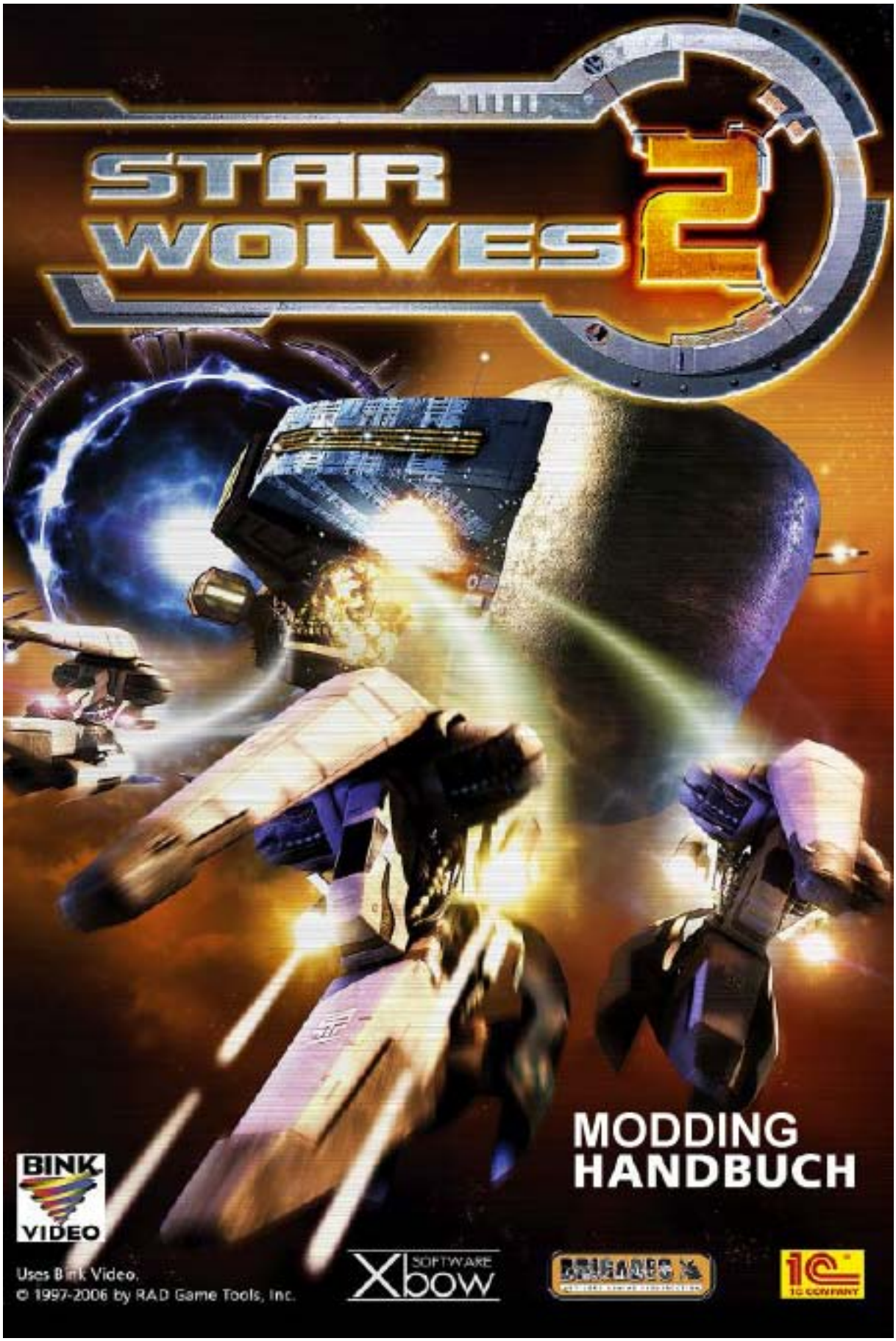

STAR WOLVES 2
MODDING HANDBUCH
BINK VIDEO
Uses Bink Video.
© 1997-2006 by RAD Game Tools, Inc.
SOFTWARE
Xbow
1C COMPANY

Inhaltsverzeichnis

1 Einleitung Star Wolves Modding Handbuch

1.1 Wichtige Dateien

Sämtliche spielrelevanten Dateien befinden sich im Hauptordner **…\Star Wolves 2\DATA**

- Im Ordner **AI** befinden sich Konfigurationseinstellungen zum **PC –Verhalten** sowie einige **globale Definitionen** die das Gameplay betreffen und die **Verkaufspreismultiplikatoren**
- Der Ordner Environment enthält die Einstellungen für die einzelnen Sternensystemarten. Jede Datei definiert eine Sternenanzahl, eine Farbe, eine Sonne, Nebelfarben etc
- Beim **Game** Ordner handelt es sich um den ersten Anlaufhafen wenn es um das editieren von Schiffen, Quests, Hauptmissionsscripts, Waffen oder Ausrüstung geht. Er beinhaltet auch die .xml Dateien, die die globale Sektorkarte, die anheuerbaren Rekruten und die Gegner beschreiben und deren Stärke definieren
- **GUI** beinhaltet die Einstellungen und das Outfit aller Dialogfenster im Spiel
- Im Ordner **LocData** befinden sich die lokalisierten Sprachausgaben und Texte jeder Spielversion. Er enthält Unterordner für die Texturen und Schriftarten, sowie für jeden einzelnen Sektor (**Locations**), dem Tutorial (**Levels**), die **Quests** und der Sprachausgabe selbst (**voice**)
- **Music** ist der Ordner, der die während dem Spiel hörbare Musik enthält, aufgeschlüsselt in **Ambient** (normale Umgebungsmusik), **Battle** (Kampfmusik) und **inter** (Musik für Hauptmenü (nehme ich an)). Außerdem enthält er eine Konfigurationsdatei, die Fade-Einstellungen beinhaltet
- Der Ordner OBJECT ist für jene interessant, die etwas an den Im Spiel vorhandenen 3D-Objekten ändern wollen. Jedes Gittermodell ist hier im Format IMD gespeichert
- **PFX** enthält sämtliche Daten, um die Spezialeffekte im Spiel wie Explosionen, Triebswerksschleier oder Waffenfeuer darstellen zu können.
- Die Konfigurations-Files im Ordner **PreCache** geben für jeden Sektor an, welche Modelle oder Grafikfiles eingelesen werden müssen, bevor ein Rendern stattfinden kann
- Nun kommen wir zu einem weiteren wichtigen Modding-Ordner. **Scripts** definieren sämtliche Aktionen und Orte in diesem Spiel. Alle Abläufe und Aktivitäten, ob das Docken des Mutterschiffs an einer Station, die erhältlichen Waren, die Häufigkeit von Händlern, der Verkehr in den Sektoren, die vorhandenen Stationen, Missionen, Formationen, etc. alles findet sich in diesen Unterordnern. Jeden einzelnen zu erklären würde den Rahmen dieser Aufschlüsselung jedoch sprengen. Ich werde das in naher Zukunft aber sicher nachholen
- **SOUND** beinhaltet die Töne der Spezialeffekte wie Waffenfeuer und Explosionen
- Der Ordner **TEXTURE** enthält alle im Spiel auftretenden Texturen
- Im Ordner **Video** sind die im Spiel vorkommenden, nicht gescripteten Filme zu finden während die Dateien im Ordner **VideoDescriptions** Konfigurationswerte für jedes Video enthalten
- XMLSchema zuletzt enthält die Outfit-Dateien für die Anzeige der .xml-Dateien

1.2 Dateitypen

Um Star Wolves 2 erfolgreich verändern zu können, müssen mehrere unterschiedliche Dateiarten verändert werden. Die meisten können mit einem normalen Texteditor oder dem Wordpad geöffnet werden. Einige Dateien erfordern jedoch eigene Tools um vernünftig editiert werden zu können.

1.2.1 .ini Dateien

Die wohl bekannteste Dateiart der wir auf unserem Ausflug begegnen ist die .ini Datei. Geöffnet wird sie mit einem normalen Texteditor. Praktisch jeder der oben genannten Ordner verfügt über eine oder mehrere dieser Files. Sie dienen dazu, Konfigurationswerte für das Spiel zu definieren oder um andere Dateien in diesem Ordner näher zu beschreiben.
Der folgende Abschnitt zeigt ein solches File:

```
[Global parameters]

;äèñòàíöèÿ ïðåñëåäîâàíèÿ
ChaseDistance = 130

;äèñòàíöèÿ áëèæíåãî áîÿ
CloseCombatDistance = 60

; ïåðèîäè÷íîñòü îáíîâëåíèÿ ñïèñêà âèäèìûõ êîðàáëåé (â ñåêóíäàõ)
GroupVisibleUpdatePeriod = 1
; ïåðèîäè÷íîñòü í÷èùåíèÿ ñïèñêà âèäèìûõ êîðàáëåé îò DeadPointer (â ñåêóíäàõ)
groupVisibleListCheckTime = 2

;âêëþ÷åíû ëè ñòðåëêè íà öåëè
showPointers = 1

;Ìèíèìàëüíàÿ äëèíà ñòðåëêè ïîñëå êîòîðîé îíà íà÷èíàåò ïëàâíî èñ÷åçàòü
MinFadeDistHelper = 10
; äèñòàíöèÿ, â ïðåäåëàõ êîòîðîé àòàêîâàííûé êîðàáëü áóäåò ïðîèãíîðèðîâàí ó äðóãèõ
distanceToReportAggressiveBehaviour = 200
```

Abbildung 1-1: Ausschnitt aus der originalen Datei AI.ini

Die vielen zahllosen Zeilen unleserlicher Text sind Kommentare, die die Programmierer in ihrer grenzenlosen Weisheit zu jeder Zeile geschrieben haben. Sie würden erklären, was der Wert, der darunter steht, im Spiel bewirkt. Leider verstehe ich ihre Sprache nicht☺.

Diese Kommentarzeilen werden durch einen „**;**“ gekennzeichnet. Zur leichteren Übersichtlichkeit kann man diese weglöschen. Das Spiel nimmt dadurch keinen Schaden.

```
[Global parameters]
ChaseDistance = 130
CloseCombatDistance = 60
GroupVisibleUpdatePeriod = 1
groupVisibleListCheckTime = 2
showPointers = 1
MinFadeDistHelper = 10
distanceToReportAggressiveBehaviour = 200
collisionAvoidanceDistance = 14
diameterVisibleMax = 20
diameterVisibleMin = 3
kinematicGunsRangeIncrement = 30;
frameWidthMin = 5
splashDamageMaxRadius = 8;
ExpandDistance = 100;
wayCheckPeriod = 1;
HoldTimeLag = 0.5f;
```

Abbildung 1-2: Ausschnitt aus der übersichtlicheren AI.ini

Nun zum Aufbau der .ini Dateien. Die Datei besteht aus Kategorien und Variablen, die in diesen Kategorien zugewiesen werden. Kategorien werden durch die eckigen Klammern symbolisiert. [Global parameter] zeigt in unserem Beispiel, dass alle Werte darunter bis zur nächsten Kategorieschrift den globalen Spielverlauf betreffen, also in allen Sektoren zu jeder Zeit gelten.

1.2.2 .env Dateien

Die .env Dateien sind gleich aufgebaut wie die .ini Dateien und können daher auch mit dem gleichen Texteditor geöffnet werden. Sie dienen als Konfigurationsfiles für die im Spiel sichtbaren Sternenhimmel.

Diese Datei ändert aber noch nicht den Sternenhimmel. Zu jeder dieser Files gehört auch eine .str Datei, von der ich noch nicht weiß wie sie aufgebaut ist.

1.2.3 .xml Dateien

Neben den Scripts die zweithäufigste Datei der wir auf unserer Reise über den Weg laufen. Auch sie kann mit einem Texteditor geöffnet werden, jedoch würden sich an dieser Stelle bereits XML-Editoren anbieten. Peter's XML Editor ist einer der vielen, die das Editieren solcher Dateien wesentlich vereinfachen, da er die einzelnen Subkategorien übersichtlich anordnet und überschaubar macht. Auch Visual Studio kann solche Dateien besser gruppiert darstellen.

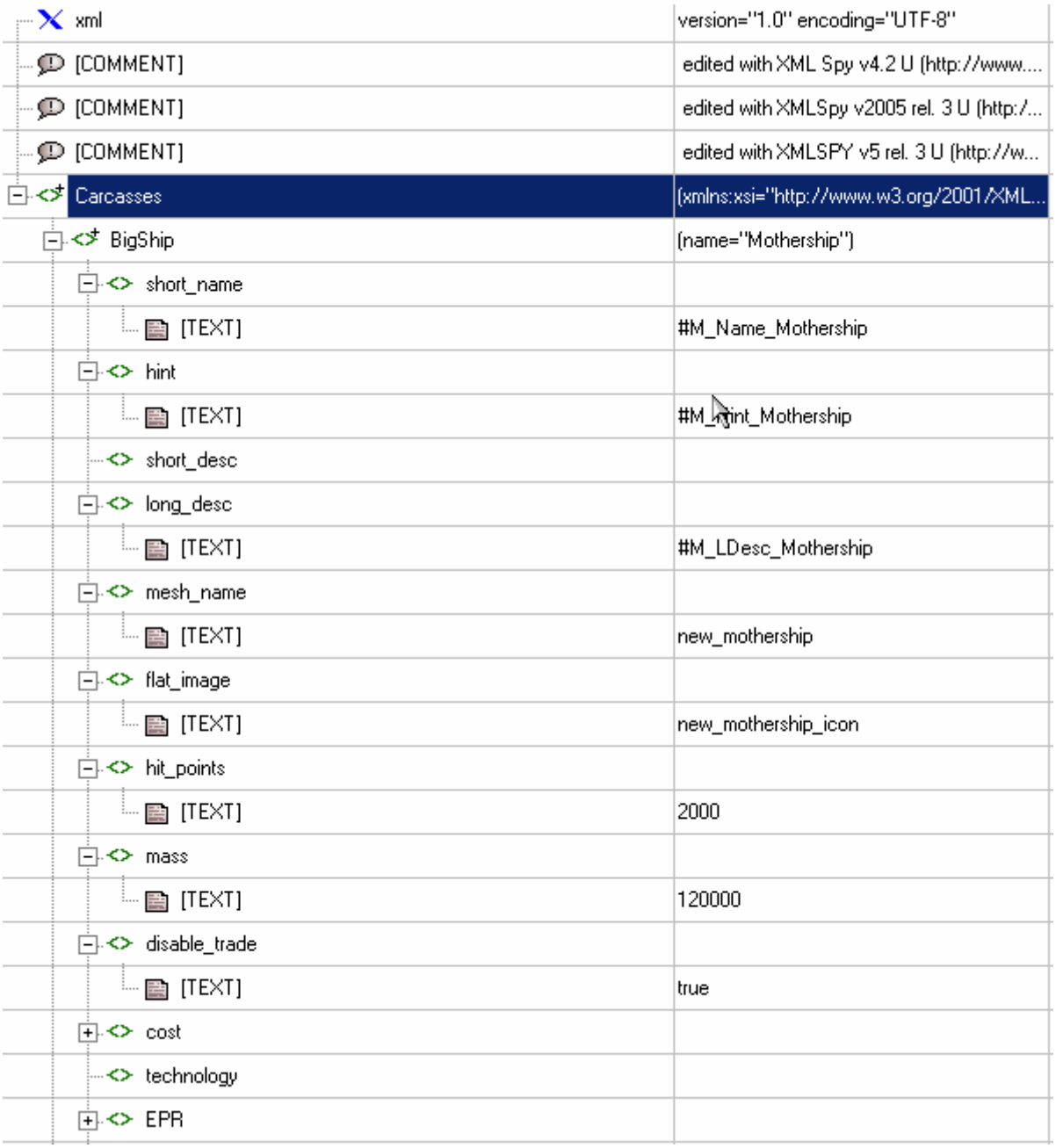


Abbildung 1-3: Die Datei carcasses.xml in Peter's XML Editor

Abbildung 1-4: Visual Studio Ansicht der carcasses.xml

Zur Not kann jedoch auch ein normaler Texteditor verwendet werden. Bevorzugt sollten dabei aber jene werden, die eine Syntaxmarkierung ermöglichen. Als Beispiel siehe die nächste Abbildung.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- edited with XML Spy v4.2 U (http://www.xmlspy.com) by Ba-k (ZonaWarez.
<!-- edited with XMLSpy v2005 rel. 3 U (http://www.altova.com) by Andrew De
<!-- edited with XMLSPY v5 rel. 3 U (http://www.xmlspy.com) by Aliende (XBC
<Carcasses xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance" xsi:noName
  <BigShip name="Mothership">
    <short_name>#M_Name_Mothership</short_name>
    <hint>#M_Hint_Mothership</hint>
    <short_desc/>
    <long_desc>#M_LDesc_Mothership</long_desc>
    <mesh_name>new_mothership</mesh_name>
    <flat_image>new_mothership_icon</flat_image>
    <hit_points>2000</hit_points>
    <mass>120000</mass>
    <disable_trade>true</disable_trade>
    <cost>99999</cost>
    <technology/>
    <EPR>60</EPR>
    <explosion_script>Big Ship</explosion_script>
    <work_sound/>
    <silence/>
    <max_energy>250</max_energy>
    <energy_restore>3</energy_restore>
    <max_speed>9</max_speed>
```

Abbildung 1-5: Texteditor-Ansicht der carcasses.xml

Die Bezeichnungen innerhalb der spitzen Klammern (Tags) definieren den eingeschlossenen Wert. Ein Tag kann auch mehrere Werte enthalten, also in Subtags unterteilt werden. Ein Tag reicht immer vom Anfangstag <Tag> bis zum Endtag </Tag>, einzige Ausnahme bilden leere Tags, da reicht eine einzelne Zeile <Tag/>.

1.2.4 .def und .dat Dateien

.def Dateien können einfach mit einem Texteditor geöffnet werden. Sie beinhalten Definitionen und Daten für die anzuzeigenden Dialogfenster.

1.2.5 .ogg und .wav Dateien

Hierbei handelt es sich um Musikdateien, die im Windows PCM WAV Format und im OGG Vorbis Format gespeichert sind. Jedes bessere Bearbeitungsprogramm (Cool Edit Pro 2, Adobe Audition, etc) kann beide Formate öffnen und beliebig konvertieren.

1.2.6 .loc, .txt, .txtcache und .lst Dateien

Die Location-Dateien beinhalten die lokalisierten Texte des Spiels. Darunter fallen die angezeigten Texte bei den Dialogen, die Bezeichnungen der Waffen und Schiffe und die Namen der Skills. Es handelt sich um einfache Texte und die Dateien benötigen kein spezielles Programm um diese zu öffnen. Um Zeilenumbrüche im angezeigten Text eingeben zu können muss das Zeichen | (ALT GR + <) gesetzt werden.

```
1  LOCALE_FILE
2  FILETYPE STANDART
3  #M_Name_Excalibur = Excalibur
4  #M_Hint_Excalibur = Excalibur|Vielseitig einsetzbares Kampfschiff
5  #M_SDesc_Excalibur = Vielseitig einsetzbares Kampfschiff.
6  #M_LDesc_Excalibur = Das Excalibur wurde ursprünglich als taktisches Kontrc
7  #M_Name_Excalibur_wolf = Korsaren-Excalibur
8  #M_Hint_Excalibur_wolf = Korsaren-Excalibur|Aufgerüstetes Excalibur
9  #M_SDesc_Excalibur_wolf = Vielseitig einsetzbares Kampfschiff.
10 #M_LDesc_Excalibur_wolf = Auf Befehl des Roten Korsaren wurde sein persönli
11 #M_Name_Hatchet = Machete
12 #M_Hint_Hatchet = Machete|Spezialkampfschiff
13 #M_SDesc_Hatchet = Spezialkampfschiff
14 #M_LDesc_Hatchet = Das ungewöhnliche Design dieses Schiffes ist auf seinen
15 #M_Name_Naginata = Naginata
16 #M_Hint_Naginata = Naginata|Leichter Bomber
17 #M_SDesc_Naginata = Leichter Bomber
18 #M_LDesc_Naginata = Dieser Bomber der ersten Generation beeindruckte durch
```

Abbildung 1-6: Inhalt der m_carcasses.loc

1.2.7 .xsd Dateien

Die Styles für die .xml-Dateien Sie müssen nicht verändert werden. Eine Änderung könnte höchstens die .xml-Dateien unbrauchbar machen die damit verlinkt sind. Editierbar sind sie jedoch mit einem einfachen Texteditor.

1.2.8 .script Dateien

Auch .script Dateien sind einfache Textdateien, jedoch mit dem Unterschied, dass sie nicht einfach nur Wertezuweisungen enthalten, sondern ganze Funktionen definieren, die dann nach einander aufgerufen und abgearbeitet werden. Einige der .script Dateien (vor allem im Ordner „include“) definieren sämtliche Funktionen, die im Spiel benötigt werden. Von Zufallsflotten bis hin zum Handel, von den Piloten bis hin zu den Stationen. Andere wie die im Ordner „Locations“ definierend en Aufbau von Sektoren durch Aufrufen mehrerer Funktionen und Prozeduren.

Auf die einzelnen Funktionen und deren Arbeitsweise komme ich später zu sprechen.

1.2.9 .dds Dateien

DDS-Dateien sind Texturdateien, die mit Photoshop und dem NVidia-DDS-Plugin aufgemacht werden können. Als Beispiel: Das Icon für das Mutterschiff liegt unter \TEXTURE\Interface\Carcass in den Ordnern 32, 64 und 128 und heißt new_mothership_icon. Dies findet man heraus wenn man sich die carcasses.xml in Abbildung 1-3 ansieht. Im Tag flat_image steht der Name der DDS Datei. Diese 3 Dateien kann man nun öffnen.

Abbildung 1-7: Photoshopansicht der Icon-Datei new_mothership_icon

Die Ordnernamen 128, 64 und 32 stehen für die Auflösung des Bildes (W = H = 128 Pixel). In diesem Ordner sind auch die Abbildungen jedes Schiffes enthalten. Je nach Fraktion ist die Farbe des Schiffes auf dem Icon anders. Hier am Beispiel einer EvilEye.

Abbildung 1-8: Unterschiedliche Fraktionsfarben bei einer EvilEye

2 Scripting System – Funktionen

Um die Abläufe im Spiel verstehen zu können sind im folgenden Abschnitt die einzelnen Include-Dateien genauer unter die Lupe genommen worden. Jede von ihnen behandelt einen anderen Part aus dem Spiel viele Dinge lassen sich durch Skripts verändern, beispielsweise das Auftreten von Zufallsflotten oder Kämpfe zwischen den einzelnen Parteien. Sogar ganze Sektoren können so geschaffen werden.

Um die Scriptdateien jedoch erfolgreich ändern oder gar neu erstellen zu können, müssen wir erst die Werkzeuge, in diesem Fall die Funktionen kennen lernen, die das Scripting-System uns bietet.

Im den folgenden Codetabellen sind Syntaxbefehle blau gekennzeichnet während Variablen grün hervorgehoben werden. Die Laufvariablen in Schleifen sind rot, Kommentarzeilen sind grau.

2.1 Trading und Cargo

Vier der Scriptdateien im Include-Ordner regeln alle Geschehnisse, die mit dem Finden, Stehlen und Handeln von Waren zu tun haben. Die Datei FloodTradeStations.script definiert, wann welche Waren zu haben sind, indem sie die Waffen, Raketen und Ausrüstungsgegenstände, genauso wie die Schiffe, in vier separate Listen aufteilt, welche dann mittels einer weiteren Funktion an die einzelnen Shops zugewiesen werden. Die Datei Create_shop.script hingegen regelt die Waren, die in Handelsschiffen zu finden sind. Zusätzlich enthält die Datei stocks.script einen Kostenfaktor für verschiedene Schwierigkeitsstufen. Das Script CostOfPirates.script regelt das Einkommen durch Kopfgelder.

2.1.1 FloodTradeStations.script

2.1.1.1 function InitShopList()

```
function InitShopList()
      ShopList =
      {"AREKO","EGLEON","GAEON","HEOS","LASAD","LIDAU","GESS","GENAU",
      "TRUR","TARNRED","DANAAST","HALLRAD","LIND","MAEN","NOE2","NOE1",
      "KRAON","DANSTAG","KRID","GENTIS","LENG","FIASAR","KONDEL","BAE",
      "RASPIAN","SVERES","DORIA","VILFOR","TIBBORG","KELIN","LIT","LERIGOR",
      "KASANO"};
end;
```

Diese Funktion initialisiert eine Liste, die die Namen sämtlicher Sektoren beinhaltet. Wenn ein neuer Sektor erstellt wird muss auch hier eine Eintragung mit dem Namen des Sektors gemacht werden um eine dortige Handelsstation auch betreiben zu können.

2.1.1.2 function CreateShopList()

```
function CreateShopList()
      ShopList =
      {"AREKO","EGLEON","GAEON","HEOS","LASAD","LIDAU","GESS","GENAU",
      "TRUR","TARNRED","DANAAST","HALLRAD","LIND","MAEN","NOE2","NOE1",
      "KRAON","DANSTAG","KRID","GENTIS","LENG","FIASAR","KONDEL","BAE",
      "RASPIAN","SVERES","DORIA","VILFOR","TIBBORG","KELIN","LIT","LERIGOR",
      "KASANO"};
      for i=1,getn(ShopList),1 do
            CreateShop(ShopList[i]);
      end;
end;
```

Diese Funktion baut aus der Shopliste einen Shop für jeden Sektor. Dieser Shop wird dann von Spiel als Dreh- und Angelpunkt für alle Handlungsaktionen genutzt. Auch hier muss ein neuer Sektor eingetragen werden soll er über ein Handelsterminal verfügen.

2.1.1.3 function CreateGoodList()

```
function CreateGoodList()
      ShipList1 =
      {"Excalibur_pl0",."Hatchet_pl0",."Naginata_pl0",."Yari_pl0",.
      "Brigand_pl0", "Hammerhead_pl0"};
      ShipList2 =
      {"Bident_pl0",."Cleaner_pl0",."Raptor_pl0",."Stormcrow_pl0",
      "TieFly_pl0"};

      ShipList3 =
      {"EvilEye_pl0",."Gunslinger_pl0",."Hrimturs_pl0",."Tiger_pl0",.
      "Trident_pl0"};

      ShipList4 =
      {"Scolm_0","Mataris_0","Bastard_0","Raven_0"};

      --ModuleList1 =
      --{"LRS1", "ECM1", "AMS1", "Stealth1", "CloD1", "LRRNB1", "RBot1",
      --"ShAmp1", "EngAmp1","EngBoost1", "RNBot1", "HC1", "VC1", "PC1",
      --"LC1", "CG1", "AC1", "CargoExpander1", "LRS1", "ECM1", "RBot1",
      --"BS_RBot1", "ShAmp1"};
      --
      --ModuleList2 =
      --{"LC2", "ECM2", "AMS1", "Stealth2", "CloD1", "LRRNB1", "RBot1",
      --"ShAmp2", "EngAmp2", "GunAmpPH", "EngBoost1", "RNBot1", "ALS1",
      --"HC1", "VC1","PC1", "LC2", "PLC1", "CG2", "AC2", "LRS2", "ECM2",
      --"RBot2", "BS_RBot2", "ShAmp2"};
      --
      --ModuleList3 = {"LRS3", "ECM3", "AMS2", "Stealth3", "CloD2",
      --"LRRNB2", "RBot2", "ShAmp3", "EngAmp3", "GunAmpPH", "EngBoost2",
      --"RNBot2", "ALS1", "HC2", "VC2", "PLC2", "AC3", "LRS3", "ECM3",
      --"BS_RBot3", "ShAmp3", "EngAmp1"};
      --
      --ModuleList4 = {"LRS4", "ECM4", "AMS2", "Stealth3", "CloD2",
      --"LRRNB2", "RBot2", "ShAmpPH", "EngAmp3", "EngBoost2", "RNBot2",
      --"ALS1", "HC2", "VC2", "PLC2", "RG1", "ASG", "BSL1", "EngAmp2"};

      ModuleList1 =
      {"HC1", "HC2", "VC1", "CG1", "CG2", "AC1", "MM", "DF", "SRM1",
      "LRM1", "T1", "Turret_BG1", "Turret_BG2", "Turret_BG3", "LRS1",
      "BS_LRS1", "ECM1", "ECM2", "AMS1", "Stealth1", "RBot1", "ShAmp1",
      "EngAmp1", "WBoost1", "EngBoost1", "RNBot1"};

      ModuleList2 =
      {"VC2", "PC1", "LC1", "AC2", "SRM2", "LRM2", "T2", "MIRV1",
      "Turret_BG4", "Turret_BG5", "Turret_BG7", "LRS2", "LRS3", "BS_LRS1",
      "ECM3", "AMS2", "BS_ECM1", "Stealth2", "CloD1", "LRRNB1", "RBot2",
      "ShAmp2", "BS_ShAmp1", "EngAmp2", "WBoost2", "EngBoost2", "RNBot2"};

      ModuleList3 =
      {"LC2", "PLC1", "AC3", "SRM3", "LRM3", "MIRV2", "Turret_BG6",
      "Turret_BG8", "LRS4", "ECM4", "BS_ECM2", "BS_ECM3", "Stealth3",
      "CloD2", "LRRNB2", "BS_RBot1", "BS_RBot2", "ShAmp3", "BS_ShAmp2",
      "EngAmp3", "ALS1", "CargoExpander1"};

      ModuleList4 =
      {"PLC2", "RG1", "Turret_BG9", "Turret_BG10", "Turret_BG11",
      "BS_LRS3", "BS_RBot3", "BS_ShAmp3"};
end;
```

Diese Funktion erstellt insgesamt 8 Listen, wobei die Zahl nach dem Namen die Generation widerspiegelt. Mit Hilfe dieser Listen fügen andere Funktionen die Geräte und Schiffe den Handelsoptionen bei, wenn die Erfahrungsstufe des Spielers dies ermöglicht.

2.1.1.4 function AddShipToAllShops(ShopList,GoodList,Chance,Black)

```
function AddShipToAllShops(ShopList,GoodList,Chance,Black)
      for i=1,getn(ShopList),1 do
            price = 1;
            if (Black==TRUE) then
                  Market=GetBlackMarket(ShopList[i]);
                  price=price*2;
            else
                  Market=GetWhiteMarket(ShopList[i]);
            end;
            for j=1,getn(GoodList),1 do
                  if (RandomBool(Chance)==TRUE) then
                        count = floor(random()*5)+1;
                        Market:ShipGoodsState(GoodList[j] , count,
                                    price, 0.7);
                  end;
            end;
      end;
end;
```

Durch diese Funktion wird jeder einzelne Shop aus der Shopliste nacheinander abgearbeitet und dabei festgelegt, welche Schiffe er aus der vorgegebenen Goods-Liste anbieten kann und ob es sich dabei um einen Schwarzmarkthandelsstützpunkt handelt oder nicht (Schwarzmarkt hätte höhere Chancen zu teureren Preisen zur Folge). Dadurch werden die einzelnen Märkte in den Sektoren definiert und deren Warenangebot ermittelt.

2.1.1.5 function AddModuleToAllShops(ShopList,GoodList,Chance,Black)

```
function AddModuleToAllShops(ShopList,GoodList,Chance,Black)
      for i=1,getn(ShopList),1 do
            price = 1;
            if (Black==TRUE) then
                  Market=GetBlackMarket(ShopList[i]);
                  price=price*2;
            else
                  Market=GetWhiteMarket(ShopList[i]);
            end;
            for j=1,getn(GoodList),1 do
                  if (RandomBool(Chance)==TRUE) then
                        count = floor(random()*11)+1;
                        Market:ModuleGoodsState(GoodList[j], count,
                                    price, 0.7);
                  end;
            end;
      end;
end;
```

Durch diese Funktion wird jeder einzelne Shop aus der Shopliste nacheinander abgearbeitet und dabei festgelegt, welche Module er aus der vorgegebenen Goods-Liste anbieten kann und ob es sich dabei um einen Schwarzmarkthandelsstützpunkt handelt oder nicht (Schwarzmarkt hätte höhere Chancen zu teureren Preisen zur Folge). Dadurch werden die einzelnen Märkte in den Sektoren definiert und deren Warenangebot ermittelt.

2.1.1.6 function RandomBool(Chance)

```
function RandomBool(Chance)
      result = floor(random()*100);
      if (Chance<result) then
            return FALSE;
      else
            return TRUE;
      end;
end;
```

RandomBool errechnet auf Basis des Chancen-Wertes ein positives (TRUE) oder negatives (FALSE) Ergebnis, indem sie den Wert mit einem Zufallswert vergleicht. Sollte die Chance größer sein, wird TRUE an das aufrufende Script zurückgegeben.

2.1.1.7 Globale Aktionen in der FloodTradeStations.script

```
InitShopList();
CreateGoodList();
```

In dieser Scriptdatei werden direkt die beiden Hauptfunktionen, die die Shop- und Warenlisten erstellen, direkt aufgerufen. Sie stehen also von Spielstart an zur Verfügung und müssen nicht separat in anderen Scripts definiert werden.

2.1.2 Create_shop.script

2.1.2.1 function CreateNormalShop(inv_ship,rank)

```
function CreateNormalShop(inv_ship,rank)
      local quantity;
      local i1;
      local i2;
      local level = 0;

      local InventoryModules =
      {{"Cargo_10","CG1", "HC1", "ECM1", "RNBot1", "Turret_BG1", "BS_LRS1",
      "BS_ECM1"},{"CG2", "LRS1", "EngBoost1", "Turret_BG2", "Turret_BG1",
      "BS_ShAmp1"},{"HC2", "AMS1", "RBot1", "LRRNB1", "WBoost1",
      "Turret_BG3", "BS_RBot1", "BS_LRS2", "BS_ECM2"},{"LC1", "VC1", "AC1",
      "ShAmp1", "RNBot1", "EngAmp1", "ECM2", "LRS2", "Stealth1",
      "Turret_BG5", "Turret_BG7"},{"PC1", "CloD1", "WBoost2", "RNBot2",
      "Turret_BG4", "Turret_BG8", "BS_ShAmp2"},{"VC2", "AC2", "ECM3",
      "EngAmp2", "EngBoost2", "ShAmp2", "Turret_BG10", "BS_LRS3"},{"LC2",
      "AMS2", "Stealth2", "LRS3", "Turret_BG9", "Turret_BG6", "BS_ECM3",
      "BS_RBot2"},{"AC3", "LRRNB2", "RBot2", "CloD2", "Turret_BG11",
      "BS_RBot3", "BS_ShAmp3"}};

      local AdditionalMissiles =
      {{"MIRV1","MIRV2","DF", "SRM1"},{"MM", "LRM1"},{"SRM2"},{"LRM2"},
      {"T1"},{"SRM3"},{"LRM3"},{"MIRV1"}};

      local InventoryShips =
      {{"Brigand_pl0","Hatchet_pl0","Hatchet_pl0","Hatchet_pl0"},
      {"Naginata_pl0","Yari_pl0"},{"Excalibur_pl0","Hammerhead_pl0"},
      {"Stormcrow_pl0"},{"Raptor_pl0", "Bident_pl0"},{"Cleaner_pl0"},
      {"EvilEye_pl0"},{"Trident_pl0"}};

      for level, levelmodules in InventoryShips do
            for i1, modulename in levelmodules do
                  if (rank > level * 2 - 3 - RAND(5)) then
                        if(rank > level * 2) then
                              quantity = RAND(3) + 1;
                        else
                              quantity = 1;
                        end;
                        if (quantity > 0) then
                              for i2 = 1, quantity do
                                    inv_ship:AddShipToInventory(modulename,
                                                100);
                              end;
                        end;
                  end;
            end;
      end;
```

```
	for level, levelmodules in InventoryModules do
		for i1, modulename in levelmodules do
			if (rank > level * 2 - 3 - RAND(5)) then
				if(rank > level * 2) then
					quantity = RAND(5) + 1;
				else
					quantity = RAND(2) + 1;
				end;
				if (quantity > 0) then
					inv_ship:AddModuleToInventory(modulename,
							quantity);
				end;
			end;
		end;
	end;

	for level, levelmodules in AdditionalMissiles do
		for i1, modulename in levelmodules do
			if (rank > level * 2 - 6) then
				if(rank > level * 2 - 2) then
					quantity = RAND(15) + 2;
				else
					quantity = RAND(5) + 1;
				end;
				if (quantity > 0) then
					inv_ship:AddModuleToInventory(modulename,
							quantity);
				end;
			end;
		end;
	end;
end;
```

Ein eher langes Script, das dem Zielschiff (inv_ship) einen Inhalt in den Frachtraum zaubert. Dieser ist Abhängig vom Rang und vom Level, letzteres spiegelt eine weitere Generationsaufteilung wieder (diesmal aber 6 statt 4 Generationen). Errechnet das System eine Menge größer 0 wird diese auch in das Zielschiff integriert.

2.1.2.2 function CreateBlackShop(inv_ship, rank)

```
function CreateBlackShop(inv_ship,rank)
	local quantity;
	local i1;
	local i2;
	local level = 0;

	local InventoryModules =
	{{"CG1", "HC1", "CG2", "DF", "SRM1", "MM", "LRM1", "ECM1", "LRS1",
	"ShAmp1", "EngBoost1", "RNBot1", "Turret_BG1", "BS_LRS1", "BS_ECM1",
	"Turret_BG2", "Turret_BG1", "BS_ShAmp1"},{"AC1", "HC2", "VC1",
	"SRM2", "ECM2", "AMS1", "LRRNB1", "EngAmp1", "WBoost1", "Turret_BG3",
	"BS_RBot1", "BS_LRS2", "BS_ECM2"},{"AC2", "PC1", "LC1", "VC2",
	"LRM2", "T1", "LRS2", "Stealth1", "CloD1", "ShAmp2", "RNBot2",
	"ECM3", "AMS2", "Stealth2", "RBot1", "WBoost2", "Turret_BG5",
	"Turret_BG7"},{"LC2", "AC3", "SRM3", "LRS3", "LRRNB2", "ShAmp3",
	"EngAmp2", "EngBoost2", "Turret_BG4", "Turret_BG8", "BS_ShAmp2"},
	{"PLC1", "LRM3", "ECM4", "Stealth3", "CloD2", "Turret_BG10",
	"BS_LRS3"},{"PC2", "RG1", "MIRV1", "T2", "LRS4", "RBot2", "EngAmp3",
	"Turret_BG9", "Turret_BG6", "BS_ECM3", "BS_RBot2"},{"PLC2", "MIRV2",
	"Turret_BG11", "BS_RBot3", "BS_ShAmp3"}};


	local InventoryShips =
	{{"Brigand_pl0", "Hatchet_pl0", "Naginata_pl0", "Yari_pl0"},
```

```
      {"Excalibur_pl0", "Stormcrow_pl0", "Hammerhead_pl0"},
      {"Raptor_pl0","TieFly_pl0"},{"Cleaner_pl0","Bident_pl0"},
      {"EvilEye_pl0"},{"Trident_pl0"},{"Tiger_pl0","Hrimturs_pl0"},
      {"Gunslinger_pl0"}};

      for level, levelmodules in InventoryShips do
            for i1, modulename in levelmodules do
                  if (rank > level * 2 - 3 - RAND(5)) then
                        if(rank > level * 2) then
                              quantity = RAND(3);
                        else
                              quantity = RAND(2);
                        end;
                        if (quantity > 0) then
                              for i2 = 1, quantity do
                                    inv_ship:AddShipToInventory(modulename,
                                                1);
                              end;
                        end;
                  end;
            end;
      end;

      for level, levelmodules in InventoryModules do
            for i1, modulename in levelmodules do
                  if (rank > level * 2 - 3 - RAND(5)) then
                        if(rank > level * 2) then
                              quantity = RAND(4);
                        else
                              quantity = RAND(2);
                        end;
                        if (quantity > 0) then
                              inv_ship:AddModuleToInventory(modulename,
                                          quantity);
                        end;
                  end;
            end;
      end;
end;
```

Dieses Script fügt, wie schon das Script vorhin, dem Zielschiff (inv_ship) einen Inhalt in den Frachtraum ein. Dieser ist Abhängig vom Rang und vom Level, letzteres spiegelt eine weitere Generationsaufteilung wieder (diesmal aber 6 statt 4 Generationen). Errechnet das System eine Menge größer 0 wird diese auch in das Zielschiff integriert. BlackShop bedeutet jedoch, dass die Items wesentlich seltener sind und daher auch teurer. Die Liste deckt sich nicht mit der aus 2.1.8. manche Schiffe beispielsweise können nur in BlackShops gefunden werden.

2.1.2.3 function CreateUpgradeShop(inv_ship, rank)

```
function CreateUpgradeShop(inv_ship,rank)
      local quantity;
      local i1;
      local i2;
      local level = 0;

      local InventoryModules =
      {{"Turret_BG1", "BS_LRS1", "BS_ECM1"},{"Turret_BG2", "Turret_BG1",
      "BS_ShAmp1"},{"Turret_BG3", "BS_RBot1", "BS_LRS2", "BS_ECM2"},
      {"Turret_BG5", "Turret_BG7"},{"Turret_BG4", "Turret_BG8",
      "BS_ShAmp2"},{"Turret_BG10", "BS_LRS3"},{"Turret_BG9", "Turret_BG6",
      "BS_ECM3",  "BS_RBot2" },{"Turret_BG11", BS_RBot3", "BS_ShAmp3"}};

      for level, levelmodules in InventoryModules do
            for i1, modulename in levelmodules do
```

```
                  if (rank > level * 2 - 3 - RAND(5)) then
                        if(rank > level * 2) then
                              quantity = RAND(4) + 1;
                        else
                              quantity = RAND(2);
                        end;
                        if (quantity > 0) then
                              inv_ship:AddModuleToInventory(modulename,
                                          quantity);
                        end;
                  end;
            end;
      end;
end;
```

Wie schon die Scripte 2.1.8 und 2.1.9 fügt auch diese Funktion dem Zielschiff einen Inhalt hinzu. Jedoch handelt es sich bei dem Inhalt um Upgrades für das Mutterschiff.

2.1.2.4 Traderfunktionen

```
function NormalTraider(level_challenge)
      n_trader = CreateCarcass("NormalMarketStation", Vector3(1000, 0, 0),
                        Vector3(0,0,1));
      CreateNormalShop(n_trader, level_challenge);
      return n_trader.id;
end;

function BlackTraider(level_challenge)
      b_trader = CreateCarcass("BlackMarketStation", Vector3(-1000, 0, 0),
                        Vector3(0,0,1));
      CreateBlackShop(b_trader, level_challenge);
      return b_trader.id;
end;

function UpgradeTraider(level_challenge)
      u_trader = CreateCarcass("NormalMarketStation", Vector3(0, 0, 1000),
                        Vector3(0,0,1));
      CreateUpgradeShop(u_trader, level_challenge);
      return u_trader.id;
end;
```

Dieser Trader-Funktionen erstellen Handelsstationen an einem definierten Ort und geben deren ID zurück. Der erste Vector gibt die Position an, der zweite die Rotation. Der Text definiert den Typ des Objektes, also deren Trefferpunkte, Textur, etc. Siehe dazu die Datei Carcasses.xml.

2.1.3 Stocks.script

2.1.3.1 function CostOfStocks(count,level_difficult,base_cost)

```
function CostOfStocks(count,level_difficult,base_cost)
      return count * level_difficult * base_cost / 110;
end;
```

Ein einfachen Script das den Wert einer Ware anhand folgender Formel ermittelt:

$$X = n \cdot \frac{Basiskosten \cdot Schwierigkeitsgrad}{110}$$

2.1.3.2 function SellCostOfStocks(level_difficult,base_cost)

```
function SellCostOfStocks(level_difficult,base_cost)
      local cost = base_cost;

      -- 1 episode
      if (level_difficult == 4) then
            cost = cost * 2.6;
```

```
	end;
	if (level_difficult == 5) then
		cost = cost * 3.9;
	end;

	-- 2 episode
	if (level_difficult == 6) then
		cost = cost * 4.2;
	end;
	if (level_difficult == 7) then
		cost = cost * 4.4;
	end;
	if (level_difficult == 8) then
		cost = cost * 4.8;
	end;

	-- 3 episode
	if (level_difficult == 9) then
		cost = cost * 6.1;
	end;
	if (level_difficult == 10) then
		cost = cost * 2.8;
	end;
	if (level_difficult == 11) then
		cost = cost * 6.8;
	end;

	-- 4 episode
	if (level_difficult > 11) then
		cost = cost * 8.2;
	end;
	return cost;
end;
```

Je nach Fortschritt im Spiel werden alle Kosten mit einem Faktor multipliziert.

2.1.3.3 function StocksName()

```
function StocksName()
	return "stocks";
end;
```

Diese Funktion gibt den Namen der Waren wieder. Jedoch scheint sie mir momentan erfrischend sinnlos zu sein, da sie immer nur denselben Wert zurückgibt.

2.1.4 CostOfPirates.script

2.1.4.1 function CostOfHead(pilot_name)

```
function CostOfHead(pilot_name)
	local price = 0;
	if(pilot_name == "PirateRookie") then
		price = 100;
	end;

	if(pilot_name == "PirateTough") then
		price = 300;
	end;

	if(pilot_name == "PirateCaptain") then
		price = 1000;
	end;

	return price;
end;
```

Dieses Script definiert, wie viel die Piloten durch Kopfgelder verdienen. Da hier nur Piraten aufgeführt sind verdient der Spieler auch nur an Ihnen. Jedoch kann man andere Pilotennamen angeben und so auch Kopfgeld für andere Abschüsse geltend machen.

2.2 Verkehr im All und Zufallskontakte

Es ist immer wieder ein schönes Gefühl in der Weite des Alls auf andere Piloten zu treffen oder in der Ferne ein Portal aufflammen zu sehen. Wer weiß vielleicht wartet ja fette Beute auf unsere Söldner oder wir laufen direkt in die Arme eines nicht gerade freundlich gesinnten Schlachtschiffes samt Geleitschutz…

All dies ist vielleicht noch ein wenig Zukunftsmusik aber SW2 verfügt durchaus über die Werkzeuge, diese Träume Realität werden zu lassen. Alles was wir brauchen steckt in den Dateien Create_flight.script, Create_flight_2.script und RandomContacts.script

2.2.1 Create_flight.script

2.2.1.1 function CreateTeamFlight(teamName, signName, location_x,location_y,location_z, number, skill, purpose, strength, special)

```
function CreateTeamFlight(teamName, signName, location_x, location_y,
                  location_z, number, skill, purpose, strength, special)

      if (teamName=="Pirates1") then
            skill = 3;
            team=CreateTeam("Pirates");
            groupName="group_.."Pirates"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreatePirateFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Pirates2") then
            skill = 4;
            team=CreateTeam("Pirates");
            groupName="group_.."Pirates"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreatePirateFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Pirates3") then
            skill = 5;
            team=CreateTeam("Pirates");
            groupName="group_.."Pirates"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreatePirateFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Berserk1") then
```

```
            skill = 3;
            team=CreateTeam("Berserk");
            groupName="group_.."Berserk"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreateBerserkFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Berserk2") then
            skill = 5;
            team=CreateTeam("Berserk");
            groupName="group_.."Berserk"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreateBerserkFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Vks1") then
            skill = 5;
            team=CreateTeam("Vks");
            groupName="group_.."Vks"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreatePatrolFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      if (teamName=="Vks1") then
            skill = 5;
            team=CreateTeam("Vks");
            groupName="group_.."Vks"..signName;
            ___group=team:CreateGroup(groupName);
            flight = CreatePatrolFlight("___group", signName,
                              Vector3(location_x,location_y,location_z),
                              number, skill, purpose, strength, special);
            return flight["id"];
      end;

      ------------------------------------------------------------------------

      return nil;
end;
```

Anscheinend die „Lite“-Version des CreateTeamFlight. Vor allem die beiden Funktionen für die VKS Patrouillen sind komplett identisch. Ich nehme an, dass dieses Script vorerst als einstweilige Lösung gedacht war, jetzt jedoch die Hauptfunktion tragen muss. Prinzipiell funktioniert sie ganz einfach. Über teamName definiert man welche Partei man will, daraufhin stellt das Script die Stärke über den Wert skill ein und erstellt ein Team und eine Group. Weiters ruft die Funktion eine weitere Unterfunktion auf, die anhand der restlichen Parameter die einzelnen Schiffe und Piloten erstellt und ihnen eine Aufgabe zuteilt.

2.2.1.2 function CreatePirateFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreatePirateFlight(group, sign, location, number, skill, purpose,
                  strength, special,orient)
      local flight;
      local Pilots =    {{"PirateRookie", "PirateRookie"},
                        {"PirateTough", "PirateRookie"},
                        {"PirateTough", "PirateTough"},
                        {"PirateCaptain", "PirateTough"},
                        {"PirateCaptain", "PirateCaptain"}};

      local Fighters =
      {{"Brigand_pir1", "Brigand_pir1", "Yari_pir1", "Excalibur_pir1",
      "Cleaner_pir1", "Gunslinger_pir1"},Brigand_pir2", "Yari_pir2",
      "Excalibur_pir2", "Stormcrow_pir1", "EvilEye_pir1", "Trident_pir1"},
      {"Naginata_pir1", "Hammerhead_pir1", "Naginata_pir2","Naginata_pir3",
      "Bident_pir1", "Hrimturs_pir1"},{"Hammerhead_pir1","Hammerhead_pir1",
      "Hammerhead_pir2", "Bident_pir2", "Bident_pir3", "Hrimturs_pir1"},
      {"Hammerhead_pir1", "Hammerhead_pir2", "Hammerhead_pir3",
      "Bident_pir1", "Bident_pir3", "Hrimturs_pir1"}};

      local Specials =
      {{"Hatchet_pir1", "Hatchet_pir1", "Hatchet_pir1", "Raptor_pir1",
      "Trident_pir2"},{"Hatchet_pir2", "Hatchet_pir2", "Hatchet_pir2",
      "Raptor_pir2", "EvilEye_pir2"},{"Hatchet_pir3", "Hatchet_pir3",
      "Hatchet_pir3", "TieFly_pir1", "Tiger_pir1"}};

      local ship;
      local pilot;
      local grp = getglobal(group);

      local orient = orient or Vector3(0,0,1);

      if(sign == nil) then
            SLOG("Sign is not defined! Cannot create flight!");
            return FALSE;
      end;

      if(number < 1 or number > 6) then
            SLOG("Number is not in [1;6]! Cannot create flight!");
            return FALSE;
      end;

      if(skill < 1 or skill > 5) then
            SLOG("Skill is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(purpose < 1 or purpose > 5) then
            SLOG("Purpose is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(strength < 1 or strength > 5) then
            SLOG("Strength is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(grp) then
            flight = grp:CreateFlight("flight_"..sign);
      else
            SLOG("No Group to create flight!");
```

```
            return FALSE;
      end;

      setglobal("flight_"..sign, flight);

      for i = 1, number do
            if(i == number and special > 0) then
                  ship = CreateCarcass(Specials[special][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
            else
                  if(i == 1) then
                        ship = CreateCarcass(Fighters[purpose][strength +
                              1], Vector3( 0, 0, (i-1)*5) + location);
                  else
                        ship = CreateCarcass(Fighters[purpose][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
                  end;
            end;

            if(i == 1) then
                  pilot = CreatePilot(Pilots[skill][1]);
            else
                  pilot = CreatePilot(Pilots[skill][2]);
            end;

            ship:AssignPilot(pilot);
            flight:AddShip(ship);
            setglobal("ship_"..sign.."_"..i, ship);
            setglobal("pilot_"..sign.."_"..i, pilot);
      end;
      flight:SetFormation("shipFormation");
      flight:SetOrientation(orient);

      return flight;
end;
PIRATE_FLIGHT = "CreatePirateFlight";
```

Dieses Script ist für die Piratenflotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Rookies besteht. Bei skill = 4 jedoch wäre der Anführer ein Captain, alle anderen im Team Tough.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.3 function CreateBerserkFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreateBerserkFlight(group, sign, location, number, skill, purpose,
strength, special,orient)
      local flight;
      local Pilots =    {{"BerserkBot", "BerserkBot"},
                        {"BerserkLeader", "BerserkBot"},
                        {"BerserkLeader", "BerserkLeader"},
                        {"BerserkLeader", "BerserkLeader"},
                        {"BerserkLeader", "BerserkLeader"}};

      local Fighters =
```

```
	{{"Breeze_ber1", "Breeze_ber1", "Breeze_leader_ber1",
	"Breeze_leader_ber1", "Breeze_elite_ber1", "Breeze_elite_ber1"},
	{"Breeze_ber1", "Breeze_ber1", "Breeze_leader_ber1",
	"Breeze_leader_ber1", "Breeze_elite_ber1", "Breeze_elite_ber1"},
	{"Cancer_ber1", "Cancer_ber1", "Cancer_leader_ber1",
	"Cancer_leader_ber1", "Cancer_elite_ber1", "Cancer_elite_ber1"},
	{"Cancer_ber2", "Cancer_ber2", "Cancer_leader_ber2",
	"Cancer_leader_ber2", "Cancer_elite_ber2", "Cancer_elite_ber2"},
	{"Cancer_ber3", "Cancer_ber3", "Cancer_leader_ber3",
	"Cancer_leader_ber3", "Cancer_elite_ber3", "Cancer_elite_ber3"}};

	local Specials =
	{{"Breeze_ber1", "Breeze_ber1", "Breeze_leader_ber1",
	"Breeze_leader_ber1", "Breeze_elite_ber1", "Breeze_elite_ber1"},
	{"Breeze_ber1", "Breeze_ber1", "Breeze_leader_ber1",
	"Breeze_leader_ber1", "Breeze_elite_ber1", "Breeze_elite_ber1"},
	{"Cancer_ber1", "Cancer_ber1", "Cancer_leader_ber1",
	"Cancer_leader_ber1", "Cancer_elite_ber1", "Cancer_elite_ber1"},
	{"Cancer_ber2", "Cancer_ber2", "Cancer_leader_ber2",
	"Cancer_leader_ber2", "Cancer_elite_ber2", "Cancer_elite_ber2"},
	{"Cancer_ber3", "Cancer_ber3", "Cancer_leader_ber3",
	"Cancer_leader_ber3", "Cancer_elite_ber3", "Cancer_elite_ber3"}};

	local ship;
	local pilot;
	local grp = getglobal(group);

	local orient = orient or Vector3(0,0,1);

	if(sign == nil) then
		SLOG("Sign is not defined! Cannot create flight!");
		return FALSE;
	end;

	if(number < 1 or number > 6) then
		SLOG("Number is not in [1;6]! Cannot create flight!");
		return FALSE;
	end;

	if(skill < 1 or skill > 5) then
		SLOG("Skill is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(purpose < 1 or purpose > 5) then
		SLOG("Purpose is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(strength < 1 or strength > 5) then
		SLOG("Strength is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(grp) then
		flight = grp:CreateFlight("flight_"..sign);
	else
		SLOG("No Group to create flight!");
		return FALSE;
	end;

	setglobal("flight_"..sign, flight);
```

```
	for i = 1, number do
		if(i == number and special > 0) then
			ship = CreateCarcass(Specials[special][strength],
					Vector3( 0, 0, (i-1)*5) + location);
		else
			if(i == 1) then
				ship = CreateCarcass(Fighters[purpose][strength +
						1], Vector3( 0, 0, (i-1)*5) + location);
			else
				ship = CreateCarcass(Fighters[purpose][strength],
						Vector3( 0, 0, (i-1)*5) + location);
			end;
		end;

		if(i == 1) then
			pilot = CreatePilot(Pilots[skill][1]);
		else
			pilot = CreatePilot(Pilots[skill][2]);
		end;

		ship:AssignPilot(pilot);
		flight:AddShip(ship);
		setglobal("ship_"..sign.."_"..i, ship);
		setglobal("pilot_"..sign.."_"..i, pilot);
	end;
	flight:SetFormation("shipFormation");
	flight:SetOrientation(orient);

	return flight;
end;
BERSERK_FLIGHT = "CreateBerserkFlight";
```

Dieses Script Dieses Script ist für die Bererkerflotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Bots besteht. Bei skill = 2 jedoch wäre der Anführer ein Leader, alle anderen im Team Bots.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.4 function CreatePatrolFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreatePatrolFlight(group, sign, location, number, skill, purpose,
strength, special, orient)
	local flight;
	local Pilots =	{{"PatrolPilot", "PatrolPilot"},
			{"PatrolOfficer", "PatrolPilot"},
			{"PatrolOfficer", "PatrolOfficer"},
			{"PatrolAce", "PatrolOfficer"},
			{"PatrolAce", "PatrolAce"}};

	local Fighters =
	{{"Brigand_ino1", "Brigand_ino1", "Yari_ino1", "Excalibur_ino1",
	"Stormcrow_ino1", "EvilEye_ino1"},{"Yari_ino1", "Yari_ino2",
	"Excalibur_ino1", "Excalibur_ino2", "Stormcrow_ino1", "Tiger_ino1"},
	{"Naginata_ino1", "Naginata_ino1", "Naginata_ino2", "Naginata_ino2",
```

```
"Bident_ino1", "Hrimturs_ino1"},{"Hammerhead_ino1", "Naginata_ino2",
"Naginata_ino2", "Bident_ino1", "Bident_ino2", "Hrimturs_ino2"},
{"Naginata_ino1", "Hammerhead_ino1", "Naginata_ino3",
"Naginata_ino3", "Bident_ino1", "Hrimturs_ino1"}};

local Specials =
{{"Hatchet_ino1", "Hatchet_ino1", "Hatchet_ino1", "Raptor_ino1",
"Trident_ino1"},{"Hatchet_ino2", "Hatchet_ino2", "Hatchet_ino2",
"Raptor_ino2", "Trident_ino2"},{"Hatchet_ino3", "Hatchet_ino3",
"Hatchet_ino3", "Raptor_ino3", "Trident_ino3"}};

local ship;
local pilot;
local grp = getglobal(group);

local orient = orient or Vector3(0,0,1);

if(sign == nil) then
      SLOG("Sign is not defined! Cannot create flight!");
      return FALSE;
end;

if(number < 1 or number > 6) then
      SLOG("Number is not in [1;6]! Cannot create flight!");
      return FALSE;
end;

if(skill < 1 or skill > 5) then
      SLOG("Skill is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(purpose < 1 or purpose > 5) then
      SLOG("Purpose is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(strength < 1 or strength > 5) then
      SLOG("Strength is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(grp) then
      flight = grp:CreateFlight("flight_"..sign);
else
      SLOG("No Group to create flight!");
      return FALSE;
end;

setglobal("flight_"..sign, flight);

for i = 1, number do
      if(i == number and special > 0) then
            ship = CreateCarcass(Specials[special][strength],
                        Vector3( 0, 0, (i-1)*5) + location);
      else
            if(i == 1) then
                  ship = CreateCarcass(Fighters[purpose][strength +
                              1], Vector3( 0, 0, (i-1)*5) + location);
            else
                  ship = CreateCarcass(Fighters[purpose][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
```

```
                  end;
            end;

            if(i == 1) then
                  pilot = CreatePilot(Pilots[skill][1]);
            else
                  pilot = CreatePilot(Pilots[skill][2]);
            end;

            ship:AssignPilot(pilot);
            flight:AddShip(ship);
            setglobal("ship_"..sign.."_"..i, ship);
            setglobal("pilot_"..sign.."_"..i, pilot);
      end;
      flight:SetFormation("shipFormation");
      flight:SetOrientation(orient);

      return flight;
end;
PATROL_FLIGHT = "CreatePatrolFlight";
```

Dieses Script Dieses Script ist für die Vks- oder Patrouillen-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Pilots besteht. Bei skill = 4 jedoch wäre der Anführer ein Ace, alle anderen im Team Officers.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.5 function CreateNavyFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreateNavyFlight(group, sign, location, number, skill, purpose,
strength, special, orient)
      local flight;
      local Pilots =    {{"NAVYPilot", "NAVYPilot"},
                        {"NAVYOfficer", "NAVYPilot"},
                        {"NAVYOfficer", "NAVYOfficer"},
                        {"NAVYAce", "NAVYOfficer"},
                        {"NAVYAce", "NAVYAce"}};

      local Fighters =
      {{"TieFly_nav4", "TieFly_nav4", "TieFly_nav4", "Stormcrow_nav1",
      "Stormcrow_nav1", "Stormcrow_nav1"},{"Cleaner_nav1", "Cleaner_nav1",
      "Tiger_nav1", "Tiger_nav1", "Tiger_nav2", "Tiger_nav2"},
      {"Bident_nav1", "Bident_nav1", "Bident_nav2", "Bident_nav2",
      "Hrimturs_nav3", "Hrimturs_nav3"},{"Bident_nav3", "Bident_nav3",
      "Bident_nav5", "Bident_nav5", "Hrimturs_nav2", "Hrimturs_nav2"},
      {"Bident_nav4", "Bident_nav4", "Bident_nav4", "Hrimturs_nav2",
      "Hrimturs_nav2", "Hrimturs_nav2"}};

      local Specials =
      {{"TieFly_nav1", "TieFly_nav1", "Trident_nav1", "Trident_nav1",
      "Trident_nav1"},{"TieFly_nav2", "TieFly_nav2", "Trident_nav2",
      "Trident_nav2", "Trident_nav2"},{"TieFly_nav3", "TieFly_nav3",
      "Trident_nav3", "Trident_nav3", "Trident_nav3"}};

      local ship;
```

```
	local pilot;
	local grp = getglobal(group);

	local orient = orient or Vector3(0,0,1);

	if(sign == nil) then
		SLOG("Sign is not defined! Cannot create flight!");
		return FALSE;
	end;

	if(number < 1 or number > 6) then
		SLOG("Number is not in [1;6]! Cannot create flight!");
		return FALSE;
	end;

	if(skill < 1 or skill > 5) then
		SLOG("Skill is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(purpose < 1 or purpose > 5) then
		SLOG("Purpose is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(strength < 1 or strength > 5) then
		SLOG("Strength is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(grp) then
		flight = grp:CreateFlight("flight_"..sign);
	else
		SLOG("No Group to create flight!");
		return FALSE;
	end;

	setglobal("flight_"..sign, flight);

	for i = 1, number do
		if(i == number and special > 0) then
			ship = CreateCarcass(Specials[special][strength],
					Vector3( 0, 0, (i-1)*5) + location);
		else
			if(i == 1) then
				ship = CreateCarcass(Fighters[purpose][strength +
					1], Vector3( 0, 0, (i-1)*5) + location);
			else
				ship = CreateCarcass(Fighters[purpose][strength],
					Vector3( 0, 0, (i-1)*5) + location);
			end;
		end;

		if(i == 1) then
			pilot = CreatePilot(Pilots[skill][1]);
		else
			pilot = CreatePilot(Pilots[skill][2]);
		end;

		ship:AssignPilot(pilot);
		flight:AddShip(ship);
		setglobal("ship_"..sign.."_"..i, ship);
```

```
            setglobal("pilot_"..sign.."_"..i, pilot);
      end;
      flight:SetFormation("shipFormation");
      flight:SetOrientation(orient);

      return flight;
end;
NAVY_FLIGHT = "CreateNavyFlight";
```

Dieses Script Dieses Script ist für die Navy-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Pilots besteht. Bei skill = 4 jedoch wäre der Anführer ein Ace, alle anderen im Team Officers.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.6 function CreateUSSFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreateUSSFlight(group, sign, location, number, skill, purpose,
strength, special, orient)
      local flight;
      local Pilots =    {{"SnkPilot", "SnkPilot"},
                        {"SnkOfficer", "SnkPilot"},
                        {"SnkOfficer", "SnkOfficer"},
                        {"SnkAce", "SnkOfficer"},
                        {"SnkAce", "SnkAce"}};

      local Fighters =
      {{"Brigand_pat1", "Excalibur_pat1", "Stormcrow_pat1",
      "Stormcrow_pat1", "EvilEye_pat1", "EvilEye_pat1"},{"Yari_pat1",
      "TieFly_pat1", "Cleaner_pat1", "Tiger_pat1", "Gunslinger_pat1",
      "Gunslinger_pat1"},{"Naginata_pat1", "Bident_uss1", "Bident_uss1",
      "Hrimturs_pat1", "Hrimturs_pat1", "Hrimturs_pat1"},
      {"Naginata_pat1", "Bident_uss1", "Bident_uss1", "Hrimturs_pat1",
      "Hrimturs_pat1", "Hrimturs_pat1"},{"Hammerhead_pat1", "Bident_uss2",
      "Bident_uss2", "Hrimturs_pat2", "Hrimturs_pat2", "Hrimturs_pat2"}};

      local Specials =
      {{"Hatchet_pat1", "Raptor_pat1", "Trident_pat1", "Trident_pat1",
      "Trident_pat1"},{"Hatchet_pat2", "Raptor_pat2", "Trident_pat2",
      "Trident_pat2", "Trident_pat2"},{"Hatchet_pat3", "Raptor_pat3",
      "Trident_pat3", "Trident_pat3", "Trident_pat3"}};

      local ship;
      local pilot;
      local grp = getglobal(group);

      local orient = orient or Vector3(0,0,1);

      if(sign == nil) then
            SLOG("Sign is not defined! Cannot create flight!");
            return FALSE;
      end;

      if(number < 1 or number > 6) then
            SLOG("Number is not in [1;6]! Cannot create flight!");
```

```
            return FALSE;
      end;

      if(skill < 1 or skill > 5) then
            SLOG("Skill is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(purpose < 1 or purpose > 5) then
            SLOG("Purpose is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(strength < 1 or strength > 5) then
            SLOG("Strength is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(grp) then
            flight = grp:CreateFlight("flight_"..sign);
      else
            SLOG("No Group to create flight!");
            return FALSE;
      end;

      setglobal("flight_"..sign, flight);

      for i = 1, number do
            if(i == number and special > 0) then
                  ship = CreateCarcass(Specials[special][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
            else
                  if(i == 1) then
                        ship = CreateCarcass(Fighters[purpose][strength +
                              1], Vector3( 0, 0, (i-1)*5) + location);
                  else
                        ship = CreateCarcass(Fighters[purpose][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
                  end;
            end;

            if(i == 1) then
                  pilot = CreatePilot(Pilots[skill][1]);
            else
                  pilot = CreatePilot(Pilots[skill][2]);
            end;

            ship:AssignPilot(pilot);
            flight:AddShip(ship);
            setglobal("ship_"..sign.."_"..i, ship);
            setglobal("pilot_"..sign.."_"..i, pilot);
      end;
      flight:SetFormation("shipFormation");
      flight:SetOrientation(orient);

      return flight;
end;
USS_FLIGHT = "CreateUSSFlight";
```

Dieses Script Dieses Script ist für die USS-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende

Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Pilots besteht. Bei skill = 4 jedoch wäre der Anführer ein Ace, alle anderen im Team Officers.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.7 function CreateInocoFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreateInocoFlight(group, sign, location, number, skill, purpose,
strength, special, orient)
	local flight;
	local Pilots =	{{"InocoEnforcer", "InocoEnforcer"},
			{"InocoOperative", "InocoEnforcer"},
			{"InocoOperative", "InocoOperative"},
			{"InocoAgent", "InocoOperative"},
			{"InocoAgent", "InocoAgent"}};

	local Fighters =
	{{"Brigand_ino1", "Excalibur_ino1", "Stormcrow_ino1",
	"Stormcrow_ino1", "EvilEye_ino1", "EvilEye_ino1"},{"Yari_ino1",
	"TieFly_ino1", "Cleaner_ino1", "Tiger_ino1", "Gunslinger_ino1",
	"Gunslinger_ino1"},{"Naginata_ino1", "Bident_ino1", "Bident_ino1",
	"Hrimturs_ino1", "Hrimturs_ino1", "Hrimturs_ino1"},{"Naginata_ino1",
	"Bident_ino1", "Bident_ino1", "Hrimturs_ino1", "Hrimturs_ino1",
	"Hrimturs_ino1"},{"Hammerhead_ino1", "Bident_ino2", "Bident_ino2",
	"Hrimturs_ino2", "Hrimturs_ino2", "Hrimturs_ino2"}};

	local Specials =
	{{"Hatchet_ino1", "Raptor_ino1", "Trident_ino1", "Trident_ino1",
	"Trident_ino1"},{"Hatchet_ino2", "Raptor_ino2", "Trident_ino2",
	"Trident_ino2", "Trident_ino2"},{"Hatchet_ino3", "Raptor_ino3",
	"Trident_ino3", "Trident_ino3", "Trident_ino3"}};

	local ship;
	local pilot;
	local grp = getglobal(group);

	local orient = orient or Vector3(0,0,1);

	if(sign == nil) then
		SLOG("Sign is not defined! Cannot create flight!");
		return FALSE;
	end;

	if(number < 1 or number > 6) then
		SLOG("Number is not in [1;6]! Cannot create flight!");
		return FALSE;
	end;

	if(skill < 1 or skill > 5) then
		SLOG("Skill is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(purpose < 1 or purpose > 5) then
		SLOG("Purpose is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;
```

```
	if(strength < 1 or strength > 5) then
		SLOG("Strength is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(grp) then
		flight = grp:CreateFlight("flight_"..sign);
	else
		SLOG("No Group to create flight!");
		return FALSE;
	end;

	setglobal("flight_"..sign, flight);

	for i = 1, number do
		if(i == number and special > 0) then
			ship = CreateCarcass(Specials[special][strength],
					Vector3( 0, 0, (i-1)*5) + location);
		else
			if(i == 1) then
				ship = CreateCarcass(Fighters[purpose][strength +
					1], Vector3( 0, 0, (i-1)*5) + location);
			else
				ship = CreateCarcass(Fighters[purpose][strength],
					Vector3( 0, 0, (i-1)*5) + location);
			end;
		end;

		if(i == 1) then
			pilot = CreatePilot(Pilots[skill][1]);
		else
			pilot = CreatePilot(Pilots[skill][2]);
		end;

		ship:AssignPilot(pilot);
		flight:AddShip(ship);
		setglobal("ship_"..sign.."_"..i, ship);
		setglobal("pilot_"..sign.."_"..i, pilot);
	end;
	flight:SetFormation("shipFormation");
	flight:SetOrientation(orient);

	return flight;
end;
INOCO_FLIGHT = "CreateInocoFlight";
```

Dieses Script Dieses Script ist für die Inoco-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Pilots besteht. Bei skill = 4 jedoch wäre der Anführer ein Agent, alle anderen im Team Operatives.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.8 function CreateTriadaFlight(group, sign, location, number, skill, purpose, strength, special, orient)

```
function CreateTriadaFlight(group, sign, location, number, skill, purpose,
```

```
strength, special, orient)
	local flight;
	local Pilots =	{{"TriadaThug", "TriadaThug"},
			{"TriadaYakuza", "TriadaThug"},
			{"TriadaYakuza", "TriadaYakuza"},
			{"TriadaAgent", "TriadaYakuza"},
			{"TriadaAgent", "TriadaAgent"}};

	local Fighters =
	{{"Brigand_tri1", "Yari_tri1", "Cleaner_tri1", "Tiger_tri1",
	"Gunslinger_tri1", "Gunslinger_tri1"},{"Excalibur_tri1",
	"TieFly_tri1", "Stormcrow_tri1", "Stormcrow_tri1", "EvilEye_tri1",
	"EvilEye_tri1"},{"Naginata_tri1", "Bident_tri1", "Bident_tri2",
	"Bident_tri2", "Hrimturs_tri1", "Hrimturs_tri1"},{"Naginata_tri1",
	"Bident_tri1", "Bident_tri4", "Bident_tri4", "Hrimturs_tri1",
	"Hrimturs_tri1"},{"Hammerhead_tri1", "Bident_tri1", "Bident_tri3",
	"Bident_tri3", "Hrimturs_tri2", "Hrimturs_tri2"}};

	local Specials =
	{{"Hatchet_tri1", "Raptor_tri1", "Raptor_tri1", "Trident_tri1",
	"Trident_tri1"},{"Hatchet_tri2", "Raptor_tri2", "Raptor_tri2",
	"Trident_tri2", "Trident_tri2"},{"Hatchet_tri3", "Raptor_tri3",
	"Raptor_tri3", "Trident_tri3", "Trident_tri3"}};

	local ship;
	local pilot;
	local grp = getglobal(group);

	local orient = orient or Vector3(0,0,1);

	if(sign == nil) then
		SLOG("Sign is not defined! Cannot create flight!");
		return FALSE;
	end;

	if(number < 1 or number > 6) then
		SLOG("Number is not in [1;6]! Cannot create flight!");
		return FALSE;
	end;

	if(skill < 1 or skill > 5) then
		SLOG("Skill is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(purpose < 1 or purpose > 5) then
		SLOG("Purpose is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(strength < 1 or strength > 5) then
		SLOG("Strength is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(grp) then
		flight = grp:CreateFlight("flight_"..sign);
	else
		SLOG("No Group to create flight!");
		return FALSE;
	end;
```

```
	setglobal("flight_"..sign, flight);

	for i = 1, number do
		if(i == number and special > 0) then
			ship = CreateCarcass(Specials[special][strength],
					Vector3( 0, 0, (i-1)*5) + location);
		else
			if(i == 1) then
				ship = CreateCarcass(Fighters[purpose][strength +
					1], Vector3( 0, 0, (i-1)*5) + location);
			else
				ship = CreateCarcass(Fighters[purpose][strength],
					Vector3( 0, 0, (i-1)*5) + location);
			end;
		end;

		if(i == 1) then
			pilot = CreatePilot(Pilots[skill][1]);
		else
			pilot = CreatePilot(Pilots[skill][2]);
		end;

		ship:AssignPilot(pilot);
		flight:AddShip(ship);
		setglobal("ship_"..sign.."_"..i, ship);
		setglobal("pilot_"..sign.."_"..i, pilot);
	end;
	flight:SetFormation("shipFormation");

	flight:SetOrientation(orient);

	return flight;
end;
TRIADA_FLIGHT = "CreateUSSFlight";
```

Dieses Script Dieses Script ist für die Triaden-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, wird der entsprechende Schiffstyp anhand seiner Stärke strength aus der Liste gewählt. Selbiges passiert bei den Jägern anhand der Stärke und der Aufgabe purpose.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Thugs besteht. Bei skill = 4 jedoch wäre der Anführer ein Agent, alle anderen im Team Yakuza.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.1.9 function CreateTraderFlight(group, sign, location, number, skill, purpose, strength, special,orient)

```
function CreateTraderFlight(group, sign, location, number, skill, purpose,
strength, special,orient)
	local flight;
	local Pilots =	{{"Trader", "Trader"},
			{"Trader", "Trader"},
			{"Trader", "Trader"},
			{"Trader", "Trader"},
			{"Trader", "Trader"}};

	local Fighters =
	{"HM_Queen_02", "HM_Queen_01", "Btransport", "HM_Queen_02", "Arba"};
```

```
local CargoType =
{"Cargo_01", "Cargo_02", "Cargo_03", "Cargo_04", "Cargo_05",
"Cargo_06", "Cargo_07", "Cargo_08", "Cargo_09", "Cargo_10"};

local ship;
local pilot;
local grp = getglobal(group);

local orient = orient or Vector3(0,0,1);

if(sign == nil) then
      SLOG("Sign is not defined! Cannot create flight!");
      return FALSE;
end;

if(number < 1 or number > 6) then
      SLOG("Number is not in [1;6]! Cannot create flight!");
      return FALSE;
end;

if(skill < 1 or skill > 5) then
      SLOG("Skill is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(purpose < 1 or purpose > 5) then
      SLOG("Purpose is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(strength < 1 or strength > 5) then
      SLOG("Strength is not in [1;5]! Cannot create flight!");
      return FALSE;
end;

if(grp) then
      flight = grp:CreateFlight("flight_"..sign);
else
      SLOG("No Group to create flight!");
      return FALSE;
end;

local i = 1;

setglobal("flight_"..sign, flight);

ship = CreateCarcass(Fighters[strength], location);

pilot = CreatePilot(Pilots[skill][1]);

ship:AssignPilot(pilot);
flight:AddShip(ship);
setglobal("ship_"..sign.."_1", ship);
setglobal("pilot_"..sign.."_1", pilot);

local MainCargo = RAND(9) + 1;
local MainCargoNum = strength + RAND(6);
ship:AddModuleToInventory(CargoType[MainCargo],MainCargoNum);

for i = 1, strength do
      ship:AddModuleToInventory(CargoType[RAND(9) + 1],1);
end;
```

```
	flight:SetFormation("shipFormation");
	flight:SetOrientation(orient);

	return flight;
end;
TRADER_FLIGHT = "CreateTraderFlight";
```

Im Gegensatz zu den vorherigen Scripts baut dieses hier nur ein Schiff erstellt. Auf Basis der Stärke werden dann mehr oder weniger Waren in Frachtraum des Händlers verstaut. Der Pilot bleibt derselbe. Die Güte der Waren wird über einen Zufallsgenerator ermittelt.

Dem Schiff wird dann noch eine Formation gegeben und weiter geht's.

2.2.1.10 function CreateAlienFlight(group, sign, location, number, skill, purpose, strength, special,orient)

```
function CreateAlienFlight(group, sign, location, number, skill, purpose,
strength, special,orient)
	local flight;
	local Pilots =	{{"AlienStd", "AlienStd"},
			{"AlienStd", "AlienStd"},
			{"AlienTough", "AlienStd"},
			{"AlienTough", "AlienTough"},
			{"AlienBoss", "AlienTough"}};

	local Fighters =
	{{"A_Fighter_1", "A_Fighter_1", "A_Fighter_1", "A_Fighter_1",
	"A_Fighter_1", "A_Fighter_1"},{"A_Fighter_1", "A_Fighter_1",
	"A_Fighter_1", "A_Fighter_1", "A_Fighter_1", "A_Fighter_1"},
	{"A_Bomber_1", "A_Bomber_1", "A_Bomber_1", "A_Bomber_1",
	"A_Bomber_1", "A_Bomber_1"},{"A_Bomber_1", "A_Bomber_1",
	"A_Bomber_1", "A_Bomber_1", "A_Bomber_1", "A_Bomber_1"},
	{"A_Bomber_1", "A_Bomber_1", "A_Bomber_1", "A_Bomber_1",
	"A_Bomber_1", "A_Bomber_1"}};

	local Specials =
	{{"Hatchet_pir1", "Hatchet_pir1", "Hatchet_pir1", "Hatchet_pir1",
	"Hatchet_pir1"},{"Hatchet_pir2", "Hatchet_pir2", "Hatchet_pir2",
	"Hatchet_pir2", "Hatchet_pir2"},{"Hatchet_pir3", "Hatchet_pir3",
	"Hatchet_pir3", "Hatchet_pir3", "Hatchet_pir3"}};

	local ship;
	local pilot;
	local grp = getglobal(group);

	local orient = orient or Vector3(0,0,1);

	if(sign == nil) then
		SLOG("Sign is not defined! Cannot create flight!");
		return FALSE;
	end;

	if(number < 1 or number > 6) then
		SLOG("Number is not in [1;6]! Cannot create flight!");
		return FALSE;
	end;

	if(skill < 1 or skill > 5) then
		SLOG("Skill is not in [1;5]! Cannot create flight!");
		return FALSE;
	end;

	if(purpose < 1 or purpose > 5) then
```

```
            SLOG("Purpose is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(strength < 1 or strength > 5) then
            SLOG("Strength is not in [1;5]! Cannot create flight!");
            return FALSE;
      end;

      if(grp) then
            flight = grp:CreateFlight("flight_"..sign);
      else
            SLOG("No Group to create flight!");
            return FALSE;
      end;

      setglobal("flight_"..sign, flight);

      for i = 1, number do
            if(i == number and special > 0) then
                  ship = CreateCarcass(Fighters[purpose][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
            else
                  if(i == 1) then
                        ship = CreateCarcass(Fighters[purpose][strength +
                              1], Vector3( 0, 0, (i-1)*5) + location);
                  else
                        ship = CreateCarcass(Fighters[purpose][strength],
                              Vector3( 0, 0, (i-1)*5) + location);
                  end;
            end;

            if(i == 1) then
                  pilot = CreatePilot(Pilots[skill][1]);
            else
                  pilot = CreatePilot(Pilots[skill][2]);
            end;

            ship:AssignPilot(pilot);
            flight:AddShip(ship);
            setglobal("ship_"..sign.."_"..i, ship);
            setglobal("pilot_"..sign.."_"..i, pilot);
      end;
      flight:SetFormation("shipFormation");
      flight:SetOrientation(orient);

      return flight;
end;
ALIEN_FLIGHT = "CreateAlienFlight";
```

Dieses Script Dieses Script ist für die Alien-Flotten zuständig und wirkt auf den ersten Blick sehr unübersichtlich. Tatsächlich jedoch besteht es zu 50% aus Variablen-Überprüfungen. Erst die letzt If-Abfrage und die For-Schleife sind der eigentliche Kern des Scripts. Es wird eine Anzahl an Schiffen erstellt, die der Zahl number entspricht. Sollte der Wert special 0 übersteigen, passiert das gleiche als würde man special 0 setzen, da keine Spezialschiffe für Aliens existieren. Die Programmierer dürften etwas faul gewesen sein und haben es (meiner Meinung nach) mit dem Copy-Paste etwas übertrieben. In der Spezialliste dieser Funktion stehen Piratenschiffe, damit der Platz gefüllt ist. Das Script selbst wählt immer auf Basis der der Stärke und der Aufgabe purpose die Schiffe aus, die erstellt werden sollen.

Bei den Piloten wird über den skill immer ein Paar gewählt. Der Leader ist der stärkere Typ. Beispielsweise würde das Script bei skill = 1 ein Team erstellen, das nur aus Std besteht. Bei skill = 3 jedoch wäre der Anführer ein Tough, alle anderen im Team Std.

Zum Schluß wird dem Haufen noch eine Formation zugewiesen und los geht's.

2.2.2 RandomContacts.script

2.2.2.1 function RegisterRandomContact(contacttype, contactName, portalList, navPointList, probability, minShips, maxShips, power)

```
function RegisterRandomContact(contacttype, contactName, portalList,
navPointList, probability, minShips, maxShips, power)
      local prob = probability or 100;

      if (minShips == nil) then
            minShips = 0;
      end;

      maxShips = maxShips or 6;
      local power = power or 1;

      local table =
      {contacttype, contactName, portalList, navPointList, prob, minShips,
      maxShips, power};

      local count = getn(__sys_LevelRegisteredContacts);
      count = count + 1;
      __sys_LevelRegisteredContacts[count] = table;

      LOG(count);
end;
```

Dieses Script ist quasi die Initialisierung neuer Kontakte in einem Sektor. Man gibt Typ, Name, eine Liste der im Sektor vorhandenen Portale, eine Navpoint-Liste, eine Wahrscheinlichkeit, minimale und maximale Schiffszahl sowie die Stärke an und das Script legt dies in eine globale Liste ab. Für jeden neuen Kontakt wird diese Liste um einen Eintrag verlängert.

Nach Eintragung in diese Liste arbeitet das System völlig autark. Ein nachträgliches Löschen daraus ist scheinbar nicht vorgesehen worden. Es existieren lediglich Funktionen zum Anzeigen dieser Liste, die aber scheinbar nur von den Entwicklern genutzt wurden.

2.2.2.2 function ContactListView()

```
function ContactListView()
      --SLOG("----------- Contact list --------------");
      for i, val in __sys_LevelRegisteredContacts do
            local contactName = val[2];
            local portals = getn(val[3]);
            local navpoints = getn(val[4]);

            --SLOG(i..". "..contactName.." portals - "..portals.."
                  navpoints - "..navpoints);
      end
      --SLOG("---------------------------------------");
end;
```

Ein sehr simples Script, das jeden Eintrag der registrierten Kontaktliste ausgibt. Mir ist nicht bekannt, dass dieses Script für Spieler sichtbar oder wichtig wäre. Es diente lediglich den Entwicklern um überprüfen zu können, ob die Eintragungen in der Liste auch stimmen.

2.2.2.3 function ComputeProbability(probability)

```
function ComputeProbability(probability)
      local val = RAND(100);
      if val > (100 - probability) then
            return TRUE;
      else
```

```
            return FALSE;
      end;
end;
```

Diese Funktion ermittelt das Resultat einer Wahrscheinlichkeitsentscheidung. Die Variable val wird mit einem Zufallswert zwischen 0 und 100 initialisiert und anschließend mit der Schwelle 100 – probability verglichen. Ist der Wert von val über der Schwelle wird True returniert, sonst False.

2.2.2.4 function ComputeRandomInvervalValue(min, max)

```
function ComputeRandomInvervalValue(min, max)
      local val = max - min + 1; -- because RAND is [0, val) constructed
      local rnd = RAND(val);
      val = min + rnd;
      return val;
end;
```

Dieses Script errechnet einen Zufallswert für das Intervall. Dieser basiert auf folgender Formel.

$$\text{Intervall} = \min + RAND(\max - \min + 1)$$

2.2.2.5 function GetRandomPortalIndexes(portalList)

```
function GetRandomPortalIndexes(portalList)
      local portalCount = getn(portalList);

      if portalCount == 1 then
            return {1, 1};
      end;

      local randomIndex = RAND(portalCount) + 1;
      local newPortalIndex = randomIndex;

      while (newPortalIndex == randomIndex) do
            randomIndex = RAND(portalCount) + 1; -- RAND is ZERO based
      end;
      return {randomIndex, newPortalIndex};
end;
```

Mit diesem Script werden aus der Portalliste zwei Portale entnommen. Das Ziel und der Ursprung der erzeugten Staffel. Existiert nur ein Portal in einem Sektor, so sind Ziel und Ursprung der selbe, andernfalls wird auf Basis einer Zufallszahl ein Portal gewählt und dann so lange der Zufallsgenerator abgerufen, bis dieser ein Portal liefert, das nicht das Ursprungsportal ist.

2.2.2.6 function GetRandomWayPointList(pointsList, maxPointCount)

```
function GetRandomWayPointList(pointsList, maxPointCount)
      local points = getn(pointsList);

      if maxPointCount == nil then
            maxPointCount = points;
      end;

      local path = {};

      if  maxPointCount == 0 then
            return path;
      end;

      for i = 1, maxPointCount do
            local rnd = RAND(maxPointCount) + 1;
            path[i] = pointsList[rnd];
      end;
      return path;
end;
```

Diese Funktion stellt aus der PointsListe eine Abfolge von Navigationspunkte zusammen. Die Anzahl der Punkte im vorgegebenen Pfad ist gleich maxPointCount. Wird der Wert auf nil gesetzt ist der Weg so lange wie es Punkte in der Liste gibt. Wird er auf 0 gesetzt, handelt es sich um eine leere Liste.

2.2.2.7 function GetRandomContact()

```
function GetRandomContact()
	local contactCount = getn(__sys_LevelRegisteredContacts);

	--SLOG("Contact count is "..contactCount);

	local randomContactIndex = RAND(contactCount) + 1;

	--SLOG("RandomContactIndex is "..randomContactIndex);

	return randomContactIndex;
end;
```

Ein Script das auf Basis einer Zufallszahl einfach einen Eintrag aus den registrierten Zufallskontakten entnimmt.

2.2.2.8 Trigger

Das Betreten eines Portals und das Entstehen einer Flotte passiert über Event-Trigger, die im Folgenden ein wenig erklärt werden.

Ein Trigger besteht immer aus einer Condition, die angibt, wann der Trigger einzutreffen hat, einem Event, das angibt wann die Condition überprüft werden soll und einer Action, die ausgeführt werden soll wenn die Condition zutrifft. Außerdem verfügt jeder Trigger über eine Activate-Funktion, die den Trigger freigibt. Zeigen wir dies an einem Beispiel:

2.2.2.8.1 function StartRandomContact(firstInterval, interval)

```
function StartRandomContacts(firstInterval, interval)
	_sys_RandomContactsInterval = interval;
	-- create Trigger
	trigger_generator = CreateTrigger();
	trigger_generator:AttachEvent(_EVENT_TIMER);
	trigger_generator:SetTime(firstInterval);

	trigger_generator:AttachCondition(trigger_generator_Condition);
	trigger_generator:AttachAction(trigger_generator_Action);
	trigger_generator:Activate();
end;
```

Man sieht, dass hier ein Trigger erstellt wird, der trigger_generator heißt. An diesen wird ein Timer_Event gehängt um daraus einen Countdown zu machen. Weiters wird über SetTime eine Intervallzeit angegeben.

Die nächsten beiden Zeilen definieren die Aktionen, die ausgeführt werden sollen, wenn das event eintritt.

2.2.2.8.2 function trigger_generator_Condition()

```
function trigger_generator_Condition()
	return TRUE;
end;
```

Da es sich bei dem Trigger um einen Timer handelt existieren keine Bedingungen. Deshalb gibt diese Funktion permanent True zurück.

2.2.2.8.3 function trigger_generator_Action()

```
function trigger_generator_Action()
	local this = GetTriggerContext();
	this:SetTime(_sys_RandomContactsInterval);
	this:Activate();
```

```
local contactIndex = GetRandomContact();
local currentContact = __sys_LevelRegisteredContacts[contactIndex];
local probability = currentContact[5];

local execute = ComputeProbability(probability);
if (execute == FALSE) then
      return;
end;

local portalIndexes = GetRandomPortalIndexes(currentContact[3]);
__CurrentPortalIndex = portalIndexes[1];

-- start and end portal indexes
local portalStartIndex  = portalIndexes[1];
local portalEndIndex    = portalIndexes[2];

--get current function to create a flight, it's a contact type
local f_name = currentContact[1];
local currentFunction = getglobal(f_name);
local groupName = currentContact[2];
local shipCount = ComputeRandomInvervalValue(currentContact[6],
                                             currentContact[7]);
local purpose = ComputeRandomInvervalValue(1, 3);
local power = currentContact[8];
local skill = ComputeRandomInvervalValue(power, power + 2);
local strength = ComputeRandomInvervalValue(power, power + 2);

currentContact[3][portalStartIndex]:Start(8, 0);
local flight = currentFunction(groupName,
            "randomformation_"..__formation_number,
            currentContact[3][portalStartIndex]:GetEntryPoint(10),
            shipCount, skill, purpose, strength, 1,
            currentContact[3][portalStartIndex]:GetOrientation());

__formation_number = __formation_number + 1;

local patrolPath =
{currentContact[3][portalStartIndex]:GetEntryPoint(50)};

local wayPointsPath = GetRandomWayPointList(currentContact[4]);
local wayPointsCount = getn(wayPointsPath);

local addIndex = 2;
local endIndex = addIndex + wayPointsCount - 1;
local wayPointIndex = 1;

for i = addIndex, endIndex do
      patrolPath[i] = wayPointsPath[wayPointIndex];
      wayPointIndex = wayPointIndex + 1;
end;

patrolPath[endIndex + 1] =
currentContact[3][portalEndIndex]:GetEntryPoint(50);

patrolPath[endIndex + 2] =
currentContact[3][portalEndIndex]:GetEntryPoint(1);

local patrolPathLength = getn(patrolPath);
flight:Patrol(patrolPath, FALSE);

local trigger_inside2portal = CreateTrigger();
trigger_inside2portal:AttachEvent(_EVENT_FLIGHTINSIDEOBJECTVOLUME);
```

```
	trigger_inside2portal:SetObjects(flight,
					currentContact[3][portalEndIndex], 40);
	trigger_inside2portal:AttachCondition(trigger_inside2portal_Condition
					);
	trigger_inside2portal:AttachAction(trigger_inside2portal_Action);

	local trigger_exit_point = CreateTrigger();
	trigger_exit_point:AttachEvent(_EVENT_FLIGHTINSIDEVOLUME);
	trigger_exit_point:SetObject(flight);
	trigger_exit_point:SetVolume(patrolPath[1], 5);
	trigger_exit_point:AttachCondition(trigger_exit_point_Condition);
	trigger_exit_point:AttachAction(trigger_exit_point_Action);
	trigger_exit_point:AssotiateTrigger(trigger_inside2portal);
	trigger_exit_point:Activate();
end;
```

Dieses Script muss man in mehrere Etappen unterteilen um es besser beschreiben zu können.

2.2.2.8.3.1 Reinitialisierung trigger_generator

```
	local this = GetTriggerContext();
	this:SetTime(_sys_RandomContactsInterval);
	this:Activate();
```

Hier wird der Trigger neu initialisiert und gestartet, damit er den nächsten Event auslösen kann.

2.2.2.8.3.2 Evaluierung der Wahrscheinlichkeit sowie Auswahl einer Flotte

```
	local contactIndex = GetRandomContact();
	local currentContact = __sys_LevelRegisteredContacts[contactIndex];
	local probability = currentContact[5];

	local execute = ComputeProbability(probability);
	if (execute == FALSE) then
		return;
	end;
```

Dieser Part ist auch noch relativ selbsterklärend. Ein Aufruf der Zufallsfunktion ermittelt die Flotte, die nun vielleicht erscheinen soll. Anhand deren Wahrscheinlichkeit kann der zweite Funktionsaufruf dann entscheiden ob dies auch tatsächlich passiert.

2.2.2.8.3.3 Ermittelung der Start- und Endportale

```
	local portalIndexes = GetRandomPortalIndexes(currentContact[3]);
	__CurrentPortalIndex = portalIndexes[1];

	-- start and end portal indexes
	local portalStartIndex  = portalIndexes[1];
	local portalEndIndex    = portalIndexes[2];
```

Hier werden aus der Liste per Zufallsgenerator zwei Portale aus der Portalliste des Zufallskontaktes entnommen. Sollte dieser nur ein Portal enthalten, so ist dies Start- und Endpunkt zugleich.

2.2.2.8.3.4 Errechnung der Kampfkraft und Zusammensetzung der Flotte

```
	--get current function to create a flight, it's a contact type
	local f_name = currentContact[1];
	local currentFunction = getglobal(f_name);
	local groupName = currentContact[2];
	local shipCount = ComputeRandomInvervalValue(currentContact[6],
						currentContact[7]);
	local purpose = ComputeRandomInvervalValue(1, 3);
	local power = currentContact[8];
	local skill = ComputeRandomInvervalValue(power, power + 2);
	local strength = ComputeRandomInvervalValue(power, power + 2);
```

Hier wird's schon etwas komplexer. In der Liste currentContact repräsentiert der erste Eintrag gleichzeitig die Funktion, die zum Erstellen der Flotte verwendet wird. Siehe dazu das zweite Tutorial für ein Beispiel.

Mit den restlichen Werten wird ein Zufallsgenerator gefüttert, der die Werte Stärke, Anzahl und Zusammensetzung für die Zufallsflotte generiert.

2.2.2.8.3.5 Erzeugen der Flotte

```
      currentContact[3][portalStartIndex]:Start(8, 0);
      local flight = currentFunction(groupName,
                  "randomformation_"..__formation_number,
                  currentContact[3][portalStartIndex]:GetEntryPoint(10),
                  shipCount, skill, purpose, strength, 1,
                  currentContact[3][portalStartIndex]:GetOrientation());

__formation_number = __formation_number + 1;
```

Durch die globale Variable weiß nun das Script welche Funktion zum Erstellen der Flotte benötigt wird. Am Startportal wird eine Eintrittssequenz durch den Startbefehl initiiert und gleich darauf die Flotte mit den zuvor ermittelten Werten im Zentrum des Portals erstellt. Um unterschiedliche Formationen zu haben wird nach jedem erfolgten Erzeugen der Formationsindex inkrementiert.

2.2.2.8.3.6 Festlegung eines Navigationsplanes

```
      local patrolPath =
      {currentContact[3][portalStartIndex]:GetEntryPoint(50)};

      local wayPointsPath = GetRandomWayPointList(currentContact[4]);
      local wayPointsCount = getn(wayPointsPath);

      local addIndex = 2;
      local endIndex = addIndex + wayPointsCount - 1;
      local wayPointIndex = 1;

      for i = addIndex, endIndex do
            patrolPath[i] = wayPointsPath[wayPointIndex];
            wayPointIndex = wayPointIndex + 1;
      end;

      patrolPath[endIndex + 1] =
      currentContact[3][portalEndIndex]:GetEntryPoint(50);

      patrolPath[endIndex + 2] =
      currentContact[3][portalEndIndex]:GetEntryPoint(1);

      local patrolPathLength = getn(patrolPath);
      flight:Patrol(patrolPath, FALSE);
```

Die Start- und Endpunkte des Navigationsplanes der Zufallsflotten sind die Portale, dazwischen jedoch können sie mehrere Orte wie Handelsstationen oder Patrouillenstationen anfliegen. Dieser Part generiert die Navigationsliste und startet den Rundflug.

2.2.2.8.3.7 Trigger inside2Portal

```
      local trigger_inside2portal = CreateTrigger();
      trigger_inside2portal:AttachEvent(_EVENT_FLIGHTINSIDEOBJECTVOLUME);
      trigger_inside2portal:SetObjects(flight,
                              currentContact[3][portalEndIndex], 40);
      trigger_inside2portal:AttachCondition(trigger_inside2portal_Condition
                              );
      trigger_inside2portal:AttachAction(trigger_inside2portal_Action);
```

Auch das Entfernen der Flotte muss gescriptet werden. Hierzu bedient man sich zwei Trigger. Der erste wird aktiviert, wenn folgende Bedingung erfüllt ist.

```
function trigger_inside2portal_Condition()
      local myTriggerContext = GetTriggerContext();
      local portal = myTriggerContext:GetDestinationObject();
      local flight = myTriggerContext:GetFlight();
      local pos = flight:GetPosition();
```

```
	local v = portal:GetPosition() - pos;
	length = GetLength(v);
	if (portal_started == FALSE) then
		portal:Start(10, 0);
		portal_started = TRUE;
	end;
	if (length<10) then
		portal_started = FALSE;
		return TRUE;
	end;
	return FALSE;
end;
portal_started = FALSE;
```

Beträgt die Länge zwischen Flotte und Tor weniger als 10 wird das Tor gestartet und TRUE zurückgegeben um den Trigger zu aktivieren. Danach führt der Trigger folgende Funktion aus.

```
function trigger_inside2portal_Action()
	local myTriggerContext = GetTriggerContext();
	local flight = myTriggerContext:GetFlight();
	flight:Delete();
end;
```

Womit die Flotte gelöscht wird. Dieser Trigger wird jedoch nicht gleich aktiviert sondern wird durch einen zweiten Trigger freigegeben der im Hauptscript so aussieht.

2.2.2.8.3.8 Trigger Exit_Point

```
	local trigger_exit_point = CreateTrigger();
	trigger_exit_point:AttachEvent(_EVENT_FLIGHTINSIDEVOLUME);
	trigger_exit_point:SetObject(flight);
	trigger_exit_point:SetVolume(patrolPath[1], 5);
	trigger_exit_point:AttachCondition(trigger_exit_point_Condition);
	trigger_exit_point:AttachAction(trigger_exit_point_Action);
	trigger_exit_point:AssotiateTrigger(trigger_inside2portal);
	trigger_exit_point:Activate();
```

Dieser Trigger ist aktiv, solange die Flotte ihrem navigationsplan folgt. Die Bedingung, dass der Trigger ausgelöst wird, lautet wie folgt.

```
function trigger_generator_Condition()
	return TRUE;
end;
```

Was bedeutet, dass der Trigger ständig aktiv ist und seine Action auch ständig ausgeführt wird.

```
function trigger_exit_point_Action()
	local triggerContext = GetTriggerContext();
	local assotiatedTriggerId = triggerContext:GetAssotiatedData();
	if (assotiatedTriggerId) then
		anotherTrigger = GetTriggerContext(assotiatedTriggerId);
		anotherTrigger:Activate();
	end;
	triggerContext:Delete();
end;
```

Somit sieht man, dass dieser Trigger ständig den anderen aktiviert, damit dieser überprüfen kann, ob die Flotte ihren Endpunkt erreicht hat.

3 Tutorials

3.1 Änderung eines Schiffsicons

Wie in 1.2.8 beschrieben braucht man dazu den DDS-Plugin von NVidia sowie ein passendes Bildbearbeitungsprogramm wie z.B. Photoshop. Nehmen wir uns die EvilEye des Spielers vor (weil diese sehr früh im Spiel zu sehen und zu überprüfen ist). Wir beschränken uns als erstes darauf, das bestehende Bild zu verändern, da es einfacher ist als ein neues zu gestalten.

1.) Öffnen wir mit Photoshop oder einem anderen Programm die entsprechende Bilddatei im Ordner \TEXTURE\Interface\Carcass\128, in unserem Fall die Datei EvilEye_Player.dds
2.) Der folgende Dialog erscheint:

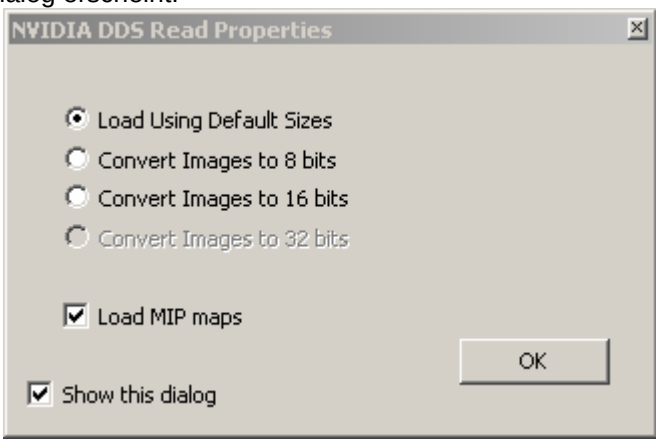


Abbildung 3-1: Setup zum Öffnen einer .dds Datei

3.) Wir wählen Default Sizes und wählen das Kontrollkästen MIP maps aus. Default sizes bedeutet eine 8Bit Farbinformation für Rot, Grün und Blau. Daraufhin öffnet sich das Bild im Photoshop.

Abbildung 3-2: Photoshop-Ansicht der EvilEye

4.) Nun verkleinern wir das Bild auf eine Größe von 128x128 Pixel, indem wir die Funktion Image → Canvas Size verwenden. Die Abmessungen setzen wir gleich auf Pixel um leichter arbeiten zu können.

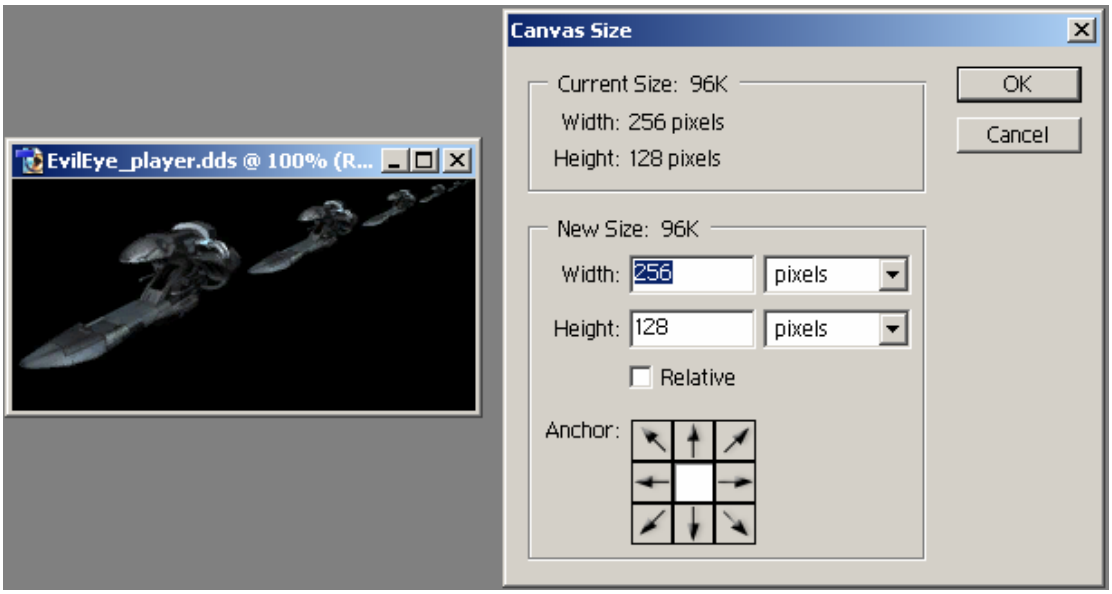


Abbildung 3-3: Canvas Size Dialog mit Pixelabmessungen

5.) Die kleineren Images rechts werden vom DDS-Plugin erzeugt, wir benötigen nur das Bild ganz links. Deshalb setzen wir den Canvas auf left und geben eine neue Breite von 128 ein.

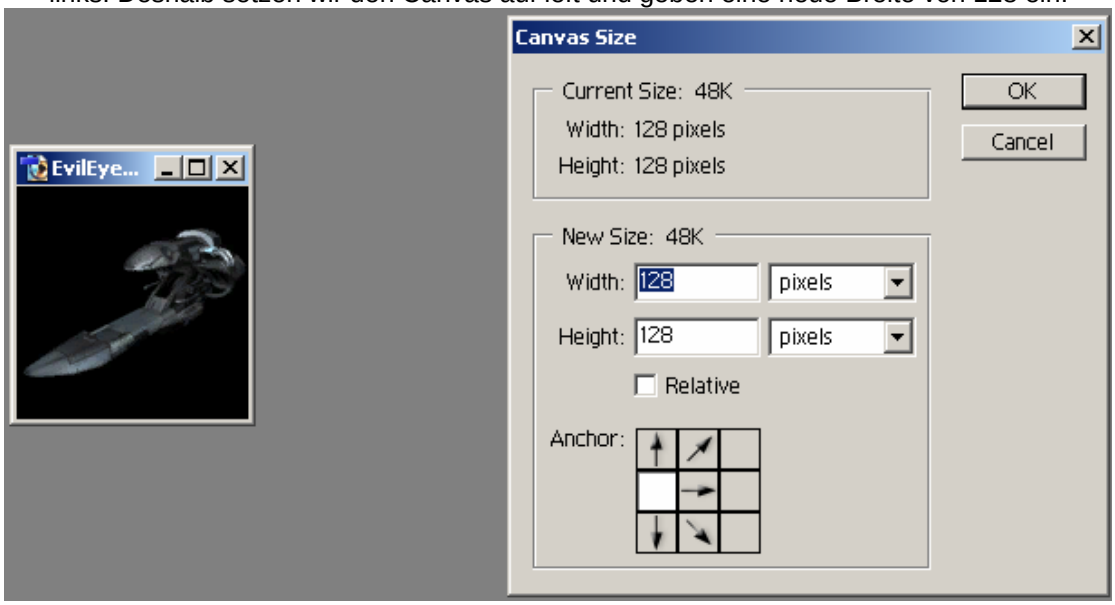


Abbildung 3-4: Extraktion des Grundbildes

6.) Dieses Bild dient uns nun als Grundmaske. Nun können wir die gewünschten Änderungen vornehmen. Ich habe zur Veranschaulichung einfach ein paar rote Punkte hineingesetzt.

Abbildung 3-5: Neue EvilEye

7.) Nun kommt das Speichern. Vorher müssen wir uns aber vergewissern das wir eine Image-Größe von 128 haben. Dir Canvas Size haben wir das hier gemacht. In einem anderen Fall müssen wir diesen Wert durch Image -→ Image Size überprüfen und gegebenenfalls ändern.

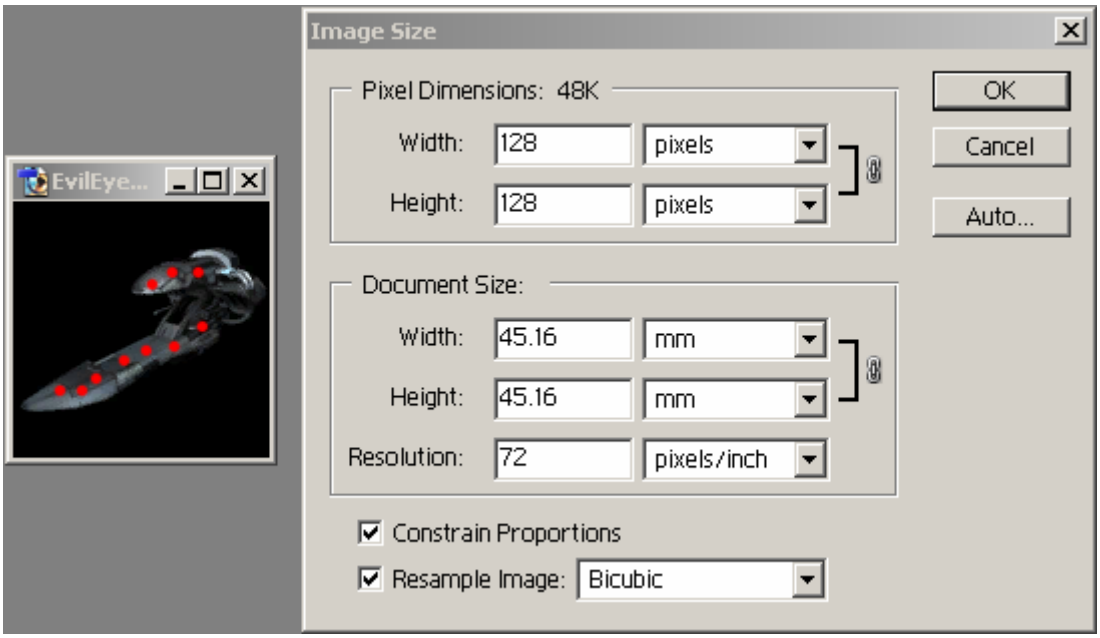


Abbildung 3-6: Image Size

8.) Dieses Bild speichern wir nun insgesamt 4 mal:

a. mit der Funktion Save As in einem eigenen Ordner im Format .PSD damit wir später diese Datei unverfälscht wieder öffnen können,

b. mit der Funktion Save As als .DDS Format im Ordner \TEXTURE\Interface\Carcass\128 überschreiben wir die Datei EvilEye_player.dds,

c. im folgenden Dialog wählen wir die folgenden Einstellungen:

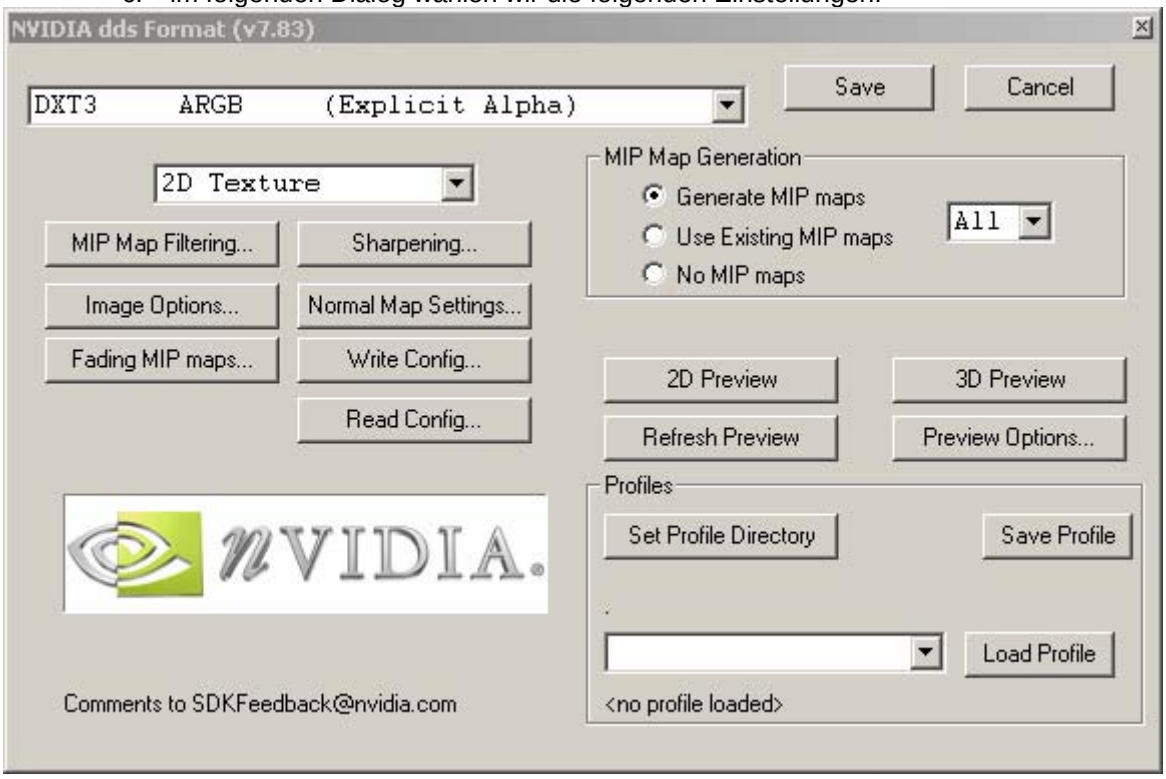


Abbildung 3-7: DDS-Plugin Setup Dialog

d. Mit einem Klick auf 2D Preview kann die Ausgabe überprüft werden. Sie sollte ähnlich der sein, die wir zuvor geöffnet haben um sie zu ändern, mit dem Unterschied, dass nun alle kleineren Images auch die Änderungen beinhalten,

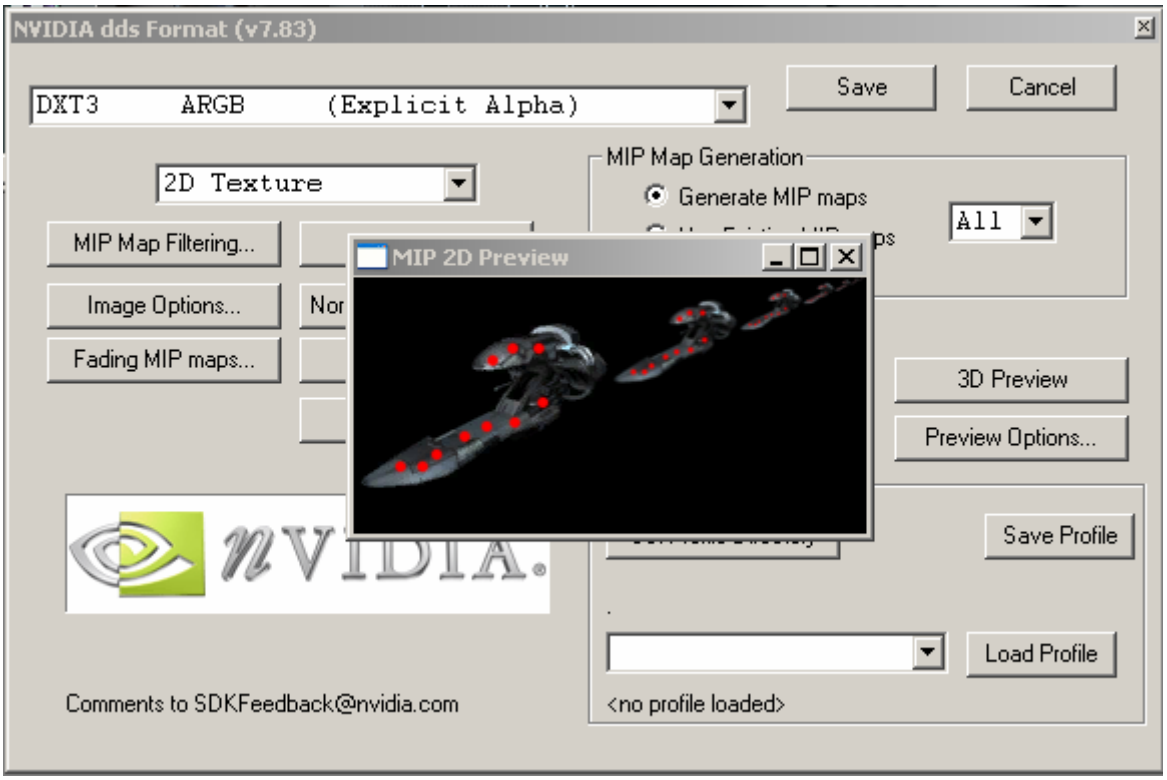


Abbildung 3-8: Preview des neuen Icons

e. Wir speichern ab und öffnen die zuvor gemachte PSD Datei, sollte Photoshop nun in der Ansicht sämtliche Images anzeigen. Die Grundversion (nur das geänderte Bild ohne Images rechts) wird nun über Image → Image Size verkleinert auf die Auflösung 64,

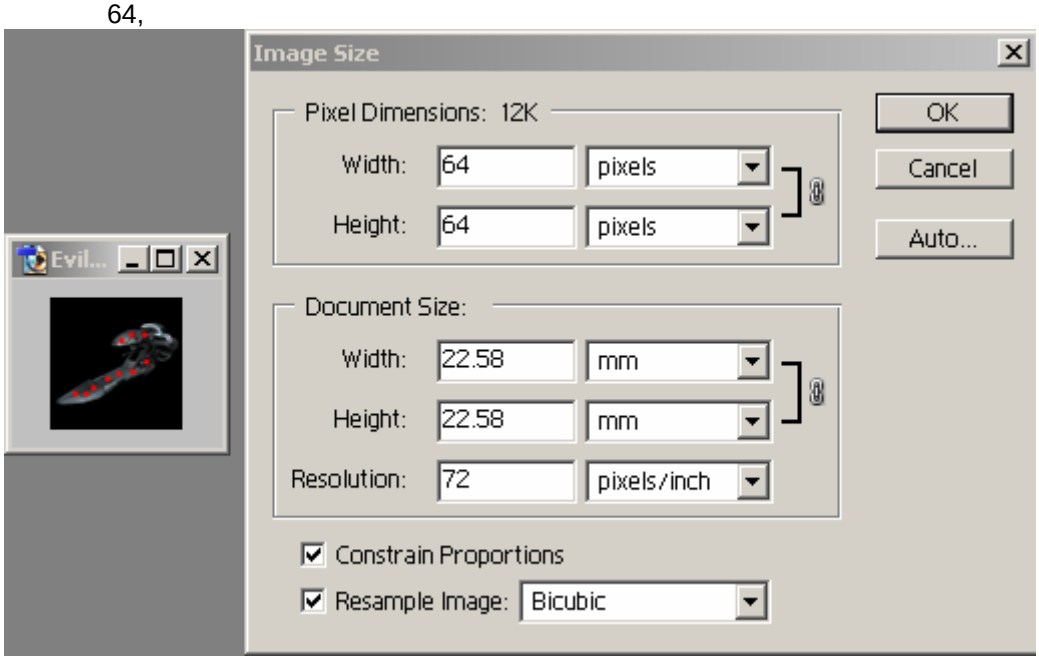


Abbildung 3-9: Änderung der Größe auf 64 Pixel

f. Diese Datei wird nun mit der Funktion Save As im Ordner \TEXTURE\Interface\Carcass\64 plaziert und überschreibt dort die EvilEye_player.dds,

g. Nun wiederholen wir die Schritte e und f mit einer Auflösung von 32 Pixel und speichern die Datei im Ordner \TEXTURE\Interface\Carcass\32 ab. Überschreiben dabei wieder die vorhandene Datei,

9.) Fertig. Nun sind alle Files platziert und wir können uns das Ergebnis im Spiel ansehen.

Abbildung 3-10: Das Schiff selbst ändert sich durch das Icon nicht

Abbildung 3-11: Jedoch im Hangar sehen wir das Resultat

3.2 Erstellen eigener Zufallsbegegnungen

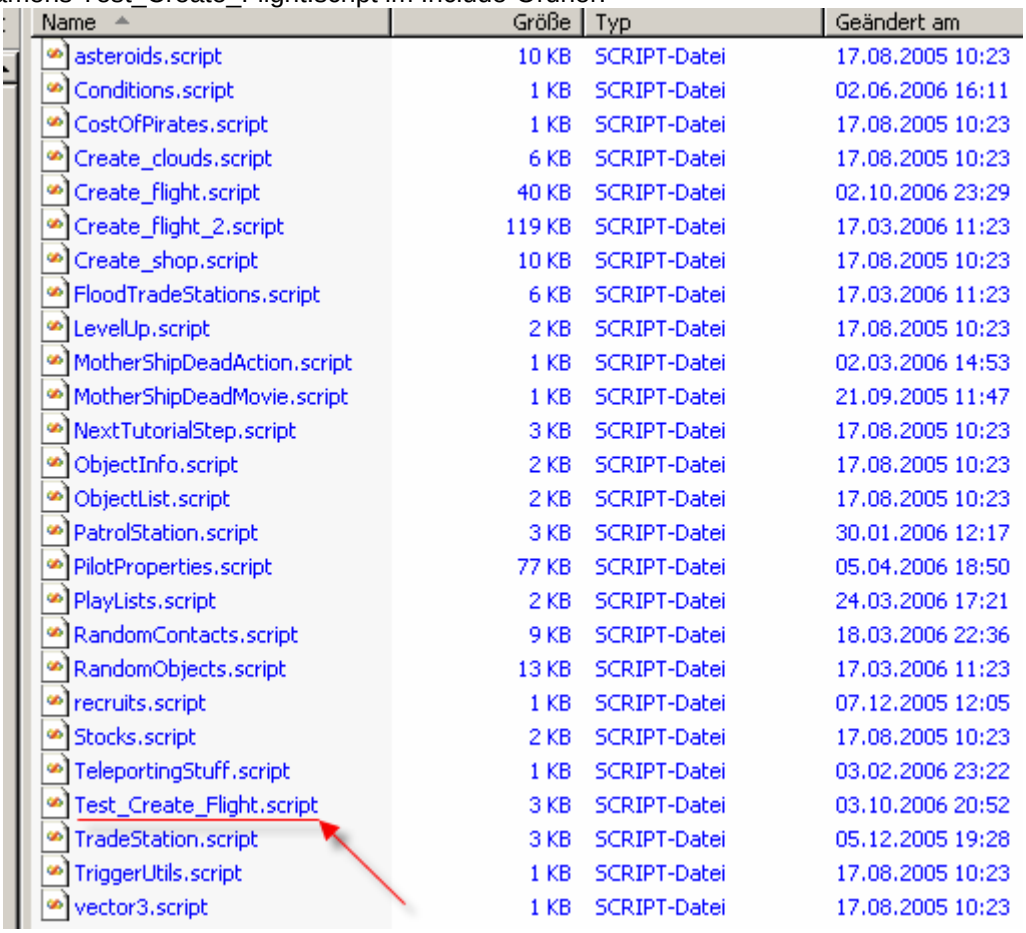
Es ist sehr ratsam sich das Scripting-System gut anzusehen bevor man sich an einen solchen Eingriff wagt. Für all jene verwegenen Modder, die sich der Gefahren bewusst sind, hier dennoch zwei wichtige Hinweise:

1.) die Datei LOGFile.txt im Hauptspielordner \StarWolves2\ zeigt alle Aktionen, die die Spieleengine durchführt, darunter auch Scriptaufrufe. Sie zeigt auch, welcher Fehler passiert ist, sollte ein Script nicht funktionieren. Dies ist die Debug-Funktion.
2.) Erstellt von jeder veränderten Datei eine Sicherheitskopie, um nicht neu installieren zu müssen sollte was schief gehen.

Und nun frisch ans Werk.

3.2.1 Eine neue Include-Datei

Um nicht in bestehenden Dateien rumpfuschen zu müssen erstellen wir eine neue Scriptdatei im Ordner Include. Öffnet dazu einen beliebigen Texteditor (Notepad, UltraEdit, etc.) und erstellt eine Datei namens Test_Create_Flight.script im Include-Ordner.



Name	Größe	Typ	Geändert am
asteroids.script	10 KB	SCRIPT-Datei	17.08.2005 10:23
Conditions.script	1 KB	SCRIPT-Datei	02.06.2006 16:11
CostOfPirates.script	1 KB	SCRIPT-Datei	17.08.2005 10:23
Create_clouds.script	6 KB	SCRIPT-Datei	17.08.2005 10:23
Create_flight.script	40 KB	SCRIPT-Datei	02.10.2006 23:29
Create_flight_2.script	119 KB	SCRIPT-Datei	17.03.2006 11:23
Create_shop.script	10 KB	SCRIPT-Datei	17.08.2005 10:23
FloodTradeStations.script	6 KB	SCRIPT-Datei	17.03.2006 11:23
LevelUp.script	2 KB	SCRIPT-Datei	17.08.2005 10:23
MotherShipDeadAction.script	1 KB	SCRIPT-Datei	02.03.2006 14:53
MotherShipDeadMovie.script	1 KB	SCRIPT-Datei	21.09.2005 11:47
NextTutorialStep.script	3 KB	SCRIPT-Datei	17.08.2005 10:23
ObjectInfo.script	2 KB	SCRIPT-Datei	17.08.2005 10:23
ObjectList.script	2 KB	SCRIPT-Datei	17.08.2005 10:23
PatrolStation.script	3 KB	SCRIPT-Datei	30.01.2006 12:17
PilotProperties.script	77 KB	SCRIPT-Datei	05.04.2006 18:50
PlayLists.script	2 KB	SCRIPT-Datei	24.03.2006 17:21
RandomContacts.script	9 KB	SCRIPT-Datei	18.03.2006 22:36
RandomObjects.script	13 KB	SCRIPT-Datei	17.03.2006 11:23
recruits.script	1 KB	SCRIPT-Datei	07.12.2005 12:05
Stocks.script	2 KB	SCRIPT-Datei	17.08.2005 10:23
TeleportingStuff.script	1 KB	SCRIPT-Datei	03.02.2006 23:22
Test_Create_Flight.script	3 KB	SCRIPT-Datei	03.10.2006 20:52
TradeStation.script	3 KB	SCRIPT-Datei	05.12.2005 19:28
TriggerUtils.script	1 KB	SCRIPT-Datei	17.08.2005 10:23
vector3.script	1 KB	SCRIPT-Datei	17.08.2005 10:23

Abbildung 3-12: Neue Scriptdatei

Damit diese Datei vom Spiel auch gelesen und ausgewertet wird, muss sie in der system.lst im Ordner \DATA\Scripts\ aufgeführt sein. Also schreiben wir sie an den Schluss der Datei.

```
31   "include/MotherShipDeadAction.script",
32   "include/TradeStation.script",
33   "include/PatrolStation.script",
34   "include/RandomContacts.script",
35   "include/RandomObjects.script",
36   "include/ObjectList.script",
37   "include/TriggerUtils.script",
38   "include/Conditions.script",
39   "include/TeleportingStuff.script",
40   "include/Recruits.script",
41   "include/FloodTradeStations.script",
42   "include/PlayLists.script",
43   "include/Test_Create_Flight.script"
44
45 }
```

Abbildung 3-13: Neue system.lst

Jetzt können wir neue Funktionen erstellen ohne die alten Dateien verändern zu müssen.

3.2.2 Die neue Funktion

Als Beispiel hab ich mir gedacht wir erstellen eine Funktion, die auf Zufallsbasis Großkampfschiffe der USS erzeugt. Dazu schreiben wir in unsere nagelneue Datei folgende Funktion:

```
function BigUSSFlight(group, sign, location, number, skill, purpose,
strength, special, orient)
      local flight;
      local Pilots =    {"USSEnforcer",
                         "USSOperative",
                         "USSAgent"};

      local BigShips = {{"StoneArrow0"},
                         {"BattleShip_0"},
                         {"Starhammer_0"}};

      local BigShipsPilotRoles =    {{"1", "A"},      --1--
                                     {"1", "A"},      --2--
                                     {"1", "A"},      --3--
                                     {"1", "A"},      --4--
                                     {"1", "A"},      --5--
                                     {"2", "A"},      --6--
                                     {"2", "A"},      --7--
                                     {"2", "A"},      --8--
                                     {"2", "A"},      --9--
                                     {"2", "A"}};     --10--

      local ship;
      local pilot;
      local grp = getglobal(group);

      local orient = orient or Vector3(0,0,1);
      flight = grp:CreateFlight("flight_"..sign);

      setglobal("flight_"..sign, flight);

      local level = strength or 1;
      local i1;
      local skill;

      local shiptype = RAND(2) + 1;

      ship = CreateCarcass(BigShips[shiptype][1], Vector3( 0, 0, 0) +
                                                  location);
```

```
	if (BigShipsPilotRoles[level][1] == "3") then
		skill = 3
	else
		if (BigShipsPilotRoles[level][1] == "2") then
			skill = 2
		else
			skill = 1
		end;
	end;

	pilot = CreatePilot(Pilots[skill]);

	ship:AssignPilot(pilot);
	flight:AddShip(ship);

	setglobal("ship_"..sign, ship);
	setglobal("pilot_"..sign, pilot);

	flight:SetFormation("shipFormation");
	flight:SetOrientation(orient);

	return flight;
end;
USS_BFLIGHT = "BigUSSFlight";
```

Der Hauptteil sieht aus wie die Funktionen aus der Create_flight.script. Der Unterschied besteht darin, das hier nur ein Schiff erzeugt wird (um es einfach zu halten) und das die Art des Schiffes über einen Zufallsgenerator bestimmt wird.

Die meisten der Übergabevariablen der Funktion werden effektiv nicht gebraucht, sind aber nötig, da der Trigger der RandomContact-Funktion diese Parameter erwartet. Ansonsten alles wie gehabt. Ein Schiff wird erstellt, ein Pilot zugewiesen und die Flotte retourniert.

Wichtig ist noch die letzte Zeile. Durch sie weiß der Trigger später, welche Funktion für unseren neu definierten Typ „USS_BFLIGHT“ zuständig ist. Der String danach muss der Name der Funktion sein.

3.2.3 Eintragung in einem Sektor

Jeder Sektor beinhaltet in seiner activate.script die Definitionen für die Zufallsbegegnungen. Nun wollen wir diese Datei des Sektors Lasad mal ändern. Die dazugehörige Datei befindet sich unter \DATA\Scripts\Locations\Lasad\. An ihr Ende fügen wir folgenden Eintrag.

```
	USS_Team = CreateTeam("USS");
	USS_Group_1 = USS_Team:CreateGroup("USS_Group_1");
	RegisterRandomContact(USS_BFLIGHT, "USS_Group_1",{PORTAL_TO_AREKO ,
		PORTAL_TO_TARROT}, {TRADESTATION:GetPosition()+Vector3(0, 30,
		0)}, 100, 1, 1, power);

	StartRandomContacts(100, 100);
```

Damit erstellen wir einen Eintrag in der RandomContacts-Liste, der die Flotte, die als USS_BFLIGHT definiert wurde, mit einer 100%igen Wahrscheinlichkeit alle 100 Sekunden an einem der beiden Tore entstehen lässt. USS_BFLIGHT weist dabei auf die zuvor von uns erstellte Funktion.

Nun können wir unser Werk im Spiel betrachten. Das Schiff taucht am Portal mit der dazugehörigen Animation auf und verschindet an ihrem Ziel. Dabei fliegt es wie gewohnt Navpunkte an.

Abbildung 3-14: Script in Aktion