Projekt Informatik 13/II

Jack
–
The Adventureman

04. April 2003

Matthias Voigt

Inhalt

Einleitung

Wie es sich bei dem Titel des Programms vielleicht vermuten lässt, handelt es sich bei meinem Programm um ein Spiel – genauer ein Jump and Run. Ziel des Spieles ist es, seine Spielfigur möglichst unbeschadet zum Ausgang kommen zu lassen. Um dieses Ziel etwas schwerer zu gestalten, befinden sich auf dem Weg einige Hindernisse und Gegner, welche es zu überwinden gilt.

Nachdem dass Spiel gestartet wurde, befindet man sich nach dem Eröffnungsschirm im Hauptmenü.
Hier hat man die Auswahl zwischen mehreren Punkten:

- *Spiel Starten*
- *Highscore*
- *Optionen*
- *Hilfe*
- *Pingpong*
- *Paintbrush*
- *Welt-Editor*
- *Credits*
- *Beenden*

Spiel Starten
Hier kann man ein neues Spiel starten. Man befindet sich dann sofort im ersten Level und darf anfangen zu spielen.

Highscore
Hier sieht man die Liste der besten Spieler. Je mehr Punkte man erreicht hat, desto höher ist man in der Wertung. Dabei wird auch das Datum gespeichert, damit man seine zeitliche Verbesserung nachvollziehen kann.

Optionen
Verschiedene Einstellungen können hier verändert werden.

Hilfe
Um das Spiel spielen zu können, auch ohne die Dokumentation gelesen zu haben, ist hier eine knappe Hilfe eingebaut. Damit weiß man sofort die wichtigsten Tasten zum spielen.

Pingpong
Hierbei wird das Computerspiel Pingpong gestartet, welches auch aus dem Spiel heraus öfters gespielt werden kann.

Paintbrush
Unter diesem Punkt befinden sich Editoren, um die Grafiken im Spiel zu verändern. So kann man aus den Gegnern Teddybären oder aus Jack ein Alien machen.

Welt-Editor
Wem ein Level zu langweilig ist, kann dieses hier schnell und leicht ändern.

Das Spiel

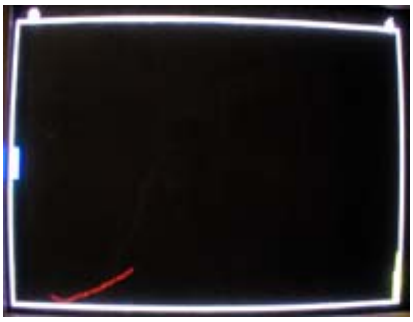
Unter dem ersten Menüpunkt gelangt man direkt zum Spiel. Hier sieht man im Mittelpunkt des Bildschirmes die Hauptfigur: Jack. Mit ihm muss man versuchen möglichst heil zum Ende des Levels zu kommen, und den Ausgang zu finden.

Links oben im Bild ist der aktuelle Punktestand zu sehen. Je höher der ist, umso weiter oben wird man auch in der Highscore Liste erscheinen, also sollte man möglichste viele Punkte machen. Eine schnelle und leichte Möglichkeit dazu ist, die vielen Goldmünzen einzusammeln die überall verstreut herum liegen. In diesem Bild ist sogar eine ganze Kolonne am rechten Rand zu sehen. Die Münzen kann man problemlos durch berühren einsammeln, und erhält dafür standardmäßig 100 Punkte gutgeschrieben.
Eine andere Möglichkeit Punkte zu erhalten ist, einen Gegner zu eliminieren. Dafür gibt es zwei verschiedene Möglichkeiten. Entweder man beschießt ihn mit der *Leertaste*, bekommt dafür aber nur die Hälfte der Punkte gutgeschrieben. Dafür ist es aber sicherer als die andere Art. Dafür springt man einfach auf den Gegner herauf. Man bekommt volle Punktzahl aber hat das Risiko, wenn der Gegner zu schnell ist, dass man ein Herz – also ein Lebenspunkt – abgezogen bekommt.
Die Lebenspunkte sind rechts oben verzeichnet. Sobald kein Herz mehr übrig ist, ist das Spiel zu Ende. Daher sollte man immer vorsichtig laufen, und keine zu gefährlichen Attacken ausprobieren. Aber es gibt auch Wege seinen Herzvorrat wieder aufzufüllen. Direkt unter der Wolke in diesem Bild ist ein Herz, welches man einsammeln kann. Allerdings befinden sich viele Herzen an Stellen, an die man nicht leicht herankommt, und sind vielleicht von Monstern bewacht. Die Lebenspunkte können auch am Computer erhöht werden, allerdings nicht ohne den Meister im Pingpong zu besiegen. Aber keine Angst so gut spielt auch der Computer nicht, und nach ein bisschen Übung kann man ihn leicht besiegen.
Das letzte wichtige Objekt ist der Ausgang. Sobald man ihn erreicht hat, ist der Level geschafft.

Die Steuerung

Um Jack laufen zu lassen, benutzt man die Pfeiltasten nach *Links* und nach *Rechts*. Falls man mal eine Treppe hinauflaufen will, einen Gegner zerspringen, auf eine höher gelegene Plattform oder einfach nur über eine tiefe Schlucht – in welcher man sofort tot ist, egal wie viele Herzen man hat – springen will, benutzt man die Pfeiltaste nach *Oben*. Objekte einsammeln kann man ganz leicht in dem man drüber hinweg läuft. Andere Gegenstände wie den Computer benutzt man auch durch einfaches vorbeilaufen.
Beenden kann man das Spiel durch die *ESC* Taste. Dann sieht man seine erreichte Punktzahl und darf sich in die Highscore Liste eintragen. Zumindest der zehnte Platz ist als Trostpreis sicher, egal wie schlecht man gespielt hat.

Das Pingpong-Spiel



Dieses Spiel kennt sicherlich jeder. Zwei Paddles liegen auf gegenüberliegenden Seiten und können sich nur hoch und herunter bewegen. Der Ball, der sich von Seite zu Seite bewegt, darf sich nicht am Paddle vorbeibewegen, sonst gibt es einen Punkt für den Gegner und der Ball startet wieder aus der Mitte.
Den Paddle bewegt man durch die Cursortasten *Hoch* und *Runter*. An den oberen Ecken sind die Punktzahlen der beiden Spieler ersichtlich. Wer zuerst den 11. Punkt macht, hat das Spiel gewonnen. Bei einer Runde erhöht sich die Geschwindigkeit langsam aber stetig. Dadurch wird das Spielchen immer schneller. Je weiter außen man den Ball am Paddle trifft, umso größer ist der Winkel. Wenn man den Ball genau im Mittelpunkt des Paddles trifft, bewegt sich der Ball senkrecht vom Paddle weg.

Die Editoren

Wie bereits erwähnt kann man mit den Editoren die Graphiken der Spielfigur, der Gegner, der Gegenstände und der Wände und Hintergründe verändern. Als erstes wird hier der ***Sprite-Editor*** genannt. Mit ihm kann man alle festen Wände und nicht feste Wände sowie die Hintergründe umgestalten. Die Bedienung ist relativ einfach. Mit der Leertaste wechselt man zwischen der Farbpalette und dem zu bearbeitenden Objekt hin und her. Mit den Cursortasten wird das Auswahlrechteck in jede Richtung gesteuert. Eine Farbe wählt man so ganz einfach aus. Ist man auf dem Objektteil, so kann man die gewählte Farbe mit *Enter* an der ausgewählten Position setzen. Mit den Tasten *Y* und *C* kann man zwischen den 256 Objekten hin- und herschalten. Das nullte kann nicht verändert werden, weil es zum Löschen dient und an jeder freien Stelle im Level besteht. Mit der *D* Taste kann die Durchgehbarkeit eingestellt werden, d.h. ob man sich durch das Objekt hindurch bewegen kann oder nicht. Die aktuelle Farbe ist unten rechts zu sehen, mit der Farbnummer darunter.
Die anderen Editoren sind dem Sprite-Editor in der Bedienung sehr ähnlich. Der ***Player-Editor*** hat anstatt dem 10x10 Pixel großen Sprite ein 12x24 Pixel großes Feld für den Spieler. Für die Spielfigur gibt es keine weiteren Einstellungen, aber hier bewirken die Tasten *Y* und *C* den Wechsel zwischen den einzelnen Animationsphasen der Figur.
Beim ***Item-Editor***, der dem Sprite-Editor stark ähnelt, editiert man die Objekte die benutzt oder welche man aufnehmen kann. Hier kann deshalb durch drücken der Taste *P* die Punktzahl eingestellt werden. Durch die Tasten *A* und *B* kann man wählen, ob das Objekt beim Berühren benutzt werden kann, oder aufgenommen wird. Beides ist auch möglich. Auch hier wird das aktuelle Item durch die Tasten *Y* und *C* ausgewählt, wobei wiederum das Objekt 0 nicht anwählbar ist, da es auch zum Löschen verwendet wird.
Zum verändern der Gegner wird der ***Enemy-Editor*** benutzt. Hier gibt es 16 verschiedene Gegner zur Auswahl, welche verändert werden können. Auch wird hier wieder wie bei den anderen Editoren zwischen den Farben und den verschiedenen Objekten gewählt. Die Punktzahl entspricht der Punktzahl die man bekommt, wenn man den Gegner einmal trifft.
Der wohl wichtigste Editor ist der ***Welt-Editor***. Mit ihm kann man ein Level nach Lust und Laune bearbeiten und gestalten. Mit den Pfeiltasten wird hier im Levelfenster der Block gewählt der bearbeitet werden soll. Mit den Tasten *A* und *D* wählt man ein Sprite aus und setzt ihn mit *S*. Ein Objekt kann mit *Y* und *C* gewählt werden und mit *X* gesetzt werden. Auf einem Feld dürfen ein Sprite und ein Objekt stehen. Die X und Y Koordinaten stehen rechts unter der Auswahl.

Beschreibung und Spezifikation

Dieses Spiel wurde mit Turbo Pascal 7.0 programmiert. Die Hauptgliederung geschieht in Units. Hier sind alle einmal genannt:

- JackUtil.Pas
- JackGrph.Pas
- JackTCV.Pas
- JackPing.Pas
- JackScrs.Pas

Die Unit JackGrph.Pas

In dieser Unit sind alle mehr oder weniger wichtigen Routinen enthalten, die etwas mit der Graphik-Programmierung zu tun haben. Da ich den Graphikmodus 19 also 13Hex benutze, der nicht kompatibel zu den BGI Graphikmodi von Turbo Pascal ist, musste ich auch alle Standardprozeduren wie PutPixel etc. neu schreiben. Der Grund, warum ich diesen Graphikmodus gewählt habe, liegt an der Geschwindigkeit. Die BGI-Treiber sind ziemlich langsam, und bei einem Spiel kommt es eben auf Geschwindigkeit im Graphikaufbau an.
Die Unit hat eine einzige Globale Variable. Diese Variable heißt ScreenBuf und ist ein Pointer auf den Typ TScreenBuf:

```
TScreenBuf = Array[0..319, 0..199] of Byte;
```

Der Screen-Buffer ist eine Variable, um ein Flackern des Bildschirmes zu verhindern, und um den Graphikaufbau schneller zu gewährleisten, denn der Hauptspeicher ist viel schneller als der Graphikspeicher. Das Flackern wird verhindert, weil alle Bildpunkte später auf einem Schlag mit der Prozedur Move in den Graphikspeicher geschoben werden. Die Variable ist als Global definiert, da auf sie aus verschiedenen Prozeduren zugegriffen wird, die nicht immer etwas miteinander zu tun haben.
Zur Aktivierung des Graphikmodus wird die Prozedur Graphic_Mode mit dem Parameter Mode = 13h aufgerufen:

```
Procedure Graphic_Mode(Mode : Byte); Assembler;
Asm
  mov ah, 00
  mov al, Mode
  int 10h
End;
```

In dieser Prozedur wird die Programmiersprache ***Assembler*** benutzt, welche nur aus Maschinencode besteht. Hiermit kann man sehr Hardware nah programmieren. In dieser Prozedur wird an das Register AX der Wert 0013h (AH sind die hohen 8 Bit und AL die unteren 8 Bit) übergeben. Mit int 10h wird die BIOS-Prozedur 16 (=10Hex) aufgerufen, mit dem Wert 13h. Dadurch stellt das BIOS den Graphikmodus um. Wenn der Wert Mode = 03h übergeben wird, wird wieder in den ursprünglichen Textmodus zurückgewechselt.
In dem Spiel benutze ich noch eine weitere Assembler Routine, die Wait_Retrace heißt. Diese habe ich aus einem Forum im Internet übernommen. Der Zweck ist, dass mit der Routine auf den Zeitpunkt gewartet wird, an dem der Elektronenstrahl im Monitor auf die Position (1; 1) springt. Damit wird weiteres Flackern verhindert, da sich das Bild zeitgleich mit dem Strahl aufbaut.
Ich habe ScreenBuf als einen ***Pointer*** auf den Typ TScreenBuf gewählt. Man hätte auch direkt den Typ TScreenBuf nehmen können, allerdings hat ein Pointer mehrere Vorteile, denn durch

ihn kann man mehr Speicher als 64 KByte benutzen, da ein Pointer ein Zeiger auf eine Adresse im Heap ist. Durch die Prozeduren *GetMem* und *FreeMem* kann dem Pointer Speicher zur Verfügung gestellt und auch wieder entrissen werden. Mit dem Zeichen ^ wird ein Pointer dereferenziert. Wenn man einem Pointer einen Wert direkt zuweist, ändert man nur die Adresse, auf die der Pointer zeigen soll, mit dem Dereferenzierungszeichen wird aber wie gewohnt auf die Variablen zugegriffen.
Um den gesamten Bildschirm schwarz zu bekommen, muss man den gesamten Speicherbereich auf den der Pointer zeigt mit Nullen belegen. Hier könnte man eine Schleife machen oder die Prozedur *FillChar* benutzen. Diese füllt einen Speicherbereich mit der angegebenen Anzahl mit einem bestimmten Byte-Wert aus – wie hier mit einer Null.

```
Procedure Clear_Screen;
Begin
  FillChar(ScreenBuf^, 64000, 0);
End;
```

Um den Bildschirm farbig zu bekommen, muss man natürlich einzelne Pixel setzen können. Diese Aufgabe wird von meiner eigenen PutPixel Prozedur übernommen. Die Parameter sind natürlich die X und Y Koordinaten sowie die Farbe, die gesetzt werden soll. Der Mode 13h hat 256 Farben die gleichzeitig darstellbar sind, deshalb reicht für Color ein Byte. Um bei einem Wert von X oder Y der außerhalb des Monitors ist nichts zu schreiben wird natürlich eine Bereichsüberprüfung vorgenommen, da es fatal wäre in fremde Speisherbreiche zu schreiben. Als nächstes wird überprüft ob die Farbe Null ist. Falls dem so ist, wird an der Stelle kein Pixel gesetzt. Ich habe das so gemacht, damit die Farbe 0 transparent ist, und darunterliegende Bilder nicht übermalt werden.
Wie in der Prozedur Clear_Screen benutze ich ScreenBuf nicht als ein zweidimensionales Array sondern als ein 200x320=64000 Byte großes eindimensionales Array. Das ist deshalb möglich, weil im Speicher alles hintereinanderweg gespeichert wird, und bei einem 2D Array zuerst die erste Zeile, dann die zweite usw. In dieser Prozedur benutze ich ein neues Rechensymbol: **SHL** (SHiftLeft). Damit werden alle Bits in einem Byte oder anderen Typen um eins nach Links versetzt. Das fehlende rechte Bit wird mit einer 0 aufgefüllt. Das ergibt also eine Multiplikation mit 2, da es ein Binäres System ist (wie es im Dezimalen System einer Multiplikation mit 10 entsprechen würde). Das ist aber im Gegensatz zur normalen Multiplikation um einiges leichter für die CPU und damit schneller.
Die Berechnung für die Speicheradresse ist:
$A = X + Y * 320 = X + Y * (256 + 64) = X + Y * 2^8 + Y * 2^6$
Man sieht, dass man Y einmal 8mal mit 2 multiplizieren muss und einmal 6mal mit 2. Das ergibt eine Multiplikation mit 320, geht aber um einiges schneller, und bei 64000 Durchläufen ist das eine große und auch merkliche Zeitersparnis.

```
Procedure PutPixel(X, Y : Word; Color : Byte);
Begin
  If (X >= 0) and (X < 320) and
     (Y >= 0) and (Y < 200) and
     (Color <> 0) then
   Byte(Ptr(Seg(ScreenBuf^),Ofs(ScreenBuf^)+X + Y SHL 8 + Y SHL 6)^)
     := Color;
End;
```

Eine weitere Neuerung ist in der Prozedur SetPalCol. Hier greift man auf die *Ports* zu (die Ein- und Ausgabeadressen der 80x86 CPU), welche als ein großes Array behandelt werden können. Durch den Port 3C8 gibt man Bescheid, dass man die Rot, Grün und Blau Werte der

Farbe Nr setzen will. Dann wird mit 3C9 nacheinander in der Reihenfolge Rot, Grün, Blau der neue Farbwert gesetzt.

```
Procedure SetPalCol(Nr, R, G, B : Byte);
Begin
  Port[$3c8] := Nr;
  Port[$3c9] := R;
  Port[$3c9] := G;
  Port[$3c9] := B;
End;
```

Die Unit JackTCV.Pas

Dies ist meine Konstanten- und Typ-Deklarations-Unit. Hier werden alle Typen und Konstanten definiert, die öfters im Programm gebraucht werden. Ganz Oben steht dabei TPlayer.

```
TPlayer = Record
   Sprite    : Array[0..2] of TPlayer_Sprite;
   ActSprite : Byte;
   Life      : Byte;
   Pos       : TPoint;
   Speed     : TSpeed;
   Dir       : TDir;
   Move      : Boolean;
   Action    : (Walk, Stand, Duck, Fall);
   Score     : LongInt;
   Time      : LongInt;
   JTime     : LongInt;
   LevelDone : Boolean;
 End;
```

Hier wird der Spieler-Typ definiert. Unter anderem die verschiedenen Animationsphasen, die Anzahl der Leben, die Position in der Spielwelt, die Bewegungsgeschwindigkeiten und Bewegungsrichtung.
In dieser Unit gibt es nur eine Prozedur, welche dafür zuständig ist, die Dateinamen für den aktuellen Level zurückzugeben.

```
Procedure Level_Name(var Lvl, Itm, Enm : String; I : Byte);
```

Das Hauptprogramm Jack.Pas

Hierbei handelt es sich um meine Hauptprogrammdatei. Hier befindet sich die Prozedur Main_Proc. Dies ist die Steuerprozedur für alle anderen Prozesse. So werden ersteinmal die Initialisierungsprozeduren aufgerufen, dann wird die Farbpalette gesetzt, der Willkommenschirm gezeigt und das Hauptmenü aufgerufen, welches als Schleife immer wieder eine Auswahl anzeigt. Von hier ist eine Verzweigung über ein Case zu allen anderen übergeordneten Prozeduren wie Play_Game, Show_Highscore oder der World_Editor. Bei einem Abbruch wird die Schleife verlassen, der Abschiedsschirm gezeigt und die Deinitialisierungsprozeduren aufgerufen.

Die Pingpong-Unit JackPing.Pas

Diese Unit ist an sich ein komplett eigenes kleines Spiel, und läuft fast unabhängig von anderen Units – von der Graphikunit und allgemeinen Prozeduren aus der GraphUtil mal abgesehen. Hier gibt es eine Reihe lokaler Variablen, wie zum Beispiel Y_Player und Y_Comp für die Y-Koordinaten der beiden Paddles. Die X-Werte sind jeweils fest. Der Ball hat Gleitkomma-Koordinaten, so dass sich der Ball auch mal 1.5 Einheiten bewegen kann und damit eine exaktere Ballbewegung möglich ist. Damit der Ball gezeichnet werden kann, werden diese Single-Werte vorm Zeichnen in Integer-Werte gerundet. Die Bewegung des Balls ergibt sich aus der Addition der Koordinaten und des Geschwindigkeitsvektors:

```
Ball_XS := Ball_XS + Speed_X/4;
Ball_YS := Ball_YS + Speed_Y/4;
```

Zum Anfang eines jeden Schleifendurchganges wird die Gesamtgeschwindigkeit des Balles erhöht, damit auch längere Runden nicht allzu langweilig werden, und der Schwierigkeitsgrad langsam ansteigt. Der Computer geht dem Ball immer direkt hinterher, indem er vergleicht ob der Ball über oder unter ihm ist, und sich dann in die entsprechende Richtung bewegt.
Der Spieler steuert hier mit den Pfeiltasten. Für dieses Programm habe ich eine neue Tastatur-Prozedur geschrieben, welche auf Taste drücken und Taste loslassen reagiert. Hierzu aber später mehr. Wenn der Ball auf ein Paddle kommt, wird der Y-Bewegungsvektor in Abhängigkeit von der Entfernung zum Mittelpunkt des Vektors neu berechnet. Die X-Bewegungskoordinate wird mit dem Satz des Pythagoras aus der Gesamtgeschwindigkeit minus der Y-Bewegungskoordinate berechnet. Damit ergibt sich immer ein anderer Winkel aber die gleiche Geschwindigkeit (die aber langsam ansteigend ist). Falls der Ball auf eine Außenwand stößt, der Y-Koordinate also einen bestimmten Wert unter- oder überschreitet, wird die Y-Bewegungskoordinate einfach auf der Zahlengerade zu 0 gespiegelt.
Da das Zwischenspiel im Zusammenhang mit dem Jump'n'Run noch etwas zu tun haben soll – denn wenn man das Spiel gewinnt, wird einem ein Leben geschenkt, sowie 1000 Punkte – hat die Steuerfunktion einen Rückgabewert, der angibt, ob man das Spiel gewonnen oder verloren hat.

Die Bildschirme-Unit JackScrs.Pas

In dieser kleinen überschaubaren Unit sind alle Menü-Bildschirme enthalten, wie zum Beispiel der Credits-Screen, der Highscore oder auch das Hauptmenü an sich. Außerdem ist hier auch die Spiel-Steuerprozedur enthalten. Im Hauptmenü wird in einer Schleife jeder Menüpunkt mit leicht veränderter Farbe pro Durchganz aufgebaut. Mit den Tasten Hoch und Runter kann man die einzelnen Punkte wählen. Sobald die Taste Enter gedrückt worden ist, wird die Schleife verlassen, und der Funktionswert ergibt sich aus dem letzten markierten Menüpunkt, welcher mit „Select" dargestellt ist.
Die nächst-oberstgelegene Steuerprozedur für das eigentliche Spiel ist Play_Game. Diese wird bei jedem Spielstart aufgerufen und initialisiert den Spieler (die Variable Player) und lädt die Level in den Speicher. Danach wird wieder die andere Tastatur-Prozedur verknüpft. Dazu wird der Ort des alten Moduls für die Tastatur gespeichert, und die neue Prozedur Check_KeyB referenziert mit dem Keyboard Handler verknüpft. Dann beginnt die eigentliche Steuerschleife, in der alle Aktionen ausgeführt werden. Der Spieler, die Gegner und der Schuß bewegen sich, dann wird überprüft welche Objekte zusammen interagieren, wie zum Beispiel Spieler und Gegenstände:

```
Check_Items(Player);
```

Und zum Schluss wird der Bildschirminhalt auf einem Schlag mit Draw_Screen auf den Monitor gebracht, und ist dort sichtbar für den Benutzer des Spiels.

Nachdem das Spiel beendet ist, wird der alte Tastatur-Handler wieder eingesetzt und der Spieler wird nach seinem Namen gefragt. Anschließend wird der Name und die Punktzahl an die Add_To_Highscore Prozedur übergeben, damit die Leistung in der Datei gespeichert wird.

Die Unit JackUtil.Pas

Die letzte und auch zugleich größte Unit ist die JackUtil.Pas, die alle mehr oder weniger wichtigen Prozeduren enthält, die für das Spiel nötig sind. Diese Unit enthält auch die einzigste echt globale Variable KeyB_Table, welche alle gedrückten Taste als True speichert und nicht gedrückte Tasten als False. Da nur Check_KeyB Prozedure eine Taste empfängt und eine andere Prozedur, welche wissen muss welche Tasten gedrückt sind, nicht interagieren können, musste für diesen Zweck eine globale Variable her. Diese ist Projektweit sichtbar und erlaubt es immer zu wissen ob bestimmte Tasten gedrückt sind.
Unit-globale Parameter sind hier die Puffer für den Levelaufbau, die Items und die Gegner, sowie für die 256 Teilstücke, die 256 Items und die 16 Gegner. Dann ist nur noch ein kleines 128*8 Byte großes Array für die Buchstaben.
Die erste wichtige Prozedur ist hier die Check_KeyB Prozedur. Diese wird aufgerufen, jedes Mal wenn eine Taste gedrückt oder losgelassen wird. Aus dem Port $60 bekommt Key den Wert der Taste. Wenn der Wert kleiner als 128 ist, dann wurde die Taste gedrückt, ansonsten wurde die Taste losgelassen.
Die nächste wichtigere Prozedur ist Text_Out, da nur so eine Ausgabe von Text im Grafikmodus möglich ist. Hier wird jeder Buchstabe einzeln an die Prozedure Draw_Letter übergeben, welche aus dem TFont-Feld den Buchstaben bekommt und daraus dann entweder einen Punkt setzt, oder die Stelle frei lässt – Bit für Bit, Zeile für Zeile.
Danach kommen die P2?? Prozeduren. Durch sie wird aus den Koordinaten der Spielfigur (Player) der Block in dem Feld bekannt. Die einzelnen Prozeduren unterscheiden sich durch X und Y und da jeweils, ob z.B. die linke Seite oder die rechte Seite genommen werden soll.
Als nächstes kommt die Add_To_Highscore Prozedur. Hier wird die Highscore Datei geöffnet und alle Sätze in ein Feld geladen. Dort wird der neue Satz auf den letzten Platz geschrieben und danach wird das Feld mit BubbleSort nach der Punktzahl sortiert. Dann folgt das speichern in die Datei und die Anzeige mit Show_Highscore.
Die nächsten wichtigen Prozeduren sind die Editor-Prozeduren, von denen es ja 5 Stück gibt. Alle sind an sich sehr ähnlich aufgebaut. Denn zuerst wird die betreffende Datei geladen. In der Zählschleife wird zunächst der Farbverlauf gemalt, danach die verschiedenen Ansichten des Sprites und der Nachbar-Sprites. Nun kommt schon die Tastatur-Abfrage, in der entweder das Auswahlfenster verschoben wird oder eine Farbe gesetzt wird. Danach wird das Objekt gespeichert. Falls die Schleife abgebrochen wird, wird die Datei noch geschlossen.
Um etwas Ordnung beim schreiben zu bringen, kommen nach den Editoren die LS und LZ Prozeduren, was für „LeadingZero“ und „LeadingSpace“ steht. Damit wird ein String auf eine bestimmte Anzahl von Stellen gebracht.
Nun folgt die Steuerprozedur zum Zeichnen der Welt. Darin wird das Level, dann die Objekte, dann die Gegner der Player und zuletzt der Schuß gezeichnet. Dann folt eine Ausgabe über Lebenspunkte und die Punkte.
Die Move_Shot Prozedur bewegt den Schuß auf einer horizontalen Strecke bis irgendwann ein Hindernis wie eine Wand oder ein Gegner kommt. Move_Enemies dagegen bewegt die Gegner, welche sich ca. 140 Pixel um die Spielfigur herum befinden. Im Case Teil befinden sich die Steuerstrukturen für die verschiedenen Gegnersorten.
Mit der Prozedur Init_Player werden alle Eigenschaften von Player auf Null zurückgesetzt, damit ein neues Spiel gestartet werden kann zum Beispiel.

In dem Check_Items Modul wird zunächst jede Stelle von der Spielfigur überprüft ob sie ein Objekt berührt. Ist dies der Fall, wird getestet um was für ein Objekt es sich handelt und wie weiter gemacht werden soll – bei einem Computer soll das Spiel Pingpong gestartet werden, und wenn es sogar gewonnen wurde erhält man ein Herz sowie 1000 Punkte.
Mit Get_Number und Get_Name ist eine sicher Eingabe von Zahlen und Strings wie Namen gewährleistet. In einer Schleife wird die gedrückte Taste überprüft. Falls diese nicht zulässig ist, passiert gar nichts, ansonsten wird der String weitergeschrieben.
Mit Check_Enemy wird überprüft, ob der Spieler oder der Schuss mit dem Gegner in Berührung kommen. Wenn dem so ist, dann wird überprüft mit welchem Teil. Falls der Spieler mit den Füßen draufspringt, dann ist der Gegner ohne Lebensabzug tot. Ansonsten verliert man ein Herz. Mit dem Schuss erhält man nie Herzabzug, dafür bekommt man aber nur die halbe Punktzahl.
Die nun folgenden Save und Load Module sind dazu da um Objekte etc. aus Dateien zu laden und in Dateien wieder zu speichern.
Zu guter Letzt folgen noch die Initialisierungsprozedur sowie die Exit-Prozedur der Unit. Dabei werden Speicherbereiche für die Pointer reserviert und Objekte werden geladen.
Am Ende ist noch die Move_Player Prozedur zu erkennen, bei der auf viele Eventualitäten eingegangen wird, die sich beim Bewegen der Spielfigur ergeben könnten. Wenn die Figur in der Luft hängt, wird sie zum Beispiel dann herunterfallen, oder wenn die Leertaste gedrückt wird, wird der Schuss erstellt. Oder bei Taste nach oben und man steht auf dem Boden fängt die Figur an zu springen. Und auch wenn die ESC Taste gedrückt wird, wird hier die Abbruchbedingung für die Steuerprozedur geschaffen.

Versicherung

Hiermit versichere ich, dass ich dieses Program eigenständig erarbeitet habe, und keine Rechte Dritter veletzt habe.

Matthias Voigt

Anhang

Quelltexte:

```
Program JumpinJack;

Uses
  Dos, Crt, JackUtil, JackTCV, JackGrph, JackPing, JackScrs;

Procedure Main_Proc;
Var
  Exit_Game : Boolean;
Begin
  Init_Util;
  Graphic_Mode(19);
  Set_Palette;

  Exit_Game := False;

  Show_Welcome_Screen;
  Repeat
    Case Menu of
      1 : Play_Game;
      2 : Show_HighScore;
      3 : Show_Options;
      4 : Show_Help;
      5 : Start_Ping_Pong;
      6 : Editors;
      7 : World_Editor;
      8 : Show_Credits;
      9 : Exit_Game := True;
    End;
  Until Exit_Game;

  Show_Exit_Screen;

  Graphic_Mode(3);
  Exit_Util;
End;

Begin
  Randomize;
  Main_Proc;
End.
```

```
Unit JackGrph;

Interface

Uses
  JackTCV;

Procedure Graphic_Mode(Mode : Byte);

Procedure Clear_Screen;
Procedure Fill_Screen(Color : Byte);
Procedure Draw_Screen;
Procedure Wait_Retrace;

Procedure PutPixel(X, Y : Word; Color : Byte);
Function  GetPixel(X, Y : Word) : Byte;

Procedure Set_Palette;
Procedure Set_PaletteBW;
Procedure SetPalCol(Nr, R, G, B : Byte);
Procedure GetPalCol(Nr : Byte; var R, G, B : Byte);

Procedure Interpol(I : Byte);

Procedure Graph_Init;
Procedure Graph_Exit;

Implementation

Uses
  JackUtil;

Var
  ScreenBuf : ^TScreenBuf;

Procedure Graphic_Mode(Mode : Byte); Assembler;
Asm
  mov ah, 00
  mov al, Mode
  int 10h
End;

Procedure Clear_Screen;
Begin
  FillChar(ScreenBuf^, 64000, 0);
End;

Procedure Fill_Screen(Color : Byte);
Begin
  FillChar(ScreenBuf^, 64000, Color);
End;

Procedure Wait_Retrace; Assembler;
Asm
  mov dx, 3DAh
  @l1:
  in  al, dx
  and al, 08h
  jnz @l1
  @l2:
  in  al, dx
  and al, 08h
  jz  @l2
End;

Procedure PutPixel(X, Y : Word; Color : Byte);
Begin
  If (X >= 0) and (X < 320) and
     (Y >= 0) and (Y < 200) and
     (Color <> 0) then
   Byte(Ptr(Seg(ScreenBuf^),Ofs(ScreenBuf^)+X + Y SHL 8 + Y SHL 6)^) := Color;
End;

Function GetPixel(X, Y : Word) : Byte;
Begin
  GetPixel := Byte(Ptr(Seg(ScreenBuf^),Ofs(ScreenBuf^)+X + Y SHL 8 + Y SHL 6)^);
{  GetPixel := Byte(Ptr($A000, X + Y SHL 8 + Y SHL 6)^);}
End;

Procedure Set_PaletteBW;
Var
  I : Integer;
Begin
  For I := 0 to 255 do
  Begin
    SetPalCol(I, I div 4, I div 4, I div 4);
  End;
End;

Procedure Set_Palette;
Var
  I, J : Integer;
Begin
  For I := 0 to 15 do
  Begin
    J := I * 4;
    If I > 0 then
      SetPalCol(  I,    J-4,    J-4,  J-4);   { Schwarz Weiss Verlauf }
    SetPalCol( I+16,   47-J div 3,  47-J div 3*2,  0); { Hautfarben }
    SetPalCol( I+32,    J,    0,  0);         { Schwarz Rot }
    SetPalCol( I+48,   63,    J,  J);         { Rot Weiss   }
    SetPalCol( I+64,    0,    J,  0);         {  }
    SetPalCol( I+80,    J,   63,  J);         {  }
    SetPalCol( I+96,    0,    0,  J);         {  }
    SetPalCol(I+112,    J,    J,  63);        {  }
```

```
    SetPalCol(I+128,    J,    J,   0);        {  }
    SetPalCol(I+144,   63,   63,   J);        {  }
    SetPalCol(I+160,    J,    0,   J);        {  }
    SetPalCol(I+176,   63,    J,  63);        {  }
    SetPalCol(I+192,    0,    J,   J);        {  }
    SetPalCol(I+208,    J,   63,  63);        {  }

    SetPalCol(I+224,    16+I,    32-I,  0);
    SetPalCol(I+240,  32+I,    16-I,  0);
  End;
  SetPalCol(15, 63, 63, 63);
End;

Procedure SetPalCol(Nr, R, G, B : Byte);
Begin
  Port[$3c8] := Nr;
  Port[$3c9] := R;
  Port[$3c9] := G;
  Port[$3c9] := B;
End;

Procedure GetPalCol(Nr : Byte; var R, G, B : Byte);
Begin
  Port[$3c7] := Nr;
  R := Port[$3c9];
  G := Port[$3c9];
  B := Port[$3c9];
End;

Procedure Interpol(I : Byte);
Var
  Offs, Addr, C, Col : Word;
  Color, Loop           : Byte;
Begin
  Addr    := Seg(ScreenBuf^);
  For Loop := 1 to I do
  Begin
    For Offs := 0 to 63999 do
    Begin
      Col := 0;
      If Offs > 320 then Inc(Col, MEM[Addr:Offs-320]);
      If Offs >   1 then Inc(Col, MEM[Addr:Offs-  1]);
      If Offs < 63999-  1 then Inc(Col, MEM[Addr:Offs+  1]);
      If Offs < 63999-320 then Inc(Col, MEM[Addr:Offs+320]);
{      If Offs > 321 then Inc(Col, MEM[Addr:Offs-321]);
      If Offs > 319 then Inc(Col, MEM[Addr:Offs-319]);
      If Offs < 63999-319 then Inc(Col, MEM[Addr:Offs+319]);
      If Offs < 63999-321 then Inc(Col, MEM[Addr:Offs+321]);
 }     Color := Col shr 2;
      MEM[Addr:Offs] := Color;
    End;
  End;
End;


Procedure Draw_Screen;
Var
  X, Y : Word;
  B    : Byte;
Begin
  Wait_Retrace;
  Move(ScreenBuf^, MEM[$A000:0000], 64000);
End;


Procedure Graph_Init;
Begin
  GetMem(ScreenBuf, 64000);
End;

Procedure Graph_Exit;
Begin
  FreeMem(ScreenBuf, 64000);
End;

End.
```

```
Unit JackTCV;

Interface

Const
  PL_X = 12;
  PL_Y = 24;

  BLS_X = 10;   { Breite der Blocks }
  BLS_Y = 10;

  SCR_X = 10;   { Anzahl der Bildschirme }
  SCR_Y = 1;

  BL_X = 32;    { Anzahl der Blocks pro Bildschirm }
  BL_Y = 20;

  LVL_X = BL_X * SCR_X;  { Anzahl der Blocks im Level }
  LVL_Y = BL_Y * SCR_Y;

  PX_X = BLS_X * LVL_X;  { Breite des Levels in Pixel }
  PX_Y = BLS_Y * LVL_Y;

  START_LIFE = 3;
  START_X    = 190;
  START_Y    = 90;

  GRAVITY    = 10 SHL 16 div 10;

  FONTS_FILE = 'Fonts.dat';
  PIECE_FILE = 'Pieces.dat';
  ITEM_FILE  = 'Items.dat';
  ENEMY_FILE = 'Enemies.dat';
  PLAYER_FILE = 'Player.dat';
  HIGHSCORE_FILE = 'HiScore.dat';


Type
  TPoint = Record
             X, Y : LongInt;
           End;
  TSpeed = Record
             X, Y : LongInt;
           End;
  TDir    = (Left, Right);

  TPlayer_Sprite      = Array[1..PL_X, 1..PL_Y] of Byte;
  TPlayer_Sprite_File = File of TPlayer_Sprite;
  TPlayer = Record
    Sprite   : Array[0..2] of TPlayer_Sprite;
    ActSprite : Byte;
    Life     : Byte;
    Pos      : TPoint;
    Speed    : TSpeed;
    Dir      : TDir;
    Move     : Boolean;
    Action   : (Walk, Stand, Duck, Fall);
    Score    : LongInt;
    Time     : LongInt;
    JTime    : LongInt;
    LevelDone : Boolean;
  End;

  TShoot = Record
             Pos : TPoint;
             Dir : TDir;
             Active : Boolean;
           End;

  TPlace  = Record
               Day, Month, Year : Word;
              Name : String[14];
              Points : LongInt;
            End;
  THighScore = Array[1..10] of TPlace;
  THigh_File = File of TPlace;


  TSprite       = Array[1..BLS_X, 1..BLS_Y] of Byte;
  TPiece        = Record
                    Sprite : TSprite;
                    Walk   : Boolean;
                    Death  : Boolean;
                  End;

  TPieces       = Array[0..255] of TPiece;
  TPiece_File   = File of TPiece;
  PPieces       = ^TPieces;

  TItem         = Record
                        Sprite  : TSprite;
                    Collect : Boolean;
                    Use          : Boolean;
                    Points  : Integer;
                    Visible : Boolean;
                  End;
  TAllItems     = Array[0..255] of TItem;
  TAllItem_File = File of TItem;
  PAllItems      = ^TAllItems;


  TLevel        = Array[1..LVL_X, 1..LVL_Y] of Byte;

  TItems        = Array[1..LVL_X, 1..LVL_Y] of Byte;
  TItem_File    = File of Byte;
```

```
  PItems        = ^TItems;

  TEnemy  = Record
                   Sprite : TSprite;
                       Points : Word;
                     End;
  TEnemy_Pos    = Record
                         X, Y : Word;
                         Kind : Byte;
                         OX, OY : Integer;
                     Tag  : LongInt;
                         Life : ShortInt;
                  end;
  TEnemy_Kind   = Array[0..15] of TEnemy;
  TEnemy_File       = File of TEnemy;
  PEnemy_Kind   = ^TEnemy_Kind;
  TEnemies      = Array[1..100] of TEnemy_Pos;
  TEnemies_File = File of TEnemy_Pos;
  PEnemies      = ^TEnemies;

  TLevel_File   = File of Byte;
  PLevel        = ^TLevel;

  TScreenBuf        = Array[0..319, 0..199] of Byte;

Procedure Level_Name(var Lvl, Itm, Enm : String; I : Byte);

Implementation

Uses
  JackUtil;

Procedure Level_Name(var Lvl, Itm, Enm : String; I : Byte);
Begin
  Lvl := 'Level' + IntToStr(I) + '.lvl';
  Itm := 'Level' + IntToStr(I) + '.itm';
  Enm := 'Level' + IntToStr(I) + '.enm';
end;


End.
```

```
Unit JackPing;

Interface

Function Start_Ping_Pong : Boolean;

Implementation

uses
  JackUTIL, JackGrph, Dos, Crt;

Procedure Draw_Ball(Bx, By : Integer);
Begin
  PutPixel(  Bx,   By, 50);
  PutPixel(Bx-1,   By, 50);
  PutPixel(Bx+1,   By, 50);
  PutPixel(  Bx, By-1, 50);
  PutPixel(  Bx, By+1, 50);
end;
Procedure Draw_PaddleP(Y : Word);
Var
  i, j : Integer;
Begin
  For I := 1 to 20 do
   For J := 1 to 5 do
    PutPixel(J, Y + I-10, 115);
End;
Procedure Draw_PaddleC(Y : Word);
var
  i, j : Integer;
Begin
  For I := 1 to 20 do
   For J := 1 to 5 do
    PutPixel(320-J, Y + I-10, 145);
End;
Procedure Draw_Cage;
var
  i : Integer;
Begin
  For I := 10 to 199 do
  Begin
    PutPixel(  0, I, 15);
    PutPixel(319, I, 15);
  End;
  For I := 0 to 319 do
  Begin
    PutPixel(I,  10, 15);
    PutPixel(I, 199, 15);
  End;
End;


Function Start_Ping_Pong : Boolean;
Var
  AllUp            : Boolean;
  Y_Player, Y_Comp : Byte;
  Ball_X, Ball_Y   : Word;
  Ball_XS, Ball_YS : Single;
  Ch                 : Char;
  I, J             : Integer;
  Ball_Speed       : Single;
  Speed_X, Speed_Y : Single;
  OldBKHandler     : Pointer;
  CPoints, PPoints : Byte;

Begin
  Y_Player := 99;
  Y_Comp   := 99;

  PPoints  := 0;
  CPoints  := 0;

  Ball_XS  := 159;
  Ball_YS  := 99;

  Speed_X    := 10;
  Speed_Y    := 0;
  Ball_Speed := 10;

  GetIntVec(9, OldBKHandler);
  SetIntVec(9, @Check_KeyB);

  Repeat
    Ball_X := Round(Ball_XS);
    Ball_Y := Round(Ball_YS);
    Ball_Speed := Ball_Speed + 0.01;

    Clear_Screen;
    Draw_Cage;
    Draw_Ball(Ball_X, Ball_Y);
    Draw_PaddleP(Y_Player);
    Draw_PaddleC(Y_Comp);
    Text_out(1, 10, 15, IntToStr(PPoints));
    Text_out(300, 10, 15, IntToStr(CPoints));
    Draw_Screen;
    Delay(10);

    Ball_XS := Ball_XS + Speed_X/4;
    Ball_YS := Ball_YS + Speed_Y/4;

    If ((Ball_XS     >= 314) and
        (Y_Comp + 10 > Ball_YS) and
        (Y_Comp - 10 < Ball_YS)) then
    Begin
      Speed_Y := (Ball_YS - Y_Comp);
```

```
      Speed_X := -Sqrt(Sqr(Ball_Speed) - Sqr(Speed_Y));
    End;

    If ((Ball_XS       <= 6) and
        (Y_Player + 10 > Ball_YS) and
          (Y_Player - 10 < Ball_YS)) then
    Begin
      Speed_Y := (Ball_YS - Y_Player);
      Speed_X := Sqrt(Sqr(Ball_Speed) - Sqr(Speed_Y));
    end;

    If (Ball_YS < 12) or (Ball_YS > 196) then Speed_Y := -Speed_Y;
    If (Ball_XS < 310) then
    Begin
      If (Y_Comp > Ball_YS) Then
      Begin
        If Y_Comp > 20 then Dec(Y_Comp, 2);
      end else
      If Y_Comp < 188 then Inc(Y_Comp, 2);
    end else
      Y_Comp := Random(5)-2 + Y_Comp;

    If Ball_XS <   0 then Inc(CPoints);
    If Ball_XS > 320 then Inc(PPoints);
    If (Ball_XS < 0) or (Ball_XS > 320) then
    Begin
      If Ball_XS < 0 then Speed_X := -10 else Speed_X := 10;
      Speed_Y    := 0;
      Ball_XS    := 159;
      Ball_YS    := 99;
      Ball_Speed := 10;
    End;
    If Keyb_Table[72] then If Y_Player >  20 then Dec(Y_Player, 2);
    If Keyb_Table[80] then If Y_Player < 188 then Inc(Y_Player, 2);
  Until KeyB_Table[1] or (CPoints = 11) or (PPoints = 11);

  Repeat
    AllUp := True;
    For I := 1 to 128 do If Keyb_Table[I] then AllUp := False;
  Until AllUp = True;
  SetIntVec(9, OldBkHandler);

  If PPoints = 11 then
    Start_Ping_Pong := True
  else
    Start_Ping_Pong := False;
End;

End.
```

```
Unit JackScrs;

Interface

Procedure Show_Credits;
Procedure Show_Options;
Procedure Show_Help;
Procedure Show_HighScore;
Procedure Show_Welcome_Screen;
Procedure Show_Exit_Screen;
Function Menu : Byte;
Procedure Play_Game;



Implementation

Uses
  Dos, Crt, JackUtil, JackGrph, JackTCV;

Procedure Show_Credits;
Const
  T : Array[0..10]  of String =
    ('Dieses Spiel wurde',
     'programmiert von',
     'Matthias Voigt',
     '',
     '(C)opyright 2003',
     '',
     'Informatik 13/II',
     'Saengerstadt-Gymnasium',
     'Fachleher Bernd Schmidt',
     '',
     '');

var B,  I, J : Word;
    C : LongInt;
Begin
  Clear_Screen;
  Draw_Screen;
  Set_PaletteBW;
  C := 220;
  Repeat
  Repeat
    Dec(C,2);
    Clear_Screen;
    For I:= 0 to 10 do
      Text_Out(160 - (Length(T[I])*8) div 2, I*10+C,255, T[I]);
    For J:=0 to 199 do
     For I:= 0 to 319 do
     Begin
       B := GetPixel(I, J);
       If B = 255 then
       PutPixel(I, J, B - (255-(j*256)div 199));
     End;
    Draw_Screen;
    Delay(30);
  Until Keypressed;
  Until Ord(ReadKey)<>0;
  Set_Palette;
End;



Procedure Show_Options;
Begin
  Clear_Screen;
  Text_Out(0, 40, 15, 'Leider kann man nichts aendern');
  Text_Out(0, 90, 15, 'Taste druecken um zum Menue zu kommen');
  Draw_Screen;
  ReadKey;
End;


Procedure Show_Help;
Begin
  Clear_Screen;
  Text_Out(0, 40, 15, 'Jack wird mit den Pfeiltasten gesteuert');
  Text_Out(0, 50, 15, 'Mit Leertaste kann man schiessen');
  Text_Out(0, 60, 15, 'Aber Vorsicht! Das gibt weniger Punkte');
  Text_Out(0, 70, 15, 'Viel Spass beim spielen!');
  Text_Out(0, 90, 15, 'Taste druecken um zum Menue zu kommen');
  Draw_Screen;
  ReadKey;
End;


Procedure Show_HighScore;
Var
  Place : TPlace;
  High_File : THigh_File;
  I : Integer;
Begin
  Assign(High_File, HIGHSCORE_FILE);
  Reset(High_File);
  I := 1;
  Clear_Screen;
    Text_Out(10, 30, 145,
             'Nr.          Name   Punkte      Datum');
    Text_Out(10, 40, 145,
            '====================================');
  While not EOF(High_File) do
  Begin
    Read(High_File, Place);
    Text_Out(10, I * 10 + 40, 145,
```

```
            LZ(IntToStr(I), 2)+' '+LS(Place.Name, 14)+' '+
            LZ(IntToStr(Place.Points), 8)+' '+
              LZ(IntToStr(Place.Day),2)+'.'+LZ(IntToStr(Place.Month),2)+'.'+
              LZ(IntToStr(Place.Year),2));
    Inc(I);
    Text_Out(10, 160, 140, 'Zum loeschen ''L'' druecken');
    Draw_Screen;
  End;
  Close(High_File);
  If ReadKey = 'l' then Del_Highscore;
End;


Procedure Show_Welcome_Screen;
Var
  I : Integer;
Begin
  I := 96;
  Repeat
    Clear_Screen;
    Text_Out(20, 40, I-32, 'Willkommen bei');
    Text_Out(40, 80, I, '''Jack The Adventureman''');
    Text_Out(30, 150, I div 112 * 15, 'created by Matthias Voigt');
    Draw_Screen;
    Delay(50);
    If I < 116 then Inc(I);
  Until Keypressed;
  ReadKey;
End;

Procedure Show_Exit_Screen;
Var
  I : Integer;
  Darken : Boolean;
Begin
  I := 110;
  Darken := True;
  Repeat
    Clear_Screen;
    if Darken then Dec(I) else Inc(I);
    If (I = 100) or (I = 120) then Darken := not Darken;
    Text_Out(30, 90, I, 'Viel Spass beim naechsten Mal');
    Draw_Screen;
    Delay(50);
  Until KeyPressed;
  ReadKey;
End;

Function Menu : Byte;
Var
  Select, I, J : Byte;
  Done : Boolean;
  ActSel : String;
  ActCnt : Byte;
  SelCol : Byte;
  Col : Byte;
Const
  AllSel : Array[1..9] of String =
          ('Spiel Starten',
           'Highscore-Liste zeigen',
           'Optionen',
           'Hilfe anzeigen',
           'Ping Pong Clone Spielen',
           'Paintbrush',
           'Welt-Editor',
           'Credits zeigen',
           'Spiel Beenden');
Begin
  Select := 1;
  ActCnt := 0;
  Col := 105;
  Done := False;
  Repeat
    Inc(ActCnt);
    Fill_Screen(abs((ActCnt div 8) mod 32 - 16)+234);
    If Keypressed then
    Case Ord(Readkey) of
      72 : Begin Dec(Select); If Select = 0 then Select := 9; end;
      80 : Begin Inc(Select); If Select = 10 then Select := 1; end;
      13 : Done := True;
    End;
    For I := 1 to 9 do
    Begin
      ActSel := AllSel[I];
      SelCol := col;
      For J := 1 to Length(ActSel) do
      Begin
        SelCol := abs(((ActCnt + J + I*I) mod 32) - 16) + 110;
        If I = Select then Inc(SelCol, 32);
        Text_Out(160-(Length(ActSel)*8) div 2 + 8 * (J-1)-8, I * 14 + 50, SelCol, ActSel[J]);
      End;
    End;
    Draw_Screen;
    Delay(30);
  Until Done;

  Menu := Select;
End;


Procedure Play_Game;
Var
  Player            : TPlayer;
  Shot                 : TShoot;
  Piece                : TPiece;
```

```
  Count              : LongInt;
  Stop_Game          : Boolean;
  OldBKHandler       : Pointer;
  ActLevel : Byte;
  FileLvl, FileItm, FileEnm   : String;
  AllUp        : Boolean;
  I : Byte;
  Name : String;
Begin
  Init_Player(Player, Shot);
  ActLevel := 1;
  Level_Name(FileLvl, FileItm, FileEnm, ActLevel);
  Load_World(FileLvl);
  Load_Items(FileItm);
  Load_Enemy(FileEnm);

  GetIntVec(9, OldBKHandler);
  SetIntVec(9, @Check_KeyB);

  Stop_Game := False;
  Count := 0;
  Repeat
    Inc(Count);
    Inc(Player.Time);
    Move_Player(Player, Shot, Count, Stop_Game);
    Move_Enemies(Player);

    Move_Shot(Shot, Player);

    Check_Items(Player);
    Check_Enemy(Player, Shot);


    Draw_World(Player, Shot);

    Draw_Screen;
    Delay(10);
    If (Player.LevelDone) and (ActLevel = 1) then Stop_Game := True;
  Until Stop_Game or (Player.Life <= 0);
  Repeat
    AllUp := True;
    For I := 1 to 128 do If KeyB_Table[I] then AllUp := False;
  Until AllUp = True;
  SetIntVec(9, OldBkHandler);

  Clear_Screen;
  Text_Out(10, 100, 15, 'Du hast '+IntToStr(Player.Score)+' Punkte erreicht!');
  Draw_Screen;
  ReadKEy;

  Name := Get_Name;
  Add_To_HighScore(Name, Player.Score);
End;


End.
```

```
Unit JackUtil;

Interface

Uses
  JackTCV;


Var
  KeyB_Table : Array[1..128] of Boolean;
Function IntToStr(I : LongInt) : String;

Function File_Exist(Var S : String) : Boolean;


Function  Pow2(Exponent : Byte) : Byte;


Function LZ(S : String; N : Byte) : String;
Function LS(S : String; N : Byte) : String;



Function P2X(P : TPlayer) : LongInt;
Function P2Y(P : TPlayer) : LongInt;

Function P2XL(P : TPlayer) : LongInt;
Function P2XR(P : TPlayer) : LongInt;
Function P2XLB(P : TPlayer) : LongInt;
Function P2XRB(P : TPlayer) : LongInt;
Function P2YD(P : TPlayer) : LongInt;
Function P2YU(P : TPlayer) : LongInt;

Procedure Draw_Player(X, Y : Integer; Var P : TPlayer);
Procedure Draw_Piece(X, Y : Integer; Sp : TSprite);
Procedure Draw_World(P : TPlayer; S : TShoot);

Procedure Add_To_HighScore(Name : String; P : LongInt);
Procedure Del_HighScore;

Procedure Move_Player(Var P : TPlayer; Var S : TShoot; C : LongInt; Var Quit : Boolean);

Function Get_Number : Word;
Function Get_Name : String;

Procedure Create_Shoot(P : TPlayer; Var S : TShoot);
Procedure Move_Shot(Var S : TShoot; P : TPlayer);

Procedure Move_Enemies(P : TPlayer);

Procedure Enemy_Editor;
Procedure Item_Editor;
Procedure Sprite_Editor;
Procedure Player_Editor;
Procedure World_Editor;
Procedure Editors;


Procedure Text_out(X, Y : Word; Color : Byte; Text : String);

Procedure Init_Player(Var Player : TPlayer; Var Shot : TShoot);

Procedure FLoad_Items;
Procedure FLoad_Pieces;
Procedure FLoad_Enemies;

Procedure Check_KeyB; Interrupt;

Procedure Save_World(Name : String);
Procedure Load_World(Name : String);
Procedure Save_Items(Name : String);
Procedure Load_Items(Name : String);
Procedure Load_Enemy(Name : String);


Procedure Check_Items(var P : TPlayer);
Procedure Check_Enemy(var P : TPlayer; Var S : TShoot);


Procedure Show_Enemy(Var P : TEnemy; I : Byte);
Procedure Get_Enemy(Var P : TEnemy; X : Word);
Procedure Set_Enemy(PNr : TEnemy_Pos; X : Word);

Procedure Show_Item(Var P :TItem; I : Byte);
Procedure Get_Item(Var P : TItem; X, Y : Word);
Procedure Set_Item(PNr : Byte; X, Y : Word);

Procedure Show_Piece(Var P :TPiece; I : Byte);
Procedure Get_Piece(Var P : TPiece; X, Y : Word);
Procedure Set_Piece(PNr : Byte; X, Y : Word);

Procedure Init_Util;
Procedure Exit_Util;


Implementation
Uses
  Dos, Crt, JackGrph, JackPing, JackScrs;

Type
  TFont = Array[1..8] of Byte;

Var
  Pieces    : PPieces;
  AllItems  : PAllItems;
```

```
  AllEnemy  : PEnemy_Kind;

  WorldBuf  : PLevel;
  ItemBuf   : PItems;
  EnemyBuf  : PEnemies;

  PlFile    : TPlayer_Sprite_File;

  Fonts     : Array[0..127] of TFont;

Function IntToStr(I : LongInt) : String;
Var
  S : string;
Begin
  Str(I, S);
  IntToStr := S;
End;
Procedure Check_KeyB;
Var
  Key : Byte;
Begin
  Key := Port[$60];
  If Key < 128 then
    KeyB_Table[Key] := True else
      KeyB_Table[Key - 128] := False;
  Port[$20] := $20;
End;

Function Pow2(Exponent : Byte) : Byte;
Begin
  Pow2 := 1 shl Exponent;
End;

Function File_Exist(Var S : String) : Boolean;
Var
  F : File;
Begin
  Assign(F, S);
  {$I-}
  Reset(F);
  {$I+}
  If IOResult = 0 then File_Exist := True else File_Exist := False;
  Close(F);
End;


Procedure Draw_Letter(X, Y : Word; Color : Byte; Text : TFont);
Var
  I, J, B : Byte;
Begin
  For I := 1 to 8 do
  Begin
    B := Text[I];
    For J := 0 to 7 do
      If (B and Pow2(J)) = Pow2(J) then PutPixel(X + (8-J), Y - (8-I), Color);
  End;
End;

Procedure Text_Out(X, Y : Word; Color : Byte; Text : String);
Var
  I : Byte;
  S : TFont;
Begin
  For I := 1 to Length(Text) do
  Begin
    If Ord(Text[I]) > 31 then
    Begin
      S := Fonts[Ord(Text[I])];
      Draw_Letter(X + I*8, Y, Color, S);
    End;
  End;
End;


Function P2X(P : TPlayer) : LongInt;
Begin
  P2X := P.Pos.X SHR 16 div BLS_X;
End;

Function P2Y(P : TPlayer) : LongInt;
Begin
  P2Y := P.Pos.Y SHR 16 div BLS_Y + 1;
End;

Function P2XL(P : TPlayer) : LongInt;
Begin
  P.Pos.X := P.Pos.X - 7 SHL 16;
  P2XL := P.Pos.X SHR 16 div 10;
End;
Function P2XR(P : TPlayer) : LongInt;
Begin
  P.Pos.X := P.Pos.X + 6 SHL 16;
  P2XR := P.Pos.X SHR 16 div 10;
End;

Function P2XLB(P : TPlayer) : LongInt;
Begin
  P.Pos.X := P.Pos.X - 6 SHL 16;
  P2XLB := P.Pos.X SHR 16 div 10;
End;
Function P2XRB(P : TPlayer) : LongInt;
Begin
  P.Pos.X := P.Pos.X + 5 SHL 16;
  P2XRB := P.Pos.X SHR 16 div 10;
End;
```

```
Function P2YD(P : TPlayer) : LongInt;
Begin
  P.Pos.Y := P.Pos.Y + 1 SHL 16;
  P2YD := P.Pos.Y SHR 16 div 10 + 1;
End;
Function P2YU(P : TPlayer) : LongInt;
Begin
  P.Pos.Y := P.Pos.Y - 23 SHL 16;
  P2YU := P.Pos.Y SHR 16 div 10 + 1;
End;

Procedure Add_To_HighScore(Name : String; P : LongInt);
Var
  Place : TPlace;
  High_File : THigh_File;
  HScore : THighScore;
  B : Word;
  I, J : Integer;
Begin
  Assign(High_File, HIGHSCORE_FILE);
  Reset(High_File);
  I := 1;
  While not EOF(High_File) do
  Begin
    Read(High_File, Place);
    HScore[I] := Place;
    Inc(I);
  End;
  Place.Name := Name;
  Place.Points := P;
  GetDate(Place.Year, Place.Month, Place.Day, B);
  HScore[10] := Place;

  For I := 1 to 9 do
   For J := I to 10 do
    If HScore[I].Points < HScore[J].Points then
    Begin
      Place := HScore[I];
      HScore[I] := HScore[J];
      HScore[J] := Place;
    End;
  Rewrite(High_File);
  For I := 1 to 10 do
    Write(High_File, HScore[I]);
  Close(High_File);
  Show_HighScore;
End;

Procedure Del_HighScore;
Var
  Place : TPlace;
  High_File : THigh_File;
  B : Word;
  I, J : Integer;
Begin
  Assign(High_File, HIGHSCORE_FILE);
  Reset(High_File);
  Place.Name := 'Jack';
  Place.Points := 0;
  GetDate(Place.Year, Place.Month, Place.Day, B);

  Rewrite(High_File);
  For I := 1 to 10 do
    Write(High_File, Place);
  Close(High_File);
  Show_HighScore;
End;



Procedure Draw_Player(X, Y : Integer; Var P : TPlayer);
Var
  I, J : Integer;
Begin
  For J := 1 to PL_Y do
    For I := 1 to PL_X do
      If P.Dir = Right then
        PutPixel(X + I - 7, Y + 1 - J, P.Sprite[P.ActSprite, I,J])
      else
        PutPixel(X - I + 7, Y + 1 - J, P.Sprite[P.ActSprite, I,J])

End;
Procedure Put_Rect(X, Y, R : Integer; Color : Byte);
Var
  I, J : Integer;
Begin
  For I := X - R to X + R do
   For J := Y - R to Y + R do
    PutPixel(I, J, Color);
End;


Procedure Put_Rect2(X, Y, R : Integer; Color : Byte);
Var
  I, J : Integer;
Begin
  For I := X - R + 1 to X + R do
   For J := Y - R to Y + R-1 do
    PutPixel(I, J, Color);
End;

Procedure Enemy_Editor;
Var
```

```
  Piece, T : TEnemy;
  Ch     : Char;
  I, J : Integer;
  Col : Byte;
  X, Y : Byte;
  Auswahl : Boolean;
  Select : Byte;
  Pfile : TEnemy_File;
Begin
  Assign(PFile, ENEMY_FILE);
  Reset(PFile);

  Auswahl := true;

  Select := 1;
  Seek(PFile, Select);
  Read(PFile, Piece);
  X := 1;
  Y := 1;
  Col := 0;
  Repeat
    Clear_Screen;
    If Auswahl then Put_Rect((Col mod 64)*4+20, (Col div 64) * 5+5, 2, 15);
    Put_Rect(50, 150, 10, Col);
    Text_Out(30, 170, 15, LZ(IntToStr(Col), 3));
    For I := 0 to 63 do
    Begin
      Put_Rect(I*4+20, 5, 1, I);
      Put_Rect(I*4+20, 10, 1, I+64);
      Put_Rect(I*4+20, 15, 1, I+128);
      Put_Rect(I*4+20, 20, 1, I+192);
    End;

    Put_Rect2(88, 35, 6, 15);
    For I := -4 to 4 do Begin
      Show_Enemy(T, (Select+16+I) mod 16);
      Draw_Piece(84+I*12, 40, T.Sprite);
    End;
    Text_Out(28, 50, 15, '<- Y '+LZ(IntToStr(Select), 3)+' C ->');

    Draw_Piece(50, 100, Piece.Sprite);
    Text_Out(10, 60, 15, '(P)unkte    : '+IntToStr(Piece.Points));
    If not Auswahl then Put_Rect(X*6+100, -Y*6+180, 3, 15);
    For I := 1 to 10 do
     for J := 1 to 10 do
        Put_Rect(I*6+100, -J*6+180, 2, Piece.Sprite[I, J]);
    Draw_Screen;
    CH := ReadKey;
    Case Ord(CH) of
      75 : If Auswahl then Dec(Col) else If X > 1 then Dec(X);
      77 : If Auswahl then Inc(Col) else If X < 10 then Inc(X);
      72 : If Auswahl then Dec(Col, 64) else If Y < 10 then Inc(Y);
      80 : If Auswahl then Inc(Col, 64) else If Y > 1 then Dec(Y);
      ord('c') : Begin
                   Inc(Select);
                   If Select = 16 then Select := 1;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('y') : Begin
                   Dec(Select);
                   If Select = 0 then Select := 15;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('p') : Piece.Points := Get_Number;
      32 : Auswahl := not Auswahl;
      13 : If not Auswahl then Piece.Sprite[X, Y] := Col else
             Auswahl := not Auswahl;
    End;
    Seek(Pfile, Select);
    Write(PFile, Piece);
    FLoad_Enemies;
  Until Ord(Ch) = 27;
  Close(PFile);
End;

Procedure Item_Editor;
Var
  Piece, T : TItem;
  Ch     : Char;
  I, J : Integer;
  Col : Byte;
  X, Y : Byte;
  Auswahl : Boolean;
  Select : Byte;
  Pfile : TAllItem_File;
Begin
  Assign(PFile, ITEM_FILE);
  Reset(PFile);

  Auswahl := true;

  Select := 1;
  Seek(PFile, Select);
  Read(PFile, Piece);
  X := 1;
  Y := 1;
  Col := 0;
  Repeat
    Clear_Screen;
    If Auswahl then Put_Rect((Col mod 64)*4+20, (Col div 64) * 5+5, 2, 15);
    Put_Rect(50, 150, 10, Col);
    Text_Out(30, 170, 15, LZ(IntToStr(Col), 3));
    For I := 0 to 63 do
```

```
    Begin
      Put_Rect(I*4+20, 5, 1, I);
      Put_Rect(I*4+20, 10, 1, I+64);
      Put_Rect(I*4+20, 15, 1, I+128);
      Put_Rect(I*4+20, 20, 1, I+192);
    End;

    Put_Rect2(88, 35, 6, 15);
    For I := -4 to 4 do Begin
      Show_Item(T, Select+I);
      Draw_Piece(84+I*12, 40, T.Sprite);
    End;
    Text_Out(28, 50, 15, '<- Y '+LZ(IntToStr(Select), 3)+' C ->');

    Draw_Piece(50, 100, Piece.Sprite);
    Text_Out(10, 60, 15, '(A)ufnehmbar: ');
    If Piece.Collect then Text_Out(122, 60, 15, 'J') else
      Text_Out(122, 60, 15, 'N');
    Text_Out(10, 70, 15, '(B)enutzbar : ');
    If Piece.Use then Text_Out(122, 70, 15, 'J') else
      Text_Out(122, 70, 15, 'N');
    Text_Out(10, 80, 15, '(P)unkte    : '+IntToStr(Piece.Points));
    If not Auswahl then Put_Rect(X*6+100, -Y*6+180, 3, 15);
    For I := 1 to 10 do
     for J := 1 to 10 do
        Put_Rect(I*6+100, -J*6+180, 2, Piece.Sprite[I, J]);
    Draw_Screen;
    CH := ReadKey;
    Case Ord(CH) of
      75 : If Auswahl then Dec(Col) else If X > 1 then Dec(X);
      77 : If Auswahl then Inc(Col) else If X < 10 then Inc(X);
      72 : If Auswahl then Dec(Col, 64) else If Y < 10 then Inc(Y);
      80 : If Auswahl then Inc(Col, 64) else If Y > 1 then Dec(Y);
      ord('c') : Begin
                   Inc(Select);
                   If Select = 0 then Select := 1;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('y') : Begin
                   Dec(Select);
                   If Select = 0 then Select := 255;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('a') : Piece.Collect := not Piece.Collect;
      ord('b') : Piece.Use := not Piece.Use;
      ord('p') : Piece.Points := Get_Number;
      32 : Auswahl := not Auswahl;
      13 : If not Auswahl then Piece.Sprite[X, Y] := Col else
           Auswahl := not Auswahl;
    End;
    Seek(Pfile, Select);
    Write(PFile, Piece);
    FLoad_Items;
  Until Ord(Ch) = 27;
  Close(PFile);
End;

Procedure Sprite_Editor;
Var
  Piece, P : TPiece;
  Ch     : Char;
  I, J : Integer;
  Col : Byte;
  X, Y : Byte;
  Auswahl : Boolean;
  Select : Byte;
  Pfile : TPiece_File;
Begin
  Assign(PFile, PIECE_FILE);
  Reset(PFile);

  Auswahl := true;

  Select := 1;
  Seek(PFile, Select);
  Read(PFile, Piece);
  X := 1;
  Y := 1;
  Col := 0;
  Repeat
    Clear_Screen;
    If Auswahl then Put_Rect((Col mod 64)*4+20, (Col div 64) * 5+5, 2, 15);
    Put_Rect(50, 150, 10, Col);
    Text_Out(30, 170, 15, LZ(IntToStr(Col), 3));
    For I := 0 to 63 do
    Begin
      Put_Rect(I*4+20, 5, 1, I);
      Put_Rect(I*4+20, 10, 1, I+64);
      Put_Rect(I*4+20, 15, 1, I+128);
      Put_Rect(I*4+20, 20, 1, I+192);
    End;

    Put_Rect2(88, 35, 6, 15);
    For I := -4 to 4 do Begin
      Show_Piece(P, Select+I);
      Draw_Piece(84+I*12, 40, P.Sprite);
    End;
    Text_Out(28, 50, 15, '<- Y '+LZ(IntToStr(Select), 3)+' C ->');

    Draw_Piece(50, 100, Piece.Sprite);
    Text_Out(10, 60, 15, '(D)urchgehbar: ');
    If Piece.Walk then Text_Out(150, 60, 15, 'J') else
      Text_Out(150, 60, 15, 'N');
```

```
    If not Auswahl then Put_Rect(X*6+100, -Y*6+180, 3, 15);
    For I := 1 to 10 do
     for J := 1 to 10 do
        Put_Rect(I*6+100, -J*6+180, 2, Piece.Sprite[I, J]);
    Draw_Screen;
    CH := ReadKey;
    Case Ord(CH) of
      75 : If Auswahl then Dec(Col) else If X > 1 then Dec(X);
      77 : If Auswahl then Inc(Col) else If X < 10 then Inc(X);
      72 : If Auswahl then Dec(Col, 64) else If Y < 10 then Inc(Y);
      80 : If Auswahl then Inc(Col, 64) else If Y > 1 then Dec(Y);
      ord('c') : Begin
                   Inc(Select);
                   If Select = 0 then Select := 1;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('y') : Begin
                   Dec(Select);
                   If Select = 0 then Select := 255;
                       Seek(PFile, Select);
                       Read(PFile, Piece);
                   End;
      ord('d') : Piece.Walk := not Piece.Walk;
      32 : Auswahl := not Auswahl;
      13 : If not Auswahl then Piece.Sprite[X, Y] := Col else
            Auswahl := not Auswahl;
    End;
    Seek(Pfile, Select);
    Write(PFile, Piece);
    FLoad_Pieces;
  Until Ord(Ch) = 27;
  Close(PFile);
End;

Procedure Player_Editor;
Var
  PlSprite : TPlayer_Sprite;
  Shot       : TShoot;
  Ch       : Char;
  I, J     : Integer;
  Col      : Byte;
  X, Y     : Byte;
  Auswahl  : Boolean;
  Select   : Byte;
  Player   : TPlayer;
Begin
  Init_Player(Player, Shot);
  Auswahl := true;

  X := 1;
  Y := 1;
  Select := 0;
  Col := 0;
  Repeat
    Clear_Screen;
    If Auswahl then Put_Rect((Col mod 64)*4+20, (Col div 64) * 5+5, 2, 15);
    Put_Rect(50, 150, 10, Col);
    Text_Out(30, 170, 15, LZ(IntToStr(Col), 3));
    For I := 0 to 63 do
    Begin
      Put_Rect(I*4+20, 5, 1, I);
      Put_Rect(I*4+20, 10, 1, I+64);
      Put_Rect(I*4+20, 15, 1, I+128);
      Put_Rect(I*4+20, 20, 1, I+192);
    End;
    Put_Rect(Select * 31 + 20, 69, 13, 15);
    For I := 0 to 2 do
    Begin
      Player.ActSprite := I;
      Draw_Player(I*31 + 20, 80, Player);
    End;
    Text_Out(0, 90, 15, '<- Y | C ->');
    Player.ActSprite := Select;

    If not Auswahl then Put_Rect(X*6+100, -Y*6+180, 3, 15);
    For I := 1 to 12 do
     for J := 1 to 24 do
        Put_Rect(I*6+100, -J*6+180, 2, Player.Sprite[Select, I, J]);
    Draw_Screen;
    CH := ReadKey;
    Case Ord(CH) of
      75 : If Auswahl then Dec(Col) else If X > 1 then Dec(X);
      77 : If Auswahl then Inc(Col) else If X < 12 then Inc(X);
      72 : If Auswahl then Dec(Col, 64) else If Y < 24 then Inc(Y);
      80 : If Auswahl then Inc(Col, 64) else If Y > 1 then Dec(Y);
      ord('c') : Begin Inc(Select); If Select = 3 then Select := 0; end;
      ord('y') : Begin Dec(Select); If Select = 255 then Select := 2; end;
      32 : Auswahl := not Auswahl;
      13 : If not Auswahl then Player.Sprite[Select, X, Y] := Col else
            Auswahl := not Auswahl;
    End;
  Until Ord(Ch) = 27;
  Assign(PlFile, PLAYER_FILE);
  Rewrite(PlFile);
  For I := 0 to 2 do
  Begin
    Write(PlFile, Player.Sprite[I]);
  End;
  Close(PlFile);
End;


Procedure Draw_Piece(X, Y : Integer; Sp : TSprite);
Var
```

```
  I, J : Integer;
Begin
  For J := 1 to BLS_Y do
    For I := 1 to BLS_X do
      PutPixel(X + (I-1), Y - J, Sp[I,J]);
End;

Procedure World_Editor;
Var
  ch : Char;
  I, J : Integer;
  X, Y : Word;
  P : TPiece;
  T : TItem;
  Select, SelectT : Byte;
  FileLvl, FileItm, FileEnm : String;
Begin
  X := 20;
  Y := 10;
  Select := 0;
  SelectT := 0;
  Level_Name(FileLvl, FileItm, FileEnm, 1);
  Load_World(FileLvl);
  Load_Items(FileItm);
  Load_Enemy(FileEnm);
  Repeat
    Clear_Screen;
    For I := 1 to 16 do
     For J := 1 to 20 do
       Put_Rect2(I*11+4, J*10-5, 6, 212);

    If X < 8 then Put_Rect2(X * 11+4, Y * 10-5, 6, 15)
     else If X > 312 then Put_Rect2((X - 304) * 11+4, Y * 10-5, 6, 15)
      else Put_Rect2(8 * 11+4, Y * 10-5, 6, 15);
    For I := 1 to 16 do
     For J := 1 to 20 do
     Begin
       If X < 8 then Get_Piece(P, I, J)
        else If X > 312 then Get_Piece(P, 304 + I, J)
         else Get_Piece(P, I + X - 8, J);
       Draw_Piece(I * 11, J * 10, P.Sprite);

       If X < 8 then Get_Item(T, I, J)
        else If X > 312 then Get_Item(T, 304 + I, J)
         else Get_Item(T, I + X - 8, J);
       Draw_Piece(I * 11, J * 10, T.Sprite);
     End;

    Put_Rect2(258, 15, 6, 15);
    For I := -4 to 4 do Begin
      Show_Piece(P, Select+I);
      Draw_Piece(254+I*12, 20, P.Sprite);
    End;
    Text_Out(198, 30, 15, '<- A '+LZ(IntToStr(Select), 3)+' D ->');

    Put_Rect2(258, 45, 6, 15);
    For I := -4 to 4 do Begin
      Show_Item(T, SelectT+I);
      Draw_Piece(254+I*12, 50, T.Sprite);
    End;
    Text_Out(198, 60, 15, '<- Y '+LZ(IntToStr(SelectT), 3)+' C ->');

    Text_Out(220, 80, 15, 'X: '+IntToStr(X));
    Text_Out(220, 90, 15, 'Y: '+IntToStr(Y));

    Draw_Screen;
    Ch := Readkey;
    Case Ord(CH) of
      75 : If X > 1 then Dec(X);
      77 : If X < 320 then Inc(X);
      80 : If Y < 20 then Inc(Y);
      72 : If Y > 1 then Dec(Y);
      Ord('a') : Dec(Select);
      Ord('d') : Inc(Select);
      Ord('y') : Dec(SelectT);
      Ord('c') : Inc(SelectT);
      ord('s') : Set_Piece(Select, X, Y);
      ord('x') : Set_Item(SelectT, X, Y);
    End;
  Until Ord(Ch) = 27;
  Save_World(FileLvl);
  Save_Items(FileItm);
End;


Function LZ(S : String; N : Byte) : String;
Var
  B : Byte;
  T : String;
Begin
  T := S;
  For B := Length(S) to N-1 do
   T := '0' + T;
  LZ := T;
End;

Function LS(S : String; N : Byte) : String;
Var
  B : Byte;
  T : String;
Begin
  T := S;
  For B := Length(S) to N-1 do
   T := ' ' + T;
  LS := T;
```

```
End;

Procedure Editors;
Var
  Select, ActSel : Byte;
Begin
  ActSel := 1;
  Select := 1;
  Repeat
    Clear_Screen;
    If ActSel = 1 then Text_out(90, 80, 144, 'Sprite Editor') else
      Text_out(90, 80, 115, 'Sprite Editor');
    If ActSel = 2 then Text_out(90, 100, 144, 'Player Editor') else
      Text_out(90, 100, 115, 'Player Editor');
    If ActSel = 3 then Text_out(90, 120, 144, 'Item Editor') else
      Text_out(90, 120, 115, 'Item Editor');
    If ActSel = 4 then Text_out(90, 140, 144, 'Enemy Editor') else
      Text_out(90, 140, 115, 'Enemy Editor');
    If ActSel = 5 then Text_out(90, 160, 144, 'Hauptmenue') else
      Text_out(90, 160, 115, 'Hauptmenue');
    Draw_Screen;
    Case Ord(ReadKey) of
      72 : Begin Dec(ActSel); If ActSel = 0 then ActSel := 5; End;
      80 : Begin Inc(ActSel); If ActSel = 6 then ActSel := 1; End;
      13 : Case ActSel of
             1 : Sprite_Editor;
               2 : Player_Editor;
               3 : Item_Editor;
             4 : Enemy_Editor;
               5 : Select := 0;
           End;
    End;
  Until Select = 0;
End;
Procedure Draw_Level(P : TPlayer);
Var
  I, J   : Integer;
  Piece    : TPiece;
Begin
  For I := P2X(P) - 16 to P2X(P) + 16 do
    For J := 1 to LVL_Y do
      Draw_Piece(I * 10 - P.Pos.X SHR 16 + 160, J * 10,
                 Pieces^[WorldBuf^[I, J]].Sprite);
End;

Procedure Draw_Items(P : TPlayer);
Var
  I, J   : Integer;
  Item   : TItem;
Begin
  For I := P2X(P) - 16 to P2X(P) + 16 do
    For J := 1 to LVL_Y do
      If ItemBuf^[I, J] <> 0 then
      Draw_Piece(I * 10 - P.Pos.X SHR 16 + 160, J * 10,
                 AllItems^[ItemBuf^[I, J]].Sprite);
End;
Procedure Draw_Enemies(P : TPlayer);
Var
  I : Integer;
Begin
  For I := 1 to 100 do
  Begin
    If (EnemyBuf^[I].X > P.Pos.X SHR 16 - 180) And (EnemyBuf^[I].X < P.Pos.X SHR 16 + 180) then
      Draw_Piece(EnemyBuf^[I].X - P.Pos.X SHR 16 + 160, EnemyBuf^[I].Y,
                 AllEnemy^[EnemyBuf^[I].Kind].Sprite);
  End;
End;

Procedure Draw_World(P : TPlayer; S : TShoot);
Var
  I : Integer;
Begin
  Fill_Screen(212);
  Draw_Level(P);
  Draw_Items(P);
  Draw_Enemies(P);
  Draw_Player(160, P.Pos.Y SHR 16, P);
  If S.Active then
    Draw_Piece(S.Pos.X SHR 16 - P.Pos.X SHR 16 + 160, S.Pos.Y SHR 16, AllItems^[254].Sprite);
  Text_Out(2, 8, 128, 'Punkte: '+IntToStr(P.Score));
  For I := P.Life downto 1 do Draw_Piece(320 - I *12, 10, AllItems^[255].Sprite);
End;

Procedure Move_Shot(Var S : TShoot; P : TPlayer);
Var
  XP : Word;
Begin
  If S.Active then
  Begin
    If S.Dir = Left then
     Dec(S.Pos.X, 5 SHL 16) else
     Inc(S.Pos.X, 5 SHL 16);
     If Abs(S.Pos.X SHR 16 - P.Pos.X SHR 16) > 180 then
        S.Active := False;

    If S.Dir = Right then
      XP := (S.Pos.X SHR 16 + 9) div BLS_X else
      XP := (S.Pos.X SHR 16 + 1) div BLS_X;

    If (not Pieces^[WorldBuf^[XP, (S.Pos.Y SHR 16 + 9) div BLS_Y]].Walk) or
       (not Pieces^[WorldBuf^[XP, (S.Pos.Y SHR 16 + 1) div BLS_Y]].Walk) then
       S.Active := False;
  End;
End;
```

```
Procedure Move_Enemies(P : TPlayer);
Var
  I : Integer;
Begin
  For I := 1 to 100 do
    If EnemyBuf^[I].Kind <> 0 then
    If (EnemyBuf^[I].X > P.Pos.X SHR 16 - 240) And
       (EnemyBuf^[I].X < P.Pos.X SHR 16 + 240) then
     with enemyBuf^[I] do
  begin
    If Life <= 0 then Kind := 0;
    Inc(Tag);
    Case Kind of
      1 : Begin
            X := OX;
            Y := OY - Abs(Tag mod 80 - 40);
          End;
      2 : Begin
            X := OX - Abs(Tag mod 160 - 80) + 40;
            Y := OY;
          End;
      3 : Begin
            X := OX;
            Y := OY;
          End;
    End;
  End;
End;


Procedure Init_Player(Var Player : TPlayer; Var Shot : TShoot);
Var
  X, Y : Integer;
  B : Byte;
Begin
  Reset(PlFile);
  With Player do
  Begin
    B := 0;
    While not EOF(PlFile) do
    Begin
      Read(PlFile, Sprite[B]);
      Inc(B);
    End;
    ActSprite := 0;
    Life     := START_LIFE;
    Pos.X    := START_X SHL 16;
    Pos.Y     := START_Y SHL 16;
    Speed.X := 0;
    Speed.Y := 0;
    Dir      := Right;
    Move     := False;
    Action   := Fall;
    Score    := 0;
    Time     := 0;
    Shot.Active := False;
    LevelDone := False;
  End;
  Close(PlFile);
End;


Procedure Check_Items(var P : TPlayer);
Var
  T : TItem;
  I : Byte;
  X, Y : Word;
Begin
  For I:= 1 to 8 do
  Begin
    Case I of
      1 : Begin X := P2XLB(P); Y := P2Y(P); End;
      2 : Begin X := P2XLB(P); Y := P2Y(P)-1; End;
      3 : Begin X := P2XLB(P); Y := P2Y(P)-2; End;
      4 : Begin X := P2XLB(P); Y := P2YU(P); End;
      5 : Begin X := P2XRB(P); Y := P2Y(P); End;
      6 : Begin X := P2XRB(P); Y := P2Y(P)-1; End;
      7 : Begin X := P2XRB(P); Y := P2Y(P)-2; End;
      8 : Begin X := P2XRB(P); Y := P2YU(P); End;
    End;
    If ItemBuf^[X, Y] <> 0 then
    Begin
      T := AllItems^[ItemBuf^[X, Y]];
      If T.Collect then
      Begin
        Inc(P.Score, T.Points);
        If ItemBuf^[X, Y] = 255 then Inc(P.Life);
        ItemBuf^[X, Y] := 0;
      End;
      If T.Use then
      Begin
        If (ItemBuf^[X, Y] = 2) then
          Begin
            If Start_Ping_Pong then
            Begin
             Inc(P.Life);
             Inc(P.Score, AllItems^[ItemBuf^[X, Y]].Points);
            End;
          ItemBuf^[X, Y] := ItemBuf^[X, Y] + 1;
        End else
        If ItemBuf^[X, Y] = 253 then
        Begin
          Inc(P.Score, AllItems^[ItemBuf^[X, Y]].Points);
          P.LevelDone := True;
```

```
       End;
      End;
    End;
  End;
End;

Function Get_Number : Word;
Var
  S : String;
  N : WorD;
  ch : Char;
  code : Integer;
Begin
  S := '';
  Repeat
    Clear_Screen;
    Text_Out(10, 10, 15, 'Bitte neuen Wert eingeben: ');
    Text_Out(240, 10, 15, S);
    Draw_Screen;
    ch := ReadKey;
    case ch of
      '1', '2', '3', '4', '5', '6', '7', '8', '9', '0' : s := s + ch;
    end;
  Until ord(ch) = 13;
  Val(S, N, Code);
  Get_Number := N;
End;
Function Get_Name : String;
Var
  S : String;
  ch : Char;
Begin
  S := '';
  Repeat
    Clear_Screen;
    Text_Out(10, 10, 15, 'Bitte Namen eingeben: ');
    Text_Out(200, 10, 15, S);
    Draw_Screen;
    ch := ReadKey;
    case ord(ch) of
      8 : If ord(S[0]) > 0 then S[0] := Chr(Ord(S[0]) - 1);
      13 : S := S;
      27 : S := '-UNNAMED-';
    else
      s := s + ch;
    End;
  Until (ord(ch) = 13) or (ord(ch) = 27);
  Get_Name := S;
End;
Procedure Create_Shoot(P : TPlayer; Var S : TShoot);
Begin
  If not S. Active then
  Begin
    S.Pos.X := P.Pos.X - 5;
    S.Pos.Y := P.Pos.Y - 12;
    S.Active := True;
    S.Dir := P.Dir;
  End;
End;

Procedure Check_Enemy(var P : TPlayer; Var S : TShoot);
Var
  T : TItem;
  I : Byte;
  X, Y : Word;
Begin
  For I := 1 to 100 do
    If EnemyBuf^[I].Kind <> 0 then
    Begin
    If ((EnemyBuf^[I].X > P.Pos.X SHR 16 - 16) And
       (EnemyBuf^[I].X < P.Pos.X SHR 16 + 6) And
       (EnemyBuf^[I].Y > P.Pos.Y SHR 16 - 20) And
       (EnemyBuf^[I].Y < P.Pos.Y SHR 16 + 15)) then
     with enemyBuf^[I] do
     Begin
       If (EnemyBuf^[I].Y < P.Pos.Y SHR 16 + 5) then Dec(P.Life) else
         P.Speed.Y := -10 SHL 16;
       Dec(Life, 20);
       Inc(P.Score, AllEnemy^[EnemyBuf^[I].Kind].Points);
     End;

     If S.Active and
       ((EnemyBuf^[I].X > S.Pos.X SHR 16 - 9) And
       (EnemyBuf^[I].X < S.Pos.X SHR 16 + 9) And
       (EnemyBuf^[I].Y > S.Pos.Y SHR 16 - 9) And
       (EnemyBuf^[I].Y < S.Pos.Y SHR 16 + 9)) Then
     with EnemyBuf^[I] do
     Begin
       Dec(Life, 20);
       S.Active := False;
       Inc(P.Score, AllEnemy^[EnemyBuf^[I].Kind].Points div 2);
     End;
    End;
End;

Procedure Save_World(Name : String);
Var
  I, J : Integer;
  B    : Byte;
  WC   : LongInt;
  W    : TLevel_File;
Begin
  Assign(W, Name);
  Rewrite(W);
  For  J := 1 to 20 do
```

```
   For I := 1 to 320 do
   Begin
    B := WorldBuf^[I, J];
    Write(W, B);
  end;
  Close(W);
End;


Procedure Load_World(Name : String);
Var
  I, J : Integer;
  B    : Byte;
  WC   : LongInt;
  W    : TLevel_File;
Begin
  Assign(W, Name);
  Reset(W);
  I := 1;
  J := 1;
  While not EOF(W) do Begin
    Read(W, B);
    WorldBuf^[I, J] := B;
    Inc(I);
    If I > 320 then Begin
      I := 1;
      Inc(J);
    End;
  end;
  Close(W);
End;

Procedure Save_Items(Name : String);
Var
  I, J : Integer;
  B    : Byte;
  WC   : LongInt;
  W    : TItem_File;
Begin
  Assign(W, Name);
  Rewrite(W);
  For  J := 1 to 20 do
   For I := 1 to 320 do
   Begin
    B := ItemBuf^[I, J];
    Write(W, B);
  end;
  Close(W);
End;

Procedure Load_Items(Name : String);
Var
  I, J : Integer;
  B    : Byte;
  WC   : LongInt;
  T    : TItem_File;
Begin
  Assign(T, Name);
  Reset(T);
  I := 1;
  J := 1;
  While not EOF(T) do Begin
    Read(T, B);
    ItemBuf^[I, J] := B;
    Inc(I);
    If I > 320 then Begin
      I := 1;
      Inc(J);
    End;
  end;
  Close(T);
End;

Procedure Load_Enemy(Name : String);
Var
  I, J : Integer;
  B    : TEnemy_Pos;
  T    : TEnemies_File;
Begin
  Assign(T, Name);
  Reset(T);
  I := 1;
  While not EOF(T) do Begin
    Read(T, B);
    EnemyBuf^[I] := B;
    Inc(I);
  end;
  Close(T);
End;

Procedure Show_Enemy(Var P : TEnemy; I : Byte);
Begin
  P := AllEnemy^[I];
End;

Procedure Get_Enemy(Var P : TEnemy; X : Word);
Begin
  P := AllEnemy^[EnemyBuf^[X].Kind];
End;

Procedure Set_Enemy(PNr : TEnemy_Pos; X : Word);
Begin
  EnemyBuf^[X] := PNr;
End;
```

```
Procedure Show_Item(Var P : TItem; I : Byte);
Begin
  P := AllItems^[I];
End;

Procedure Get_Item(Var P : TItem; X, Y : Word);
Begin
  P := AllItems^[ItemBuf^[X][Y]];
End;

Procedure Set_Item(PNr : Byte; X, Y : Word);
Begin
  ItemBuf^[X][Y] := PNr;
End;

Procedure Show_Piece(Var P :TPiece; I : Byte);
Begin
  P := Pieces^[I];
End;
Procedure Get_Piece(Var P : TPiece; X, Y : Word);
Begin
  If (Y <= 0) or (Y >= 21) then P := Pieces^[0] else
  P := Pieces^[WorldBuf^[X][Y]];
End;

Procedure Set_Piece(PNr : Byte; X, Y : Word);
Begin
  WorldBuf^[X][Y] := PNr;
End;

Procedure FLoad_Fonts;
Var
  FFonts : File of TFont;
  I : Byte;
  S : TFont;
Begin
  Assign(FFonts, FONTS_FILE);
  Reset(FFonts);
  I := 0;
  While not EOF(FFonts) do
  Begin
    Read(FFonts, S);
    Fonts[I] := S;
    Inc(I);
  End;
  Close(FFonts);
End;


Procedure FLoad_Pieces;
Var
  P : TPiece;
  I : Integer;
  PFile : TPiece_File;
Begin
  Assign(PFile, PIECE_FILE);
  Reset(PFile);
  I := 0;
  While not EOF(PFile) do Begin
    Read(PFile, P);
    Pieces^[I] := P;
    Inc(I);
  end;
  Close(PFile);
End;


Procedure FLoad_Items;
Var
  P : TItem;
  I : Integer;
  PFile : TAllItem_File;
Begin
  Assign(PFile, ITEM_FILE);
  Reset(PFile);
  I := 0;
  While not EOF(PFile) do Begin
    Read(PFile, P);
    AllItems^[I] := P;
    Inc(I);
  end;
  Close(PFile);
End;

Procedure FLoad_Enemies;
Var
  P : TEnemy;
  I : Integer;
  PFile : TEnemy_File;
Begin
  Assign(PFile, ENEMY_FILE);
  Reset(PFile);
  I := 0;
  While not EOF(PFile) do Begin
    Read(PFile, P);
    AllEnemy^[I] := P;
    Inc(I);
  end;
  Close(PFile);
End;

Procedure Init_Util;
Var
  I : Integer;
Begin
```

```
  Graph_Init;
  Assign(PlFile, PLAYER_FILE);
  GetMem(Pieces, SizeOf(TPieces));
  GetMem(AllItems, SizeOf(TAllItems));
  GetMem(WorldBuf, SizeOf(TLevel));
  GetMem(ItemBuf, SizeOf(TItems));
  GetMem(AllEnemy, SizeOf(TEnemy_Kind));
  GetMem(EnemyBuf, SizeOf(TEnemies));
  FLoad_Fonts;
  FLoad_Pieces;
  FLoad_Items;
  FLoad_Enemies;
End;

Procedure Exit_Util;
Var
  I : Integer;
Begin
  FreeMem(Pieces, SizeOf(TPieces));
  FreeMem(AllItems, SizeOf(TAllItems));
  FreeMem(WorldBuf, SizeOf(TLevel));
  FreeMem(ItemBuf, SizeOf(TItems));
  FreeMem(AllEnemy, SizeOf(TEnemy_Kind));
  FreeMem(EnemyBuf, SizeOf(TEnemies));
  Graph_Exit;
End;

Procedure Move_Player(Var P : TPlayer; Var S : TShoot; C : LongInt; Var Quit : Boolean);
Var
  Piece1, Piece2, Piece3, Piece4 : TPiece;
  TP    : TPlayer;
  KB_Left, KB_Right, KB_Up, KB_Down : Boolean;
  Shoot : Boolean;
Begin
  KB_Left  := KeyB_Table[75];
  KB_Right := KeyB_Table[77];
  KB_Up    := KeyB_Table[72];
  KB_Down  := KeyB_Table[80];
  Shoot    := KeyB_Table[57];

  If (P.Time mod 5 = 0) and P.Move then Inc(P.ActSprite);
  If P.ActSprite = 3 then P.ActSprite := 0;

  If Shoot then Create_Shoot(P, S);

  If (KB_Left and KB_Right) or
   not (KB_Left or KB_Right) then P.Move := False else
    P.Move := True;

  If P2XLB(P) = P2XRB(P) then
  Begin
    Get_Piece(Piece1, P2X(P), P2YD(P));
    If Piece1.Walk then P.Action := Fall else P.Action := Stand;
  End else
  Begin
    Get_Piece(Piece1, P2XLB(P), P2YD(P));
    Get_Piece(Piece2, P2XRB(P), P2YD(P));
    If Piece1.Walk and Piece2.Walk then P.Action := Fall else P.Action := Stand;
  End;

  If not (P.Action = Fall) and KB_Up then
  Begin
    P.Action := Fall;
    P.Speed.Y := -10 SHL 16;
  End;

  If KB_Left then
  Begin
    P.Dir := Left;
    Get_Piece(Piece1, P2XL(P), P2Y(P));
    Get_Piece(Piece2, P2XL(P), P2Y(P)-1);
    Get_Piece(Piece3, P2XL(P), P2Y(P)-2);
    Get_Piece(Piece4, P2XL(P), P2YU(P));
    If Piece1.Walk and Piece2.Walk and Piece3.Walk and Piece4.Walk then
      P.Speed.X := 2 SHL 16 else P.Speed.X := 0;
  End;
  If KB_Right then
  Begin
    P.Dir := Right;
    Get_Piece(Piece1, P2XR(P), P2Y(P));
    Get_Piece(Piece2, P2XR(P), P2Y(P)-1);
    Get_Piece(Piece3, P2XR(P), P2Y(P)-2);
    Get_Piece(Piece4, P2XR(P), P2YU(P));
    If Piece1.Walk and Piece2.Walk and Piece3.Walk and Piece4.Walk then
      P.Speed.X := 2 SHL 16 else P.Speed.X := 0;
  End;
  If P.Action = Fall then
  Begin
    Inc(P.Speed.Y, Gravity);
    If P2XRB(P) = P2XLB(P) then
    Begin
        Get_Piece(Piece1, P2X(P), P2YU(P));
      If not Piece1.Walk then
      Begin
        P.Speed.Y := Abs(P.Speed.Y);
      End;
    End else
    Begin
      Get_Piece(Piece1, P2XLB(P), P2YU(P));
      Get_Piece(Piece2, P2XRB(P), P2YU(P));
      If not Piece1.Walk or not Piece2.Walk then
      Begin
        P.Speed.Y := Abs(P.Speed.Y);
      End;
    End;
```

```
  End else
    P.Speed.Y := 0;

  Quit := KeyB_Table[1];

  If P.Dir = Left then P.Speed.X := -P.Speed.X;
  If P.Move then P.Pos.X := P.Pos.X + P.Speed.X;

  P.Pos.Y := P.Pos.Y + P.Speed.Y;
  If P.Action = Fall then
  If P2XRB(P) = P2XLB(P) then
  Begin
    Get_Piece(Piece1, P2X(P), P2Y(P));
    If not Piece1.Walk then
    Begin
      P.Pos.Y := P.Pos.Y - P.Speed.Y;
      P.Pos.Y := (P2Y(P) * 10 - 1) SHL 16;
    End;
  End else
  Begin
    Get_Piece(Piece1, P2XLB(P), P2Y(P));
    Get_Piece(Piece2, P2XRB(P), P2Y(P));
    If not Piece1.Walk or not Piece2.Walk then
    Begin
      P.Pos.Y := P.Pos.Y - P.Speed.Y;
      P.Pos.Y := (P2Y(P) * 10 - 1) SHL 16;
    End;
  End;
  If P2Y(P) > 23 then Dec(P.Life);
End;

End.
```