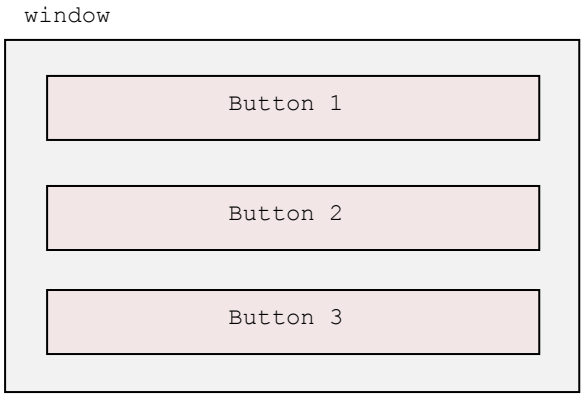
BUTTON MENU
DIALOG TUTORIAL

1. Overview

This tutorial will show you how to build a simple menu for the **Operation Flashpoint** and **Arma: Cold War Assault** in a way that will make it easily customizable. I assume you already know the basics of dialogs. If not then see Vektorboson's Dialog Tutorial.

2. Plan

Here's a representation of what I'm going to create:



It's a background with a few buttons placed inside.

3. Preparation

I'm going to need two classes from the *resource.cpp* file – one that defines a background and the other one – a button. By inheriting from them I'll be able to quickly create controls that I want.

```
class RscBackground
    {
    access=3;
    type=0;
    idc=-1;
    style=80;
    x=0.15;
    y=0.15;
    w=0.7;
    h=0.7;
    text="";
    colorBackground[]={1,1,1,1};
    colorText[]={0,0,0,0};
    font="tahomaB24";
    sizeEx=0;
    };

class RscButton
    {
    access=3;
    type=1;
    style=2;
    w=0.16;
    h=0.06;
    colorText[]={0.08,0.08,0.12,1};
    font="tahomaB24";
    sizeEx=0.02;
    default=0;
    soundPush[]={"ui\ui_ok",0.2,1};
    soundClick[]={"",0.2,1};
    soundEscape[]={"ui\ui_cc",0.2,1};
    };
```

Next I add an empty dialog class:

```
class Button_Menu {
    idd          = 0
    movingEnable = 0
    controls[]   = {};
}
```

To display it in the game I write in the *init.sqs*:

```
~0.01
createDialog "Button_Menu"
```

I have all the necessities now. It’s time to start coding what I’ve actually planned.

4. Adding Controls

This is the background element:

```
class Window : RscBackground {
    x = 0.1
    y = 0.1
    w = 0.3
    h = 0.3
}
```

The first button:

```
class Button1 : RscButton {
    idc  = 1
    x    = 0.13
    y    = 0.13
    w    = 0.24
    h    = 0.06
    text = "Button 1";
}
```

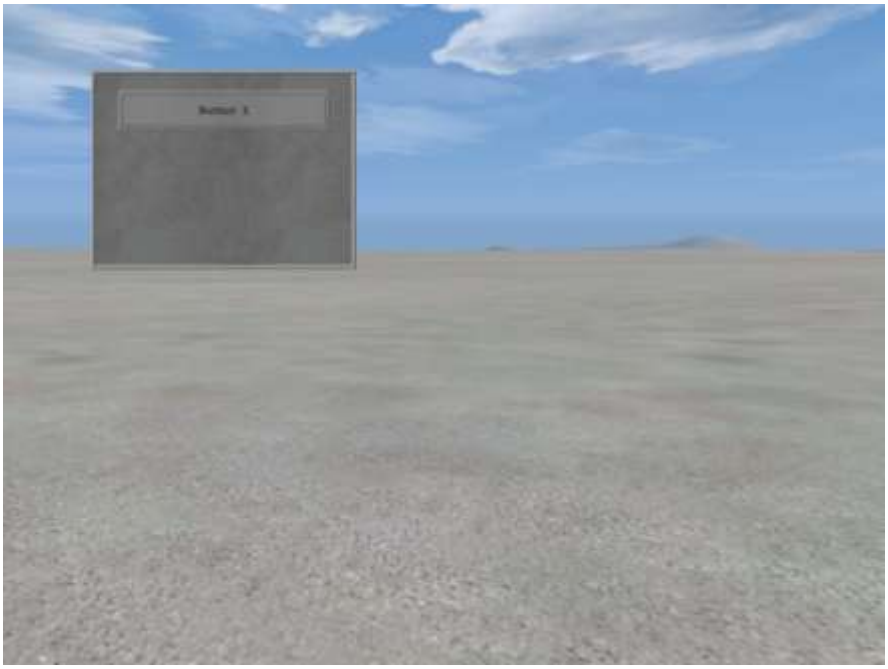
I'm placing buttons inside a window so X, Y need to be larger. I've decided that space between the window edge and a button will be `0.03` so `0.1 + 0.03 = 0.13`

Width of the button should be smaller than the window width. To keep spacing symmetrical I'm subtracting two spaces from the window size so `0.3 - 0.03*2 = 0.24`

Don't forget to add these classes to the `controls[]` array:

```
controls[] = {Window, Button1};
```

Preview:



Let’s keep going and add more buttons:

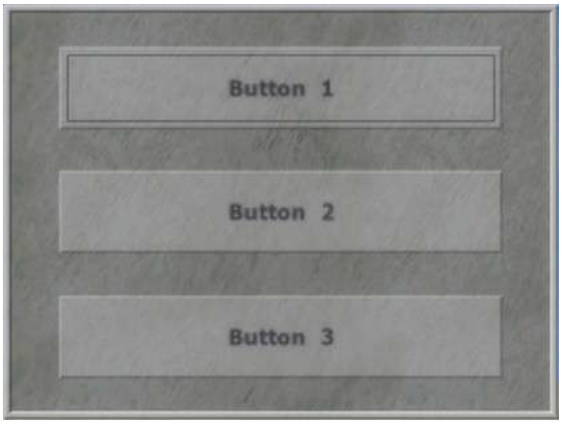
```
class Button2 : Button1 {
    idc  = 2
    y    = 0.22
    text = "Button 2";
}

class Button3 : Button1 {
    idc  = 3
    y    = 0.31
    text = "Button 3";
}
```

X coordinate, width and height do not change so I inherit them from the first button.

To find the Y coordinate for the next button I take the previous value (`0.13`), add button height to it (`+0.06`) and a space (`+0.03`). So `0.13 + 0.09 = 0.22` for the second button and `0.22 + 0.09 = 0.31` for the third button.

Here’s full code:

```
class Button_Menu {
      idd          = 0
      movingEnable = 0
      controls[]   = {Window, Button1, Button2, Button3};

      class Window : RscBackground {
        x = 0.1
        y = 0.1
        w = 0.3
        h = 0.3
      }

      class Button1 : RscButton {
            idc  = 1
            x    = 0.13
            y    = 0.13
            w    = 0.24
            h    = 0.06
            text = "Button 1"
      }

      class Button2 : Button1 {
            idc  = 2
            y    = 0.22
            text = "Button 2";
      }

      class Button3 : Button1 {
            idc  = 3
            y    = 0.31
            text = "Button 3";
      }
}
```



That was easy, right? Except if I want to make changes to the position or size I'll have to recalculate all the values.

5. Formulas

Game can actually compute the numbers for me if I write formulas in the class properties.

```
class Button1
    x = 0.1 + 0.03
    y = 0.1 + 0.03
    w = 0.3 - 0.03*2

class Button2
    y = 0.1 + 0.03*2 + 0.06

class Button3
    y = 0.1 + 0.03*3 + 0.06*2
```

To find position for the next button I'm summing up:

- window position `(0.1)`
- number of spaces multiplied by amount of buttons `(0.03 * 3)`
- total size of the previous buttons `(0.06 * 2)`

For consistency I'll write formulas in their full form:

```
class Button1
    y = 0.1 + 0.03*1 + 0.06*0

class Button2
    y = 0.1 + 0.03*2 + 0.06*1

class Button3
    y = 0.1 + 0.03*3 + 0.06*2
```

Notice how the multlipliers increase.

That's better but I still have to modify values in multiple places. To deal with that problem I'm going to use preprocessor macros.

6. Constant Values

If you're not familiar with the preprocessor then read this document before moving on.

I'm going to assign number to a word and then replace all occurences of that number with the word. Let's start with window position. 0.1 is going to be substituted with `WINDOW_X` and `WINDOW_Y` and 0.3 with `WINDOW_W` and `WINDOW_H`.

```
#define WINDOW_X  0.1
#define WINDOW_Y  0.1
#define WINDOW_W  0.3
#define WINDOW_H  0.3

class Window
    x = WINDOW_X
    y = WINDOW_Y
    w = WINDOW_W
    h = WINDOW_H

class Button1
    x = WINDOW_X + 0.03
    y = WINDOW_Y + 0.03*1 + 0.06*0
    w = WINDOW_W - 0.03*2

class Button2
    y = WINDOW_Y + 0.03*2 + 0.06*1

class Button3
    y = WINDOW_Y + 0.03*3 + 0.06*2
```

Next is space size. I'm replacing 0.03 with `SPACE_W` and `SPACE_H`.

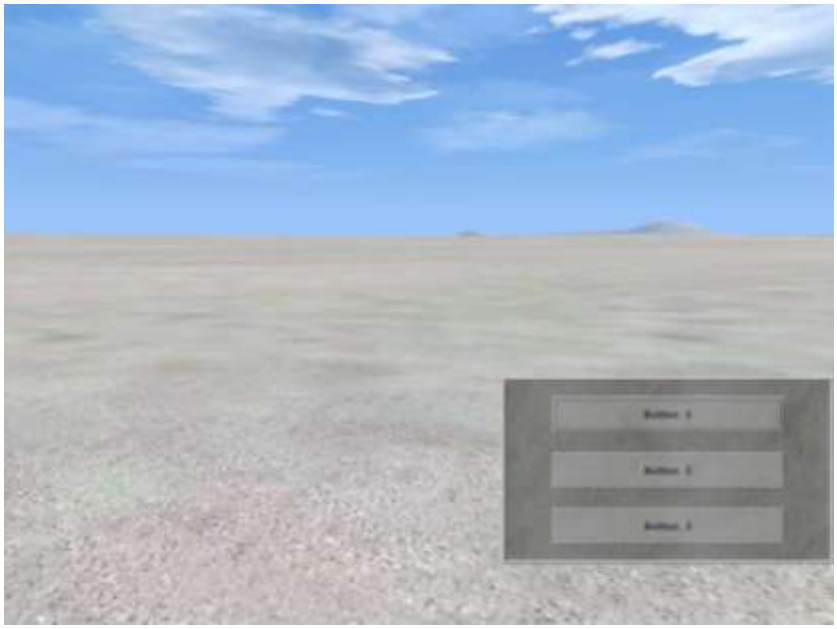
```
#define SPACE_W  0.03
#define SPACE_H  0.03

class Button1
    x = WINDOW_X + SPACE_W
    y = WINDOW_Y + SPACE_H*1 + 0.06*0
    w = WINDOW_W - SPACE_W*2

class Button2
    y = WINDOW_Y + SPACE_H*2 + 0.06*1

class Button3
    y = WINDOW_Y + SPACE_H*3 + 0.06*2
```

Lastly, 0.06 will be `BUTTON_H` and formulas for calculating X coordinate and width will be written as `BUTTON_X` and `BUTTON_W` respectively.

```
#define BUTTON_X  WINDOW_X + SPACE_W
#define BUTTON_W  WINDOW_W - SPACE_W*2
#define BUTTON_H  0.06

class Button1
    x = BUTTON_X
    y = WINDOW_Y + SPACE_H*1 + BUTTON_H*0
    w = BUTTON_W
    w = BUTTON_H

class Button2
    y = WINDOW_Y + SPACE_H*2 + BUTTON_H*1

class Button3
    y = WINDOW_Y + SPACE_H*3 + BUTTON_H*2
```

This is more convenient. Now I can quickly move menu to the right bottom corner and change its width.

```
class Button_Menu {
      idd          = 0
      movingEnable = 0
      controls[]   = {Window, Button1, Button2, Button3};

      #define WINDOW_X  0.6
      #define WINDOW_Y  0.6
      #define WINDOW_W  0.4
      #define WINDOW_H  0.3

      #define SPACE_W   0.06
      #define SPACE_H   0.03

      #define BUTTON_X  WINDOW_X + SPACE_W
      #define BUTTON_W  WINDOW_W - SPACE_W*2
      #define BUTTON_H  0.06

      class Window : RscBackground {
        x = WINDOW_X
        y = WINDOW_Y
        w = WINDOW_W
        h = WINDOW_H
      }

      class Button1 : RscButton {
            idc  = 1
            x    = BUTTON_X
            y    = WINDOW_Y + SPACE_H*1 + BUTTON_H*0
            w    = BUTTON_W
            h    = BUTTON_H
            text = "Button 1"
      }

      class Button2 : Button1 {
            idc  = 2
            y    = WINDOW_Y + SPACE_H*2 + BUTTON_H*1
            text = "Button 2";
      }

      class Button3 : Button1 {
            idc  = 3
            y    = WINDOW_Y + SPACE_H*3 + BUTTON_H*2
            text = "Button 3";
      }
}
```



However, I still can't change height without recalculating and adding a new button is still a hassle.

7. Flexible Macros

I'm going to replace formula for calculating button Y coordinate with a function-like macro.

```
#define BUTTON_Y(AMOUNT)  WINDOW_Y + SPACE_H*AMOUNT + BUTTON_H*(AMOUNT-1)

class Button1
    y = BUTTON_Y(1)

class Button2
    y = BUTTON_Y(2)

class Button3
    y = BUTTON_Y(3)
```

This way with every new button I only have to increment one number instead of two.

Height of the window is equal to the height of all buttons and spaces. I move `WINDOW_H` macro to the end and define it as:

```
#define WINDOW_H(AMOUNT)  SPACE_H*(AMOUNT+1) + BUTTON_H*AMOUNT

class Window
    h = WINDOW_H(3)
```

Currently I have three buttons so I pass number 3 to the `WINDOW_H` macro. Now if I change button height or vertical space then the window size will be adjusted automatically.

Lastly we can shorten button class. Another function-like macro `MAKE_BUTTON` is going to represent an entire class:

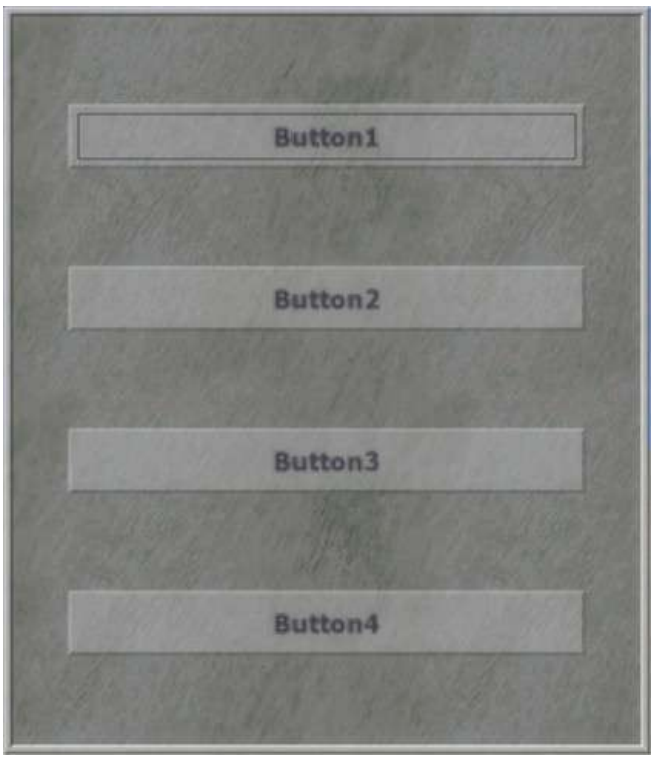
```
#define MAKE_BUTTON(NUM) \
    class Button##NUM : RscButton { \
    idc  = NUM; \
    x    = BUTTON_X; \
    y    = BUTTON_Y(NUM); \
    w    = BUTTON_W; \
    h    = BUTTON_H; \
    text = Button##NUM; \
    };

MAKE_BUTTON(1)
MAKE_BUTTON(2)
MAKE_BUTTON(3)
```

Now to add a new button I only need to:

1. Copy `MAKE_BUTTON` line
2. Add class name to the list of controls
3. Increase number passed to the `WINDOW_H` macro

Full code below:

```
class Button_Menu {
    idd          = 0
    movingEnable = 0
    controls[]   = {Window, Button1, Button2, Button3, Button4};

       #define WINDOW_X          0.1
       #define WINDOW_Y          0.1
       #define WINDOW_W          0.4

       #define SPACE_W           0.03
       #define SPACE_H           0.06

       #define BUTTON_X          WINDOW_X + SPACE_W
       #define BUTTON_Y(AMOUNT)  WINDOW_Y + SPACE_H*AMOUNT + BUTTON_H*(AMOUNT-1)
       #define BUTTON_W          WINDOW_W - SPACE_W*2
       #define BUTTON_H          0.04

       #define WINDOW_H(AMOUNT)  SPACE_H*(AMOUNT+1) + BUTTON_H*AMOUNT

       #define MAKE_BUTTON(NUM) \
              class Button##NUM : RscButton { \
              idc  = NUM; \
              x    = BUTTON_X; \
              y    = BUTTON_Y(NUM); \
              w    = BUTTON_W; \
              h    = BUTTON_H; \
              text = Button##NUM; \
              };

       class Window : RscBackground {
         x = WINDOW_X
         y = WINDOW_Y
         w = WINDOW_W
         h = WINDOW_H(4)
    }

       MAKE_BUTTON(1)
       MAKE_BUTTON(2)
       MAKE_BUTTON(3)
       MAKE_BUTTON(4)
}
```



8. Variables

Automation could go even further. I'd like the background size to be set dynamically instead of manually. To achieve that I'm going to create a variable indicating amount of buttons and increment it with each added button.

OFP configuration files feature two commands:

- **__EXEC** – for executing statements (code that does something and doesn't return a value)
- **__EVAL** – for executing expressions (code that returns a value)

Code passed to these commands is a normal OFP scripting language.

To create a new variable I write on the beginning:

```
__EXEC(NUMBER_OF_BUTTONS = 0)
```

Then I modify `MAKE_BUTTON` macro to contain this line:

```
#define MAKE_BUTTON(NUM) \
    __EXEC(NUMBER_OF_BUTTONS = NUMBER_OF_BUTTONS + 1); \
```

Afterwards I move `class Window` so that it's defined after all the buttons. I update height property so that it will use this variable to calculate window height.

```
class Window
    h = __EVAL(SPACE_H * (NUMBER_OF_BUTTONS+1) + BUTTON_H * NUMBER_OF_BUTTONS)
```

Now adding a new button requires only two steps (creating class and adding it to the list) instead of three (I don't have to resize the window anymore).

Similarly, I'm going to turn button Y coordinate into a variable that changes with each new button. This is the initial position:

```
__EXEC(BUTTON_Y = WINDOW_Y + SPACE_H)
```

And that's how it changes when a new button class is created:

```
#define MAKE_BUTTON(NUM) \
    y = __EVAL(BUTTON_Y); \
    __EXEC(BUTTON_Y = BUTTON_Y + BUTTON_H + SPACE_H); \
```

I'm going to do the same for the `idc` property because I want id numbers to start from 10.

```
__EXEC(BUTTON_IDC = 10)

#define MAKE_BUTTON(NUM) \
    idc = __EVAL(BUTTON_IDC); \
    __EXEC(BUTTON_IDC = BUTTON_IDC + 1); \
```

For consistency I'm going to turn remaining macros into variables. Here's full code:

```
class Button_Menu {
    idd          = 0
    movingEnable = 0
    controls[]   = {Window, Button1, Button2, Button3, Button4};

      // Customizable values
      __EXEC(WINDOW_X   = 0.1)
      __EXEC(WINDOW_Y   = 0.1)

      __EXEC(SPACE_W    = 0.03)
      __EXEC(SPACE_H    = 0.03)

      __EXEC(BUTTON_W   = 0.24)
      __EXEC(BUTTON_H   = 0.06)
      __EXEC(BUTTON_IDC = 10)


      // Calculations
      __EXEC(WINDOW_W          = BUTTON_W + SPACE_W*2)
      __EXEC(BUTTON_X          = WINDOW_X + SPACE_W)
      __EXEC(BUTTON_Y          = WINDOW_Y + SPACE_H)
      __EXEC(NUMBER_OF_BUTTONS = 0)


      // Macro for inserting button class
      #define MAKE_BUTTON(NUM) \
            class Button##NUM : RscButton { \
            idc  = __EVAL(BUTTON_IDC); \
            x    = __EVAL(BUTTON_X); \
            y    = __EVAL(BUTTON_Y); \
            w    = __EVAL(BUTTON_W); \
            h    = __EVAL(BUTTON_H); \
            text = Button##NUM; \
            }; \
            __EXEC(NUMBER_OF_BUTTONS = NUMBER_OF_BUTTONS + 1); \
            __EXEC(BUTTON_Y          = BUTTON_Y + BUTTON_H + SPACE_H); \
            __EXEC(BUTTON_IDC        = BUTTON_IDC + 1);


      // Classes
      MAKE_BUTTON(1)
      MAKE_BUTTON(2)
      MAKE_BUTTON(3)
      MAKE_BUTTON(4)

      class Window : RscBackground {
        x = __EVAL(WINDOW_X)
        y = __EVAL(WINDOW_Y)
        w = __EVAL(WINDOW_W)
        h = __EVAL(SPACE_H*(NUMBER_OF_BUTTONS+1) + BUTTON_H*NUMBER_OF_BUTTONS)
    }
}
```

Also window width is now calculated out of the button width (previously it was the opposite).

9. Dynamic Position

I'd like to have my menu always in the center of the screen. In that case the window coordinates will be calculated based on the number of buttons and their size.

Unfortunately it's not possible to create buttons and window and then update them to be in the middle. I must calculate center position initially and that requires knowing the number of buttons in advance. Downside is that adding a new button will again require 3 steps.

`NUMBER_OF_BUTTONS` variable will now indicate how many buttons I've planned to define.

```
__EXEC(NUMBER_OF_BUTTONS = 4)
```

Also I remove line that changes this variable from the `MAKE_BUTTON` macro.

With the amount of buttons and their size I can now determine the window size:

```
__EXEC(NUMBER_OF_SPACES = NUMBER_OF_BUTTONS + 1)
__EXEC(WINDOW_H         = BUTTON_H*NUMBER_OF_BUTTONS + SPACE_H*NUMBER_OF_SPACES)
```

Because I can't use parenthesis in `__EXEC` I created a new variable called `NUMBER_OF_SPACES` which is just a number of buttons increased by one.

Knowing window width and height I can now find where it should be placed. Formula to get center is: `(1 - size) / 2`

To work around the lack of parentheses I update the variable twice:

```
__EXEC(WINDOW_X = 1 - WINDOW_W)
__EXEC(WINDOW_X = WINDOW_X / 2)

__EXEC(WINDOW_Y = 1 - WINDOW_H)
__EXEC(WINDOW_Y = WINDOW_Y / 2)
```

And that's it. Final code below:

```
class Button_Menu {
    idd          = 0
    movingEnable = 0
    controls[]   = {Window, Button1, Button2, Button3, Button4};

      // Customizable values
      __EXEC(SPACE_W           = 0.03)
      __EXEC(SPACE_H           = 0.03)
      __EXEC(BUTTON_W          = 0.24)
      __EXEC(BUTTON_H          = 0.06)
      __EXEC(BUTTON_IDC        = 10)
      __EXEC(NUMBER_OF_BUTTONS = 4)

      // Calculations
      __EXEC(NUMBER_OF_SPACES  = NUMBER_OF_BUTTONS + 1)
      __EXEC(WINDOW_W          = BUTTON_W + SPACE_W*2)
      __EXEC(WINDOW_H          = BUTTON_H*NUMBER_OF_BUTTONS + SPACE_H*NUMBER_OF_SPACES)
      __EXEC(WINDOW_X          = 1 - WINDOW_W)
      __EXEC(WINDOW_X          = WINDOW_X / 2)
      __EXEC(WINDOW_Y          = 1 - WINDOW_H)
      __EXEC(WINDOW_Y          = WINDOW_Y / 2)
      __EXEC(BUTTON_X          = WINDOW_X + SPACE_W)
      __EXEC(BUTTON_Y          = WINDOW_Y + SPACE_H)

      // Macro for inserting button class
      #define MAKE_BUTTON(NUM) \
             class Button##NUM : RscButton { \
             idc  = __EVAL(BUTTON_IDC); \
             x    = __EVAL(BUTTON_X); \
             y    = __EVAL(BUTTON_Y); \
             w    = __EVAL(BUTTON_W); \
             h    = __EVAL(BUTTON_H); \
             text = Button##NUM; \
             }; \
             __EXEC(BUTTON_Y   = BUTTON_Y + BUTTON_H + SPACE_H); \
             __EXEC(BUTTON_IDC = BUTTON_IDC + 1);

      // Classes
      class Window : RscBackground {
        x = __EVAL(WINDOW_X)
        y = __EVAL(WINDOW_Y)
        w = __EVAL(WINDOW_W)
        h = __EVAL(WINDOW_H)
      }

      MAKE_BUTTON(1)
      MAKE_BUTTON(2)
      MAKE_BUTTON(3)
      MAKE_BUTTON(4)
}
```

10. Scripting

I'll briefly show you how make this menu functional. During the mission I'm going to change button text and action. I store this data in a two-dimensional array:

```
_button_properties = [["Hint", "hint ""test"""], ["Weather", "0
setOvercast 1"], ["Heal", "player setDammage 0"], ["Weapon","player
addWeapon ""M16"""]]
```

Variables created in the *description.ext* are not transfered to other scripts so again I need to declare how many buttons there are and what's their identification number:

```
_button_idc         = 10
_number_of_buttons = 4
```

Finally I write a loop that will take data from the array and set new labels and actions for the buttons starting from the initial id number.

```
_i = 0

#Loop
_idc  = _button_idc + _i
_text = (_button_properties select _i) select 0
_code = (_button_properties select _i) select 1

ctrlSetText     [_idc, _text]
buttonSetAction [_idc, _code]

? _i < _number_of_buttons : _i=_i+1; goto "Loop"
```

11. Conclusion

That's the end of this tutorial. To illustrate how macros and variables become useful I started with the simplest example but I don't recommend for you to do the same. Instead use variables (page 11) from the very beginning.

Have fun!

Faguss (ofp-faguss.com)