NAVAL FORCES

STREITMÄCHTE DER MEERE

Dokumentation

Die objektorientierte Programmierung des Netzwerkspiels „Naval Forces“ mit C++ und Microsoft DirectX™.

Von Valentin Schwind, Martin Goihl und Andre Grumbach im 2BKI2 – 2002/03 im Auftrag der Gottlieb-Daimler-Schule II Sindelfingen.

Vorwort

Nun ist es schon zu Ende das zweite und letzte Jahr im Berufskolleg. Anstrengend und sehr schwierig war die Zeit, und trotz all unserer Bemühungen um gute Noten und Zensuren, haben wir noch etwas mehr gelernt, als nur Formeln, Vokabeln und Texte. Wir haben mit unserer Projektarbeit etwas verwirklicht, was uns schon seit unserer Zeit vor dem Berufskolleg beschäftigt und Spaß gemacht hat – dem Spiel, der Programmierung und der Grafik. Und nun, da die Zeit vorüber ist, sind wir enttäuscht und traurig darüber, dass wir nur neun Monate mit der Programmierung von „Naval Forces“ verbringen konnten, denn nichts hat uns während unserer Zeit auf dem Berufskolleg mehr Spaß gemacht, als dieses Projekt.

Dies ist die Dokumentation und Entstehungsgeschichte von „Naval Forces“, ein „kleines“ Spiel dessen Titel treffender nicht sein kann. Und auch wenn das Programm schon so gut wie fertig ist, sind wir der festen Überzeugung, auch noch weit nach unserer Zeit am Berufskolleg, weiter an „Naval Forces“ zu arbeiten.

Mit freundlichen Grüßen

Andre Grumbach
Martin Goihl
Valentin Schwind

Inhalt

Diese Dokumentation ist in vier Hauptkategorien unterteilt, die in weiteren Unterkategorien gegliedert sind. Zum Schluss finden wir noch einiges zusätzlich Interessantes zum Spiel.

Das Spiel

1.1 Allgemeine Informationen

Das Projekt eines Netwerkspiels mit dem Namen „Naval Forces“ und dem Codenamen „Admiral“ umfasst die Objektorientierte Programmierung eines Multiplayerspiels (Internet & Lokales Netzwerk) mit der Programmiersprache C++. Das Spiel wird hierbei dem bekannten „Schiffe-Versenken“ ähneln, aber durch zahlreiche neue Features erweitert.

Die Programmierung des Spiels zeigte sich sehr viel schwieriger und aufwendiger als im Pflichtenheft beschrieben wurde. Wir bemühen uns nun sämtliche wichtige Programmroutinen und Abläufe darzustellen und zu beschreiben. Wir geben eine Übersicht über die wichtigsten Algorithmen und Objektstrukturen des Spiels.

Darüber hinaus sollte man sehen, dass es einen „endgültigen“ Zustand bei einem Spiel dieser Art nicht geben kann. Verbesserungen und Updates sind deshalb jederzeit möglich!

Teamadressen

Valentin Schwind

Klosterallee 9
73733 Esslingen
0711/38 53 52
Valentin@valisoft.de
ICQ: 115598917

Martin Goihl

Marienbader Weg 90
71067 Sindelfingen
07031/38 15 96
Martin@valisoft.de
ICQ: 97266437

Andre Grumbach

Tessinerstr.22
71069 Sindelfingen
07031/760866/67
Andre@valisoft.de
ICQ: 164383480

Firmenadresse

valisoft | interactive pictures

Klosterallee 9
73733 Esslingen
0711/385352
www.valisoft.com

Auftraggeber

Gottlieb-Daimer-Schule II

Böblingerstr. 73
Postfach 448
71065/46 Sindelfingen

Hr. Birk, Hr. Greiner

Leitung

Die Leitung des Projekts liegt bei den drei Teilnehmern. Das Projekt entsteht mit der Hilfe der „obligatorischen“ Firma valisoft | interactive pictures (weitere Informationen unter www.valisoft.com)

Ansprechpartner und Projektleiter sind die drei Teamteilnehmer unter oben genannten Adressen.

1.2 Voraussetzungen

Das Spiel und die Programmierung des Computerspiels „Naval Forces“ setzt einige Vorrausetzungen an Soft- und Hardware voraus, die beachtet und erfüllt werden müssen. In der späteren Programmierung wurde dies auch zu einigen Problemen, da wir eine kompatible Version zu den unterschiedlichsten Rechnern programmieren mussten.

Hardware (Mindestvorausetzungen):

- Intel Pentium II 400 MHz
- 128 MB RAM
- DirectX kompatible Grafikkarte
- Ethernet-Netzwerkkarte und eine andere Internet-/LAN-Verbindung

Betriebssystem

- Windows 98, Windows NT, Windows 2000, Windows XP
- Microsoft DirectX 8.1

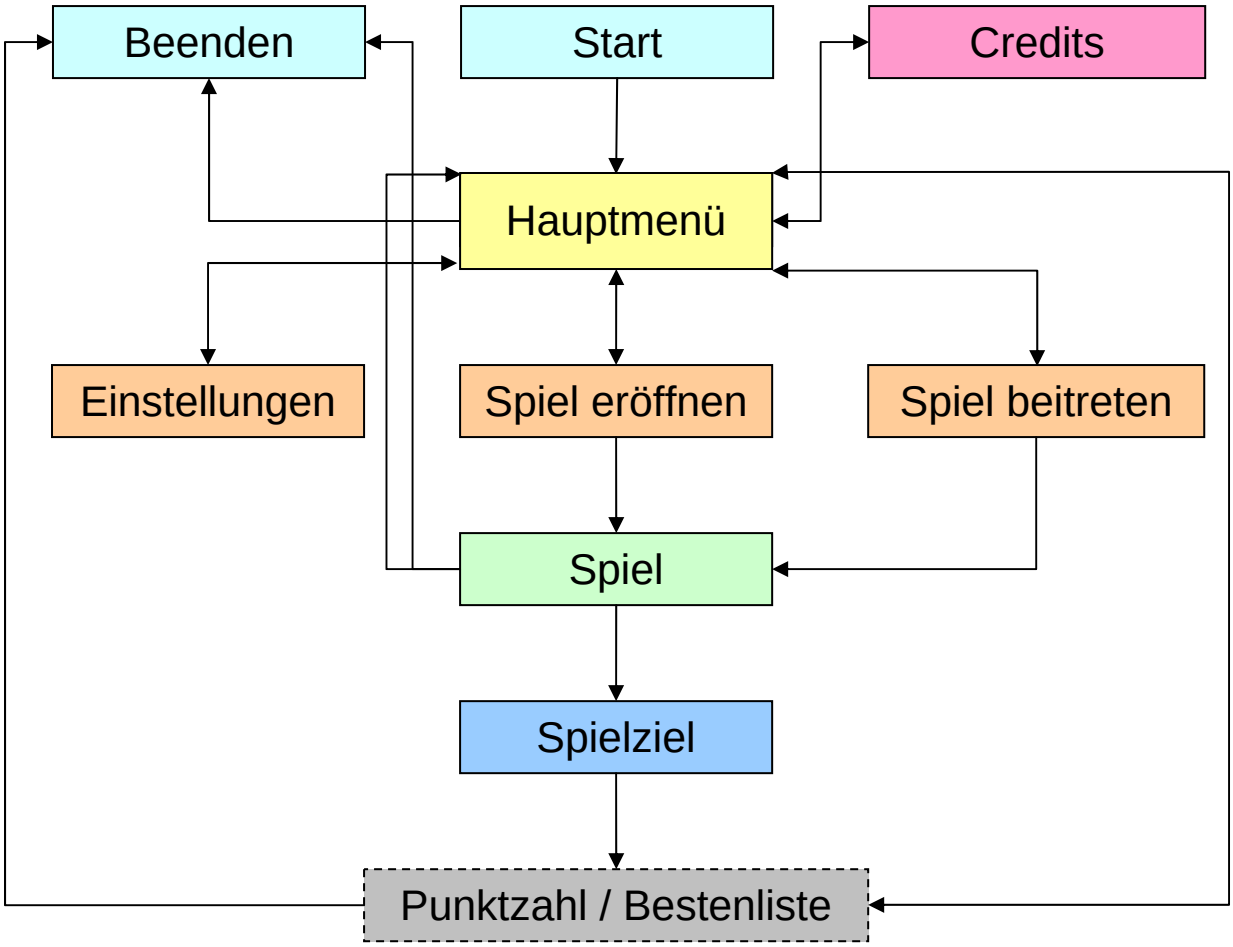
Sonstiges

Zur Programmierung und Bearbeitung aller Ressourcen, die für das Spiel erforderlich waren werden folgende Programme benötigt:

- Borland Builder 5 C++
- Corel Photo-Paint 11
- 3ds max 5
- Microsoft Word
- Microsoft DirectX SDK 8.1

Grundlegende Vorkenntnisse in C++ und für eine grafische Realisierung in 3ds max, sowie in einem 2D Programm, waren erforderlich. Wir empfehlen vor der Programmierung dringend Tutorials und die Onlinehilfe von DirectX zu studieren.

1.3 Schematischer Menüaufbau



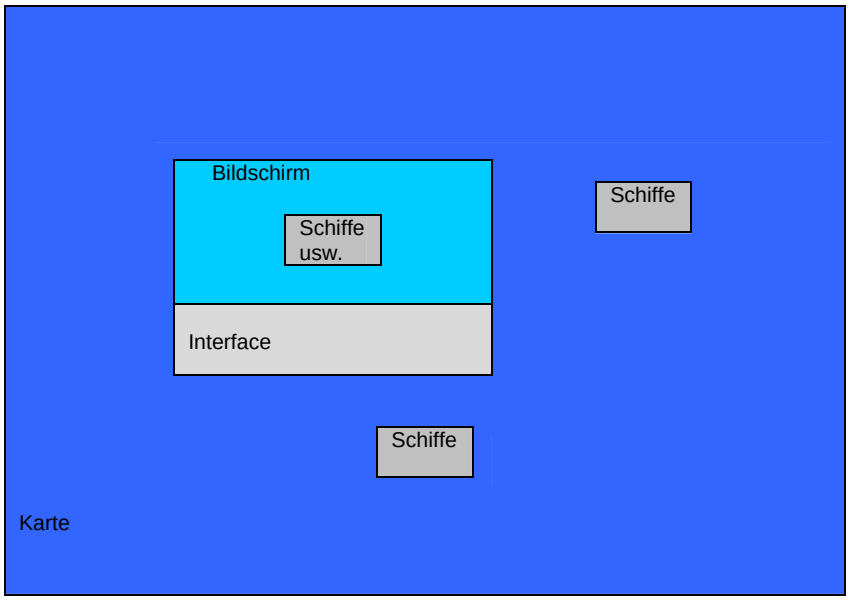
Einrichtung des Spiels
Hauptmenü (und Initialisierung des Programms)
Betriebssystemebene
Informationsebene
Spielebene (komplexester und aufwendigster Programmteil)
Spielschluss (Sieg/Niederlage/Unentschieden/Nur noch ein Spieler)
Noch nicht entschieden

Die Programmierung der einzelnen Menüpunkte erfolgt mit C++. Die Erstellung von Grafiken, Hintergründen, Bildern und grafischem Text erfolgt mit Corel PHOTO-PAINT.

Als Hilfe zur Umsetzung dienen Ressourcen, Tutorials (Übungsbände) und die Hilfe des Borland Builders. Zur grafischen Beschleunigung wird DirectX verwendet. Der offen gelegte Quellcode des Microsoft Programmiertools beschleunigt Grafik- und Netzwerkfunktionen des Spiels. Weitere optionale Funktionen (WinAPI, Ressourcen-Builder usw.) werden auch noch zur Programmierung verwendet.

1.4 Schematischer Spielaufbau

Das Spiel besteht aus einer „Map“ über die der Bildschirm „scrollt“ (die Programmiertechnische Realisierung sieht vor, dass Schiffe und „Map“ unter dem Bildschirm rollen). Es soll maximal vier verschiedene Spieler geben (Grün, Gelb, Rot, Blau) in vier verschiedenen Ecken einer editierbaren Karte. Für die Vollbilddarstellung auf dem Bildschirm wählten wir 800 x 600 und eine Farbtiefe von 32 Bit. Allerdings haben die meisten Bitmaps 256 Farben, um die Dateigröße gering zu halten. Das Spielprinzip ist das eines strategischen Onlinespiels zwischen verschiedenen Gegnern an unterschiedlichen Orten, um taktisch gegeneinander vorzugehen.



Die Technik

Informationen zu DirectX

Zur technischen Realisierung komplexer und äußerst schnell ablaufenden Funktionen, verwendeten wir DirectX SDK (Software Development Kit) von Microsoft. Ein Library und Treiberpaket zur hardwarebeschleunigten Programmierung. Der Vorteil von DirectX gegenüber anderen Grafikoptionen liegt auf Hand: Hardwarebeschleunigungen in nahezu allen Bereichen eines Computersystems, Programmierungsfreiheit, systemnahe Programmierung und weltweite Unterstützung. DirectX bietet optimalen Treibersupport und eine gigantische Bibliothek an Funktionen, die zu alle dem noch umsonst zum Herunterladen auf www.microsoft.com sind. Die Online-Hilfe zu Mircosoft DirectX finden wir unter www.msdn.com. Eine deutsche Hilfe gibt es leider noch nicht.

2.1 DirectDraw

DirectDraw ist die 2D-Komponente von DirectX. Es wird in der Regel in Multimediaprogrammen und Spielen verwendet. Die meisten unserer grafischer Spielevorbilder (Starcraft, Age of Empires) sind mit DirectDraw programmiert. Mit DirectDraw ist es möglich, direkt auf den Grafikspeicher zuzugreifen, in dem Bilder gespeichert und zu Szenen zusammengestellt werden können. Es gibt einige Ausdrücke, die man kennen sollte, bevor man DirectDraw verwendet:

Buffer:	In einem Buffer werden grafische Daten gespeichert (z.B. Bitmaps)
Primary Buffer (Primärbuffer):	Stellt die Bildschirmanzeige dar. Alles, was hierhin gespeichert wird, erscheint auf dem Bildschirm.
Blitting:	Wirft Grafiken auf den Bildschirm. Das bedeutet, dass Grafiken im RAM durch die **Grafikkarte** und nicht den Prozessor berechnet werden.
Flipping:	Freigeben des Speichers der Grafikkarten. Fehlt das Blitting werden Grafikelemente durch neue oder keine im RAM gelassen und nicht im Grafikspeicher überschrieben.
Frames per Second (FPS):	So oft wird die Anzeige in einer Sekunde neu gezeichnet.

Jede DirectDraw-Anwendung ist nach folgendem Muster aufgebaut:

Laden der Grafiken (einmalige Initialisierung)	
Schleifenanfang	
	Darstellung auf dem Bildschirm
	Freigeben von Speicher
Schleifenende	

Surfaces

Surfaces sind einer der wichtigsten Bestandteile von DirectDraw. Sie werden wie PictureBox-Steuerelemente in einer normalen C++ Anwendung verwendet, um grafische Daten zu

speichern, allerdings ohne einen sichtbaren Editor. Zwei spezielle Surfaces sind der Primary Buffer, der auf dem Bildschirm dargestellt wird und der Backbuffer, in dem eine Szene vor dem Anzeigen zusammengestellt wird.

Deklaration von DirectDraw:

```
class TFormMain : public TForm
{
  ...

  private:
        LPDIRECTDRAW         lpDD;           // Haupftunktions-Objekt
        LPDIRECTDRAWSURFACE  lpDDSPrimary;   // Die Grundfläche wird deklariert
        LPDIRECTDRAWSURFACE  lpDDSBack;      // Ein Zwischenspeicher wird deklariert
        LPDIRECTDRAWSURFACE  lpDDSOne;       // Unsere Hauptoberfläche wird erstellt
        LPDIRECTDRAWPALETTE  lpDDPal;        // Eine Farbpalette wird kreiert

        LPDIRECTDRAWSURFACE  lpDDSSchiff;    // Eine weitere Oberfläche wird erstellt

        BOOL FActive;                        // Zwei Hilfsvariablen für spätere Überprüfungen
        BOOL FRunApp;

        void UpdateFrame(void);              // UpdateFrame – Spätere Ablauffunktion
        void Start();                        // Initialisierungsfunktion
        HRESULT RestoreAll(void);            // Alles wiederherstellen, wenn es verloren geht

        MESSAGE void MyMove(TMessage &Message);     // MyMove:  aufgerufen durch den:
  public:                                           // MESSAGE HANDLER!!!
        __fastcall TFormMain(TComponent* Owner);
        BEGIN_MESSAGE_MAP
                  MESSAGE_HANDLER(WM_INFOSTART, TMessage, MyMove); //siehe unten!
        END_MESSAGE_MAP(TForm);

// Hier wird eine Prozessorabhängige Routine gebildet, die bei ihrem Beenden sofort wieder
// gestartet wird. MyMove ist hierbei die Funktion, die immer aufgerufen wird. Der
// MessageHandler stellte unser ganzes Programm auf den Kopf. Da der MessageHandler
// der DirectDraw-Funktion als Objekt in keiner Form deklarierbar war, ließ sich der
// MessageHandler in DirectPlay nicht in einer Klasser unterbringen, was die Programmierung
// enorm erschwerte.
}
```

Initialisierung von DirectDraw:

```
void TFormMain::Start()
{
 HRESULT ddrval;                    // Überprüfungsvariable
 DDSURFACEDESC ddsd;                // Oberflächenbeschreibung
 DDSCAPS ddscaps;                   // Umschaltemodus

 ddrval = DirectDrawCreate( NULL, &lpDD, NULL );
                                    // Erstellung einer DDraw Oberfläche

 ddrval = lpDD->SetCooperativeLevel(Handle, DDSCL_EXCLUSIVE | DDSCL_FULLSCREEN );
                                    // Vollbildmodus

 ddrval = lpDD->SetDisplayMode( 800, 600, 32);
                                    // Auflösung und Bitmodus

 ddsd.dwSize = sizeof( ddsd );
 ddsd.dwFlags = DDSD_CAPS | DDSD_BACKBUFFERCOUNT;
 ddsd.ddsCaps.dwCaps = DDSCAPS_PRIMARYSURFACE | DDSCAPS_FLIP | SCAPS_COMPLEX;
 ddsd.dwBackBufferCount = 1;

 ddrval = lpDD->CreateSurface( &ddsd, &lpDDSPrimary, NULL );
                                    // Erstellen der ersten Oberfläche
 ddscaps.dwCaps = DDSCAPS_BACKBUFFER;
 ddrval = lpDDSPrimary->GetAttachedSurface(&ddscaps, &lpDDSBack);
                                    // Eine zusätzliche Oberfäche wird generiert

 lpDDPal = DDLoadPalette(lpDD, szBitmap);
                                    // DDraw ermittelt die Farbpalette des zu ladenden Bildes
 lpDDSOne = DDLoadBitmap(lpDD, szBitmap, 0, 0);
                                    // DDraw lädt eine Bitmap in die Oberfläche
```

```
 DDSetColorKey(lpDDSOne, RGB(0,0,0));// Hier setzt DDraw den Farbkey für die transparente Key

 lpDDSSchiff = DDLoadBitmap(lpDD, szSchiff, 0, 0);

                              // DDraw ermittelt die Farbpalette des zu ladenden Bildes

// Bis auf alles andere ist diese Zeile veränderbar für alle weiteren Schiffstypen: Fregatte,
// Zerstörer usw.szSchiff muss hierbei mit einem Bildchen in einer Script Datei initialisiert
// werden.

 DDSetColorKey(lpDDSSchiff, RGB(255,0,255));//DDraw ermittelt die Farbpalette des zu ladenden
                                            // Bildes. In diesem Fall „Rosa“...
}
```

Funktionsaufruf über den MessageHandler (siehe oben):

```
MESSAGE void TFormMain::MyMove(TMessage &Message)
{
  do                                  // Do-While - Schleife
  {
    UpdateFrame();                    // Hauptfunktion
    Application->ProcessMessages();   // Hauptprozedur
  }
  while(FRunApp == true);             //
}
```

Funktionsaufruf UpdateFrame (+ gekachelte Unterfunktionen):

```
void TFormMain::UpdateFrame( void )
{
 {...}
}
```

Blitting-Funktion

Die normale Blitting - Funktion sieht eigentlich vor Grafiken einfach auf den Monitor zu „werfen“. Allerdings beinhalten auch die einfachsten Funktionen einige Probleme. Jedes Objekt, das geblittert werden soll, muss aus einer Grafik und einem Rechteck bestehen, sowie aus der Position zu der die obere linke Ecke des Rechtecks gebracht werden soll und dem ColorKey, der festlegt welche Farbe transparent gemacht werden soll. Dieser RGB (Rot/Grün/Blau)-Wert ist bei uns in der Regel auf 255, 0, 255 (rosa). Darüber hinaus werden Positionen die außerhalb des Monitors sind nicht mehr geblittert. Dafür mussten wir eine erweiterte Blitting-Funktion entwickeln (Clipping), die Positionen rechtzeitig korrigiert, Rechtecke richtig schneidet und die Grafiken dann so schnell wie möglich auf den Monitor wirft.

Unsere wichtigste Blitting-Funktion ist hier beschrieben:

```
void TFormGame::BlitterObject( int xpos , int ypos , IDirectDrawSurface
                        *DirectDrawSurface , tagRECT *RECT , long SRCCOLORKEY )
{
 HRESULT ddrval;                                    // Rückgabewert für erfolgreiches Blitten

 if( xpos < 0 )                                     //
 {                                                  //
  RECT->left = RECT->left - xpos;                   //
  xpos = 0;                                         //
 }                                                  // Das sogenannte Clipping berechnet vor
 if( ypos < 0 )                                     // dem Flipping, dass die Kanten richtig
 {                                                  // abgeschnitten warden.
  RECT->top = RECT->top - ypos;                     //
  ypos = 0;                                         // Diese wenigen Zeilen haben mehr als
 }                                                  // 13 Stunden gebraucht.
 if( xpos + RECT->right - RECT->left > 800 )        //
 {                                                  //
  RECT->right = 800 - xpos + RECT->left;            //
 }                                                  //
 if( ypos + RECT->bottom - RECT->top > 600 )        //
```

```
 {                                                    //
  RECT->bottom = 600 - ypos + RECT->top;              //
 }                                                    //

 while( 1 )
 {
  ddrval = DDraw.lpDDSBack->BltFast( xpos , ypos , DirectDrawSurface  ,RECT , SRCCOLORKEY |
                                                                       DDBLTFAST_WAIT);
                                                     // Blitterung und ihre Übergabewerte

  if( ddrval == DD_OK ) break;                       // Überprüfungsverfahren bei
  if( ddrval == DDERR_SURFACELOST )                  // möglichen Fehlern...
  {
   ddrval = RestoreAll();
   if( ddrval != DD_OK ) return;
  }
  if( ddrval != DDERR_WASSTILLDRAWING ) return;
 }
}
```

Flipping-Funktion

Das Flipping bezieht sich auf den ganzen Grafikspeicher und muss am Ende jedes Programmdurchlaufs einmal ausgeführt werden. Diese Funktion besteht eigentlich nur noch aus Überprüfungen und dem Aufruf der Flipping-Funktion. An diesen Zeilen sollte nichts geändert werden.

Flipping-Funktion:

```
void TFormGame::Flipper( void )
{
 HRESULT       ddrval;

 while( 1 )
 {
  ddrval = DDraw.lpDDSPrimary->Flip( NULL, 0 );
  if( ddrval == DD_OK ) break;
  if( ddrval == DDERR_SURFACELOST )
  {
   ddrval = RestoreAll();
   if( ddrval != DD_OK ) break;
  }
  if( ddrval != DDERR_WASSTILLDRAWING ) break;
 }
```

2.2 DirectPlay

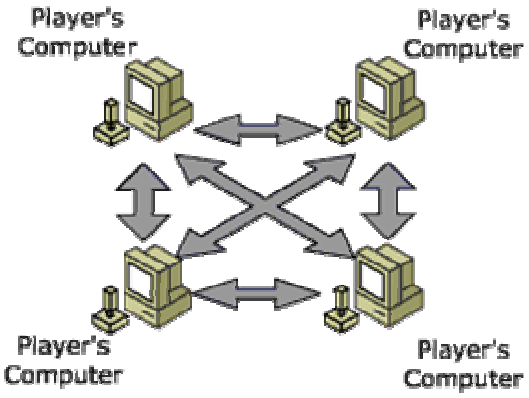
DirectPlay erleichtert einem das Programmieren von Netzwerkspielen. Es kann eine Verbindung über das LAN, das Internet oder ein Modem aufnehmen. Ist eine Verbindung hergestellt, können die Daten relativ einfach ausgetauscht werden. Im folgenden Quelltext werden die wesentlichen Funktionen geschrieben die man benötigt.

Jede DirectPlay-Anwendung ist nach folgendem Muster aufgebaut:

Initialisieren von DirectPlay
Auflisten der Auswahlmöglichkeiten (z.B. Verbindungsarten)
Sitzung starten / beitreten
Kommunikation innerhalb der Sitzung
Beenden von DirectPlay

DirectPlay bietet einem mehrere Möglichkeiten sein Spiel aufzubauen:

1. Peer-to-Peer
2. Client / Server



Unser Spiel basiert auf der Peer-to-Peer Topologie die den Vorteil bietet, dass sie recht einfach und anpassungsfähig ist. Man braucht nur einige Clients und einige Funktionen um diese zu Organisieren.

Die maximale Anzahl der Spieler hängt natürlich von der Bandbreite und dem Spiel aber 20-30 Spieler sollte man mit Peer-to-Peer immer verwalten können.

Deklaration von DirectPlay

```
IDirectPlay8Peer        *dpPeer;     // Das DPlay8-Peer-Device

IDirectPlay8Address     *dpDevAd;    // Die Adresse für das DirectPlay-Device

IDirectPlay8Address     *dpHostAd;   // Die Adresse für den Host

GUID NV_FORCES = { 0xe47ba3a5, 0xd874, 0x4086, { 0x91, 0xf3, 0xa6, 0xe, 0x4a, 0xa0, 0xf6, 0x89
} };
// {E47BA3A5-D874-4086-91F3-A60E4AA0F689}
```

Jede Anwendung braucht einen Global Unique Identifier damit man es von anderen Anwendungen unterscheiden kann.

Initialisierung von DirectPlay

```
CoInitialize( NULL ); // Schnittstelle für COM zur Verfügung stellen

CoCreateInstance( CLSID_DirectPlay8Peer, NULL, CLSCTX_INPROC_SERVER,
                      IID_IDirectPlay8Peer, (LPVOID *) &dpPeer );

dpPeer->Initialize( NULL, DPlayMessageHandler, 0 );

CoCreateInstance( CLSID_DirectPlay8Address, NULL, CLSCTX_INPROC_SERVER,
                      IID_IDirectPlay8Address, ( LPVOID * ) &dpDevAd );
```

```
CoCreateInstance( CLSID_DirectPlay8Address, NULL, CLSCTX_INPROC_SERVER,
                      IID_IDirectPlay8Address, ( LPVOID * ) &dpHostAd );

dpDevAd->SetSP( &CLSID_DP8SP_TCPIP );

dpHostAd->SetSP( &CLSID_DP8SP_TCPIP );
```

Beim Initialisieren überprüft man welche Protokolle installiert sind und wählt eines davon aus. Außerdem muss man noch einige Schnittstellenfreigeben und den DirectPlayMessageHandler initialisieren.

DirectPlayMessageHandler

Der hier verwendetete DirectPlayMessageHandler bietet Systemnachrichten bei denen er aktiviert wird. Diese Nachrichten dienen dazu, dass man verschiedene Aktionen von Spielern leicht erkennen kann.

```
HRESULT WINAPI DPlayMessageHandler( PVOID pvUserContext, DWORD dwMessageType, PVOID pMessage )
{
 switch( dwMessageType )
 {
  case DPN_MSGID_CREATE_PLAYER:
  {
     [...] //Wird aufgerufen sobald ein neuer Spieler sich erfolgreich verbunden hat
  }

  case DPN_MSGID_RECEIVE:
  {
     [...] // Wird Aufgerufen sobald man Daten an einen gesendet wurden
  }
 }
}
```

Spiel starten/beitreten

Jedes Spiel benötigt einen „Host“ der eine Sitzung für ein Spiel öffnet und der verschiedene Optionen hat um dieses Spiel zu Verwalten. Nun können mehrere Clients diesem Spiel beitreten.

```
bool HostSpiel()
{
 [...] // Einlesen der Werte die man zum Starten des Spiels benötigt und ausführen der Host
Funktion

 dpPeer->Host(&appDesc, &dpDevAd, 1, NULL, NULL, NULL, 0);
}
```

```
bool JoinSpiel()
{
 [...] // Einlesen der Werte die man zum Beitreten in ein Spiel benötigt und ausführen der
Join Funktion


 dpPeer->Connect(&appDesc, dpHostAd, dpDevAd, NULL, NULL, NULL, 0, NULL, NULL, NULL,
DPNCONNECT_SYNC);
}
```

Mit dieser Funktion kann ein „Client“ einem bereits geöffneten Spiel beitreten. Sobald mehrere Spieler sich in einer Sitzung treffen fangen sie an für das Spiel relevante Daten auszutauschen.

```
void SendFunktion()
{
 [...] // Berechnen der Größe des Pakets das versendet werden soll und aufrufen der SendTo
Funktion

 DpPeer->SendTo( DPNID_ALL_PLAYERS_GROUP, &dpnBuffer, 1, 0, NULL, NULL, DPNSEND_SYNC |
DPNSEND_NOLOOPBACK) ;
}
```

Die SendTo Funktion ist eine der effektivsten Möglichkeiten die DirectPlay bietet Daten zu versenden. Außerdem bietet DirectPlay mehrere Flags um diese Funktion seiner Anwendung perfekt anpassen zu können.

Die Programmierung

3.1 Dateien

Die Programmstruktur unseres Projektes wird durch die Verwendungen von Funktionen bestimmt. Durch die schlichte Projektverwaltung des Borland Builders, durch Headerdateien und durch die problematische Verwendung der DirectDraw und DirectPlay MessageHandler hat sich eine Dateistruktur entwickelt, die nicht verändert werden sollte. Wir empfehlen die Dateiköpfe auf keinen Fall zu verändern. Das schließt auch die Einbindung von neuen Dateien mit ein. Dies bedeutet dass man nichts verändern sollte, da es so funktioniert und recht selten anders!

Name	Beschreibung	Dateien
Quellcodes	Hauptverzeichnis	/src/
Projektdatei	Hauptprojektdatei des Quellcodes	navalforces.bpr navalforces.cpp
Startfenster	Kleines Startmenü beim Beginn des Spiels	nf_main.cpp nf_main.h
Hauptmenü	Das Vollbildmenü des Startbildschirms.	nf_menu.cpp nf_menu.h
Netzwerkinhalte	Netzwerkform und Netzwerkfunktionen inkl. Sendestruktur des Quellcodes	nf_net.cpp nf_net.h
Spieldatei	Hauptspiel, DirectDraw Routinen, Anzeige usw. des Quellcodes	nf_game.cpp nf_game.h
Schiffsdatei	Schiffe (Schiffe, Funktionen und Werte) des Quellcodes	nf_ships.h
Spieldatei	Datei zum Ausführen des Quellcodes	navalforces.exe
Bilddateien	Die Dateien im Image-Verzeichnis werden die Bilder, die im Spiel benötigt werden geladen.	/images/
	Wasser	waves256.bmp
	Interface	gameface256.bmp
	Fregatte	frigate256.bmp
	Zerstörer	destroyer256.bmp
	Kreuzer	cruiser256.bmp
	U-Boot	submarine256.bmp
	Schlachtschiff	battleship256.bmp
	Untergrund	underground256.bmp
	Effekte	effects256.bmp
Scriptdatei	Die RC-Datei ist ein Ressourcenscript, dass externe Dateien in die EXE laden kann, ohne im unkompilierten Quellcode geladen werden zu müssen	nf_graphics.rc
3D Zerstörer	3ds max Datei des Zerstörers	Destroyer.max
3D U-Boot	3ds max Datei des U-Boots	Uboot.max
3D Schlachtschiff	3ds max Datei des Schlachtschiffs	Schlachtschiff.max
3D Kreuzer	3ds max Datei des Kreuzers	Cruiser.max
3D Fregatte	3ds max Datei der Fregatte	Fregatte.max
Beitreten-Menü	Corel PhotoPaint Datei der Beitreten-Form	beitreten-bg.cpt
Credits -Menü	Corel PhotoPaint Datei der Credits-Form	credits.cpt
Eröffnen -Menü	Corel PhotoPaint Datei der Eröffnen-Form	eröffnen-bg.cpt

Hauptmenü - Menü	Corel PhotoPaint Datei der Hauptmenü-Form	mainmenu-bg.cpt
Einstellungen - Menü	Corel PhotoPaint Datei der Einstellungen-Form	settings-bg.cpt
Startbildschirm -Menü	Corel PhotoPaint Datei der Startbildschirm-Form	startscreen-bg.cpt

3.2 Objektorientierte Programmierung

Das Problem mit vielen Einheiten (Schiffen), die gesteuert werden mussten konnte mit Funktionsgesteuerten Abläufen innerhalb der DirectDraw – Funktionen nicht mehr realisiert werden. So mussten wir auf die Objektorientierte Programmierung umsteigen. Dazu erstellten wir folgende Klassen.

3.2.1 Spielerklasse

Die Spielerklasse wird maximal 4x dynamisch erstellt. Die Aggregation beinhaltet nun Zeiger auf die abgeleiteten Schiffsklassen (siehe unten). Diese können dann im Spiel ebenfalls dynamisch erstellt werden.

```
class TPlayer
{
 public:

 unsigned char ID;
 AnsiString    Name;
 unsigned char Color;

 TFrigate      *Frigate;
 TDestroyer    *Destroyer;
 TCruiser      *Cruiser
 TSubmarine    *Submarine
 TBattleship   *Battleship;
};
```

3.2.2 Hauptschiffsklasse

Um allen Schiffen gemeinsame und nicht gemeinsame Funktionen zu geben mussten wir eine Hauptschiffsklasse erstellen, von der später alle weiteren Schiffe abgeleitet werden.

```
class TShipMain
{
 // Variablendeklaration ------
 // z.B. Xpos und Ypos etc.

 // Funktionen ----------------

 bool SelectShip ( short int, short int, short int, short int );
 // Überprüfung ob sich ein Schiff innerhalb der angegebenen Koordinaten befindet

 bool IsShipBetweenBorders( short int,short int,short int,short int,short int,short int,bool);
 // Überpürfung ob sich das Schiff innerhalb eines Rahmens befindet

 void MoveShip( short int, short int, short int, short int );
 // Bewegungsberechnung

 void Go( void );
 // Kürzester Weg

 void Timer( void );
```

```
 // Selbstprogrammierter Timer für Geschwindigkeitsberechnung

 void Accelerate( void );
 // Beschleunigen

 void Stop( void );
 // Stoppen und Bremsen

 void Turn( void );
 // Drehfunktion

 void GetDistance( void );
 // Entfernungsberechnung

 void SetStartValues( short int, short int, unsigned char _color );
 // Startwerte jedes Schiffes


 void SetTarget( short int, short int, short int, short int );
 // Zielübergabe an das Schiff

 void Attacking( void );
 // Angriffsfunktion

 void Explosion( void );
 // Explosionen
}
```

3.2.3 Abgeleitete Klassen

```
class TFrigate    : public TShipMain {...};
class TDestroyer  : public TShipMain {...};
class TCruiser    : public TShipMain {...};
class TSubmarine  : public TShipMain {...};
class TBattleship : public TShipMain {...};

// Ein Flugzeugträger konnte bis zum jetzigen Zeitpunkt nicht realisiert werden. Flugzeuge
// erwiesen sich bis jetzt als noch zu komplex.
```

3.3 UML-Übersicht

Zur Übersichtlichkeit der verwendeten Klassen haben wir zusätzlich dieses UML-Klassendiagramm erstellt.

Die Klasse ***ShareShipData*** ist die Klasse, dessen Objekte im Programm später als „minimale Datenmenge von Schiffen“ versendet werden. Um die gesendete Datenmenge gering zu halten haben wir hier nur die wichtigsten Daten zum Versenden bereitgestellt.

Hierbei wird in der Klasse ***ShareData*** eine maximale Anzahl von Schiffen festgelegt die schließlich versendet werden können.

Die Klasse ***startvalue*** setzt alle Startwerte fest (Spieler, Schiffe usw.).

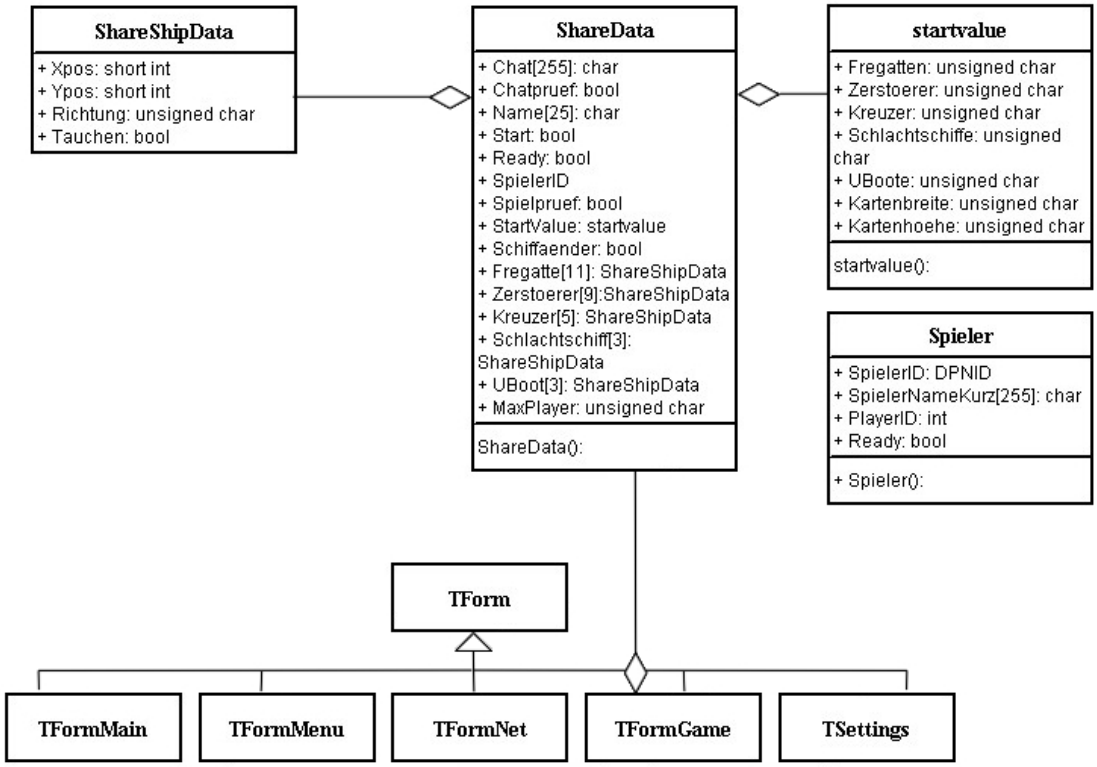
Die Klasse ***Spieler*** wird von DirectPlay benötigt, dessen DPNID die ID jedes Spielers vergibt.

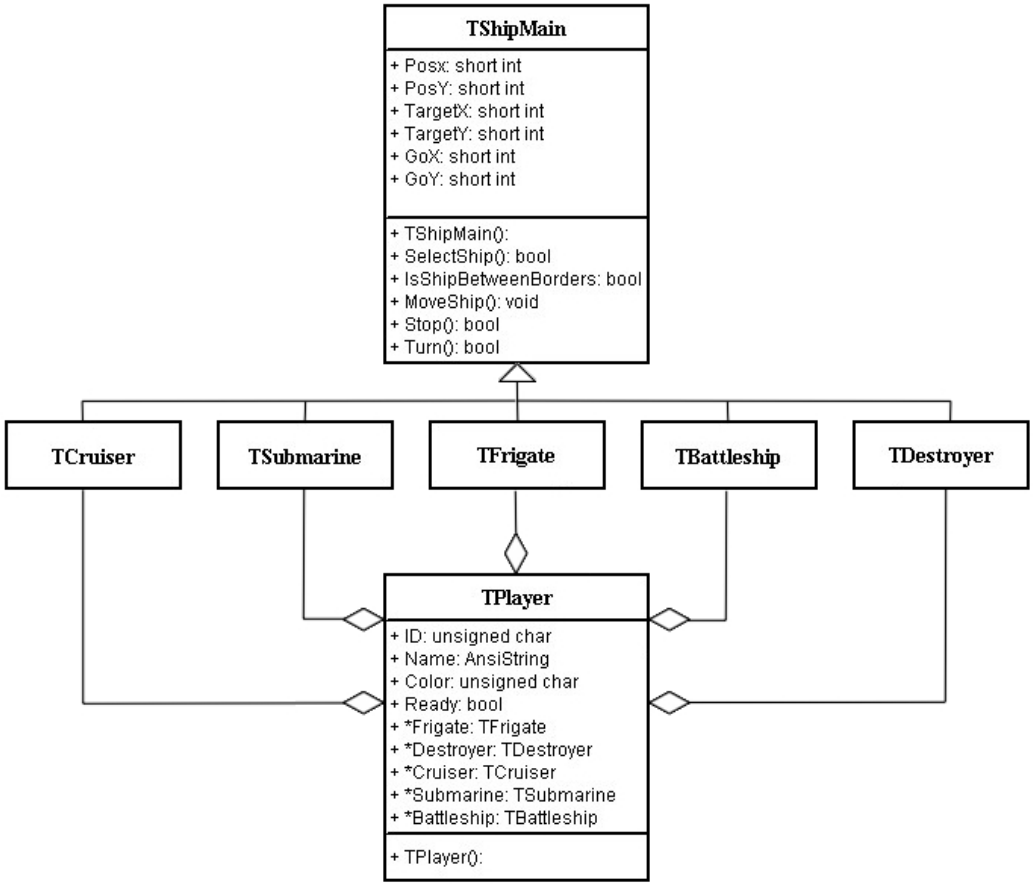
Die abgeleitenden Klassen der ***TForm*** bilden das Menü. Um den Netzwerkteil von dem eigentlichen Spiel auseinander zu halten, ist die UML des Spiels und die UML des Netzwerkteils getrennt - so wird es für später noch ermöglicht einen Singleplayer

hinzuzufügen. Aus Platzgründen sind nicht alle Attribute der Schiffsklassen und anderen Klassen erwähnt.

Die Klasse ***TShipMain*** und ihre vererbten Unterklassen sind selbsterklärend.

Die Klasse ***TPlayer*** ist die Zusammenführung aller Schiffe zu einem Spieler. Die Zeiger ermöglichen deren dynamische Erzeugung.


ShareShipData
+ Xpos: short int
+ Ypos: short int
+ Richtung: unsigned char
+ Tauchen: bool
ShareData
+ Chat[255]: char
+ Chatpruef: bool
+ Name[25]: char
+ Start: bool
+ Ready: bool
+ SpielerID
+ Spielpruef: bool
+ StartValue: startvalue
+ Schiffaender: bool
+ Fregatte[11]: ShareShipData
+ Zerstoerer[9]:ShareShipData
+ Kreuzer[5]: ShareShipData
+ Schlachtschiff[3]: ShareShipData
+ UBoot[3]: ShareShipData
+ MaxPlayer: unsigned char
ShareData():
startvalue
+ Fregatten: unsigned char
+ Zerstoerer: unsigned char
+ Kreuzer: unsigned char
+ Schlachtschiffe: unsigned char
+ UBoote: unsigned char
+ Kartenbreite: unsigned char
+ Kartenhoehe: unsigned char
startvalue():
Spieler
+ SpielerID: DPNID
+ SpielerNameKurz[255]: char
+ PlayerID: int
+ Ready: bool
+ Spieler():
TForm
TFormMain
TFormMenu
TFormNet
TFormGame
TSettings


TShipMain
+ Posx: short int
+ PosY: short int
+ TargetX: short int
+ TargetY: short int
+ GoX: short int
+ GoY: short int
+ TShipMain():
+ SelectShip(): bool
+ IsShipBetweenBorders: bool
+ MoveShip(): void
+ Stop(): bool
+ Turn(): bool
TCruiser
TSubmarine
TFrigate
TBattleship
TDestroyer
TPlayer
+ ID: unsigned char
+ Name: AnsiString
+ Color: unsigned char
+ Ready: bool
+ *Frigate: TFrigate
+ *Destroyer: TDestroyer
+ *Cruiser: TCruiser
+ *Submarine: TSubmarine
+ *Battleship: TBattleship
+ TPlayer():

3.4 Methoden der Schiffe

3.4.1 SelectShip(…) & IsShipBetweenBorders(…):

Überprüft ob innerhalb der Koordinaten, die Position des Schiffes liegt. Der Quellcode ist selbsterklärend.

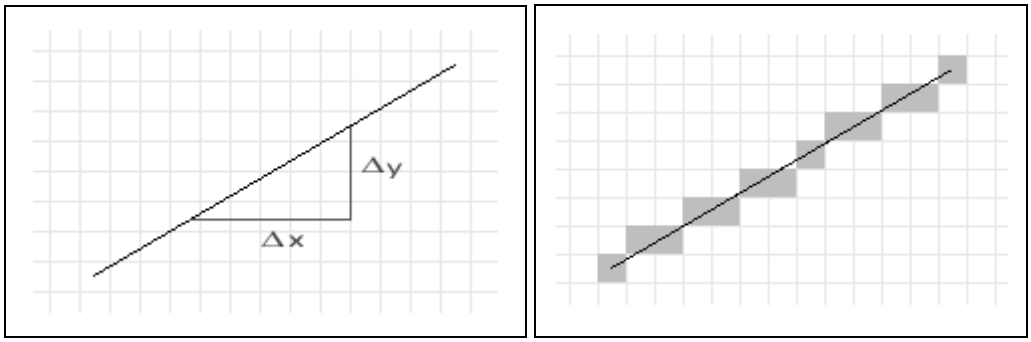
3.4.2 MoveShip(…):

Gibt dem Schiff ein Ziel – selbsterklärend.

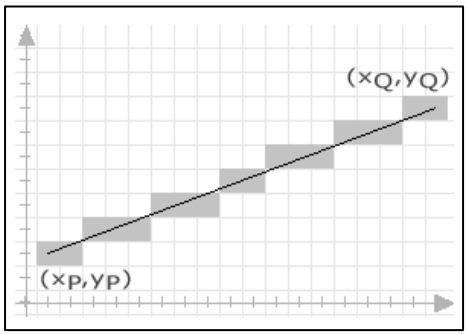
3.4.3 Go():

Der kürzeste Weg von A nach B innerhalb eines zweidimensionalen Rastersystems wurde 1969 von J.H. Bresenham für grafikfähige Rechner entwickelt:

Gegeben ist das Pixelraster und die Delta-Abstände zwischen den Zielen



Die Herleitung des Verfahrens:



Präziser ausgedrückt:

$$xP < xQ$$
$$yP <= yQ$$
$$yQ - yP <= xQ – xP$$

Wobei der letzte Ausdruck besagt, dass die Steigung der Geraden <= 1 ist. Das Ziel ist es, für jeden Integer Wert von xP,xP+1,xP+2, ... ,xQ denjenigen y-Wert zu finden, der die gesuchte Linie am besten approximiert.

Der exakte y-Wert für einen gegebenen x-Wert *x* kann wie folgt berechnet werden (sei m die Steigung der Geraden):

$$y_{exact} = yP + m \cdot (x - xp)$$

Der exakte y-Wert kann somit durch runden des obigen Ausdrucks gefunden werden. Da sich zwei aufeinanderfolgende x-Werte um 1 unterscheiden (x wird ja immer um 1 inkrementiert), ist die Differenz der dazugehörenden y-Werte gleich m, denn es gilt:

$y_{exact}\_k = yP + m \cdot (x - xp)$
$y_{exact}\_k + 1 = yP + m \cdot ((x + 1) - xp) = yP + m \cdot (x - xp) + m$
$y_{exact}\_k + 1 - y_{exact}\_k = m$

Herleitung:

Solange,dass $|y_Q - y_P| <= |x_Q - x_P|$ gilt, bleibt x die unabhängige Variable (d.h. wird bei jedem Schritt inkrementiert). Andernfalls tauschen x und y die Rollen. Zudem können Symmetrien berücksichtigt werden.

Daraus ergibt sich schliesslich der folgende Pseudocode:

```
* yP: y-value of the startpoint
* xP: x-value of the startpoint
* xQ: x-value of the endpoint
* yQ: y-value of the endpoint
 ******************************/
 drawLine(xP, yP, xQ, yQ) {
       x = xP;
       y = yP;
       D = 0;
       HX = xQ - xP;
       HY = yQ - yP;
       xInc = 1;
       yInc = 1;

       if(HX < 0) {
              xInc = -1; HX = -HX;
       }
       if(HY < 0) {
              yInc = -1; HY = -HY;
       }
       if(HY <= HX) {
              c = 2 * HX; M = 2 * HY;
              while(true) {
                     putPixel(x, y, g);
                     if(x == xQ) {
                            break;
                     }
                     x += xInc;
                     D += M;
                     if(D > HX) {
                            y += yInc; D -=c;
                     }
              }
       }
       else {
              c = 2 * HY; M = 2 * HX;
              while(true) {
                     putPixel(x, y, g);
                     if(y == yQ) {
                            break;
                     }
                     y += yInc;
                     D += M;
                     if(D > HY) {
                            x += xInc; D -=c;
                     }
              }

       }
 }
```

Der Originalquellcode wurde schließlich an diesen hier angepasst, ändert sich aber nur noch in den Variablennamen. Die beiden while()-Schleifen haben sich erübrigt, da das Schiff langsam und nicht auf einmal an sein Ziel fahren soll.

3.4.4 Timer():

Jedes Schiff besitzt seinen eigenen Timer für zeitabhängige Abläufe wie Bremsweg oder Beschleunigung. Dabei werden in einem vom System gegebene Zeitintervalle bestimmte Abläufe ausgeführt, dass mit GetTickCount() ermittelt wird. In dieser Funktion sehen wir auch schon den größten Teil des Timers – größtenteils selbsterklärend.

```
TimeTickCount = GetTickCount();

if( Tick == true )
{
 TimeTickNew = TimeTickCount + TimeInterval;
 Tick = false;
 Accelerate();
}
else
{
 if( TimeTickNew <= TimeTickCount )
 {
  Tick = true;
 }
}
```

3.4.5 Accelerate():

Jedes Schiff hat eine Geschwindigkeit. Die Beschleunigung wird über den Timer an diese angeglichen und einfach erhöht. Zu beachten ist jedoch auch die Entfernung und eine relative Zuweisung von Werten an reale Werte (in unserem Fall haben wir die realen maximalen Geschwindigkeitswerte verwendet, die die Schiffstypen besitzen z.b. Fregatte 35 kn).

3.4.6 Stop():

Die Geschwindigkeit wird solang verringert bis das Schiff zum stehen geblieben ist und das Bewegungsziel wird der Schiffsposition gleichgesetzt).

3.4.7 Turn():

Die Drehungsfunktion eines Schiffes wird ermittelt aus 36 Bildern für jedes Schiff (das Verfahren zur Erstellung dieser Bilder siehe im Kapitel 3D Bearbeitung). Dies ermöglicht ein fast flüssiges Drehen in alle Richtungen und eine relativ hohe Gradauflösung (ein Kreis beschreibt 360°). Die Winkelfunktion – in diesem Fall der Tangens - ermitteln wir aus dem DeltaX und DeltaY zum Ziel. Wir nehmen nun alle vier Quadranten, in die das Schiff wandern kann, überprüfen wohin es sich drehen soll und erhöhen/verringern den Bildindex, bis das richtige Bild zur entsprechenden Drehung angezeigt wird.

36 Bilder pro Schiff mal die Farben (Grün, Rot, Blau, Gelb, Silber)

für…

```
if( DeltaX >= 0 && DeltaY >= 0 )            // 1. Quadrant – unten rechts
if( DeltaX >= 0 && DeltaY <= 0 )            // 2. Quadrant – unten links
if( DeltaX <= 0 && DeltaY >= 0 )            // 3. Quadrant – oben rechts
if( DeltaX <= 0 && DeltaY <= 0 )            // 4. Quadrant – oben links
```

…wird bei negativen Werten inkrementiert, da aus negativen Werten keinen Tangens ziehen kann.

```
tanAlpha = ArcTan2( DeltaY ,DeltaX );       // ArcTangens aus den positiven Abständen
tanAlpha = ceil( tanAlpha * ( 180/M_PI ) ); // Runden und in Grad (DEGREE) umwandeln
```

Nun erhöht/verringert die Funktion den Wert des Richtungsindexes und erhöht/verringert den aktuellen Schrittweise, damit das Schiff nicht plötzlich sich in die gewählte Richtung dreht. Dieser Index wird nun bei der Blitterung genommen. Mit einem Modular und einem Faktor (der Höhe und Breite jedes Schiffbildchens) multipliziert.

3.4.8 GetDistance():

Berechnet über den Pythagoras den Abstand zwischen Schiff und Ziel – sonst selbsterklärend.

3.4.9 SetStartValues():

Setzt alle Startwerte, die vom Programm bestimmt werden (Positionen, Freund-/Feinerkennung…) – sonst selbsterklärend,

3.4.10 SetTarget():

Übergibt dem Schiff ein Ziel – selbsterklärend.

Alle weiteren oder nicht genannten Funktionen sind entweder inaktiv oder z.Zt. kommentiert. Wir bitten dies zu diesem Zeitpunkt zu entschuldigen, da das Spiel in zahllosen Bereichen optimiert werden kann/muss.

3.5 Hauptfunktionen im Spiel

Eine komplette Übersicht aller Funktionen scheint uns angesichts des enormen Umfangs unseres Projekts zwar als angemessen, eine genaue Beschreibung und eine Dokumentation aller Ereignisse sind allerdings aufgrund des Umfangs abzuarbeitender Funktionen nicht realisierbar. Wir empfehlen wieder an dieser Stelle alle Funktionen, die so zusammenhängen nicht zu verändern oder gar zu löschen. Zusätzliche Erweiterungen in der Funktion UpdateFrame(); sind natürlich möglich.

3.5.1 UpdateFrame():

Weiter unten werden wir den MessageHandler sehen, der die MyMove-Funktion aufruft. Diese wiederum ruft die Update Funktion auf, die in unserem Programm den zentralen Kern darstellt. Der GetTickCount bewirkt, dass bei schnelleren Prozessoren das Programm nicht zu schnell abläuft. Es bremst also das Programm. Eine Änderung der 10 ms wird nicht empfohlen (Bei 41 ms pro Bild sieht das Auge die Bewegungen noch flüssig). Wir sehen in der UpdateFrame folgende Funktionen, die in der Reihenfolge der angezeigten Ebenen auftauchen:

```
void TFormGame::UpdateFrame( void )
{
 int startzeit;

 startzeit = GetTickCount();

 Level();
       // Die Levelfunktion berechnet die 1. Ebene, den Hintergrund des Spiels.

 ShowShips();
       // Zeigt über die Blitterfunktionen die 2.Ebenen – alle Schiffe – an.

 ShowEffects();
       // Zeigt alle Effekte im Spiel

 Fog();
       // Zeigt den „Kriegsnebel“ – Schwarze Felder an Stellen wo die Schiffe nicht sehen
       // können.

 Mover( MouseClickX, MouseClickY );
       // Ein zusätzlicher Mauszeiger, dessen Ebene sich ändert, wenn man z.B. rechts klickt.

 ShowBorder();
       // Zeigt den Markierungsrahmen an, der um die Maus hängt, wenn man einen Rahmen zieht.

 Chat();
       // Die Chatebene. Eine Funktion, die auch Netzwerkroutinen beinhaltet.

 Interface();
       // Alle Oberflächen des Menüs, der Knöpfe, der Schalter etc.

 MouseStatus();
       // Mausfunktionen

 Flipper();
       // Flipping - Funktion

 while( ( GetTickCount() - startzeit ) < 10 ){};
}
```

3.5.2 Anzeige anhand eines Beispiels: Level() & ShowShips():

Die Level Funktion stellt später nur das Wasser da, anhand dieser Funktion lässt sich gut das Prinzip aller anderen Anzeige-Funktionen erklären.

Zu Beginn haben wir in der Strukturvariablen (char) DDraw.lpDDSLevel ein Bild mit 5 Wassermotiven in waagrechter Reihenfolge. Diese sollen nun im Spiel gekachelt und alle halbe Sekunde zufällig angezeigt werden.

Map/Level-Darstellung.

```
void TFormGame::Level ( void )
{
 RECT            rcRect;                          // Ein Rechtecks-Datentyp
 WaveCounter++;                                   // Der Wassertimer

 if ( WaveCounter > 1000 ) WaveCounter = 0;       //  ...soll  wieder  zurückgesetzt  werden,
                                                  // wenn Timer überläuft
 randomize();                                     // Randomizefunktion für ein zufälliges
                                                  // Bild

 for ( int y = 0; y <= int( MapGridHeight ); y++ )  // Für die Höhe der Karte
 {
  for ( int x = 0; x <= int( MapGridWidth ); x++ )  // Für die Breite der Karte
  {
  if ( WaveCounter == 0   ||
       WaveCounter == 100 ||
       WaveCounter == 200 ||
       WaveCounter == 300 ||
       WaveCounter == 400 ||
       WaveCounter == 500 ||
       WaveCounter == 600 ||
       WaveCounter == 700 ||
       WaveCounter == 800 ||
       WaveCounter == 900 ||
       WaveCounter == 1000 ) WaveCounter = random(1000);

       rcRect.left   = 256 * ( WaveCounter / 100 );        // Rechtechtsberechnung
       rcRect.top    = 0;                                  // zum Anzeigen des nächsten der
       rcRect.right  = 256 * ( WaveCounter / 100 ) + 256;  // waagrechten Bildchen
       rcRect.bottom = 256;

       BlitterObject ( ( x * 256 ) - ScreenLeft, ( y * 256 ) - ScreenTop, DDraw.lpDDSLevel ,
       &rcRect, DDBLTFAST_SRCCOLORKEY );

       // Die selbstgeschriebene BlitterFunktion wirft unser Bild auf den Monitor, schneidet
       // es auf den Monitor zurecht, da es ohne einem korrekten Schnitt und außerhalb des
       // Monitors nicht angezeigt werden kann.
  }
 }
}:
```

ShowShips ist die Anzeigefunktion für alle Schiffe und deren evtl. Markierungen.

```
void TFormGame::ShowShips( void )
{
 RECT         rcRect; // Ein Rechteckstyp

[...]  // Zusätzliche Anzeigetypen wie z.B. das Selektieren eines Schiffs werden davor
       // ausgeführt, da sie unter den Schiffen angezeigt werden sollen.

 for( unsigned char j = 0; j <= Players - 1; j++ ) // Für alle Spieler
 {
  for ( unsigned char i = 0; i <= Frigates - 1; i++ ) // Für alle Fregatten
  {
   Player[j].Frigate[i].Turn(); // Drehung
   Player[j].Frigate[i].SetRect(); // Rechtecksbestimmung
   BlitterObject   (  Player[j].Frigate[i].PosX - ScreenLeft,
                      Player[j].Frigate[i].PosY - ScreenTop,
                      DDraw.lpDDSFrigate ,
                      &Player[j].Frigate[i].rcShip,
                      DDBLTFAST_SRCCOLORKEY );

   // Blitterung der Fregatten für alle Spieler. ScreenLeft/ScrollTop bedeutet die Scrollweite
   // des Monitors. Die andern Übergabewerte sind schon erklärt worden.
  }

[...]   // Alle weiteren Schiffe nach demselben Prinzip
```

```
}
```

3.5.3 Interface():

Das Spielerinterface setzt sich aus hunderten kleiner Frames zusammen, die später alle im Spiel interaktiv zu sehen sind. Wir sehen hier z.B. das Hauptmenü, die ganzen Knöpfe, Mauszeiger, Schatten und die spieleigene Schriftart.

Der Farbwert 255,0,255 (rosa) wird später im Spiel ignoriert (siehe ColorKey bei Blitting).

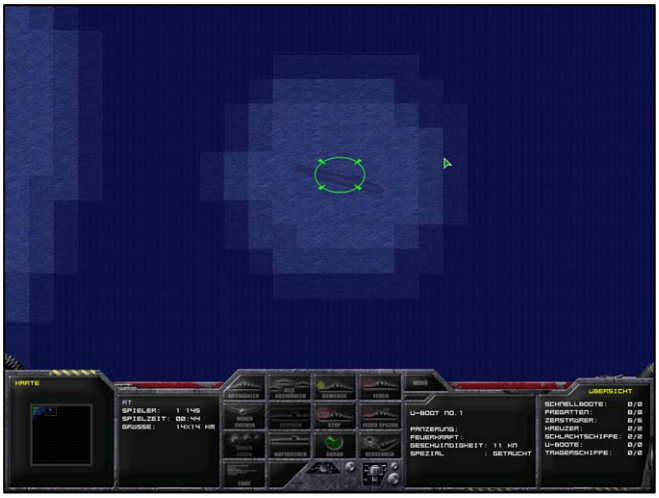
Alle Berechnungen im Detail zu erklären ist vermutlich wenig sinnvoll. Im Quellcode sieht man zwar dass sie „ziemlich“ lang ist, jedoch auf demselben Prinzip beruht wie alle anderen Anzeige-Funktionen. Im Unterschied zu den Schiffen haben wir jedoch eine schnellere Blitting-Funktion verwendet, die ohne Clipping-Eigenschaften auskommt, da sich alle Bilder auf dem Bildschirm befinden und nicht den Rand schneiden. Dies beschleunigt den Rechenprozess.

Die Interface-Funktion hat aber auch noch weitere Aufgaben: Über sie werden die Aufrufe der Button-Funktionen gesteuert (Angriff, Bewegen, Beenden, usw.). Darüber hinaus erkennen wir die vier Status-Monitore, einige Effekte, die Maps, die eigens entworfene Schriftart, etc. Die Interface-Funktion ergibt äußerlich ein kompliziertes Geflecht an Bedingungen und Ereignissen, ist jedoch eine der einfachsten Funktionen.

Original-Bild des Interfaces

3.5.4 Fog(), MapFog() und MapMonitor():

Der Nebel wird durch ein zweidimensionales Array berechnet, dass 15 x 15 Pixel große unterschiedlich helle Kästchen abgestuft wird. Durch eine radiale Berechnung aller Schiffe und ihrer Sichtweitendistanz über den Pythagoras und der Objektorientierten Programmierung von Schiffen und deren Sichtweitenwerte können unterschiedliche Sichtweiten auf der Map entstehen. Wir müssen hierzu noch die Reihenfolge der Nebel – Ebene beachten, die Auflösung der Kästchen und darüber hinaus in dem MapMonitor unten rechts auch noch einen ähnlichen Schatten kreieren. Diese Berechnung geschieht natürlich auf ähnliche Weise. Hier wird eine simulierte Schiffsreihe erstellt, deren Berechnungsangaben exakt, aber im kleineren Maßstab den originalen entsprechen und als Punkte geblittert werden. Beim U-Boot existiert die Besonderheit, dass es einen kleineren Sichtradius besitzt, wenn es abgetaucht ist (siehe Bild).



Screenshot aus Naval Forces – ein getauchtes U-Boot, umgeben von Schatten

3.5.4 Chat():

Der Chat blittert die gegebenen Zeichenketten auf den Monitor. Programmiert haben wir dazu die Funktion EnterText(), falls jemand etwas was eingeben will (aufgerufen durch Enter), die Funktion PrintText um Zeichenketten (Datentyp Char) zu blittern. Darüber hinaus existiert eine Chat-Strukturvariable, die für jede Chatlinien bestimmte Eigenschaften besitzt (z.B. verschwinden nach einer bestimmten Zeit die angezeigten Linien).

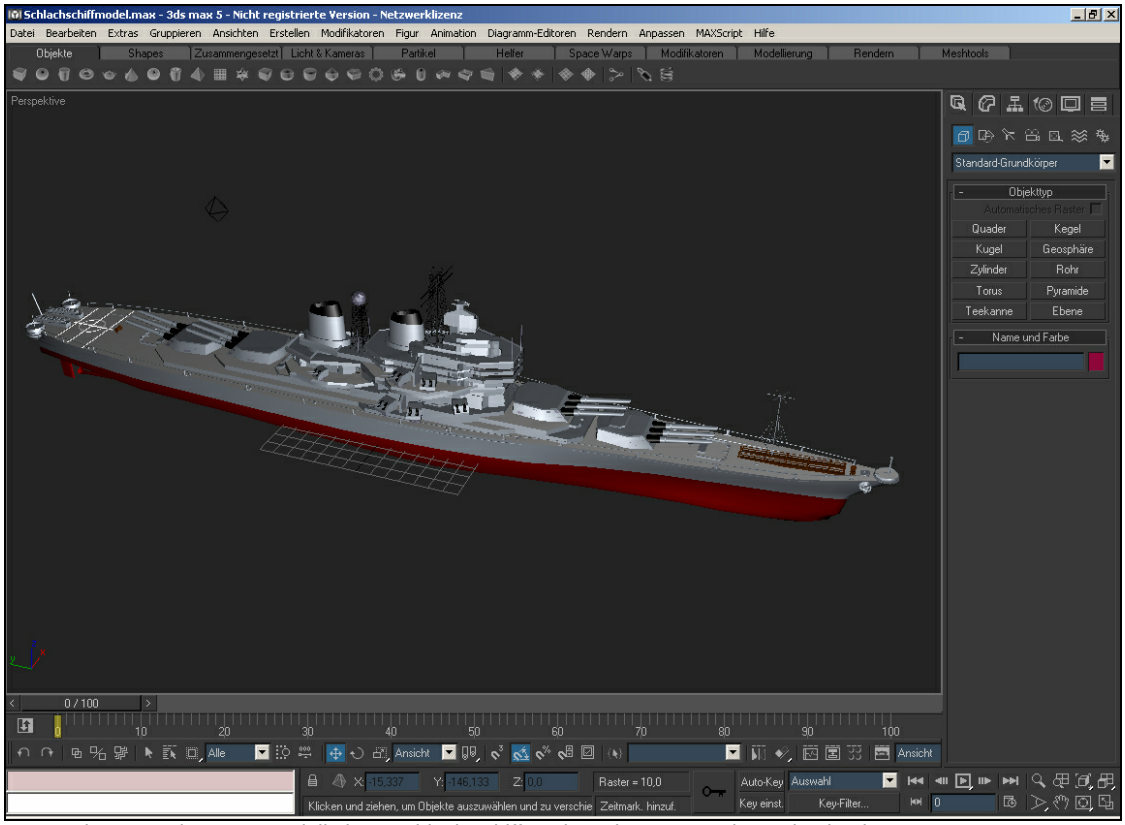
3.5.4 SendAndRecieve():

Die Senden- und Empfangen-Funktion dient (wie könnte es anders sein) zum Senden- und Empfangen von Daten. Im ersten Teil der Funktion werden alle relevanten Schiffsdaten in das zu sendende Objekt kopiert und schließlich auf einen Schlag versendet. Im Schlussteil der Funktion werden alle erhaltenen Daten auf die Schiffe übertragen zu denen sie gehören. Über die PlayerID erhalten schließlich alle Schiffe die erforderlichen Daten.

Grafik & Design

4.1 3D - Bildbearbeitung

Bevor wir ein Schiff im Spiel als Grafik anzeigen konnten, mussten wir mithilfe von „3ds max“ ein dreidimensionales Abbild jedes Schiffes erstellen. So erhielt jedes Schiff ein anderes Aussehen. Diese so genannten 3D Modelle oder Meshes, sind nicht wie von vielen behauptet, aus dem Internet oder irgendwoher kopiert, sondern alle einzeln erstellt und modelliert. Als Referenz diente hier die Homepage des U.S. Navy und Blauzeichnungen von Modellbauern.

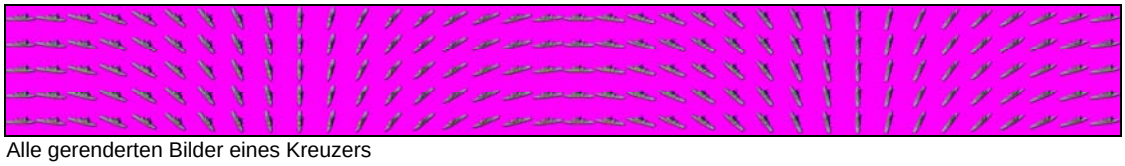


Screenshot aus 3ds max – Modell eines Schlachtschiffes, dass der U.S.S. Wisconsin ähnelt

Da DirectDraw nicht mit realen 3D – Dateien arbeiten kann, sondern nur mit Bitmaps, mussten die Grafiken gerendert (rendern = von 3d und 2d wiedergeben) werden. Dazu mussten wir eine einheitliche Ansicht und Beleuchtung für alle Schiffe verwenden. Eine Kamera renderte dann im 10° Abstand 36 Mal das Model. Zudem mussten zusätzliche Farbflächen eingeführt werden, da es mindestens vier verschiedene Spieler geben sollte. Zusätzlich fügten wir jedoch noch einen fünften Spieler hinzu. Schließlich wurden die 5 x 36 Bilder gerendert und in einer regelmäßigen Anordnung als Bitmap gespeichert.

Detaillierte Ansicht eines Kreuzers

Die gesammelten Bitmaps wurden schließlich alle aneinandergereiht und in 256 Farben gespeichert, was die Dateigröße erheblich verminderte. (aus grafiktechnischen Gründen verwendeten wir keine JPGs zur Darstellung, da wir exakte Color-Key-Werte benötigten, die bei der Komprimierung zu JPG verloren gehen).



Alle gerenderten Bilder eines Kreuzers

Da DirectDraw einen bestimmten Farbwert transparent machen kann, verwendeten wir wie schon erwähnt den RGB – Farbwert 255,0,255 (rosa), da dieser keinerlei Anteil in einem Spiel dieser Art enthielt. Nun konnten wir mit dieser Bitmap im Programm ein drehendes, flüssiges Bild eines Schiffes darstellen. Dass auf einem blauen Untergrund fahren konnte.

4.1 Design

Für die 2D Bildbearbeitung verwendeten wir Corel Photo-Paint. Dort steuerten wir mithilfe von Ebenen und Objekten zahlreiche grafische Effekte innerhalb der Menüs. Da wir während der Entwicklung zahlreiche Änderungen an der Funktionalität unseres Projektes vornahmen, änderte sich auch das Gesicht der Menüs. Zu beachten war dabei die Größe der Auflösung (Höhe 600, Breite 800), damit alle Grafiken korrekt innerhalb des Bildschirms angezeigt wurden. Zur großen Schwierigkeit wurde es dem Spiel ein einheitliches Gesicht und seinen eigenen Stil zu geben.

Screenshot aus Corel Photo-Paint – Bearbeitbares Hauptmenü von “Naval Forces”

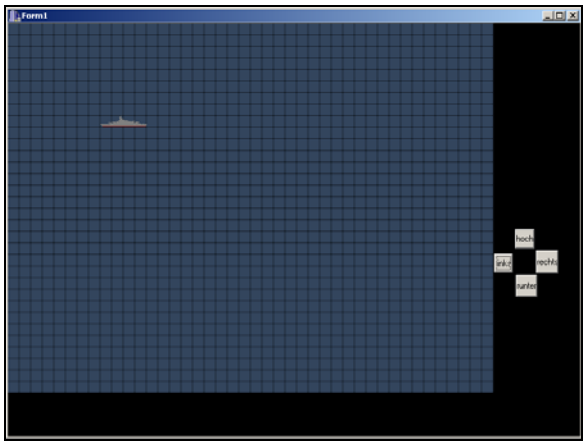
Einzelne Bilder wie die Buttons oder wenn die Maus darüber fährt wurden separat als Dateien gespeichert und als Images eingefügt.

Die Entwicklung

Information

Die Entwicklung von Naval Forces hat zahllose Phasen und Versionen durchlebt. Da die die anfängliche Entwicklung das Aussehen des Spiels bis heute geprägt hatte, sind wir der Meinung, dass die Entwicklung des Spiels sehr viel über „Naval Forces“ aussagt. Wir beginnen mit der Entwicklung der Grafiken und von DirectDraw, gehen über die Entwicklung von DirectPlay bis zu den ersten Versuchen die Programme zu verbinden. Die Ideen und Vorstellungen über unser Spiel waren zunächst völlig unterschiedlich. Sie auf einen gemeinsamen Nenner zu bringen hat uns die gesamte Entwicklung hindurch begleitet und war auch das aufregendste an der Programmierung. Unsere erste Spielvorstellung war etwas wie das Linux-Spiel „TRON“ für Windows zu programmieren. Später wird aus dieser Idee die des „Schiffe-Versenken“-Prinzips. Zu Beginn erstellten wir Pläne zur Realisierung. Schon seit damals war Microsofts DirectX Software Developer Kit ein möglicher Kandidat für die Programmierung. Später entschieden wir uns - aufgrund unserer wenigen Erfahrungen - für die Programmierung mit Borland C++. Heute wissen wir, dass die Programmierung mit Microsoft Visual C++ um einiges verbessert worden wäre. Das Projekt wurde in zwei Bereiche gespalten: Die Programmierung mit DirectDraw für Valentin Schwind und DirectPlay für Martin Goihl. Die Zusammenführung sollte Andre Grumbach leiten.

5.1 DirectDraw



24. Oktober 2002

Angefangen hat alles mit einer einfachen Version, wo wir mit den Canvas-Funktionen von C++ ein Schiffchen innerhalb eines Rastersystems (erstellt mit einem 2 dimensionalen Array).

15. November 2002

Später begannen wir mit einem Menü, in welchem wir die ersten DirectX Funktionen testeten.

25. November 2002

Daraufhin folgte das erste Schiff, dass mit DirectDraw und einigen Funktionen tatsächlich zu bewegen war.

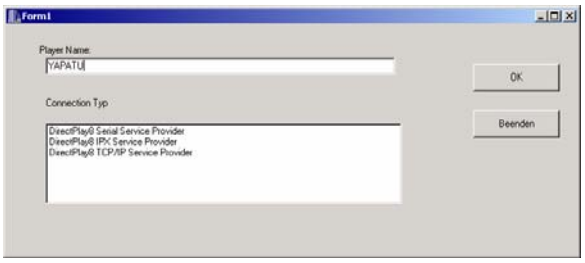
4. Januar 2002

Schließlich folge ein provisorisches Menü, ein neues Schiff, Benutzergesteuerte Abläufe (Bewegen, Markieren, etc.). Objektorientierte Programmierung, eigene Clipping & Blitting-Funktionen uvm.

2. Februar 2003

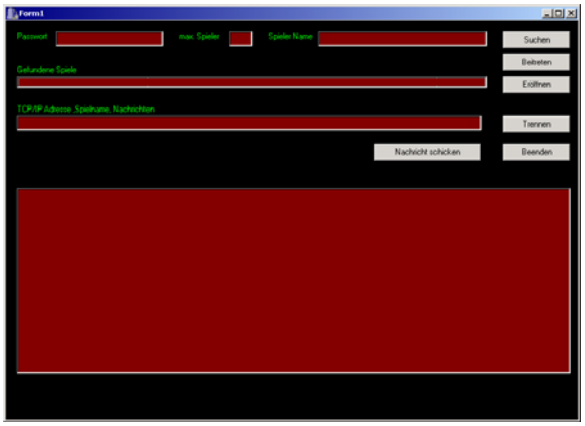
Heute ist unser Spiel etwas ausgereifter und hat sein Gesicht seit Februar 2003 nicht groß verändert. Es wurden neue Schiffe, die Karten, Wassereffekte, neue Menüs, ein neues Interface und zahlreiche Methoden optimiert, erweitert und verbessert. Wir nutzen jetzt die DirectPlay Funktionen effizienter und verwenden einige Optionen die DirectPlay anbietet um den Spielern den best möglichen Komfort zu bieten.

5.2 DirectPlay



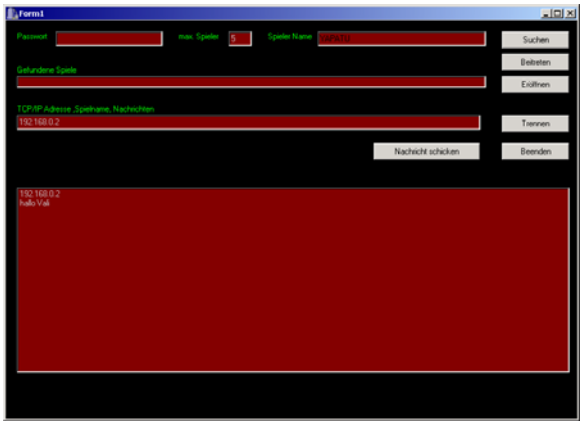
24. Oktober 2002

Am Anfang haben wir versucht durch Quellen aus dem Internet und Beispielprogrammen den Grundaufbau von DirectPlay zu verstehen, dazu haben wir ein kleines Testprogramm geschrieben in dem wir einige so genannte Enumerations (siehe Bild) getestet haben.



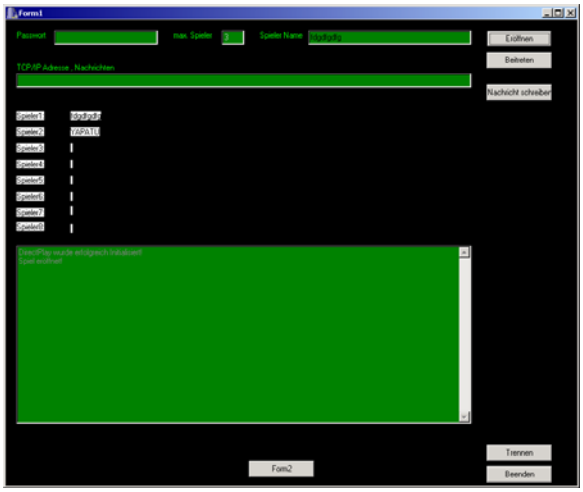
15. November 2002

Das Testprogramm wurde erweitert durch die Funktionen, Eröffnen, Beitreten, Suchen.
Dabei entstanden die ersten Datentypen-konvertierungsprobleme (wie unten beschrieben).



7. Dezember 2002

Inzwischen wurden die Funktion Senden in das Testprogramm implementiert und am 7.12. war es dann soweit, dass die ersten Chatzeilen von Sindelfingen nach Esslingen über das Internet übertragen werden konnten.
Ebenfalls konnte man seine Sitzung jetzt durch ein Passwort schützen.



1. März 2003

Der Quellcode des Testprogramms wurde völlig überarbeitet und dadurch um die Hälfte verkürzt.
Wir beschlossen unnötige Enumerations zu löschen und unser Programm nur auf das Protokolle TCP/IP aufzubauen, das fürs Internet genauso genutzt werden kann wie fürs Netzwerk und im Moment das gängigste Protokoll ist.
Außerdem wurde die Option hinzugefügt, dass alle Spieler angezeigt werden.

14. April 2003

Wir haben versucht durch Shapes fahrende Schiffe in unserem Testprogramm zu simulieren, wobei wir viele Erkenntnisse gewannen, die wir später beim Verbinden der beiden Quellcodes gut gebrauchen konnten.

5.3 Verbinden der Quellcodes

2. Januar 2003 bis 5. Januar 2003

Unser erstes Treffen ging 4 Tage und wir haben vor allem unseren Quellcode optimiert, z.B. durch das dynamische Erstellen aller Schiffe.
Es war auch das erste Mal, dass wir zusammen an unserem Quellcode arbeiten konnten, was ein großer Vorteil war. Auf Grund dieser positiven Erfahrung haben wir schon damals ausgemacht, dass wir uns auf jeden Fall noch mal mehrere Tage treffen werden.

6. Juni 2003 bis 14. Juni 2003

Bei unserem zweiten eineinhalbwöchigen Treffen, haben wir die beiden bis jetzt völlig getrennten Quellcodes in einem Projekt verbunden, dabei den Chat im Spiel vervollständigt, einen Bereitbutton entwickelt, die Spieleinstellungen eingebunden und das gesamte Menü entwickelt.
Außerdem haben wir alle Verbindungs- und Beendenprobleme, sowie weitere zahllose „Bugs“ und Probleme behoben.
Das Highlight war jedoch die Zusammenführung von DirectDraw, Direct Play und der objektorientierten Programmierung. Dadurch konnten wir alle Bewegungen der Spielereinheiten an den teilnehmenden Computern darstellen.

5.4 Probleme

Während unserer Programmierung gab es wohl keinen Bereich und keine Situation in der keine Probleme aufgetreten sind. Einige unserer größten Hindernisse war wohl der fehlende Debugmodus im Borland Editor während der DDraw - Laufzeit, der „Bresenham-Algorithmus“ und die Zusammenführung von DirectDraw und DirectPlay. Hier sind oft mehrere Wochen verstrichen, bevor wir auf die Lösung der Probleme kamen.

Große Probleme gab es zu Beginn mit den Klassen, später mit der dynamischen Erzeugung von Spielern und Schiffen, Schiffsalgorithmen, der Einbindung in Borland, Datentypenkonvertierungen, das Verständnis für DirectPlay und DirectDraw.

Für die größten Probleme gab es meist die einfachsten Lösungen, während manch andere Probleme nie gelöst wurden. Wir haben eine kleine Liste von den größten Problemen und deren Lösung erstellt:

Probleme	Lösungen

Die Einbindung in Borland Builder 5 C++

Die größten Schwierigkeiten entstanden durch den relativ günstigen Editor. Durch die schwierige Einbindung von DirectX in den Quelltext mussten wir auf die komplizierte „Includierungen“ und deren Folgen beachten, sowie auf die rudimentäre Projektverwaltung von Borland.

Wie schon erwähnt sind wir heute der Ansicht, dass Microsofts Visual C++ uns durch seine einfache Funktions- und Klassenübersicht weitergebracht hätte. Da wir jedoch seit Beginn und während unserer Schulzeit mit Borland gearbeitet hatten, entschieden wir uns das Projekt nicht auf Visual C++ zu übertragen.

Die Verbindung zwischen DirectDraw und DirectPlay

Durch unsere getrennte Arbeit, die durch die Summe kleinerer Schwierigkeiten aufgehalten wurde, konnten wir erst sehr spät mit der

Heute würden wir wohl kaum anders vorgehen, jedoch lässt sich mit Sicherheit sagen, dass Strukturgramme, UML - Klassendiagramme oder gar ein Extra - Projektmanager den

Verbindung beider Quelltexte beginnen. Durch den Zeitdruck und mehrerer Komplikationen vor allem durch die oben erwähnte „Includierung“, sowie die primitive Klassenverwaltung in Borland kamen wir nur sehr schwer voran.

Entwicklungsprozess bei DirectX mehr als behindert hätten. Einzig und allein unsere Erfahrung und unsere Kenntnisse mit dem Umgang des Quellcodes von DirectDraw und DirectPlay haben uns schließlich vorangebracht!

Der „Bresenham“ - Algorithmus

Der schon oben erwähnte Algorithmus zur Berechnung des kürzesten Weges innerhalb eines Rastersystems hat uns ein weiteres Problem beschert.

Eine Doktorarbeit eines Studenten im Internet hat uns auf die Lösung des Problems geführt. Jedoch sei zu erwähnen, dass wir nur sehr kurz vor der selbstständigen Lösung des ganzen Problems standen.

Die dynamische Erzeugung von Schiffen und Spielern

Durch simple Funktionen und Struktur-Variablen kamen wir sehr schnell an die Grenzen unseres Programms, denn eine nachträgliche Fehlerbehebung oder eine dynamische Erzeugung von Einheiten und Spielern erwies sich so als nahezu unmöglich.

Im Laufe der Programmierung kamen wir auf Klassen und Objekte und nach unserem ersten Treffen hatten wir dynamisch Objekte erzeugt. Später fügten wir auch dynamisch Spieler hinzu.

Datentypkonvertierungen

Einige weitere Probleme entstanden durch Datentypenkonvertierungen.

Da manche Datentypen sich nicht direkt in andere kopieren ließen (z.B. WCHAR in CHAR) mussten wir erst auf Zwischenspeichervariablen zurückgreifen. (Später entdeckten wir eine MultiByteToWideChar(…) – Funktion, die das Problem komplett löste.)

Wenig Referenzhilfen

Nur einige wenigen Foren im Internet, die Microsofthilfe und einige Kontakte im Internet halfen uns mit unseren zahllosen Problemen und Fragen zu DirectDraw und DirectPlay oder zu Algorithmen.

www.gamedev.net, www.zfx.info und www.msdn.com waren eine große Hilfe, ansonsten fanden wir nur sehr wenig Ansprechpartner zu unseren Fragen.

Hardwareeinschränkungen

Zur großen Kunst wurde es im Laufe der Programmierung ein Spiel zu entwickeln, welches entweder auf dem neuesten Stand der Technik ist oder auch auf älteren Rechnern funktioniert, was enorme Einschränkungen in den abzuarbeitenden Funktionen bedeutete.

Eine große Hilfe war das Betriebssystem Windows 2000, was relativ stabil lief und weniger Abstürze als z.B. Windows 98 während der Programmierung verursachte.

Der fehlende Debugmodus während DirectDraw

Eins der oder vermutlich sogar das größte Problem während der Programmierung war der fehlende Debugmodus während der Laufzeit von DirectDraw. Der Borland Editor setzte zwar Haltpunkte, ließ aber nicht zu die Anwendung mit Alt-TAB zu wechseln, was bedeutete, dass man das Programm beenden musste.

Heute wissen wir, dass wir mit Microsoft Visual C++ dieses Problem hätten beheben können. So mussten wir eine PrintText-Funktion entwickeln, die uns während der DirectDraw - Anwendung alle Werte über den Bildschirm ausgab.

Die Hinderniserkennung

Das größte und nahezu unlösbarste Problem im Laufe der Programmierung wurde die Hinderniserkennung. Da wir das Problem der Hinderniserkennung nicht in Bezug auf den kompletten Spielaufbau setzen, erkannten wir zu spät die Unmöglichkeit in der späteren Einbindung in unseren Quellcode.

Der so genannte A-Star-Algorithmus ist die Lösung des Problems, dass wir zu spät erkannten. Da dieser jedoch auf einem zusätzlichen Rastersystemorientierten Aufbau basiert, dass das gesamte Spielprinzip komplett verändert hätte und auch unsere bisherige Bewegungssteuerung außer Kraft setzen würde, ließen wir die Hinderniserkennung außen vor.

Der Angriff

Die Zielübergabe wurde zu einem weiteren Problem, da wir einen gesendeten Datentyp, der einen empfangen Datentyp enthielt nicht noch mal versenden konnten.

Eine Lösung für dieses Problem wäre eine Filterung aller Daten. Durch die mangelnde Programmierzeit ließ sich die jedoch nicht mehr realisieren.

5.5 Kosten

Da das meiste an Bedarf schon vor dem Projekt vorhanden war, gibt es keine Werte, die hier besonders ernst genommen werden sollten. Dieses Kapitel dient also als Platzhalter für Werte, die vielleicht irgendwann entstehen mögen:

Software für V. Schwind:

Borland® Builder 5™ C++ für Windows™	Professionelle Programmierung	160,00 €
Corel Draw 11 für Windows™	Grafiken, Schriften, Effekte, und Animationsbearbeitung	399,00 €
Microsoft® Windows 2000™ Professional	Benötigtes Betriebssystem	179,00 €
Microsoft® Office XP für Windows™	Benötigte Software für Dokumentation, Berichte, Hefte, E-Mails, Präsentation und Internet Publikation des Programms.	699,00 €
		1437,00 €

Software für M. Goihl:

Borland® Builder 5™ C++	Professionelle Programmierung	160,00 €

für Windows™		
Microsoft® Windows 2000™ Professional	Benötigtes Betriebssystem	179,00 €
Microsoft® Office XP für Windows™	Benötigte	699,00 €
		1437,00 €

Software für A. Grumbach:

Borland® Builder 5™ C++ für Windows™	Professionelle Programmierung	160,00 €
Microsoft® Windows 2000™ Professional	Benötigtes Betriebssystem	179,00 €
Microsoft® Office XP für Windows™	Benötigte	699,00 €
		1437,00 €

Quelle: www.alternate.de

Softwarepreise: schon bezahlt

Weitere Software, die für die Erstellung notwendig sein wird:

Weitere Software/Internet

Internet (WAN)	Zum Datenaustausch und für die Internetfähigkeit des Programms	0,00 €
Microsoft Direct X SDK (Libraries)	Archiv für die Programmierung mit Direct X	0,00 €
ZoneAlarm (Firewall)	Schutz der Rechner	0,00 €
ICQ (Chatprogramm)	Zum Absprechen im Internet	0,00 €
Bullet FTP/Orgdns (FTP Server)	Datenaustausch zwischen den beiden Rechnern	0,00 €
WinAmp	Ich rühr' keinen Finger ohne Musik!!!	0,00 €
		0,00 €

Sonstiges

Information

„Naval Forces“ erwies sich als eine der schwierigsten Aufgaben und Herausforderungen, an denen wir je gearbeitet hatten. Der jetzige Status des Spiels ist gut zu beschreiben mit „fast fertig und man kann es immer erweitern“. Mögliche Optimierungen und Erweiterungen zum Spiel, sowie unser Fazit zum Spiel werden wir zum Schluss noch beschreiben.

6.1 Zusätzliche Optimierungen

Die wichtigste Eigenschaft, die dem Spiel fehlt, ist der Angriff auf andere Schiffe. Durch erhebliche Probleme mit der Einbindung von DirectPlay und der Versendung der Schiffsdaten konnten wir den Angriff bis zur Projektabgabe nicht fertig stellen. Daher ist es Ziel den Angriff den Angriff noch in das Spiel einzubinden.

Eine komplette Neuerung wäre die Einbindung des schon erwähnten A-Star-Algorithmus, der auf einem zweidimensionalen Rastersystem basiert, das gleichzeitig mit einer Wegfindung und zusätzlichen Höhenebenen arbeitet. Eine nachträgliche Einbindung erweist sich als nicht machbar, würde der Quellcode jedoch neu aufgesetzt wäre der A-Star-Algorithmus eine äußerst aufwendige, aber dennoch großartige Lösung vieler Probleme und Features unseres Spiels.

Eine Neuentwicklung des Codes würde ebenfalls die Einbindung einer kompletten 3D Engine bedeuten. Da Direct3D wie DirectDraw arbeitet, aber zusätzlich noch Höhenkoordinaten einbindet, ließe sich mit Direct3D ein komplett dreidimensionales Spiel programmieren.

6.2 Verwendungen

Gewaltfreie Unterhaltungsspiele am PC sind gefragter denn je. „Naval Forces“ ist ein ideales Spiel für mehrere Spieler gegeneinander auf hoher See anzutreten. Hier kann jeder sein strategisches Können auf die Probe stellen. Da unser Spiel auf dem altbewährten „Schiff-Versenken“ aufbaut, was in einer breiten Palette von Spielarten (Brettspiel, Computer, Konsole usw.) schon vermarktet wurde. Naval Forces“ kommt mit neuen Features und einer neuen Grafik.

Die Wirtschaftlichkeit unseres Projekts zeigt sich allein darin, dass es umsonst ist. Es entstehen keinerlei Kosten und Zahlungen.

So bleibt nur noch die Frage was das Programm an Geld erwirtschaften könnte. Bei Fertigstellung des Spiels (in ca. 6 – 12 Monaten) kann es bis zu ca. 20,00 € pro Stück verkauft werden. Bei allein einer Auflage von 10.000 Stück kann dies einen Ertrag von mehr als 200.000 € bedeuten (abzüglich Steuern usw.).

Zwar lässt sich „Naval Forces“ noch nicht verkaufen, doch bei einer weiteren Erweiterung der oben genannten Algorithmen wäre es denkbar das Spiel zu vermarkten. Ansonsten lässt es sich prima als Freeware – Netzwerkspiel verwenden.

6.3 Fazit

Unser Fazit zu „Naval Forces“ ist gespalten. Zum einen sind wir sehr stolz in der kurzen Zeit etwas Derartiges auf die Beine gestellt zu haben, allerdings war einzig und allein unsere mangelnde Erfahrung in der Spieleprogrammierung die Ursache die Hinderniserkennung nicht rechtzeitig implementiert zu haben. Die Einbindung der A-Star-Algorithmus von Anfang an, hätte unsere Programmierung komplettiert und im Endeffekt auch sehr vereinfacht. Darüber hinaus sind wir enttäuscht von der mangelnden Funktionalität des Borland Builders und von den wenigen Ressourcen im Internet. Dennoch ist „Naval Forces“ etwas sehr besonderes für uns, da wir unglaublichen Spaß an dessen Entwicklung hatten und unglaublich viel Erfahrung gesammelt haben. Die erfreuliche Resonanz bei unseren Mitschülern spornten uns schließlich noch mehr an und wir kamen teilweise weiter, als wir es uns anfangs erträumt hatten…

Merkblatt