# Delphi Vordiplomsprogramm

Thema: Space Invaders
„Fighter“

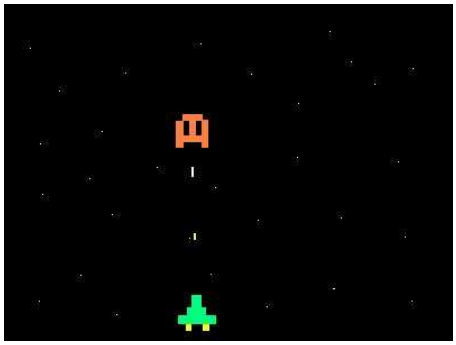


| | |
|---|---|
| Name: | Andreas Schibilla |
| E-Mail: | Andreas@Schibilla.de |
| Fachrichtung: | Technische Informatik |
| Matrikelnummer: | ii4900 |
| Fachsemester: | 3 |

# 1. Inhaltsverzeichnis

## 2. Allgemeine Problemstellung

Das zu erstellende Spiel besteht aus einem quadratischen Spielfeld auf dessen unteren Kante auf einer horizontalen Achse die Spielfigur des Spielers (ein Raumschiff) nach links und rechts bewegt werden kann. Darüber bewegen sich die gegnerischen Spielfiguren (Alien-Raumschiffe) in quasi-zufälligen (dazu später mehr) Flugbahnen. Die gegnerischen Spielfiguren geben in quasi-zufälligen Zeitabständen Schüsse vertikal nach unten ab, während der Spieler die Möglichkeit hat Schüsse vertikal nach oben abzuschießen. Trifft ein Schuss einer gegnerischen Spielfigur die Spielfigur des Spielers, dann wird dessen Spielfigur zerstört, er verliert eines von n (dazu später mehr) Leben. Wenn der Spieler noch mindestens ein Leben übrig hat, erscheint die Spielfigur wieder und das Spiel geht weiter. Wenn der Spieler kein Leben mehr übrig hat dann ist das Spiel beendet. Trifft ein Schuss des Spielers eine gegnerische Spielfigur, dann wird diese zerstört und der Spieler erhält eine bestimmte Anzahl von Punkten. Wenn ein gegnerisches Raumschiff auf seiner Flugbahn mit dem Raumschiff des Spielers kollidiert, so wird das Raumschiff des Gegners zerstört, ohne dass der Spieler dafür Punkte gut geschrieben kriegt, und das Raumschiff des Spielers wird zerstört und er verliert eines seiner n Leben, und wenn der Spieler noch ein Leben übrig hat, geht das Spiel weiter.



Um im Spiel den Eindruck von zusätzlicher Bewegung (zusätzlich zu den Bewegungen des Spielers und der Gegner) zu verleihen, bewegen sich im Hintergrund des Spielfeldes (d.h. hinter den Spielfiguren - d.h. von den Spielfiguren überdeckt) kleine weiße (annähernd) runde Objekte (Sterne) vom oberen Rand des Spielfeldes vertikal nach unten. Die Anordnung der in jedem Augenblick des Spieles sichtbaren 'Sterne' ist dabei quasi-zufällig, d.h. der Zufall (=random) unterliegt der Einschränkung, dass die Sterne gleichmäßig über den Hintergrund gestreut sein müssen und sich nicht nur in einem Bereich des Bildschirms konzentrieren dürfen. Die Anzahl der Sterne die in jedem Augenblick sichtbar sind, ist vom Benutzer einstellbar (dazu später mehr). Die Hintergrundfarbe des Spielfeldes ist natürlich immer schwarz!
Das Spiel ist in unterschiedliche Spielstufen eingeteilt. In jeder Spielstufe muss der Spieler alle sich auf dem Spielfeld befindenden gegnerischen Spielfiguren zerstören, bevor die nächste Spielstufe gestartet wird. Der Unterschied zwischen den Spielstufen besteht:

- im Aussehen der gegnerischen Spielfiguren (dazu später mehr)
- in der Anzahl der gegnerischen Spielfiguren und
- in der Geschwindigkeit der Bewegungen der gegnerischen Spielfiguren und
- in der Schussfrequenz der gegnerischen Spielfiguren.

Sowohl die Anzahl der gegnerischen Spielfiguren, ihre Geschwindigkeit, als auch ihre Schussfrequenz wird mit jeder neuen Spielstufe um jeweils einen bestimmten Anteil erhöht. Um eine Spielstufe zu beenden steht dem Spieler jeweils nur eine begrenzte Zeitspanne zur Verfügung, die vom Benutzer einstellbar ist. Für jede Spielstufe steht dem Spieler dieselbe Zeitspanne zur Verfügung (der 'Countdown' beginnt also mit jeder Spielstufe von neuem). Schafft der Spieler es nicht, alle Gegner einer Spielstufe in der vorgegebenen Zeit zu zerstören, dann verliert der Spieler eines seiner Leben und wenn er dann noch mindestens ein Leben übrig hat, beginnt die gleiche Spielstufe von neuem <- der Countdown startet von neuem und es sind wieder alle Gegner dieser Spielstufe vorhanden. Dies steht im Gegensatz zur Zerstörung einer Spielfigur durch einen Schuss eines Gegners oder durch eine Kollision mit einem Gegner während einer Spielstufe. In so einem Fall wird (wenn der Spieler noch mindestens ein Leben hat) die Spielstufe ohne Verzögerung fortgesetzt, d.h. mit der Anzahl der Gegner die direkt vor der Zerstörung des Spielers übrig waren und mit dem Countdownstand der unmittelbar vor der Zerstörung des Spielers aktuell war.

Während der kurzen Phase der Zerstörung des Spieler-Raumschiffes und dessen Neuerscheinen auf dem Spielfeld (falls noch ein Leben übrig ist), schießt kein Gegner (sie bewegen sich jedoch) und der Count-down bleibt so lange stehen.
Es gibt theoretisch unendlich viele Spielstufen, ihr erhöht also einfach immer nur die Anzahl, Geschwindigkeit und Schussfrequenz - irgendwann wird das Spiel eh unspielbar - und wie sich das Aussehen der Gegner verändert dazu später mehr.

Um dem Spieler eine zusätzliche Orientierung über den Spielverlauf zu geben, werden außerhalb des Spielfeldes (wo genau ist egal) folgende Angaben angezeigt:

1. Die Anzahl der dem Spieler zur Verfügung stehenden Leben.
2. Die Nummer der aktuellen Spielstufe (Zählung beginnt bei eins mit der ersten Spielstufe und wird mit jeder neuen Spielstufe um eins erhöht).
3. Die Anzahl der Punkte des Spielers.
4. Die noch verbleibende Zeit zum Beenden der aktuellen Spielstufe. Diese wird dabei in Stunden, Minuten und Sekunden angezeigt und während einer Spielstufe rückwärts (also von der vollen zur Verfügung stehenden Zeit runter auf null Stunden, null Minuten und null Sekunden) gezählt.

Nach dem Ende eines Spiels wird, wenn der Spieler genug Punkte gesammelt hat, um in eine Highscore mit insgesamt 10 Einträgen zu kommen, dieselbe angezeigt. Außerdem soll diese Highscore jederzeit (aber nicht mitten im Spiel) auf Benutzerwunsch angezeigt werden können. Um den Inhalt der Highscore über das Ende der Programmbenutzung hinaus zu erhalten, soll dieser in einer Datei abgespeichert werden. Welcher Art diese Datei ist, oder wie die Informationen in der Datei gespeichert werden, ist euch überlassen. Während eines Spiels soll der Spieler die Möglichkeit haben, dieses zu pausieren als auch ganz zu beenden.

- Weite Information zur genauen Aufgabenstellung sind in der Original-Aufgabenstellung, die im Anhang zu finden ist!

# 3. Benutzerhandbuch

## *3.1 Ablaufbedingungen*

Für den Ablauf des Programms werden folgende Hard- und Softwarekomponenten vorausgesetzt:

| **Hardware** |
|---|
| • IBM-kompatibler PC<br>• schneller Prozessor wird vorausgesetzt, je nach verwendeten Bildern, mind. 500 MHz empfohlen<br>• mindestens 2 Mega Byte freier Festplattenspeicher<br>• mind. 64 MB RAM<br>• VGA-Grafikkarte, minimale Auflösung von 800x600, mind. 256 Farben (auf langsamen Systemen unbedingt Windows-Beschleuniger Karten)<br>• Soundkarte empfohlen<br>• CD-ROM-Laufwerk zur Installation |

| **Software** |
|---|
| • Betriebssystem:<br>- Windows 2000 oder Windows XP<br>- Windows Win 9x / Me (evtl. mit reduzierter Spielgeschwindigkeit)<br>- Linux mit entspr. Wine-Konfiguration (nur für Experten)<br>• Grafikbearbeitungsprogramm zum Entwurf eigener Spielgrafiken empfohlen aber nicht zwingend notwendig |

## 3.2 Programminstallation

Um „Fighter“ Spielen zu können, muss das Spiel zuerst installiert werden. Zur Installation des Programms muss das Verzeichnis *„Fighter“* inklusive der darin enthaltenden Dateien (mindestens: *Fighter.exe, default.dat, config.ini*) von der Installations-CD auf die Festplatte kopiert werden. Dort kann dann das Spiel durch Aufruf von *„Fighter.exe“* gestartet werden.

Durch Löschen des kopierten Verzeichnisses (inkl. aller Unterdateien) kann das Spiel einfach wieder deinstalliert werden.

## 3.3 Programmstart

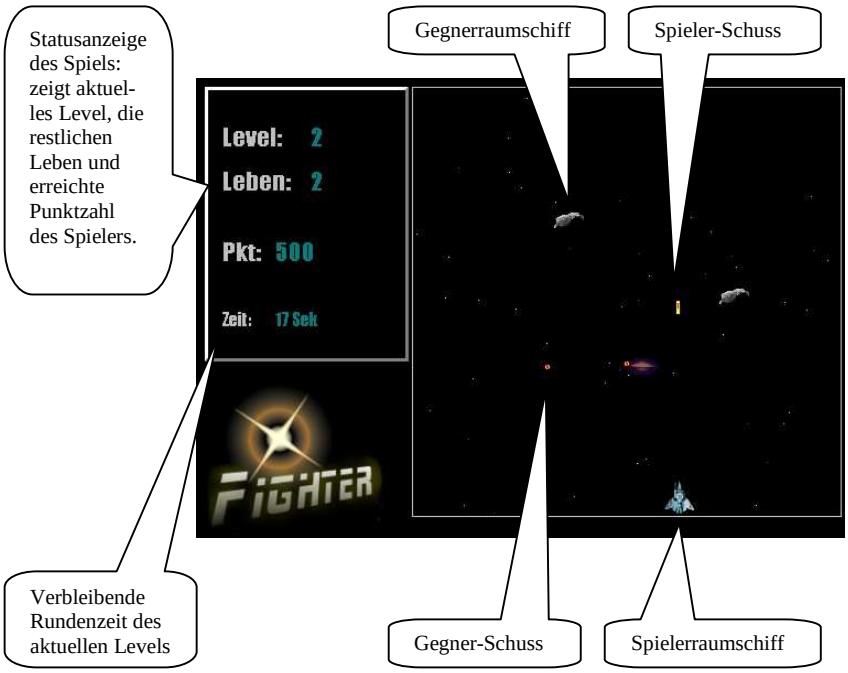
Das Spiel wird nach der Installation durch Aufruf der Datei *„Fighter.exe“* aus dem Spielverzeichnis von der Festplatte gestartet. Dies erreichen Sie, indem Sie mit der linken Maustaste im Windows-Explorer doppelt auf den Dateinamen „Fighter.exe“ klicken. Der Programmstart direkt von CD ist nicht möglich (da auf diesem Medium kein Schreibzugriff gestattet ist)!
Sobald das Spielfenster erscheint kann durch Klick auf das Menü *„Spiel→Neues Spiel“* ein neues Spiel gestartet werden, woraufhin der Spieler in Level 1 beginnt!
Wenn das Programm fragt, ob der Schreibschutz einer Datei entfernt werden soll, so ist dies unbedingt mit *„Ja“* zu beantworten.


## 3.4 Bedienungsanleitung

„Fighter“ ist ein Geschicklichkeitsspiel, bei dem der Spieler ein Raumschiff steuert und versucht Gegnerraumschiffe abzuschießen oder deren feindlichen Geschossen auszuweichen.

### 3.4.1 Das Spielfeld

Durch Klick auf das Menü *„Spiel→Neues Spiel“* wird ein neues Spiel gestartet, woraufhin der Spieler in Level 1 beginnt! Das quadratische Spielfeld zeigt am unteren Rand das Spielerraumschiff, das mit Hilfe der Tastatur nach links und rechts bewegt werden kann. Darüber fliegen auf zufälligen Bahnen Gegnerraumschiffe, die versuchen, je nach Level unterschiedlich gut, den Spieler abzuschießen. Ebenso kann auch der Spieler Schüsse nach oben abfeuern, um die feindlichen Raumschiffe vom Spielfeld zu verbannen und Punkte zu ergattern. Den aktuellen Punktestand des Spielers, sowie die verbleibende Rundenzeit und die Anzahl der restlichen Leben des Spielers sind zur besseren Orientierung immer links neben dem Spielfeld zu sehen.

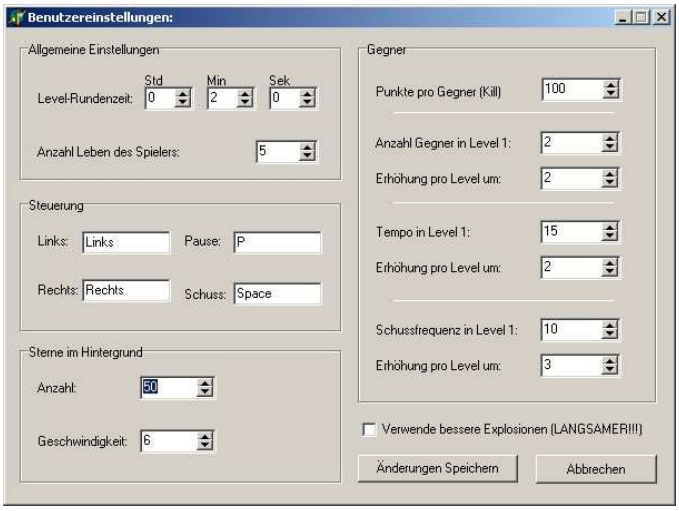



### 3.4.2 Spielsteuerung und Ablauf


Sobald ein neues Spiel gestartet wurde, erscheint die Meldung „*Achtung das Spiel geht los*“ auf dem Spielfeld und der Spieler hat noch kurz Zeit, sich darauf einzustellen, bis kurz darauf die ersten Gegner erscheinen. Ab diesem Zeitpunkt können sowohl der Spieler als auch die Gegner beginnen sich zu bewegen oder Schüsse abzufeuern. Der Spieler kann sein Raumschiff mit den Cursor-Tasten [←] und [→] seitlich hin und herbewegen, jeweils bis zum linken oder rechten Rand des quadratischen Spielfelds, weiter geht es nicht. Außerdem hat er zu jedem Zeitpunkt die Möglichkeit, also auch während einer Bewegung, einen Schuss von seiner aktuellen Position senkrecht nach oben abzufeuern, um eines der gegnerischen Raumschiffe zu treffen. Dies geschieht mit der [Leertaste]. Jeder getroffene Gegner bringt dem Spieler 100 Punkte. Der Spieler hat ein Level geschafft, wenn kein Gegner mehr auf dem Spielfeld ist, was durch den Text „*Level erfolgreich*“ bestätigt wird. Der Spieler hat wieder einen kurzen Moment Ruhe, bis es im nächsten Level mit neuen Gegner weitergeht, solange bis er alle seine Leben verloren hat und das Spiel „GameOver“ ist. Soll ein laufendes Spiel pausiert werden, so kann dies durch einen Druck auf die Taste „*P*“ geschehen, ein erneutes Drücken setzt das Spiel fort.

### 3.4.3 Einstellungen


„Fighter“ bietet Ihnen die Möglichkeit, bestimmte Einstellungen des Spielverhaltens zu verändern und an Ihre eigenen Vorstellungen und Wünsche anzupassen. Durch Klick auf das Menü „*Spiel→Einstellungen*“ wird ein neues Formular geöffnet, mit dessen Hilfe Sie diese Änderungen vornehmen können.
Das Formular unterteilt sich in vier verschiedene Rubriken:




In der Rubrik „**Allgemeine Einstellungen**“ kann die Rundenzeit für jeweils einen Level eingestellt werden. Schafft der Spieler es nicht innerhalb dieser Zeit alle Gegner zu zerstören, verliert er selbst ein Leben und muss es noch einmal versuchen.
Die Anzahl wie oft der Spieler von Gegnern getroffen werden darf, bis das Spiel „GameOver“ ist, kann hier ebenfalls unter „*Anzahl Leben des Spielers*“ gesetzt werden.

Die Rubrik „**Steuerung**“ enthält die zu verwendenden Tasten für das Bewegen des Raumschiffes (Links/Rechts), das Abfeuern eines Schusses und die Pause-Taste. Um eine Funktion neu zu belegen, klicken Sie mit der Maus einfach auf das entsprechende, weiße Editfeld und drücken die gewünschte neue Taste. Nach dem Loslassen der Taste erscheint dann automatisch im Editfeld die entsprechende Beschreibung.

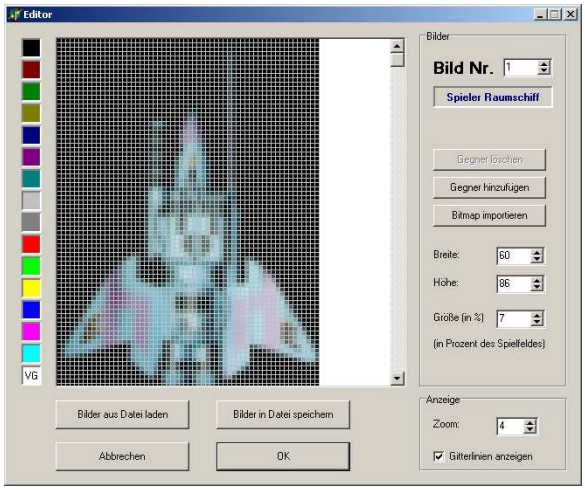
In der Rubrik „**Sterne im Hintergrund**“ können Sie die Anzahl der im Hintergrund des Spielfelds fliegenden Sterne bestimmen. Bitte Beachten Sie, dass zu viele Sterne das Spielen evtl. erschweren können. Ebenso kann die Rechenleistung Ihres PCs stark darunter leiden. Der Wert „*Geschwindigkeit*“ gibt an, wie schnell die Sterne sich maximal nach unten bewegen können.

Auf der rechten Seite des Formulars finden Sie die Rubrik „**Gegner**“. Dort können Sie das Verhalten der Gegner in großem Umfang kontrollieren. Der oberste Wert „*Punkte pro Gegner (Kill)*“ besagt, wie viele Punkte dem Spieler für jeden getroffenen Gegner gutgeschrieben werden. Um die Highscore nicht zu verfälschen, sollten Sie hier keine großen Änderungen vornehmen. Die darunter liegenden Werte sind immer in Zweier-Paaren aufgeteilt. Der jeweils obere Wert gibt an, wie viele Gegner im ersten Level vorhanden sind und wie schnell und mit welcher Schussfrequenz sie zu Beginn starten. Darunter steht jeweils die Erhöhung pro Level, wie viele Gegner pro Level dazukommen, wie viel schneller sie werden und wie viel öfter sie schießen. Der Wert der Schussfrequenz besagt wie viele Schüsse ein Gegner in einer Minute abfeuert.

Die Auswahlbox „**Verwende bessere Explosionen (LANGSAMER!!!)**“ bietet dem Spieler die Option, realistischere Explosionen einzuschalten, so dass getroffene Raumschiffe grafisch ansprechender zerfallen. Aber Vorsicht, diese Option verlangt deutlich mehr Rechenleistung und sollte nur auf schnellen Computern aktiviert werden!

## 3.4.4 Editor


Durch Klick auf den Menüpunkt „*Editor*“ wird das Editor-Formular geladen und angezeigt. Mit Hilfe des Editors ist es für Sie möglich, das Aussehen der Spielfiguren (Spielerraumschiff, Gegnerraumschiffe und Schüsse) individuell an Ihre Vorstellungen und Wünsche anzupassen.




### 3.4.4.1 Aufbau der Bilderlisten

Die in „Fighter“ verwendeten Bilder der einzelnen Spielobjekte werden in einer Bilderliste verwaltet. Diese Liste besteht aus mindestens vier Bildern:

1. Bild → Bild des Spielerraumschiffs
2. Bild → Bild für die vom Spieler abgefeuerten Schüsse
3. Bild → Bild für die von Gegnern abgefeuerten Schüsse
4. [5,6,7...] Bild → Bild 4 (und falls vorhanden alle nachfolgenden Bilder) stellt einen Gegner dar

Wenn man den Editor aufruft, werden automatisch die Bilder der Bilderliste aus Datei „*default.dat*“ (aus dem Spielverzeichnis, in dem auch das Programm „*Fighter.exe*“ enthalten ist) geladen. Schlägt dies fehl, wird eine neue Bilderliste mit vier leeren Standard-Bildern zum Bearbeiten erzeugt. Es kann immer nur ein Bild zurzeit angezeigt und bearbeitet werden. Nach dem Aufruf des Editors wird standardmäßig immer das erste Bild (das Spielerraumschiff) geladen.

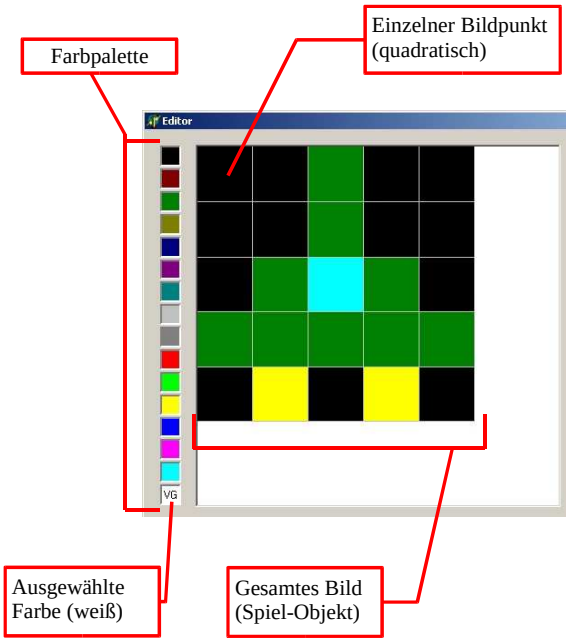

Welches Bild gerade im Editor zur Bearbeitung bereitsteht wird oben rechts in der Ecke unter „*Bild Nr.*“ angezeigt. Durch klicken auf die Pfeile (↑, ↓) neben der Bild Nr. können Sie zu anderen Bildern der Liste wechseln. Das neue Bild wird dann geladen und steht sofort zur Bearbeitung bereit.
Mit Hilfe des Buttons „*Gegner hinzufügen*“ können Sie außerdem die Bildliste um beliebig viele, neue Bilder für Gegner ergänzen. Nach Klick auf den Button wird automatisch ein neues, leeres, schwarzes Bild erzeugt und als letzter Gegner bereitgestellt.
Der Button „*Gegner löschen*“ ist nur aktiv, wenn sie einen Gegner ausgewählt haben und noch mindestens ein weiterer Gegner in der Liste ist. Wenn Sie den Button drücken, wird der gerade angezeigte Gegner gelöscht!

### 3.4.4.2 Laden und Speichern von Bilderlisten


Nachdem Sie jede Grafik der Bilderliste nach Ihren Wünschen fertig gestellt haben, ist es sinnvoll diese Bilderliste in einer Datei abzuspeichern. Klicken Sie dafür auf den Button „*Bilder in Datei speichern*“. Daraufhin erscheint ein neues Fenster, dass Sie auffordert einen Namen für die zu speichernde Datei einzugeben. Tippen Sie einen Dateinamen ein und bestätigen Sie den Speichervorgang durch Klick auf den Button „*Speichern*“. Ihre Bilderliste ist nun in der Datei abgelegt.
Auf ähnliche Weise können sie auch eine fertige Bilderliste aus einer Datei laden, um sie beispielsweise im Editor zu ändern oder ihre Bilder für ein neues Spiel zu benutzen. Klicken Sie diesmal auf den Button „*Bilder aus Datei laden*“ und suchen Sie die gewünschte Datei aus. Daraufhin wird Ihre gewählte Datei untersucht und deren Bilderliste in den Editor geladen.

**Achtung:** Sobald Sie im Editor auf den Button „*OK*“ drücken, wird die aktuell geladene Bilderliste in der Datei „*default.dat*“ gespeichert, eine vorhandene Datei wird überschrieben! „Fighter“ verwendet immer die Datei „*default.dat*“, wenn Sie ein neues Spiel starten!

### 3.4.4.3 Bearbeiten eines Bildes




Sobald Sie ein Bild zum Bearbeiten in den Editor geladen haben, erscheint es (wie links im Bild) in vergrößerter Darstellung als Matrixstruktur. Wenn Sie einen einzelnen Bildpunkt mit der linken Maustaste anklicken, nimmt dieser Bildpunkt die aktuell ausgewählte Farbe an. Möchten Sie eine andere Farbe verwenden, können Sie durch einen Links-Mausklick, auf ein Quadrat der Farbplatte eine neue Farbe auswählen, die zukünftig als Zeichenfarbe verwendet wird. Welcher Farbwert gerade aktiv ist, können Sie anhand der beiden Buchstaben ‚*VG*‘ auf dem entsprechenden Quadratfeld der Farbpalette erkennen (links im Bild ist dies die Farbe weiß). Bitte beachten Sie, dass später der Hintergrund im Spielfeld immer schwarz ist, schwarze Bildpunkte sich also nicht vom Hintergrund absetzen können. Darum sollten Sie möglichst keinen größeren schwarzen Rahmen um das eigentliche Raumschiff zeichnen, da dieser später für den Spieler nicht sichtbar ist, aber für das Spielverhalten mit einbezogen wird (der Spieler sich dann also evtl. über für ihn nicht erkennbare Treffer wundert).


Mit Hilfe der „*Zoom*“-Einstellung, die sie unten rechts im Formular in der Rubrik Anzeige finden, wird Ihnen das Bearbeiten und Überschauen des Bildes erleichtert. Wird hier ein Wert von „1“ eingestellt, so können Sie sich das Bild in „Originalgröße“ ansehen (da das Bild später auf die aktuelle Spielfeldgröße angepasst wird, weicht diese Größe später im Spielfeld etwas ab!). Größere Werte zeigen das Bild zum Bearbeiten um den ausgewählten Faktor vergrößert an, das gilt jedoch nur für den Bildeditor und wirkt sich nicht auf die spätere Größe auf dem Spielfeld aus! Mit der Option „*Gitterlinien anzeigen*“ können Sie außerdem festlegen, ob die einzelnen Bildpunkte im Editor durch Gitterlinien getrennt angezeigt werden. Die Darstellung mit Gitterlinien vereinfacht in der Regel die Bearbeitung, die Darstellung ohne Gitterlinien bietet dafür eine bessere Vorschau der gemachten Änderungen an der Grafik.


Als weitere Option finden sie rechts neben dem zu bearbeitenden Bild die Felder „*Breite*“ und „*Höhe*“. Durch Verändern dieser Werte, können Sie das Bild um Bildpunkte vergrößern bzw. verkleinern. Der Wert „*Breite*“ fügt am rechten Bildrand der Grafik neue Bildpunkte hinzu bzw. entfernt die äußerste Reihe, die Option „*Höhe*“ funktioniert entsprechend, mit dem Unterschied, dass die Bildpunkte am unteren Ende der Grafik eingefügt bzw. entfernt werden.

Mit Hilfe des wichtigen Schalters „*Größe (in %)*“ müssen Sie festlegen, wie groß die aktuelle Grafik später als Spielobjekt auf dem Spielfeld erscheinen soll. Geben Sie beispielsweise für das Bild vom Spieler einen Wert von 10% ein, so belegt das Spielerraumschiff im Spiel alleine schon 10% des gesamten Spielfeldes.

### 3.4.5 Highscore



Durch Klick auf den Menüpunkt „*Highscore*“ wird ein Formular geöffnet, das die Highscore anzeigt. Hier sind die zehn besten Spieler mit Namen und erreichter Punktzahl aufgelistet (wie links im Beispielbild zu erkennen; Der Spieler „*Bender*“ auf Platz 1 und „*Slurm Mc Kenzie*“ an 10. Stelle auf dem untersten Platz).


Um sich selbst in dieser Liste verewigen zu können, muss man ein neues Spiel starten und mehr Punkte erzielen, als der unterste Spieler aus der Highscore (in diesem Beispiel also mehr als „*Slurm Mc Kenzie*“ / 1000 Punkte). Nach Ende des Spiels wird man dann aufgefordert seinen Namen einzugeben, der Spieler mit der geringsten Punktzahl wird dann automatisch entfernt und die Highscore wird neu sortiert!

## *3.5 Fehlermeldungen*

In „Fighter“ können folgende Fehlersituationen auftreten:

| **Fehlermeldung** | **Fehlerursache** | **Behebungsmaßnahme** |
|---|---|---|
| „Ihre Eingaben sind fehlerhaft, bitte keine gleichen Steuerungstasten verwenden!“ | Bei den Benutzereinstellungen sind in der Rubrik „*Steuerung*“ für mindestens zwei unterschiedliche Funktionen die gleiche Taste definiert worden. | - Definieren Sie für jede Steuerfunktion eine von den anderen verschiedene, eindeutige Taste. |
| „Fehler beim Speichern der Benutzereinstellungen in Datei: <Dateiname>“ | I/O-Fehler beim Speichern. Dieser Fehler kann nach dem Ändern der Benutzereinstellungen auftreten, wenn z.B. die Datei von einer anderen Anwendung blockiert oder das Dateisystem fehlerhaft ist. | - Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Kontrollieren Sie, ob genügend Platz auf dem Datenträger vorhanden ist und „Fighter“ Schreibrechte auf dem Datenträger besitzt.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |
| „Fehler beim Laden der Benutzereinstellungen aus Datei: <Dateiname>“ | I/O-Fehler beim Laden. Dieser Fehler kann beim Laden der Benutzereinstellungen (Aufruf von „Einstellungen“ oder „Neues Spiel“) auftreten. Möglicherweise ist die Datei von einer anderen Anwendung geöffnet oder das Dateisystem ist beschädigt. | - Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |
| „Fehler beim Laden der Highscore aus Datei: <Dateiname>“ | I/O-Fehler beim Laden. Dieser Fehler kann beim Laden der Highscoreliste (Durch Aufruf aus dem Menü oder nach Ende eines Spiels) auftreten. Möglicherweise ist die Datei von einer anderen Anwendung geöffnet oder das Dateisystem ist beschädigt. | - Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |
| „Fehler beim Speichern der Highscore in Datei: <Dateiname>“ | I/O-Fehler beim Speichern. Dieser Fehler kann auftreten, wenn am Ende eines Spiels die neue Highscore nicht gespeichert werden kann, weil z.B. die Datei von einer anderen Anwendung blockiert oder das Dateisystem fehlerhaft ist. | - Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Kontrollieren Sie, ob genügend Platz auf dem Datenträger vorhanden ist und „Fighter“ Schreibrechte auf dem Datenträger besitzt.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |

| | | |
|---|---|---|
| „<Dateiname> nicht gefunden! Verwende Standard-Werte!“ | I/O-Fehler beim Laden. Der Fehler kann auftreten, wenn die Benutzereinstellungen oder die Highscoreliste nicht geladen werden konnte, weil die Datei evtl. gelöscht wurde und nicht mehr existiert. | - Stellen Sie sicher, dass die Datei im angegebenen Verzeichnis existiert. Ist sie nicht aufzufinden, kopieren Sie die Datei erneut von der Installations-CD in das Spielverzeichnis. |
| „Fehler beim Speichern der Spiel-Grafiken in Datei: <Dateiname>“ | I/O-Fehler beim Speichern. Dieser Fehler kann auftreten, wenn im Editor eine Bilderliste gespeichert werden soll, aber z.B. die Festplatte voll ist oder ein ungültiger Dateiname/Pfad angegeben wurde. | - Versuchen Sie es noch einmal mit einem anderen Dateinamen.<br>- Kontrollieren Sie, ob genügend Platz auf dem Datenträger vorhanden ist und „Fighter“ Schreibrechte auf dem Datenträger besitzt.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem.<br>- Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift. |
| „Fehler beim Importieren der Bitmap aus Datei: <Dateiname>“ | Dieser Fehler kann auftreten, wenn im Editor eine Spielgrafik aus einer Bitmap-Datei geladen werden soll. Vermutlich ist das Bitmap-Format nicht kompatibel oder es trat ein I/O-Fehler auf. | - Versuchen Sie eine andere Bitmap-Datei zu laden und überprüfen Sie, ob es sich dabei um eine Standard-Bitmap-Datei ohne Komprimierung handelt.<br>- Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |
| „Fehler beim Laden der Spiel-Grafiken aus Datei: <Dateiname>!<br>Lege vier leere Standard Bilder als Grafiken an!“ | Dieser Fehler tritt auf, wenn das Laden einer Bilderliste im Editor fehlschlägt. Möglicherweise ist die Datei in einem ungültigen Format/beschädigt, existiert nicht oder ist von einer anderen Anwendung blockiert. Im Editor werden daher vier leere Bilder zur Bearbeitung erzeugt! | - Stellen Sie sicher, dass die Datei im angegebenen Verzeichnis existiert.<br>- Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem.<br>- Wenn Sie eine andere Datei mit den selben Bildern zur Sicherheit angelegt haben, versuchen Sie diese zu öffnen, andernfalls kann diese Bilderliste nicht mehr geladen werden und sie müssen eine andere verwenden (evtl. die Original-Bilderliste von der Installations-CD erneut kopieren und öffnen) oder selbst eine neue Bilderliste erstellen. |

| | | |
|---|---|---|
| „Fehler beim Laden der Spiel-Grafiken, das Spiel kann ohne Bilder nicht starten.“ | Dieser Fehler tritt auf, wenn ein neues Spiel gestartet wird, aber keine gültige Bilderliste aus der Datei „*default.dat*“ geladen werden konnte. | - Stellen Sie sicher, dass die Datei „*default.dat*“ im Spielverzeichnis existiert und gültige Spiel-Grafiken enthält. (Mit dem Editor kann dies überprüft werden). Gelingt dies nicht, so kann die Datei „*default.dat*“ erneut von der Installations-CD in das Spielverzeichnis kopiert werden. |
| „Soll der Schreibschutz der Datei: <Dateiname> entfernt werden?“ | Diese Fehlermeldung tritt beim Laden bzw. Speichern der Datei auf, wenn sie schreibgeschützt ist. | Um erfolgreich aus der Datei zu lesen bzw. Änderungen abzuspeichern, muss der Schreibschutz entfernt werden. Dies kann durch bejahen der Frage geschehen, andernfalls bleibt die Datei unverändert und die Anweisung wird abgebrochen. |
| „Fehler: Datei <Dateiname> existiert nicht!“ | Dieser Fehler tritt auf, wenn versucht wird eine Datei zu Laden, die nicht existiert. | - Stellen Sie sicher, dass die Datei vorhanden ist und der Dateiname/Pfad korrekt ist, erscheitn die Fehlermeldung weiterhin, dann<br>- überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |
| „Fehler beim Entfernen des Datei-Schreibschutzes von Datei: <Dateiname>“ | Dieser Fehler kann auftreten, wenn versucht wird eine schreibgeschützte Datei zu laden bzw. speichern, aber z.B. eine andere Anwendung den Zugriff blockiert oder das Medium keinen Schreibzugriff bietet (z.B. bei CD-ROM-Medien) | - Stellen Sie sicher, dass keine andere Anwendung auf diese Datei zugreift.<br>- Kontrollieren Sie, ob „Fighter“ Schreibrechte auf dem Datenträger besitzt (von CD kann nur gelesen werden!).<br>- überprüfen Sie Ihre Festplatte auf fehlerhafte Strukturen im Dateisystem. |

## *3.6 Wiederanlaufbedingungen*

Beim Spielen von „Fighter“ kann es zur Laufzeit evtl. passieren, dass das Programm z.B. aufgrund eines Stromausfalls oder Hardware- bzw. Systemfehlers plötzlich ungewollt abgebrochen wird. In diesem Fall ist das aktuelle Spiel vorbei und kann leider nicht weiter fortgesetzt werden. Wurden gerade Änderungen an Einstellungen oder Grafiken im Editor gemacht, so sind auch diese unwiderruflich verloren gegangen. Im Normalfall kann „Fighter“ aber trotzdem weiterhin verwendet werden, allerdings ohne die gerade gemachten Änderungen. In seltenen Fällen kann es passieren, dass die Einstellungen oder Bilderlisten derart zerstört sind, dass sie erneut von der Installations-CD kopiert werden müssen. In diesem Fall müssen Sie die beiden Dateien „*default.dat*“ und „*config.ini*“ aus dem Verzeichnis „*Fighter*“ der Installations-CD in das Spielverzeichnis auf Ihrer Festplatte kopieren und die beschädigten Dateien überschreiben.

# 4. Programmierhandbuch

## *4.1 Entwicklungskonfiguration*

Die Entwicklung von „Fighter“ wurde mit Hilfe folgender Konfiguration durchgeführt:

**Hardware**

- AMD Athlon XP 1600+ @ 1,4 GHz
- 512 MB Arbeitsspeicher
- Ati Radeon 8500 Grafikkarte, 64 MB
- 19” Monitor mit einer Auflösung von 1280x1024 und 32-Bit Farbtiefe (True Color)
- Stereo-Soundkarte
- CD-ROM und Diskettenlaufwerk

**Software**

- Betriebssystem: Windows XP
- Programmierumgebung: Delphi 6.0 (Personal und Professional Edition)
- Grafikprogramm Photo Impact 5 zum Entwurf eigener Spielgrafiken
- UltraEdit-32 Version 9.0 zum Bearbeiten der Quellcodes

## *4.2 Problemanalyse und Realisation*

Das Realisieren des Spiels „Fighter“ stellt mehrere Anforderungen an den Programmierer. Die wichtigsten Bestandteile des Programms sind:

- Das Konzept zur Spielsteuerung (Game-Loop)
- Die Realisierung des Bewegungsablaufs der verschiedenen Objekte (Spieler- und Gegnerraumschiffe, Sterne, Schüsse)
- Das Bereitstellen eines Editors für den Anwender zum Entwurf eigener Spielgrafiken
- Die Funktionen zum Speichern und Laden von Benutzereinstellungen, der Highscoreliste und Spielgrafiken

### 4.2.1 Das Konzept zur Spielsteuerung (Game-Loop)

#### 4.2.1.1 Beschreibung des Spielfelds durch das Hauptformular

Bei der Programmierung von „Fighter“ spielt das Hauptformular eine sehr wesentliche Rolle. Das Formular enthält sowohl das Spielfeld (*Image*) und Spielstatus-Anzeigen (Labels), die Image-Liste für die Explosionen, als auch die drei notwendigen Timer. Die Behandlung der gedrückten Spieltasten und das Ändern der Spielfeldgröße sind außerdem Ereignisse, die ebenfalls über das Formular abgewickelt werden müssen. Aus diesen Gründen findet man auch in der zum Hauptformular gehörenden Unit (*UMain*) einige kleine Routinen, die für Spielsteuerung notwendig sind, die eigentliche Spiellogik ist jedoch auf die Units der einzelnen Objekte verteilt.

**Anmerkung:**
Einige Routinen im Hauptformular wie z.B. „*CheckGameStatus*“ oder „*NextLevel*“ ließen sich evtl. in die „*UGame*“-Unit auslagern, weil sie unter anderem kleine Teile der Spiellogik enthalten. Ich habe das jedoch bewusst vermieden, da die Übersichtlichkeit der kurzen Funktionen dann durch sehr lange Parameterlisten (bedingt durch die vielen Formular-Komponenten, die mit übergeben werden müssten) stark leiden würde!

#### 4.2.1.2 Aufbau der Game-Loop durch Timer-Ereignisse

Um ein flüssiges Darstellen des Spielfelds mit allen Spiel-Objekten (Spieler, Gegner, Schüsse, Sterne, Explosionen, Texte) mit den Windows Befehlen zu realisieren, ist folgender Aufwand sinnvoll:

- Eine Ereignis-Funktion, die durch einen Windows-Timer (*DrawTimer*) alle 10ms aufgerufen wird, ist für das Zeichnen des gesamten Spielfelds verantwortlich.
- Innerhalb der Zeichenroutine, werden alle Grafikaufrufe zunächst auf einen ‚Buffer' im Arbeitsspeicher geschrieben. Erst am Ende der Zeichenroutine wird dann dieser ‚Buffer' auf die echte Image-Komponente übertragen, so dass die Änderungen sichtbar werden. Dieser Ansatz reduziert das ungewollte Flackern im Spielverlauf stark, auch wenn für derartige Spiele eine direkte Nutzung der Grafikkarte z.B. via DirectX oder OpenGl wesentlich bessere Ergebnisse liefern würde (bei höherem Aufwand).

Innerhalb der Ereignis-Funktion zum Zeichnen des Spiels, wird folgender Ablauf alle 10 ms wiederholt. Der Ablauf ist der eigentliche Spielverlauf (etwas vereinfacht) und entspricht sozusagen der Game-Loop:

<table>
<tr><td colspan="3">Das gesamte Spielfeld wird gelöscht (alles schwarz gefüllt) und ein dünner Rahmen wird zur Abgrenzung um das Spielfeld gezeichnet</td></tr>
<tr><td colspan="3">Die Sterne werden je nach Geschwindigkeit ein kleines Stück weiter nach unten bewegt und anschließend gezeichnet</td></tr>
<tr><td colspan="3">Die Gegner werden ein kleines Stück bewegt</td></tr>
<tr><td colspan="3">Soll ein Gegner gerade schießen?</td></tr>
<tr><td colspan="2">ja</td><td>nein</td></tr>
<tr><td colspan="2">Ein neuer Gegnerschuss wird abgesetzt</td><td></td></tr>
<tr><td colspan="3">Zeichne die Gegner</td></tr>
<tr><td colspan="3">Die gegnerischen Schüsse werden ein kleines Stück nach unten bewegt und anschließend gezeichnet.</td></tr>
<tr><td colspan="3">Die Spieler-Schüsse werden ein kleines Stück nach oben bewegt und anschließend gezeichnet.</td></tr>
<tr><td colspan="3">Wurde ein Gegner durch ein Spieler-Schuss getroffen?</td></tr>
<tr><td colspan="2">ja</td><td>nein</td></tr>
<tr><td colspan="2">Entferne den Gegner, löse Explosion aus und schreibe dem Spieler Punkte gut.</td><td></td></tr>
<tr><td colspan="3">Jetzt wird der Spieler evtl. bewegt, anschließend gezeichnet.</td></tr>
<tr><td colspan="3">Wurde der Spieler von einem Gegner / Gegner-Schuss getroffen?</td></tr>
<tr><td colspan="2">ja</td><td>nein</td></tr>
<tr><td colspan="2">Ziehe dem Spieler ein Leben ab</td><td rowspan="3"></td></tr>
<tr><td colspan="2">Hat der Spieler noch mehr als ein Leben?</td></tr>
<tr><td>ja</td><td>nein</td></tr>
<tr><td></td><td>Das Spiel wird beendet, evtl. neue Highscore.</td><td></td></tr>
<tr><td colspan="3">Jetzt werden die Explosionen vorbereitet (das richtige Bild wird ermittelt) und angezeigt.</td></tr>
<tr><td colspan="3">Nun werden vorhandene Text-Meldungen angezeigt und anschließend evtl. abgedimmt bzw. entfernt.</td></tr>
<tr><td colspan="3">Jetzt wird der Spielstatus überprüft bzw. gesetzt (Levelzeit abgelaufen? Level fertig?).</td></tr>
<tr><td colspan="3">Als letztes wird der ‚Buffer' sichtbar gezeichnet und die Spieler-Punktzahl aktualisiert.</td></tr>
</table>

„*Fighter*" verwendet neben dem eigentlichen Timer zum Zeichnen des Spiels noch zwei weitere Timer:

- LevelTimer: Dieser Timer wird jede Sekunde aufgerufen und zählt somit die aktuelle Rundenzeit.
- EnemyTimer. Dieser Timer wird zweimal die Sekunde aufgerufen und ist für das Verhalten der gegnerischen Schüsse relevant (Siehe Kap. 4.2.2.4 *Das Schussverhalten der Gegner*).

## 4.2.2 Die Realisierung des Bewegungsablaufs der Objekte

### 4.2.2.1 Der Objektorientierte Ansatz

Da in dem Spiel „Fighter“ die verschiedensten Objekte (Sterne, Spieler- und Gegnerraumschiffe, Schüsse, Explosionen und Texte) auf dem Spielfeld erscheinen, kann es auch bei der Programmierung sinnvoll sein, diese Spielobjekte zu übernehmen und sie nach objektorientertem Ansatz entsprechend zu kapseln. Diese Vorgehensweise hat bei der Programmierung den Vorteil, dass das Programm übersichtlicher wird, da die zusammenhängenden Spielobjekte auch während der Programmierung zusammen in einer Struktur erhalten bleiben. Der zweite große Vorteil der Objektorienterten Programmierung ist die Vererbung. Da die verschiedenen Spiel-Objekte sehr ähnliche Eigenschaften besitzen (z.B. Position auf dem Spielfeld, Geschwindigkeit, Größe...), kann man mit Hilfe einer Basis-Klasse gleiche Eigenschaften für die daraus resultierenden Klassen geschickt zusammenfassen. In diesem Fall ist die Basis-Klasse „*TFighterObject*“ in der Unit „*UObjFighter*“ vorhanden. Sie stellt allen von ihr abgeleiteten Klassen bestimmte Grundfunktionen zur Verfügung. Dazu gehört z.B. das Aufbauen und Bearbeiten von Listen mit den von allen Objekten genutzten Grundelementen (wie Position auf dem Spielfeld, Breite und Höhe etc.), sowie Positionierungsberechnungen auf dem Spielfeld oder Bildgrößenanpassungen der Objekte.
Der für die Listenfunktionen benötigte Datentyp (*TFighterRec*), der alle ähnlichen Eigenschaften der unterschiedlichen Spielobjekte zusammenfasst, enthält folgende Elemente:
X/Y-Position, Geschwindigkeit, Breite und Höhe, Farbe, Gültigkeit (Leben), Größe, Text-Infos und erweiterte Bewegungsinfos eines Objekts.

### 4.2.2.2 Die Bewegung der Sterne im Hintergrund

Die Funktionen zur Darstellung der während eines Spiels im Hintergrund fliegenden Sterne sind in der Klasse „*TStars*“ verankert. Diese Klasse verwaltet eine eigene Liste, in der jeder Stern mit aktueller Position und seinen Eigenschaften (z.B. Farbe, Geschwindigkeit, Größe) vermerkt ist. Die Zeichenfunktion der Klasse (*Draw*) kann dann zu jedem Zeitpunkt alle Sterne mit ihren individuellen Eigenschaften auf das Spielfeld zeichnen und dabei jeden Stern für sich entsprechend weiterbewegen.
Die Zeichenfunktion arbeitet nach folgendem Prinzip:

| Durchlaufe jeden Stern der Liste | |
|---|---|
| | Ermittle seine Eigenschaften (Position, Farbe, Geschwindigkeit etc.) |
| | Verändere seine Position nach seiner Geschwindigkeit (bewege ihn) |
| | Zeichne ihn mit seinen Eigenschaften auf das Spielfeld |

Diese Vorgehensweise hat den Vorteil, dass nicht alle Sterne identisch aussehen. Außerdem können sie sich mit unterschiedlicher Geschwindigkeit bewegen, was eine realistischere Umgebung erzeugt.

### 4.2.2.3 Die Bewegung des Spielers

Das Spieler-Objekt „*TPlayer*“ steuert die Bewegungen, die Anzeige und das Verhalten des Spielerraumschiffs und der vom Spieler abgefeuerten Schüsse. Die Klasse enthält alle dazu notwendigen Eigenschaften, Listen und Funktionen.

► *Die Anzeige des Spielers:*
Die Zeichenroutine des Spielers (*Draw*) wird mit jedem Timer-Ereignis alle 10ms aufgerufen. In Abhängigkeit des aktuellen Spielstatus wird zunächst das Spielerraumschiff bewegt und anschließend gezeichnet. Das Spielerraumschiff wird dabei zweimal als Bitmap-Grafik im Objekt gespeichert. Das Original-Bild dient als Ausgangsbild, aus dem das aktuelle Spielerraumschiff abgeleitet werden kann. Dies ist notwendig, damit nach Größenänderungen keine Bildinformationen verloren gehen (siehe Kap. 4.2.2.8).
Nach dem Zeichnen wird geprüft, ob das Spielerraumschiff mit einem Gegner oder einem gegnerischen Schuss kollidiert.

► *Die Bewegung des Spielers:*
Um die Steuerung des Spielerraumschiffs für den Spieler möglichst komfortabel zu gestalten, wird hier ein besonderes Verfahren angewendet. Durch Herunterdrücken der Spielertasten Links und Rechts wird nicht direkt das Raumschiff bewegt, sondern zunächst ein Bewegungsmodus ein bzw. durch Loslassen der Taste ausgeschaltet. Wird z.B. die Links-Bewegung eingeleitet, so wird das Raumschiff mit jedem neuem Timer-Ereignis weiter nach Links bewegt, bis das Raumschiff am Rand angelangt ist oder der Bewegungsmodus durch Loslassen der Taste ausgeschaltet wird.

► *Die Bewegung und Anzeige abgefeuerter Schüsse:*
Die Spieler-Klasse verwaltet außerdem eine Liste mit allen vom Spieler abgefeuerten Schüssen. Die Funktion „*DrawShoot*“ ist für die Anzeige und Bewegung der einzelnen Schüsse dieser Liste zuständig. Sie arbeitet nach folgendem Prinzip:

<table>
<tr><td colspan="4">Durchlaufe jeden Schuss der Liste</td></tr>
<tr><td></td><td colspan="3">Ermittle seine Eigenschaften (Position, Größe, Geschwindigkeit etc.)</td></tr>
<tr><td></td><td colspan="3">Verändere seine Position nach seiner Geschwindigkeit (bewege ihn)</td></tr>
<tr><td></td><td colspan="3">Ist Schuss noch im Spielfeld?</td></tr>
<tr><td></td><td colspan="2">ja</td><td>nein</td></tr>
<tr><td></td><td colspan="2">Zeichne ihn mit seinen Eigenschaften auf das Spielfeld</td><td rowspan="4">Entferne Schuss aus der Liste</td></tr>
<tr><td></td><td colspan="2">Hat Schuss einen Gegner getroffen?</td></tr>
<tr><td></td><td>ja</td><td>nein</td></tr>
<tr><td></td><td>Zeige Trefferpunkte<br>Erhöhe Spielerpunktzahl<br>Entferne Schuss aus der Liste</td><td></td></tr>
</table>

### 4.2.2.4 Die Bewegung der Gegner

Auch das gesamte Gegnerverhalten ist in eine Klasse gekapselt (TEnemy). Dieses Objekt verwaltet zwei Listen. Die erste Liste enthält alle auf dem Spielfeld vorhandenen Gegner inklusive ihren Eigenschaften. Die zweite Liste beinhaltet jeden abgefeuerten Schuss (irgendeines Gegners), der sich noch auf dem Spielfeld befindet. Darüber hinaus beinhaltet die Klasse zwei Bilderlisten, in denen die Grafiken der verschiedenen Gegner sowie die Grafik des Gegner-Schusses enthalten sind. Die eine Liste enthält die Bilder in Originalgröße, die andere in an das Spielfeld angepasster Größe (siehe Kap. 4.2.2.8).
Besonders wichtig sind die verwendeten Algorithmen zur Steuerung der Schüsse, die Kollisionsabfragen, sowie die Bewegungen der Gegner.

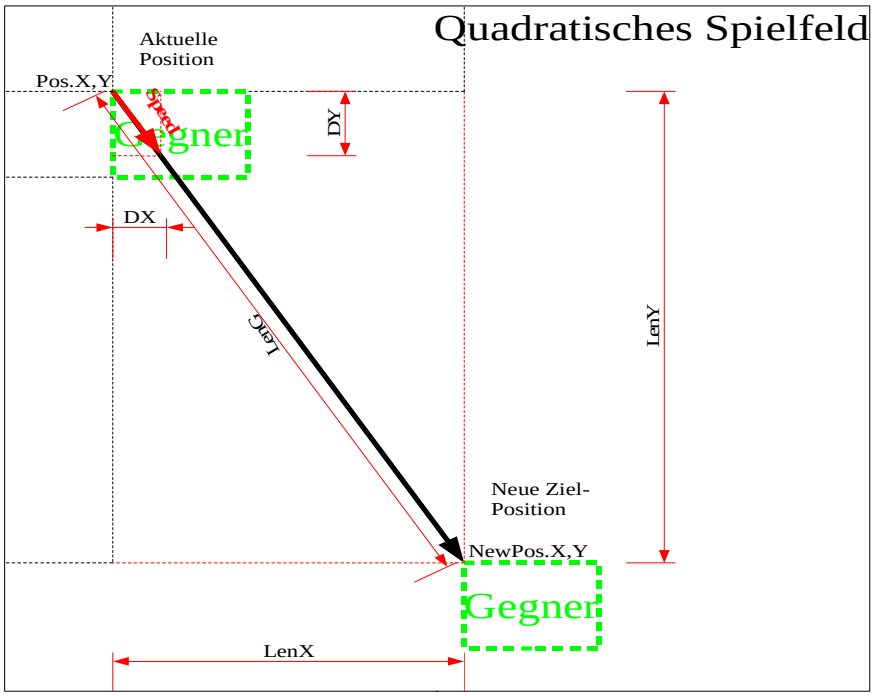
► *Das Anzeigen und Bewegen der Gegner:*
Die Zeichenfunktion (*Draw*) des Gegner-Objekts ist für das Zeichnen, Bewegen und Schießen aller Gegner verantwortlich. Die Methode durchläuft alle Gegner der Liste. Jeder Gegner wird dabei zunächst bewegt, danach wird überprüft, ob ein Schuss abgesetzt werden soll und zuletzt wird der Gegner auf das Spielfeld gezeichnet:

<table>
<tr><td rowspan="5">Durchlaufe jeden Gegner der Liste</td><td colspan="2">Ermittle seine Eigenschaften (Position, Größe, Geschwindigkeit etc.)</td></tr>
<tr><td colspan="2">Verändere seine Position nach seiner Geschwindigkeit (bewege ihn)</td></tr>
<tr><td colspan="2">Ist es Zeit für den Gegner einen Schuss abzusetzen?<br>ja | nein</td></tr>
<tr><td>Setze einen neuen Schuss von der aktuellen Position ab (neues Element in der Schuss-Liste)</td><td></td></tr>
<tr><td colspan="2">Zeichne den Gegner mit seinen Eigenschaften auf das Spielfeld</td></tr>
</table>

Die einzelne Bewegung eines Gegners geschieht mit Hilfe des folgenden Verfahrens:

- Ein Gegner erscheint an einer zufälligen Position im Spielfeld (aufgrund der Fairness zum Spieler immer über ihm).
- Es wird eine neue zufällige Position (= zufällige X,Y-Werte) ermittelt, zu dieser „fliegt“ das Gegnerraumschiff auf linearem Wege.
- Ist der Gegner an der neuen Position angekommen, wird eine neue Zufallsposition als Ziel generiert und dasselbe Verfahren wird erneut angewendet.

Um den linearen Weg fliegen zu können muss jeweils berechnet werden, wie weit sich der Gegner in X- und Y-Richtung bewegen muss, damit er auf geradem Wege am Ziel ankommt. Aus diesem Grunde werden die Positions-Koordinaten (und Zielkoordinaten) jedes Gegners durch Kommazahlen angegeben, obwohl das Objekt beim Zeichnen auf dem Spielfeld natürlich nur mit ganzen Zahlen beschrieben werden kann (es gibt keine halben Pixel, vor dem Zeichnen wird die Positionsangabe also gerundet). Der dafür geeignete Typ wurde „*TRealPoint*“ genannt. Die Informationen der aktuellen und Zielposition stehen im Typen „*TMoveRec*“, welches als Element der Eigenschaften jedes Gegners vorhanden ist.




Wie die Berechnung (wie weit der Gegner in X- bzw. Y-Richtung bewegt werden muss) arbeitet, wird anhand der linken Zeichnung erklärt. Der rote Vektor „*Speed*“ gibt die Geschwindigkeit an, um die das Raumschiff bei jedem Schritt zum Ziel hin bewegt wird. Je länger er ist, umso schneller ist das Raumschiff am Ziel. Die benötigten Werte DX und DY können mit Hilfe des Strahlensatzes ermittelt werden:

$$\frac{DX}{Speed} = \frac{LenX}{LenG} \text{ bzw. } \frac{DY}{Speed} = \frac{LenY}{LenG}$$

Der Gesamtweg (*LenG*) kann mit Hilfe des Satzes von Pythagoras berechnet werden:

$$LenG = \sqrt{(LenX)^2 + (LenY)^2}$$

► *Das Schussverhalten der Gegner:*
Da die Gegner eine bestimmte vom Benutzer festgelegte Anzahl an Schüssen pro Minute abfeuern sollen, dient ein (Enemy-) Timer dazu, 120 Ereignisse pro Minute auszulösen. Das ist die maximal höchstmögliche Schussfrequenz. Um nun zu wissen, wann ein Gegner einen Schuss abfeuern soll, wird eine Schuss-Menge mit (je nach festgelegter Anzahl vielen) zufälligen Werten zwischen 1 und 120 für jeden Gegner generiert. Wird ein Gegner gezeichnet, so wird anschließend überprüft, auf welchem Wert der (Enemy-) Timer gerade steht. Ist der Wert in der Schuss-Menge des Gegners enthalten, so wird ein neuer Schuss von seiner aktuellen Position abgesetzt. Und der Wert aus der Schuss-Menge entfernt. Ist der (Enemy-) Timer bei 120 angekommen, so wird er wieder auf 1 gesetzt, alle Gegner bekommen neue zufällige Schuss-Mengen und dasselbe Verfahren beginnt erneut.

► *Die Kollisionsabfragen – Berührung mit Gegner oder Gegnerschuss:*
Die Überprüfung, ob ein anderes Objekt (z.B. das Spielerraumschiff oder Spielerschuss) einen Gegner oder einen Gegnerschuss berührt/trifft, wird ebenfalls von der Gegner-Klasse übernommen. Die Methoden (HitEnemy bzw. HitEnemyShoot), die diese Kollision kontrollieren, arbeiten nach folgendem Prinzip:

<table>
<tr><td colspan="3">Durchlaufe jeden Gegner / Gegnerschuss der Liste</td></tr>
<tr><td></td><td colspan="2">Ermittle seine Eigenschaften (Position, Größe, Geschwindigkeit etc.)</td></tr>
<tr><td></td><td colspan="2">Berechne seine maximalen Ausmaße auf dem Spielfeld (sein Rechteck)</td></tr>
<tr><td></td><td colspan="2">Überlappt sich sein Rechteck mit dem des zu prüfenden Objekts?</td></tr>
<tr><td></td><td>ja</td><td>nein</td></tr>
<tr><td></td><td>Entferne Gegner / Gegnerschuss aus der Liste</td><td rowspan="2"></td></tr>
<tr><td></td><td>Zeige ggfs. eine Explosion an</td></tr>
</table>

#### 4.2.2.5 Das Anzeigen von Explosionen
Die Klasse „*TFire*“ dient zur Anzeige von Explosionen auf dem Spielfeld. Auch sie enthält und verwaltet eine Liste. In dieser Liste sind die zurzeit auf dem Spielfeld sichtbaren Explosionen gespeichert. Das Anzeigen der Explosionen funktioniert ähnlich wie bei den anderen Objekten. Es wurden zwei verschiedene Methoden implementiert. Die eine Methode arbeitet mit einer Image-Liste (*FireImageList*), in der acht Einzelbilder nacheinander gezeigt werden. Da die Einzelbilder sich im Spielverlauf an die jeweilige Größe des getroffenen Gegners oder Spielers anpassen müssen und die von Windows bereitgestellte Funktion zum ‚Stretchen' eines Bildes relativ langsam ist, ist für langsamere Computer die zweite Methode eher empfehlenswert. In diesem Fall wird einfach ein gelbes Rechteck mit schwarzem Rand in der Größe des getroffenen Objekts gezeichnet. Nach kurzer Zeit wird der schwarze Rand immer größer und größer, bis die Explosion komplett schwarz ist und schließlich ganz verschwindet.

#### 4.2.2.6 Das Anzeigen von Informationen für den Spieler
Im Spielverlauf von „Fighter“ müssen des Öfteren Nachrichten und Spielstatusinformationen an den Spieler gegeben werden. Da es besser aussieht, wenn diese Text-Meldungen direkt auf dem Spielfeld erscheinen, stellt die Klasse „*TText*“ genau diese Funktionalität zur Verfügung. Sie verwaltet eine Liste mit beliebig vielen Texten, die auf dem Spielfeld erscheinen, blinken oder durch Dimmen langsam verschwinden können. Die Zeichenfunktion (*Draw*) arbeitet sehr ähnlich wie die der anderen Objekte, besitzt aber zusätzlich die Option, Texte zu dimmen (schwarz werden lassen). Um dies zu erreichen, wird die aktuelle Farbe zunächst in ihre Rot-, Grün- und Blauanteile zerlegt. Diese Werte geben an, wie intensiv (hell) der jeweilige Farbanteil ist. Zum Dimmen wird nun je nach gewünschter Dimm-Geschwindigkeit von jedem Farbanteil ein konstanter Wert abgezogen. Die resultierende Farbe wird also immer dunkler und schließlich schwarz, wenn alle Farbanteile Null erreicht haben.

#### 4.2.2.7 Die Klasse „TExtImgLst“ zur Verwaltung von Bilderlisten

Diese Klasse ist nicht wie die anderen Klassen von der Basis-Klasse „*TFighterObject*“ abgeleitet, sonder ist eine eigene Klasse, die als Erweiterung der Delphi-Klasse „*TImageList*“ angesehen werden kann. Sie verwaltet ähnlich wie das Delphi-Pendant unterschiedliche Bitmap-Grafiken in einer Liste, kann aber auch mit Bildern unterschiedlicher Größen umgehen.

#### 4.2.2.8 Anpassungen an Spielfeldgrößenänderung

Da der Anwender bei diesem Spiel in der Lage sein soll, das Fenster und somit das Spielfeld jederzeit nach seinen Wünschen in der Größe zu verändern, müssen alle Objekte auf dem Spielfeld in der Lage sein, sich an die neue Größe anzupassen. Aus diesem Grund besitzen die Klassen „*TStars*“, „*TPlayer*“ und „*TEnemy*“ allesamt die jeweils ähnliche Methode „*Resize*“. Die beiden Klassen „*TText*“ und „*TFire*“ haben diese Funktionalität nicht, da deren Elemente jeweils nur kurz auf dem Spielfeld zu sehen sind, so dass sich eine Anpassung nicht lohnt. Die „*Resize*“-Methode kümmert sich jeweils um folgende Aufgaben:

► *Anpassen der Größen der Spiel-Grafiken (Bitmaps):*

Da die Spiel-Objekte intern durch Bitmaps repräsentiert werden, muss jedes Bild (Bitmap) zweimal vorliegen. Die eine Version enthält das Bild in Originalgröße, das andere Bild die zum aktuellen Spielfeld passenden Größe. Ändert sich nun das Spielfeld, so kann ein Kopie des Originalbildes in die neue richtige Größe ‚gestretcht’ werden und anschließend als aktuelles Bitmap benutzt werden. Dadurch vermeidet man die durch mehrmaliges Ändern des gleichen Bildes entstehende schlechte Bildqualität.
Die neuen Abmessungen eines aktuellen Bildes berechnet man mit Hilfe des Verhältnissatzes (jede Grafik soll ja einen bestimmten Prozentanteil des Spielfelds belegen):

$$\frac{NeueBreite/Höhe}{Prozent} = \frac{NeueSpiefeldgröße}{100}$$

Die Basis-Klasse „*TFighterObject*“ bietet hierfür die Methode „*ResizeBmp*“ an, die nach diesem Prinzip arbeitet.

► *Anpassen der neuen Geschwindigkeit:*

Der Wert der Geschwindigkeit gibt an, um wie viele Pixel (evtl. gerundet) ein Spiel-Objekt verschoben wird. Aus diesem Grund muss auch dieser Wert an eine neue Spielfeldgröße angepasst werden, damit z.B. bei doppelter Spielfeldgröße das Spielerraumschiff nicht plötzlich langsamer wird. Die Umrechnung geschieht auch hier mit Hilfe des Verhältnissatzes:

$$\frac{NeueGeschwindigkeit}{AlteGeschwindigkeit} = \frac{NeueSpielfeldgröße}{AlteSpielfeldgröße}$$

► *Anpassen der Positionen der einzelnen Spiel-Objekte (Gegner, Spieler, Schüsse, Sterne):*

Auch die Position eines Spiel-Objekts muss an eine neue Spielfeldgröße angepasst werden, damit z.B. ein Spiel-Objekt in der Mitte auch in der Mitte bleibt. Auch dies geschieht mit Hilfe des Verhältnissatzes:

$$\frac{NeuePosition}{NeueSpielfeldgröße} = \frac{AltePosition}{AlteSpielfeldgröße}$$

Die Basis-Klasse „*TFighterObject*“ bietet hierfür die Methode „*PosChange*“ an, die nach diesem Prinzip arbeitet.

## 4.2.3 Das Bereitstellen eines Editors für den Anwender

### 4.2.3.1 Bildverwaltung durch Bilderlisten

Im Editor können die Spielgrafiken editiert werden. Durch Aufrufen des Formulars oder explizites Laden einer Datei, wird eine Bilderliste („*TExtImgLst*“) erzeugt, in der alle Spielgrafiken als Bitmaps mit ihrem zugehörigen Protzenanteil abgelegt werden. Das aktuell zu bearbeitende Bild wird zunächst in einem ‚Buffer’ zwischengespeichert und dann im Grid-Feld angezeigt. Beim Wechsel des aktiven Bildes (z.B. durch Klicken auf das nächste Bild der Liste) wird zunächst die Grafik aus dem aktuelle ‚Buffer’ in die Bilderliste zurückgesichert. Dann wird das neue Bild aus der Bilderliste geladen, anstelle des alten Bildes in den ‚Buffer’ geschrieben und schließlich angezeigt. Somit ist immer gewährleistet, dass alle Änderungen am aktiven Bild gesichert werden.

### 4.2.3.2 Die Zeichenfunktionalität des Grid-Felds

Um dem Anwender einen funktionsfähigen Grafik-Editor zu präsentieren, wurde bei der Realisierung ein Grid-Feld umfunktioniert. Durch Abfangen und Bearbeiten des Ereignisses „*OnDrawCell*“, kann jede Zelle des Grid-Feldes selbst gezeichnet werden. Da die Matrix des Grid-Feldes immer genauso groß ist, wie die des Bitmaps im ‚Buffer’, kann beim Zeichnen einer Grid-Zelle einfach der entsprechende Farbwert aus dem ‚Buffer’-Bitmap an derselben Stelle (Reihe, Spalte) entnommen werden, so dass die Zelle mit dem richtigen Farbwert gefüllt wird. Klickt der Anwender nun mit der Maus auf eine Zelle, so wird die aktuelle Farbe des markierten Eintrags der Farbplatte ausgelesen. Diese Farbe wird nun zunächst im ‚Buffer’-Bitmap an der entsprechenden Position (Reihe, Spalte) geändert, dann wird das Grid neugezeichnet. Die Änderung wird also automatisch sichtbar, da der ‚Buffer’ zum Zeichnen ausgelesen wird.

## 4.2.4 Die Funktionen zum Speichern und Laden

### 4.2.4.1 Speichern und Laden der Benutzereinstellungen

Die Benutzereinstellungen werden in der Datei „*config.ini*“ abgespeichert. Diese typische Ini-Datei ist eine Textdatei, die aus mehreren Abschnitten besteht, in denen wiederum mehrere Schlüsselvariablen stehen, die alle jeweils einen Wert besitzen. Ein Abschnitt ist immer durch einen Bezeichner in eckigen Klammern [,] gekennzeichnet. In den nächsten Zeilen folgen dann die Schlüsselvariablen, die jeweils nach einem ‚=’ einen Wert festhalten. Das Laden und Speichern der Benutzereinstellungen geschieht mit Hilfe der fertigen Funktionen der Delphi-Klasse „*TIniFile*“, denen man nur entsprechende Abschnitte und Schlüsselvariablen übergibt, ohne sich um die eigene Dateiverarbeitung zu kümmern. Sollte es beim Laden der Einstellungen zu Fehlern kommen (z.B. weil Datei nicht vorhanden oder Abschnitte in der Datei fehlen), so werden alle Einträge mit Standard-Werten belegt!

| | |
|---|---|
| Abschnitt: | [General] |
| Schlüsselvariable: | LevelTime |
| Wert: | 45 |

```
[General]
LevelTime=45
Lives=3

[Stars]
Count=50
MaxSpeed=6

[Control]
Left=37
Right=39
Fire=32
Pause=80

[Enemy]
Points=100
IncPoints=100
Count=3
IncCount=2
Speed=15
IncSpeed=2
Fire=10
IncFire=2
```

#### 4.2.4.2 Speichern und Laden der Highscoreliste

Ähnlich wie die Benutzereinstellungen wird auch die Highscoreliste in der Ini-Datei „*config.ini*“ im Abschnitt „*[Highscore]*“ abgespeichert. Die Schlüsselvariablen für die einzelnen Spieler sind:

PlayerName1=Bender, PlayerPoints1=10000, PlayerName2=Fry, PlayerPoints2=9000, PlayerName3=Nibbler...

Auch hier werden beim Laden im Fehlerfall (z.B. weil Datei nicht vorhanden oder Abschnitte in der Datei fehlen) Standardwerte für die Highscoreliste verwendet!

```
[Highscore]
PlayerName1=Bender
PlayerPoints1=10000
PlayerName2=Fry
PlayerPoints2=9000
PlayerName3=Nibbler
PlayerPoints3=8000
PlayerName4=Amy Wong
PlayerPoints4=7000
PlayerName5=Hermes Conrad
PlayerPoints5=6000
PlayerName6=Leela
PlayerPoints6=5000
PlayerName7=Dr. Zoidberg
PlayerPoints7=4000
PlayerName8=Zapp Brannigan
PlayerPoints8=3000
PlayerName9=Prof. Farnsworth
PlayerPoints9=2000
PlayerName10=Slurm Mc Kenzie
PlayerPoints10=1000
```

#### 4.2.4.3 Speichern und Laden der Spielgrafiken

Die Spielgrafiken sind bei „Fighter“ in Dat-Dateien abgelegt. Das Laden (Speichern ist analog) einer Bilderliste geschieht nach folgendem Verfahren:

- Zunächst wird überprüft, ob die zu ladende Datei überhaupt existiert. Wurde die Datei gefunden, so wird kontrolliert, ob die Datei schreibgeschützt ist. Wenn dies der Fall ist, wird der Schreibschutz entfernt.
- Als nächstes wird eine leere Bilderliste (*TExtImgLst*) erzeugt und initialisiert.
- Nun wird solange ein Grafik-Block aus der Datei gelesen, bis das Dateiende erreicht ist. Dazu wird jeweils erstmal ein neuer Speicherbereich (*TMemorySteam*) bereitgestellt. Dann wird ein Byte aus der Datei gelesen, welches die prozentuale Größe auf dem Spielfeld des folgenden Bildes repräsentiert. Daraufhin wird die Größe (in Bytes) des folgenden Bitmap-Streams gelesen. Mit Hilfe dieser Größeninformation kann nun der zuvor bereitgestellte Speicherbereich auf die richtige Größe eingestellt werden. Danach kann der folgende Bitmap-Stream aus der Datei direkt in den reservierten Speicherbereich gelesen werden.
- Jetzt wird der Bitmap-Stream aus dem Speicher in ein Bitmap-Objekt umgewandelt und anschließend am Ende in die Bilderliste eingefügt.

## *4.3 Beschreibung grundlegender Datenstrukturen*

### 4.3.1 Verwendete Konstanten

In dem Programm kommen viele Konstanten zum Einsatz, da mit Ihnen wichtige Parameter des Spiels beschrieben werden, ohne dass der Programmierer den gesamten Quelltext durchsuchen muss. Alle Konstanten befinden sich in der Unit „*UTypes*“, werden komplett großgeschrieben (jeder Buchstabe) und beginnen mit einem „C_“ für „const“.

Die wichtigsten Konstanten werden im Folgenden kurz beschrieben.

| **Name der Konstanten** | **Kurze Beschreibung** |
|---|---|
| C_STARCOLORS : array[1..5] of TColor = ( $FFFFFF, $FFFF00, $FFC0C0, $C0FFFF, $FF4040 ); | Definiert ein Array mit fünf konstanten Farbwerten. Die Sterne im Hintergrund werden in einer dieser Farben gezeichnet. |
| C_DEF_PLAYFIELD = 500; | Gibt die Standard Spielfeldgröße an, an der sich die Größenänderungen orientieren. Da das Spielfeld von einem „*TImage*“ dargestellt wird, muss auch die Größe auf dieser Komponente beachtet werden. |
| C_DEF_ENEMY_SPEED_POTI = 0.05; | Mit Hilfe dieses Faktors kann die Gegner-Geschwindigkeit auf dem Spielfeld beeinflusst werden. |
| C_PLAYER_MOVE_SPEED = 3; | Gibt an, wie schnell sich das Spielerraumschiff hin- und her bewegen kann |
| C_PLAYER_SHOOT_SPEED = 6; | Gibt an, wie schnell ein Schuss nach oben läuft |
| C_MAX_PLAYER_SHOOT = 5; | Gibt an, wie viele Schüsse der Spieler abfeuern kann. Ist diese Konstante z.B. Eins, darf der Spieler immer nur einen Schuss zurzeit abfeuern (auch sehr interessanter Spielmodi!) |
| C_ENEMY_SHOOT_SPEED = 2; | Gibt an, wie schnell ein Gegnerschuss sich nach unten bewegt. |
| C_FIRE_SPEED = 10; | Gibt an, wie lange die einzelnen Phasen (Bilder) einer Explosion sichtbar sind. |
| C_SND_??? | Alle Konstanten mit diesem Anfang weisen auf Dateinamen der abzuspielenden Sounds |
| C_ERR_??? | Alle Konstanten mit diesem Anfang enthalten die unterschiedlichen Fehlermeldungen des Programms |
| C_MSG_??? | Alle Konstanten mit diesem Anfang enthalten unterschiedliche Nachrichten an den Benutzer |

## 4.3.2 Eigene Typen

### 4.3.2.1 Auflistung und Beschreibung aller eigenen Typen

Hier finden Sie eine Auflistung aller Typen, die im Programm verwendet werden:

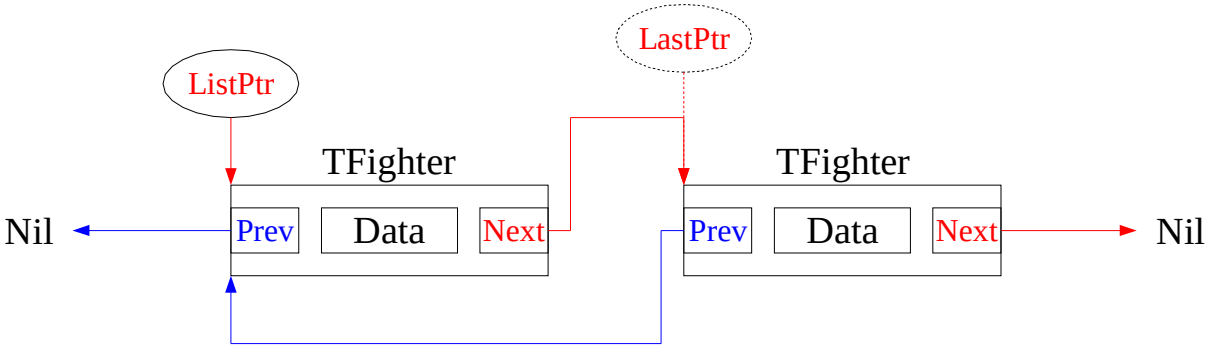
| **Typdeklaration** | **Beschreibung** |
|---|---|
| TTimeRange = 0..59; | Gibt den Bereich einer Minute in Sekunden an und wird zur Berechnung / Anzeige der Levelzeit benötigt. |
| TEnemyShoot = 1..C_MAX_ENEMY_SHOOT; | Dieser Typ definiert den möglichen Bereich (Anzahl) für Gegnerschüsse in einer Minute |

| | |
|---|---|
| TEnemyShootSet = set of TEnemyShoot; | Jeder Gegner besitzt eine Menge dieses Typen, damit er weiß, wann er scheißen soll |
| TGameStatus = ( St_Playing, St_TimeOver, St_FinishedLevel, St_PlayerDead, St_PlayerComeBack, St_Pause, St_NoGame); | Dieser Typ definiert die verschiedenen Spiel-Status-Informationen, die im Spiel auftreten können. |
| TPlayTime = record<br>Hours : TTimeRange;<br>Minutes : TTimeRange;<br>Seconds : TTimeRange;<br>end; | Dieser Typ wird verwendet, um die aktuelle übrige Levelzeit zu speichern bzw. anzuzeigen. |
| TBmpLstPtr = ^TBmpLst; | Zeiger auf eine Bildliste (für „*TExtImgLst*“) |
| TBmpLst = record<br>Size : Byte;<br>Bmp : TBitmap;<br>Prev : TBmplstPtr;<br>Next : TBmpLstPtr;<br>end; | Dieser Typ stellt ein Element der Bilderliste dar. Die beiden Zeiger „*Prev*“ und „*Next*“ zeigen dabei auf den Vorgänger bzw. Nachfolger der Liste. „*Size*“ gibt an, wie viel Prozent das Bild in „*Bmp*“ auf dem Spielfeld belegen soll. |
| TRealPoint = record<br>X,Y : Real;<br>end; | Dieser Typ ist ein Pendant zu „*TPoint*“, nur mit Kommazahlen. Der Typ wird für die genauen Positionen der Gegner benötigt. |
| TMoveRec = record<br>Pos : TRealPoint;<br>NewPos : TRealPoint;<br>Shoot : TEnemyShootSet;<br>end; | Dieser Typ dient zur Speicherung der Bewegungs- und Schussinformationen der Gegner.<br>▪ „*Pos*“ entspricht der aktuellen Position des Gegners auf dem Spielfeld.<br>▪ „*NewPos*“ speichert den nächsten Punkt, zu dem der Gegner bewegt wird<br>▪ „*Shoot*“ enthält die Schuss-Menge des Gegners, damit dieser weiß, wann er schießen darf. |
| TFighterRec = record<br>x,y : Integer;<br>Speed : Integer;<br>Width,<br>Height : Integer;<br>Color : TColor;<br>Lives : Integer;<br>Size : Integer;<br>Str : string;<br>Move : TMoveRec;<br>end; | Dieser Typ enthält Infos für ein Spiel-Objekt. Er ist ein wichtigster Typ, da er von allen Objekten als Listenelement eingesetzt wird.<br>▪ „*X,Y*“ speichert die Position des Objekts<br>▪ „*Speed*“ speichert die Geschwindigkeit des Objekts<br>▪ „*Width*“, „*Height*“ sind für Breite und Höhe<br>▪ „*Color*“ enthält die Farbe des Objekts<br>▪ „*Lives*“ speichert die Anzahl Leben oder wie lange ein Objekt zu sehen ist<br>▪ „*Size*“ speichert die Größe oder bei den Gegnern das Level des Objekts<br>▪ „*Str*“ enthält Text-Infos<br>▪ „*Move*“ speichert die Bewegungs-/Schuss-Infos für ein Gegner-Objekt |
| TFighterPtr = ^TFighter; | Dieser Typ wird als Zeiger auf eine Fighter-Objekt-Liste eingesetzt, die von allen eigenen Objekten (über die Basisklasse „*TFighterObject*“) benutzt wird. |

| | |
|---|---|
| TFighter = record<br>Data : TFighterRec;<br>Prev : TFighterPtr;<br>Next : TFighterPtr;<br>end; | Dieser Typ stellt ein Element der Fighter-Objekt-Liste dar.<br>▪ In „*Data*“ ist der gesamte Inhalt enthalten<br>▪ „*Prev*“ und „*Next*“ sind Zeiger auf Vorgänger bzw. Nachfolger in der Liste. |
| TPlayername = String[C_MAX_PLAYERNAME_LEN]; | Dieser Typ definiert einen begrenzten String für den Spielername in der Highscore. |
| TScoreElement = record<br>PlayerName : TPlayerName;<br>PlayerPoints : Cardinal;<br>end; | Dieser Typ wird für einen Eintrag aus der Highscore mit Spieler-Namen und erreichter Punktzahl verwendet. |
| THighscore = array[1..C_MAX_HIGHSCORE_ENTRIES] of TScoreElement; | Dieser Typ wird ebenfalls für die Highscore verwendet. Eine Variable dieses Typen beinhaltet die gesamte Highscoreliste. |
| TGame = record<br>FieldSize : Integer;<br>Status : TGameStatus;<br>LevelTime : Cardinal;<br>Lives : Integer;<br>NiceLook : Boolean;<br><br>StCount : Word;<br>StMaxSpeed : Byte;<br><br>CtrLeft : Word;<br>CtrRight : Word;<br>CtrFire : Word;<br>CtrPause : Word;<br><br>EnPoints : Cardinal;<br>EnCount : Word;<br>EnIncCount : Word;<br>EnSpeed : Byte;<br>EnIncSpeed : Byte;<br>EnFire : Byte;<br>EnIncFire : Byte;<br>end; | Dieser Typ speichert alle wichtigen Informationen für ein Spiel:<br>▪ „*FieldSize*“ - Spielfeldgröße<br>▪ „*Status*“ - aktueller Spiel-Status (Pause, Level geschafft...)<br>▪ “*LevelTime*“ - Rundenzeit des Levels<br>▪ “*Lives*“ - Anzahl Leben des Spielers<br>▪ „*NiceLook*“ - Ob Hochauflösende Explosionen benutzt werden sollen<br><br>▪ „*StCount*“ - Anzahl der Sterne in Hintergrund<br>▪ “*StMaxSpeed*“ - Maximale Geschwindigkeit der Sterne<br><br>▪ “*CtrLeft*“ - Steuertaste Links<br>▪ “*CtrRight*“ - Steuertaste Rechts<br>▪ “*CtrFire*“ - Steuertaste Schießen<br>▪ “*CtrPause*“ - Steuertaste Pause<br><br>▪ „*EnPoints*“ - Punktzahl für ein getroffenen Gegner in Level 1<br>▪ „*EnCount*“ - Anzahl der Gegner im ersten Level<br>▪ “*EnIncCount*“ - Wie viele Gegner pro Level mehr?<br>▪ „*EnSpeed*“ – Geschw. der Gegner in Level 1<br>▪ „*EnIncSpeed*“ - Erhöhung der Geschwindigkeit pro Level<br>▪ „*EnFire*“ - Schussfrequenz der Gegner im ersten Level<br>▪ „*EnIncFire*“ - Erhöhung der Schussfrequenz pro Level |

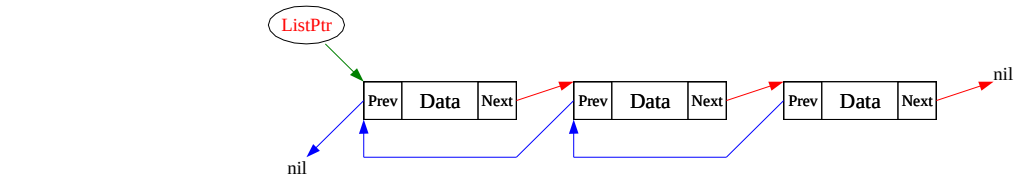
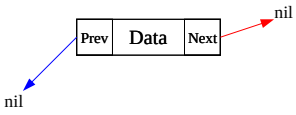
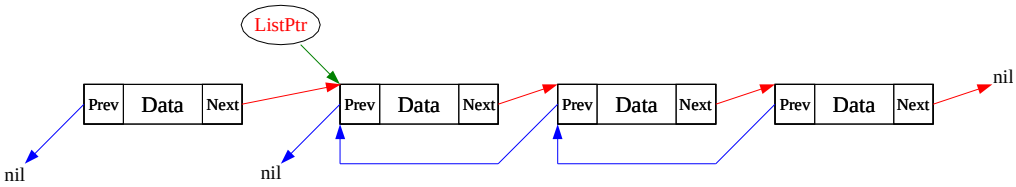
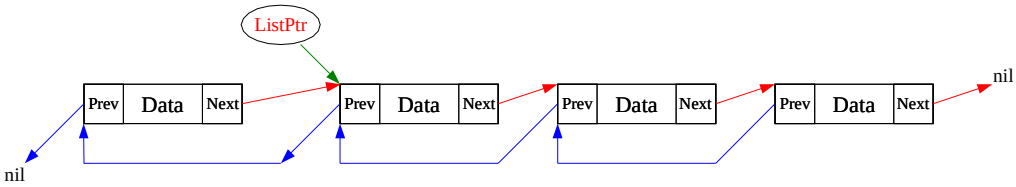
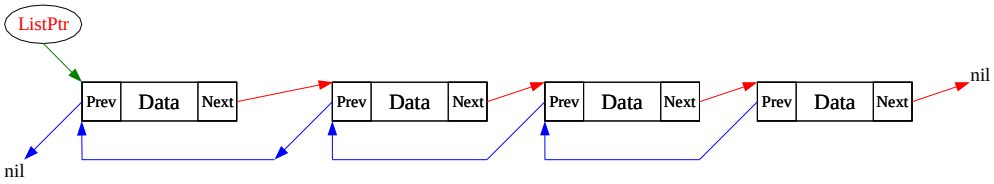
### 4.3.2.2 Die Zeigerstruktur der Bilderlisten (TExtImgLst) und Fighter-Objekt-Listen (TFighterPtr)

Bei den hier eingesetzten Listen handelt es sich in beiden Fällen um doppelt verkettete lineare Listen. Beide Listen sind über einen Zeiger „*ListPtr*“, der auf das erste Element zeigt, zu erreichen. Im Gegensatz zur Fighter-Objekt-Liste besitzt die Bilderliste jedoch zusätzlich einen Zeiger auf das letzte Element („*LastPtr*“), da dies das Einfügen eines Elements am Ende der Liste vereinfacht. Die folgende Grafik zeigt den Aufbau der doppelt verketteten linearen Liste mit zwei Elementen:




Bei dieser Liste stellt „*Data*“ die Info-Komponente dar, „*Prev*“ und „*Next*“ sind jeweils Zeiger auf den Vorgänger bzw. Nachfolger. „*Data*“ ist ein Record vom Typ „*TFighterRec*“, das die Informationen eines Objekts wie z.B. Position, Geschwindigkeit, Farbe, Größe etc. speichert.

► *Beispiel zum Einfügen eines neuen Elements am Anfang der Liste:*

| | |
|---|---|
|  | Dies ist die Ausgangsliste. Hier soll am Anfang ein neues Element eingefügt werden. |
|  | Zuerst wird ein neues Element erstellt, dessen Zeiger beide zunächst auf „*nil*“ zeigen. |
|  | Als nächster Schritt zeigt das neue Element auf das alte erste Element. |
|  | Jetzt muss das alte erste Element auf das neue Element zeigen. |
|  | Als letzter Schritt muss der „*ListPtr*“ auf das neue erste Element zeigen! |

## 4.3.3 Aufbau der verwendeten Dateien

### 4.3.3.1 Die Datei „config.ini“ für Highscore und Benutzereinstellungen

Die Datei „*config.ini*“ ist (wie schon in Kap. 4.2.4.1 beschrieben) eine Textdatei, die nach dem Schema von Windows-Konfigurationsdateien (Ini-Dateien) aufgebaut ist. Sie besteht aus mehreren Abschnitten, in denen wiederum mehrere Schlüsselvariablen stehen, die alle jeweils einen Wert besitzen.
Der Aufbau der Datei sieht folgendermaßen aus:

| Die Abschnitte der Benutzereinstellungen | Der Abschnitt für die Highscore |
|---|---|
| [General]<br>LevelTime=45<br>Lives=3<br><br>[Stars]<br>Count=50<br>MaxSpeed=6<br><br>[Control]<br>Left=37<br>Right=39<br>Fire=32<br>Pause=80<br><br>[Enemy]<br>Points=100<br>IncPoints=100<br>Count=3<br>IncCount=2<br>Speed=15<br>IncSpeed=2<br>Fire=10<br>IncFire=2 | [Highscore]<br>PlayerName1=Bender<br>PlayerPoints1=10000<br>PlayerName2=Fry<br>PlayerPoints2=9000<br>PlayerName3=Nibbler<br>PlayerPoints3=8000<br>PlayerName4=Amy Wong<br>PlayerPoints4=7000<br>PlayerName5=Hermes Conrad<br>PlayerPoints5=6000<br>PlayerName6=Leela<br>PlayerPoints6=5000<br>PlayerName7=Dr. Zoidberg<br>PlayerPoints7=4000<br>PlayerName8=Zapp Brannigan<br>PlayerPoints8=3000<br>PlayerName9=Prof. Farnsworth<br>PlayerPoints9=2000<br>PlayerName10=Slurm Mc Kenzie<br>PlayerPoints10=1000 |

### 4.3.3.2 Die Dateien für Spielgrafiken

Der Aufbau der Dateien für Spielgrafiken ist typisiert und kann nicht mit einem Editor geöffnet werden.
Jede gültige Datei, die Spielgrafiken enthält, besteht mindestens aus vier Bitmap-Blöcken, die unbedingt in der Reihenfolge „*Spieler, Spieler-Schuss, Gegner-Schuss, Gegner*“ vorliegen müssen. Nach den vier Blöcken können (wie rechts angedeutet) weitere, beliebig viele Blöcke mit Gegner-Grafiken folgen.

| | |
|---|---|
| Block 1 | Grafik des Spielers |
| Block 2 | Grafik des Spieler-Schusses |
| Block 3 | Grafik des Gegner-Schusses |
| Block 4 | Grafik des 1. Gegner |
| Block 5 | Grafik des 2. Gegner |
| Block 6 | Grafik des 3. Gegner |
| .<br>:<br>. | .<br>:<br>. |
| Block n | Grafik des n. Gegner |

Jeder einzelne dieser Bitmap-Blöcke besitzt folgenden Aufbau:

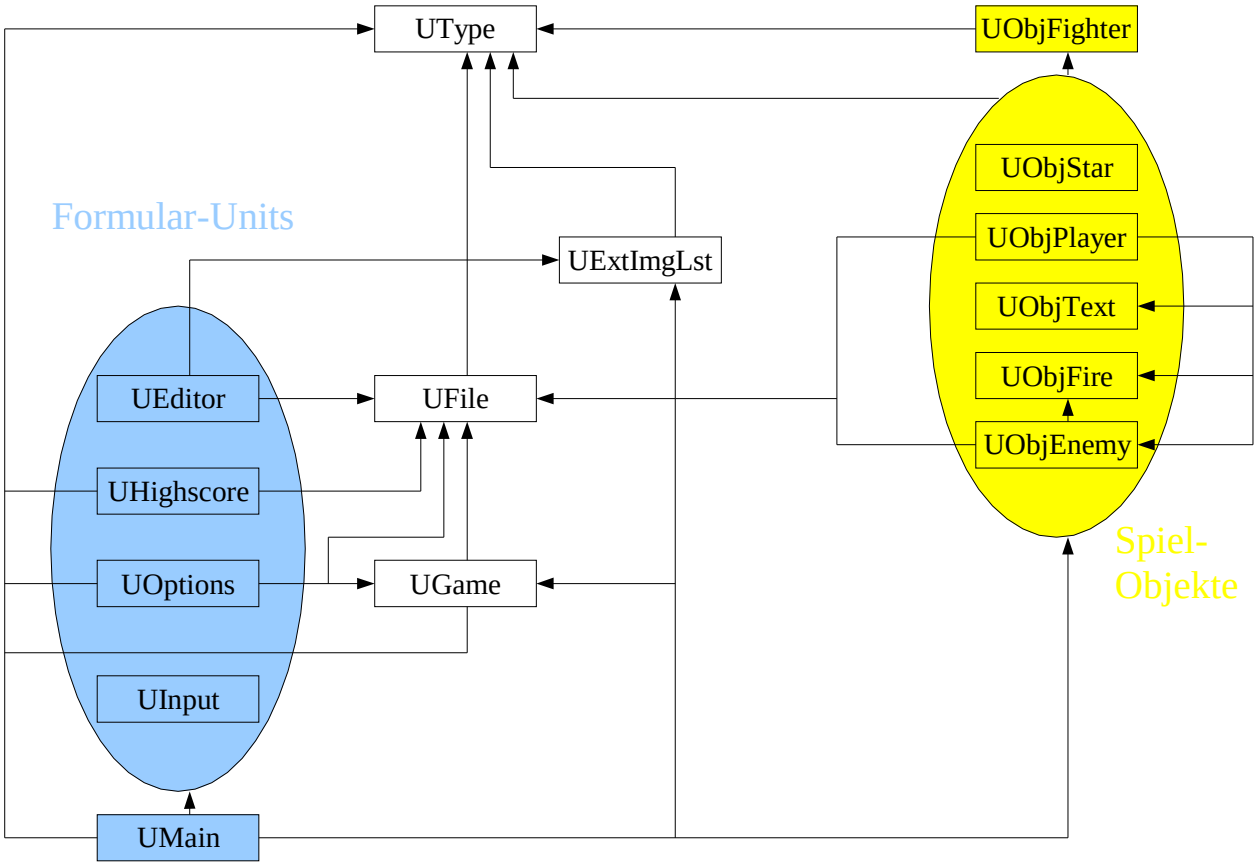
| **Bytes:** | **Datentyp:** | **Inhaltliche Bedeutung:** |
|---|---|---|
| Byte 0 | Byte | Das erste Byte enthält die Prozent-Information (wie viel % das Bild auf dem Spielfeld einnehmen soll). |
| Byte 1-8 | Int64 | StreamSize: Die nächsten acht Bytes enthalten die Größe des darauf folgenden Bitmap-Streams. Diese Angabe muss mitgespeichert werden, da ein Laden ohne Größeninformationen des Bitmap-Streams sonst unmöglich wäre. |
| Byte 9-(StreamSize+8) | TMemoryStream | Hier ist die eigentliche Grafik als „*Stream*“ gespeichert. |

## *4.4 Programmorganisationsplan*

### 4.4.1 Übersicht der Zugriffe der eigenen Units untereinander

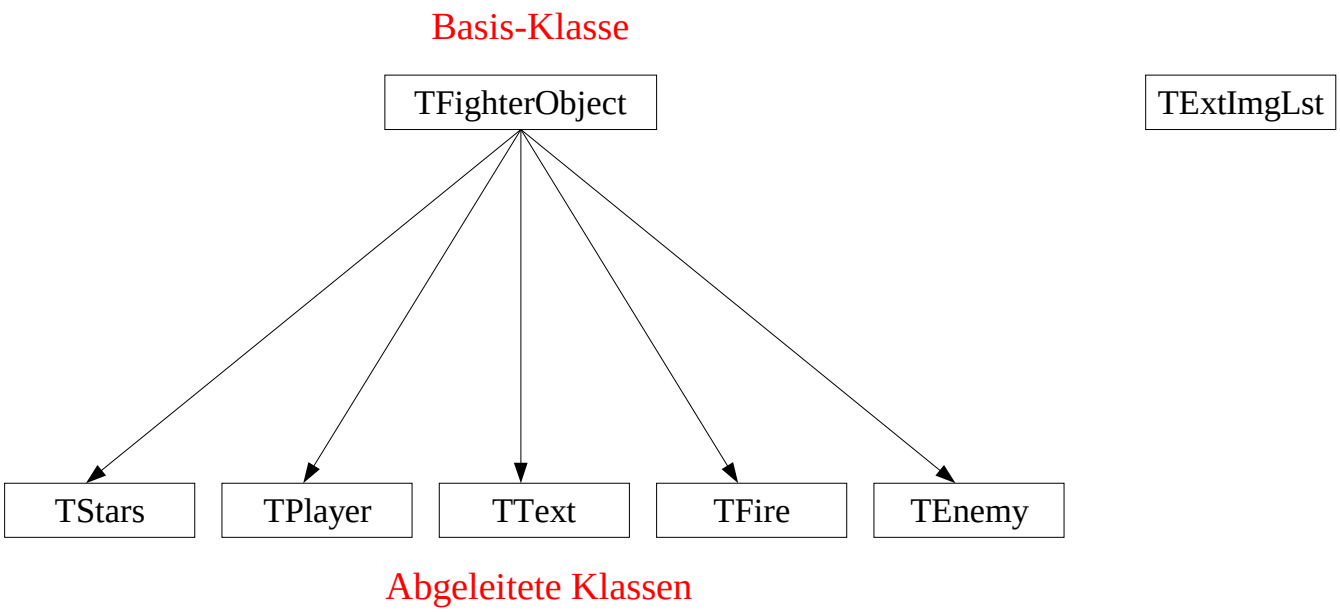
| **Unit** | **Aufgerufen Units im Interface-Teil** | **Aufgerufene Units im Implementation-Teil** | **Beschreibung** |
|---|---|---|---|
| UMain | UType, UObjPlayer, UObjEnemy, UObjStar, UObjFire, UObjText, UExtImglst | UEditor, UInput, UHighscore, UFile, UGame, UOptions | Hauptformular-Unit, die das Spielfeld, die Spiel-Infos und das Hauptmenü darstellt |
| UObjEnemy | UType, UObjFighter, UObjFire, UExtImgLst | UFile | Dient zur Anzeige und Bewegung der Gegner-Raumschiffe |
| UObjFighter | UType | | Basis-Klasse: Stellt Routinen zur Listen-Bearbeitung etc. |
| UObjFire | UType, UObjFighter | | Dient zur Anzeige von Explosionen |
| UObjPlayer | UType, UObjFighter, UObjEnemy, UObjText, UObjFire, UExtImgLst | UFile | Dient zur Anzeige und Bewegung des Spieler-raumschiffs |
| UObjStar | UType, UObjFighter | | Dient zur Anzeige und Bewegung der Sterne |
| UObjText | UType, UObjFighter | | Dient zur Anzeige von Texten |
| UOptions | UType | UGame, UFile | Formular-Unit des Optionen-Dialogs |
| UType | | | Enthält alle Datentypen und Konstanten |
| UEditor | UType, UExtImgLst | UFile | Stellt den Editor der Spiel-Grafiken |
| UExtImgLst | UType | | Stellt Routinen für Bilderlisten |
| UFile | UType, UExtImgLst | | Stellt Routinen zur Dateibearbeitung |
| UGame | UType | UFile | Stellt einige Routinen für den Spielverlauf |
| UHighscore | UType | UFile | Anzeige der Highscore-Liste |
| UInput | | | Bietet Formular, wo der Spieler seinen Namen eingeben kann |

### 4.4.2 Grafische Übersicht der Abhängigkeiten




### 4.4.3 Klassendiagramm der eigenen Klassen

Alle Spiel-Objekt-Klassen sind von einer Basis-Klasse (TFighterObject) abgeleitet. Außerdem gibt es die Klasse „*TExtImgLst*“, die als Ergänzung zu Borlands „*TImageList*“ gedacht ist, die auch mit Bitmaps unterschiedlicher Größe umgehen kann.



## *4.5 Programmtests*

Zunächst werden alle kritischen Dateioperationen mit der Datei „*config.ini*“ getestet.

| **Testfall** | **Erwartetes Ergebnis** | **Erzieltes Ergebnis** |
|---|---|---|
| „*Config.ini*“ ist schreibgeschützt. Versuch sie zu durch Start eines Neuen Spiels zu Laden | Keine Fehlermeldung. Spiel sollte starten! | Keine Fehlermeldung. Spiel startet. |
| „*Config.ini*“ ist schreibgeschützt. Versuch sie zu durch Aufruf der Benutzereinstellungen zu Laden | Keine Fehlermeldung. Dialog sollte mit den Werten angezeigt werden. | Keine Fehlermeldung. Dialog wird angezeigt. |
| „*Config.ini*“ ist schreibgeschützt. Versuch sie zu in den Benutzereinstellungen zu Ändern | Es sollte gefragt werden, ob der Dateischutz entfernt werden soll. Nur dann kann fehlerfrei gespeichert werden. | Fehlermeldung: „Soll der Schreibschutz der Datei: C:\test\config.ini entfernt werden?“<br>Klick auf „*Ja*“ entfernt Schreibschutz, speichert die Änderungen, Klick auf „*Nein*“ gibt Fehlermeldung: „Fehler beim Speichern der Benutzereinstellungen in Datei: C:\test\config.ini“. |
| „*Config.ini*“ ist nicht vorhanden. Versuch sie durch Start eines Neuen Spiels zu starten | Es soll ein Fehlermeldung kommen, die darauf hinweist, dass die Datei nicht existiert. Das Spiel sollte trotzdem starten können! | Fehlermeldung: „C:\test\config.ini nicht gefunden! Verwende Standard-Werte!“. Spiel startet mit Standard-Werten. |
| „*Config.ini*“ ist nicht vorhanden. Versuche sie über die Benutzereinstellungen zu Laden und zu Ändern. | Das Laden sollte mit den Standard-Werten geschehen. Beim Speichern sollte anschließend kein Fehler passieren! | Fehlermeldung: „C:\test\config.ini nicht gefunden! Verwende Standard-Werte!“. Danach wird das Formular geladen. Beim Speichern tritt kein Fehler auf. |
| „*Config.ini*“ enthält fehlerhaften Inhalt. Versuch sie über die Benutzereinstellungen zu Laden und zu Speichern. | Beim Laden sollten die fehlerhaften Abschnitte einfach durch Standardwerte ergänzt werden. Beim Speichern wird die Datei dann mit korrektem Inhalt abgelegt. | Die Einstellungen konnten fehlerfrei geladen werden (die fehlerhaften Abschnitte wurden durch Standard-Werte ersetzt). Das Speichern ist fehlerfrei. |
| „*Config.ini*“ ist schreibgeschützt. Versuch die Highscore zu Laden. | Dies sollte ohne Fehler funktionieren. | Keine Fehlermeldung. Die Highscore wird angezeigt. |
| „*Config.ini*“ ist nicht vorhanden. Versuch die Highscore zu Laden. | Das Laden sollte mit den Standard-Werten geschehen. | Fehlermeldung: „C:\test\config.ini nicht gefunden! Verwende Standard-Werte!“. Danach wird die Highscore mit Standard-Werten angezeigt! |

Im Folgenden werden alle kritischen Dateioperationen mit der Datei „*default.dat*“ getestet.

| **Testfall** | **Erwartetes Ergebnis** | **Erzieltes Ergebnis** |
|---|---|---|
| „*Default.dat*“ ist schreibgeschützt. Versuch sie zu durch Start eines Neuen Spiels zu Laden | Es sollte gefragt werden, ob der Dateischutz entfernt werden soll. Nur dann kann das Spiel fehlerfrei starten. | Fehlermeldung: „Soll der Schreibschutz der Datei: C:\test\default.dat entfernt werden?“<br>Klick auf „*Ja*“ entfernt Schreibschutz, Spiel startet.<br>Klick auf „*Nein*“ gibt Fehlermeldung: „Fehler beim Laden der Spiel-Grafiken, das Spiel kann ohne Bilder nicht starten.“. |
| „*Default.dat*“ ist schreibgeschützt. Versuch sie im Editor zu Laden, die Bilder zu Ändern und die Datei abzuspeichern. | Es sollte gefragt werden, ob der Dateischutz entfernt werden soll. Nur dann kann der Editor die Bilder Laden. Andernfalls sollten leere Standardbilder erstellt werden. | Fehlermeldung: „Soll der Schreibschutz der Datei: C:\test\default.dat entfernt werden?“<br>Klick auf „*Ja*“ entfernt Schreibschutz, Editor zeigt die Bilder an.<br>Klick auf „*Nein*“ ergibt die Fehlermeldung: „Fehler beim Laden der Spiel-Grafiken aus Datei: C:\test\default.dat. Lege vier leere Standard Bilder als Grafiken an!“ Es werden vier Standardbilder erzeugt und angezeigt. |
| „*Default.dat*“ ist nicht vorhanden. Versuch ein neues Spiel zu starten. | Es sollte eine Fehlermeldung kommen, die darauf hinweist, dass die Datei nicht vorhanden ist und das Spiel darum nicht starten kann. | Fehlermeldung: „Fehler: Datei C:\test\default.dat existiert nicht! Fehler beim Laden der Spiel-Grafiken, das Spiel kann ohne Bilder nicht starten.“ |
| „*Default.dat*“ ist nicht vorhanden. Versuch im Editor neue Standard-Bilder zu erzeugen. | Es sollte eine Fehlermeldung kommen, die darauf hinweist, dass die Datei nicht vorhanden ist. Anschließend sollte das Bearbeiten mit 4 leeren Standardbildern möglich sein. | Fehlermeldung: „Fehler: Datei C:\test\default.dat existiert nicht! Fehler beim Laden der Spiel-Grafiken aus Datei: C:\test\default.dat. Lege vier leere Standard Bilder als Grafiken an!“ Es werden vier Standardbilder erzeugt und angezeigt, das bearbeiten ist möglich. |
| „*Default.dat*“ ist fehlerhaft. Versuch ein neues Spiel zu starten. | Es sollte eine Fehlermeldung erscheinen, die dem Anwender mitteilt, dass die Bilder nicht geladen werden konnten und das neue Spiel somit nicht gestartet werden kann. | Fehlermeldung: „Fehler beim Laden der Spiel-Grafiken, das Spiel kann ohne Bilder nicht starten.“ |

Im Folgenden werden andere Dateien mit Bilderlisten im Editor getestet.

| **Testfall** | **Erwartetes Ergebnis** | **Erzieltes Ergebnis** |
|---|---|---|
| Versuch eine schreibgeschützte Bilderliste („*Oldschool.dat*“) im Editor zu öffnen. | Es sollte gefragt werden, ob der Dateischutz entfernt werden soll. Nur dann kann der Editor die Bilder Laden. Andernfalls sollten leere Standardbilder erstellt werden. | Fehlermeldung: „Soll der Schreibschutz der Datei: C:\test\Oldschool.dat entfernt werden?“<br>Klick auf „*Ja*“ entfernt Schreibschutz, Editor zeigt die Bilder an.<br>Klick auf „*Nein*“ ergibt die Fehlermeldung: „Fehler beim Laden der Spiel-Grafiken aus Datei: C:\test\Oldschool.dat. Lege vier leere Standard Bilder als Grafiken an!“<br>Es werden vier Standardbilder erzeugt und angezeigt. |
| Versuch eine fehlerhafte Bilderliste („*Defekt.dat*“) im Editor zu Laden. | Es sollte eine Fehlermeldung ausgegeben werden. Anschließend soll das Bearbeiten mit vier Standardbildern möglich sein. | Fehlermeldung: „Fehler beim Laden der Spiel-Grafiken aus Datei: C:\test\defekt.dat. Lege vier leere Standard Bilder als Grafiken an!“<br>Es werden vier Standardbilder erzeugt und angezeigt. |
| Versuch eine Bilderliste im Editor zu Speichern, dabei soll eine schreibgeschützte Datei überschrieben werden. | Zunächst soll darauf hingewiesen werden, dass bereits eine Datei unter dem gewählten Namen existiert. Möchte der Anwender die Datei ersetzen, so sollte anschließend gefragt werden, ob der Schreibschutz entfernt werden soll. Nur dann kann der Editor die Bilderliste speichern. | Fehlermeldung: „C:\test\Schreibschutz.dat besteht bereits. Möchten Sie sie ersetzen?“<br>Nach Klick auf „*Ja*“: Fehlermeldung: „Soll der Schreibschutz der Datei: C:\test\Schreibschutz.dat entfernt werden?“ Speichern funktioniert. |
| Versuch eine Bilderliste im Editor auf einem ungültigen (nur Lesezugriff) Medium zu speichern. | Es sollte eine Fehlermeldung ausgegeben werden, dass das Speichern nicht möglich war. | Fehlermeldung: „Fehler beim Speichern der Spiel-Grafiken in Datei: Z:\test.dat“ |

Im Folgenden werden fehlerhafte Benutzereingaben getestet.

| **Testfall** | **Erwartetes Ergebnis** | **Erzieltes Ergebnis** |
|---|---|---|
| In den Benutzereinstellungen wird für alle Steuerfunktionen die gleiche Taste definiert. | Der Anwender sollte darauf hingewiesen werden, dass seine gewählten Tasten unzulässig sind. Danach soll er dies korrigieren können. | Fehlermeldung: „Ihre Eingaben sind fehlerhaft, bitte keine gleichen Steuerungstasten verwenden!“.<br>Der Dialog wird nicht geschlossen, der Anwender kann seinen Fehler korrigieren! |

| | | |
|---|---|---|
| In den Benutzereinstellungen wird von Hand in der Levelzeit ein zu hoher Wert eingegeben (z.B. 80 Sek.) | Der fehlerhafte Wert sollte nicht mit abgespeichert werden. Er sollte vorher korrigiert werden. | Alle SpinEdit-Komponenten haben einen bestimmten Werte-Bereich (Minimum, Maximum). Sobald sie den Focus verlieren, wird ein fehlerhafter Wert korrigiert. |
| Im Editor wird von Hand ein ungültiger Wert z.B. im Feld „Breite“ eingetragen. | Der fehlerhafte Wert sollte nicht mit abgespeichert werden. Er sollte vorher korrigiert werden. | Alle SpinEdit-Komponenten haben einen bestimmten Werte-Bereich (Minimum, Maximum). Sobald sie den Focus verlieren, wird ein fehlerhafter Wert korrigiert. |

Nun werden die Zeigerroutinen (der Listen) getestet.

| **Testfall** | **Erwartetes Ergebnis** | **Erzieltes Ergebnis** |
|---|---|---|
| Test, ob alle Zeiger der Fighter-Listen nach dem Beenden des Programms wieder freigegeben werden. Dies wird mehrfach getestet. Mit Starten eines neuen Spiels, Pausieren, erneutes Starten, etc., um Überprüfen zu können, ob in allen Fällen der Speicher wieder freigegeben wird! | Zum Test wird eine Variable „*PtrDebug*“ mit jedem „*New()*“ um Eins erhöht und mit jedem „*Dispose()*“ ums Eins erniedrigt. Beim Beenden des Programms muss „*PtrDebug*“ also in jedem Fall Null sein, wenn alle reservierten Speicherbereiche ordentlich wieder freigegeben wurden!!! | In alle getesteten Konstellationen: PtrDebug = 0 |
| Test, ob alle Zeiger der Bilderlisten nach Beenden des Programms wieder freigegeben werden. | Zum Test wird eine Variable „*LstDebug*“ mit jedem „*New()*“ um Eins erhöht und mit jedem „*Dispose()*“ ums Eins erniedrigt. Beim Beenden des Programms muss „*LstDebug*“ also in jedem Fall Null sein, wenn alle reservierten Speicherbereiche ordentlich wieder freigegeben wurden!!! | In alle getesteten Konstellationen: LstDebug = 0 |
| Test, ob Zeiger der Bilderlisten auch beim fehlerhaften Laden wieder freigegeben werden. | Dieser Test ist ähnlich wie der vorige. Auch hier wird erwartet, dass „*LstDebug*“ am Ende immer Null ist. | In alle getesteten Konstellationen: LstDebug = 0 |
| Anmerkung: Die beiden Zählvariablen „LstDebug“ und „PtrDebug“ sind in den Units „*UType, UMain, UFighterObject und UExtImgLst*“ auskommentiert! | | |

# 5. Anhang

## *5.1 Original Aufgabenstellung Delphi Vordiplomsaufgabe für IIs und WIs*