseIf aiButton == 4
   ElseIf aiButton == 5
   ElseIf aiButton == 6
   ElseIf aiButton == 7
   ElseIf aiButton == 8 ; Back
    aiMessage = 1
   ElseIf aiButton == 9 ; Done
    abMenu = False
   EndIf
  EndIf
 EndWhile
EndFunction
```

Notes

- Given the buttons in Skyrim are listed from side to side, it is easy to spill over the edges of the user's monitor, particularly if it's a 4:3, in the event either the options are too verbose or there are too many options presented by a single message form. Currently, there's no way to list them from top to bottom as they were in previous Bethesda games. To mitigate this, keep the button text to a minimum and/or make sure to always set up conditions on mutually exclusive buttons to ensure only applicable options are presented.
- To conditionalize buttons using variables declared in your script/quest, use GetVMScriptVariable and GetVMQuestVariable.
- To hide buttons you wish to fill in later, add an impossible condition like 'IsXBox == -1'.

- For debugging purposes, you could configure hidden menu buttons (in the Message forms) that only show when you set a GlobalVariable flag. For instance, create a GlobalVariable called "myDebugFlag" in the CK. Set a condition on one of your menu buttons to be "GetGlobalValue myDebugFlag == 1". In your script, have that button activate your debugging function. Now if you set myGlobalValue to 1 in the CK your button will appear in the menu. Before releasing the mod, remember to set myDebugFlag to 0 to keep the button hidden.
- Conditionalizing MessageBox buttons will not change their indices such that, for instance, button 9 will still execute the "Done" code in the last example even if buttons 0-8 are hidden.
- To learn how to assign user-created messageboxes as values to the message box Properties defined in the above scripts, see the Papyrus tutorial's page on Properties and Functions
- If the Message Property isn't filled in the CK the Show() will always return a 0, **and the Message will not be shown**.
- If the Message is a Notification (without buttons) instead of a Message Box the Show() will return a -1, **in which case you will never be presented with an options menu**.

See Also

Show - Message