Spiele unter Linux (SDL)

Holger Dammertz

24. Juni 2002

Zusammenfassung

In den letzten Jahren haben sich die Möglichkeiten, Spiele unter Linux zu programmieren, schnell entwickelt. Ein kurzer Überblick über die geschichtliche Entwicklung von Spielen unter Linux und eine Zusammenfassung des aktuellen Standes stellen Linux als Spieleplattform vor. Als Spieleentwicklungsplattform stellt Linux nützliche Tools und Bibliotheken zu Verfügung, von denen einige näher betrachtet werden. Am Beispiel der SDL-Bibliothek wird außerdem gezeigt, wie Teile einer Game-Engine unter Linux entwickelt werden können.

1 Linux als Spieleplattform

1.1 Historische Entwicklung

Die ersten Spiele unter Linux waren diejenigen, die auf allen Unix-Plattformen verbreitet waren. Sie besaßen kaum oder gar keine Grafik und ließen sich somit einfach auf verschiedene Plattformen portieren. Diese Spiele mussten mit einem zeilenbasierten Display auskommen, und somit handelte es sich bei den meisten um Textadventures. Als dann an der U.C. Berkeley eine Bibliothek zur beliebigen Positionierung des Cursors auf dem Bildschirm entwickelt wurde[1], war zwar immer noch keine Grafik möglich, aber man konnte nun den Zeichensatz dazu verwenden, einen Character auf dem Bildschirm zu repräsentieren.[1] Der Klassiker dieser Art von Spielen ist Rogue. Andere bekannte Beispiele sind Angband und Nethack. Sie sind auch heute noch in der Entwicklung, benötigen aber in der Grundversion kein grafikfähiges Display. Abbildung 1 zeigt als Beispiel die am 20.3.2002 veröffentlichte Version 3.4.0 von Nethack.

Im Laufe der Zeit gab es dann auch immer mehr Möglichkeiten, graphische Anwendungen unter Linux zu entwickeln. So ermöglicht die SVGALib (mit Root-Rechten) direkten Zugriff auf SVGA-kompatible Grafikkarten und ist damit eine Möglichkeit, direkt Grafik darzustellen.

Das für graphische Anwendungen entworfene X Window System (unter Linux meist die XFree86 Implementierung) ist heute die Grundlage für fast alle grafischen Spiele unter Linux. So unterstützt XFree86 mittlerweile fast jede erhältliche 2D Hardware. Da aber X relativ kompliziert zu programmieren ist, benötigen "reine" X-Spiele eine lange Entwicklungszeit. Hauptsächlich sind Open Source Spiele wie Tetris oder Minesweeper entwickelt worden. Vor allem die fehlende Unterstützung für 3D-Beschleuniger ließ die Entwickler komplexer kommerzieller Spiele fast ausschließlich für die Windows-Plattform programmieren.[2]

Das bedeutet aber nicht, dass es keine guten Spiele für die Linux-Plattform gibt. So entwickelten Fans von erfolgreichen Windows-Spielen sogenannte Clones. Diese Spiele besitzen dieselbe Logik wie ihre Vorbilder, sind aber mit vollständigem Game Content[2] frei erhältilich. Ein bekanntes Beispiel hierfür ist Free-Civ, das mittlerweile von einer großen Fan-Gemeinde weiterentwickelt wird (Abbildung 2). Ein anderes

[1] von Ken Arnold mit dem Namen "curses"

[2] Game Content ist ein Sammelbegiff für Grafik, Sound und Leveldaten, die das eigentliche Spiel ausmachen

Abbildung 1: Nethack Version 3.4.0 (tty)

Beispiel ist AleClone/FreeCraft (WarCraft II), eine Echtzeit-Strategie Spielengine, die es sogar ermöglicht, die Grafiken des Originals zu verwenden. Zusätzlich wurden Spiele wie Doom und Descent nach Linux

Abbildung 2: FreeCiv mit Civ II basiertem Tile Set

portiert. Das heißt, dass auf Basis der original Game Engine eine Linux-Version entwickelt wurde. Die Portierungen basieren meistens auf dem vom Hersteller unter der GPL veröffentlichten Source Code der Windows-Version. Trotzdem benötigt man zum Spielen die ursprüngliche (Windows-)Version des Spiels, da der Game Content natürlich nicht frei erhältlich gemacht wurde.

1.2 Aktueller Stand

Mittlerweile sind eine ganze Reihe guter Spiele nach Linux portiert worden. Die Firma Loki hatte sich sogar auf das Portieren von Spielen auf Linux spezialiert. So wurden 1999 von Loki das Spiel “Civilization: Call to Power” und 2000 die Spiele “Soldier of Fortune” und “Descent 3” nach Linux portiert. Es folgten noch zahlreiche andere Spiele, bis die Firma am 30.01.2002 aus Liquiditätsgründen schließen musste. Der portierte Quellcode wurde an die ursprünglichen Hersteller zurückgegeben und man kann hoffen, dass sie weiterhin Support für die Linux-Versionen ihrer Spiele anbieten. [3]

Es gibt aber nicht nur Portierungen von Windows Spielen. Einige Hersteller entwickeln mittlerweile parallel für Windows und Linux (und teilweise auch noch für MacOS X). Zum Beispiel war die Linux-Version von Quake 3 Arena nur wenige Wochen nach dem Windows-Release erhältlich. Das Multiplayer-Online-Spiel “Neverwinter Nights” von Bioware Corp. soll gleichzeitig für Windows, Linux und Mac erscheinen.

Auch das Angebot an kostenlosen Spielen ist um ein Vielfaches größer geworden, wobei ein großer Teil davon noch in der Entwicklung ist. So gibt es unzählige kleine Spiele, die sich zum Teil an alten “Klassikern” orientieren und im Internet kostenlos dDownloadbar sind. Zusätzlich gibt es einige ehrgeizigen Projekte, die schon vielversprechend aussehen. Auch liefern die großen Linux-Desktops Gnome und KDE

einen ganzen Satz von Desktop Spielen gleich mit. Für KDE gibt es sogar ein Projekt, welches zentral verwaltete Highscore Listen im Internet ermöglichen soll.

1.3 Wine und WineX

Wine ist eine Implementierung der Win32 und Win16 API unter X. Es ermöglicht die Ausführung von Windowsprogrammen direkt unter Linux, ohne dass Windows auf dem Rechner installiert ist. Zusätzlich gibt es noch eine Bibliothek (Winelib), die das Portieren von Windows Quellcode nach Linux ermöglicht. Wine ist noch stark in der Entwicklung, implementiert aber schon 90% der Windows Systemaufrufe. Somit laufen schon zahlreiche Spiele mit Hilfe von Wine unter Linux. Im Internet findet man einige Anleitungen, wie Wine für ein bestimmtes Spiel konfiguriert werden muss. [4]

WineX basiert auf den Wine Sourcen. Es erweitert die Wine-API um DirectX (speziell Direct3D mit Hardwarebeschleunigung unter Linux). WineX ist ein kommerzielles Projekt, welches extra an neue Spiele angepasst wird, und unterstützt zahlreiche aktuelle Titel.

2 Linux als Spieleentwicklungsplattform

Ein wichtiger Grund dafür, dass es heute zahlreiche freie Spiele gibt, ist wohl die Verfügbarkeit einer großen Auswahl von frei erhältlichen Bibliotheken und Tools, die die Entwicklung von Spielen unter Linux vereinfachen.

Im Folgenden wird kurz erläutert, aus welchen einzelnen Komponenten ein aktuelles Spiel bestehen kann. Dann werden kurz ein paar frei erhältliche Bibliotheken und Tools vorgestellt, die zur Spieleentwicklung unter Linux eingesetzt werden können.

2.1 Komponenten eines Spiels

Ein Spiel lässt sich in einzelne zentrale Komponenten unterteilen. Abbildung 3 zeigt eine mögliche Untergliederung. Hier sind die beiden Schichten Game Content und Game Engine abgebildet. Bei dem Entwurf

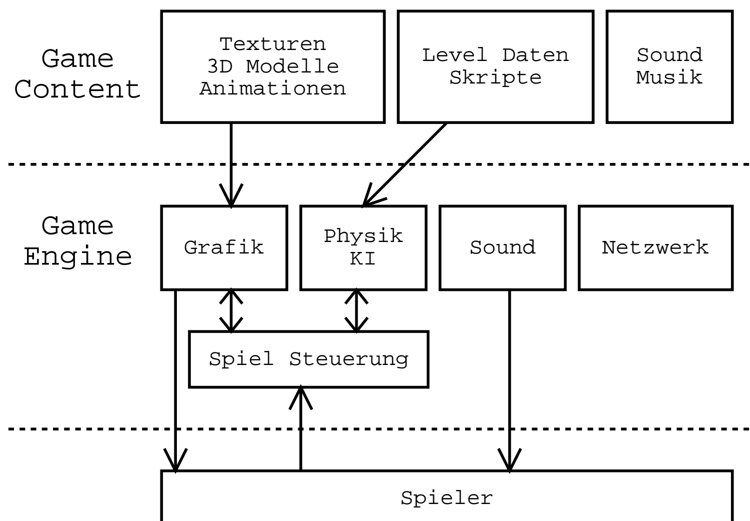

Abbildung 3: Aufbau eines Spiels

eines Spiels kommt in der Praxis noch eine übergeordnete dritte Schicht hinzu, welche u.a. Spiellogik, Character Design und die Story enthält.

2.1.1 Game Content

Unter diesem Begriff werden alle Daten zusammengefasst, die zu einem Spiel gehören und die der Spieler letztendlich hört und sieht. Bei heutigen Spielen gehört es schon fast zum Standard, die Grafik in 3D darzustellen. Um das in Spielen zu ermöglichen, benötigt man eine Vielzahl von 3D Modellen, die die einzelnen Objekte darstellen. Zu einem 3D Modell gehören meist auch eine oder mehrere Texturen, welche diese viel realistischer erscheinen lassen. Wenn auch Charaktere in einer 3D-Welt dargestellt werden sollen, benötigt man noch Animationen für ihre Bewegungsabläufe. Bei einem 2D-Spiel werden meist nur sogenannte Sprites für die Grafikdarstellung benötigt. Das sind ein oder mehrere flache BitMap-Grafiken, die das Spiel am Bildschirm anzeigt. Abbildung 4 zeigt auf der linken Seite ein 3D-Modell aus der Cha-

Abbildung 4: 3D-Modell und 2D-Sprites

racteranimationsbibliothek cal3d 0.8 und rechts Sprites aus dem Spiel Pingus (ein Lemmings-Klon). Bei dem 3D Modell ist zusätzlich noch das Animationsskelett abgebildet. Die Animationen für die Pinguine sind wie bei einem Zeichentrickfilm nacheinander abfolgende Einzelbilder.

Level-Daten werden benötigt, um ein Spiel mit Inhalt zu füllen. Diese stellen meist den logischen Ablauf des Spiels dar und sind die "Welt", in der das Spiel stattfindet. Sound und Musik unterstützen das Geschehen im Spiel und tragen entscheidend zur Atmosphäre bei.

2.1.2 Game Engine

Sie ist der Kern eines Spiels und stellt unter anderem die Verbindung zwischen Spieler und Spiel her. Ein vor allem bei 3D-Spielen elementarer Teil ist die Grafik. Sie ist dafür zuständig, das gesamte Geschehen auf den Bildschirm zu bringen. Die Physik und die KI (Künstliche Intelligenz) sorgen dafür, dass sich das Spiel realistisch verhält. So muss für eine korrekte Kollisionsabfrage[3] gesorgt werden und die Gegner sollten sich so verhalten, dass das Spiel eine Herausforderung darstellt. Die Spielsteuerung muss auf Eingaben vom Benutzer über Tastatur, Maus oder Joystick reagieren und die geeigneten Aktionen im Spiel auslösen. In einfachen Spielen kann sie aus nur wenigen Tasten bestehen, bei komplexeren Spielen aus einer Kombination von Maus/Tastatur, Joystick oder anderen Eingabegeräten.

Die Game Engine muss gegebenenfalls für die Wiedergabe von Musik sorgen. Bei dem Abspielen von Sound-Effekten muss sie eng mit der Physik des Spiels zusammenarbeiten, um das korrekte Timing einzuhalten. Wenn mehrere Personen gleichzeitig (mit- oder gegeneinander) spielen sollen, muss das Spiel zusätzlich noch Netzwerkfunktionen besitzen. Die etwas einfachere Möglichkeit ist, nur das Spielen über ein lokales Netzwerk (LAN) zu ermöglichen. Beim Spielen über das Internet müssen noch zusätzliche Probleme wie Verzögerungen und Cheater[4] berücksichtigt werden.

[3] Prüfen, ob ein Objekt ein anderes berührt

[4] Das sind Leute, die sich durch Veränderungen von Spielinformationen oder der Darstellung Vorteile verschaffen.

2.2 Tools zur “Content Creation”

Für die Linux Plattform gibt es mittlerweile eine enorme Anzahl von Anwendungen, die sich zur Content Entwicklung für Spiele eignen.
Vor allem im Grafik Bereich gibt es leistungsfähige Anwendungen:

GIMP: Ein allgemeines Bitmap-basiertes Bildbearbeitungs- und Zeichenprogramm, mit dem Texturen und Grafiken für 2D Spiele gezeichnet werden können.

PovRay: Ein Raytracer mit dem z.B. Grafiken für Hintergründe in Spielen gerendert werden können.

Blender: Ein 3D Modeller und Renderer mit integrierter Game Engine. Ein komplexes Programm, mit dem auch 3D Modelle für Spiele entworfen werden können. Im Moment ist die weitere Entwicklung von Blender jedoch unklar, da die Mutterfirma finanzielle Probleme hat.

Wenn man bei einem Spiel auf ein eigenes Level Format verzichten kann, so gibt es auch für Leveldesign Anwendungen, die man direkt verwenden kann. Für Spiele die z.B. auf den Quake Sourcen von id Software basieren, gibt es gute Level Editoren.

2.3 Bibliotheken zur Engine Entwicklung

Heutzutage ist es nicht mehr nötig, eine Game Engine von Grund auf neu zu entwickeln. Zahlreiche frei erhältliche Bibliotheken ermöglichen es, einzelne Komponenten direkt oder mit wenig Aufwand in die eigene Game Engine zu integrieren. Es folgt nun eine kleine Auswahl der erhältlichen Bibliotheken mit einer kurzen Beschreibung. Im Anhang befinden sich Links zu diesen und noch einigen weitern:

Allegro: Seit dem 10. Dezember 2001 ist die Allegro Bibliothek in der Version 4 erhältlich. Sie läuft auf einer Vielzahl von Plattformen (DOS, Unix (Linux, FreeBSD, Irix, Solaris), Windows, QNX und BeOS) und besitzt nahezu alle grundlegenden Funktionen, die man zur Programmierung einer Game Engine benötigt. Dazu gehört unter anderen die Unterstützung von 2D Grafik, Sound, Ein/Ausgabe, 3D Routinen und GUI Funktionen.[5]

ClanLib: Die Bibliothek ClanLib ist noch stark in der Entwicklung besitzt aber schon einige sehr nützliche Funktionen und wird auch für einige (einfachere) Spieleprojekte eingesetzt. Es gibt, wie in der Allegro Bibliothek, Funktionen für Grafik, Sound, GUI. Zusätzlich hat man die Möglichkeit, OpenGL zu verwenden und es gibt integrierte Netzwerkfunktionen.[6]

SDL: Die Bibliothek SDL wurde nicht dafür entwickelt, eine vollständige Sammlung von Funktionen zur Spieleprogrammierung zu liefern, sondern stellt eine Abstraktionsebene zwischen der benutzten Hardware und dem Programm dar. Eine ausführlichere Behandlung der SDL ist im 3. Abschnitt zu finden.

2.4 Vollständige Game Engines

Für Linux gibt es einige vollständige (3D) Game Engines. Sie ermöglichen die schnellere Realisierung eines Spiels, allerdings mit dem Nachteil, dass man bei der Gestaltung weniger Freiheit hat:

Crystalspace: Eine vollständige 3D Engine mit 6 Freiheitsgraden. Sie unterstützt verschiede Rendering APIs, besitzt eine integrierte Scriptsprache und sowohl Skeletal- als auch Frame Animationen. Mit Importern kann eine Vielzahl von Formaten in das von Crystalspace konvertiert werden. Weiterhin gibt es Netzwerk Untertüzung, eine Terrain Engine und 3D Sound Funktionen.

Quake I/II: id Software hat den Sourcecode ihrer beiden erfolgreichen Quake Titel unter der GPL veröffentlicht. Auf Grund des Ursprungs eignen sie sich hervorragend für die Entwicklung von 3D Action Spielen. Sie können aber auch für andere 3D-Spiele (z.B. Rollenspiele) eingesetzt werden.

FreeCraft: Die Engine dieses Spiels wird mit dem Ziel entwicklet, eine Basis für beliebige Echtzeitstrategiespiele im Stil von Age of Empires oder Command&Conquer zu bieten.

3 Spieleentwicklung am Beispiel SDL

SDL steht für Simple Directmedia Layer und ist eine freie Cross-Plattform Multimedia API. Ursprünglich wurde sie von Sam Lantinga für die Firma Loki Games zur Portierung von Windows-Spielen auf die Linux-Plattform programmiert. Dank der LGPL Lizenz wird sie nun von einer Vielzahl von Programmierern weiterentwickelt. Sie eignet sich nicht nur für Spiele, sondern wird auch in Multimedia-Programmen wie Video-Player, für Emulatoren und bei Demos eingesetzt.

Alle Code-Beispiele, die in diesem Abschnitt vorgestellt werden, sind dem Mini-Spiel "Chalkboard Pong" entnommen. [7]

3.1 Inhalt der SDL ([8], [9])

Die SDL Bibliothek stellt Funktionen für einen einheitlichen low level Zugriff auf Video-, Audio- und Eingabehardware zur Verfügung. Sie läuft auf einer Vielzahl von verschiedenen Plattformen[5] und eignet sich somit hervorragend für die Cross-Plattform Entwicklung.

Die SDL besteht aus mehreren Teilsystemen, die im Folgenden kurz erläutert werden, wobei einige Besonderheiten erwähnt werden:

3.1.1 Video

Es ist das vollständigste und am meisten benutzte Sub-System der SDL. Neben der Möglichkeit, Bilder im BMP-Format zu laden, gibt es zahlreiche Methoden zum Bearbeiten und Verwalten von Bildflächen.

- Setzen eines Videomodes mit 8bpp oder mehr, mit optionaler Konvertierung, falls der gewünschte Modus von der Hardware nicht unterstützt wird.
- Direkt in einen linearen Graphik Framebuffer schreiben.
- Möglichkeit, Surfaces (Bildflächen) mit einem Colorkey[6] und mit Alphawerten zu erzeugen.
- Beim Kopieren von Surfaces wird automatisch in das Zielformat konvertiert wobei wenn möglich sogar hardwarebeschleunigte Routinen verwendet werden. So werden auf x86 Plattformen MMX optimierte Funktionen verwendet.

3.1.2 Threads und Timer

Die SDL enthält einfache Funktionen für die Thread Erzeugung, Semaphore und Mutexes. Zusätzlich gibt es einige Cross Plattform Funktionen, um Timing in Programmen zu realisieren. So kann man die vergangene Zeit in Millisekunden stoppen, eine bestimmte Zeit warten und einen periodischen Timer mit einer Auflösung von 10ms programmieren.

3.1.3 Datei Ein-/Ausgabe und Endian Unabhängigkeit

Man kann die "Endianness[7]" des Systems feststellen und hat die Möglichkeit Daten mit einer einstellbaren Endianness zu lesen und zu schreiben. Zusätzlich gibt es einen Satz von Makros zum schnellen Konvertieren.

3.1.4 Audio

Es gibt Unterstützung für die Wiedergabe von Audiodatien mit 8-bit und 16-bit Auflösungen, in Mono und Stereo. Dabei wird automatisch konvertiert, falls die Hardware ein gewünschtes Format nicht unterstützt. Die Wiedergabe läuft in einem eigenen Thread, der über Callbacks vom Benutzer mit Daten gefüllt werden kann. Man kann mehrere Sounddateien bei der Wiedergabe mischen, und es gibt eine Funktion zum Laden von Audiodateien im WAVE Format.

[5] In der Version 1.2.4 auf Linux, Win32, BeOS, MacOS und MacOSX.
[6] Die Möglichkeit, eine Farbe als dursichtig zu markieren und so transparente Pixel zu erhalten
[7] Little Endian oder Big Endian

3.1.5 Ereignisse und Fenstermanagement

Die SDL bietet eine einfache Möglichkeit plattformunabhängig ein Fenster zu erzeugen und die grundlegenden Eigenschaften wie Größe und Titel festzulegen.

Die Ereignisbehandlungs-Routinen ermöglichen, auf Eingaben vom Benutzer zu reagieren. Dazu gehören Maus und Tastatur Ereignisse, aber auch Veränderungen am Fenster (z.B. Beenden oder Größe ändern).

3.1.6 Joystick und CD-ROM

Die SDL erkennt an das System angeschlossene Joysticks oder ähnlich Eingabegeräte und kann spezielle Eigenschaften wie Achsenneigung und Knöpfe abfragen. Die CD-ROM API der SDL unterstützt bis zu 32 lokal angeschlossene Laufwerke und bietet alle Funktionen, die man für einen CD Spieler benötigt (u.a. Auflisten der Spuren, Abspielen und Auswerfen der CD).

3.2 Initialisieren der SDL

Bevor man die SDL Bibliothek in eigenen Programmen verwenden kann, müssen die benötigten Sub-Systeme initialisiert werden. Dies geschieht über den Aufruf von `SDL_Init(Uint32 Flags)`. Dabei werden die Standardsysteme Ereignis-Behandlung, Datei Ein/Ausgabe und Threads automatisch mit initalisiert. Den einzelnen auswählbaren Sub-Systemen sind Konstanten zugewiesen, die über ODER verknüpft an die Funktion übergeben werden können.

Um ein Programm, das die SDL initialisiert hat, korrekt zu beenden, muss am Schluss die Funktion `SDL_Quit()` aufgerufen werden. Diese stellt sicher, dass alle Sub-Systeme korrekt heruntergefahren werden und dass der allozierte Speicher wieder freigegeben wird.

Die Fehlerbehandlung geschieht bei den meisten Funktionen über Rückgabewerte. So bedeutet ein Rückgabewert von -1 bei der Funktion `SDL_Init(Flags)`, dass ein Fehler aufgetreten ist. Die Fehlermeldung wird dabei abgespeichert und kann über einen Aufruf von `SDL_GetError()` abgerufen werden.

Wenn man nun Grafik darstellen will, muss man zuerst mit dem Parameter `SDL_INIT_VIDEO` die SDL initialisieren. Nun kann man versuchen einen Bildschirm mit einer wählbaren Auflösung und Farbtiefe zu erzeugen. Diese geschieht über einen Aufruf der Funktion:

```
SDL_Surface *SDL_SetVideoMode(int width, int height, int bpp, Uint32
                              flags);
```

Als Parameter bekommt sie die gewünschte horizontale (width) und vertikale (height) Auflösung des Bildschirms und die Farbtiefe (bpp). Der Parameter Flags gibt den Typ an, und mit welchen Funktionen die Bildschirm Surface erzeugt werden soll. So kann die SDL zum Beispiel versuchen, die Bildfläche direkt im Videospeicher (Flag `SDL_HWSURFACE`) abzulegen. Will man bewegte Grafiken darstellen, so sollte mit dem Parameter `SDL_DOUBLEBUF` die Verwendung des Double Buffers[8] aktiviert werden. Mit weiteren Flags kann man das Fensterverhalten steuern (`SDL_FULLSCREEN, SDL_RESIZABLE`) und die OpenGL Unterstützung aktivieren (`SDL_OPENGL`).

Der Aufruf von `SDL_SetVideoMode(...)` liefert als Rückgabewert einen Zeiger auf eine Struktur vom Typ `SDL_Surface`. Über dieses Handle kann man nun auf die Bildfläche zugreifen.

Das Initialisieren der SDL für die Grafikausgabe kann dann zum Beispiel folgendermaßen aussehen:

```
SDL_Surface *screen;

void init()
{
  /* versuchen das SDL Sub-System Video zu starten */
  if (SDL_Init(SDL_INIT_VIDEO) < 0)
  {
    fprintf(stderr, "\nFehler in der Video Initalisierung: %s", SDL_GetError());
    quit(1);    // zuerst die SDL Beenden, und dann das Programm
  }
```

[8] Zuerst wird das Bild vollständig im Hintergrund gezeichnet und dann vollständig in den Video Speicher kopieren

```
    /* die Bildschimauflösung setzten */
    screen = SDL_SetVideoMode(640, 480, 16, SDL_SWSURFACE | SDL_DOUBLEBUF);

    /* Sicherstellen, das wir ein Handle auf unseren Bildschim bekommen haben */
    if (!screen)
    {
      fprintf(stderr, "\nVideo Modus setzen fehlgeschlagen: %s", SDL_GetError());
      quit(1);
    }
  }

  void quit(int returnCode)
  {
    SDL_Quit();
    exit(returnCode);
  }
```

3.3 Laden und Anzeigen von Bitmaps

Die SDL Bibliothek bietet direkt die Möglichkeit Bilddateien im Windows BMP Format zu laden. Der Aufruf der Funktion `SDL_Surface *SDL_LoadBMP(const char *file)` hat als Rückgabetyp einen Zeiger auf eine `SDL_Surface` und erwartet als Parameter einen Dateinamen.e

In folgendem Beispiel wird nach dem Laden der beiden Bilder “paddle.bmp” und “ball.bmp” noch der Farbwert 0 (in diesem Falle Schwarz, da es sich um 24Bit Bilder handelt) als transparent markiert. Das bedeutet, dass bei einem späteren Anzeigen (Blit) der Bildflächen alle schwarzen Stellen nicht gezeichnet werden, und so der Hintergrund an dieser Stelle sichtbar bleibt.

```
SDL_Surface *paddleIMG;
SDL_Surface *ballIMG;

void loadImages()
{
  paddleIMG = SDL_LoadBMP("paddle.bmp");
  ballIMG = SDL_LoadBMP("ball.bmp");

  /* nun werden alle schwarzen Pixel auf transparent gesetzt */
  SDL_SetColorKey(paddleIMG, SDL_SRCCOLORKEY, 0);
  SDL_SetColorKey(ballIMG, SDL_SRCCOLORKEY, 0);
}
```

Neben der Möglichkeit, Bilder im BMP Format zu laden, kann man auch eine Bildfläche in eine BMP Datei speichern. Ein Aufruf der Funktion `SDL_SaveBMP(SDL_Surface *surface, const char *file)` speichert die übergebene Surface in eine Datei. Dies kann z.B. dazu verwendet werden, um Screenshots im Spiel zu schießen.

Das Anzeigen von Bildern erfolgt über sogenannte Blits. Das bedeutet, dass die Grafikinformationen von einer Quelle in ein Ziel kopiert werden. Bei der SDL lautet die zugehörige Funktion:

```
int SDL_BlitSurface(SDL_Surface *src, SDL_Rect *srcrect, SDL_Surface
                    *dst, SDL_Rect *dstrect);
```

Hier werden die Daten der Quell-Bildfläche (src) in die Ziel-Bildfläche (dst) kopiert. Mit der Struktur `*srcrect` lässt sich ein rechteckiger Bereich der Quelle auswählen und über `*dstrect` lässt sich die Position im Ziel angeben. Sowohl für `*srcrect` als auch für `*dstrect` kann `NULL` übergeben werden. Dann wird das ganze Quellbild kopiert und beim Ziel die Position (0, 0) verwendet.

```
void render()
{
  SDL_Rect dst_rect;

  /* zeichnen des Schlägers von Spieler 1 */
  dst_rect.x = s1X;
  dst_rect.y = s1Y;
  dst_rect.w = paddleIMG->w;
  dst_rect.h = paddleIMG->h;
```

```
  SDL_BlitSurface(paddleIMG, NULL, screen, &dst_rect);

  /* zeichnen des Schlägers von Spieler 2 */
  dst_rect.x = s2X;
  dst_rect.y = s2Y;
  SDL_BlitSurface(paddleIMG, NULL, screen, &dst_rect);

  /* und den Ball zeichnen */
  dst_rect.x = ballX;
  dst_rect.y = ballY;
  dst_rect.w = ballIMG->w;
  dst_rect.h = ballIMG->h;
  SDL_BlitSurface(ballIMG, NULL, screen, &dst_rect);
}
```

3.4 Beispiel einer Gameevent Loop

Nach dem Initalisieren der benötigten Variablen und dem Laden aller verwendeten Grafiken geht ein Spiel normalerweise in eine Schleife über. Diese Gameevent Loop ruft nacheinander alle für das Spiel relevanten Operationen auf. Zu diesen gehören unter anderem:

1. Löschen des Grafikspeichers und/oder Zeichnen eines Hintergrundes.
2. Zeichnen aller sichtbaren Figuren und Objekte.
3. Abfragen der Benutzereingaben (Tastatur, Maus, Joystick).
4. Die Spiellogik abwicklen (u.a. Kollisionen, Spielerbewegung)

Beispiel für eine Gameevent Loop:

```
int finished = 0;

while(finished == 0)
{
  //SDL_FillRect(screen, NULL, 0); // nur nötig wenn kein Hintergrund gemalt wird
  render();

  process_keyboard();
  process_gameLogic();

  SDL_Flip(screen); // anzeigen des im Hintergrund gemalten Bildes
}
```

Die Funktion `SDL_FillRect(SDL_Surface *dst, SDL_Rect *dstrect, Uint32 color)`, die zu Beginn der Schleife aufgerufen wird, zeichnet ein gefülltes Rechteck in der übergebenen Farbe (hier schwarz). Das führt dazu, dass alles, was bisher im Bildspeicher war, gelöscht wird. Anstatt den Bildschirm einfach zu löschen, könnte man aber auch ein Hintergrundbild in der Größe der Bildschirmauflösung zeichnen.

Nach dem Löschen und Zeichnen wird nun die Tastaturbehandlungsroutine aufgerufen:

```
int s1KeyUp, s1KeyDown; // die Tasten für Spieler 1
int s2KeyUp, s2KeyDown; // und die Tasten für Spieler 2

void process_keyboard()
{
  SDL_Event event;

    while(SDL_PollEvent(&event))
    {
      /* Die SDL Event-Behandlung in der Hauptschleife */
      switch(event.type)
      {
        case SDL_KEYDOWN:
          if (event.key.keysym.sym == SDLK_q) s1KeyUp = 1;
```

```
          else if (event.key.keysym.sym == SDLK_a) s1KeyDown = 1;
          else if (event.key.keysym.sym == SDLK_UP) s2KeyUp = 1;
          else if (event.key.keysym.sym == SDLK_DOWN) s2KeyDown = 1;
          break;

        case SDL_KEYUP:
          if (event.key.keysym.sym == SDLK_q) s1KeyUp = 0;
          else if (event.key.keysym.sym == SDLK_a) s1KeyDown = 0;
          else if (event.key.keysym.sym == SDLK_UP) s2KeyUp = 0;
          else if (event.key.keysym.sym == SDLK_DOWN) s2KeyDown = 0;

          else if (event.key.keysym.sym == SDLK_ESCAPE) finished = 1;
          break;

        case SDL_QUIT:
          finished = 1;
          break;

        default:
          break;
      }
    }
}
```

Die SDL Event Behandlung wird automatisch bei einem Aufruf von `SDL_Init(Uint32 Flags)` initalisiert. Intern werden dann ankommende Events in einer Warteschlange gespeichert und können mit der Funktion `int SDL_PollEvent(SDL_Event *event)` nacheinander abgerufen werden. Sind keine weiteren Events mehr vorhanden, so gibt der Aufruf 0 zurück.

Die Event-Struktur enthält Informationen über den Typ des Ereignisses, anhand derer man entscheiden kann, wie die weiteren Informationen des Events abgefragt werden können. Im obigen Beispiel werden die Eventtypen `SDL_KEYUP, SDL_KEYDOWN` und `SDL_QUIT` abgefragt. Weiterhin könnte man noch auf Mouse-, Joystick- und Fensterveränderungen reagieren.

Wenn nun ein `SDL_KEYDOWN` Event erkannt wird, kann über die Struktur `SDL_KeyboardEvent` mit den Namen `key` die Taste abgefragt werden. Falls diese für das Spiel relevant ist, wird die entsprechende globale Variable auf 1 gesetzt. Bei einem `SDL_KEYUP`, was dem Loslassen einer Taste entspricht, wird die Variable wieder zurück auf 0 gesetzt. Die Funktion `process_gameLogic()` kann so auf die gedrückten Tasten reagieren:

```
void process_gameLogic()
{
  /* Bewegung des Schläger von Spieler 1 */
  if ((s1KeyUp == 1) && (s1Y > MIN_PLAYER_Y)) s1Y -= PLAYER_SPEED;
  if ((s1KeyDown == 1) && (s1Y < MAX_PLAYER_Y)) s1Y += PLAYER_SPEED;
  /* Bewegung des Schläger von Spieler 1 */
  if ((s2KeyUp == 1) && (s2Y > MIN_PLAYER_Y)) s2Y -= PLAYER_SPEED;
  if ((s2KeyDown == 1) && (s2Y < MAX_PLAYER_Y)) s2Y += PLAYER_SPEED;

  /* den Ball in Richtung seines Bewegungsvektors verschieben */
  ballX += balldX;
  ballY += balldY;

  .
  . // Die Kollisionsabfrage für Ball<->Schläger
  .

  /* Abprallen des Balls am Bildschirmrand */
  if ((ballX <= 0) || ((ballX+ballIMG->w) >= 640))
      balldX = -balldX;
  if (((ballY <= 0) || ((ballY+ballIMG->h) >= 480)))
      balldY = -balldY;

}
```

4 Ausblick

Die Möglichkeiten, Spiele unter Linux zu spielen und zu entwickeln werden immer besser. Wie schon in Abschnitt 1.2 erwähnt, werden trotz der Pleite von Loki weiterhin hervorragende Spiele für Linux entwickelt. So wird es zum Beispiel ziemlich sicher Linux-Binaries von Doom III geben[9]. Auch andere Hersteller werden, vor allem bei Multiplayerspielen, immer häufiger auch eine Linux-Version herrausbringen.

Eine interessanter Grund, Spiele für die Linux-Platform zu entwickeln ist auch, dass Linux, die SDL und OpenGL auf der Playstation 2 laufen. Seit Mai 2002 ist das PS2 Linux Kit über www.linuxplay.com auch in Europa erhältlich. Ob sich das ganze als Plattform für kommerzielle Spieleentwicklung eignet, ist fraglich. Gerade Konsolenspiele müssen die Hardware immer besser ausnutzen, da eine Erweiterung der Hardware nich möglich ist; aber für interessante Experimente eignet es sich auf jeden Fall.

Die erwähnten Tools und Bibliotheken werden kontinuierlich weiterentwickelt. Damit wird es zum einen immer einfacher, kleine und mittlere Spiele für Linux zu entwickeln. Zum anderen werden dank Projekten wie Mesa 3D und SDL auch weiterhin "State of the Art" Spiele unter Linux möglich sein.

5 Links

Im Text erwähnte Spiele:
Nethack: http://www.nethack.org
Angband: http://www.phial.com/angband
FreeCiv: http://www.freeciv.org
FreeCraft: http://freecraft.sourceforge.net
Descent 1 (Source Project): http://d1x.warpcore.org
Quake (I/II/III): http://www.idsoftware.com

Spiele-Bibliotheken:
SDL: http://www.libsdl.org
Allegro: http://alleg.sourceforge.net
ClanLib: http://www.clanlib.org
PLIB: http://plib.sourceforge.net
Power Pak Game SDK: http://www.angelfire.com/va/powerpakgsdk
GLUT: http://www.opengl.org/developers/documentation/glut
Crystal Space 3D Engine: http://crystal.sourceforge.net

Hilfreiche Programme zur Spiele Entwicklung:
The Gimp: http://www.gimp.org
POV-Ray: http://www.povray.org
Blender: http://www.blender3d.com

Quest: http://quest-ed.sourceforge.net (ein 3D Karten Editor)

Literatur

[1] Glenn R. Wichman. A brief history of rogue. *http://www.wichman.org/roguehistory.html*, 1997. 1

[2] Harald Radke. Tux's secret obsession - gaming under linux. *http://www.linuxfocus.org/English/November1999/article118.html*, November 1999. 1

[3] Loki games. *http://www.lokigames.com*, 2002. 2

[4] Wine development hq. *http://www.winhq.com*, 2002. 3

[9] jedoch vorerst nur mit NVIDIA unterstützung (http://slashdot.org/comments.pl?sid=33453&cid=3619372)

[5] The allegro faq. *http://alleg.sourceforge.net/faq.html*, Mai 2002. 5

[6] Clanlib documentation 0.7 overview.
http://dark.x.dtu.dk/ sphair/cvs/Libs/ClanLib-0.7/Documentation/Overview/index.html, 2002. 5

[7] Holger Dammertz. Chalkboard pong v 0.4. Mai 2002. 6

[8] Sam Lantinga Bernd Kreimeier. Linux in game development.
http://www.gdconf.com/archives/proceedings/2001/kreimeier, 2001. 6

[9] Martin Donlon Sam Lantinga. Sdl library documentation - v1.2.3-rev1. *http://sdldoc.csn.ul.ie*, September 2001. 6