

Home



c-programmieren.com

Allegro Tutorial

[This Website in bad Google-English, but in English.](#)

Inhaltsangabe

[Einleitung](#)

[Installation unter Dev-C++](#)

[Das erste Projekt](#)

["Hallo Welt" mit Allegro](#)

[Textausgabe](#)

[Eine andere Schriftart wählen](#)

[Die Alfont-Methode](#)

[load_font](#)

[Bilder zeichnen](#)

[Bilder bewegen](#)

[Geometrie - Kreise, Linien, Punkte...](#)

[Eingabeaufforderung](#)

[Maus](#)

[Windows.h + Allegro = Error](#)

[Der Allegro-Thread](#)

["Hallo Welt" im Fenster](#)

[Allegro Makro Liste](#)

[alleg42.dll](#)

[Bild-Kollision](#)

[Transparentes Zeichnen](#)

Einleitung:

(Wenn du dieses Allegro Tutorial drucken willst, [klicke hier.](#))

Allegro ist eine C(und C++)-Bibliothek, die Funktionen zur Spieleprogrammierung bereitstellt. Sie ist kostenlos, aber sehr umfangreich für jemand, der sich zuvor mit einer Dos-Konsole begnügt hat, ist sie phänomenal. Ich bin auf jeden Fall begeistert von der Vielfalt Möglichkeiten und wie einfach es geht mit Allegro unter C ein Spiel zu programmieren. Wenn dir Englisch liegt, kannst du auf die noch einmal alle Allegrofunktionen nachschlagen: <http://www.allegro.cc/manual/>

Die einzige Voraussetzung dieses Tutorials ist es, dass man bereits C programmieren kann. Wenn du Probleme mit C hast, kann einfach schnell in meinem anderen Tutorial [C-Lernen.html](#) nachschauen, oder mir bei größeren Problemen eine Email schicken. Viel Spaß mit Allegro!

Installation unter Dev-C++

Ich machs kurz mit Pfeilen:

Dev-C++ öffnen -> Werkzeuge -> Auf Updates/Pakete prüfen... -> Select devpak server: devpaks.org Community Devpaks -> Groups -> Check for updates -> herunterscrollen bis Allegro -> einen Hacken bei Allegro machen -> Download selected -> ein File "Package Manager" erscheint nach dem Download -> klicke den Next-Button -> nochmal -> Install-Button -> Finish -> Fertig!

Das erste Projekt

Wenn man ein neues Spiel programmieren will, ist die Projekterstellung bei Dev-C++ absolut identisch zur Erstellung eines ganz C-Konsolen-Programmes. (Dev-C++ -> Datei -> Neu -> Projekt -> Console Application -> C-Projekt ->...) Wenn du nicht weisst, wie geht: [C-Lernen.html#Dein erstes Programm](#)

Bei der Programmstruktur gibt es allerdings schon Neuheiten, wie du hier siehst:

```
#include <allegro.h>

int main(){

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    while(key[KEY_ESC]==0);

    return 0;
}
END_OF_MAIN();
```

Zunächst wirst du vergeblich nach den Bibliotheken stdio.h und stdlib.h suchen. Diese braucht man bei "normalen" Allegro Programmen meist nicht. Damit du ein Allegro-Source kompilieren kannst, musst du die liballeg.a einbinden (Projekt -> Projekt Optionen -> Parameter Bibliothek/Objekt hinzuf. -> C:\Dev-Cpp\lib\liballeg.a öffnen).

Die erste Allegro Funktion lautet allegro_init(). Mit ihr wird Allegro gestartet und initialisiert. Wie man am Namen der nächsten Funktion "install_keyboard" erkennt, installiert sie die Tastatur. Das heißt man kann mit Funktionen auf die Eingabe des Benutzers zugreifen. set_color_depth setzt die Farbtiefe des Bildschirms. Ich hab hier 16 gewählt. Bei der nächsten Funktion set_gfx_mode kannst du die Bildschirmauflösung wählen, die dir gefällt. Bei meinem Programm ist der Bildschirm 640 Pixel breit und 480 Pixel hoch. Die anderen Parameter sind fürs Erste unwichtig.

Im "key"-Array steht der Status aller Tasten, also, ob sie gedrückt werden oder nicht. Solange also das Element key[59] "0" bleibt, bleibt Escape nicht gedrückt. Die Konstante KEY_ESC hat nämlich den Wert 59. So muss man sich nicht die Zahl merken und kann den Namen eingeben. Alle key-Konstanten findest du hier: www.allegro.cc/manual/api/keyboard-routines/key. Zum Schluss wird mit dem Makro END_OF_MAIN alles beendet. Das Ergebnis des Programmes ist ein schwarzer Bildschirm. Wenn Escape drückst, wird das Programm beendet.

"Hallo Welt" mit Allegro

Für die Textausgabe gibt es einige Funktionen. Ich halte textprintf_ex für die beste Wahl. Deswegen hat diese Funktion die ehrenvolle Aufgabe "Hallo Welt!" auszugeben:

```
#include <allegro.h>

int main(){
    int x=0,y=0;

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 800, 600, 0, 0);

    while(key[KEY_ESC]==0)
    {
```

```

    textprintf_ex(screen, font, x, y, makecol(255, 255, 255), makecol(0, 0, 0), "Hallo Welt!");
}

return 0;
}
END_OF_MAIN();

```

Der erste Parameter von `textprintf_ex` ist immer `screen`. `screen` ist der Bildschirm und da wir auf diesen schreiben wollen, nehmen vordefinierte Makro. `font` ist die von Allegro vordefinierte Schriftart. Wie man eine andere Schriftart wählt, kommt noch. Die nächsten Parameter sind die Koordinaten, auf deren Platz der Schriftzug erscheinen soll. Mit der Funktion `makecol` kann man zunächst die Schriftfarbe wählen und beim nächsten Parameter die Hintergrundfarbe. `makecol` hat selbst drei Parameter. Sie stehen für rot, gr Mit diesen kann man alle beliebigen Farben ermischen. Hier einige Beispiele:

```

schwarz: 0, 0, 0
weis: 255, 255, 255
rot: 255, 0, 0
grün: 0, 255, 0
blau: 0, 0, 255
gelb: 255, 255, 0
orange: 255, 135, 0

```

Auf dieser Website kannst du wunderbare Farben mischen: www.farbenlehre.com/rgb-farben/rgb-bildschirm.htm
Der letzte Parameter funktioniert wieder genau wie bei `printf()`.

Textausgabe

Wie ich vorhin bereits erwähnt habe, gibt es viele Möglichkeiten etwas auf den Bildschirm zu schreiben. Hier sind sie:

```

#include <allegro.h>

int main(){
    char Wort[]={"textprintf"};
    int Zahl=1;
    char A='A';

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 800 , 600, 0, 0);

    while(key[KEY_ESC]==0)
    {
        textprintf_ex(screen, font, 0, 0, makecol(255, 255, 255), -1, "%s ist %d%c",Wort,Zahl,A);
        textprintf_centre_ex(screen, font, 800 / 2, 20, makecol(255, 255, 255), -1, "%s ist %d%c",Wort,Zahl
        textprintf_right_ex(screen, font, 800, 40, makecol(255, 255, 255), -1, "%s ist %d%c",Wort,Zahl,A);

        textout_ex(screen, font, "textout", 0, 60, makecol(255, 0, 0), -1);
        textout_centre_ex(screen, font, "textout", 800 / 2, 80, makecol(255, 0, 0), -1);
        textout_right_ex(screen, font, "textout", 800, 100, makecol(255, 0, 0), -1);
    }

    return 0;
}
END_OF_MAIN();

```

`textprintf` ist schon vollständig im Hallo Welt Kapitel beschrieben worden. Die einzige Neuheit ist, dass beim Parameter Hintergrund eine `-1` steht. Dies bewirkt, dass es keinen Hintergrund gibt. Die Abwandlung `_centre` der Funktion erzielt eine zentrierte Stellung r Ausgabe. Mit `_right` erreicht man eine rechtsbündige Ausgabe. Der Unterschied zwischen `textprintf` und `textout` ist, dass man mit alle verschiedenen Variablen ausgeben kann und mit `textout` nur Strings. Deswegen rate ich zur Benutzung von `textprintf`.

Eine andere Schriftart wählen

Um eine andere Schriftart zu wählen gibt es zwei Möglichkeiten. Die eine ist aufwendiger(Download und Installation einer Biblioth eleganten und die andere ist eher leicht(nur Download eines Programmes) anzuwenden, aber unschön und nicht variabel. Die elegante Methode ist die `Alfont`-Bibliothek. Mit ihr kann man direkt auf die Schriftarten zugreifen. Die zweite Möglichkeit ist die Funktion, die bei Allegro dabei ist. Mit ihr kann man aber nur `pcx`-Schriftdateien laden. Das heißt, man muss sich erst ein Programm downloaden dem man `ttf`-Dateien in `pcx` konvertieren kann. Zuerst ist die `Alfont`Methode an der Reihe:

Die Alfont-Methode

Für dieses Kapitel musst du erst die `alfont`-Bibliotheken downloaden. Das geht, wie folgt:

```

Dev-C++ öffnen -> Werkzeuge -> Auf Updates/Pakete prüfen... -> Select devpak server: devpaks.org Community Devpaks -> Gr
groups -> Check for updates -> herunterscrollen bis AllegroFont -> einen Hacken bei AllegroFont machen -> Download selectec

```

Fenster "Package Manager" erscheint nach dem Download -> klicke den Next-Button -> nochmal -> Install-Button -> Finish -> Fe
Jetzt ist alles dafür bereit, dass du alle Schriftarten auf deinem Computer benutzen kannst. Sie befinden sich im Verzeichnis
C:\Windows\Fonts\ . Im Beispiel habe ich Arial verwendet:

```
#include <allegro.h>
#include <alfont.h>

int main( void ) {
    allegro_init();
    install_keyboard();
    alfont_init();
    set_color_depth(32);
    set_gfx_mode(GFX_AUTODETECT, 640, 480, 0, 0);

    ALFONT_FONT *user_font;
    user_font = alfont_load_font("C:\\Windows\\Fonts\\arial.ttf");
    alfont_set_font_size(user_font, 100);

    alfont_textprintf(screen, user_font, 0, 0, makecol(255, 255, 255), "Test 1");
    readkey();

    alfont_destroy_font(user_font);
    alfont_exit();

    return 0;
}
END_OF_MAIN();
```

Damit dieses Programm läuft, musst du libalfont.a und wie immer liballeg.a in der richtigen Reihenfolge einbinden. Zuerst wird hier mit der Funktion alfont_init gestartet und am Schluss wieder mit alfont_exit beendet. Aber zuvor wird noch die Schriftart mit alfont in die Variable des Types ALFONT_FONT gespeichert. Die Größe der Schrift setzt man mit alfont_set_font_size. Um diese Schri benutzen muss man einfach ein "alfont_" vor die entsprechende Ausgabefunktion setzen. Z.B:

```
alfont_textprintf
alfont_textout
alfont_textprintf_centre
```

Nach der Benutzung der Schrift kann man deren Speicher mit alfont_destroy_font wieder freigeben.

load_font

Wie ich im vorherigen Kapitel schon erklärt habe, sind die Schriftarten in Form vieler ttf Dateien im Ordner C:\Windows\Fonts\. W
jetzt z.B. die arial.ttf im Format .pcx benötigt, muss man diese konvertieren. Das funktioniert mit dem Programm ttf2pcx, das auf c
Website zum Download angeboten wird: www.allegro.cc/resource/Tools/Fonts/ttf2pcx
Nachdem du die zip-Datei gedownloadet hast, entpackst du sie und öffnest das Programm ttf2pcx.exe. Jetzt kannst du auswähle
Schriftart, in welchem Stil(fett,normal,...) und in welcher Größe du deine Schriftart einsetzen willst. Ich wähle Arial, normal und 20
deine Auswahl getroffen hast, klicke auf Export und speichere die Datei in dem gleichen Verzeichnis, in dem du später dein Proje
und die exe sein wird. Jetzt kommt das Programm:

```
#include <allegro.h>

int main(){
    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    FONT *myfont;
    myfont = load_font("Arial.pcx", NULL, NULL);

    while( !key[KEY_ESC])
    {
        textprintf_centre_ex(screen, myfont, 320, 240, makecol(0, 0, 255),-1,"Arial");
    }
    destroy_font(myfont);
    return 0;
}
END_OF_MAIN();
```

Diesmal ist der Quelltext leichter zu verstehen. Mit load_font wird deine pcx Datei in die Variable myfont geladen. Danach kann m
normal den Text ausgeben. Am Schluss kann man wieder mit destroy_front den Speicher freigeben. Der Nachteil der pcx-Methoc
dass man erstens die pcx Datei mitliefern muss, wenn das Programm auf einem anderen Computer laufen soll und zweitens, das
Größe der Schriftart nicht variabel ist. Deswegen benütze ich die Alfont-Methode.

Bilder zeichnen

Um ein bmp-Bild auf den Bildschirm zeichnen zu können, muss man dieses zuerst laden. Dies geschieht mit der Funktion `load_bit`. Danach kann man das Bild wie in diesem Beispiel malen:

```
#include <allegro.h>

int main(){

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    BITMAP *Bild;
    Bild=load_bitmap("Bild.bmp", NULL);

    while( !key[KEY_ESC])
    {
        draw_sprite(screen, Bild, 0,0);
    }
    destroy_bitmap(Bild);

    return 0;
}
END_OF_MAIN();
```

Die erste Neuerung ist der neue Variablen-Typ `BITMAP`. In diesem werden die Daten des Bildes gespeichert, die mit `load_bitmap` werden. Mit `drawsprite` wird das Bild schließlich auf dem Bildschirm(`screen`) an den angegebenen Koordinaten angezeigt. `destroy` gibt am Schluss den Speicher für das Bild wieder frei.

Bilder bewegen

Im folgenden Programm soll das Bild erst von links nach rechts über den Bildschirm und dann wieder zurück "fliegen":

```
#include <allegro.h>

int main(){
    int Richtung=10; //10 Pixel in 0,1 Sekunden
    int x;

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    BITMAP *Bild;
    BITMAP *Hintergrund;

    Bild=load_bitmap("Bild.bmp", NULL);
    Hintergrund=create_bitmap(640, 480);

    while( !key[KEY_ESC])
    {
        draw_sprite(screen, Hintergrund, 0,0);
        draw_sprite(screen, Bild, x,0);
        x+=Richtung;
        if(x>(640-(Bild->w)) || x<0) Richtung*=-1;//Bild stößt an einen Bildschirmrand
        sleep(100);
    }
    destroy_bitmap(Bild);
    destroy_bitmap(Hintergrund);

    return 0;
}
END_OF_MAIN();
```

Mit diesem Programm wird es ein wenig schwieriger. Die erste neue Funktion in diesem Programm heißt `create_bitmap`. Die Variable `Hintergrund` beinhaltet nach der Zuweisung von `create_bitmap` ein schwarzes Bild, das 640 Pixel breit und 480 Pixel hoch ist. Die `Hintergrund` wird an die Koordinaten 0,0 gemalt. Der Parameter der x-Koordinate des "Bildes" ist diesmal die Variable `x`. Sie wird Schleifendurchlauf um den Wert `Richtung(10 || -10)` vergrößert/verkleinert. Wenn das Bild an den rechten Bildschirmrand stößt, `w` Richtung mit `-1` multipliziert. Das heißt der Wert der Variable ist jetzt `-10` und `x` wird jetzt in jedem Schleifendurchlauf um 10 kleiner. Wenn das Bild auf den linken Bildschirmrand trifft, wird der Wert wieder `+10`. So geht es immer hin und her. Mit `(Bild->w)` erhält man die Breite der Variable `Bild`.

Geometrie - Kreise, Linien, Punkte...

Da du jetzt weißt, wie man Bilder auf den Bildschirm zeichnet, malen wir jetzt selbst ein Bild:

```
#include <allegro.h>

int main(){
    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    putpixel(screen, 10, 30, makecol(255, 255, 255));//macht einen Punkt
    vline(screen, 30, 80, 300, makecol(0, 255, 0));//senkrechte Linie
    hline(screen, 40, 20, 400, makecol(0, 0, 255));//waagrechte Linie
    line(screen, 0, 0, 640, 480, makecol(255, 0, 0)); //Linie
    triangle(screen, 0, 400, 0, 450, 100, 450, makecol(255, 0, 255));//gefülltes Dreieck
    rect(screen, 100, 200, 200, 250, makecol(255, 255, 0));//leeres Viereck
    rectfill(screen, 125, 210, 175, 240, makecol(255, 255, 0));//gefülltes Viereck
    circle(screen, 500, 400, 50, makecol(255, 0, 0)); //leerer Kreis
    circlefill(screen, 500, 400, 25, makecol(255, 255, 255));//gefüllter Kreis

    readkey();
    return 0;
}
END_OF_MAIN();
```

Damit du verstehst, was die ganzen Zahlen bedeuten, hier die Definition der Funktionen:

```
void vline(BITMAP *bmp, int x, int y1, int y2, int Farbe);
void hline(BITMAP *bmp, int x1, int x2, int y, int Farbe);
void line(BITMAP *bmp, int x1, int y1, int x2, int y2, int Farbe);
void triangle(BITMAP *bmp, int x1, y1, x2, y2, x3, y3, int Farbe);
void rect(BITMAP *bmp, int x1, int y1, int x2, int y2, int Farbe);
void rectfill(BITMAP *bmp, int x1, int y1, int x2, int y2, int Farbe);
void circle(BITMAP *bmp, int x, int y, int Radius, int Farbe);
void circlefill(BITMAP *bmp, int x, int y, int Radius, int Farbe);
```

Das Problem sind die vielen x und ys. Sie beschreiben die Koordinaten bestimmter wichtiger Punkte. Z.B. bei der Funktion line ist für den Ort des ersten Punktes verantwortlich. Von diesem Punkt wird zu einem 2. Punkt eine Linie gezogen, der die Koordinaten besitzt. Um das mit den Koordinaten zu verstehen, musst du ein wenig mit diesen Funktionen spielen und ausprobieren. Das hat geholfen da durchzublicken.

Eingabeaufforderung

In den Allegro-Bibliotheken gibt es leider keine Funktion wie scanf. Deswegen muss man sich die Funktion selbst schreiben. Meir sieht so aus:

```
#include <allegro.h>

void Eingabe(FONT* myfont, char *Schreib1, int x, int y, int Schriftfarbe, int Hintergrund, char *Ausgabe)
{
    int val, i, anz;
    char Eingabestring[1024]={" "};
    char Schreib[1024];

    sprintf(Schreib,"%s",Schreib1);

    while( !key[KEY_ENTER])
    {
        textout_ex(screen, myfont, Schreib, x, y, Schriftfarbe, Hintergrund);
        val = readkey();
        if ((val >> 8) != KEY_ENTER && (val >> 8) != KEY_ESC)
        {
            anz=strlen(Eingabestring);
            if ((val >> 8) == KEY_BACKSPACE)
            {
                Eingabestring[anz-1]='\0';
                clear_bitmap(screen);
            }
            else if ((val >> 8) == KEY_TAB)
            {
                for(i=0;i<9;i++) Eingabestring[anz+i]=' ';
                Eingabestring[anz+i]='\0';
            }
            else
            {
                Eingabestring[anz]=(char)(val & 0xff);
            }
        }
    }
}
```

```

        Eingabestring[anz+1]='\0';
    }
}
textout_ex(screen, myfont, Eingabestring, x+text_length(myfont,Schreib), y, Schriftfarbe, Hintergrund)
}
sprintf(Ausgabe,"%s",Eingabestring);
}

int main(void){
    char Name[900];

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    Eingabe(font,"Bitte geben Sie Ihren Namen ein: ",0,0, makecol(0, 0, 255), -1, Name);

    clear_bitmap(screen);
    while( !key[KEY_ESC])
    {
        textprintf_ex(screen, font, 0, 0, makecol(0, 0, 255), -1,"Ihr Name lautet: ");
        textprintf_ex(screen, font, text_length(font),"Ihr Name lautet: ", 0, makecol(0, 0, 255), -1, Name)
    }

    return 0;
}
END_OF_MAIN();

```

Meine scanf Funktion heißt Eingabe. Sie hat einige Parameter. Die Parameter bestimmen zuerst die Schriftart, dann den Text der Eingabeaufforderung("Bitte geben Sie Ihren Namen ein: "), die Koordinaten der Eingabeaufforderung, die Schriftfarbe, die Hintergrundfarbe und der letzte Parameter ist der String, in den die Eingabe des Benutzers geschrieben wird. Mit der Funktion readkey wird ermittelt, wann eine Taste gedrückt wurde. Wenn man das Ergebnis in Form (intErgebnis & 0xff) einer char Variable übergibt, erhält man ein Zeichen, das durch einen Tastendruck entsteht. Es gibt aber auch Tasten bei denen kein Zeichen kommen soll, wie z.B bei KEY_BACKSPACE. Von dieser Taste ist man gewohnt, dass bei ihrer Betätigung der letzte Buchstabe gelöscht wird. Genau das in der if-Verzweigung. Das letzte Element von Eingabestring wird entfernt und der ganze Bildschirm wird mit clear_bitmap(screen) damit der letzte Buchstabe bei dem nächsten Schleifendurchlauf nicht mehr zu sehen ist. Weitere Ausnahmen sind die Leertaste Tabulatortaste. Bei der ersten Ausnahme wird einfach ein Leerzeichen hinzugefügt und bei der zweiten 9 Stück davon. Eingabe v call-by-reference an den letzten Parameter übergeben. Im Beispiel ist es der String Name. Falls du eine bessere Eingabefunktion programmiert hast, würde die mich sehr interessieren. Ich freu mich auf deine Email.

Maus

Eines der wichtigsten Themen bei der Spieleprogrammierung ist das Eingabegerät "Maus". Bei vielen Spielen muss das Programm wissen, ob die Maus gedrückt ist oder nicht. Diese Daten werden in dem folgenden Programm ermittelt:

```

#include <allegro.h>

int main(){
    char Press[3][10];
    strcpy(Press[0],"NEIN");//if(mouse_b == 0) kein Klick
    strcpy(Press[1],"JA"); //if((mouse_b & 1)==1) Linksklick
    strcpy(Press[2],"JA"); //if((mouse_b & 2)==2) Rechtsklick

    allegro_init();
    install_keyboard();
    install_mouse();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);

    BITMAP* buffer;
    BITMAP* Cursorbild;
    buffer = create_bitmap( 640, 480);
    Cursorbild = load_bitmap("maus.bmp",NULL);

    while( !key[KEY_ESC])
    {
        draw_sprite(screen, buffer, 0, 0);
        draw_sprite(screen, Cursorbild, mouse_x, mouse_y);
        textprintf_ex(screen,font,0,10,makecol(0,0,255),-1,"Linke Maustaste gedruickt: %s",Press[mouse_b&1]);
        textprintf_ex(screen,font,0,30,makecol(0,0,255),-1,"Rechte Maustaste gedruickt: %s",Press[mouse_b&2]);
        textprintf_ex(screen,font,0,50,makecol(0,0,255),-1,"Maus-Koordinaten: x=%d, y=%d",mouse_x,mouse_y);
        sleep(100);
    }
    destroy_bitmap(buffer);

    return 0;
}
END_OF_MAIN();

```

Die Maus muss wie die Tastatur installiert werden. Das geschieht mit der Funktion `install_mouse`. Ich male ein Bild an die Stelle, Maus liegt. Die Koordinaten der Maus sind in den Variablen `mouse_x` und `mouse_y`, die von Allegro selbstständig aktualisiert werden. Wenn der Wert von `(mouse_b&1) == 1` ist, dann wird die linke Taste der Maus gedrückt. Wenn `(mouse_b&2) == 2`, dann wird die Taste gedrückt. Ansonsten sind die Werte 0.

Windows.h + Allegro = Error

Wenn man diesen Sourcecode ausführen will, bekommt man ein Problem:

```
#include <allegro.h>
#include <windows.h>

int main()
{
    return 0;
}
```

Das Problem hat mit den 2 verschiedenen Definitionen der Variable `BITMAP` zu tun. Um dieses Problem zu umgehen, kann man vorgehen:

```
#include <allegro.h>
#include <winalleg.h>

int main()
{
    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 640, 480, 0, 0);
    PlaySound( "Lied.wav", NULL, SND_FILENAME);
    readkey();
    return 0;
}
END_OF_MAIN();
```

Bei diesem Programm braucht man wie immer die `liballeg.a` und `libwinmm.a`. Wie du siehst ist `winalleg.h` die Lösung. Sie entspricht `windows.h` und macht keine Probleme mit Allegro. Schon ist das Problem gelöst.

Der Allegro-Thread

Nachdem du weißt, dass man WinAPI eigentlich auch einsetzen kann, ist dir hoffentlich bekannt, dass es dann möglich ist ein Thread zu starten. Mit `windows.h` geht das mit der Funktion `CreateThread`. Wenn du nicht mehr weißt, was ein Thread ist, kannst du es hier nachlesen: [C-Lernen.html#Der Thread](#)

In Allegro gibt es auch eine eigene Funktion dafür, die hier erwähnt wird:

```
#include <allegro.h>

int o=0;

void Thread(void* Zahl)
{
    int Meine_Zahl = (int)Zahl;
    while(o==0)
    {
        textprintf_ex(screen,font,0,10,makecol(0,0,255),-1,"Die schoene Zahl: %d",Meine_Zahl);
        Meine_Zahl++;
        sleep(100);
        clear_bitmap(screen);
    }
    _endthread();
}

int main(){
    int Anfang=0;

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 800, 600, 0, 0);

    _beginthread(Thread, 0, Anfang);

    readkey();//Warten, bis Tastendruck
    o=1;
    allegro_exit();
}
```

```

    return 0;
}
END_OF_MAIN();

```

Um den Thread diesmal zu starten, verwendet man `_beginthread()`. Der erste Parameter ist die Funktion, die als Thread benutzt soll. Der zweite Parameter ist unwichtig und der dritte ist der Wert, der an die Funktion übergeben werden soll. Während das Hauptprogramm auf eine Eingabe wartet, addiert der Thread die Zahl mit eins und gibt sie aus. Mit der Funktion `_endthread`, kann Thread beenden. Diesmal habe ich Allegro auch mit `allegro_exit()` beendet, damit es hier nicht zu einem Fehler kommt.

"Hallo Welt" im Fenster

Bisher waren alle Programme, die du programmiert hast im Vollbildmodus. Jetzt kommt das Hallo Welt Programm nochmal in ein Fenster zum Vorschein:

```

#include <allegro.h>

int main(){
    int x=0,y=0;

    allegro_init();
    install_keyboard();
    install_mouse(); //Installation der Maus
    set_color_depth(16);
    set_gfx_mode(GFX_AUTODETECT_WINDOWED, 640, 480, 0, 0);
    set_window_title("Das ist der Hallo-Welt-Titel"); //Titel des Fensters wird benannt
    show_os_cursor(MOUSE_CURSOR_ARROW); //die Maus wird angezeigt

    textprintf_ex(screen, font, x, y, makecol(255, 255, 255), makecol(0, 0, 0), "Hallo Welt!");
    readkey();

    return 0;
}
END_OF_MAIN();

```

Der Grund, warum der Compiler jetzt ein Fenster ausspuckt, ist der Parameter `GFX_AUTODETECT_WINDOWED`. Mit diesem wird ein Fenster angefordert. Den Titel des Fensters setzt man mit der Funktion `set_window_title`. Wenn man die Maus zu Gesicht bekommt, kann man sich diese mit der Funktion `show_os_cursor` und dem entsprechenden Parameter anzeigen lassen. Der Rest funktioniert wie beim Vollbild-Modus.

Allegro Makro Liste

Du hast bereits einige Allegro-Makros kennen gelernt, hier eine Liste aller wichtigen Makros und Variablen:

Maus:

```

mouse_x //x-Koordinate der Maus
mouse_y //y-Koordinate der Maus
mouse_z //Drehung des Mousrades
mouse_b //Status der Tasten

```

Bildschirmdaten:

```

screen //Bild des Bildschirms
SCREEN_W //Breite des Bildschirms
SCREEN_H //Höhe des Bildschirms

```

Schrift:

```

font //eine Standard-Schriftart

```

alleg42.dll

Wenn du eines deiner Allegroprogramme auf einem anderen Computer startest, kommt die Fehlermeldung: "Die Anwendung konnte nicht gestartet werden, weil alleg42.dll nicht gefunden wurde. Neuinstallation der Anwendung könnte das Problem beheben.". Wie sich die Fehlermeldung angefordert, benötigt das Programm die `alleg42.dll`. Diese befindet sich im Ordner `C:\Dev-Cpp\bin`. Sie muss in das Verzeichnis des Programmes hineinkopiert werden, dann läuft das Programm wie gewohnt.

Bild-Kollision

Dieses Thema wird dir in der Spieleprogrammierung immer wieder begegnen. Z.B. ist es bei Shootern wichtig zu wissen, ob das der "Kugel" getroffen wurde oder nicht. Beim nächsten Beispiel ist aber nur ein Bild auf Kollisionskurs mit einem anderen Bild. Bei Kollision wird eine Warnung ausgegeben. Jetzt aber das Programm:

```

#include <allegro.h>

int BildKollision(BITMAP* Bild1,int x1, int y1, BITMAP* Bild2,int x2, int y2)
{
    if(x1>=(x2+Bild2->w) || y1>=(y2+Bild2->h) || x2>=(x1+Bild1->w) || y2>=(y1+Bild1->h)) return 0;
    else return -1;
}

int main(){
    int x1=0 , y1=0;//Koordinaten des ersten Bildes
    int x2=500, y2=0;//Koordinaten des zweiten Bildes

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode( GFX_AUTODETECT, 800, 600, 0, 0);

    BITMAP* Bild = load_bitmap("Bild.bmp",NULL);
    BITMAP* Hintergrund = create_bitmap(800,600);

    while(!keypressed())//solange keine Taste gedrückt wird
    {
        draw_sprite(screen,Hintergrund,0,0);
        draw_sprite(screen,Bild,x1,y1);
        draw_sprite(screen,Bild,x2,y2);
        x1+=10;//das erste Bild bewegt sich in Richtung des zweiten Bildes
        if(BildKollision(Bild,x1,y1,Bild,x2,y2)) textprintf_ex(screen,font,350,280,makecol(255,0,0),-1,"Koll
        sleep(100);
    }
    destroy_bitmap(Bild);
    destroy_bitmap(Hintergrund);

    return 0;
}
END_OF_MAIN();

```

Wie du siehst, habe ich eine eigene Funktion geschrieben, die überprüft, ob die als Parameter angegebenen Bilder kollidieren. Eine Neuerung ist noch die Funktion `keypressed()`. Solange Sie den Wert -1 zurückgibt, wurde keine Taste gedrückt.

Transparentes Zeichnen

Warum laufen eigentlich in all diesen Spiele keine Figuren in einem rechteckigen Kasten herum? Die Antwort auf diese Frage ist zumindest trifft das für Allegro zu. Diejenigen Flächen auf den Bildern, die die Farbe RGB(255,0,255) besitzen, werden von `draw` transparent angezeigt. Das heißt man sieht das Bild, das unter dem Bild liegt. Dazu ein Beispiel:

```

#include <allegro.h>

int main(){

    allegro_init();
    install_keyboard();
    set_color_depth(16);
    set_gfx_mode(GFX_AUTODETECT, 640, 480, 0, 0);

    BITMAP* Hintergrund=load_bitmap("Wiese.bmp", NULL);
    BITMAP* Figur=load_bitmap("Figur.bmp", NULL);

    draw_sprite(screen, Hintergrund, 0,0);
    blit(Figur, screen, 0, 0, 0, 50, Figur->w, Figur->h);//Bild normal zeichnen
    draw_sprite(screen, Figur,Figur->w+50, 50);//Bild ohne rosa zeichnen

    readkey();

    destroy_bitmap(Figur);
    destroy_bitmap(Hintergrund);

    return 0;
}
END_OF_MAIN();

```

Damit du das selbe Ergebnis, wie ich auf dem Bildschirm hast, kannst du hier meine Bilder downloaden:

[Wiese.bmp](#)

[Figur.bmp](#)

Wenn du dich jetzt fragst, wie man ein Bild, wie `Figur.bmp` erstellt, ist hier die Antwort:

Paint öffnen -> das zeichnen, was gesehen werden soll -> im Menü auf: "Farben" -> Palette bearbeiten -> Benutzerdefinierte Farl

Kästchen auswählen-> Farben definieren -> Parameter: Farbt.: 0, Sätt.: 0, Hell.: 0, Rot: 255, Grün: 0, Blau: 255 -> Farben hinzufügen
OK -> den Hintergrund mit dieser Farbe füllen -> Speichern unter -> Dateityp: 16-Farben-Bitmap (*.bmp;*.dib) -> Name eingeben -
speichern -> Fertig!

Website template provided by [Photoshop Tutorials](#)

Design downloaded from [Free Templates](#) - your source for free web templates