Das Space-Alien Spiel

Erik Bartmanns

Amiga 1200 Buch

Maschinensprache, C-Programmierung und Shell

bombini verlag

Ein einfaches Ballerspiel

Folgende Themen werden besprochen.

- Die Programmierung von Space-Alien
- Die Erläuterung von Sprites
- Wie erfolgt eine Kollisionsermittlung zwischen Sprites

Space-Alien

In meinem Amiga 1200-Buch habe ich schon im letzten Kapitel der C-Programmierung ein einfaches Spiel vorgestellt, wo es darum geht, ein Herz mit einem Projektil zu treffen. In diesem Zusatzkapitel möchte ich das etwas erweitern und hoffentlich spannender gestalten.

Abbildung 1 - Das Space-Alien Spiel

Du hast es also mit einem Alien-Raumschiff zu tun, das sich von links nach rechts und umgekehrt bewegt und bei jeder Randberührung etwas nach unten wandert. Gelangt es ganz nach unten, wird das Spiel beendet. Du musst nun versuchen, mit deiner Kanone das Schiff zu treffen, was bei einem Treffer dazu führt, dass dein Punktestand erhöht wird. Um es jedoch noch etwas schwieriger zu gestalten, wird das Raumschiff bei jedem Treffer etwas schneller, was einen Treffer etwas schwieriger gestaltet. Für ein derartiges Spiel, was ja schon recht einfach gehalten ist, werden jedoch ca. 200 Codezeilen benötigt, die ich aber alle mit dir besprechen werde. Ich habe zur Programmierung den Aztec-C Compiler 5 genutzt. Eine Anleitung zur Installation ist auf meiner Internetseite zu finden.

Steigen wir ein, in die C-Programmierung für das Spiel.

Die Programmierung von Space-Alien

Wie jeder weiß, der sich schon einmal mit der C-Programmierung befasst hat, benötigt es zu Beginn die Einbindung erforderlicher Bibliotheken über die Include-Anweisungen.

```
1  /*  Scope:  Space-Alien
2      Autor:  Erik Bartmann
3      Datum:  02.03.2016
4      URL:    https://erik-bartmann.de/   */
5
6  #include <exec/types.h>
7  #include <exec/memory.h>
8  #include <intuition/intuition.h>
9  #include <graphics/gfx.h>
10 #include <graphics/sprite.h>
```

Abbildung 2 - Der Quellcode für Space-Alien - Teil 1

Über die Zeile

```
#include <exec/types.h>
```

werden grundlegende Datentypen des AmigaOS (UWORD, ULONG, BOOL usw.) genutzt. Mit der Zeile

```
#include <exec/memory.h>
```

können wir später Funktionen für Speicherverwaltung (AllocMem, FreeMem) verwenden. Die Zeile

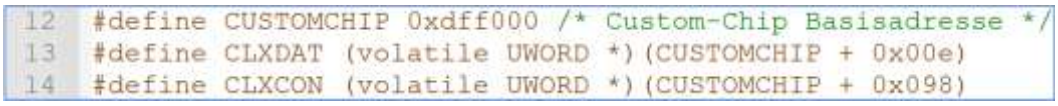
```
#include <intuition/intuition.h>
```

wird für das Fenster- und Event-System (Intuition ist das GUI-System des Amigas) benötigt. Mit den Zeilen

```
#include <graphics/gfx.h>
```

```
#include <graphics/sprite.h>
```

werden die Grafikfunktionen und Sprite-Steuerung bereitgestellt. Gehen wir über zum 2. Teil.



```
12 #define CUSTOMCHIP 0xdff000 /* Custom-Chip Basisadresse */
13 #define CLXDAT (volatile UWORD *)(CUSTOMCHIP + 0x00e)
14 #define CLXCON (volatile UWORD *)(CUSTOMCHIP + 0x098)
```

Abbildung 3 - Der Quellcode für Space-Alien - Teil 2

Da wir später auf Custom-Chip-Adressen zugreifen wollen, müssen diese über die folgenden Zeilen definiert werden. Die Basisadresse der Amiga-Custom-Chips (Agnus, Denise etc.) ist dabei *$dff000*.

```
#define CUSTOMCHIP 0xdff000 /* Custom-Chip Basisadresse */
```

Das Kollisionsdaten-Register (Denise Chip) wird über die folgende Zeile definiert.

```
#define CLXDAT (volatile UWORD *)(CUSTOMCHIP + 0x00e)
```

Das Kollisionskontroll-Register wird mittels dieser Zeile definiert.

```
#define CLXCON (volatile UWORD *)(CUSTOMCHIP + 0x098)
```

Was bedeutet der Zusatz volatile?
Mit dem Zusatz *volatile* sagst du dem Compiler: „Fass diese Variable nicht an!“ Normalerweise versucht der Compiler, Code schneller zu machen, indem er Werte in superschnellen Registern zwischenspeichert. Wenn sich ein Wert aber „hinter dem Rücken“ des Programms ändert – zum Beispiel durch die Hardware oder einen Alarm – würde der Compiler das nicht merken. Über volatile wird sichergestellt, dass das Programm immer den echten, aktuellen Wert direkt aus dem Hauptspeicher liest.

Gehen wir über zum 3. Teil. Hier werden die Funktionsprototypen definiert.

```
16  /* Prototypen für Aztec C / */
17  extern struct Library *OpenLibrary();
18  struct Window *OpenWindow();
19  struct IntuitionBase *IntuitionBase;
20  struct GfxBase *GfxBase;
```

Abbildung 4 - Der Quellcode für Space-Alien - Teil 3

Mit der Zeile

```
extern struct Library *OpenLibrary();
```

werden die Systembibliotheken geöffnet.

Über

```
struct Window *OpenWindow();
```

wird eine Fenster geöffnet. Mittels

```
struct IntuitionBase *IntuitionBase;
```

```
struct GfxBase *GfxBase;
```

werden globale Zeiger auf Systembibliotheken erzeugt. Ich fahre fort mit den Definitionen der drei Sprites, die für das Spiel benötigt werden. Im 4. Teil wird die Kanone beschrieben.

```
22  /* Sprite-Daten (Chip-RAM) */
23  /* Sprite 1: Kanone        */
24  UWORD rawData1[] = {
25      0x0000, 0x0000, //  ................
26      0x0180, 0x0180, //  .......XX.......
27      0x0180, 0x0180, //  .......XX.......
28      0x0180, 0x0180, //  .......XX.......
29      0x0FF0, 0x0FF0, //  ....XXXXXXXX....
30      0x0FF0, 0x0FF0, //  ....XXXXXXXX....
31      0x0FF0, 0x0FF0, //  ....XXXXXXXX....
32      0x0FF0, 0x0FF0, //  ....XXXXXXXX....
33      0x0FF0, 0x0FF0, //  ....XXXXXXXX....
34      0x0000, 0x0000  //  ................
35  };
```

Abbildung 5 - Der Quellcode für Space-Alien - Teil 4

Im 5. Teil das Alien-Raumschiff.

```
/* Sprite 2: Space Invader */
UWORD rawData2[] = {
    0x0000, 0x0000, // ................
    0x1100, 0x1100, // ...X...X........
    0x0A00, 0x0A00, // ....X.X.........
    0x3F80, 0x3F80, // ..XXXXXX........
    0x66C0, 0x66C0, // .XX..XX.XX......
    0x7FC0, 0x7FC0, // .XXXXXXXXX......
    0x1500, 0x1500, // ...X.X.X........
    0x2080, 0x2080, // ..X.....X.......
    0x4040, 0x4040, // .X.......X......
    0x0000, 0x0000  // ................
};
```

Abbildung 6 - Der Quellcode für Space-Alien - Teil 5

Und letztendlich im 6. Teil kommt es zur Definition des Projektils.

```
/* Sprite 3: Projektil */
UWORD rawData3[] = {
    0x0000, 0x0000, // ................
    0x0180, 0x0180, // .......XX.......
    0x03C0, 0x03C0, // ......XXXX......
    0x03C0, 0x03C0, // ......XXXX......
    0x0180, 0x0180, // .......XX.......
    0x0000, 0x0000  // ................
};
```

Abbildung 7 - Der Quellcode für Space-Alien - Teil 6

Da ich für das Spiel eigene Farben nutze, werden diese über die folgenden Zeilen definiert. Ich teile das in zwei separate Blöcke, weil es ansonsten zu lang ist für einen Screenshot.

```
UWORD Colors[32] = {
    /* 0-3: Hintergrund & UI (Tiefes Blau/Schwarz bis Weiss) */
    0x001, /* 0: Sehr dunkles Nachtblau (Hintergrund)         */
    0x0ff, /* 1: Cyan (fuer Texte/Score)                      */
    0xf0f, /* 2: Magenta (Akzente)                            */
    0xaaa, /* 3: Grau (Sterne/Rahmen)                         */

    /* 4-15: Weitere Playfield-Farben (optional) */
    0x224, 0x446, 0x05a, 0x08d, 0x789, 0xa2e,
    0x5ad, 0x8ed, 0xdb9, 0xa56, 0x789, 0xabc,

    /* --- SPRITE FARBEN (Ab Register 16) --- */
    /* 16-19: Sprite Paar 0 & 1 (Deine Kanone)             */
    0x000, /* 16: (Transparent)                            */
    0x000, /* 17: ---                                      */
    0x000, /* 18: ---                                      */
    0xff0, /* 19: Gelb (Deine Kanone)                      */
```

Abbildung 8 - Der Quellcode für Space-Alien - Teil 7a

Und hier der zweite Block.

```
77      /* 20-23: Sprite Paar 2 & 3 (Das Herz / Invader)        */
78      0x000, /* 20: (Transparent)                             */
79      0x000, /* 21: ---                                       */
80      0x000, /* 22: ---                                       */
81      0xF88, /* 23: Rosa/Lachs (Dein Herz)                    */
82
83      /* 24-27: Sprite Paar 4 & 5 (Das Projektil)             */
84      0x000, /* 24: (Transparent)                             */
85      0xccc, /* 25: ---                                       */
86      0x8e0, /* 26: ---                                       */
87      0xf00, /* 27: Rot (Dein Projektil)                      */
88
89      /* 28-31: Sprite Paar 6 & 7 (...)                       */
90      0xfac, /* 28: (---)                                     */
91      0x93f, /* 29: (---)                                     */
92      0x06d, /* 30: (---)                                     */
93      0x6fe  /* 31: (---)                                     */
94  };
```

Abbildung 9 - Der Quellcode für Space-Alien - Teil 7b

Sehen wir weiter mit dem 8. Teil, der ein Array definiert, um die durch das Spiel veränderten Farben vorher sichert. Nach der Beendigung des Spiels können diese Werte gelesen und wieder hergestellt werden.

```
96  UWORD OldColors[32]; /* Array fuer vorherige Farbwerte */
```

Abbildung 10 - Der Quellcode für Space-Alien - Teil 8

Der Amiga besitzt 32 Farbregister. Das Format ist: 0xRGB (je 4 Bit pro Farbe). Im nächsten Teil 9 wird das Fenster definiert.

```
98  /* Das Fenster definieren */
99  struct NewWindow nw = {
100     20, 20, 320, 160, -1, -1,
101     RAWKEY | CLOSEWINDOW,
102     WINDOWCLOSE | WINDOWDRAG | ACTIVATE | SMART_REFRESH,
103     NULL, NULL, (UBYTE *)"Space Alien", NULL, NULL,
104     50, 50, 640, 400, WBENCHSCREEN
105 };
```

Abbildung 11 - Der Quellcode für Space-Alien - Teil 9

Über 20, 20 wird die Fensterposition festgelegt. Mit den Werten 320, 160 ist die Breite und Höhe definiert. Die Werte

`RAWKEY | CLOSEWINDOW`

legen fest, welche Events empfangen werden. *RAWKEY* ist für die Tasten zuständig und *CLOSEWINDOW* für das Schließen des Programms. Die Werte

`WINDOWCLOSE | WINDOWDRAG | ACTIVATE | SMART_REFRESH`

sind für das Fensterverhalten zuständig. Der Fenstertitel wird mit der Zeile

`(UBYTE *)"Space Alien"`

festgelegt und mit

WBENCHSCREEN

wird der Workbench-Screen geöffnet. Sehen wir weiter mit dem 10. Teil. Es handelt sich um eine Funktionsdefinition, die später zur Anzeige des Scores genutzt wird.

```
107 /* Score zum Zeichnen im Fenster */
108 void drawScoreInWindow(struct Window *win, int score, BOOL isGameOver) {
109     char buffer[64];
110     struct RastPort *rp = win->RPort;
111     if (isGameOver)
112         sprintf(buffer, " GAME OVER! Punkte: %d ", score);
113     else
114         sprintf(buffer, "Punkte: %d  ", score); // Leerzeichen am Ende löschen Reste
115     /* 1. Hintergrund-Rechteck zeichnen, um alten Text zu löschen */
116     SetAPen(rp, 0); // Farbe 0
117     /* Zeichne einen Kasten oben links im Fenster */
118     RectFill(rp, win->BorderLeft + 10, win->BorderTop + 5,
119                  win->BorderLeft + 200, win->BorderTop + 15);
120     /* 2. Textfarbe setzen und Text schreiben */
121     SetAPen(rp, 1); // Farbe 1
122     Move(rp, win->BorderLeft + 10, win->BorderTop + 13); // Position im Fenster
123     Text(rp, (UBYTE *)buffer, strlen(buffer));
124 }
```

Abbildung 12 - Der Quellcode für Space-Alien - Teil 10

Über die Zeile

```
sprintf(buffer, "Punkte: %d", score);
```

wird ein anzuzeigender Text erzeugt. Über die Zeilen

```
SetAPen(rp, 0);
```

```
RectFill(...)
```

wird alter Text gelöscht und über

```
SetAPen(rp, 1);
```

```
Move(...)
```

```
Text(...)
```

der neue Text geschrieben. Sehen wir weiter mit dem 11. Teil, in dem die aktuellen Farben vor der Änderung über eine Funktion gesichert werden.

```
126 /* Die aktuellen Farben sichern */
127 void saveColors(struct Window *w){
128     int i;
129     for(i = 0; i < 32; i++){
130         OldColors[i] =
131             GetRGB4(w->WScreen->ViewPort.ColorMap, i);
132     }
133 }
```

Abbildung 13 - Der Quellcode für Space-Alien - Teil 11

Die Zeile

```
GetRGB4(w->WScreen->ViewPort.ColorMap, i);
```

liest die aktuellen Bildschirmfarben und speichert sie in das Array. Starten wir im 12. Teil mit dem Beginn der *main*-Funktion, die sich aufgrund ihrer Länge über mehrere Teile erstreckt.

```
135 /* Die Haupt-Funktion */
136 int main(void) {
137     struct Window *win;
138     struct SimpleSprite sPlayer, sTarget, sProj;
139     struct IntuiMessage *msg;
140     UWORD *cData1, *cData2, *cData3;
141     SHORT sn1, sn2, snP;
142     UWORD collision;
143     BOOL hitRegistered = FALSE;
144
145     /* Spielzustand */
146     WORD pX = 140, pY = 130;
147     /* tDir = Invader-Geschwindigkeit */
148     WORD tX = 60, tY = 30, tDir = 2;
149     WORD prX = 0, prY = 0;
150     WORD score = 0;
151     BOOL prActive = FALSE, running = TRUE;
152     BOOL keyLeft = FALSE, keyRight = FALSE;
```

Abbildung 14 - Der Quellcode für Space-Alien - Teil 12

Ich konzentriere mich im Moment auf die unteren Zeilen. Alles andere wird später erläutert. Die Zeilen

```
WORD pX = 140, pY = 130;
```

definieren die Spielerposition und

```
WORD tX = 60, tY = 30
```

die Gegnerposition. Über

```
tDir = 2;
```

wird die Bewegungsrichtung mit der angegebenen Geschwindigkeit festgelegt. Dieser Wert kann positiv und negativ werden. Über

```
prActive
```

wird ermittelt, ob das Projektil aktiv, also unterwegs ist. Mit

```
score
```

wird der Punktestand verwaltet. Im nächsten Teil 13, werden die Libraries geöffnet.

```
154     /* Libraries öffnen */
155     if (!(IntuitionBase = (struct IntuitionBase *)
156         OpenLibrary("intuition.library", 37)))
157         return 20;
158     if (!(GfxBase = (struct GfxBase *)
159         OpenLibrary("graphics.library", 37))) {
160         CloseLibrary((struct Library *)IntuitionBase);
161             return 20;
162     }
163     if (!(win = (struct Window *)OpenWindow(&nw))) {
164         CloseLibrary((struct Library *)GfxBase);
165         CloseLibrary((struct Library *)IntuitionBase);
166             return 20;
167     }
```

Abbildung 15 - Der Quellcode für Space-Alien - Teil 13

Die angegebene Version 37 bedeutet mindestens ein AmigaOS 2.0 und falls das nicht vorhanden ist, kommt es zu einem Programmende. Für ältere Amiga-Computer, wie zum Beispiel den Amiga 500 kannst du diese Werte auf 0 setzen. Über die Zeile

```
OpenWindow(&nw)
```

wird letztendlich das Fenster geöffnet. Sehen wir weiter mit Teil 14, in dem der Aufruf der Funktion zur Sicherung der aktuellen Farben erfolgt.

```
169     /* Die aktuellen Farben sichern */
170     saveColors(win);
```

Abbildung 16 - Der Quellcode für Space-Alien - Teil 14

Nachdem die aktuellen Farben gesichert wurden, können die neuen Farben für das Spiel geladen werden. Dazu wird die LoadRGB4-Funktion mit der Angabe des Arrays *Colors* aufgerufen.

```
172     /* win->WScreen ist der Pointer auf den Screen */
173     LoadRGB4(&win->WScreen->ViewPort, Colors, 32);
```

Abbildung 17 - Der Quellcode für Space-Alien - Teil 15

Jetzt geht es im 16. Teil um die Konfiguration der *Collision* über das *CLXCON*-Register und der Aktivierung der Kollision.

```
174     /* Initialisierung von Collision                */
175     *CLXCON = 0x0000;
```

Abbildung 18 - Der Quellcode für Space-Alien - Teil 16

Kommen wir zum nächsten Teil und wichtigen Aspekt, dass die Sprite-Daten immer im Chip-RAM liegen müssen, um einen DMA-Zugriff zu ermöglichen. Im 17. Teil geht es um die Reservierung des Chip-RAMs.

```
177     /* Speicher im Chip-RAM reservieren */
178     cData1 = (UWORD *)AllocMem((LONG)sizeof(rawData1), MEMF_CHIP | MEMF_CLEAR);
179     cData2 = (UWORD *)AllocMem((LONG)sizeof(rawData2), MEMF_CHIP | MEMF_CLEAR);
180     cData3 = (UWORD *)AllocMem((LONG)sizeof(rawData3), MEMF_CHIP | MEMF_CLEAR);
```

Abbildung 19 - Der Quellcode für Space-Alien - Teil 17

Die *AllocMem*-Funktion sorgt für diese Reservierung. AllocMem steht kurz für *Allocate Memory* (Speicherreservierung) und beschreibt einen Vorgang, bei dem ein Programm das Betriebssystem oder die Laufzeitumgebung um einen bestimmten Bereich im Arbeitsspeicher (RAM) bittet. Im 18. Teil werden dann die zu Beginn Sprite-Daten (rawData1, rawData2 und rawData3) in die zuvor angeforderten Chip-RAM-Bereiche kopiert.

```
182     if (cData1 && cData2 && cData3) {
183         CopyMem(rawData1, cData1, (LONG)sizeof(rawData1));
184         CopyMem(rawData2, cData2, (LONG)sizeof(rawData2));
185         CopyMem(rawData3, cData3, (LONG)sizeof(rawData3));
```

Abbildung 20 - Der Quellcode für Space-Alien - Teil 18

Die Funktion *CopyMem* (kurz für Copy Memory) ist das logische Gegenstück zu *AllocMem*. Während AllocMem Platz schafft, füllt CopyMem diesen Platz, indem es Daten von einer Stelle im Speicher an eine andere kopiert. Sehen wir weiter mit dem 19. Teil des Quellcodes.

```
187         if ((sn1 = GetSprite(&sPlayer, 1)) >= 0 &&
188             (sn2 = GetSprite(&sTarget, 2)) >= 0 &&
189             (snP = GetSprite(&sProj, 4))   >= 0) {
```

Abbildung 21 - Der Quellcode für Space-Alien - Teil 19

Über die *GetSprite*-Funktion wird das entsprechende Hardware-Sprite angefordert. Die angegebenen IDs spielen später noch eine entscheidende Rolle. Sehen wir weiter mit dem 20. Teil, in dem es um die Konfiguration der Sprites geht.

```
191             sPlayer.num = sn1; sPlayer.height = 8;
192             sTarget.num = sn2; sTarget.height = 8;
193             sProj.num   = snP;   sProj.height = 4;
```

Abbildung 22 - Der Quellcode für Space-Alien - Teil 20

Über die *height*-Eigenschaft wird festgelegt, wie viele Pixel das jeweilige Sprite hoch ist. Im 21. Teil wird über die *ChangeSprite*-Funktion das aktuelle Bild eines Grafikobjekts gegen ein neues ausgetauscht, ohne das Objekt selbst (seine Position oder seine Logik) zu löschen.

```
195             ChangeSprite(&win->WScreen->ViewPort, &sPlayer, cData1);
196             ChangeSprite(&win->WScreen->ViewPort, &sTarget, cData2);
197             ChangeSprite(&win->WScreen->ViewPort, &sProj, cData3);
```

Abbildung 23 - Der Quellcode für Space-Alien - Teil 21

Sehen wir weiter mit dem 22. Teil mit dem Aufruf der *SetRGB4*-Funktion und dem Setzen der Farbregister.

```
199             SetRGB4(&win->WScreen->ViewPort, 17, 15, 15, 15);
200             SetRGB4(&win->WScreen->ViewPort, 21, 15,  0,  0);
201             SetRGB4(&win->WScreen->ViewPort, 25,  0, 15,  0);
```

Abbildung 24 - Der Quellcode für Space-Alien - Teil 22

Die *SetRGB4*-Funktion ist ein klassischer Befehl aus der Grafikprogrammierung, den man vor allem von Systemen wie dem Amiga (OCS/ECS) oder aus der hardwarenahen C-Programmierung kennt. Sie dient dazu, die Farbe eines bestimmten Eintrags in einer Farbtabelle (Palette) zu definieren. In Systemen, die mit Paletten arbeiten (indizierte Farben),

speichert ein Sprite oder ein Hintergrundbild nicht die Farbe selbst, sondern nur eine Nummer (z. B. "Farbe 1"). SetRGB4 legt fest, welche echte Farbe sich hinter dieser Nummer verbirgt.

- Set: Setze/Definiere.
- RGB: Nutzt das Rot-Grün-Blau-Farbmodell.
- 4: Steht für 4-Bit pro Farbkanal.

Im 23. Teil wird über den Aufruf der *drawScoreInWindow*-Funktion, der initiale Score angezeigt.

```
203             /* Initialer Score */
204             drawScoreInWindow(win, score, FALSE);
```

Abbildung 25 - Der Quellcode für Space-Alien - Teil 23

Im 24. Teil geht es dann mit der Haupt-Spielschleife los. Solange die Variable *running* den Wert *TRUE* besitzt, wird diese Schleife durchlaufen.

```
205             /* Haupt-Spielschleife */
206             while(running) {
207                 WaitTOF();
208                 collision = *CLXDAT; /* Denise-Register abfragen */
```

Abbildung 26 - Der Quellcode für Space-Alien - Teil 24

Was hat es mit dem Aufruf

```
WaitTOF();
```

auf sich? Die Funktion *WaitTOF()* (Wait for Top of Frame) ist ein essentieller Befehl aus der Amiga-Programmierung (speziell der graphics.library). Sie dient dazu, das Programm mit dem vertikalen Rücklauf des Monitors zu synchronisieren. WaitTOF() pausiert die Ausführung deines Programms so lange, bis der Elektronenstrahl des Monitors das Ende des aktuellen Bildes erreicht hat und oben links wieder neu ansetzt. Ohne die Nutzung dieser Funktion entsteht "*Tearing*" (Zerreißen des Bildes), da die obere Hälfte des Bildschirms noch den alten Zustand zeigt und die untere Hälfte bereits den neuen. Mit der Nutzung werden alle Grafik-Updates im "unsichtbaren" Moment durchgeführt, bevor der Strahl das nächste Bild zeichnet. Das Ergebnis ist eine butterweiche Animation. Die Funktion ist sehr ressourcensparend. Anstatt die CPU in einer "*Busy Loop*" (Dauerschleife) quasi heißlaufen zu lassen, versetzt WaitTOF() den Task in einen Wartezustand. Das Betriebssystem (AmigaOS) weckt das Programm erst wieder auf, wenn der entsprechende Hardware-Interrupt der Grafik-Hardware signalisiert, dass das Bildende erreicht ist. Kommen wir zur nächsten Zeile.

```
collision = *CLXDAT;
```

Hierüber wird das Denise-Register abgefragt und auf eine mögliche Kollision der Sprites hin untersucht. Hier muss ich ein wenig ausholen. Die Bits in *CLXDAT* sind wie folgt aufgeteilt, wobei ich mich auf eine verkürzte Version der Tabelle beziehe, die nur Sprites beziehungsweise Sprite-Gruppen behandelt.

Bit-Nummer	Kollision
15	N/A - Nicht verwendet
14	Sprite 4/5 mit Sprite 6/7
13	Sprite 2/3 mit Sprite 6/7
12	Sprite 2/3 mit Sprite 4/5
11	Sprite 0/1 mit Sprite 6/7
10	Sprite 0/1 mit Sprite 4/5
9	Sprite 0/1 mit Sprite 2/3

Tabelle 1 - Sprite-Kollisionen

Sehen wir uns noch einmal die schon gezeigten Zeilen an, in denen die Sprite-IDs vergeben wurden.

```
if ((sn1 = GetSprite(&sPlayer, 1)) >= 0 &&
    (sn2 = GetSprite(&sTarget, 2)) >= 0 &&
    (snP = GetSprite(&sProj,   4)) >= 0) {...
```

Wichtig ist hier natürlich die *ID 2* des Zieles (sTarget) und die *ID 4* des Projektils (sProj). In der Tabelle habe ich die entsprechende Zeile farblich hervorgehoben, die das zu überprüfende Bit 12 zeigt. Wenn also das Sprite mit der ID 2 oder 3 mit dem Sprite der ID 4 oder 5 kollidiert, wird das Bit 12 gesetzt. Nun müssen wir dieses später abfragen, um auf eine Kollision zu schließen. Ich fahre mit dem 25. Teil fort.

```
209         /* 1. EVENTS */
210         while(msg = (struct IntuiMessage*)GetMsg(win->UserPort)) {
211             if(msg->Class == CLOSEWINDOW) running = FALSE;
212             if(msg->Class == RAWKEY) {
213                 UWORD code = msg->Code & 0x7F;
214                 BOOL isKeyUp = (msg->Code & 0x80);
215                 if (code == 0x4E) keyRight = !isKeyUp;
216                 if (code == 0x4F) keyLeft  = !isKeyUp;
217                 if (code == 0x40 && !isKeyUp && !prActive) {
218                     prActive = TRUE; prX = pX + 4; prY = pY - 4;
219                 }
220             }
221             ReplyMsg((struct Message*)msg);
222         }
```

Abbildung 27 - Der Quellcode für Space-Alien - Teil 25

Hier erfolgt die Abfrage der Ereignisse, die abgefangen werden. Über den Ausdruck

```
GetMsg(win->UserPort)
```

wird das Event ausgelesen. Wird

```
if(msg->Class == CLOSEWINDOW)
```

erkannt, kommt es zum Schließen des Fensters, respektive zur Beendigung des Programms, weil die Variable *running* auf *FALSE* gesetzt wird, was zum Verlassen der Haupt-Spielschleife führt. Hinsichtlich der Tastatur werden die folgenden Tasten abgefragt.

was über die Zeile

```
if(msg->Class == RAWKEY)
```

erfolgt. Dabei stehen die folgenden Werte für die Auswertung bereit.

- 0x4E → Rechts
- 0x4F → Links
- 0x40 → Space

Wichtig ist zu sehen, dass beim Drücken der Leertaste für einen Schuss die Variable *prActive* auf *TRUE* gesetzt wird, was anzeigt, dass das Projektil unterwegs ist. Wir kommen gleich noch einmal darauf zu sprechen. Der Code

```
msg->Code & 0x80
```

prüft, ob die Taste losgelassen wurde. Sehen wir weiter mit dem 26. Teil des Codes, in dem die Spielerbewegung und Begrenzung innerhalb des Fensters stattfindet.

```
224         /* 2. LOGIK: Spieler */
225         if (keyRight && pX < 280) pX += 4;
226         if (keyLeft  && pX > 10)  pX -= 4;
```

Abbildung 28 - Der Quellcode für Space-Alien - Teil 26

Im 27. Teil des Codes erfolgt die horizontale Bewegung des Gegners.

```
228         /* 3. LOGIK: Ziel (sinkt bei Wandberührung) */
229         tX += tDir;
230         if(tX > 280 || tX < 10) {
231             tDir = -tDir;
232             tY += 10;
233
234             /* Game Over Bedingung */
235             if(tY > 125) {
236                 running = FALSE;
237                 drawScoreInWindow(win, score, TRUE);
238                 DisplayBeep(NULL);
239             }
240         }
```

Abbildung 29 - Der Quellcode für Space-Alien - Teil 27

Über die Zeile

```
tX += tDir;
```

wird das Space-Alien horizontal positioniert. Wenn die Wand erreicht wird, erfolgt ein Richtungswechsel und Tiefersetzen des Raumschiffs über

```
if(tX > 280 || tX < 10) {
  tDir = -tDir;
```

```
tY += 10;
```

Wurde der untere Rand erreicht, endet das Spiel und es wird der Score über den Aufruf der *drawScoreInWindow*-Funktion angezeigt. Im 28. Teil erfolgt die Steuerung des Projektils und die Kollisionsüberprüfung.

```
242         /* 4. LOGIK: Projektil & Kollision */
243         if(prActive) {
244             prY -= 6;
245             if((collision & 0x1000) && !hitRegistered) {
246                 hitRegistered = TRUE;
247                 score++;
248                 /* Geschwindigkeit bei Treffer erhoehen */
249                 if (tDir>0) tDir++;
250                 else tDir--;
251                 tY -= 5;
252                 if (tY < 20) tY = 20;
253                 prActive = FALSE;      /* Schuss sofort beenden */
254                 drawScoreInWindow(win, score, FALSE);
255                 DisplayBeep(NULL);
256             }
```

Abbildung 30 - Der Quellcode für Space-Alien - Teil 28

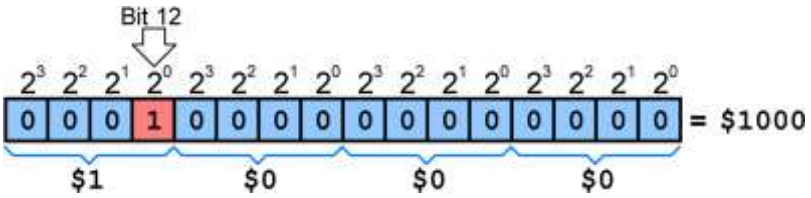
Wenn die Variable *prActive* über das Abfeuern des Projektils auf *TRUE* gesetzt wurde, kann das Projektil bewegt werden. Die Zeile

```
prY -= 6;
```

bewegt dabei das Projektil schrittweise nach oben in Richtung des vermeintlichen Zieles. Kommen wir zur Treffererkennung über die Zeile

```
if((collision & 0x1000) && !hitRegistered) {...
```

Der hexadezimale Wert *0x1000* steht dabei für das Bit 12.



Wird dieser Wert über eine binäre UND-Verknüpfung mit *collision* erkannt, ist es zu einer Kollision mit den beiden Sprites gekommen. Der Amiga unterscheidet nicht, ob Sprite 2 oder 3 trafen, sondern nur das Paar. Es wird ein optisches Signal gegeben und *prActive* auf FALSE gesetzt. Bei einem Treffer erfolgen die nachfolgenden Punkte:

- Score++
- Geschwindigkeit erhöhen
- Gegner leicht nach oben
- Schuss deaktivieren
- Beep

Ist das Projektil außerhalb des Bildschirms zu verorten, wird der Schuss beendet, wie das im nachfolgenden 29. Teil zu sehen ist.

```
258             if(prY < win->BorderTop) {
259                 prActive = FALSE;
260                 hitRegistered = FALSE;
261             }
262         } else hitRegistered = FALSE;
```

Abbildung 31 - Der Quellcode für Space-Alien - Teil 29

Der Schuss wird über die Zeilen

```
prActive = FALSE;
```

```
hitRegistered = FALSE;
```

deaktiviert. Die Variable *hitRegistered* ist dafür zuständig, dass bei einer Kollision auch nur ein Treffer erkannt wird. Kommen wir nun im 30. Teil zum Hardware-Sprite-Update, bei dem die Sprite-Positionen aktualisiert werden.

```
264         /* 5. HARDWARE UPDATE */
265         MoveSprite(&win->WScreen->ViewPort, &sPlayer,
266             (LONG)win->LeftEdge + pX, (LONG)win->TopEdge + pY);
267         MoveSprite(&win->WScreen->ViewPort, &sTarget,
268             (LONG)win->LeftEdge + tX, (LONG)win->TopEdge + tY);
269         if(prActive)
270             MoveSprite(&win->WScreen->ViewPort, &sProj,
271                 (LONG)win->LeftEdge + prX,
272                 (LONG)win->TopEdge + prY);
273         else
274             MoveSprite(&win->WScreen->ViewPort, &sProj,
275                 (LONG)win->LeftEdge + pX,
276                 (LONG)win->TopEdge  + pY - 5); /* An Kanone setzen */
277     }
```

Abbildung 32 - Der Quellcode für Space-Alien - Teil 30

Die *MoveSprite*-Funktion ist das Gegenstück zu *ChangeSprite*. Während ChangeSprite das Aussehen verändert, kümmert sich MoveSprite um die Position eines Sprites auf dem Bildschirm. Wenn kein Schuss aktiv ist, befindet sich das Projektil oberhalb der Kanone. Im 31. Teil wird die Beendigung des Spiels eingeleitet und über die *Delay*-Funktion eine kurze Pause für das Lesen des Punktestandes bewirkt.

```
279             /* Pause nach Spielende, damit der Text gelesen werden kann */
280             Delay(200);
281             /* Sprites freigeben */
282             FreeSprite(sn1); FreeSprite(sn2); FreeSprite(snP);
283         }
284     }
```

Abbildung 33 - Der Quellcode für Space-Alien - Teil 31

Die Zeilen

```
FreeSprite(sn1);
```

```
FreeSprite(sn2);
```

```
FreeSprite(snP);
```

geben die Sprite-Ressourcen wieder frei. Kommen wir zum letzten Teil des Quellcodes, in dem über die *FreeMem*-Funktion der Speicher freigegeben wird.

```
286     /* Ressourcen freigeben */
287     if (cData1) FreeMem(cData1, (LONG)sizeof(rawData1));
288     if (cData2) FreeMem(cData2, (LONG)sizeof(rawData2));
289     if (cData3) FreeMem(cData3, (LONG)sizeof(rawData3));
290
291     /* Gesicherte Farben wieder herstellen */
292     LoadRGB4(&win->WScreen->ViewPort, OldColors, 32);
293
294     CloseWindow(win);
295     CloseLibrary((struct Library *)GfxBase);
296     CloseLibrary((struct Library *)IntuitionBase);
297     return 0;
298 }
```

Abbildung 34 - Der Quellcode für Space-Alien - Teil 32

Bevor das Programm beendet wird, werden über die Zeile

```
LoadRGB4(&win->WScreen->ViewPort, OldColors, 32);
```

die zuvor gesicherten Farben aus dem *OldColors*-Array wieder geladen und die Palette restauriert. Zum Schluss erfolgt das Schließen des Fensters und der Libraries über

```
CloseWindow(win);

CloseLibrary((struct Library *)GfxBase);

CloseLibrary((struct Library *)IntuitionBase);
```

Nähere Informationen sind natürlich in meinem Buch zu finden.

Viel Spaß beim Programmieren und Ausprobieren!

Erik Bartmann

https://erik-bartmann.de/