Informatik

Projekt **P**rotokoll

GTML- Interpreter

Stichworte:

Der Interpreter „Gygo Text Markup Language“ ist der Versuch, in einem Pascal-Programmierprojekt einen HTML-Interpreter nachzubauen.

Dazu wurden verwendet:

- ARRAYs
- STRINGs
- FILEs
- UNITs
- MOUSEhandling

Schule Gymnasium Gonsenheim Gygo, Mainz

Kurs GK Inf 12.1 99/00

Lehrer Josef Glöckler

Projekt-leitung Eine Schülergruppe

Inhalt

Arbeitsgruppe Informatik LMZ

Inhaltsverzeichnis

1 Die Vorbereitung

1.2 Allgemeines: Erwartungshaltung des Kurses und des Lehrers, Aufstellung von Verhaltensregeln und Beurteilungskriterien

Bei einem kleinen Vorprojekt in 12.1 zum Thema „Grafik“, bei dem ich als streng reglementierender Projektleiter fungierte, gewannen die Kursteilnehmer einen Einblick in die Chancen und Probleme der Projektarbeit. Auf der einen Seite trauten sie sich zu, selbstständig ein Projekt zum Abschluss zu bringen, auf der anderen Seite waren sie vorgewarnt, dass ein Projekt viel Arbeit und eigenes Engagement bedeutet. Trotzdem war der Kurs heiß auf ein Projekt: das Selbermachen, das selbstgesteuerte Arbeiten lockte.

Aufgrund dieser Erfahrungen trafen wir von vorneherein Absprachen über das Verhalten und die Beurteilungskriterien während der Projektzeit.

- Die Schüler wollten Themen selbst beibringen und auswählen. Ich wollte mich allerdings als Ideengeber nicht ausschließen lassen.
- Die Schüler wollten das Projekt selbst leiten. Der Lehrer sollte nur als wandelndes Lexikon im Hintergrund bereitstehen. Dazu fand ich mich gerne bereit. Allerdings behielt ich mir vor einzugreifen, falls die Dinge nicht so laufen, wie ich es verantworten konnte.
- Die Schüler wollten selbst die Zeitplanung übernehmen und die Einhaltung des Zeitrasters überwachen. Ich hatte nichts dagegen, allerdings legte ich Wert darauf, dass die Projektarbeit in der Schule stattfand und nicht in außerschulische Räume verlegt wurde. Ich wollte sehen, wer die Arbeit macht und das auch als eine Bewertungsgrundlage verwenden.
- Die Schüler wollten die Projektarbeit mitbewerten. Ich war gerne dazu bereit, behielt mir aber doch als verantwortlicher Lehrer die letzte Kompetenz vor.

1.3 Themenfindung

Die Themenfindung für das Projekt erwies sich als nicht so einfach wie gedacht. Von den Schülern kamen nur diffuse Vorschläge, wie „Etwas zum Internet“, „Homepage gestalten“, „Grafik-Programm“, „Sound und Multimedia“. Mein erster Vorschlag war, ein Spiel zu gestalten, bei dem die Spieler nicht gegeneinander spielen, sondern zusammenarbeiten müssen, um das Spielziel zu erreichen. Aber das war für meine Schüler offenbar nicht spannend genug.

Für die weitere Diskussion schob ich die Bedingung nach, dass das Thema mit Dateien und Datenschutz etwas zu tun haben sollte.

Aber auch ohne diese Anforderungen hatten die Schüler große Schwierigkeiten, ihre Themen zu konkretisieren. Gründe dafür waren wohl, dass sie weder den Aufwand, noch die eigene Leistungsfähigkeit einschätzen konnten. Darüber hinaus fehlte ihnen das Wissen darüber, was mit Turbo Pascal machbar ist und was nicht.

So versuchte ich auf der einen Seite, die vorgeschlagenen Schüler-Themen aufzunehmen, gleichzeitig aber auf einen realisierbaren Umfang zu reduzieren. Nach einiger Diskussion einigten wir uns darauf, einen Internetbrowser vereinfacht nachzubauen, um zu verstehen, wie ein solches Programm im Prinzip funktioniert. Wir beschlossen, eine eigene Sprache zu verwenden, die wir

in Anlehnung an HTML „**G**ygo **T**ext **M**arkup **L**anguage", kurz GTML nannten. Dabei steht „Gygo" für „Gymnasium Gonsenheim".

1.4 Organisation: Wahl der Projektleitung, Verteilung der „Funktionsstellen"

Da Schüler die Projektleitung übernehmen wollten, musste diese Aufgabe genau definiert werden. Dazu ergaben sich aus der Diskussion folgende Schwerpunkte:

- Aufgliederung der Themenstellung und Aufgabenverteilung.
- Erstellung des Zeitrahmens.
- Kontrolle des Arbeitsfortschrittes und Einhaltung des Zeitrahmens.
- Überwachung der Protokollierung der Arbeitsergebnisse.

Für die Projektleitung stellten sich zwei Teams zur Verfügung, denen ich die nötige Durchsetzungskraft zutraute. Die Schüler wählten in geheimer Abstimmung ihre Projektleiter, einen Stellvertreter und zwei Protokollanten. Dem nichtgewählten Team fiel die Aufgabe zu, das Hauptprogramm zu erstellen und die getesteten Module einzubinden. Die restlichen Schüler fanden sich ohne mein Eingreifen zu zweit oder zu dritt in Arbeitsgruppen zusammen.

1.5 Programmiertechnik: Dynamische Datenstrukturen(File, Listen), Maussteuerung

Turbo Pascal ist zurzeit noch die eingeführte Programmiersprache am Gymnasium Gonsenheim. Deshalb mussten zur Vorbereitung der Projektarbeit spezielle programmtechnische Kenntnisse aufgefrischt bzw erarbeitet. werden.

- Den Umgang mit der Maus und das Verstecken der Mausbefehle in einer Unit wurden schon in 12.1 im Rahmen eines kleinen Grafikprojektes erledigt.
- Ohne den Umgang mit Files war keines der Projekte, die mir denkbar erschienen, möglich. Deshalb war „Filehandling" das erste Thema in 12.2.
- Damit nicht „Files" als einzige dynamische Datenstruktur stehen blieb, gab ich dem Kurs einen Überblick darüber, was an dynamischen Datentypen in Turbo Pascal möglich ist. Die Datenstruktur „verkettete Liste" wurde zwar behandelt, aber bei der Realisierung des Projektes nicht wieder aufgegriffen.

2 Die Projektarbeit

2.1 Planung

Die Grobstruktur des geplanten Browsers wurde gemeinsam entwickelt, damit jeder verstand, wie sein eigener Teil im Rahmen des Ganzen funktionieren sollte. Danach hatte jede Arbeitsgruppe ihre Aufgaben auftragsgemäß selbstständig zu erledigen. Es sollte mit „einfachen“ GTML-Befehlen angefangen werden, und erst wenn diese zufrieden stellend funktionierten, sollten komplexere Befehle in Angriff genommen werden. Jede Arbeitsgruppe hatte also nicht nur einen GTML-Befehl zu realisieren, sondern je nach Arbeitsfortschritt mehrere und verschieden schwere.

Vom Zeitrahmen her ergab sich für das Projekt ein natürliches Ende: die Präsentation des Produktes am Tag der offenen Tür im Dezember. Insgesamt wollte ich das Projekt aber früher fertig gestellt haben, da ich bei Projekten in vorangegangenen Jahren schon negative Erfahrungen mit der Abgabepünktlichkeit gemacht hatte.

Es wurde beschlossen, dass nach den Herbstferien Mitte Oktober ein erstes Programm mit einfachen Befehlen laufen sollte, bevor man sich in schwierigere Aufgaben stürzte. Anfang Dezember sollte der endgültige Abgabetermin liegen. Dabei war klar, dass die Projektarbeit immer wieder durch Phasen „normalen“ Unterrichts unterbrochen werden musste, beispielsweise um Programmierungsgrundlagen zu erarbeiten oder um die Kursarbeit vorzubereiten.

2.2 Aufteilung des Themas

Die Aufteilung des Themas ergab sich auf ganz natürliche Weise:

- Das Hauptprogramm sollte jeweils die Informationen, die zu einer Seite gehören, aus einer Textdatei lesen, die GTML-Befehle an dem Zeichen < am Zeilenanfang erkennen und die Befehlszeile insgesamt an die entsprechende Arbeitsgruppe zur weiteren Analyse und Bearbeitung übergeben.
- Die Arbeitsgruppen sollten aus der übernommenen Befehlszeile den Befehl und die Parameter für den Befehl isolieren und anwenden. Der Einfachheit halber sollten zuerst Turbo Pascal nahe Befehle wie <FARBE: ....> (entspr. setcolor ....) oder <LINIENSTIL: ...> bearbeitet werden, bevor man sich z. B. Laufschriften oder blinkenden Schriften zuwandte. Da speziell beim Teilthema „Hintergrundgrafiken“ größere Probleme gesehen wurden, wurde eine „Forschungsgruppe” eingerichtet, die sich hierüber informieren und dann ihr Know-how einbringen sollte.

2.3 Startschwierigkeiten, Machbarkeitsstudien

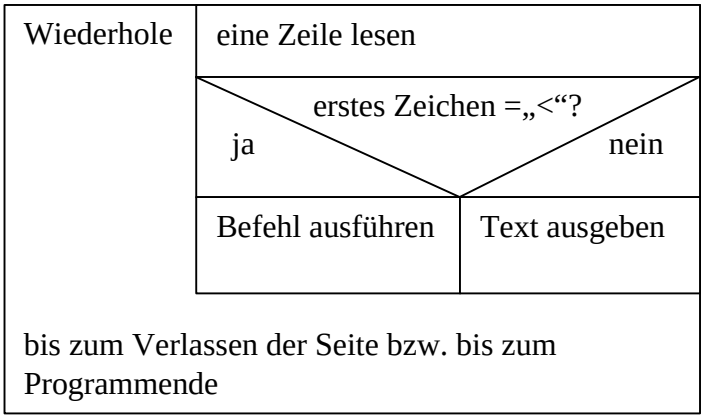
Schon zu Beginn der Projektarbeit ergaben sich riesige Unsicherheiten, wie denn nun die Arbeit konkret vonstatten gehen sollte. Immer wieder konfrontierten mich die Schüler mit dem Satz: „Ich weiß nicht, wie ich anfangen soll.“. Sie konnten sich das Zusammenwirken ihrer Prozeduren mit dem Hauptprogramm nicht vorstellen. Zusammen mit der Projektleitung gab ich den Rat, jeweils ein eigenes rudimentäres Hauptprogramm zu schreiben, in dem die eigene Prozedur getestet werden konnte. Ich ermutigte die Projektteilnehmer, erst einmal einfache Teillösungen in

Angriff zu nehmen und zu testen, um zu sehen, was man denn überhaupt erreichen konnte. Anhand eines Beispiels waren dann auch die Zögerlichen zu überzeugen

2.4 Groblösung

Im Kern sollte der Interpreter aus einer Schleife bestehen, die

- eine Textzeile einliest,
- überprüft, ob ein Befehle vorliegt
- wenn ja, die entsprechende Prozedur aufruft, wenn nein, den Text auf dem Bildschirm darstellt.



2.5 Verfeinerungen

Im Verlauf der weiteren Planung wurde klar, dass die einfache Struktur dynamischen Befehlen wie LAUFSCHRIFT oder LABEL und auch der Behandlung dynamischer Ereignisse wie z. B. Mausabfragen nicht gerecht werden konnte. Eine Lösung wurde dadurch erreicht, dass dynamische Befehle zuerst nur gesammelt und dann in einer weiteren Schleife immer wieder abgearbeitet wurden, bis die Seite verlassen wurde. Für das Sammeln bot ich eine verkette Liste aus Strings an, die dem Kurs aber zu kompliziert erschien. Man entschied sich für ein ARRAY, obwohl damit die Allgemeinheit der Lösung aufgegeben wurde.

Bei weiteren Lösungsversuchen wurde auch deutlich, dass die Werteübergabe eines Strings folgendes Problem nicht lösen konnte: dynamische Befehle müssen bei jeder Wiederholung da weitermachen, wo sie zuvor stehen geblieben sind. Dieses Problem konnte der Kurs nicht ohne meine Hilfe bewältigen. Er erhielt deshalb die Information, dass hierzu ein Variablenparameter für die Stringübergabe erforderlich ist.

2.6 Integration der Teillösungen

Die Integration der Teillösungen schien bei den statischen Befehlen kein Problem. Aber der Teufel saß im Detail: Manche Turbo Pascal Befehle beeinflussen sich gegenseitig. So arbeitet z. B. die Liniendicke nicht unabhängig vom Linienstil. Wir mussten also jedes mal vor dem Aufruf der Liniendicke den Linienstil feststellen, damit nicht bei jeder Anwendung von Liniendicke der Linienstil unabsichtlich verändert wurde. Im Prinzip betraf das alle Befehle, bei denen mehrere Eigenschaften in einem Funktionsaufruf angesprochen werden.

Die Integration der dynamischen Befehle gelang zunächst überhaupt nicht. Die Programmierer, die diese Teillösungen erarbeiteten, nahmen die Abkapselung der Prozedur vom Hauptprogramm nicht ernst genug. Deshalb kam es immer wieder vor, dass aus der Prozedur auf globale Variable zurückgegriffen wurde, die zwar im Testprogramm vorhanden waren, aber nicht im Original des Hauptprogramms.

Ein ungelöstes Problem blieb, wie man ohne Timer die Geschwindigkeit dynamischer Abläufe steuern kann. Je mehr dynamische Befehle und je zeitaufwendiger die dynamischen Befehle waren, umso langsamer lief die Dynamik ab. Der Befehl „delay" war in diesem Zusammenhang untauglich, da er die ganze Schleife blockierte.

Da die Programmierer des Hauptprogrammes mit einigen Programmierern von Teillösungen die Geduld verloren, schrieben sie z. T. die fehlenden Prozeduren der Teillösungen trotz meiner Einwände selbst. Dabei hielten sie sich nicht streng an die verabredete Schnittstelle, weil sie zu Recht glaubten, dass man einige Dinge wesentlich effektiver programmieren könne, als es mit unserem Datenmodell möglich war. Leider funktionierte zu diesem späten Zeitpunkt die gemeinsame Lösungssuche nicht wie geplant. Es kam zu keiner Diskussion, da die meisten Gruppen mit ihrem Projektteil beschäftigt waren und nicht bereit waren, den Lösungsansatz noch einmal umzustellen. Nach der ursprünglichen Planung sollte jede Gruppe ihren String selbst analysieren. Das ist natürlich nicht notwendig. Aber der Verbesserungsvorschlag hätte ausdiskutiert werden müssen, auf keinen Fall dufte er jedoch eigenmächtig umgesetzt werden. Durch dieses Verhalten der „Hauptgruppe" kam es schließlich dazu, dass am Ende manche Prozeduren gar nicht mehr zum Laufen kamen.

Positiv zu sehen ist, dass das Projekt unter diesen Umständen dennoch mit einem lauffähigen Produkt endete.

2.7 Realisierung

Die Tabelle auf der folgenden Seite stellt einen zeitlichen und inhaltlichen Gesamtüberblick dar. Die Kernphasen der Projektarbeit wurden durch fette Schrift gekennzeichnet.

17.08.99	„Projektarbeit“ allgemein, Einführung, Erwartungen
19.08.99	Projektthemenentwicklung, Brainstorming
24.08.99	Machbarkeitsstudien zu den Projektthemen „Wie könnte so etwas aussehen?“
26.08.99	Vorstellung der Themen, **Themenauswahl, Wahl der Projektleitung** usw.
31.08.99	**Entwicklung einer groben Lösungsstruktur** **„Was können wir? Was müssen wir noch lernen?“**
02.09.99	**Überlegungen zur Datenstruktur** **Überlegungen zur Schnittstellendefinition**
07.09.99	Wiederholung programmiertechnischer Grundlagen (Mousehandling) Einführung: Files, Filehandling
09.09.99	Beispielprogramm zum Thema Filehandling Textfiles, typisierte Files
21.09.99	**Datenstrukturfestlegung (Textfile)** **Schnittstellenfestlegung (Übergange in einem String)**
23.09.99	**Aufteilung in Arbeitsgruppen** **„Forschungsaufgaben“ zur Grafik**
28.09.99	**Arbeitsteilige Realisierung „statischer“ Befehle**
30.09.99	Einführung in „Datenschutz“
19.10.99	**Lösung der Probleme beim Zusammenwirken einiger statischer Befehle (z. B. Linienstil und Liniendicke setzen)**
21.10.99	Forts. Datenschutz
26.10.99	**Grundsätzliche Probleme bei „dynamischen“ Befehlen, Lösung durch Variablen-parameter (z. B. Laufschrift oder Mausabfrage)**
28.10.99	Weitere Lösungsversuche der Probleme bei „dynamischen“ Befehlen („Man weiß nicht, wie viele dynamische Befehle auf einer Seite stehen“)
02.11.99	Einschub: Listen in Pascal Beispielprogramm: verkette Liste
04.11.99	Beispielprogramm: doppelt verkettete Liste
09.11.99	**Entschluss der Projektgruppe, doch nur ein Array von maximal 50 dynamischen Befehlen pro Seite zuzulassen, Realisierung**
11.11.99	Datenschutz in der Schule
16.11.99	**Versuche, arbeitsteilig dynamische Befehle zu realisieren**
18.11.99	Organisatorische Maßnahmen für den Datenschutz
23.11.99	**Realisierung dynamischer Befehle** **Einbau in das Hauptprogramm**
25.11.99	Datenschutzrelevante Beispiele Aufgaben des Datenschutzbeauftragen
30.11.99	Wiederholung für die Kursarbeit
02.12.99	Wiederholung für die Kursarbeit
07.12.99	Kursarbeit
09.12.99	**Erstellung einer Beispieldatei zur Demonstration des Projektes** **Fertigstellung der Dokumentation**
14.12.99	**Abgabe des Projektes** **Vorbereitung der Präsentation des Projektes am Tag der offenen Tür**
16.12.99	**Bewertung des Projektes**

3 Die Projektergebnisse

3.1 Listing

```
Program Interpreter; {Version 1.4}
Uses DOS, CRT, Graph, MouseLib {, PCXUnit};
Var TextDatei:Text;
    Zaehler1,Zaehler2,Zaehler3,Zaehler4,Zaehler5,Zaehler6:Integer;
    Texthoehe,Zeilenabstand,x,y,Clickx,Clicky,MausTaste,j,k,Fehler:Integer;
    W:Word;
    Parameter,Dateiname,Zeile,ProcedureWahl,S,S1:String;
    C:Char;
    Textsettings:Textsettingstype;
    LinkClick:Boolean;
    Dynamisch:Array[0..10,0..5] of String;
    TextArt:Array[0..10] of Textsettingstype;

Procedure BildschirmFaerben(S:String); {Doppelt!!!!!!!!!!!!!}
Begin
  ClearDevice;
  Val(S,j,Fehler);
  Setbkcolor(j);
End;

Procedure Pause(S:String);
Begin
  Val(S,j,Fehler);
  Delay(j);
End;

{ Die Procedure PCX fünktioniert nur notdürftig!
  Procedure PCX(S:String);
  Begin
    j:=1;
    While (S[j] <> ',') and (j<255) Do j:=j+1; {Ende - X - Koordinate}
    Val(Copy(S,1,j-1),x,Fehler);
    Delete(s,1,j);
    j:=1;
    While (S[j] <> ',') and (j<255) Do j:=j+1; {Ende - Y - Koordinate}
    Val(Copy(S,1,j-1),y,Fehler);
    Delete(s,1,j);
    ShowPcxImage(x,y,S);
  End;
}

Procedure TextFarbe(S:String);
Var Fehler,co:Integer;
Begin
  Val(s,co,Fehler);
  if Fehler > 0 then Val(Copy(s,1,Fehler-1),co,Fehler);
  SetColor(co);
End;
```

```
Procedure HintergrundFarbe(S:String);
Var Fehler,co:Integer;
Begin
  Val(s,co,Fehler);
  if Fehler > 0 then Val(Copy(s,1,Fehler-1),co,Fehler);
  Setbkcolor(co);
End;

Procedure Position(S:String);
Var Fehler,i:Integer;
Begin
  i:=pos(',',S);
  if i>0 then begin
    Val(Copy(S,1,i-1),x,Fehler);
    Val(Copy(S,i+1,length(S)-i),y,Fehler);
    Moveto(x,y);
    end
  else outtextxy(x,y,'Fehler im Befehl Position');
End;

Procedure Zeilenachunten;
Begin
  j:=Textsettings.Direction;
  If j =1 then {vertikal}
    y:=y+textwidth(Zeile)
  Else {horizontal}
    y:=y+Textheight(Zeile);
End;

Procedure ZeilenabstandProcedure(S:String);
Var Fehler:Integer;
Begin
  Val(s,Zeilenabstand,Fehler);
  If Fehler>0 then Val(Copy(s,1,Fehler-1),Zeilenabstand,Fehler);
End;

PROCEDURE GROESSE(S:String);
VAR I,Fehler:INTEGER;
BEGIN
  VAL(S,I,Fehler);
  IF Fehler>0 THEN VAL(COPY(S,1,Fehler-1),I,Fehler);
  GETTEXTSETTINGS(Textsettings);
  SETTEXTSTYLE(Textsettings.FONT,Textsettings.DIRECTION,I);
END;

PROCEDURE FONT(S:STRING);
VAR I,Fehler:INTEGER;
BEGIN
  VAL(S,I,Fehler);
  IF Fehler>0 THEN VAL(COPY(S,1,Fehler-1),I,Fehler);
  GETTEXTSETTINGS(Textsettings);
  IF I<=10 THEN SETTEXTSTYLE(I,Textsettings.DIRECTION,Textsettings.CHARSIZE)
  ELSE OUTTEXTXY(x,y,'FEHLER!!! Ungültiger Wert bei "SCHRIFTART"!!!');
END;
```

```
PROCEDURE RICHTUNG(S:STRING);
VAR I,Fehler:INTEGER;
BEGIN
  VAL(S,I,Fehler);
  IF Fehler>0 THEN VAL(COPY(S,1,Fehler-1),I,Fehler);
  GETTEXTSETTINGS(Textsettings);
  IF I < 2 THEN SETTEXTSTYLE(Textsettings.FONT,I,Textsettings.CHARSIZE)
  ELSE OUTTEXTXY(x,y,'FEHLER!!! Ungültiger Wert bei "AUSRICHTUNG"!!!');
END;

Procedure BLINK(S:String);
Var Vordergrund:Integer;
Begin
  Vordergrund:=GetColor;
  Setcolor(Getbkcolor);
  Outtextxy(x,y,S);
  Delay(100);
  Setcolor(Vordergrund);
  Outtextxy(x,y,S);
End;

Procedure Link(S:String);
Var EndeX,EndeY:Integer;
Begin
  Zaehler5:=0;
  j:=1;
  While Dynamisch[Zaehler5,0] <> '' do Zaehler5:=Zaehler5+1; {Suche eine
                                                 freie Position im Array}
  Dynamisch[Zaehler5,0]:='LINK';
  While (S[j] <> ',') and (j<255) Do j:=j+1; {X - Koordinate}
  Val(Copy(S,1,j-1),x,Fehler);
  Dynamisch[Zaehler5,1]:=Copy(S,1,j-1);
  Delete(s,1,j);
  j:=1;
  While (S[j] <> ',') and (j<255) Do j:=j+1; {Y - Koordinate}
  Val(Copy(S,1,j-1),y,Fehler);
  Dynamisch[Zaehler5,2]:=Copy(S,1,j-1);
  Delete(s,1,j);
  j:=1;
  While (S[j] <> ',') and (j<255) Do j:=j+1; {Ende - X - Koordinate}
  Dynamisch[Zaehler5,3]:=Copy(S,1,j-1);
  Val(Dynamisch[Zaehler5,3],EndeX,Fehler); {nur für Kontrolle unten}
  Delete(s,1,j);
  j:=1;
  While (S[j] <> ',') and (j<255) Do j:=j+1; {Ende - Y - Koordinate}
  Dynamisch[Zaehler5,4]:=Copy(S,1,j-1);
  Val(Dynamisch[Zaehler5,4],EndeY,Fehler); {Nur für Kontrolle unten}
  Delete(s,1,j);
  j:=1;
  While (S[j] <> ',') and (j<255) Do j:=j+1; {LinkZiel}
  Dynamisch[Zaehler5,5]:=Copy(S,1,j-1);
  Delete(s,1,j);
  If (S <> '') and (X<EndeX) and (Y<EndeY) then OuttextXY(X,Y,S)
  Else  Begin  {Ein Komma zu wenig oder negatives Feld}
    j:=GetColor;
    SetColor(j+1);
    ClearDevice;
```

```
    OuttextXY(1,1,'[Taste..]Fehler bei der Linkangabe: '+Zeile);
    SetColor(j);
    c:=Readkey;
    c:=#0;
    End;
End;

Procedure Startbildschirm;
Begin
  SetTextStyle(1,0,7);
  SetTextJustify(CenterText,TopText);
  SetColor(4);
  SetBKColor(9);
  OuttextXY(GetMaxX div 2,10,'gtml-Interpreter');
  SetColor(14);
  SetTextStyle(1,0,2);
  OuttextXY(GetMaxX div 2,140,'programmiert von Schülerinnen und Schülern');
  OuttextXY(GetMaxX div 2,180,'des Informatikkurses 12 von Herrn Glöckler
(1999/2000)');
  OuttextXY(GetMaxX div 2,220,'am Gymnasium Mainz-Gonsenheim');
  SetTextStyle(0,0,0);
  SetTextJustify(LeftText, TopText);
  OuttextXY(10,462,'Bitte Taste drücken oder Mausklick');
  Repeat
    LastClickPos(Clickx,Clicky,MausTaste);
  until Keypressed or (MausTaste>=1);
  ClearDevice;
  SetTextStyle(0,0,1);
  SetColor(15);
  SetBKColor(0);
End;

Begin {of main}
  Detectgraph(x,y);
  Initgraph(x,y,'..\BGI'); {in der Schule: '..\BGI'}
  Initmouse;
  Startbildschirm;
  GetTextSettings(TextSettings);
  SettextStyle(Textsettings.Font,Textsettings.Direction,2);
  OuttextXY(1,10,'gtml-Interpreter Version 1.4');
  SettextStyle(Textsettings.Font,Textsettings.Direction,1);
  OuttextXY(1,60,'Bitte Namen der gtml-Datei eingeben:     [ESC=Abbrechen]');
  Repeat
    Setcolor(0);
    OuttextXY(1,80,DateiName);
    Setcolor(15);
    DateiName:='';
    Repeat
      c:=Readkey;
      If (c<>#0) and (c<>#13) and (c<>#8) then Dateiname:=DateiName+c;
      If (c=#8) Then
      Begin
        Setcolor(0);
        Outtextxy(1,80,DateiName);
        Setcolor(15);
        DateiName:=Copy(DateiName,1,length(DateiName)-1);
      End;
      Outtextxy(1,80,DateiName);
    Until (c=#13) or (c=#27);
    If c=#27 then Begin
```

```
    Closegraph;
    Writeln('Das Programm wurde ordnungsgemäß beendet.');
    Halt;
    End;

  Assign(TextDatei,Dateiname);
  GetFAttr(TextDatei,w);
Until w <> 0;
Reset(Textdatei);
Repeat {eine Seite}
  Cleardevice;
  LinkClick:=False;
  Zaehler1:=1; {wofür ?}
  Zaehler4:=-1;
  x:=0;  y:=0;
  For j:=0 to 25 Do Dynamisch[j,0]:='';
  Repeat
    Zaehler2:=0;
    Zaehler3:=0;
    Readln(Textdatei,Zeile);
    If Zeile[1]='<' Then Begin {Die Zeile enthält einen Befehl}
      Zaehler2:=pos(':',Zeile);
      Zaehler3:=pos('>',Zeile);
      if Zaehler2>0 then ProcedureWahl:=Copy(Zeile,2,Zaehler2-2)
      else Procedurewahl:=copy(Zeile,2,Zaehler3-2);
      For Zaehler5:=1 to Length(ProcedureWahl) do
          ProcedureWahl[Zaehler5]:=Upcase(ProcedureWahl[Zaehler5]);
      Parameter:=Copy(Zeile,Zaehler2+1,Zaehler3-Zaehler2-1);
         {Statische Befehle}
      If ProcedureWahl='TEXTFARBE' then TextFarbe(Parameter) Else
      If ProcedureWahl='HINTERGRUNDFARBE' then
         HintergrundFarbe(Parameter) Else
      If ProcedureWahl='GROESSE' then Groesse(Parameter) Else
      If ProcedureWahl='FONT' then Font(Parameter) Else
      If ProcedureWahl='RICHTUNG' then Richtung(Parameter) Else
      If ProcedureWahl='POSITION' then Position(Parameter) Else
      If ProcedureWahl='ZEILENABSTAND' then
         ZeilenabstandProcedure(Parameter) Else
      If ProcedureWahl='LINK' then Link(Parameter) Else
      {If ProcedureWahl='PCX' then PCX(Parameter) Else}
      If ProcedureWahl='PAUSE' then Pause(Parameter) Else
      if ProcedureWahl='LABEL' then Else
      if ProcedureWahl='SEITENENDE' then Else
      Begin {Dynamische Befehle}
        Zaehler4:=Zaehler4+1;
        Dynamisch[Zaehler4,0]:=ProcedureWahl;
        Dynamisch[Zaehler4,1]:=Parameter;
        If ProcedureWahl='BLINK' Then Begin {gehört zu Blink}
          Str(X,S); Dynamisch[Zaehler4,2]:=S;
          Str(Y,S); Dynamisch[Zaehler4,3]:=S;
          j:=GetColor; Str(j,S); Dynamisch[Zaehler4,4]:=S;
          Gettextsettings(Textsettings);
          TextArt[Zaehler4]:=Textsettings;
          End; {gehört zu Blink}
        End; {gehört zu dynamische Befehle}
      End {Ende der Befehle}
    Else {Zeile enthält keinen Befehl}
```

```
    Begin
      Outtextxy(x,y,Zeile);
      Gettextsettings(Textsettings); {Warum hinterher ?}
      Zeilenachunten; y:=y+Zeilenabstand;
    End;
Until (ProcedureWahl='SEITENENDE') or EOF(TextDatei);
ShowMouse;

Repeat {Verarbeitung der dynamischen Befehle}
  Zaehler4:=-1;      {Setze Listenindex zurück}
  linkclick:=false;  {Es ist noch nicht geklickt}
  LastClickPos(Clickx,Clicky,MausTaste);
  Repeat  {Durchläuft die Liste der dynamischen Befehle}
    zaehler4:=zaehler4+1;
    If (MausTaste=1) and (Dynamisch[Zaehler4,0]='LINK') Then Begin
      Val(Dynamisch[Zaehler4,1],J,Fehler);
      Val(Dynamisch[Zaehler4,3],K,Fehler);
      If (Clickx >= j) and (Clickx <= k) then Begin
        Val(Dynamisch[Zaehler4,2],J,Fehler);
        Val(Dynamisch[Zaehler4,4],K,Fehler);
        If (Clicky >= j) and (Clicky <= k) Then LinkClick:=True;
        End;
      End;
    If (Dynamisch[Zaehler4,0]='BLINK') and (MausTaste=0) then Begin
      Position(Dynamisch[Zaehler4,2]+','+Dynamisch[Zaehler4,3]);
      With TextArt[Zaehler4] do SettextStyle(Font, Direction, CharSize);
      Val(Dynamisch[Zaehler4,4],j,Fehler); SetColor(j);
      Blink(Dynamisch[Zaehler4,1]);
      {Delay(100); Wozu ?}
      End;
    c:=#0;
    If keypressed then c:=Readkey;

  until (dynamisch[zaehler4,0]='') or (c=#27) or linkClick;
Until (C=#27) or LinkClick; {Ende der dynamischen Befehle}

If LinkClick Then Begin  {War ein Click auf einen Link dabei?}
  HideMouse;
  Reset(TextDatei);
  Repeat
    Readln(TextDatei,Zeile);
    Zaehler2:=0;
    Zaehler3:=0;
    If Zeile[1]='<' Then Begin
      Zaehler2:=pos(':',Zeile);
      Zaehler3:=pos('>',Zeile);
      if Zaehler2>0 then ProcedureWahl:=Copy(Zeile,2,Zaehler2-2)
      else Procedurewahl:=Copy(Zeile,2,Zaehler3-2);
      For Zaehler5:=1 to Length(ProcedureWahl) do
          ProcedureWahl[Zaehler5]:=Upcase(ProcedureWahl[Zaehler5]);
      Parameter:=Copy(Zeile,Zaehler2+1,Zaehler3-Zaehler2-1);
      End;
    If Eof(TextDatei) then Begin
      If (Dynamisch[Zaehler4,5]<>'Programmende') then Begin
        Cleardevice;
        Position(Dynamisch[Zaehler4,1]+','+Dynamisch[Zaehler4,2]);
        Outtext('Sprungziel "'+Dynamisch[Zaehler4,5]+
                '" nicht gefunden !! [Taste ...]');
        c:=ReadKey;
        c:=#9999;
```

```
            Reset(TextDatei);
            End
          Else Begin
            RestoreCrtMode;
            Close(TextDatei);
            Writeln('Das Programm wurde ordnungsgemäß beendet.');
            Halt;
            End;
          End;
      Until (Parameter=Dynamisch[Zaehler4,5]) and (ProcedureWahl='LABEL')
             or (c=#9999);
    End;
  Until (C=#27);
  RestoreCrtMode;
  Close(TextDatei);
  Writeln('Das Programm wurde ordnungsgemäß beendet.');
End. {of main}
```

3.2 Dokumentation

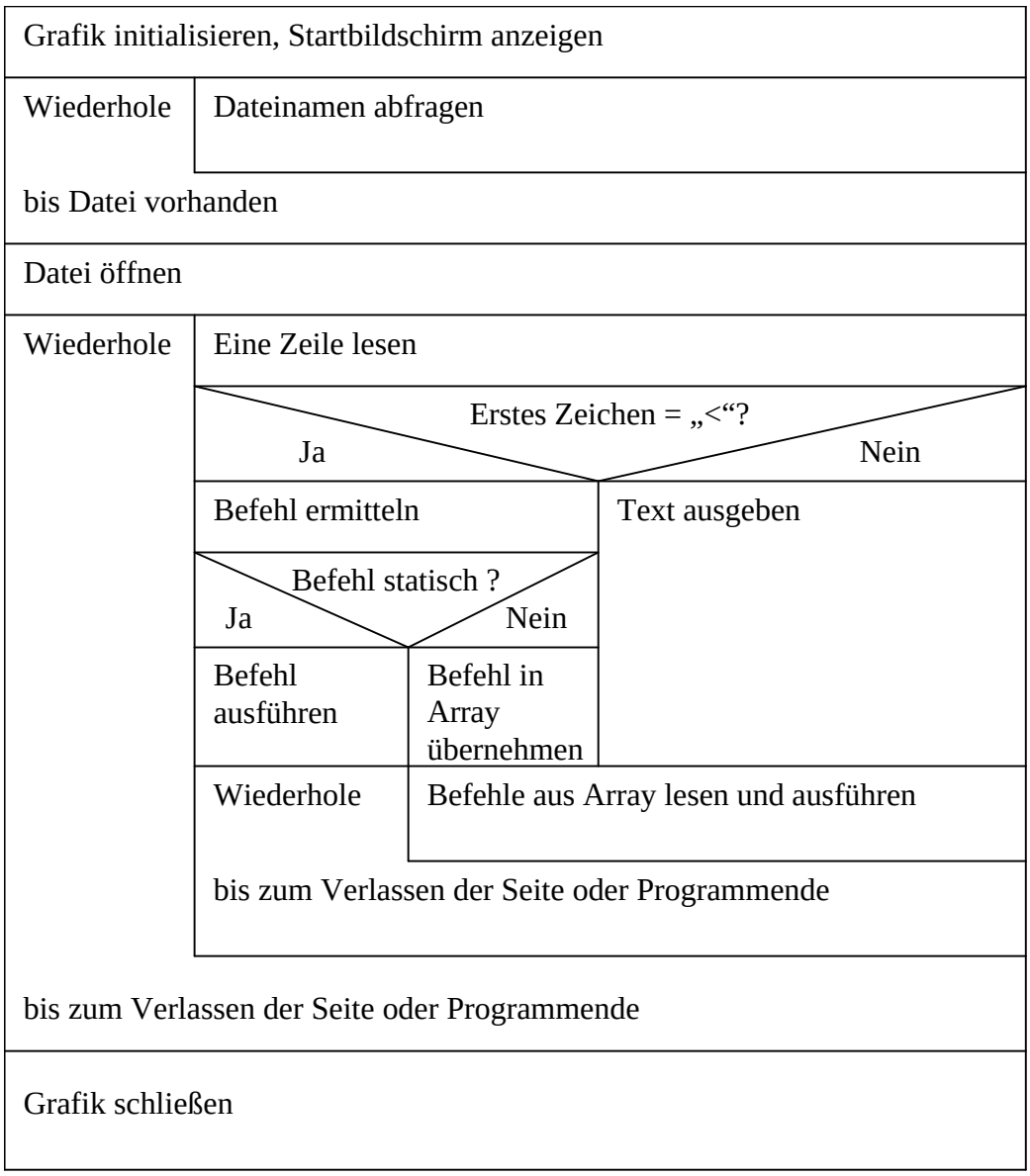
Der folgende Text ist die Schülerfassung der Projektdokumentation.

Aufgabenstellung

Das Ziel war es, ein Programm zu schreiben, welches, analog zu Internetbrowsern, Dateien interpretiert, die HTML-ähnliche Befehle enthalten. Dabei sollten nicht nur statische, sondern auch dynamische Befehle ausgeführt werden können. Als Programmiersprache sollte Pascal verwendet werden.

Arbeitsweise des Programms

Struktogramm:



Beschreibung:

Als aller erstes initialisiert das Programm den Grafikmodus, später die Maus. Nachdem der Startbildschirm angezeigt wurde, wird vom Benutzer der Name und die Position der zu interpretierenden Datei erfragt, bis der Benutzer eine existierende Datei angibt. Diese wird dann geöffnet. Nach diesem Prozess beginnt die eigentliche Arbeit des Hauptprogramms. Dessen grundsätzliche Aufgabe ist es, die GTML-Datei zeilenweise bis zum Befehl "<*Seitenende*>" oder bis zum Ende der Datei, je nachdem, was zuerst kommt, auszulesen. Bei jeder Zeile wird entschieden, ob es sich um einen Befehl oder um auszugebenden Text handelt. (Das Programm erkennt einen Befehl daran, dass die Zeile mit einem "<" beginnt.) Je nachdem, wie diese Entscheidung ausfällt, gibt es zwei mögliche Vorgehensweisen:

1. Steht in der Zeile ein Befehl, wird dieser in den Befehlnamen (bis zum ":") und den so genannten Parameter (alles zwischen ":" und ">") unterteilt. Nun wird zwischen statischen und dynamischen Befehlen unterschieden. Bei einem statischen Befehl wird anhand des Befehlnamens die entsprechende Prozedur ermittelt, diese aufgerufen und ihr der Parameter übergeben. Dynamische Prozeduren werden in einem Array gespeichert, in dem Befehlname, Parameter und die aktuellen Formatierungsattribute stehen.
2. Steht in der Zeile nur Text, wird dieser entsprechend der aktuellen Formatierungsattribute am Bildschirm ausgegeben.

Wurde die Datei bis zum Befehl "<*Seitenende*>" oder bis zu ihrem Ende ausgelesen, werden alle dynamischen Prozeduren, die in dem oben erwähnten Array stehen, aufgerufen, bis entweder die aktuelle Seite verlassen oder das Programm beendet wird. Eine dieser Prozeduren ist zum Beispiel die Mausabfrage, die überprüft, ob der Benutzer eine Maustaste drückt, um z. B. auf einen Hyperlink zu klicken.

Verlässt der Benutzer das Programm, wird die Datei, die interpretiert wurde, geschlossen und der Textmodus wieder hergestellt.

Kurze Beschreibung der einzelnen Befehle

Jede Befehlszeile muss mit einem "<" begonnen und mit einem ">" beendet werden. Andernfalls wird sie als Text interpretiert und am Bildschirm ausgegeben.

1. <*GROESSE:Wert*>
Hiermit wird eine Textgröße festgelegt. Je höher der Wert, desto größer der auf dem Bildschirm ausgegebene Text. Wird ein neuer Wert festgelegt, ist der alte für alle Zeilen darunter nicht mehr gültig. Dies gilt allgemein für alle Befehle dieser Art.

2. <*FONT:Wert*>
Hiermit wird eine Schriftart festgelegt.

3. <*AUSRICHTUNG:Wert*>
Mit diesem Befehl wird die Ausrichtung des Textes festgelegt. 0 bedeutet horizontal, 1 vertikal.

4. <*BLINK:TEXT*>
Mit diesem Befehl kann ein blinkender Text erzeugt werden. Man gibt nach dem Doppelpunkt einfach den Text an, der blinken soll.

5. *<PAUSE:Wert>*
Hiermit wird eine Pause beim Interpretieren des GTML-Dokuments gemacht. Die Dauer wird in Millisekunden angegeben. Dieser Befehl wird z. B. bei sog. Slideshows benutzt, um Bilder nacheinander anzuzeigen.

6. *<TEXTFARBE:Farbwert>*
Mit diesem Befehl wird die Textfarbe festgelegt. Der Farbwert kann von 0 (schwarz) bis 15 (weiß) angegeben werden.

7. *<HINTERGRUNDFARBE:Farbwert>*
Mit diesem Befehl wird die Hintergrundfarbe festgelegt. Der Farbwert kann wieder von 0 (schwarz) bis 15 (weiß) angegeben werden.

8. *<BILDSCHIRMFAERBEN:Farbwert>*
Durch diesen Befehl wird der komplette Bildschirm mit der angegebenen Farbe eingefärbt.

9. *<POSITION:x,y>*
Damit wird der Cursor auf die angegebene Position gesetzt. x und y werden in Pixeln angegeben.

10. *<ZEILENABSTAND:Wert>*
Der Zeilenabstand wird in Pixeln angegeben.

11. *<LABEL:ZIEL>*
Hiermit wird ein Sprungziel festgelegt, welches dann mit dem LINK-Befehl angesprungen werden kann.

12. *<LINK:vonx,vony,bisx,bisy,Ziel,Text>*
Mit diesem Befehl wird ein Link festgelegt. Man gibt mit "vonx", "vony", "bisx", "bisy" einen Bereich an, in dem der Linktext angezeigt wird. Klickt man in den Bereich, wird das Ziel angesprungen (durch *<LABEL:Ziel>* festgelegt).

13. *<PCX:x,y,Bild>(funktioniert nicht wie gewünscht)*
Hiermit kann ein PCX-Bild mit 16 Farben angezeigt werden. Man gibt dazu die Position in Pixeln (x,y) an und dann den Pfad des anzuzeigenden Bildes.

14. *<SEITENENDE>*
Dieser Befehl gibt das Ende einer Seite an.

Hinweis: Bei den Befehlen ist die Groß- und Kleinschreibung egal, da das Programm die Befehlsnamen sowieso in Großbuchstaben umwandelt.

Probleme, die bei der Programmierung auftraten

Eins der größten Probleme war es, die dynamischen Elemente so zu implementieren, dass sie kontinuierlich und nicht nur einmal ablaufen. Besonders schwierig war dies bei der Mausabfrage in Verbindung mit den Hyperlinks.

Ein weiteres Problem war die Koordination mit den anderen Gruppen. Bis zum Schluss wusste kaum jemand, was vom Hauptprogramm an die einzelnen Prozeduren übergeben wird und so waren wir immer wieder gezwungen, Prozeduren zu korrigieren, komplett umzuschreiben oder sogar neu zu schreiben. (Eine Prozedur, die für den Textmodus programmiert wurde, wird z. B. nicht im Grafikmodus laufen. Und Prozeduren, die von vielen global definierten Variablen ausgehen sorgen auch immer wieder für Probleme.)

Funktionen, die noch nicht implementiert wurden und Bugs des Programms

1.
Die Laufschrift-Prozedur konnte noch nicht eingebaut werden, da die Programmierer davon ausgingen, dass das Hauptprogramm zwei Variablen (Laufrichtung und Text) und nicht nur den Text übergibt. Die Prozedur an sich funktioniert jedoch.

2.
Die PCX-Prozedur nicht voll funktionstüchtig. Es ist zwar möglich, Bilder im PCX-Format darzustellen, jedoch nur mit 16 Farben. Außerdem sind die Farben verfälscht und die Bilder manchmal verzerrt.

Anhang

GTML Beipielseiten

<Label:Index>
<TextFarbe:14>
<Hintergrundfarbe:7>
<Font:1>
<Groesse:8>
<Position:180,0>
Index
<Groesse:3>
<Position:180,70>

<Font:1>
<Groesse:2>
<TextFarbe:9>
<Link:50,180,500,220,Demo,zur Demonstration der Befehle>
<Position:50,187>

<Link:50,220,500,260,Syntax,zur Syntax und Befehlsliste>
<Position:50,227>

<Link:50,260,500,300,Beschreibung1,zur Beschreibung des Programms>
<Position:50,267>

<Link:50,300,500,400,Programmende,Programm beenden>
<Position:50,307>

<Seitenende>
<Label:Syntax>
<Hintergrundfarbe:7>
<Textfarbe:11>
<Font:1>
<Groesse:10>
<Position:180,0>
Syntax
<Textfarbe:15>
<Font:0>
<Groesse:0>
<Position:10,150>
<Zeilenabstand:2>
Befehle beginnen in der ersten Spalte mit <, sie werden mit > beendet.
Parameter zu den Befehlen folgen auf das Schlüsselwort nach einem :
Mehrere Parameter werden durch Komma getrennt.
Befehle müssen allein in einer Zeile stehen. Zeichen nach > werden
nicht beachtet.
Zeilen, die nicht mit < beginnen, enthalten Text, der auf dem Bild-

schirm dargestellt wird. Der Zeilenumbruch muss selbst beachtet werden.
Folgende Befehle sind implementiert:

Textfarbe:0..15	kontrolliert die Textfarbe für nachfolgende Texte
Hintergrundfarbe:0..7	setzt die Hintergrundfarbe
Groesse:0..10	setzt die Textgröße
Font:0..4	setzt die Schriftart
Richtung:0..1	Schriftrichtung (0:waagrecht, 1:senkrecht)
Position:x,y	legt die Position der nächsten Ausgabe fest
Zeilenabstand:y	legt den Zeilenabstand fest
Link:x1,y1,x2,y2,Label,Text	legt eine Verzweigung mit sensitivem Text fest
PCX:Dateiname	erlaubt das Laden einer PCX-Datei (16 Farben)
Pause:x	legt eine Pause fest
Label:Name	bestimmt ein Sprungziel
Seitenende	sagt, dass eine Seite zu Ende ist
Blink:Text	lässt den Text blinken

<Link:10,440,500,459,Index,zurück zum Index>
<Seitenende>
<Label:Demo>
<Zeilenabstand:10>
<TextFarbe:14>
<Hintergrundfarbe:0>
<Font:1>
<Groesse:5>
<Position:250,0>
Seite 1
<Font:1>
<Groesse:1>
<Zeilenabstand:50>
<Position:20,75>
<TextFarbe:9>
Mit dem gtml-Interpreter kann man
<TextFarbe:14>
<Position:40,120>
1. die Textfarbe,
<Zeilenabstand:0>
<Groesse:2>
<TextFarbe:3>
<Position:40,160>
2. die Textgröße,
<Groesse:1>
<Font:5>
<Position:40,200>
<TextFarbe:7>
3. die Schriftart,
<Font:1>
<Position:40,220>
<Richtung:1>
<TextFarbe:15>
4. die Textrichtung

<Richtung:0>
<Position:350,150>
<TextFarbe:18>
5. und die Textposition ändern.
<TextFarbe:5>
<Position:150,300>
<TextFarbe:13>
<Font:8>
<Groesse:2>
<Blink:Außerdem kann man Text blinken lassen.>
<Font:1>
<Groesse:1>
<TextFarbe:9>
<Link:10,450,450,470,Index,zurück zum Index>
<Position:10,457>

<Seitenende>
<Label:Beschreibung1>
<HintergrundFarbe:1>
<Font:1>
<Groesse:3>
<Position:40,2>
<TextFarbe:11>
<Zeilenabstand:0>
Über die grundsätzliche Funktionsweise des
<Position:40,16>
<TextFarbe:14>

<Position:190,35>
<TextFarbe:10>
GTML
<TextFarbe:11>
<Position:259,35>
-Interpreters
<TextFarbe:14>
<Position:190,49>

<TextFarbe:7>
<Position:0,76>
<Font:6>
<Groesse:2>
Der
<TextFarbe:10>
<Link:35,76,90,180,DefinitionGTML,GTML>
<TextFarbe:15>
<Position:32,83>

<TextFarbe:7>
<Position:80,76>

-Interpreter stellt Texte graphisch am Bildschirm dar. Er ist in der
<Position:0,98>
Lage, Anweisungen zwischen darzustellenden Texten zu verarbeiten und diese entsprechend umzusetzen.
Die Arbeitsweise des Interpreters ist stark an die eines Browsers wie etwa Mircosoft Internet Explorer oder des Netscape Navigator angelehnt.
Es gibt Befehle aus den Bereichen der Textgestaltung so wie der Präsentation. Diese Texte können auch mit verschiedenen Grafiken zur Veranschaulichung ansehnlicher gestaltet werden.
Der GTML-Interpreter erlaubt zudem das Erstellen mehrer unabhäniger Seiten, die man vergleichbar mit HTML, problemlos verknüpfen kann. Zudem eignet sich das Programm zum Präsentieren von beispielsweise Ergebnissen einer Projektarbeit.
<Font:2>
<Groesse:6>
<TextFarbe:15>
<Link:10,430,590,449,Beschreibung2,hier klicken, um Beispiele zu erhalten.>
<Position:10,438>

<TextFarbe:15>
<Link:10,450,590,480,Index,zurück zum Index>
<Position:10,458>

<SeitenEnde>
<Label:Beschreibung2>
<TextFarbe:15><Font:5>
<Groesse:4>
<Position:150,2>
<TextFarbe:10>
Präzisere Beschreibung
<Position:150,20>
<TextFarbe:3>

<Font:6>
<Groesse:2>
<Position:0,70>
<TextFarbe:15>
Der GTML-Interpreter liest seine Befehle sowie den darzustellenden Text aus einer Textdatei ein. Dabei werden zunächst alle Befehle und Texte einer Seite eingelesen und verarbeitet.
Aktiviert der Benutzer eine Verknüpfung (unten am Bildschirmrand zu sehen), so sucht das Programm nach dem Sprungziel. Anschließend verarbeitet der Interpreter alle dort befindlichen Elemente bis zum jeweiligen Seitenende.
<Position:0,255>
Hier einige Beispiele für mögliche Befehle :
<Position:0,265>

<Position:30,285>

<TextFarbe:7>
Ein Text nach dem Befehl "<Font : 4>" steht sieht so aus:
<Font:4>
<TextFarbe:15>
<Position:36,315>
Dies ist der Text mit dem Font 4
<Position:30,355>
<Font:6>
<TextFarbe:7>
Folgt der Befehl "<Blink : Dies ist der Text mit dem Font 4>"
so sieht dies folgendermaßen aus:
<Position:36,408>
<Font:4>
<TextFarbe:15>
<Blink:Dies ist der Text mit dem Font Nummer 4>
<Font:2>
<Groesse:6>
<TextFarbe:15>
<Link:10,450,590,480,Beschreibung1,Zurück zur grundsätzlichen Funktionsweise>
<Position:10,458>

<SeitenEnde>
<Label:DefinitionGTML>
<Position:230,20>
<Font:7>
<TextFarbe:14>
GTML
<Textfarbe:15>
<Font:1>
<Position:0,130>
<Groesse:2>
GTML ist eine Abwandlung von HTML.
<Position:0,180>
Die Bezeichnung GTML ist vom Informatikkurs der Stufe 12
erfunden. Sie bedeutet
<Textfarbe:10>
<Position:300,200>
G
<Textfarbe:15>
<Position:315,200>
ygo
<Textfarbe:11>
<Position:350,200>
T
<Textfarbe:15>
<Position:365,200>
ext
<Textfarbe:12>
<Position:400,200>

M
<Textfarbe:15>
<Position:415,200>
arkup
<Textfarbe:13>
<Position:480,200>
L
<Textfarbe:15>
<Position:495,200>
anguage.
<Position:0,270>
HTML steht eigentlich für
<TextFarbe:10>
<Position:273,270>
H
<Position:290,270>
<Textfarbe:15>
yper
<Position:336,270>
<Textfarbe:14>
T
<Position:348,270>
<Textfarbe:15>
ext-
<Position:393,270>
<Textfarbe:7>
M
<Position:410,270>
<Textfarbe:15>
arkup-
<Position:484,270>
<Textfarbe:4>
L
<Position:498,270>
<Textfarbe:15>
anguage.
<Position:0,320>
Das ist eine Formatierungs- und Strukturierungssprache in
<Position:0,350>
der die Textseiten des World Wide Web angelegt sind.
<TextFarbe:15>
<Font:2>
<Groesse:6>
<Link:10,450,590,480,Beschreibung1,Zurück zur grundsätzlichen Funktionsweise>
<Position:10,458>

<SeitenEnde>
<Programmende>

Unit Mouselib

Die Unit Mouselib wurde nicht im Rahmen dieses Projektes, sondern bereits am Ende von 11.2 in einem „Vorprojekt“ erstellt.

```
UNIT MOUSELIB;

INTERFACE
USES DOS,CRT;
TYPE MZT=ARRAY[0..31] OF WORD;
 CONST MZ:MZT=(
       $F00F,$E007,$C3C3,$78E1,$03C0,$1188,$3813,$3C3C,
       $3C3C,$3813,$1180,$03C0,$78E1,$C3C3,$E007,$F00F,
       $03C0,$0C30,$1008,$300C,$4812,$4422,$8241,$8181,
       $8181,$8241,$4422,$4812,$300C,$1008,$0C30,$03C0);

VAR REG:REGISTERS;

PROCEDURE INITMOUSE;
PROCEDURE SHOWMOUSE;
PROCEDURE HIDEMOUSE;
PROCEDURE MOUSESTATUS(VAR x,y,z:INTEGER);
PROCEDURE SETMOUSEPOS(VAR x,y:INTEGER);
PROCEDURE LASTCLICKPOS(VAR x,y,z:INTEGER);
PROCEDURE LASTRELEASEPOS(VAR x,y:INTEGER);
PROCEDURE MOUSECOLUMN(VAR x,y:INTEGER);
PROCEDURE MOUSELINE(VAR x,y:INTEGER);
PROCEDURE SETMOUSECURSOR(VAR hotx,hoty:INTEGER;MZ:Mzt);


IMPLEMENTATION

PROCEDURE INITMOUSE;
BEGIN
  REG.AX:=0;
  INTR($33,REG);
END;

PROCEDURE SHOWMOUSE;
BEGIN
  REG.AX:=1;
  INTR($33,REG);
END;

PROCEDURE HIDEMOUSE;
BEGIN
  REG.AX:=2;
  INTR($33,REG);
END;

PROCEDURE MOUSESTATUS(VAR x,y,z:INTEGER);
BEGIN
  REG.AX:=3;
  INTR($33,REG);
  z:=REG.BX;
  x:=REG.CX;
  y:=REG.DX;
END;
```

```
PROCEDURE SETMOUSEPOS(VAR x,y:INTEGER);
BEGIN
  REG.AX:=4;
  REG.CX:=x;
  REG.DX:=y;
  INTR($33,REG);
END;

PROCEDURE LASTCLICKPOS(VAR x,y,z:INTEGER);
BEGIN
  REG.AX:=5;
  INTR($33,REG);
  z:=REG.BX;
  x:=REG.CX;
  y:=REG.DX;
END;

PROCEDURE LASTRELEASEPOS(VAR x,y:INTEGER);
BEGIN
  REG.AX:=6;
  INTR($33,REG);
  x:=REG.CX;
  y:=REG.DX;
END;

PROCEDURE MOUSECOLUMN(VAR x,y:INTEGER);
BEGIN
  REG.AX:=7;
  INTR($33,REG);
  x:=REG.CX;
  y:=REG.DX;
END;

PROCEDURE MOUSELINE(VAR x,y:INTEGER);
BEGIN
  REG.AX:=8;
  INTR($33,REG);
  x:=REG.CX;
  y:=REG.DX;
END;

PROCEDURE SETMOUSECURSOR(VAR hotx,hoty:INTEGER;MZ:Mzt);
BEGIN
  REG.AX:=9;
  REG.BX:=hotx;
  REG.CX:=hoty;
  REG.DX:=ofs(MZ);
  REG.ES:=seg(MZ);
  INTR($33,REG);
END;

END.
```