Pimp my Project

2D Tutorial

for beginners

by Tobias "LordRathan" Mogdans

Version: 1.0

Index

Preamble

In this tutorial I would like to give you an understanding of the subject of game menus. Beautiful and clear menus are very important for successful games and they are a pleasure for the users eyes.

This tutorial does not raise a requirement on completeness or whereupon that it is the best or only way in order to carry out 2D-Work. It is the way, which I gained and which works for me well.

A reference still at the beginning: There is the panel editor plugin for SED. We forget it again, because we want know what we script. There is no reason to use this editor. Panels are extremely simple to create.

We will also regard now little theory with a few examples. The second part treats some codesnippets as suggestions and ideas.

Much fun thereby.

PS: Please forgive my written english word. It is easier for me to translate an english text to german. :)

Theory of 2D-programming with c-script

The most important part of panel programming is the definition of a panel. Let's have a look at the code:

```
PANEL newpanel
{
    ...
}
```

We told A6 with these lines that there is a new panel with the name "newpanel". It looks a little like a function, just without brackets after the name.

Panels are elementary components of a 2D game and of important importance for the user surface of 3D games. We can realize inventories, buttons, life announcements, compasses, crosshairs, main menues, missionsbriefings, tooltips and much, much more with panels.

To build such things we need however more than only an empty panel definition like above. Panels know background pictures, clickable buttons, sliding controls, numbers and text.

Naturally we want to determine in which place on the screen our panel appears:

```
PANEL newpanel
{
    pos_x = 100;
    pos_y = 150;
}
```

Our panel holds a distance to the left edge of the screen of 100 pixels and a distance to the top margin of 150 pixels. In functions we can access and change the position of the panel. Like that simple animations are possible, in addition, for more complex animations we must change the position.

So far our panel is still empty and invisible. A6 knows that there is also a panel named "newpanel" and it is to appear at position 100, 150. We only havn't defined WHAT to appear.

```
BMAP backpanel_tga = „green.tga“;

PANEL newpanel
{
    bmap = backpanel_tga;
    pos_x = 100;
    pos_y = 150;
    flags = visible;
}
```

Now we see the first time something at the screen. What did we do?
First we defined a graphic outside of the panel definition. BMAP is the type for graphics, **backpanel_tga** is the name, with which we can access this graphic, and **green.tga** is the name of the graphic, which was provided before with a drawing program.
Within the panel definition we assigned with **bmap = backpanel_tga;** the defined graphic to our panel as background picture.
The flag **visible** in the last line determines that the panel is visible immediately.

Expert tip!

As graphic format for panels can be used Windows bitmap (bmp), ZSoft PCX (pcx) and TarGA (tga).

Bitmap I advise, since the same quality can be achieved also with PCX, the graphic uses less video memory. The color black (0, 0, 0) defines tranzparency with both formats.

TarGA is a useful format, since it can contain an alpha channel. For more complex graphics the more is worthwhile itself at used up video memory in any case.

That is everything we need to show splashscreens or load pictures. Now we want to be able to interact with our panel. Very frequently we need buttons, in order to confirm a message or to choose a selection.

Buttons

```
BMAP backpanel_tga = „green.tga“;
BMAP button_on_tga = „on.tga“;
BMAP button_off_tga = „off.tga“;
BMAP button_over_tga = „over.tga“;

PANEL newpanel
{
    bmap = backpanel_tga;
    pos_x = 100;
    pos_y = 150;
    button = 40, 10, button_on_tga, button_off_tga, button_over_tga, NULL, NULL, NULL;
    flags = visible;
}
```

A very extensive element. What have we done here?
First we defined three new graphics again outside of the panel definition. We take these graphics for the three conditions, which the button can accept:

1. The button is clicked
2. The button isn't clicked and the mousecursor is anywhere else
3. The mousecursor floats over the button, but it isn't clicked

In the panel definition we defined a button (**button**). The two numbers indicate, in which place the button is to be. Note here that the coordinates refer to the position of the panel, i.e. the button is 40 pixels of the left panel edge and 10 pixels down from the upper panel edge. That is very useful, because if we want to move the panel we havn't be worry about the fact that the button is also along-moved. That happens automatically, because the button is always on the same position from the left upper panel edge.

The next three parameters indicate, which graphics for the three conditions are to be used. The first parameter for "clicked" must be indicated. The size of this graphic determines the size of the button. The other two parameters can be indicated also as NULL, however for this condition also no own graphic is then shown.

The remaining three parameters are for function calls. If we press the button something should happen.

The first parameter calls a function, if the button is clicked. This parameter is probably most frequently used. The second parameter calls its function, if the mouse button is released or if the mousecursor is removed from the button. The third parameter calls its function, if the button is affected. If e.g. a noise is to sound as soon as the button is affected, the last parameter is our friend.

Additionally still the number of the button can be handed over to the implementing function. That can be very useful, if several buttons use the same function. The first button is thereby automatically button 1, the second button 2 etc...

Digits

Panels can be naturally well used to present the graphical user interface. In many games the life energy of the player wants to be represented. We could do that with simple numbers.

```
PANEL newpanel
{
    pos_x = 20;
    pos_y = 10;
    digits 0, 0, 3, *, 1, player.health;
    flags = visible;
}
```

The first two parameters shows - as also with button - the position of the element within the panel.

The third parameter indicates the format. For numbers it is still very simple. The integral portion intends the entire places for the indicated number, which indicates decimal place, how many placing after the comma. A place is reserved with comma numbers for the decimal point.
6.2 would cause that we receive alltogether 6 places, whereby 2 places stand for after the comma and a place for the decimal point. Thus we would have still 3 places before the comma, e.g. 103.76. If the format is indicated negatively, thus -6,2, then zeros were placed in front with numbers under 100, e.g. 004.86

The fourth parameter determines the character set. „*" is used here for the engine-own characterset. Naturally you also can insert a characterset you defined before.

The fifth parameter stands for the multiplicator. The indicated result is multiplied by this number. For our life announcement it dosn't matter, therefore we multiply with 1.

And the last parameter shows what is to be indicated at all. That can be a variable or also a Skill of an entity.

Expert tip!

Since the version 6.4 the third parameter can be also used for c-formatstrings. In addition we write the format string in quotation marks.

"[text1]%[flags][width][.prec]f[text2]"

[text1] = An optional placed before text
[flags] = optional one ore more signs
[width] = minimum number of places
[.prec] = maximum number of places after comma
[text2] = An optionally placed behind text

Instead of these whole parameters also a maximally 100 indications long text can be used. This text may not consist however of only one number.

Window

Window indicates a cutout from a graphic. With window we can animate panels, realize health bars displays or represent a compass.

```
PANEL newpanel
{
	pos_x = 36;
	pos_y = 23;
	window = 0, 0, 124, 17, tut4_bar_tga, health_status, 1;
	layer = 2;
	flags = visible;
}
```

The first two parameters indicate, in which place of the panel the window is to appear. The two next parameters specify, how large the cutout should be. The fifth parameter specifies, which grafic is used, and the last both parameters determine, where on the graphic it begins to cut out.

Slider

Slider comes in two different versions, one for horizontal and one for vertical slider. They are suitable well for functions, in which the user can change the value of a variable between two specified values at will.

```
PANEL newpanel
{
    pos_x = 20;
    pos_y = 10;
    hslider = 0, 0, 200, tut5_slider_tga, 0, 100, test_pan_alpha;
    flags = visible;
}
```

The first two parameters indicate again the position within the panel. The third parameter describes, how many pixels the total travel of the slider amounts to. In the example I can move the slider 200 pixels to the right far from left. In the fourth parameter the grahic of the slider is indicated.
The two next parameters indicate the minimum and the maximum value, which the variable in the last parameter can accept. In my example the variable "test_pan_alpha" can be between 0 and 100.

Flags

The flags of the panels give now still a few characteristics.

- visible
- overlay
- translucent
- filter
- light

Visible participates, as responsible with all objects in C-Script, for it whether the panel is indicated or is invisible.
If the **overlay** flag is set, all black ranges (0, 0, 0) of the panel graphic is complete transparency. For TGA graphics with alpha channel we do not need this flag.
Translucent certainly influence since version 6.4 whether the alpha value of a panel possesses, or not. If you provide fading with panels, make sure that this flag is set.
Filter can lead with certain effects to an improvement of the representation, above all, if you work with scaling effects. With very small fonts however this flag is counter productive, because it smears the pixels.
Light finally permits still beautiful effects, because it modulates the elements on the panel by the Red, Blue and Green values of the panel.

If more than one flag is used, for the separation since 6.4 the or-symbol is used "|". But no fear, a normal "+" still works.

```
Flags = visible | translucent | filter;     //New code, Version 6.4 and above

or

Flags = visible + translucent + filter;  //Old code, version 6.34.1 and older
```

Red, Green, Blue and Alpha

Worth a word are still the instructions Red, Green and Blue. With them the color value of the panel can be determined, similarly as with the light. Already think of the idea to let the user adjust the color of its HUDs? With a few sliders and these three values it might be no problem.

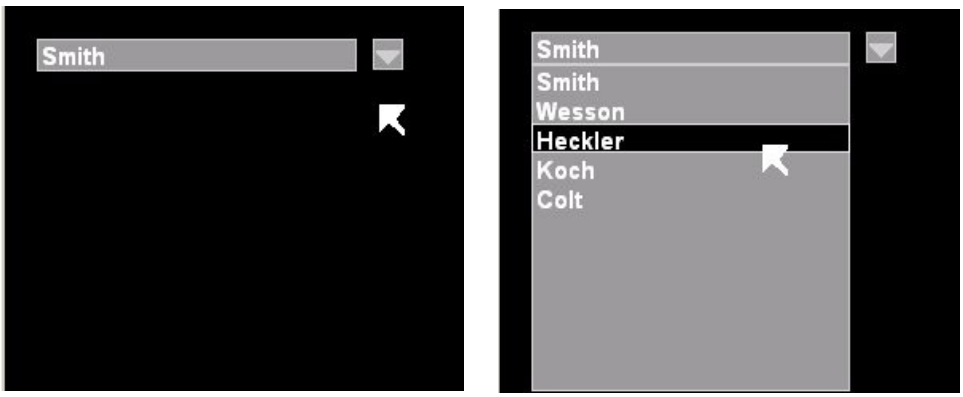
The alpha value shows, similarly as with models, the transparency of the panel. The value is ranged thereby from 0 (completely transparency) to 100 (not transparency). To consider here is however that the flag "translucent" must be set.

Suggestions, ideas & codesnippets

Now some ideas and examples follow, which can be used for animations in menus.

Drop-Down

One of the simplest menu functions is probably drop down a selection field.



We need for it three important graphic elements: the line, in which the selection is located, the selection list and optionally a button, in order to show or hide the selection list. Naturally you could integrate the Button into the selection line, but we light up here the optional variant. Furthermore we use still some dummy graphics and adornments. For this example I used the "tut1_" graphics. You find them in the folder "gfx_2d". The finished code is in the file "tut_1.wdl".

```
path "gfx_2d";                                          //Path to our graphicfolder

bmap tut_back_pcx = "tut_back.pcx";                     //backgroundgraphic
bmap mousemap = "tut_mousemap.pcx";                     //mousecursor

//graphics for this tutorial
bmap tut1_input_pcx = "tut1_input.pcx";
bmap tut1_button_pcx = "tut1_button.pcx";
bmap tut1_over_pcx = "tut1_over.pcx";
bmap tut1_choice_pcx = "tut1_choice.pcx";
bmap tut1_selector_pcx = "tut1_selector.pcx";
bmap tut1_sel_dummy_tga = "tut1_sel_dummy.tga";

//needed strings
string main_wmb = "menu.wmb";
string choosen_str[10];
string choice_str = "Smith\nWesson\nHeckler\nKoch\nColt";

font menu_font = "Arial", 1, 18;                        //Arial, bold, size 18
```

First of all I have defined some needed things. Paths, grafics, strings and the used characterset.

```
PANEL back_pan
{
	bmap = tut_back.pcx;
	layer = 1;
	flags = visible;
}

PANEL input_pan
{
	bmap = tut1_input_pcx;
	pos_x = 20;
	pos_y = 20;
	layer = 2;
	flags = visible;
}

PANEL button_pan
{
	bmap = tut1_button_pcx;
	pos_x = 230;
	pos_y = 20;
	button = 0, 0, tut1_button_pcx, NULL, tut1_over_pcx, swap_choice_panel, NULL, NULL;
	layer = 2;
	flags = visible;
}

PANEL choice_pan
{
	bmap = tut1_choice_pcx;
	button = 0, 0, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 18, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 36, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 54, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 72, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	pos_x = 20;
	pos_y = 40;
	layer = 2;
}
```

Here we have now all panels, which we need. The "back_pan" panel is the black background and always visible (flags = visible;). It is first rendered (layer = 1;) and afterwards all other layers are drawn. The other three panels are drawn after the background panel and appear in such a way before it (layer = 2;). For these I indicated still the position on the screen (pos_x, pos_y). All, up to the "choice_pan" panel are shown also immediately. The last panel is only indicated if the button for it is pressed.

```
TEXT input_txt
{
	font = menu_font;
	pos_x = 23;
	pos_y = 22;
	layer = 3;
	strings = 1;
	string = choosen_str;
	flags = visible;
}

TEXT choice_txt
{
	font = menu_font;
	pos_x = 23;
	pos_y = 42;
	layer = 3;
	strings = 1;
	string = choice_str;
}
```

These text definitions are necessary, in order to represent our selected text and the selection list. I set the layer to 3, so that they are not covered by panels.

```
function main()
{
	level_load(main_wmb);
	wait(3);

	str_cpy(choosen_str, "Smith");

	mouse_map = mousemap;
	mouse_mode = 2;

	while(1)
	{
		MOUSE_POS.X = POINTER.X;
		MOUSE_POS.Y = POINTER.Y;
		wait(1);
	}
}

function swap_choice_panel()
{
	choice_pan.visible = (choice_pan.visible == off);
	choice_txt.visible = (choice_txt.visible == off);
}
```

```
function tut1_select_item(button_number, panel)
{
	if(button_number == 1)
	{
		str_cpy(choosen_str, "Smith");
	}
	if(button_number == 2)
	{
		str_cpy(choosen_str, "Wesson");
	}
	if(button_number == 3)
	{
		str_cpy(choosen_str, "Heckler");
	}
	if(button_number == 4)
	{
		str_cpy(choosen_str, "Koch");
	}
	if(button_number == 5)
	{
		str_cpy(choosen_str, "Colt");
	}
	swap_choice_panel();
}
```

In the main function we load our dummy level. This level contains only a simple block with standard texture. Older Acknex versions needed level geometry in the level. And thus this tutorial is downwardcompatible, adheres we to this fact.
After loading the level we wait polite 3 frames, in order to go surely that the level is also completely there. Then we hand a default value over with str_cpy() to the visible string in our selection bar.
Subsequently, we give still a few instructions, so that the pointer of mouse is visible and can be used. As mouse_map the graphic of the pointer of mouse is indicated (ugly graphic, I know. Who has leisure may gladly as first paint a new pointer for the mouse). With mouse_mode = 2; we specify that the pointer of mouse is visible and not affect the standard camera. In the while loop thereafter we nonstop query the mouse coordinates; until the program is terminated.

The function "swap_choice_panel()" serves only and alone the purpose our selection panel (and its text) to show or hide. Surely I could have solved all this with If-cases, but who throws a view in the manual, under the column "ugly code" will be able to reread that the method used here is more elegant and does not need so many code lines.

In the function "tut1_select_item(button_number, panel)" we have something special. Here two parameters can be handed over, and that they become also.
(Button_number, panel) means in this case that the number of the clicked buttons is handed over to this function. The number of the buttons is assigned in sequence in the panel definition, beginning with 1. Now we need to only query, which buttonnummer was handed over and we know, which button was pressed.
Subsequently, we copy still the suitable string in our choosen_str and our selection in the line is already shown. Thus the selection field after the choice is closed automatically we still call at the end "swap_choice_panel()".

That is the whole witchcraft. In the if-cases we could write still further things. But I leave you and your Fantasie.

Roll up

Now we still come to a small change, so that real animation comes also to it. We modify the function "swap_choice_panel()" as followed:

```
function swap_choice_panel()
{
	//choice_pan.visible = (choice_pan.visible == off);
	//choice_txt.visible = (choice_txt.visible == off);
	if(choice_pan.visible == off)
	{
		choice_pan.scale_y = 0.1;
		choice_pan.visible = on;
		while(choice_pan.scale_y < 0.9)
		{
			choice_pan.scale_y += 0.1 * time;
			wait(1);
		}
		choice_pan.scale_y = 1;
		choice_txt.visible = on;
	}
	else
	{
		choice_txt.visible = off;
		while(choice_pan.scale_y > 0.1)
		{
			choice_pan.scale_y -= 0.1 * time;
			wait(1);
		}
		choice_pan.visible = off;
	}
}
```

Thus we cause that the panel with open downward increased and with close upward reduced. We change simply in a loop the y_scale value. Power nevertheless equal more ago, as if the panel is simply only indicated. Experiment a little with the values, in order to find out, what you can do with it.

Blending

A further possibility for the announcement of panels is the fading in and fading out. In addition we change the "swap_choice_panel()" function following:

```
function swap_choice_panel()
{
	if(choice_pan.visible == off)
	{
		choice_pan.alpha = 0;
		choice_txt.alpha = 0;
		choice_pan.visible = on;
		choice_txt.visible = on;
		while(choice_pan.alpha < 100)
		{
			choice_pan.alpha += 5 * time;
			choice_txt.alpha += 5 * time;
			wait(1);
		}
		choice_pan.alpha = 100;
		choice_txt.alpha = 100;
	}
	else
	{
		while(choice_pan.alpha > 0)
		{
			choice_pan.alpha -= 5 * time;
			choice_txt.alpha -= 5 * time;
			wait(1);
		}
		choice_pan.alpha = 0;
		choice_txt.alpha = 0;
		choice_pan.visible = off;
		choice_txt.visible = off;
	}
}
```

With this method we must insert in our panel and text definition still somewhat:

```
TEXT choice_txt
{
	font = menu_font;
	pos_x = 23;
	pos_y = 42;
	layer = 3;
	strings = 1;
	string = choice_str;
	flags = transparent;
}
```

```
PANEL choice_pan
{
	bmap = tut1_choice_pcx;
	button = 0, 0, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 18, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 36, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 54, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	button = 0, 72, tut1_sel_dummy_tga, NULL, tut1_selector_pcx, tut1_select_item, NULL, NULL;
	pos_x = 20;
	pos_y = 40;
	layer = 2;
	flags = transparent;
}
```

Animated buttons

For an animated button I use this graphic:

A similar technique is used also with animated sprites. Here we write our own function. That is not too complicated.

The graphic above is 100 px * 150 px largely. It is partitioned from above downward into five sections to ever 30 px. Each section is thereby a frame, which is successively played.

For this tutorial we use the "tut2_"-graphics and "the tut_2.wdl"-sourcecode.

```
path "gfx_2d";

bmap tut_back_pcx = "tut_back.pcx";
bmap mousemap = "tut_mousemap.pcx";

bmap tut2_anim_button_tga = "tut2_button.tga";
bmap tut2_dummy_tga = "tut2_dummy.tga";

string main_wmb = "menu.wmb";

var offset_y;

function exit_func();
```

First we define again a few paths, graphics, the level string, a variable and a function protoype.

New are the two graphics, but we know that they must be defined before. The variable serves us as counter for the animation. The function prototype is for the function, which with click on our Button is called. That can also be naturally another.

```
PANEL back_pan
{
	bmap = tut_back.pcx;
	layer = 1;
	flags = visible;
}

PANEL animated_button
{
	pos_x = 20;
	pos_y = 20;
	layer = 2;
	button = 0, 0, tut2_dummy_tga, NULL, NULL, exit_func, NULL, NULL;
	window = 0, 0, 100, 30, tut2_anim_button_tga, 1, offset_y;
	flags = visible;
}
```

We already know the "back_pan" panel from in former times, therefore I will say to it nothing more.
In the "animated_button" panel now the instruction "window" was added. With this we realize our animation.

```
function main()
{
	level_load(main_wmb);

	mouse_map = mousemap;
	mouse_mode = 2;

	while(1)
	{
	MOUSE_POS.X = POINTER.X;
    MOUSE_POS.Y = POINTER.Y;
    wait(1);
  }
}
```

```
starter animate_button()
{
	while(1)
	{
		offset_y += 30;
		offset_y %= 150;
		if(offset_y == 0) { wait(-5);	}
		wait(5);
	}
}

function exit_func()
{
	exit;
}
```

The main function should not represent anything special, and also to the "exit_func" function nothing needs to be said.

Interesting for us is only and alone this starter function "animate_button()". In a continuous loop we always increase the variable "offset_y" by 30. With the modulo function in the next line (% =) we tell A6 that offset_y can not be largerthan 150. It will automatically count again from zero.

The IF instruction says us the fact that the loop is to wait 5 seconds if offset_y = 0 (a negative number within a WAIT indicates seconds, positives frames). And afterwards it waits again 5 frames.

What happens exactly?

With the panel definition we used the variable „offset_y" at the "window" instruction. This value indicates the vertical position of the cutout window relative to the left upper corner of our graphic. We remember that I said at the beginning, my graphic would be highly partitioned into five parts, each part of 30 px? Exactly that is it. All 5 Frames the next cutout is shown to the graphic, except if offset_y = 0 is. Then the engine waits 5 seconds (and 5 Frames) and shows then the second part. Simply, or not?

So that the whole becomes also the button, I put simply still another completely transparent graphic over it, which is the contact surface for the button. If the button is clicked, the engine is terminated.

This effect can be used also for 2D games. With a spaceship you could animate the jet-ray with "window". By changing "pos_x" and "pos_y" you could steer the spaceship. Or animate a character, which runs from point A to point B. In game menus you could let the frameworks of the graphic flash. Everything is possible.

Animated Mousecursor

Particularly if you regards Blizzards masterpieces, you can recognize that gladly animated pointers of mouse are used. As simply it goes I show you now. (we use the graphics "tut3_" and the script "tut_3.wdl")

First we need at least two graphics for the mousepointer. In my example I use three:

The graphics are in each case 16 px * 32 px largely. That is the prescribed size for a mousepointer in fullscreen mode. The graphics possess an alpha channel and are stored as 32 bits TGA.

We now come to the complex Script:

```
path "gfx_2d";

bmap tut_back_pcx = "tut_back.pcx";

bmap mouse1_tga = "tut3_mouse1.tga";
bmap mouse2_tga = "tut3_mouse2.tga";
bmap mouse3_tga = "tut3_mouse3.tga";

string main_wmb = "menu.wmb";

PANEL back_pan
{
	bmap = tut_back.pcx;
	layer = 1;
	flags = visible;
}

function main()
{
	level_load(main_wmb);

	mouse_mode = 2;

	while(1)
	{
		MOUSE_POS.X = POINTER.X;
		MOUSE_POS.Y = POINTER.Y;
		wait(1);
	}
}
```

```
starter animate_button()
{
	while(1)
	{
		mouse_map = mouse3_tga;
		wait(50);
		mouse_map = mouse2_tga;
		wait(50);
		mouse_map = mouse1_tga;
		wait(50);
	}
}
```

That should be everything? Yes, more does not come really. Everything in the script is well-known. Completely down a starter function with a continuous loop stands. There in it we change all 50 frames the graphic for the mousepointer. That graphics can be theoretically as many as desired, but remember, an animated mousepointer is only a small plaything.

Bars

We come to bar line displays. These are useful, if we want to show the health of a character or a loading progress bar. And naturally for a quantity of other things more. We need for it 2 graphics. Those are with the prefix "tut4 _". The code stand in "tut_4.wdl".

Both are TGA graphics with alpha channel (the black ranges). The lower graphic consists of two parts. The left, multicolored part is just as long thereby as the transparent right part (both 124 px). The transparent range of the upper graphic is likewise 124 px long.

```
path "gfx_2d";

bmap tut_back_pcx = "tut_back.pcx";
bmap mousemap = "tut_mousemap.pcx";

bmap tut4_holder_tga = "tut4_holder.tga";
bmap tut4_bar_tga = "tut4_bar.tga";

string main_wmb = "menu.wmb";

var health = 100;
var health_status;

function health_bar();
```

Our usual definitions. The two variables are necessary for the computation of the bar position. In our example we want to compute the life energy of a character.

```
PANEL back_pan
{
	bmap = tut_back.pcx;
	layer = 1;
	flags = visible;
}

PANEL health_pan
{
	bmap = tut4_holder_tga;
	pos_x = 20;
	pos_y = 20;
	layer = 3;
	flags = visible;
}

PANEL health_bar_pan
{
	pos_x = 36;
	pos_y = 23;
	window = 0, 0, 124, 17, tut4_bar_tga, health_status, 1;
	layer = 2;
	flags = visible;
}
```

Also with the panel definitions there is nothing special. We work again with the window instruction, because we want always show a certain cutout of the bar. Since we shift this time the graphic horizontal, stands "health_status" in the place for the x-variable.

```
function main()
{
	level_load(main_wmb);

	mouse_map = mousemap;
	mouse_mode = 2;

	while(1)
	{
		health_bar();

		MOUSE_POS.X = POINTER.X;
		MOUSE_POS.Y = POINTER.Y;
		wait(1);
	}
}
```

```
function health_bar()
{
	if(health > 0)
	{
		health_status = 124 - int((health / 100) * 124);
	}
	else
	{
		health_status = 124;
	}
	health += 0.3 * time;
	if(health > 100) { health = 100; }
}

function hurt_func()
{
	health -= 10;
	while(key_pressed(35)){ wait(1); }
}

on_h = hurt_func;
```

We already know the "main" function. However I set function call into the continuous loop still another.
The function "health_bar()" computes us in each frame the "health status" variable, which supplies us the correct position for the bar in the "window" instruction. The last two lines regenerate the life energy and guarantee that them do not become larger than 100.

I inserted the "hurt_func()" function to test purposes. As soon as H is pressed, 10 points of life are taken off.

Slider

There are many possibilities, with which a sliding control is needed. It is to be adjusted with the gamma correction in the graphic options or around the volume of the sound. C-Script gives us for it a outstanding tool in the hand: Slider. For this example I use the graphics "tut5_", the script is in the file "tut_5.wdl".

We need two graphics:

The code is really simple and looks like:

```
path "gfx_2d";

bmap tut_back_pcx = "tut_back.pcx";
bmap mousemap = "tut_mousemap.pcx";

bmap tut5_bar_tga = "tut5_bar.tga";
bmap tut5_slider_tga = "tut5_slider.tga";
bmap tut5_pan_tga = "tut5_pan.tga";

string main_wmb = "menu.wmb";

var test_pan_alpha = 100;
```

Our usual definitions.

```
PANEL back_pan
{
	bmap = tut_back.pcx;
	layer = 1;
	flags = visible;
}

PANEL test_pan
{
	bmap = tut5_pan_tga;
	layer = 2;
	pos_x = 20;
	pos_y = 50;
	alpha = 100;
	flags = visible | translucent;
}

PANEL bar_pan
{
	bmap = tut5_bar_tga;
	layer = 2;
	pos_x = 20;
	pos_y = 20;
	hslider = 0, 0, 200, tut5_slider_tga, 0, 100, test_pan_alpha;
	flags = visible;
}
```

The „back_pan“ is well-known.

In "test_pan" I set an alpha value. It determines the transparency of the panel. At the beginning it is to be completely intransparent. You have to set the „translucent“-flag.

In "bar_pan" I used "hslider", in order to receive a horizontal sliding control. On a length of 200 pixels I can change the value of the variable "test_pan_alpha" between

0 and 100.

```
function main()
{
	level_load(main_wmb);

	mouse_map = mousemap;
	mouse_mode = 2;

	while(1)
	{
		test_pan.alpha = test_pan_alpha;
		MOUSE_POS.X = POINTER.X;
		MOUSE_POS.Y = POINTER.Y;
		wait(1);
	}
}
```

This a line in our continuous loop equates the value the variable "test_pan_alpha" with the alpha value of the panel "test_pan". That is already everything. Simple, isn't it?

Epilogue

With these suggestions it should be possible for you to arrange your own more exciting animated game menu, exite your HUDs or create your own 2D game. Experiment a bit, combine the effects or invents completely new things in addition. It is important that it looks good.

Expert tip!

No matter how ingeniously you find your effects for menus and how complex the animation was to provide, consider with the fact that the final user is possibly bored of if he must wait for a long time for the fact that a menu appears. Keep your effects therefore short and scarce.

An effect should last long enough that the user receives it, but it should be short enough, in order not to bore the user.

Crazy 3D graphic and complex shader effects is nowadays an important tool for good-looking and professional games. But evenly also in 2D area the same applies. The game menu is nevertheless the first a player gets to face. And like you know: The first impression is the crucial one.

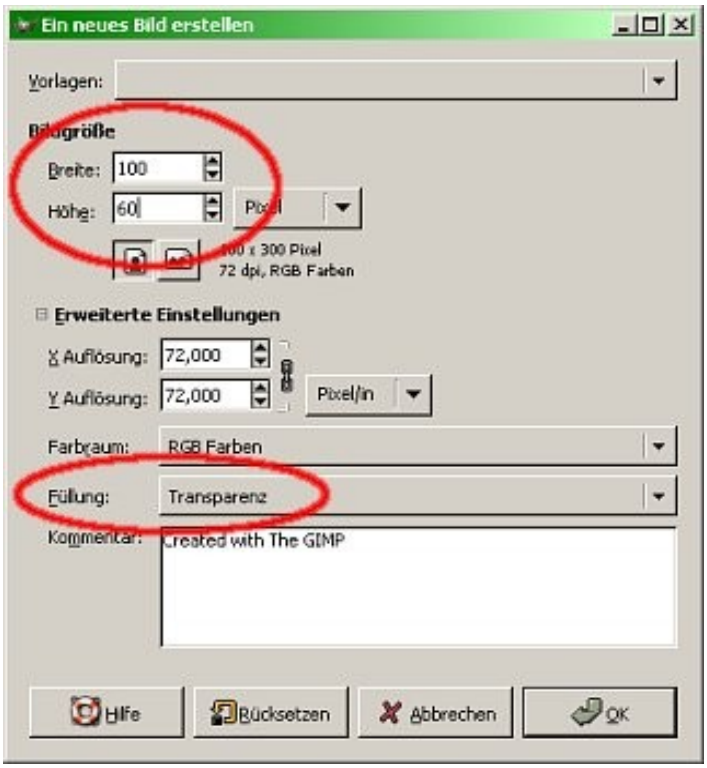
In this sense much success with your projects

Tobias „LordRathan“ Mogdans

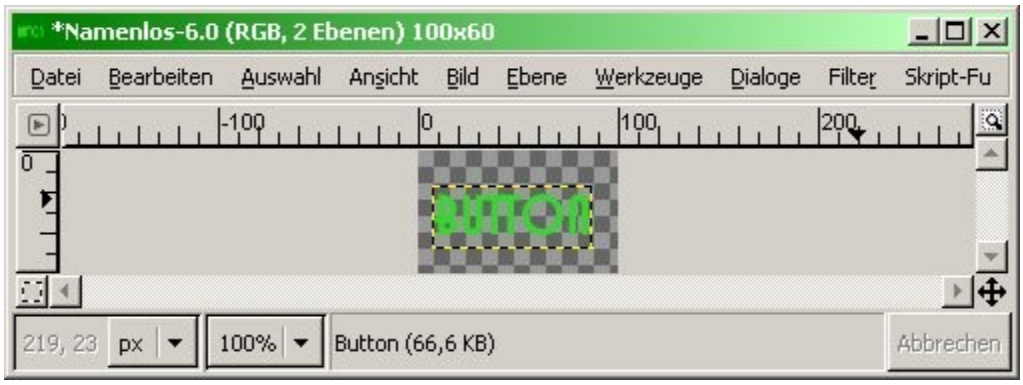
Appendix

Here still another word is said to the production of transparent TarGa graphics with TheGIMP. For new users it may come a little to confusion, it is however completely simple in principle.

With the combination of keys Ctrl + N provides it a new picture. A window appears, in which you specify the size. Click on Filling and choose „transparency". Click on "OK" provides the new picture.



Now a transparent picture expects you, recognizably from the lightgrey-darkgrey tile sample. You can paint now here your graphic as desired, e.g. a signature.



With SHIFTS + CTRL + S store the picture under a new name. If it calls after own discretion, do not forget however the ending ".tga" behind it to write. Your 32 bits TGA graphic with alpha channel is finished.