Creating beautiful menus in X^3: Terran Conflict

by ScRaT

October 16, 2009

Contents

1 Introduction

Menus are an important aspects of many scripts. They establish a link between the user and the script. That's why I believe that good looking menus are like figureheads for scripts – not necessary, but it determines the first impression you have of a script. By now I assume that you know what arrays are and how they can be read, written and altered.

2 The basics

The most important thing at front: Menus are just arrays. No black magic, but simple arrays. The SE offers various commands, which help creating menus, but always keep in mind, that these commands just create arrays with different formats.
First let's have a quick look on the main ingredients of a menu: The initialization, the headings, the info lines and the menu items.

2.1 Initialization

To initialize a menu you will usually use the command `<RetVar> = create custom menu array`. This command does nothing else that creating an empty array, so you could also use `<RetVar> = array alloc: size = 0`. The advantage of the menu command is, that it is more readable and thus helps maintaining the script later.

2.2 Menu headings

Menu headings can be created using the command `add custom menu heading to array <Value>: title = <Var/String>`. `<Value>` is of course the menu array, you want to add the heading to. Headings will always displayed in the same order they were added. Additionally they will only be displayed, when at least one *menu item* follows directly after them.
Sometimes it can be more handy to create menu headings manually. To do so, you need to create an array of size 2 – index 0 being the integer `-1`, index 1 the text.

2.3 Info lines

Info lines can be handy sometimes and in my opinion they make a menu look more classy, compared to the use of a menu description. Creating info lines is done with the command `add custom menu info line to array <Value>: text = <Var/String>`. They are displayed at the top of the menu in the order they were added to the menu. Creating info lines manually is of course also possible, but not needed normally.

2.4 Menu items

The most important parts of menus are normally the menu items. They are created via the command `add custom menu item to array <Value$_1$>: text = <Var/String> return-value = <Value$_2$>`. `Var/String` is of course the text, `<Value$_1$>` the menu array and `<Value$_2$>` the return value.
The latter are maybe the most important, since they enable you to know, which menu item was selected. Return values can be anything you can imagine. Often they are `'exit'` or any other string, identifying the menu item. Of course they can also be objects like a station or a ship.
Menu items are always displayed in the order they were added. You can simply create one manually by creating an array of size 2 with index 0 being the text and index 1 the return value.

2.5 Example code

Now that we know the most important menu commands we can create a first script, which uses these commands. The following script will let the player choose one of his own ships out of a menu, which is then destroyed.

```
001 $array = get ship array: of race {Player} class/type = null
002 $menu = create custom menu array: heading = 'Your ships'
003 $size = size of array $array
004 while $size
005   dec $size =
006   $ship = $array[$size]
007   $id = $ship -> get ID code
008   $text = sprintf: fmt = '%s (%s)', $ship, $id, null, null, null
009   skip if $ship == [PLAYERSHIP]
010     add custom menu item to array $menu: text = $text returnvalue = $ship
011   = wait 1 ms
012 end
013 $ret = open custom menu: title = 'Ship Destroyer' description = null
     option array = $menu
014 skip if not $ret == -1
015   return null
016 $ret -> destruct: show no explosion = [FALSE]
017 return null
```

Even though this script is *very* simple, I'll explain the important lines concerning menus:

002: Initialization of the menu, directly defining a heading

009: Just to prevent the `[PLAYERSHIP]` from being selected

010: Add a menu item, which returns `$ship`, to the menu

013: Open the menu and save the returnvalue of the selected menu item in the variable `$ret`

014: When a menu is closed via {`ESC`} `$ret` will have the integer value `-1`; if this is the case, the script will be terminated in line 015

016: Destroys the selected ship

3 Digging deeper

Now that you know the basics of creating menus, I'll describe some more elaborate commands and techniques, most of which were added with patch 2.5.

3.1 Columns

Until know the text entries of menu items were only simple strings, which is fine, until you want to use columns and create tables in your menus. In the past (before patch 2.5) creating columns was only possible by workarounds using special library scripts. With patch 2.5 this is not the case anylonger, because any menu command which expects a text, will also work with an array of the following structure:
`ARRAY('PosCol1', 'text1', 'PosCol2', 'text2', ... , 'PosColn', 'textn')`

'PosCol1' stands for 'position of column one' and has to be an integer value. 1 will position your column at the left side, -1 at the right side. For the two different menu types there are of course two different maximum positions. For the 'normal' menu 346 is the maximum value, for the 'info' menu it is 858.
'text1' stands for the text which will displayed at previoulsy defined position and **has** to be a string, otherwise it won't be displayed.

Actually this is all there is to say about columns. They are quite easy to use and will enhance your menus, if you need them.

3.2 Value selections

Using the command add value selection to menu: <Var/Array$_1$>, text=<Var/String$_1$>, value array=<Var/Array$_2$>, default=<Var/Number>, return id=<Var/String$_2$> it is possible to add value selections to your own menus.
<Var/Array$_1$> is of course the menu array, you want to add the selection to, <Var/String$_1$> the text, which will appear in front of the selection. <Var/Array$_2$> is the array, holding the values you want to browse through.
With <Var/Number> you can define which of the array's values should be displayed by default when opening the menu, with <Var/Number> being the index of this value.
Finally, with <Var/String$_2$> you can set a return id, which will later help us identifying the value selection.

3.2.1 Interpreting the return of a menu containing value selections

As soon as menus contain value selections the interpretation of the returns gets a bit more complicated. In the following paragraph I'll always refer to the following code for variable names:
```
$ret = open custom menu:  title = null description = null option array = $menu
```

When hitting {ESC} $ret will hold the integer value -1. This is true for any kind of menu, regardless of the used menu items. Now for the more elaborate situations:

Say we created an exit-button with the returnvalue 'exit', which is preceded by two value selections with the return ids 'selection1' and 'selection2'.
If you now select the exit-button, $ret will not simply hold 'exit', but the following array:
ARRAY('exit', ARRAY(2), ARRAY(2))
First of all, we still see 'exit' as the first entry of the $ret–array, so the first element of the return array will always hold the selected menu item's returnvalue.
The following elements in the $ret–array are arrays of the follwoing structure:
ARRAY('return id', 'index')
So in our case the second element of the $ret–array will be ARRAY('selection1', 'index'), with 'index' being the index of the selected value in the value array. But you still don't know which value was selected, because you only know its index, so in a last step, you'll normally have to get the element with this index out of the value array.

The third element of our $ret–array will be the corresponding array for the second value selection. As you see, the value selections will be listed in the $ret–array according to their position in the menu.

Working with value selections may seem complicated at first, but sooner or later you'll get used to it and your menus will benefit from their use.

3.3 Dynamic menus

Long time I didn't even know this was possible, but by accident I found the way to create menus, which can actually display data dynamically.
For the beginner it might seem complicated, but it is not. The basic idea is that you directly alter the menu array. To do so, you'll have to save the menu items in a global variable, with one script altering the values and another one opening the menu. The following example should clarify how things work. It uses two scripts, which will together make a dynamic menu displaying the playerships rotation values. The first one is the script gathering the data, the second one displays them.

```
001 $text = array alloc: size=6
002 $text[0] = 1
003 $text[2] = 100
004 $text[4] = 200
005 $item = create new array, arguments=$text, [TRUE], null, null, null
006 set global variable: name='log.rot' value=$item
007
008 while [TRUE]
009   $alpha = [PLAYERSHIP]->get rot alpha
010   $beta = [PLAYERSHIP]->get rot beta
011   $gamma = [PLAYERSHIP]->get rot gamma
012
013   $alpha = convert number $alpha to string
014   $beta = convert number $beta to string
015   $gamma = convert number $gamma to string
016
017   $item[0][1] = $alpha
018   $item[0][3] = $beta
019   $item[0][5] = $gamma
020   = wait 500 ms
021 end
022 return null
```

I'll explain the most important lines:

001–004 Setting up the text for the menu item; three columns will be positioned at 1, 100 and 200

005 Creating the menu item directly without using the menu commands, since there is yet no menu we could add the item to

006 Saving the menu item in a global variable called `'log.rot'`

008 Beginning of the infite loop, which will log the playerships rotation values

009–011 Gathering rotation values

013–015 Converting the rotation values to strings, because columns can **only** display strings

017–019 Altering the menu item array; since arrays are just pointers, this directly alters the global variable, too

020 Actually this wait is too short, as dynamic menus will only update every second

021 End of the inifte loop

Now that we have set up the script, which alters the menu item, let's have a look at the script, which opens the menu.

```
001 $heading = array alloc: size=6
002 $heading[0] = 1
003 $heading[1] = 'Alpha'
004 $heading[2] = 100
005 $heading[3] = 'Beta'
006 $heading[4] = 200
007 $heading[5] = 'Gamma'
008 $item = get global variable: name='log.rot'
009
010 $menu = create custom menu array: heading=$heading
011 append $item to array $menu
012 $ret = open custom menu: title='Rotation' description=null option array=$menu
013 return null
```

I guess by now you already know how it's working, but here's the explanation:

001–007 Setting up the menu heading

008 Loading global variable, which holds the menu item

010 Creating menu, directly appending heading

011 Appending the menu item to the menu array

012 Opening the menu

And that's it. To run the dynamic menu, you'll first have to run the first script and then the second one. Of course there are ways to do this more elegantly, but for demonstration purposes this should be enough.

4 Known Bugs

The introduction of the new menus commands in patch 2.5 also introduced several bugs, concerning the menus.

4.1 Double headings

When the first heading of your menu contains column information, the title of the menu will appear as an additional heading above the one you set yourself. Using the following code, you can reproduce that bug yourself.

```
001 $menu = create custom menu array
002 $text = create new array, arguments=1, 'Test1', -1, 'Test2', null
003 add heading to menu $menu: heading=$text
004 add custom menu item to array $menu: text='Text' return value=[TRUE]
005 $ret = open custom menu: title='MyTitle' description=null option array=$menu
```

Solution: Simply add a menu heading 'null' in front of the first heading. So the above script would look like this:

```
001 $menu = create custom menu array
002 add heading to menu $menu: heading=null
003 $text = create new array, arguments=1, 'Test1', -1, 'Test2', null
004 add heading to menu $menu: heading=$text
005 add custom menu item to array $menu: text='Text' return value=[TRUE]
006 $ret = open custom menu: title='MyTitle' description=null option array=$menu
```

Adding custom info lines to your menu will also lead to 'double headings'. However, I didn't research the exact circumstances, so you'll have to check yourself.

4.2 Moving columns

This is a very strange bug, which will appear when you have at least two different column groups in your menus. Just imagine you want to display one set of data with the column values 1 and 100 and another set with the values 20 and 70. When the menu entries of column group two (20, 70) are only visible by scrolling down the menu, the columns will not be positioned correctly. In my experience they will widen, e.g. the positions will change from (20, 70) to (10, 80). This can be very annoying, when your columns are positioned closely to each other.
Solution: Unforunately I didn't find any solution until now. If one of you does, please report.

5 Resumé

I hope you liked my introduction to the menus of X3: Terran Conflict. For any questions, ideas or mistakes feel free to contact me in the official EGOSOFT Forum or, even better, post in the thread for this tutorial. With new knowledge gathered go ahead and create beautiful menus!

Greetings,
ScRaT