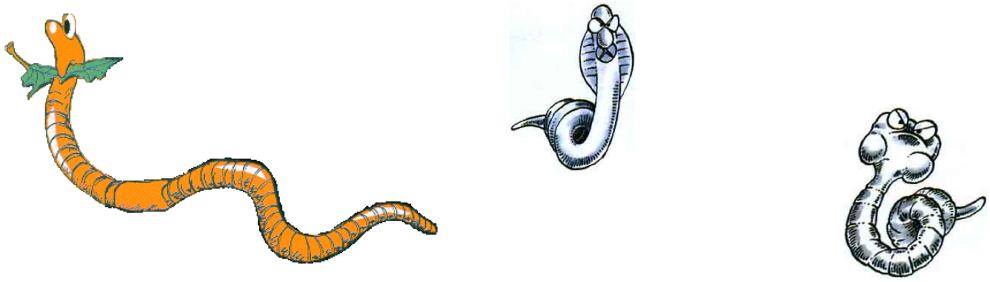


Übungsprojekt 'Wurm'

Zur Vertiefung des Kapitels SW-Engineering findet im 2. Semester des Fachs Informatik eine Projektarbeit statt. Dabei soll ein (Netzwerkfähiges) Spiel innerhalb kleiner Projektteams (3-4 Personen) entworfen und implementiert werden. Die ganze Aufgabe ist entsprechend der Regeln des SW-Engineering zu lösen und zu Dokumentieren.

Dateiname:	Projekt_Wurm_2012_v2.3.odt
Erstellt am:	9. Februar 2009
Letzte Änderung am:	17. Februar 2012
Erstellt von:	osi1
Version:	38

Projektaufgabe Informatik 2. Semester:

Aufgabe:

Sie sollen in einem Team ein Spiel (Oder ein gleichwertiges Softwareprojekt) realisieren. Sie können das vorgegebene Spiel (Standard Aufgabe 'Wurmspiel', siehe übernächste Seite) oder eine eigene Aufgabe realisieren. Wenn Sie eine eigene Aufgabenstellung wählen, sprechen Sie diese mit dem Dozenten ab, damit der Arbeitsumfang ungefähr den Anforderungen und den Möglichkeiten entspricht. Es ist Ihnen freigestellt, ob Sie Graphik und Netzwerk benutzen.

Ausgangslage:

Sie erhalten eine Bibliothek (QT-Basiert, läuft auch unter Linux und Mac OsX) für die Graphikdarstellung und die Netzwerkkommunikation zur Verfügung gestellt. Sie finden diese Bibliothek sowie die benötigten QT-Bibliotheken in einem Eclipse-Projekt auf Moodle und dem Schulnetzwerk unter
\\Blender\data\fbe\public\Labor\Info\Informatik_2\data\Projektarbeit\BasisProjektInfo2_2012.zip.

In diesem Projekt ist ebenfalls ein kleines Demoprogramm als Starthilfe enthalten. Sie können diese Projekt als Basis für Ihre Aufgabe verwenden

Wichtiger Hinweis: Wenn Sie ein neues Projekt erstellen, müssen Sie C++ und nicht C als Sprache wählen, und unter 'Projekt /Properties' unter dem Tab 'C/C++ Build/Settings/MinGW C++ Linker/Libraries' im Fenster 'Libraries (-l)' die vier Bibliotheken **'SimpleGUILib'**, **'QtGui4'**, **'QtNetwork4'** und **'QtCore4'** **in genau dieser Reihenfolge** hinzufügen. (Namen ohne Hochkomma eingeben, pro Zeile eine Bibliothek). Im gleichen Fenster unter 'Library Search Path' noch den Pfad zu diesen Bibliotheken angeben. Zudem müssen die DLLs QtCore4.dll, QtGui4.dll, QtNetwork4.dll, libgcc_s_dw2-1.dll, libstdc++-6.dll und mingwm10.dll im Suchpfad oder im Verzeichnis der ausführbaren Datei abgelegt sein.

Bewertung:

Die Arbeit wird benotet, die Note fliesst zu 1/4 in die Semesternote ein. Jedes Teammitglied muss ausweisen (Kurzvortrag/Bericht), was es zum Projekt beigetragen hat, damit die Notengebung fair und korrekt erfolgen kann. Bewertet wird die Arbeitsweise (SW-Engineering), die Dokumentation, die Funktionalität und der Code (Übersichtlichkeit, Verständlichkeit, Struktur, Kommentar).

Gruppenaufteilung:

Jede Gruppe setzt sich aus 3-**4** Mitgliedern zusammen. Es ist auf eine gute Durchmischung zu achten, es soll nicht eine Gruppe nur mit Profis, und eine nur mit Programmiereinsteigern geben, sondern es sollte idealerweise in jeder Gruppe ein Mitglied mit Programmiererfahrung vorhanden sein.

Empfehlungen:

Definieren Sie Muss-, Soll- und Kannziele, so können Sie flexibel auf Zeitprobleme reagieren.

Entwerfen (Designen) Sie Ihr Projekt so, dass weglassen oder hinzufügen von optionalen Elementen (Soll/Kann) mit wenig Aufwand möglich ist.

Sorgen Sie mit einem Versionsmanagement dafür, dass Sie jederzeit frühere Projektstände wiederherstellen können.

Vorgehensweise:
Das ist eine Gruppenarbeit, das heisst dass alle Gruppenmitglieder zum Erfolg des Projektes beitragen müssen. Sie sollen die Aufgabe in einzelne Teilaufgaben/Module zerlegen, und jedes Mitglied soll eine dieser Teilaufgaben implementieren. Damit das klappt, müssen die Schnittstellen zwischen den Modulen, und die Aufgaben der einzelnen Module klar definiert werden. Es soll in jeder Gruppe einen Projektkoordinator geben, der die Arbeiten koordiniert und für einen geordneten Projektablauf sorgt. Es ist aber nicht Aufgabe des Koordinators, alles selbst zu machen!. Unterteilen Sie das Projekt in einzelne Schritte, die Sie nacheinander implementieren und testen können, so dass Sie nicht erst am letzten Tag alles zum laufen bringen müssen, sondern bereits vorher einfachere lauffähige Varianten austesten können. Setzen Sie genug Zeit für Analyse und Design ein, bevor Sie mit programmieren beginnen.

Für die Arbeit ist projektmässig vorzugehen, d.h. dass ein Terminplan erstellt werden soll, sowie fortlaufende Statusberichte zu verfassen sind. An jedem Tag ist kurz niederzuschreiben, was erreicht worden ist, und was nicht erreicht worden ist. Der Terminplan ist zu Beginn des Projektes abzugeben. Sämtliche Entscheide und Beschlüssen sind zu Dokumentieren. Es sind mindestens 2 Meilensteine zu definieren, bei diesen findet zwingend eine Lagebesprechung mit der Geschäftsleitung (Dozent) statt. Am Schluss ist eine Projektdokumentation und der Code abzugeben. Von jedem Team stellt jedes Mitglied seine Arbeit kurz vor. (Kurzvortrag, 3 bis 5 Minuten). Die Präsentation findet am letzten Tag der Projektarbeit statt.

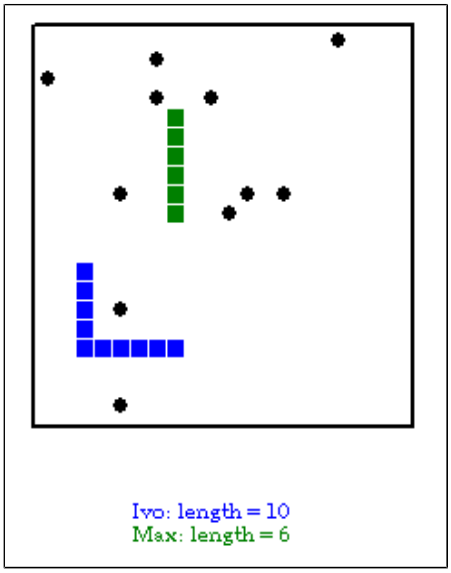
Zeitlicher Ablauf:
Sie haben im wesentlichen 8 Wochen Zeit für diese Aufgabe. Um zu einem guten Abschluss zu kommen, ist eine gute Planung unerlässlich.
Hier einige Empfehlungen:

- In der ersten Woche sollte die Aufgabestellung klar sein, und ein kurzes Pflichtenheft vorliegen, so dass allen Teammitgliedern klar ist, was eigentlich gemacht werden soll (Analyse). Zudem sollte ein grober Zeitplan erstellt sein, so dass ungefähr klar ist, was bis wann erledigt sein sollte.
- Anschliessend sollte ein Grobdesign erstellt werden, so dass einzelne Module und Ihre Aufgaben definiert sind. Daraus lassen sich schliesslich die Schnittstellen (Funktionen, Datenstrukturen, Funktionsparameter) definieren. Sobald die Schnittstellen definiert sind, können die Arbeiten auf die einzelnen Teammitglieder verteilt werden, so dass jedes Mitglied für ein bis zwei Module verantwortlich ist.
- Nun kann das Detaildesign der verschiedenen Module und deren Implementation durch die jeweiligen Verantwortlichen erledigt werden. Auch ein Einzeltest der Module ist zu empfehlen.
- Sobald Module Teilfunktionalität aufweisen, können sie für erste Tests bereits in das Gesamtprojekt integriert werden. Das vermeidet unangenehme Überraschungen am Ende.
- Am Schluss werden die einzelnen Module zum Gesamtsystem zusammengefügt, und das Gesamtsystem getestet.

Obwohl Sie das Projekt erst in der zweiten Woche der UFZ im Sommer (KW23) abgeben müssen, sollten Sie das Projekt nach 8 Wochen im wesentlichen Abgeschlossen haben, und die restliche Zeit zum Fertigstellen der Dokumentation sowie allenfalls noch zur Behebung kleinerer Mängel einsetzen.

Standard Aufgabe:

Ihr Team hat den Auftrag, das Wurm-Spiel zu implementieren. Bei diesem Spiel wird ein Wurm mittels der Cursortasten über ein Spielfeld gesteuert. Das Spiel ist für den Spieler zu Ende, sobald sein Wurm mit dem Rand, einen anderen Wurm oder sich selbst zusammenstösst. Innerhalb des Spielfeldes werden periodisch kleine Punkte verteilt, die vom Wurm gefressen werden können. Wenn ein Punkt nach einer gewissen Zeit nicht gefressen wurde, soll er wieder verschwinden. Die Futterpunkte sollten nicht mehr als etwa 2-3% der Spielfläche bedecken. Mit jedem gefressenen Punkt wird der Wurm um ein Feld länger. Der Wurm wird jeweils nach 10 Schritten automatisch um ein Feld länger. Zu Beginn hat der Wurm eine Länge von 1. Der Wurm bewegt sich alle 200ms um einen Schritt weiter, normalerweise in dieselbe Richtung wie beim vorhergehenden Schritt, ausser der Spieler hat eine neue Richtung vorgegeben. Sie können zur Steigerung der Schwierigkeit die Zeitschritte nach einer bestimmten Spielzeit verkleinern.



Jeder Wurm soll sich durch seine Farbe von den anderen unterscheiden. Das Spiel soll für mindestens 2 Spieler ausgelegt werden. Wenn via Netzwerk gespielt wird, sollten auch mehr als 2 Spieler mitspielen können. Von jedem Spieler soll der Name abgefragt werden, und in der Farbe seines Wurms unterhalb des Spielfeldes dargestellt werden. Neben dem Namen soll die aktuelle Länge des Wurms erscheinen. Es sollen zwei High-Score Listen geführt werden, eine für Einzelspieler, dabei wird die erreichte Länge des Wurms gewertet, und eine Mehrspielerliste, hier können sie selbst entscheiden was gewertet wird (Anzahl Siege (Letzter übriggebliebener Spieler, Länge des Wurms sobald man einziger Spieler ist, Verhältnis Siege zu Verlusten)) Die HighScore-Listen sollen dauerhaft in einer Datei abgespeichert werden und bei jedem Spiel aktualisiert werden.

Das Spiel wird über die Tastatur gesteuert: Der erste Spieler spielt mit den Cursortasten, der zweite mit den Funktionstasten:

Cursur Up/F2 -> der Wurm geht geradeaus weiter.
Cursor Left/F1 der Wurm dreht sich nach links,
Cursor Right/ F3 der Wurm dreht sich nach rechts.

(Wobei links und rechts immer von der Laufrichtung des Wurms aus gesehen sind.). Mit Escape kann das Spiel vorzeitig abgebrochen werden. (Sie dürfen selbstverständlich auch eine eigene Steuerungsphilosophie definieren).

Das Spielfeld ist in ein Netz aus quadratischen Feldern unterteilt, Wurmsegmente und Futter bewegen sich nur von Feld zu Feld und können sich nicht zwischen den Feldern aufhalten.

Netzwerkfähigkeit: Das Spiel soll auch über ein Netzwerk gespielt werden können. Dazu soll ein Computer als Server dienen, das Spiel läuft auf diesem Computer, die anderen Teilnehmer agieren als Client. Die Clients senden nur die Tastendrücke an den Server und empfangen Zeichenbefehle von Server, so dass sich auf ihrem Bildschirm dasselbe Abbild zeigt wie auf dem Server. Ein Client muss sich zuerst beim Server anmelden, bevor er an einem Spiel teilnehmen kann, und ein Server muss Anmeldungen akzeptieren können. Bei Netzwerkspielen sollen mindestens 2 Spieler teilnehmen können.

Zur Verfügung gestellte Module:

Auf den nachfolgenden Seiten werden die Ihnen zur Verfügung gestellten Module kurz beschrieben.

Das Modul Window "window.h"

Das Modul Window stellt Funktionen zur graphischen Ausgabe sowie zur Behandlung von Tastendrücken und Mausereignissen zur Verfügung.

Hinweis: Der letzte Zeichenbefehl eines Programms vor Programmende wird häufig nicht auf dem Bildschirm dargestellt. Das ist normal und kann durch Aufruf von WaitMs() behoben werden. Dieser Effekt tritt aber nur beim Programmende auf, sonst nicht.

Farben:

Bei den meisten Zeichenbefehlen kann eine Farbe mitgegeben werden. Diese Farbe wird mit der Struktur ColorType definiert.

```
typedef struct ColorType {
   char Alpha;
   char Red;
   char Green;
   char Blue;
} ColorType;
```

Die Struktur ColorType dient zur Aufnahme eines ARGB (Alpha Rot Grün Blau) Farbwertes. Die einzelnen Farbanteile müssen im Bereich von 0 bis 255 liegen. Wobei (255,0,0,0) Schwarz und (255,255,255,255) Weiss entspricht. Mit dem Alpha-Wert wird die Transparenz der Farbe definiert, ein Wert von 0 ist total durchsichtig, d.h Zeichnungen mit dieser Farbe bleiben unsichtbar, 255 ist ein voll deckende Farbe, und Werte dazwischen werden als teildurchsichtige Objekte gezeichnet.

Folgende Farben sind bereits vordefiniert:

COL_RED	COL_LIGHTRED	COL_DARKRED
COL_GREEN	COL_LIGHTGREEN	COL_DARKGREEN
COL_BLUE	COL_LIGHTBLUE	COL_DARKBLUE
COL_YELLOW	COL_LIGHTYELLOW	COL_DARKYELLOW
COL_BROWN		
COL_ORANGE		
COL_GREY	COL_WHITE	COL_BLACK

Diese vordefinierten Farben können direkt benutzt werden. Beispiel:

```
DrawPixel (10, 30, COL_ORANGE);
```

Das Modul Window stellt folgende Funktionen zur Verfügung:

Basisfunktionen (Reichen für normale Anwendungen):

void InitGraphic (int Width, int Height)

Öffnet ein Graphikfenster mit den angegebenen Abmessungen in Pixel. Muss aufgerufen werden bevor irgendeine der anderen Funktionen dieses Moduls verwendet werden kann. Wird für Width und Height ein negativer Wert angegeben, öffnet sich ein Bildschirm füllendes Fenster. (Grösse kann durch GetImageSize() mit `ID_WINDOW` bestimmt werden).

void CloseGraphic (void)

Schliesst das Graphikfenster und gibt alle Resourcen wieder frei. Muss vor dem Beenden des Programmes aufgerufen werden. Nach Aufruf dieser Funktion dürfen ausser InitGraphic keine anderen Funktionen dieses Moduls mehr verwendet werden.

void DrawFilledCircle(int x, int y, int Width, int Height, ColorType Color, int LineWidth)

Zeichnet einen gefüllten Kreis in das Graphikfenster, wobei x und y die linke untere Ecke des umschliessenden Rechteckes definieren, Width und Height die Dimension des umschliessenden Rechtecks festlegen und Color die Farbe des Kreises bestimmt. LineWidth definiert die Dicke der Kreislinie am Umfang.

void DrawEmptyCircle(int x, int y, int Width, int Height, ColorType Color, int LineWidth)

Zeichnet einen ungefüllten Kreis in das Graphikfenster, wobei x und y die linke untere Ecke des umschliessenden Rechteckes definieren, Width und Height die Dimension des umschliessenden Rechtecks festlegen und Color die Farbe des Kreises bestimmt. LineWidth definiert die Dicke der Kreislinie am Umfang.

void DrawFilledRectangle (int x, int y, int Width, int Height, ColorType Color, int LineWidth)

Zeichnet ein gefülltes Rechteck in das Graphikfenster, wobei x und y die linke untere Ecke des Rechteckes definieren, Width und Height die Dimension des Rechtecks festlegen und Color die Farbe des Rechtecks bestimmt. LineWidth definiert die Dicke der Umrisslinie.

void DrawEmptyRectangle (int x, int y, int Width, int Height, ColorType Color, int LineWidth)

Zeichnet ein ungefülltes Rechteck (Leeres Rechteck, nur Umrisslinien) in das Graphikfenster, wobei x und y die linke untere Ecke des Rechteckes definieren, Width und Height die Dimension des Rechtecks festlegen und Color die Rahmenfarbe des Rechtecks bestimmt. LineWidth definiert die Dicke der Umrisslinie.

void DrawLine (int x1, int y1, int x2, int y2, ColorType Color, int Width)

Zeichnet eine Linie von (x1, y1) nach (x2, y2) mit der Farbe Color und der Breite Width.

void DrawPixel (int x, int y, ColorType Color)

Zeichnet einen Bildpunkt mit der Farbe Color an die Position (X,Y).

ColorType GetPixel (int x, int y)

Liefert die Farbe des Bildpunktes an der Position (X,Y).

void DrawTextXY (int x, int y, ColorType Color, const char *Text)

Zeichet den Text 'Text' an der Koordinate (x, y) (Linke untere Ecke) mit der Farbe Color.

int GetKeyPress (void)

Liefert den nächsten noch nicht behandelten Tastendruck zurück. Wartet auf einen Tastendruck falls kein unbehandelter Tastendruck bereit ist. Die Funktion liefert den Code der Taste zurück. Bei gewöhnlichen Tasten wird der ASCII-Code zurückgeliefert, bei Spezialtasten werden Codes gemäss den definierten Konstanten des Enum KeyCodes geliefert.

Der Enum KeyCodes definiert Konstanten für Spezialtasten.

Es sind folgende Konstanten definiert:

Cursortasten:	`W_KEY_CURSOR_UP, W_KEY_CURSOR_DOWN, W_KEY_CURSOR_LEFT, W_KEY_CURSOR_RIGHT`
Funktionstasten:	`W_KEY_F1, W_KEY_F2, W_KEY_F3, W_KEY_F4, W_KEY_F5, W_KEY_F6, W_KEY_F7, W_KEY_F8, W_KEY_F9, W_KEY_F10, W_KEY_F11, W_KEY_F12`
Return, Escape und Tabulatortaste:	`W_KEY_RETURN, W_KEY_ESCAPE, W_KEY_TAB, W_KEY_CLOSE_WINDOW`
Irgend eine andere Spezialtaste:	`W_KEY_OTHER`
Zusätzliche Flags:	`W_KEY_RELEASED = 0x2000, W_KEY_AUTOREPEAT = 0x1000`

Wenn ein Tastenereignis durch ein Autorepeat (Taste lange gedrückt) zustande gekommen ist, wird der Tastencode zusätzlich mit dem Flag `W_KEY_AUTOREPEAT` oder verknüpft (Bit 24 gesetzt). Das Flag `W_KEY_RELEASED` wird gesetzt, wenn die Taste losgelassen wurde, wobei dieses Ereignis nur von der Funktion `GetKeyEvent` geliefert wird.

int IsKeyPressReady (void)

Prüft, ob ein unbehandelter Tastendruck wartet, liefert TRUE (nicht 0) wenn eine Taste gedrückt wurde und FALSE (0) wenn keine Taste gedrückt wurde. Diese Funktion wartet nicht und kehrt sofort zurück. Wenn IsKeyPressed() TRUE zurückliefert, wird GetKeyPress() beim nächsten Aufruf sicher nicht warten, sondern den Code der gedrückten Taste zurückliefern.

void WaitMs (int Time)

Wartet die gegebene Zeit in ms (Milisekunden) ab.

void ClearWindow (void)

Löscht den Inhalt des Graphikfensters.

Erweiterte Funktionen (Für spezielle Anforderungen):

Für zusätzlich Funktionalität zum Manipulieren von Bildern und Behandeln von Tastendrücken und Mausereignissen stehen folgende Funktionen zur Verfügung:

int IsKeyEventReady (void)

Prüft, ob eine Tastenereignis anliegt, liefert TRUE (nicht 0) wenn eine Taste ihren Zustand geändert hat und FALSE (0) wenn keine Taste ihren Zustand geändert hat. Diese Funktion wartet nicht und kehrt sofort zurück. Diese Funktion darf nur zusammen mit GetKeyEvent() gebraucht werden!

int GetKeyEvent (void)

Liefert das älteste, noch nicht verarbeitete Tastenereignis, oder wartet auf ein Tastenereignis falls keines gespeichert ist. Diese Funktion liefert die selben Codes wie GetKeyPress(), wobei neben Tasten Loslass-Ereignissen zusätzlich die unten aufgeführten Codes für die Steuertasten geliefert werden.

```
W_KEY_SHIFT       = 0x0201,
W_KEY_CTRL        = 0x0202,
W_KEY_ALT         = 0x0203,
W_KEY_ALTCTRL     = 0x0204,
W_KEY_META        = 0x0205,
```

MouseInfoType GetMouseState (void)

Liefert den aktuellen Status der Maus (Mausposition und Tastenzustand), der Inhalt von Buttonstate ist eine Kombination (Oder Verknüpfung) von W_BUTTON_NONE, W_BUTTON_LEFT, W_BUTTON_MIDDLE und W_BUTTON_RIGHT, entsprechend der gedrückten Maustasten. Der Status der Maus wird in einer Struktur vom Typ `MouseInfo` geliefert, die wie folgt definiert ist:

```
typedef struct MouseInfo {
  int MousePosX;   /* X-Koordinate der Maus */
  int MousePosY;   /* y-Koordinate der Maus */
  int ButtonState; /* Maustasten Zustand    */
} MouseInfoType;
```

Zur Beschreibung des Tastenzustandes sind folgende Enumkonstanten definiert:

```
W_BUTTON_NO_EVENT    =  0x0000,
W_BUTTON_NONE        =  0x0000,
W_BUTTON_LEFT        =  0x0001,
W_BUTTON_MIDDLE      =  0x0002,
W_BUTTON_RIGHT       =  0x0004,
W_BUTTON_PRESSED     =  0x1000,
W_BUTTON_RELEASED    =  0x2000,
W_BUTTON_DOUBLECLICK =  0x4000,
```

MouseInfoType GetMouseEvent (void)

Liefert das älteste, noch nicht verarbeitete Mausereignis (Mausklick) und die Mausposition bei dessen Eintreten in einer Struktur vom Typ MouseInfoType. Wenn kein Ereignis vorhanden ist wird W_BUTTON_NO_EVENT zurückgeliefert, sonst eine Kombination (Oderverknüpfung) von einem Element aus {W_BUTTON_LEFT, W_BUTTON_MIDDLE W_BUTTON_RIGHT} und einem Wert aus {W_BUTTON_PRESSED W_BUTTON_RELEASED W_BUTTON_DOUBLECLICK}, entsprechend dem Ereignis. Wenn W_BUTTON_NO_EVENT zurückgeliefert wird, haben die mitgelieferten Koordinaten keine Bedeutung.

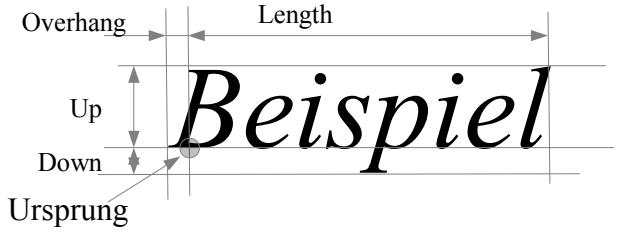
void SelectFont(const char *FontName, int Points, enum FontStyle Style)

Mit dieser Funktion kann der Zeichensatz für zukünftige Textausgaben gewählt werden. Fontname ist der Name des Zeichensatzes (Z.B. "Helvetica"), Points ist die Grösse des Zeichensatzes und in Style kann der Stil (Eine beliebige Kombination [Oderverknüpfung] von `FONT_NORMAL`, `FONT_BOLD`, `FONT_ITALIC`, `FONT_BLACK`) bestimmt werden. Bei FontName kann NULL übergeben werden, dann werden nur die Attribute des aktuellen Fonts geändert.

TextDimensionType GetTextDimensions (const char *Text)

Diese Funktion liefert den Platzbedarf des übergebenen Textes, wenn er mit den aktuellen Einstellungen gezeichnet würde. Der Platzbedarf wird in einer Struktur vom Typ TextDimensionType zurückgeliefert:

```
typedef struct TextDimension {
  int Length;
  int Overhang;
  int Up;
  int Down;
} TextDimensionType;
```



Alle Abmessungen beziehen sich auf die Ursprungskoordinaten, also wie wenn der Text an Position (0,0) gezeichent würde.

extern void SetAutoUpdate (int Mode)

Damit kann festgelegt werden, ob nach jedem Zeichenbefehl der Bilschirm aktualisiert werden soll (Mode = 1), oder ob der Bildschirm nicht aktualisiert werden soll (Mode = 0). Wenn viele Zeichenbefehle nacheinander ausgeführt werden, empfiehlt es sich aus Effizienzgründen, die aktualisierung während des Zeichnens auszuschalten und erst am Schluss wieder einzuschalten.

extern void SetDrawMode(int Mode)

Mit dieser Funktion wird der Zeichnungsmodus festgelegt, das bedeutet, wie neu gezeichnete Objekte auf den bestehenden Bildinhalt gemalt werden. Es stehen folgende Modi zur Verfügung:

DM_COPY:	Der Bildinhalt wird mit dem neuen Objekt (Ohne Berücksichtigung der Transparenz) übermalt.
DM_COPYALPHA:	Schreibt nur die Transparenzwerte des neuen Objektes in den Bildinhalt, d.h. Der Bildinhalt wird entsprechend dem Objekt Transparent gemacht, aber die Farbwerte werden nicht verändert.
DM_OVER:	Der Bildinhalt wird mit dem neuen Objekt (Unter Berücksichtigung der Transparenz) übermalt. D.h. transparente Teile des Objektes übermalen den Bildinhalt nicht.
DM_MIX:	Der Bildinhalt wir mit dem neuen Objekt unter Berücksichtigung der Transparenz gemischt.
DM_XOR:	XOR-Verknüpfung von Bildinhalt und neuem Objekt.
DM_MULTIPLY:	Multipliziert die Farbwerte vom Bildinhalt mit den Farbwerten des neuen Objektes.

(Es können auch die übrigen Modi von QT direkt übergeben werden)

int CreateImage(int Width, int Height)

Erzeugt ein neues, leeres Bild im Speicher, auf dieses kann genau wie auf den Bildschirm gezeichnet werden. Es muss die geforderte Grösse des Bildes übergeben werden. Wenn alles geklappt hat, wird eine Identifikationsnummer für das neue Bild zurückgeliefert, ansonsten ein negativer Wert.

int CreateSubImage(int SrcID, int x, int y, int Width, int Height)

Erzeugt ein neues Bild im Speicher, auf dieses kann genau wie auf den Bildschirm gezeichnet werden. Das Bild wird mit dem Inhalt des Bildes mit der Identifikationsnummer SrcID ausgehend von der Koordinate (x,y) gefüllt. Wenn vom Bildschirm kopiert werden soll, muss `ID_WINDOW` als Source angegeben werden. Die geforderte Grösse des Bildes muss in Width und Height übergeben werden. Wenn alles geklappt hat, wird eine Identifikationsnummer für das neue Bild zurückgeliefert, ansonsten ein negativer Wert.

int LoadImage(const char *FileName)

Lädt ein Bild aus einer Datei in den Speicher, auf dieses kann genau wie auf den Bildschirm gezeichnet werden. Es muss der Dateiname inklusive Pfad des zu ladenden Bildes übergeben werden. Wird NULL übergeben, wird ein Dateiauswahldialog geöffnet. Wenn alles geklappt hat, wird eine Identifikationsnummer für das neue Bild zurückgeliefert, ansonsten ein negativer Wert. Folgende Formate können gelesen werden:

BMP	Windows Bitmap
GIF	Graphic Interchange Format (optional)
JPG	Joint Photographic Experts Group
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PBM	Portable Bitmap
PGM	Portable Graymap
PPM	Portable Pixmap
TIFF	Tagged Image File Format
XBM	X11 Bitmap
XPM	X11 Pixmap

void SaveImage(int Id, const char *FileName, const char *Format, int Quality)

Speichert das mit Id bezeichnete Bild in einer Datei ab. Es muss der Dateiname inklusive Pfad des zu speichernden Bildes übergeben werden. Wird NULL übergeben, wird ein Dateiauswahldialog geöffnet. In Format kann das gewünschte Format ("BMP", "JPG", ...) ausgewählt werden, wenn NULL übergeben wird, wird die Endung des Dateinamens zur Formatbestimmung benutzt. Mit Quality kann die Qualität/Kompression der Datei eingestellt werden. (0 = Maximale Kompression, 100 = Keine Kompression, -1 = Defaulteinstellung benutzen). Folgende Formate können geschrieben werden:

BMP	Windows Bitmap
JPG	Joint Photographic Experts Group
JPEG	Joint Photographic Experts Group
PNG	Portable Network Graphics
PPM	Portable Pixmap
TIFF	Tagged Image File Format
XBM	X11 Bitmap
XPM	X11 Pixmap

`ColorType *GetPixelDataPointer`(int ImageId, int *Size)

Liefert einen Zeiger auf das Graphikarray des ausgewählten Bildes. Erlaubt die direkte Manipulation des Bildinhaltes. In *Size wird die Länge des Arrays in Bytes (Length*Width*4) geliefert (Size auf NULL setzen wenn nicht erforderlich). Achtung, falsche Anwendung kann zu Absturz führen!

void DestroyImage(int ImageId)

Entfernt das Bild mit der angegebenen Identifikationsnummer aus dem Speicher.

void SetEditedImage(int ImageId)

Alle zukünftigen Zeichenoperationen werden auf das Bild mit der angegebenen Identifikationsnummer ausgeführt. Wenn auf den Bildschirm gezeichnet werden soll, muss `ID_WINDOW` als Identifikationsnummer angegeben werden.

void DrawImage(int ImageId, int x, int y)

Zeichnet das Bild mit der angegebenen Identifikationsnummer an die angegebenen Position auf das aktuelle Bild (Normalerweise Bildschirm, kann mit SetEditedImage() geändert werden).

void CopyToImage(int x, int y, int Width, int Height, int ImageId)

Kopiert den mit Position (x,y) und Grösse (Width, Height) ausgewählten Ausschnitt vom aktuellen Bild (Normalerweise Bildschirm, kann mit SetEditedImage() geändert werden) in das Bild mit der angegebenen Identifikationsnummer.

void GetImageSize(int ImageId, int *Width, int *Height)

Speichert die Abmessungen des Bildes mit der angegebene Identifikationsnummer in *Width und *Height ab.

void AddAlphaMask(int ImageId, int Mode, ColorType Color)

Erzeugt automatisch eine Maske (Transparente Bereiche) für das mit ImageID ausgewählte Bild. Es stehen folgende Möglichkeiten zur Auswahl:

MSK_COLOR_IN:	Alle Bildpunkte, die von der angegebenen Farbe sind, werden Transparent.
MSK_COLOR_OUT:	Alle Bildpunkte, die nicht von der angegebenen Farbe sind, werden Transparent.
MSK_AUTO:	Es werden vom Rand her alle Pixel auf Transparent gesetzt, bis ein Farbwechsel auftritt. (Automatisches Maskieren). Die übergebene Farbe hat keine Bedeutung.

void Rotate(float Angle)

Das aktuelle Koordinmatensystem wird um den angegebenen Winkel (In Radiant) rotiert, alle weiteren Zeichenbefehle (Inklusive weitere Aufrufe von Rotate()) beziehen sich auf das rotierte System.

void Translate(float dx, float dy)

Das aktuelle Koordinmatensystem wird um den angegebenen Offset verschoben, alle weiteren Zeichenbefehle (Inklusive weitere Aufrufe von Translate()) beziehen sich auf das verschobene System.

void Scale(float Scalingx, float Scalingy)

Das aktuelle Koordinmatensystem wird um den angegebenen Wert skaliert, alle weiteren Zeichenbefehle (Inklusive weitere Aufrufe von Scale()) beziehen sich auf das skalierte System.

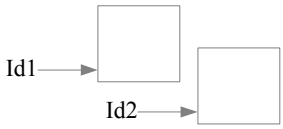
void ResetTransformations(void)

Das aktuelle Koordinatensystem wird wieder in den Ursprungszustand versetzt, d.h. Alle vorherigen Aufrufe von Rotate(), Translate() und Scale() haben keinen Effekt mehr.

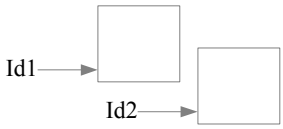
Beispiel zur Arbeit mit Bildern:

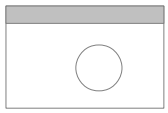
Speicher | Sichtbarer Bildschirm

```
Id1 = CreateImage(30, 30);
Id2 = CreateImage(30, 30);
```

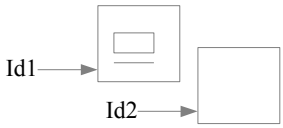


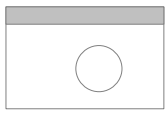
```
DrawCircle(...);
```



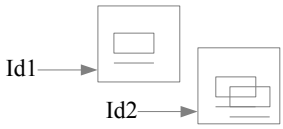


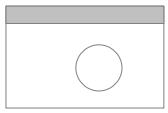
```
SetEditedImage(Id1);
DrawRectangle(...);
DrawLine(...);
```



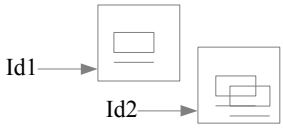


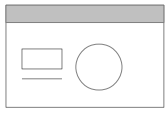
```
SetEditedImage(Id2);
DrawImage(Id1, 0, 0);
DrawImage(Id1, 10, 10);
```



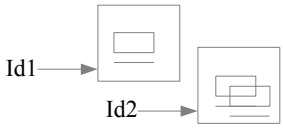


```
SetEditedImage(ID_WINDOW);
DrawImage(Id1, 0, 0);
```





```
Scale(0.5, 0.5);
DrawImage(Id2, 50, 50);
```



void PlaySoundOnce (const char *FileName)

Spielt den angegebenen Sound einmal ab. Je nach Soundkarte sind überlappende Sounds möglich.

void PlaySoundContinuous(const char *FileName)

Spielt den angegebenen Sound fortlaufend ab. Je nach Soundkarte sind überlappende Sounds möglich.

void StopContinuousSound (void)

Beendes das fortlaufende Abspielen eines Sounds.

void StartContinuousSound (void)

Startet das fortlaufende Abspielen eines Sounds erneut.

Das Modul Communication "communication.h"

Das Modul stellt Funktionen zur Kommunikation und zum Datenaustausch über das Netzwerk zur Verfügung. Zur Kommunikation kann eine der beiden Varianten 1 oder 2 verwendet werden, es sind beide Varianten im selben Modul implementiert. Empfohlen wird der Einsatz der Variante 2.

Variante 1:

Bei der Variante 1 wird auf jedem Rechner ein vollständiges Abbild des Spiels (Spielfeld) lokal gespeichert und über das Netzwerk mit den anderen Rechnern synchronisiert.

Vordefinierte Datentypen:

Zur Kommunikation zwischen Client und Server werden Telegramme verwendet. Diese Telegramme sind durch den Typ MessageTyp definiert:

```
typedef struct MessageType {
   enum CommandEnum Command;     /* Befehl, der Uebertragen wird */
   MessageData      Data;        /* Zum Befehl gehörende Daten */
} MessageType;
```

Für im Telegramm enthaltenen Befehle (Command Feld der Struktur) sind folgende Konstanten definiert:

```
CMD_NONE          /* Kein Befehl (Platzhalte, No Operationr) */
CMD_KEYSTROKE     /* Es wird ein Tastendruck übertragen */
CMD_SET_FIELD     /* Ein bestimmtes Feld im Spielfeld auf eine bestimmte Farbe und Form setzen */
CMD_INIT_FIELD    /* Das Spielfeld initialisieren (Alles löschen) und Spielfeldgrösse festlegen*/
CMD_SHOW_MESSAGE  /* Eine Nachricht Anzeigen */
CMD_SHOW_STATUS   /* Einen Status euf einer bestimmten Statuszeile anzeigen */
CMD_USER_DATA     /* Das Datenfeld enthält Benutzerdefinierte Daten */
```

Das Datenfeld besteht aus einer Union von verschiedenen Datenfeldern. Welches der Datenfelder in der Union gültig ist wird durch das Kommando in Command festgelegt:

```
typedef union MessageData {
   int               KeyCode;      /* Tastencode, CMD_KEYSTROKE */
   SetFieldData      SetField;     /* Felddaten, CMD_SET_FIELD  */
   InitFieldData     InitField;    /* Initdaten, CMD_INIT_FIELD */
   ShowMessageData   ShowMessage;  /* Messagedaten, CMD_SHOW_MESSAGE*/;
   ShowStatusData    ShowStatus;   /* Statusdaten, CMD_SHOW_STATUS */;
   UserData          User;         /* Userdaten, CMD_USER_DATA */;
} MessageData;
```

Die verschiedenen Datenfelder sind wie folgt definiert:

Daten für setfield (CMD_SET_FIELD). Dient zum Setzen und Löschen einer Zelle im Spielfeld:

```
typedef struct SetFieldData {
   unsigned int x;       /* X-Position des zu setzenden Feldes */
   unsigned int y;       /* Y-Position des zu setzenden Feldes */
   unsigned int Color;   /* Farbe, auf die das Feld zu setzen ist (Bit 0-23) */
   /* Bit 28-31 geben die Art des Feldes an: 0000 = Wurmsegment, 0001 = Futter */
} SetFieldData;
```

Daten für initfield (CMD_INIT_FIELD). Dient zur Initialisierung des Spielfeldes.

```
typedef struct InitFieldData {
   unsigned int Width;   /* Breite des Spielfeldes */
   unsigned int Height;  /* Höhe des Spielfeldes */
} InitFieldData;
```

Daten für showmessage (CMD_SHOW_MESSAGE), Dient zur Anzeige eines Textes in der Messagezeile.

```
typedef struct ShowMessageData {
   unsigned int  Length;  /* Länge des Textes in Bytes */
   char Text[1];          /* Längenvariables Textarray mit Text (Nullterminiert) */
} ShowMessageData;
```

Daten für showstatus (CMD_SHOW_STATUS). Dient zur Anzeige von Statusinformationen in der angegebenen Statuszeile.

```
typedef struct ShowStatusData {
   unsigned int  Line;    /* Nummer der Zeile, in welcher der Text angezeigt werden soll */
   unsigned int  Color;   /* Farbe in der der Text dargestellt werden soll */
   unsigned int  Length;  /* Länge des Textes in Bytes */
   char Text[1];   /* Längenvariables Textarray mit Text (Nullterminiert) */
} ShowStatusData;
```

Feld für frei definierbare Benutzerdaten. (userdatas / CMD_USER_DATA). Damit können vom Anwender beliebige Daten übertragen werden.

```
typedef struct UserData {
   unsigned int  Length;  /* Länge der Daten in Bytes */
   char          Data[1];/* Längenvariables Datenarray */
} UserData;
```

Variante 2:

Bei der Variante 2 werden nur Tastendrücke und direkte Zeichenbefehle ausgetauscht. Der Client ist also nur ein 'Dummes' Terminal welches Graphikbefehle versteht und ausführen kann.

Vordefinierte Datentypen (Variante 2):

Zur Kommunikation zwischen Client und Server werden Telegramme verwendet. Diese Telegramme sind durch den Typ MessageTyp definiert:

```
typedef struct MessageType {
   enum CommandEnum Command;   /* Befehl, der Uebertragen wird */
   MessageData      Data;      /* Zum Befehl gehörende Daten */
} MessageType;
```

Für im Telegramm enthaltenen Befehle (Command Feld der Struktur) sind folgende Konstanten definiert (Weitere Konstanten können nach Bedarf definiert werden):

```
CMD_NONE                    /* Kein Befehl (Platzhalte, No Operationr) */
CMD_KEYSTROKE               /* Es wird ein Tastendruck übertragen */
CMD_DRAW_EMPTY_CIRCLE       /* Zeichnet einen leeren Kreis  */
CMD_DRAW_FILLED_CIRCLE      /* Zeichnet einen gefüllten Kreis  */
CMD_DRAW_EMPTY_RECTANGLE    /* Zeichnet ein leeres Rechteck  */
CMD_DRAW_FILLED_RECTANGLE   /* Zeichnet ein gefülltes Rechteck  */
CMD_DRAW_LINE               /* Zeichnet eine Linie  */
CMD_DRAW_TEXT               /* Zeichnet einen Text  */
CMD_DRAW_CLEAR_ALL          /* Löscht die Zeichenfläche */
CMD_USER_DATA               /* Das Datenfeld enthält Benutzerdefinierte Daten */
```

Das Datenfeld besteht aus einer Union von verschiedenen Datenfeldern. Welches der Datenfelder in der Union gültig ist wird durch das Kommando in Command festgelegt:

```
typedef union MessageData {
   int               KeyCode;       /* Tastencode, CMD_KEYSTROKE */
   DrawCircleData    DrawCircle;    /* Zeichendaten, CMD_DRAW_EMPTY_CIRCLE ,
                                       CMD_DRAW_FILLED_CIRCLE  */
   DrawRectangleData DrawRectangle; /* Zeichendaten, CMD_DRAW_EMPTY_RECTANGLE
                                       CMD_DRAW_FILLED_RECTANGLE */
   DrawLineData      DrawLine;      /* Zeichendaten, CMD_DRAW_LINE */;
   DrawTextData      DrawText;      /* Zeichendaten, CMD_DRAW_TEXT */;
   UserData          User;          /* Userdaten, CMD_USER_DATA */;
} MessageData;
```

Die Struktur NetColorType dient zur Aufnahme eines RGB Farbwertes. Die einzelnen Farbanteile müssen im Bereich von 0 bis 255 liegen. Wobei (0,0,0) schwarz und (255,255,255) weiss entspricht .

```
typedef struct NetColorType {
   unsigned char Red;
   unsigned char Green;
   unsigned char Blue;
   unsigned char Alpha
} NetColorType;
```

Die verschiedenen Datenfelder sind wie folgt definiert (Weitere Datenfelder können nach Bedarf definiert werden):

Daten für CMD_DRAW_FILLED_CIRCLE. Dient zum Zeichen eines gefüllten Kreises.

```
typedef struct DrawCircleData {
   int x;               /* x-Position des Kreises */
   int y;               /* y-Position des Kreises */
   int Width;           /* Breite des 'Kreises'   */
   int Height;          /* Hoehe des 'Kreises'    */
   NetColorType Color; /* Farbe des Kreises      */
} DrawCircleData;
```

Daten für CMD_DRAW_FILLED_RECTANGLE und CMD_DRAW_EMPTY_RECTANGLE. Dient zum Zeichen eines (un)gefüllten Rechteckes auf dem Bildschirm:

```
typedef struct DrawRectangleData {
   int x;               /* x-Position des Rechtecks */
   int y;               /* y-Position des Rechtecks */
   int Width;           /* Breite des Rechtecks */
   int Height;          /* Hoehe des Rechtecks */
   NetColorType Color; /* Farbe des Rechtecks */
} DrawRectangleData;
```

Daten für CMD_DRAW_LINE. Dient zum zeichnen einer Linie.

```
typedef struct DrawLineData {
   int x1;              /* Start-Position der Linie */
   int y1;              /* Start-Position der Linie */
   int x2;              /* End-Position der Linie */
   int y2;              /*End-Position der Linie */
   NetColorType Color;  /* Farbe der Linie */
   int Width;           /* Breite der Linie */
} DrawLineData;
```

Daten für CMD_DRAW_TEXT, Dient zur Anzeige eines Textes.

```
typedef struct DrawTextData {
   int x;                /* Position des Textes */
   int y;                /* Position des Textes */
   NetColorType Color;   /* Farbe des Rechtecks */
   unsigned int  Length; /* Länge des Textes in Bytes */
   char Text[1];         /* Längenvariables Textarray mit Text (Nullterminiert) */
} DrawTextData ;
```

Feld für frei definierbare Benutzerdaten. (CMD_USER_DATA). Damit können vom Anwender beliebige Daten übertragen werden.

```
typedef struct UserData {
   unsigned int  Length; /* Länge der Daten in Bytes */
   char          Data[1];/* Längenvariables Datenarray */
} UserData;
```

Variante 1 und 2:

Das Modul Communication stellt folgende Funktionen zur Verfügung:

char **InitializeCommunicationWithPort(char *MyIpAddress, unsigned int MaxTgLength, unsigned int LocalPort, unsigned int RemotePort)

Initialisiert das Kommunikationsmodul. Muss aufgerufen werden, bevor irgendeine der anderen Funktionen dieses Moduls verwendet werden darf. An der Addresse MyIpAddress wird von der Funktion ein Nullterminierter String mit der ersten verfügbaren lokalen IP-Adresse abgelegt. Es muss genug Platz für 16 Zeichen vorhanden sein. MaxTgLength definiert die maximal erlaubte Länge der zu versendenden Telegramme, LocalPort und Remote Port definieren die TCP/IP Ports für Server und Client (0 = Default verwenden). Die Funktion liefert ein Array von Zeiger auf Strings zurück, welches alle verfügbaren lokalen IP Adressen enthält. Das Ende des Arrays ist mit einem Nullpointer markiert.

void ShutdownCommunication(void)

Muss aufgerufen werden bevor das Programm verlassen wird, gibt alle vom Modul belegten Resourcen wieder frei.

MessageType *CreateTelegram(unsigned int AdditionalLength)

Damit wird ein neues, leeres Telegramm erzeugt. Falls das Telegramm ein Element mit variabler Datenlänge enthält (ShowMessage, ShowStatus, DrawText, User) muss in AdditionalLength die Länge des variablen Feldes angegeben werden. Falls kein Feld mit variabler Länge vorliegt muss der Wert 0 übergeben werden. Die Funktion liefert einen Zeiger auf das erzeugte Telegramm, oder im Fehlerfall NULL zurück.

void DeleteTelegram(MessageType *Message)

Mit dieser Funktion können Telegramme nach Gebrauch wieder freigegeben werden. Es dürfen nur Telegramme freigegeben werden, die mit CreateTelegramm() oder CheckForTelegramm() erzeugt wurden.

void SendTelegramTo(int Handle, MessageType *Message)

Mit dieser Funktion wird ein Telegram an den Host der mit der Verbindung Handle verknüpft ist, geschickt. Das Telegramm wird von der Funktion selbst freigegeben und darf nicht mehr verwendet und auch nicht mit DeleteTelegram() gelöscht werden.

MessageType *CheckForTelegram(int *Handle)

Diese Funktion prüft, ob ein Telegram empfangen wurde. Wenn seit dem letzten Aufruf dieser Funktion ein Telegramm über eine bestehende Verbindung empfangen wurde, wird dieses Telegramm zurückgeliefert, sonst wird NULL zurückgeliefert. Die Funktion wartet nicht, sondern kehrt in jedem Fall sofort zurück. In Handle wird das Handle der Verbindung abgelegt, über die das Telegramm empfangen wurde. Um keine Telegramme zu verlieren muss diese Funktion periodisch aufgerufen werden. Das empfangene Telegramm muss nach der Auswertung mit DeleteTelegram() freigegeben werden.

int ConnectTo(char *Address, char *MyName, unsigned int Timeout)

Baut eine Verbindung zum Host mit der angegebenen IP-Adresse auf. Name ist ein String, mit dem sich dieser Rechner beim Host identifiziert (Und dort von Accept() weitergeleitet wird). Der String darf maximal 400 Zeichen lang sein. Address zeigt auf einen Nullterminierten String mit einer gültigen IP Adresse (xxx.xxx.xxx.xxx). Die Funktion liefert ein Handle zur Identifikation dieser Verbindung bei späterer Verwendung, oder -1 wenn die Verbindung fehlgeschlagen ist. Die Funktion versucht maximal während der Zeit Timeout (in ms) eine Verbindung aufzubauen, bevor sie mit -1 zurückkehrt. Diese Funktion wird normalerweise von Clienten verwendet.

int IsConnected(int Handle)

Diese Funktion überprüft, ob die Verbindung mit dem Handle Handle noch existiert. Liefert TRUE (1) wenn die Verbindung OK ist, und FALSE (0) wenn die Verbindung abgebrochen wurde.

void Disconnect(int Handle)

Bricht die Verbindung mit dem Handle Handle ab. Sollte für jede Verbindung aufgerufen werden, die nicht mehr benötigt wird. Damit wird auch das Handle für spätere Wiederverwendung freigegeben.

int AcceptConnection(char *NameOfConnector, int MaxLength, char *MyName)

Akzeptiert eine Verbindungsaufnahme eines anderen Rechners (Mittels Connect). Die Funktion liefert ein Handle für die Verbindung zurück, falls seit dem letzten Aufruf von AcceptConnection() ein Rechner eine Verbindung aufnehmen wollte, oder -1 wenn keine Verbindungsaufnahme stattgefunden hat. Im Buffer NameOfConnector wird die Identifikation (Der Text der bei Connect() vom Verbindungspartner übergeben wird) abgelegt. Der Buffer muss gross genug gewählt werden, es wird keine Überlaufsprüfung durchgeführt (Es werden aber nie mehr als **MaxLength** Zeichen abgelegt). Diese Funktion muss periodisch aufgerufen werden, solange Verbindungen akzeptiert werden sollen. (Serverbetrieb). Mit dem String MyName identifiziert sich der Server auf Broadcastanfragen (FindServer() von Clienten).

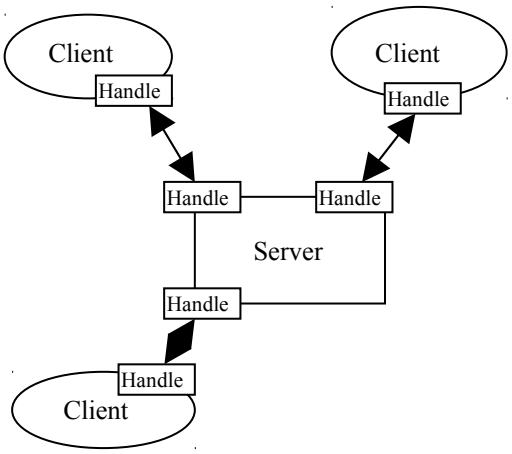
ServerInfoType *FindServer(unsigned int Timeout)

Sucht während der angegebenen Zeit nach passenden Servern und liefert einen Zeiger auf ein Array von Strukturen des Typs ServerInfoTyp zurück. Im Array sind alle gefundenen Server eingetragen, das Ende des Arrays ist mit einem Leerstring im Feld IPAdress markiert (IPAdress[0] = 0). Es werden nur Server gefunden, die periodisch AcceptConnection() aufrufen, der Name des Servers stammt von AcceptConnection() des Servers. (Router und Firewalls können das automatische Auffinden eines Servers verhindern, da diese Broadcastmeldungen meist nicht weiterleiten)

```
typedef struct ServerInfoType {
   char IPAddress[20];   /* IP Adresse des Severs */
   char Name[100];       /* Name des Servers      */
} ServerInfoType;
```

Eine Verbindung zwischen einem Server und einem Clienten ist an beiden Enden durch ein (Nicht zwingend identisches) Handle definiert. Ein Handle ist einfach eine Nummer, welche eine Verbindung auf dem jeweiligen Rechner identifiziert. Beim Aufbau einer Verbindung wird von ConnectTo() (Client) respektive AcceptConnection() (Server) je ein Handle (Nummer) zurückgeliefert, und über dieses Handle wird die Verbindung bei zukünftigem Datenverkehr (Senden und Empfangen von Telegrammen) identifiziert. Jede Verbindung besitzt an beiden Enden ein eigenes (anderes) Handle.

Beispiel für Handle und Verbindungen



Der Server hat 3 verschiedene Handles, für jede Verbindung eines.
Jeder Client hat je ein Handle.

Grundsätzlicher Ablauf:

Server	**Client**
InitializeCommunication()	InitializeCommunication()
	ConnectTo()
AcceptConnection()	
CreateTelegram()	
Telegram ausfüllen	
SendTelegramTo()	
	CheckForTelegram()
	Telegram auswerten
	DeleteTelegram()
	CreateTelegram()
	Telegram ausfüllen
	SendTelegramTo()
CheckForTelegram()	
Telegram auswerten	
DeleteTelegram()	
ShutdownCommunication()	ShutdownCommunication()

Das Zuordnen von Handles zu Spielern und das Auffüllen oder Auswerten von Telegrammen ist Sache des Anwenders dieses Moduls.

Das Modul ErrorHandler "Error.h"

Das Modul stellt Funktionen zur Behandlung von Laufzeitfehlern zur Verfügung.

Vordefinierte Konstanten:

Der Errorhandler stellt folgende drei Konstanten zur Qualifizierung von Fehlern (Errorlevels) zur Verfügung:

ERR_FATAL	*/* Fataler Fehler, das Programm wird nach Ausgabe der Fehlermeldung sofort abgebrochen */*
ERR_NON_FATAL	*/* Nicht fataler Fehler, das Programm wird nach Ausgabe der Fehlermeldung fortgesetzt */*
ERR_WARNING	*/* Warnung, es wird eine Warnungsmeldung ausgegeben, das Programm wird nach Ausgabe der Meldung fortgesetzt */*

Das Modul Errorhandler stellt folgende Makros zur Verfügung:

CODING_ERROR(Message)
Gibt die Fehlermeldung Message aus, sowie weitere Informationen wie Dateinamen und Zeilennummer des Makroaufrufs. Das Programm wird mit einem fatalen Fehler abgebrochen.

RUNTIME_ERROR(Message, Level)
Gibt die Fehlermeldung Message mit der Qualifizierung Level (ERR_WARNING, ERR_NON_FATAL, ERR_FATAL), sowie weitere Informationen wie Dateinamen und Zeilennummer des Makroaufrufs. Das Programm wird nur bei einem fatalen Fehler abgebrochen.

REJECT_NULLPOINTER(Ptr)
Gibt die Fehlermeldung "Nullpointer Error" aus, wenn Ptr ein NULL-Zeiger ist, sowie weitere Informationen wie Dateinamen und Zeilennummer des Makroaufrufs. Das Programm wird mit einem fatalen Fehler abgebrochen.

ASSURE_MEMORY_AVAILLABLE(Ptr)
Gibt die Fehlermeldung "Out of Memory" aus, wenn Ptr ein NULL-Zeiger ist, sowie weitere Informationen wie Dateinamen und Zeilennummer des Makroaufrufs. Das Programm wird mit einem fatalen Fehler abgebrochen. Diese Funktion kann bei malloc() zum Test auf erfolgreiche Speicherreservierung eingesetzt werden.

Das Modul wormmain_example.c

Das ist ein kurzes Beispiel zum Einsatz der Graphik und Netzwerk Bibliothek.

int gfxmain(int argc, char* argv[], const char *ApplicationPath)

Diese Funktion **muss in jedem Projekt** vorhanden sein, hier startet das Programm. (Entspricht main() von normalen C-Programmen. Die Argumente argc und argv entsprechen denjenigen von main(), in ApplicationPath ist der aktuelle Pfad zur Applikation abgelegt (wo auf dem Laufwerk die .exe-Datei abgespeichert ist).

Hier ist eine kleine Beispielapplikation realisiert, um die Möglichkeiten der Graphik und Netzwerk-Bibliotheken zu demonstrieren.

Sie besteht aus 5 sequentiell nacheinander Ablaufenden Teilen durch drücken einer Taste im Graphikbildschirm kann jeweils zum nächsten Teil gelangt werden..

Im ersten Teil werden normale und animierte Graphik sowie die Abfrage der Mausposition demonstriert.

Im zweiten Teil wird das Laden und (skalierte/rotierte) Darstellen von (animierten) Bildern gezeigt.(Beim Klicken und Loslassen der Maustasten werden zusätzlich farbige Kreise gezeichnet).

Im dritten Teil werden mausabhängige Animationen gezeigt (Durch klicken und ziehen [drag] kann ein Ball 'geworfen' werden).

Im vierten Teil ist ebenfalls eine mausabhängige Animation zu sehen, der Mauszeiger wirkt als Anziehungspunkt für ein Objekt. Durch doppelklick wird das Objekt wieder auf den Nullpunkt gesetzt.

Im fünften Teil ist eine kleine Netzwerkdemonstration enthalten. Wenn das Programm auf mehreren über Netzwerk verbundenen Rechnern gestartet wird, kann auf dem Server gezeichnet werden, und die Zeichnung wird auf den Clients dargestellt. Das Programm sucht zuerst nach Servern auf dem Netz, und wenn es keine Server findet, wird es selbst zum Server. Wenn es einen Server findet, wird es zum Clienten und verbindet sich mit dem Server. Es können auch zweimal auf dem selben Rechner gestartet werden, dann müssen aber beide Instanzen unterschiedliche Ports benutzen, dies wird erreicht, indem beim einen mit gross 'S' vom Teil 4 weitergeschaltet wird, und beim anderen mit gross 'C'.

void DrawTransformedImage(int x, int y, float Angle, float ScaleX, float ScaleY, int Img)

Eine Hilfsfunktion zum einfachen Zeichnen von transformierten (skalierten und rotierten) Bildern. Das Bild **Img** wird um den Winkel **Angle** (in Radiant) rotiert und um die Skalierungsfaktoren **ScaleX** und **ScaleY** skaliert an der Position (**x**, **y**) gezeichnet.

ImageMapType *CreateImageMap(int MyImage)

Eine Hilfsfunktion zum Erzeugen eines Pointervektors, erlaubt den einfachen, direkten Zugriff auf die Pixeldaten eines Bildes in Matrix-Notation, Achtung, y kommt vor x-Koordinate.

```
ImageMapType *Map    = CreateImageMap(int MyImage);
Map->ImageMap[y][x]  = Color; // x < map->Width, y < map->Height !!
```

void DestroyImageMap(ImageMapType * ImageMap)

Eine Hilfsfunktion zum Freigeben eines Imagemaps.