Pimp my Project

2D Tutorial

für Einsteiger

von Tobias "LordRathan" Mogdans

Version: 1.0

Inhalt

Vorwort

In diesem Tutorial möchte ich Einsteigern die Materie von Spielmenüs näherbringen. Schöne und übersichtliche Menüs sind sehr wichtig für erfolgreiche Spiele und erfreuen das Auge des Benutzers.

Dieses Tutorial erhebt bestimmt keinen Anspruch auf Vollständigkeit oder darauf, dass es der beste oder einzige Weg ist, um 2D-Work zu verwirklichen. Es ist der Weg, den ich mir erarbeitet habe und der für mich gut funktioniert.

Ein Hinweis noch zu Beginn: Es gibt den Panel-Editor als Plugin für SED. Das vergessen wir gleich wieder, denn wir wollen schließlich wissen was wir programmieren. Es gibt keinen Grund, den Editor zu nutzen. Panels sind äußerst einfach umzusetzen.

Wir werden uns nun ein wenig Theorie mit ein paar Beispielen ansehen. Der zweite Teil behandelt einige Codeschnipsel als Anregungen und Ideen.

Viel Spaß dabei.

Theorie der 2D-Programmierung in C-Script

Der wichtigste Teil der Panelprogrammierung ist die Definition eines Panels. Schauen wir uns den Code dafür einmal an:

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

Wir haben mit diesen Zeilen A6 mitgeteilt, dass es ein neues Panel mit dem Namen „newpanel" gibt. Sieht ein wenig wie eine Funktion aus, nur ohne Klammern nach dem Namen.

Panels sind elementare Bestandteile eines 2D-Spieles und von wichtiger Bedeutung für die Benutzeroberfläche von 3D-Spielen. Wir können mit Panels Inventare, Knöpfe, Lebensanzeigen, Kompasse, Zielkreuze, Hauptmenüs, Missionsbriefings, Tooltips und vieles, vieles mehr realisieren.

Um so etwas zu erstellen benötigen wir aber mehr als nur eine leere Panel-Definition wie oben. Panels können Hintergrundbilder, anklickbare Knöpfe, Schieberegler, Zahlen und Text.

Wir wollen natürlich bestimmen können, an welcher Stelle auf dem Bildschirm unser Panel erscheint:

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

Unser Panel hält einen Abstand zum linken Bildschirmrand von 100 Pixel und einen Abstand zum oberen Rand von 150 Pixel. In Funktionen können wir darauf zugreifen und die Position des Panels verändern. So sind einfache Animationen möglich, aber auch für komplexere Animationen müssen wir die Position ändern.

Bisher ist unser Panel noch leer und unsichtbar. A6 weiss zwar, dass es ein Panel namens „newpanel" gibt und auch das es an Position 100, 150 erscheinen soll. Wir haben bisher nur noch nicht festgelegt WAS erscheinen soll.

```
BMAP backpanel_tga = „green.tga“;

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

Jetzt gibt es zum ersten Mal etwas auf dem Bildschirm zu sehen. Was haben wir gemacht?
Zuerst haben wir außerhalb der Panel-Definition eine Grafik definiert. BMAP ist der Typ für Grafiken, **backpanel_tga** ist der Name, mit dem wir auf diese Grafik zugreifen können, und **green.tga** ist der Name der Grafik, die vorher mit einem Grafikprogramm erstellt wurde.
Innerhalb der Paneldefinition haben wir mit **bmap = backpanel_tga;** die definierte Grafik unserem Panel als Hintergrundbild zugewiesen.
Das Flag **visible** in der letzten Zeile bestimmt, dass das Panel sofort sichtbar ist.

Expertentip!

Als Grafikformat für Panels kann Windows Bitmap (.bmp), ZSoft PCX (.pcx) und TarGA (.tga) verwendet werden.

Von Bitmap rate ich ab, da die gleiche Qualität auch mit PCX erreicht werden kann, die Grafiken aber weniger Videospeicher verbrauchen. Die Farbe Schwarz (0, 0, 0) definiert bei beiden Formaten Tranzparenz.

TarGA ist ein nützliches Format, da es einen Alphakanal beinhalten kann. Für komplexere Grafiken lohnt sich das Mehr an verbrauchtem Videospeicher auf jeden Fall.

Das ist alles was wir brauchen um Splashscreens oder Ladebilder anzuzeigen. Nun wollen wir aber auch mit unserem Panel interagieren können.
Sehr häufig benötigen wir Knöpfe, um eine Meldung zu bestätigen oder um eine Auswahl treffen zu können.

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;
}
```

Ein sehr umfangreiches Element. Was genau haben wir hier gemacht?
Zuerst haben wir wieder ausserhalb der Panel-Definition drei neue Grafiken definiert. Diese Grafiken nehmen wir für die drei Zustände, die der Knopf annehmen kann:

1. Der Knopf wird angeklickt
2. Der Knopf wird nicht angeklickt und der Mauszeiger ist irgendwo anders
3. Der Mauszeiger schwebt über dem Knopf, aber es wird nicht geklickt

In der Panel-Definition selbst haben wir einen Knopf definiert (**button**). Die beiden Zahlen geben an, an welcher Stelle sich der Knopf befinden soll. Zu beachten ist hierbei, dass sich die Koordinaten auf die Position des Panels selbst beziehen, d. h. der Knopf befindet sich 40 Pixel von der linken Panelkante und 10 Pixel von der oberen Panelkante entfernt. Das ist sehr nützlich, denn wenn wir das Panel bewegen wollen müssen wir uns nicht darum kümmern, dass der Knopf auch mitbewegt wird. Das geschieht automatisch, denn der Knopf ist immer auf der selben Position von der linken oberen Panelkante entfernt.
Die nächsten drei Parameter geben an, welche Grafiken für die drei Zustände verwendet werden sollen. Der erste Parameter für „angeklickt“ muss angegeben werden. Die Größe dieser Grafik bestimmt nämlich die Größe des Knopfes. Die anderen beiden Parameter können auch mit NULL angegeben werden, allerdings wird dann für diesen Zustand auch keine eigene Grafik angezeigt.

Die restlichen drei Parameter sind für Funktionsaufrufe. Wenn wir den Knopf drücken soll schließlich auch etwas passieren.

Der erste Parameter ruft eine Funktion auf, wenn der Knopf angeklickt wird. Dieser Parameter wird wohl am häufigsten verwendet. Der zweite Parameter ruft seine Funktion auf, wenn die Maustaste losgelassen wird oder wenn der Mauszeiger von dem Knopf entfernt wird. Der Dritte Parameter ruft seine Funktion auf, wenn der Knopf berührt wird. Wenn z. B. ein Geräusch ertönen soll sobald der Knopf berührt wird, ist der letzte Parameter unser Freund.

Zusätzlich kann noch die Nummer des Knopfes an die ausführende Funktion übergeben werden. Das kann sehr nützlich sein, falls mehrere Knöpfe dieselbe Funktion verwenden. Der erste Knopf ist dabei automatisch Knopf 1, der Zweite Knopf 2 usw...

Digits

Panels können natürlich gut verwendet werden um die Grafische Benutzerschnittstelle darzustellen. In vielen Spielen will die Lebensenergie des Spielers dargestellt werden. Wir könnten das mit simplen Zahlen machen.

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

Die ersten beiden Parameter geben – wie auch bei Button - die Position des Elements innerhalb des Panels wieder.

Der dritte Parameter gibt das Format an. Für Zahlen ist das noch sehr einfach. Der ganzzahlige Anteil bestimmt die gesamten Stellen für die angezeigte Zahl, die Kommastelle gibt an, wieviele Stellen nach dem Komma stehen. Eine Stelle ist bei Kommazahlen für den Dezimalpunkt reserviert.
6.2 würde bewirken, dass wir insgesamt 6 Stellen erhalten, wobei 2 Stellen für nach dem Komma stehen und eine Stelle für den Dezimalpunkt. Somit hätten wir noch 3 Stellen vor dem Komma, z. B. 103.76. Wenn das Format negativ angegeben wird, also -6.2, dann würden bei Zahlen unter 100 Nullen vorangestellt, z. B. 004.86

Der vierte Parameter bestimmt den Zeichensatz. * wird hier für den Engineeigenen Zeichensatz verwendet. Natürlich kann man auch einen zuvor definierten Zeichensatz einfügen.

Der fünfte Parameter steht für den Multiplikator. Das angezeigte Ergebnis wird also mit dieser Zahl multipliziert. Für unsere Lebensanzeige spielt das keine Rolle, deswegen multiplizieren wir mit 1.

Und zuletzt gibt der letzte Parameter an, was überhaupt angezeigt werden soll. Das kann eine Variable sein oder auch ein Skill einer Entity.

Expertentip!

Der dritte Parameter kann seit der Version 6.4 auch dazu verwendet werden um die Ausgabe mit Hilfe von C-Formatstrings zu formatieren. Dazu schreiben wir in Anführungszeichen den Formatstring.

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

[text1] = Ein optional vorangestellter Text
[flags] = optional ein oder mehrere Zeichen
[width] = minimale Zahl der Stellen
[.prec] = maximale Zahl der Nachkommastellen
[text2] = Ein optional nachgestellter Text

Anstatt dieser ganzen Parameter kann auch ein maximal 100 Zeichen langer Text verwendet werden. Dieser Text darf allerdings nicht aus nur einer Zahl bestehen.

Window

Window zeigt einen Ausschnitt aus einer Grafik an. Mit window kann man Panels animieren, Balkenanzeigen realisieren oder einen Kompass darstellen.

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

Die ersten beiden Parameter geben an, an welcher Stelle des Panels das Window erscheinen soll. Die beiden nächsten Parameter legen fest, wie groß der Ausschnitt sein soll. Der fünfte Parameter legt fest, welche Grafik verwendet wird, und die letzten beiden Parameter bestimmen, wo auf der Grafik angefangen wird auszuschneiden.

Slider

Slider gibt es in zwei verschiedenen Ausführungen, einmal für horizontale und einmal für Vertikale Slider. Sie eignen sich gut für Funktionen, in denen der Benutzer den Wert einer Variablen zwischen zwei festgelegten Werten beliebig ändern kann.

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

Die ersten beiden Parameter geben wieder die Position innerhalb des Panels an. Der dritte Parameter beschreibt, wieviele Pixel der Gesamtweg des Sliders beträgt. Im Beispiel kann ich den Slider 200 Pixel weit von links nach rechts bewegen. Im vierten Parameter wird die Grafik des Sliders angegeben.
Die beiden nächsten Parameter geben den minimalen und den maximalen Wert an, den die Variable im letzten Parameter annehmen kann. In meinem Beispiel kann die Variable „test_pan_alpha“ also zwischen 0 und 100 liegen.

Flags

Die Flags der Panels geben jetzt noch ein paar Eigenschaften mit.

- visible
- overlay
- translucent
- filter
- light

Visible ist dabei, wie bei allen Objekten in C-Script, dafür zuständig, ob das Panel angezeigt wird oder unsichtbar ist.
Ist das **Overlay**-Flag gesetzt, sind alle schwarzen Bereiche (0, 0, 0) der Panelgrafik komplett transparent. Für TGA-Grafiken mit Alphakanal benötigen wir dieses Flag nicht.
Translucent bestimmt seit Version 6.4 ob der Alpha-Wert eines Panels Einfluß besitzt, oder nicht. Wenn ihr also einblendbare Panels erstellt, achtet darauf, dass dieses Flag gesetzt ist.
Filter kann bei bestimmten Effekten zu einer Verbesserung der Darstellung führen, vor allem, wenn ihr mit Skalierungseffekten arbeitet. Bei sehr kleinen Schriften ist dieses Flag allerdings kontraproduktiv, denn es verwischt die Pixel.
Light schlußendlich lässt noch schöne Effekte zu, denn es moduliert die Elemente auf dem Panel durch die Red, Blue und Green-Werte des Panels.

Wenn mehr als ein Flag verwendet wird, wird zur Trennung seit 6.4 das Oder-Zeichen „|" verwendet. Aber keine Angst, ein normales „+" funktioniert auch noch.

```
Flags = visible | translucent | filter;     //Neuer Code, Version 6.4 und neuer

oder

Flags = visible + translucent + filter;  //Alter Code, Version 6.34.1 und älter
```

Red, Green, Blue und Alpha

Erwähnenswert sind noch die Befehle Red, Green und Blue. Mit ihnen kann der Farbwert des Panels bestimmt werden, ähnlich wie beim Licht. Schonmal daran gedacht, den User die Farbe seines HUDs selbst einstellen zu lassen? Mit ein paar Sliedern und diesen drei Werten dürfte das überhaupt kein Problem darstellen.

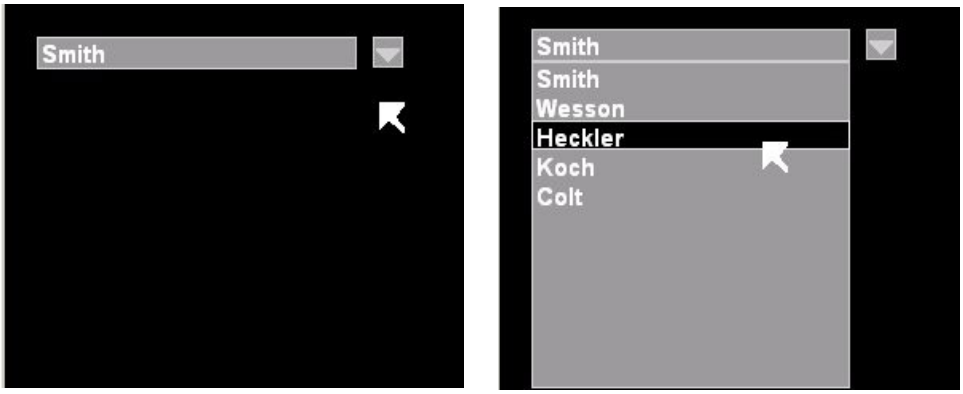
Der Alpha-Wert gibt, ähnlich wie bei Modellen, die Transparenz des Panels wieder. Der Wert reicht dabei von 0 (völlig transparent) bis 100 (nicht transparent). Zu beachten hierbei ist aber dass das Flag „translucent" gesetzt sein muss.

Anregungen, Ideen & Codeschnipsel

Es folgen nun einige Ideen und Beispiele, die für Animationen in Menüs verwendet werden können.

Drop-Down

Eines der simpelsten Menüfunktionen ist wohl ein Drop-Down Auswahlfeld.



Wir benötigen dazu drei wichtige Grafikelemente: die Zeile, in der die Auswahl steht, die Auswahlliste und optional einen Button, um die Auswahlliste anzuzeigen oder zu verstecken. Natürlich könnte man den Button in die Auswahlzeile integrieren, aber wir beleuchten hier mal die optionale Variante. Desweiteren benutzen wir noch einige Dummy-Grafiken und Verschönerungen. Für dieses Beispiel habe ich die „tut1_" Grafiken verwendet. Ihr findet sie im Unterordner „gfx_2d".
Der fertige Code findet sich in der Datei „tut_1.wdl".

```
path "gfx_2d";                                          //Der Pfad zu unserem Grafikordner

bmap tut_back_pcx = "tut_back.pcx";                     //Die Grafik des Hintergrundes
bmap mousemap = "tut_mousemap.pcx";                     //Mauszeigergrafik

//Grafiken für dieses 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";

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

font menu_font = "Arial", 1, 18;                        //Arialschrift, Fett, Größe 18
```

Zuerst habe ich hier einige Dinge definiert. Pfade, Grafiken, Strings und unseren verwendeten Zeichensatz.

```
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;
}
```

Hier haben wir jetzt alle Panels, die wir brauchen. Das „back_pan"-Panel ist der schwarze Hintergrund und immer sichtbar (flags = visible;). Es wird zuerst gerendert (layer = 1;) und danach werden alle anderen Layer gezeichnet.
Die anderen drei Panels werden nach dem Hintergrundpanel gezeichnet und erscheinen so davor (layer = 2;). Bei diesen habe ich noch die Position auf dem Bildschirm angegeben (pos_x, pos_y). Alle, bis auf das „choice_pan"-Panel werden auch sofort angezeigt. Das letzte Panel wird erst angezeigt, wenn der Button dafür gedrückt wird.

```
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;
}
```

Diese Textdefinitionen sind nötig, um unseren gewählten Text und die Auswahlliste darzustellen. Die Layer habe ich auf 3 gesetzt, damit sie nicht von Panels verdeckt werden.

```
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 der Main-Funktion laden wir unser Dummy-Level. Dieser Level beinhaltet nur einen einfachen Block mit Standard-Textur. Ältere Acknex-Versionen benötigten Levelgeometrie im Level. Und damit dieses Tutorial Abwärtskompatibel ist, halten wir uns an diese Tatsache.
Nach dem Levelladen warten wir höfliche 3 Frames, um sicher zu gehen, dass der Level auch komplett da ist. Dann übergeben wir mit str_cpy() einen Default-Wert an den sichtbaren String in unserem Auswahlbalken.
Anschließend geben wir noch ein paar Anweisungen, damit der Mauszeiger sichtbar ist und verwendet werden kann. Mit mouse_map wird die Grafik des Mauszeigers angegeben (hässliche Grafik, ich weiss. Wer Muße hat, darf gerne als erstes den Mauszeiger verschönern). Mit mouse_mode = 2; legen wir fest, dass der Mauszeiger sichtbar ist und nicht die Standardkamera beeinflußt. In der While-Schleife danach fragen wir pausenlos die Mauskoordinaten ab; solange, bis das Programm beendet wird.

Die Funktion „swap_choice_panel()" dient einzig und allein dem Zweck unser Auswahl-Panel (und seinen Text) anzuzeigen oder zu verstecken. Sicherlich hätte ich das alles mit If-Fällen lösen können, doch wer einen Blick ins Handbuch wirft, wird unter der Rubrik „Hässlicher Code" nachlesen können, dass die hier verwendete Methode eleganter ist und nicht soviele Code-Zeilen beansprucht.

In der Funktion „tut1_select_item(button_number, panel)" haben wir etwas Besonderes. Hier können zwei Parameter übergeben werden, und das werden sie auch. (Button_number, panel) bedeutet in diesem Fall, dass die Nummer des angeklickten Buttons an diese Funktion übergeben wird. Die Nummer des Buttons wird der Reihe nach in der Panel-Definition vergeben, beginnend mit 1. Jetzt brauchen wir nur noch abzufragen, welche Buttonnummer übergeben wurde und schon wissen wir, welcher Button gedrückt wurde. Anschließend kopieren wir noch den passenden

String in unseren choosen_str und schon wird unsere Auswahl in der Zeile angezeigt. Damit das Auswahlfeld nach der Wahl automatisch geschlossen wird rufen wir am Ende noch „swap_choice_panel()“ auf.

Das ist die ganze Hexerei. In den If-Fällen könnten wir noch weitere Dinge hineinschreiben, falls noch etwas weiteres mit einer getroffenen Auswahl passieren soll. Aber das überlasse ich euch und eurer Fantasie.

Aufrollen

Jetzt kommen wir noch zu einer kleinen Änderung, damit auch wirklich Animation mit dazu kommt. Wir ändern die Funktion „swap_choice_panel()“ folgend um:

```
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;
	}
}
```

Damit bewirken wir dass sich das Panel beim öffnen nach unten hin vergrößert und beim schließen nach oben verkleinert. Wir ändern einfach in einer Schleife den y_scale-Wert. Macht doch gleich mehr her, als wenn das Panel einfach nur angezeigt wird. Experimentiert ein wenig mit den Werten, um herauszufinden, was ihr noch damit anstellen könnt.

Einblenden

Eine weitere Möglichkeit für das Anzeigen von Panels ist das sanfte Ein- und Ausblenden. Dazu ändern wir die „swap_choice_panel()"-Funktion folgend ab:

```
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;
	}
}
```

Bei dieser Methode müssen wir in unserer Panel- und Textdefinition noch etwas einfügen:

```
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;
}
```

Animierte Buttons

Für einen animierten Button verwende ich diese Grafik:

So eine ähnliche Technik kommt auch bei animierten Sprites zum Einsatz. Hier schreiben wir uns die Funktion allerdings selber. Das ist nichtmal kompliziert.

Die Grafik oben ist 100 px * 150 px groß. Dabei ist sie von oben nach unten in fünf Abschnitte zu je 30 px unterteilt. Jeder Abschnitt ist dabei ein Frame, das nacheinander abgespielt wird.

Für dieses Tutorial benutzen wir die „tut2_"-Grafiken und den „tut_2.wdl"-Quellcode.

```
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();
```

Zuerst definieren wir wieder ein paar Pfade, Grafiken, den Levelstring, eine Variable und einen Funktions-Protoyp.

Neu sind zwar die beiden Grafiken, aber wir wissen ja, dass sie vorher definiert werden müssen. Die Variable dient uns als Zähler für die Animation. Der Funktionsprototyp ist für die Funktion, die mit Klick auf unseren Button ausgelöst wird. Das kann natürlich auch eine andere sein.

```
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;
}
```

Das „back_pan"-Panel kennen wir ja schon von früher, deswegen werde ich dazu nichts mehr sagen.
Im „animated_button"-Panel ist jetzt der Befehl „window" dazu gekommen. Mit diesem realisieren wir unsere 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;
}
```

Die Main-Funktion sollte nichts besonderes darstellen, und auch zur „exit_func"-Funktion braucht nichts gesagt werden.

Interessant ist für uns einzig und allein diese Starter-Funktion „animate_button()". In einer Endlos-Schleife erhöhen wir die Variable „offset_y" immer um 30. Mit der Modulo-Funktion in der nächsten Zeile (%=) sagen wir A6, dass offset_y niemals größer als 150 sein kann. Es wird also automatisch wieder von 0 angefangen.

Die IF-Anweisung sagt uns, dass die Schleife 5 Sekunden warten soll, wenn offset_y = 0 ist (eine negative Zahl bei einem Wait gibt Sekunden an, eine Positive Frames). Und anschließend wird nochmals 5 Frames gewartet.

Was passiert da nun genau?

Bei der Panel-Definition haben wir beim „window"-Befehl die Variable „offset_y" verwendet. Dieser Wert gibt die vertikale Position des Ausschnittsfensters relativ zur linken oberen Ecke unserer Grafik an. Wir erinnern uns, dass ich am Anfang sagte, meine Grafik wäre in fünf Teile, jedes Teil 30 px hoch, unterteilt? Genau das ist es. Alle 5 Frames wird der nächste Teilausschnitt der Grafik angezeigt, ausser wenn offset_y = 0 ist. Dann wartet die Engine 5 Sekunden (und 5 Frames) und zeigt dann erst den zweiten Teil. Einfach, oder?

Damit das ganze auch zum Button wird, habe ich einfach noch eine komplett transparente Grafik darüber gelegt, welche die Berührungsfläche für den Button ist. Wenn der Button also angeklickt wird, wird die Engine beendet.

Dieser Effekt kann auch für 2D-Games verwendet werden. Bei einem Raumschiff könnte man mit „window" den Antrieb animieren. Durch Ändern von „pos_x" und „pos_y" könnte man das Raumschiff steuern. Oder animiert damit einen Charakter, der von Punkt A nach Punkt B läuft. In Game-Menüs könnte man den Rahmen der Grafik blinken lassen. Alles ist möglich.

Animierter Mauszeiger

Besonders wenn man sich Blizzards Meisterwerke ansieht, kann man erkennen, dass gerne animierte Mauszeiger verwendet werden. Wie einfach das geht zeige ich euch jetzt. (Wir verwenden die Grafiken „tut3_" und das Script „tut_3.wdl")

Zunächst benötigen wir mindestens zwei Grafiken für den Mauszeiger. In meinem Beispiel verwende ich drei:

Die Grafiken sind jeweils 16 px * 32 px groß. Das ist die vorgeschriebene Größe für einen Mauszeiger im Vollbildmodus. Die Grafiken besitzen einen Alphakanal und sind als 32 Bit TGA gespeichert.

Kommen wir nur zu dem komplexen 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);
	}
}
```

Das soll alles sein? Jawohl, mehr kommt wirklich nicht. Alles in dem Script ist bekannt. Ganz unten steht eine Starter-Funktion mit einer Endlos-Schleife. Dort drin ändern wir alle 50 Frames die Grafik für den Mauszeiger. Das können theoretisch beliebig viele Grafiken sein, aber denkt daran, ein animierter Mauszeiger ist lediglich eine kleine Spielerei.

Balkenanzeige

Kommen wir zu Balkenanzeigen. Diese sind nützlich, wenn wir Ladebalken oder die Gesundheit eines Charakters anzeigen wollen. Und natürlich für eine Menge anderer Dinge mehr.
Wir benötigen dafür 2 Grafiken. Das sind die mit dem Präfix „tut4_“. Der Code steht in „tut_4.wdl“;

Beides sind TGA-Grafiken mit Alpha-Kanal (die schwarzen Bereiche). Die untere Grafik besteht aus zwei Teilen. Der linke, bunte Teil ist dabei genauso lang wie der transparente rechte Teil (beide 124 px). Der transparente Bereich der oberen Grafik ist ebenfalls 124 px lang.

```
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();
```

Unsere üblichen Definitionen. Die beiden Variablen sind für die Berechnung der Balkenposition nötig. In unserem Beispiel wollen wir die Lebensenergie eines Charakters berechnen.

```
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;
}
```

Auch bei den Paneldefinitionen gibt es nichts besonderes. Wir arbeiten wieder mit dem window-Befehl, weil wir immer einen bestimmten Ausschnitt des Balkens zeigen wollen. Da wir diesmal die Grafik horizontal verschieben, steht „health_status" an der Stelle für die 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;
```

Die „main"-Funktion kennen wir ja schon. Allerdings habe ich in die Endlosschleife noch einen Funktionsaufruf gesetzt.
Die Funktion „health_bar()" berechnet uns in jedem Frame die „health-status"-Variable, die uns die richtige Position für den Balken in der „window"-Anweisung liefert. Die letzten beiden Zeilen regenerieren die Lebensenergie und stellen sicher, dass sie nicht größer als 100 wird.

Die „hurt_func()"-Funktion habe ich zu testzwecken eingebaut. Sobald H gedrückt wird, werden 10 Lebenspunkte abgezogen.

Schieberegler

Es gibt viele Möglichkeiten, bei denen ein Schieberegler benötigt wird. Sei es bei der Gamma-Korrektur in den Grafikoptionen oder um die Lautstärke des Sounds einzustellen. C-Script gibt uns dafür ein hervorragendes Werkzeug in die Hand: Slider. Für dieses Beispiel verwende ich die Grafiken „tut5_", das Script findet sich in der Datei „tut_5.wdl".

Wir benötigen zwei Grafiken:

Der Code ist wirklich einfach und sieht so aus:

```
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;
```

Unsere üblichen Definitionen.

```
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;
}
```

Das „back_pan" ist bekannt.

Im „test_pan" habe ich einen Alpha-Wert gesetzt. Er bestimmt die Transparenz des Panels. Zu beginn soll es komplett undurchsichtig sein. Damit das aber funktioniert muss auch bei den Flags „translucent" gesetzt sein.

Im „bar_pan" habe ich „hslider" eingesetzt, um einen horizontalen Schieberegler zu

erhalten. Auf einer Länge von 200 Pixeln kann ich den Wert der Variable „test_pan_alpha" zwischen 0 und 100 ändern.

```
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);
	}
}
```

Diese eine Zeile in unserer Endlosschleife setzt den Wert der Variable „test_pan_alpha" mit dem Alpha-Wert des Panels „test_pan" gleich. Das ist schon alles. Einfach, nicht?

Nachwort

Mit diesen Anregungen sollte es euch nun möglich sein, ein eigenes, animiertes Game-Menü zu erstellen, eure HUD's aufregender zu gestalten oder gar ein 2D-Spiel zu designen. Experimentiert ein wenig rum, kombiniert die Effekte oder erfindet völlig neue Sachen dazu. Wichtig ist, dass es gut aussieht. Tolle 3D Grafik und aufwändige Shadereffekte sind heutzutage ein wichtiges Werkzeug für gutaussehende und professionelle Spiele. Doch eben auch im 2D-Bereich gilt dasselbe. Immerhin ist das Game-Menü das Erste, was ein Spieler zu Gesicht bekommt. Und wie ihr wisst: Der erste Eindruck ist der Entscheidende.

Expertentip!

Egal wie genial ihr eure Effekte für Menüs findet und wie aufwändig die Animation zu erstellen war, bedenkt dabei, dass der Endanwender möglicherweise davon gelangweilt ist, wenn er zu lange darauf warten muß, dass ein Menü erscheint. Haltet eure Effekte deswegen kurz und knapp.

Ein Effekt sollte lang genug dauern, dass der Benutzer ihn mitbekommt, aber er sollte kurz genug sein, um den Benutzer nicht zu langweilen.

Tolle 3D Grafik und aufwändige Shadereffekte sind heutzutage ein wichtiges Werkzeug für gutaussehende und professionelle Spiele. Doch eben auch im 2D-Bereich gilt dasselbe. Immerhin ist das Game-Menü das Erste, was ein Spieler zu Gesicht bekommt. Und wie ihr wisst: Der erste Eindruck ist der Entscheidende.

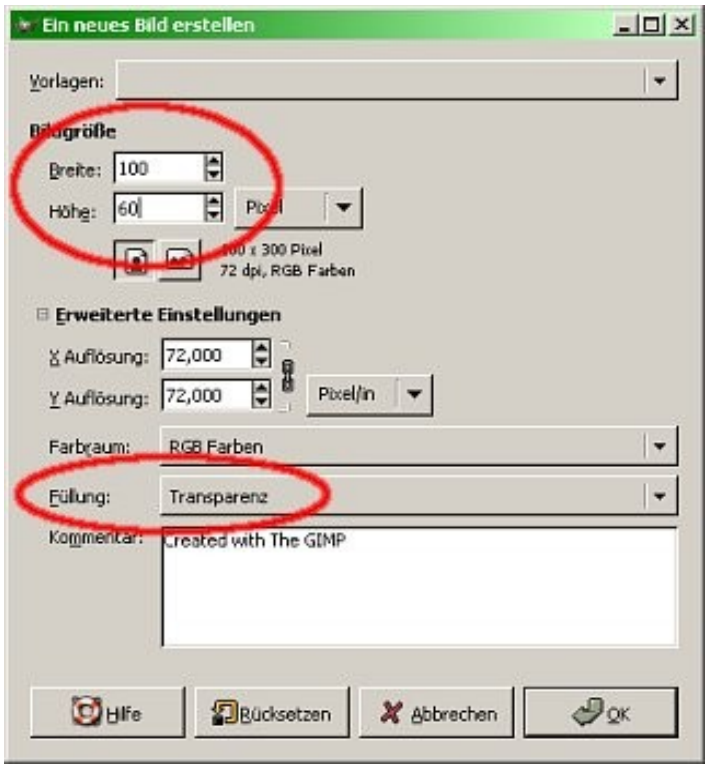
In diesem Sinne viel Erfolg mit euren Projekten

Tobias „LordRathan" Mogdans

Anhang

Hier sei noch ein Wort zur Erstellung transparenter TarGa-Grafiken mit TheGIMP gesagt. Für Neuanwender mag es ein wenig zu Verwirrungen kommen, es ist aber im Prinzip ganz einfach.

Mit der Tastenkombination Strg + N erstellt ihr ein neues Bild. Es erscheint ein Fenster, in dem ihr die Größe festlegt. Klickt auf „Erweiterte Einstellungen" und wählt als Füllung „Transparenz" aus. Ein Klick auf „OK" erstellt das neue Bild.



Euch erwartet jetzt ein transparentes Bild, erkennbar an dem hellgrau-dunkelgrauen Kachelmuster. Ihr könnt jetzt hier nach Belieben eure Grafik anfertigen, z. B. einen Schriftzug.

Mit Shift + Strg + S speichert ihr das Bild unter einem neuen Namen ab. Nennt es nach eigenem Ermessen, vergesst aber nicht die Endung „.tga" dahinter zu schreiben. Fertig ist euere 32 Bit TGA Grafik mit Alpha-Kanal.