Hausarbeiten

zu Informatik 2 mit MATLAB

Prof. Dr. Ulrich Stein
Dep. Maschinenbau und Produktion
Fakultät Technik und Informatik
HAW Hamburg

Inhalt

1. Prüfungsleistung

1.1 Anforderungen

Hausarbeiten werden **nur** an **Mitglieder von Laborgruppen** dieser Semestergruppe vergeben. Regelmäßige **Teilnahme** an den **Labor-Terminen** dieser Semestergruppe ist Voraussetzung für die Bewertung der Hausarbeit. Der Eintrag in Laborgruppen erfolgt vor Beginn der Vorlesungszeit. Eine nachträgliche Buchung ist nicht möglich. Voraussetzung für die Teilnahme an Informatik2-Laboren ist eine **bestandene Informatik1-Klausur**. Aus Kapazitäts-Gründen kann während des Studiums ein Informatik2-Labor **nur einmal** besucht werden.

Eine **Laborgruppe** (16–20 Studenten) bearbeitet gemeinsam ein **Hauptprojekt**, das vom Betreuer vorgegeben wird. Die Studenten einer Laborgruppe arbeiten (normalerweise) **zu zweit** in **Arbeitsgruppen** an einem **Teilprojekt**. Aus diesen Teilprojekten wird am Ende das Hauptprojekt zusammengebaut. Hierfür gibt es vom Betreuer vordefinierte Schnittstellen (z.B. Funktions-Signaturen), die ein gemeinsames Arbeiten am Projekt ermöglichen (Concurrent Engineering).

Aufgaben einer Arbeitsgruppe:

- Entwicklung eines **Konzeptes** zur Lösung des vorgegebenen Teilproblems.
- Programmierung der benötigten MATLAB-**Funktionen**.
- Inline-System-**Dokumentation** der Funktionen.
- Entwicklung eines **Test-Szenarios** für die geschriebenen Funktionen.
- Dokumentation des **Tests** der geschriebenen Funktionen. Analyse der aufgetretenen Fehler, Dokumentation der Fehlersuche und Lösungsstrategien
- Dokumentation des **Einbaus** der Funktionen in das Hauptprojekt.
- Ausführliche **Projektbeschreibung, 10-20 Seiten**,
  wobei **jeder Teilnehmer** der Arbeitsgruppe für ein **Teilgebiet** verantwortlich ist,
  das er innerhalb der Projektbeschreibung mit ca. 4 Seiten **dokumentiert**.
- **evtl. mündliche Präsentation** des Teilprojektes (5-10 Minuten) durch **Teilnehmer**

1.2 Abgabe

Ihr Projekt muss bereits während des laufenden Semesters so weit kommen, dass ein erster Einbau Ihrer Funktionen in das Hauptprojekt zustande kommt. Spätester Abgabetermin für die endgültige, ausgiebig getestete Version, die Dokumentationen und die Projektbeschreibung ist der Beginn des Folge-Semesters.

1.3 Benotung

Benotet werden primär folgende Punkte:

- einwandfreie Funktionalität (natürlich mit Hilfestellung durch den Betreuer)
- Projektbeschreibung, speziell auch der eigene Beitrag dazu (ca. 4 Seiten)
- Test-Szenario und Test-Dokumentation
- System-Dokumentation der eigenen Funktionen, evtl. mündliche Präsentation
- Originalität, Ideen zur Erweiterung der Funktionalität

1.4 Aufbau der Projektbeschreibung

Die Projektbeschreibung dokumentiert, in wie weit Sie verstanden haben, was Ihre Aufgabe innerhalb des Hauptprojektes ist, und wie Sie diese Aufgabe gelöst haben.

Ausgangpunkt Ihrer Arbeit ist ein Startprojekt, in dem bereits gewisse Schnittstellen (z.B. Funktions-Signaturen) vordefiniert waren, um das gemeinsames Arbeiten am Hauptprojekt zu ermöglichen (Concurrent Engineering).

Unterschiedlich, je nach Arbeitsgruppe, waren im Startprojekt auch schon einige Funktionen mehr oder weniger weit fertig ausprogrammiert – speziell solche, von denen ich annahm, dass sie für ein zweites Semester zu schwierig sind. Auch diese Funktionen gehören zu Ihrem Teilprojekt und sollten deshalb in der Projektbeschreibung dokumentiert werden. Und auch diese Funktionen sollten Sie testen!

Typischerweise könnte eine Projektbeschreibung wie folgt aussehen:

- wer ist in dieser Gruppe, Datum dieser Version der Beschreibung
- einleitende Beschreibung des **Hauptprojektes**, mit Einordnung Ihrer **Teilaufgabe**
- **Ausgangspunkt** Ihres Teilprojektes: Schnittstellen im Startprojekt, welche Funktionen sollen von Ihnen bearbeitet werden, was war bereits ausprogrammiert, etc.
- Beschreibung der zu erledigenden **Aufgaben**
- **Konzept** zur Lösung der Aufgaben
- wie wurden die Aufgaben auf die (2-3) Mitglieder der Teilgruppe **aufgeteilt**, wer ist für welchen Teil **verantwortlich** und kann dazu befragt werden – wenn Sie keine Aufteilung vornehmen, bedeutet das, dass jeder in der Gruppe über **alles** Bescheid weiß!
- Dokumentation der **Programmierung**, speziell Ihre Änderungen am Startprojekt: welche **weiteren Funktionen und Dateien** haben Sie eingefügt – den **inline dokumentierten Programm-Code** aller Ihrer Funktionen beilegen
- **Dokumentation** des Programm-Codes, der im Startprojekt bereits für Ihr Teilprojekt **ausprogrammiert** war (die Dokumentationen aller Gruppen zusammen sollen das Hauptprojekt vollständig beschreiben) – Sie sollen aber nicht die Aufgaben der anderen Teilgruppen beschreiben, sondern nur den Block, der für Sie wichtig ist!
- Beschreibung von **Problemen**, z.B. Kollisionen mit den Vorgaben durch das Hauptprojekt, Erweiterungen der Datenbasis, Kooperation mit anderen Teilprojekten
- Dokumentation von Änderungen am Startprojekt bei **Funktionen**, die eigentlich für eine **andere Gruppe** reserviert sind - den inline dokumentierten Programm-Code der geänderten Funktionen beilegen, Änderungen besonders **markieren**
- **Screen-Shots** beilegen, falls Sie die Bediener-Oberfläche verändert haben
- von Ihnen erzeugte, **weitere Dateien** belegen, z.B. CAD-Zeichnungen, Musik-Dateien, Bild-Dateien, Musik-Klänge – Binär-Dateien als E-Mail schicken
- Dokumentation Ihres Test-Szenarios und Ihrer **Tests**
- Ideen - wie könnte man das Teil- oder auch das Hauptprojekt noch erweitern, was fehlt, gibt es prinzipielle Probleme durch die Projekt-Definition, ...
- Ihre persönliche Stellungnahme: war die Vorlesung ausreichend, um dieses Projekt anzugehen, fehlte Ihnen etwas Wichtiges an Information, war die Aufgabe zu schwierig, zu einfach, langweilig, interesant, ...

2. Projektstruktur

2.1 Hauptprojekte

Jede Laborgruppe (16-20 Studenten) bearbeitet gemeinsam ein Hauptprojekt. Um ein gemeinsames Arbeiten am Projekt ermöglichen, wird jedem Projekt ein **Prefix** zugewiesen. Die Namen der MATLAB-Funktionen eines Projekts müssen mit diesem Prefix beginnen, um Kollisionen durch identisch vergebene Funktionsnamen zu verhindern.

Folgende Hauptprojekte sind für das SS 2006 vorgegeben:

- Einfaches 2D-**CAD-System** Prefix: c06 Laborgruppe: A
- **Bildverarbeitung** Prefix: b06 Laborgruppe: B
- **Musik** Prefix: m06 Laborgruppe: C
- **Strategiespiel** (Sokoban) Prefix: s06 Laborgruppe: D
- **Spiel: Tic-Tac-Toe** Prefix: t06 Laborgruppe: E

Für jedes Hauptprojekt wird eine Start-Umgebung bereitgestellt, die als Ausgangspunkt für erste Tests dienen soll.

2.2 MATLAB-Funktionen

Eine der Hauptaufgaben der Arbeitsgruppen ist die Programmierung von MATLAB-Funktionen. Folgendes ist dabei zu berücksichtigen:

- Die Namen der MATLAB-Funktionen beginnen mit der Prefix des Hauptprojekts. Darauf kann ein **weiterer Buchstabe** für die Arbeitsgruppe folgen, z.B. "d".
- Danach folgt die eigentliche **Bezeichnung** der Funktion, z.B. die Funktion "start" in der Arbeitsgruppe "d" im CAD-System hat so den Funktionsnamen "c06d_start".
- Die **Signatur** von Schnittstellen-**Funktionen** (= Funktionen die auch von anderen Arbeitsgruppen des Projekts verwendet werden) wird in der Konzeptphase festgelegt und dokumentiert. Diese Signatur darf später nur in Absprache mit allen am Projekt Beteiligten abgeändert bzw. erweitert werden.
- Der Name des zugehörigen M-Files ist identisch mit dem Namen der MATLAB-Schnittstellen-Funktion.
- Funktionen sollen möglichst **kurz** und übersichtlich gehalten werden, normalerweise nicht länger als eine Seite Quelltext. Lieber eine weitere Unterfunktion schreiben als "Spaghetti-Code" erzeugen.
- Unterfunktionen, die nur in einer einzigen Schnittstellen-Funktion aufgerufen werden, können auch als **private Funktionen** im selben M-File stehen wie die Schnittstellen-Funktion selbst.
- Neben den Schnittstellen-Funktionen kann es sinnvoll sein, sich eine gewisse Menge von **Hilfs-Funktionen** zu schreiben, falls deren Funktionalität in mehreren Schnittstellen-Funktionen benötigt wird, z.B. Tests ob eine eingegebene Zahl in einem bestimmten Intervall liegt.
- Es empfiehlt sich, die Funktionen "**inline**" zu **dokumentieren**, d.h. man erklärt den Ablauf der Funktion als Kommentar im Programm-Code.

2.3 Datenbasis

Alle MATLAB-Funktionen eines Hauptprojekts arbeiten auf einer gemeinsamen Datenbasis. Zur Laufzeit des Programm wird dafür ein Speicherbereich reserviert, der durch einen eindeutigen Namen angesprochen wird. Der Name für diese **globale Variable** beginnt mit dem Projekt-Prefix gefolgt von der Bezeichnung "**data**", also z.B. "*c06_data*" für die Daten des CAD-Systems.

Diese Variable, ein MATLAB-struct, wird in der Initialisierungs-Phase des Programms von der Start-Funktion als globale Variable angelegt - vgl. das **Labyrinth-Beispiel** aus der Vorlesung. Dieser struct besteht aus mindestens zwei Komponenten:

- allgemeine Einstellungen, z.B. maximale Modellgröße, Genauigkeiten, etc.
- Array mit den Elementen, die vom Programm bearbeitet werden

Ein Element im CAD-System wäre z.B. ein struct mit den Daten einer 2D-Linie. Außer der reinen Geometrie würde hierbei noch weitere Optionen geführt, z.B. Linienfarbe, Strichart.

Außer dem Hinzufügen von weiteren Elementen zur Datenbasis muss auch das **Löschen** von Elementen verwaltet werden. Am einfachsten kann man das Löschen dadurch erledigen, dass man im Daten-struct zu einem Element ein Flag mitführt, das angibt, ob das Element gelöscht wurde. Beim Hinzufügen weiterer Elemente kann man diesen Daten-struct dann mit neuen Daten überschreiben.

Auch das Löschen aller Elemente ist Aufgabe der Datenbasis, z.B. bevor eine neue Datenbasis aus einer Textdatei geladen wird.

Für jedes Hauptprojekt wird eine Start-Datenbasis bereitgestellt, damit die anderen Arbeitsgruppen ihre Funktionen testen können.

Die Datenbasis ist erfahrungsgemäß ein Teil des Projekts, der bis zum Ende noch (leichte) Änderungen berücksichtigen muss.

2.4 GUI

Pro Hauptprojekt erstellt eine Arbeitsgruppe die grafische Oberfläche des Programms. Hierbei fallen folgende Aufgaben an:

- Konzeptionierung des Layouts, speziell für Ausgabe der Grafik
- Menü-Steuerung, Einbau der Schnittstellen-Funktionen als Callbacks
- Maus-Steuerung, z.B. Context Menü auf rechter Maustaste
- eventuell zusätzlicher Dialog zum Einstellen von Optionen

Die grafische Oberfläche ist erfahrungsgemäß ein Teil des Projekts, der bis zum Ende noch (leichte) Änderungen berücksichtigen muss.

2.5 Modale Dialoge

Modale Dialoge blockieren das Fenster, von dem sie aufgerufen wurden und stoppen den Programm-Ablauf in der aufrufenden Datei so lange, bis das Fenster wieder freigegeben wurde.

Um einen einen Dialog modal zu machen, müssen in der ***OpeningFcn*** des zugehörigen GUI-M-Files die unten stehenden, hervorgehobenen Einträge eingebaut werden:

```
function gui2_OpeningFcn(hObject, eventdata, handles, varargin)
    ...
    % Update handles structure
    guidata( hObject, handles );

    % macht den Dialog modal
    set(hObject,'WindowStyle','modal');

    % UIWAIT makes gui2 wait for user response (see UIRESUME)
    uiwait( handles.figure1 );
```

Um das *uiwait* wieder aufzuheben, sollte ein ***uiresume*** eingebaut werden, entweder in einer zusätzlich erzeugten ***CloseRequest***-Callback oder in den Callbacks, mit denen das Fenster geschlossen wird.

Wird von der GUI-Funktion nicht explizit ein Wert zurückgegeben (bei unseren Projekten geben die GUI-Funktionen nichts zurück!), dann empfiehlt es sich, in der *OutputFcn* den Rückgabe-Wert nicht von den *handles* abhängig zu machen, da Sie evtl. im Callback für *uiresume* das Fenster und damit die *handles* bereits gelöscht haben.

Setzen Sie deshalb in der ***OutputFcn*** an Stelle von "*varargout{1} = handles.output*;" den Rückgabewert zum Beispiel auf 0: "***varargout{1} = 0***;"

Besonders professionell wird ein Optionen-Dialog, wenn er auch die Möglichkeit zum **Abbruch** hat und sich dabei die ursprünglichen Daten nicht ändern. Am einfachsten erreicht man so ein Verhalten, indem man eine **temporäre Datenbasis** einbaut und während des Dialoges nur mit dieser temporären Datenbasis arbeitet.

Zu Beginn des Dialogs muss man dazu die Daten aus der globalen Datenbasis in den temporären struct kopieren. Und nur, wenn der Dialog über ein OK verlassen wird, kopiert man die eventuell geänderten temporären Daten wieder zurück auf den globalen struct. Bei einem Abbruch braucht man gar nichts weiter zu tun – die temporären Daten bleiben dann einfach unberücksichtigt.

2.6 Datei-Operationen

Aufgabe dieser Arbeitsgruppe ist es, die Datenbasis auf Wunsch in eine **Text-Datei** abzuspeichern bzw. eine gespeicherte Datenbasis aus einer Datei in das Hauptprogramm einzulesen, um die Daten dort grafisch darzustellen.

Das Hauptproblem der Datei-Operationen besteht darin, die Zeilen der Text-Datei mit den entsprechenden Komponenten des MATLAB-Daten-structs zu identifizieren. Hierzu versieht

man die Zeilen der Text-Datei am besten mit einem **Kenner**, der angibt, welche Art von Information die aktuelle Zeile gerade enthält - vgl. das Labyrinth-Beispiel aus der Vorlesung. Für den 8. Punkt im CAD-System mit den Koordinaten (4.0,3.7) könnte die zugehörige Zeile in der Text-Datei z.B. lauten:

```
P 8 4.0 3.7
```

Für die 5. Linie vom 8. zum 23. Punkt auf dem Layer 7 könnte die Zeile z.B. lauten:

```
L 5 8 23 7
```

Die Funktionen zur Ein- und Ausgabe haben außerdem noch die Aufgabe, die (syntaktische) Korrektheit der Daten zu überprüfen, z.B. müssen nach einem Kenner "P" in den CAD-Daten immer genau drei Zahlen kommen, für Nummer der Linie und Start- und End-Koordinate.

2.7 Grafik-Ausgabe

Diese Arbeitsgruppe hat unter anderem folgende Aufgaben:

- Datenbasis im GUI-Grafikbereich auszugeben, z.B. CAD-Linien zeichnen.
- Interaktionen im GUI-Grafikbereich zu steuern, z.B. Elemente mit der Maus picken.
- Evtl. Zoomen und Verschieben des Datenbereichs organisieren

2.8 Funktionalität

Je nach Art des Hauptprojekts gibt es noch einige weitere Arbeitsgruppen, die sich mit der Implementierung weiterer Funktionalität beschäftigen. Näheres dazu finden Sie in der Beschreibung der einzelnen Hauptprojekte.

3. 2D-CAD-System (c06)

Teil-Projekte

1. Start, Datenbasis, speichern, laden
2. GUI, Optionen-Menü, Hilfe, Info
3. Elemente zeichnen, Grid, Beschriftung, Layer ausblenden
4. Layer-Steuerung, Dialog für Farbe, Strichart, ...
5. Neue Elemente, Picken, Fangen auf Punkt, …
6. Verschieben eines Punktes, Nachführen der Daten
7. Zoom, Pan, Slider-Ansteuerung
8. Element löschen, Daten-Bereinigung, Elemente identifizieren

Mögliche Erweiterungen:

- Spline-Verwaltung, Bemaßung, DXF-Filter

3.1 Start / Datenbasis

Start des Programms über die MATLAB-function ***c06*** im M-File 'c06.m', zum Beispiel für eine erste Dummy-Version (eine etwas erweiterte Version liegt im CAD-Verzeichnis):

```
function c06
   % globale Variable c06_data anlegen:
   % muss in jeder function, die c06_data verwendet, ebenfalls so deklariert werden
   global c06_data;

   % in der Funktion c06_init_data wird der Daten-struct neu angelegt
   c06_data = c06_init_data( c06_data );

   % Initialisierung des structs c06_data mit Start-Daten:
   %     Liste mit den Layern: Layer 1 ist immer vorhanden
   c06_data.layer.anzLayer = 1;       % Anzahl der Layer
   c06_data.layer.aktLayer  = 1;      % beim Start ist immer Layer 1 aktuell
   c06_data.layer.ly(1).Col   = 'k';  % Farbe:     Schwarz (k)
   c06_data.layer.ly(1).Ltyp  = '-';  % Linientyp: Solid   (-)
   c06_data.layer.ly(1).sichtbar = 1;  % Layer ist nicht ausgeblendet

   %   Initialisierung der Geometrie-Daten:
   c06_data.geom.anzPt  = 0;     % bisher keine Punkte angelegt
   c06_data.geom.anzLn = 0;      % bisher keine Linien angelegt

   c06_dummy_data;    % Dummy-Daten für dieses Test-Projekt erzeugen, später raus

   % Aufruf der GUI-Oberfläche + sämtliche weiteren Aktionen:
   c06_gui;
return;
```

Das Anlegen der **Datenbasis** wurde hierbei in die Funktion *c06_init_data* ausgelagert, da die Initilalisierung der Daten im Laufe des Programms noch öfter vorkommt, z.B. wenn eine neue Zeichnung angelegt wird.

Anschließend werden zum Start noch ein paar Daten explizit gesetzt, z.B. die Eigenschaften des ersten Layers. In dieser Test-Umgebung werden durch die Funktion *c06_dummy_data* auch Test-Daten für die anderen Arbeitsgruppen erzeugt. Am Ende wird durch den Aufruf von *c06_gui* die grafische Oberfläche gestartet, die alle weiteren Benutzer-Aktionen steuert.

Wir wollen uns die Funktion ***c06_init_data*** etwas genauer ansehen, da hier die Datenbasis definiert wird. Die Inline-Dokumentation erklärt die Komponenten des structs:

```
function  c06_data = c06_init_data( c06_data )

   clear c06_data;    % Variable zur Sicherheit löschen

   % Bedeutung der Komponenten des structs c06_data:

   %   oberste Ebene des structs c06_data mit den Komponenten:
   %      allg        allgemeine Daten zur Konfiguration
   %      layer       Layer-Konfiguration in akt. Zeichnung
   %      geom        Liste der geometrischen Elemente in akt. Zeichnung
   c06_data = struct( 'allg', [], 'layer', [], 'geom', []  );

   %   Der Unter-struct allg enthält folgende Komponenten:
   %     für den Options-Dialog:
   %      fangradius    bis zu welchem Abstand sind 2 Punkte identisch
   %      grid          1 = Grid einschalten, 0 = ausschalten
   %      ...
   %     weitere Daten:
   %      angelegt    ob schon eine Zeichnung angelegt ist
   %      filename    Dateiname für aktuelle Zeichnung
   %      version     Versions-Nummer der Zeichnung
   %      zoom        der aktuelle Zoomfaktor (1 ist Vollbild)
   %      pan         welcher 2D-Punkt liegt in der Mitte des Fensters
   %      handles     handles-struct des GUI für Text-Ausgaben, etc.
   c06_data.allg  = struct( 'fangradius', 0.1, 'grid', 0, ...
                            'angelegt', 0, 'filename', '', 'version', 1.0, ...
                            'zoom', 1, 'pan', [], 'handles', []  );

   %   Der Unter-struct layer enthält folgende Komponenten:
   %     anzLayer      Anzahl der Layer
   %     aktLayer      Nummer des aktuellen Layers
   %     ly            Vektor-struct mit allen Layern in der Zeichnung
   c06_data.layer = struct( 'anzLayer', 0, 'aktLayer', 1, 'ly', [] );

   %   ly(n) ist ein Vektor vom Typ struct mit folgenden Komponenten:
   %     Col         Farbe des Layers im Vektor (Liste), in MATLAB-Notation, erweitert auf RGB
   %     Ltyp        Linientyp des Layers im Vektor (Liste), z.B - oder :
   %     sichtbar    ob dieser Layer sichtbar ist
   c06_data.layer.ly = struct( 'Col', [], 'Ltyp', [], 'sichtbar', 1  );


   %   Der Unter-struct geom enthält folgende Komponenten:
   %     anzPt       Anzahl der Punkte (Vertices) in Zeichnung
   %     pt          Vektor-struct mit allen Punkten in der Zeichnung
   %     anzLn       Anzahl der geraden Linien (Edges) in Zeichnung
   %     ln          Vektor-struct mit allen geraden Linien in der Zeichnung
   %   ...   (evtl. weiter mit anderen Geometrie-Elementen, wie Kreisbögen, ...)
   c06_data.geom = struct( 'anzPt', 0, 'pt', [], 'anzLn', 0, 'ln', [] );

   %   pt(n) ist ein Vektor vom Typ struct mit folgenden Komponenten:
   %     x           x-Koordinate des Punktes im Vektor (Liste)
   %     y           y-Koordinate des Punktes im Vektor (Liste)
   %     del         Punkt wurde gelöscht: del=1, ansonsten del=0
   c06_data.geom.pt  = struct( 'x', 0, 'y', 0, 'del', 0 );

   %   ln(n) ist ein Vektor vom Typ struct mit folgenden Komponenten:
   %     p1          Nummer des Start-Punktes (im pt-Vektor) der Linie
   %     p2          Nummer des End-Punktes (im pt-Vektor) der Linie
   %     ly          Nummer des Layers (im ly-Vektor) der Linie
   %     del         Linie wurde gelöscht: del=1, ansonsten del=0
   c06_data.geom.ln  = struct( 'p1', [], 'p2', [], 'ly', 1, 'del', 0 );

return;
```

Struktur der Zeichnungs-Dateien:

Zum Speichern und Laden von CAD-Zeichnungen verwenden wir ein eigenes Datei-Format, das folgenden Aufbau hat:

- Dateikenner 'c06'
- Info-Bereich als Kommentar
- Zeile mit Versions-Nummer
- Zeile mit Anzahl der Layer
- Zeile mit Anzahl der Punkte
- Zeile mit Anzahl der Linien
- ... (Erweiterungen z.B. für Kreise, etc.)
- Zeilen mit der Definition der Layer
- Zeilen mit der Definition der Punkte
- Zeilen mit der Definition der geraden Linie
- ... (Kreisbögen, ...)

Die Zeichnungs-Dateien haben die Endung ***.c06***. Als Beispiel für den Aufbau dieser Dateien mag ***DummyZeichnung.c06*** dienen:

```
c06
%
%  CAD-Beispiel-Zeichnung
%  U. Stein, 14.04.2006
%
% Versions-Nummer: V 1.1
V 1.1
% Fangradius: R wert
R 0.1
%
% Anzahl Layer: y Anzahl
y 3
% Anzahl Punkte: p Anzahl
p 4
% Anzahl Linien: l Anzahl
l 3
%
% 3 Layer: Y Nr Farbe LType sichtbar
Y 1 k - 1
Y 2 g - 1
Y 3 r : 1
%
% 4 Punkte: P Nr x y
P 1 0.0 0.0
P 2 4.0 6.0
P 3 -1.0 3.0
P 4 5.0 -2.0
%
% 3 Linien: L Nr Startp. Endp. Layer
L 1 2 3 1
L 2 2 1 2
L 3 1 3 3
```

Alle Zeilen, die mit % beginnen, sind Kommentar-Zeilen und werden beim Einlesen in das CAD-System ignoriert.

Ein Beispiel für das Einlesen dieser Datei in das CAD-System liegt als eingeschränkte Test-Funktion ***c06_lese_datei*** im CAD-Verzeichnis.

Aufgaben und Erweiterungen:
Kreise und Kreisbögen in die Datenstruktur einbauen
Lesen und Speichern der Kreis-Daten

3.2 GUI / Optionen / Hilfe / Info

Das Fenster mit der grafischen Oberfläche des CAD-Systems enthält zentral eine Axis, in der die 2D-CAD-Zeichnung visualisiert wird. Darum herum liegen zwei Slider, mit dem der Bild-Ausschnitt kontrolliert wird (Arbeitsgruppe Zoom+Pan).

Als Ausgangsbasis kann die Datei ***c06_gui.fig*** mit M-File ***c06_gui.m*** verwendet werden.

Auf der Axis liegt außerdem ein Context-Menü, mit dem häufig gebrauchte Befehle direkt abgerufen werden können, z.B. Neue Linie, Zoom, ...

Das **Haupt-Menü** hat (mindestens folgende Einträge):

- Datei: Neu / Öffnen / Speichern / Speichern als / Beenden
- Erzeugen: Linie / Polylinie / Kreis / Verschieben / Löschen
- Layer: aktuell / Verwaltung
- Ansicht: Zoom / Pan / Neu zeichnen
- Extras: Optionen
- Hilfe: Info / Hilfe

Zum CAD-System gehören auch noch folgende **weitere GUI-Projekte**:

- Optionen-Dialog, z.B. für Fangradius, Grid, ...
- Info-Dialog: Anzeige der Version, der Autoren, etc.
- Hilfe-Dialog: Kurze Einführung in die Bedienung des CAD-Systems

Vor das Beenden des CAD-Systems gehört auch noch eine Abfrage, ob der Benutzer jetzt wirklich aufhören will und ob die bestehende Zeichnung noch gespeichert werden soll.

Eine weitere Option wären Icons (z.B. als Pushbuttons), mit denen man am oberen Ende des Fensters häufig genutzte Befehle durch direktes Anklicken aufrufen kann.

Aufgaben und Erweiterungen:
Kontext-Menü, Optionen, Hilfe, Info, ...

3.3 Zeichnen

Zeichnen der Element-Daten unter Berücksichtigung von Layer bzw. akt. Einstellung, Grid, Beschriftung der Achsen, etc.

Als Startpunkt kann die Dummy-Datei ***c06_dummy_drawLines.m*** im CAD-Verzeichnis dienen, die dann z.B. noch auf Kreise und Splines erweitert werden kann.

Suchen Sie Möglichkeiten zum Zeichnen von Kreisen und Splines, in Zusammenarbeit mit Datenbasis-Gruppe, die die dazu notwendigen Daten verwalten muss.

Die Farben der Linien sollte von den MATLAB-Kürzel, wie k, b oder r, auf alle RBG-Farben erweitert werden.

Aufgaben und Erweiterungen:
Check auf Länge Null bei den Geraden
Erzeugen + Zeichnen von Kreisen, etc.
Highlight von Linien ermöglichen

3.4 Layer

Durch die Zuordnung von Layern zu den CAD-Linien schafft man eine Gruppierung von CAD-Elementen. Man kann z.B. für alle Elemente au einem Layer eine gemeinsame Operation ausführen, z.B. alle diese Elemente ausblenden.

Weitere Eigenschaften, die oft über einen Layer gesteuert werden sind:
Farbe der Linien, Linien-Typ

Das CAD-System steht immer aus einem bestimmten Layer, zu Beginn normalerweise auf dem Haupt-Layer 1, der immer vorhanden sein muss. Weitere Layer sind optional und können über die Layer-Verwaltung angelegt werden.

Die Arbeitsgruppe Layer hat zwei Hauptaufgaben:

- Dialog zum **Umschalten** des aktuellen Layers, dazu Anzeige aller Layer
- Dialog, zum Anlegen eines **neuen Layers** und zum Ändern der **Attribute** (Farbe, Linientyp, Sichtbarkeit) eines Layers, mit Anzeige aller Layer und deren Attribute

In der Test-Datenbasis wurden für die **Layer-Farben** nur die MATLAB-Kürzel, wie k, b oder r zugelassen. Dies sollte daraufhin geändert werden, dass alle RGB-Farben zugelassen sind, vgl. MATLAB-Dialog *uisetcolor*. Die Arbeitsgruppen Datenbasis und Zeichnen muss diese Farben natürlich auch berücksichtigen.

Aufgaben und Erweiterungen:
Layer-Verwaltung

3.5 Neue Elemente / Picken / Fangen

Das Anlegen neuer Elemente geschieht über ein Picken von Punkten, z.B. bei einer geraden Linie über die Angabe von Start- und Endpunkt.

Die Funktion ***c06_new_line*** könnte beispielsweise so aussehen:

```
function  c06_new_line()

  n1 = c06_pick_pnt();
  n2 = c06_pick_pnt();

  c06_create_line( n1, n2 );

return;
```

Durch den zweimaligen Aufruf der Funktion ***c06_pick_pnt*** werden zwei Punkte durch Picken mit der Maus festgelegt. Anschließend wird in ***c06_create_line*** damit eine gerade Linie definiert. Die entsprechenden M-Files finden Sie im CAD-Verzeichnis.

In der Funktion *c06_pick_pnt* wird nach dem Picken eines geometrischen Punktes im Axis-Bereich aber in der Funktion ***c06_find_pnt*** noch zusätzlich untersucht, ob der gepickte Punkt bereits in der Datenbasis eingetragen ist, d.h. ob es innerhalb des vorgegebenen Fangradius bereits einen Punkt gibt. Wenn ja, wird dieser Punkt genommen. Anderenfalls wird ein neuer Punkt in die Datenbasis eingetragen.

Aufgaben und Erweiterungen:
Picken vervollständigen,
vor dem Anlegen eines neuen Punktes im Array der Daten-Basis checken, ob es im Punkte-Array gelöschte Einträge gibt, an deren Stelle der neue Punkt gespeichert werden kann,
Check, ob Länge der Linie größer ist als eps bzw. Parameter "Minimale Kantenlänge"

3.6 Verschieben eines Punktes

Zum Verschieben eines Punktes muss man als erstes einen bestehenden Punkt picken, z.B. mit Hilfe der Funktion *c06_find_pnt*. Dann definiert man durch einen weiteren Pick, wohin dieser Punkt verschoben werden soll. In der Funktion ***c06_verschieben*** ist bereits ein Grundgerüst hierzu angelegt:

```
function  c06_verschieben()

   [nr,x,y] = c06_find_pnt();

   % nr = 0: kein Punkt innerhalb des fangradius gepickt
   if( nr == 0 )
     msgbox( 'Kein Punkt ausgewählt!' );
     return;
   end

   % Zweiten Punkt zum Verschieben picken
   % und den gepickten Punkt dahin verschieben
   % am Ende alles neu zeichnen

return;
```

Das Verschieben des gepickten Punktes bedeutet, dass man dessen Koordinaten in der Datenbasis ändert. Die daran hängenden Geometrien folgen dann von selbst.

Als Erweiterung könnte man dafür sorgen, dass die beteiligten geometrischen Elemente hervorgehoben gezeichnet werden, so lange die Aktion dauert. Wie man Elemente hervorheben kann (Highlight), müsste mit den Gruppen Datenbasis und Zeichnen abgesprochen werden.

Aufgaben und Erweiterungen:
Punkt verschieben,
Check auf Länge Null bei angrenzenden Linien
Highlight der angrenzenden Linien, Abfrage, ob diese gemeint sind

3.7 Zoom / Pan

Zoom und Pan (Verschieben) sind Eigenschaften der Axis-Darstellung und sollten im Fenster über folgende Elemente steuerbar sein:

- Zoom-Faktor eingeben, evtl. auch über die Maus,
- Pan-Dialog-Führung mit Pick-Punkten bzw.
- Steuerung über Slider rechts und unterhalb des Zeichenbereichs

Zoomfaktor 1 bedeutet, dass die gesamte Zeichnung in das Fenster passt.

Für die anderen Zoom-Werte muss die Größe der Zeichnung bestimmt werden, z.B. indem man kurzfristig alles darstellt und die axis-Werte dazu ausliest.

Alternativ könnte man natürlich auch die extremalen Punkte in der Datenbasis bestimmen, was aber bei komplexeren Kurven, wie Kreisen und Splines, evtl. zu falschen Größen führt.

In *c06_gui* sind bereits folgende Callback-Funktionen für Zoom + Pan vorgesehen:

- *zoom_menu_item_Callback*
- *pan_menu_item_Callback*
- *sliderVertical_Callback*
- *sliderHorizontal_Callback*

Aufgaben und Erweiterungen:
Vorgabe von DIN-Blattgrößen,
Slider-Steuerung,
Eingabe von Zoom-Faktor, Benutzerführung, ...

3.8 Element löschen

Das Hauptproblem beim Löschen eines Elements besteht darin, ein Element durch Picken eines Punktes zu bestimmen. In unserem CAD-Projekt soll dies nur für gerade Linien durchgeführt werden.

In der Funktion ***c06_loeschen*** ist bereits das Gerüst für diesen Befehl aufgestellt. Besonders die etwas komplexere Funktion ***c06_pnt_on_line***, die untersucht, ob der Punkt auf einer der Linien liegt, ist bereits fertig vorgegeben:

```
function  c06_loeschen()

  % Punkt mit der Maus picken
  [x,y] = ginput(1);

  global c06_data;    % globale Variable c06_data:

  nr = 0;   % Vorgabe

  % alle geraden Linien durchsuchen, ob Punkt darauf liegt
  for( ln=1:c06_data.geom.anzLn )
    if( c06_pnt_on_line( x,y,ln ) )
      nr = ln;
      break;
    end
  end
```

```
  % keine Linie gefunden
  if( nr == 0 )
    msgbox( 'Keine Linie identifiziert' );
    return;
  end

  % del-Flag für die gefundene Linie nr setzen
  % alles neu zeichnen

return;
```

Das Löschen von Linien und Punkten geschieht dadurch, dass deren del-Flag gesetzt wird.

Aufgaben und Erweiterungen:
Elemente identifizieren und löschen,
Highlight der Linien, Abfrage, ob diese gemeint sind
Punkte löschen, falls diese nicht von anderen Elementen auch benutzt werden

4. Bildverarbeitung (b06)

Teil-Projekte

1. Datenbasis, speichern, laden, Check der Daten
2. GUI, Menü, Context-Menü, Bilder zeichnen, Vorschau
3. RGB-Farbfilter, GUI-Dialog
4. Hochpass, Tiefpass, GUI-Dialog
5. Schwarz-Weiss, Farbverschiebung, GUI-Dialog
6. Invertieren, Helligkeit, Original herstellen, GUI-Dialog
7. Detail auswählen

Erweiterungen

* Farbaustausch um RBG-Umgebung auf andere RGB-Umgebung

4.1 Datenbasis, speichern, laden, Check

Aufgabe dieser Gruppe ist die Pflege der Datenbasis, u.a. mit folgenden Komponenten:

- Zwei-dimensionales Array der Original-Bildpunkte
- Zwei-dimensionales Array der modifizierten Bildpunkte, Gebiet für Detail
- Array für Vorschau-Zwischenspeicher
- Parameter der einzelnen Operationen: rel. RGB-Wert, Helligkeits-Prozent, ...

Weitere Aktionen:

- bmp-Bilder laden und modifiziertes Bild speichern / speichern als
- Check, ob geladenes Bild auch ein bmp-Farb-Bild ist

Für das Laden einer bmp-Datei gibt es bereits den halb-fertigen M-File ***b06_lade_bmp.m***. Hier fehlt noch der Check, ob die zu ladende Datei wirklich eine bmp-Farb-Datei ist. Für das Speichern müssen die M-Files ***b06_save_bmp.m*** bzw. ***b06_save_bmp_as.m*** vervollständigt werden.

Das Bildverarbeitungs-Programm selbst startet über die MATLAB-function ***b06*** im M-File 'b06.m', zum Beispiel für eine erste Dummy-Version:

```
function b06
   % globale Variable b06_data anlegen:
   % muss in jeder function, die b06_data verwendet, ebenfalls so deklariert werden
   global b06_data;

   % Daten-struct anlegen
   b06_data = b06_init_data( b06_data );

   % Aufruf der GUI-Oberfläche + sämtliche weiteren Aktionen:
   b06_gui;
return;
```

Das Anlegen der Datenbasis wurde hierbei in die Funktion ***b06_init_data*** ausgelagert, da die Initilalisierung der Daten im Laufe des Programms noch öfter vorkommt, z.B. wenn ein neues Bild geladen wird.

```
function  b06_data = b06_init_data( b06_data )
   clear b06_data;   % Variable zur Sicherheit löschen

   %   oberste Ebene des structs b06_data mit den Komponenten:
   %     allg              Unter-struct mit allgemeine Daten zur Konfiguration
   %     orig              Array mit dem Original-Bild
   %     modif             Array mit dem modifizierten Bild
   %     preview           Array mit dem Vorschau-Bild
   %     rgb               Unter-struct mit Parametern zum rgb-Filter
   %     hochpass          Unter-struct mit Parametern zum Hochpass-Filter
   %     tiefpass          Unter-struct mit Parametern zum Tiefpass-Filter
   %     bw_col            Unter-struct mit Parametern zu Schwarz-Weiß/Farbe
   %     brightness        Unter-struct mit Parametern zur Helligkeit
   %     detail            Unter-struct mit Parametern zum Detail
   b06_data = struct( 'allg', [],  'orig', [], 'modif', [], 'preview', [], ...
                      'rgb', [], 'hochpass', [], 'tiefpass', [], 'bw_col', [], 'brightness', [], 'detail', [] );

   %   Der Unter-struct allg enthält folgende Komponenten:
   %     version         Version des Bild-Projekts
   %     geladen         Flag, ob Bild geladen ist: 1 = ja, 0 = nein
   %     zeilen          Anzahl der Zeilen des Bildes
   %     spalten         Anzahl der Spalten des Bildes
   %     preview         ob aktuell das Preview angezeigt wird
   %     detail          Flag, ob Detail angezeigt werden soll, 1 = ja
   %     filename        Dateiname zum Speichern des Bildes
   %     handles         handles-struct des GUI für Text-Ausgaben, etc.
   b06_data.allg  = struct( 'version', 1.0, 'geladen', 0, 'zeilen', 0, 'spalten', 0,   ...
                            'preview', 0, 'detail', 0, 'filename', '', 'handles', []  );

   %   Der Unter-struct rgb enthält folgende Komponenten:
   %     rbg           relative Änderung der rgb_Werte
   b06_data.rgb = struct( 'rgb', [1,1,1] );

   %   Der Unter-struct hochpass enthält folgende Komponenten:
   %     frequenz      Abschneide-Frequenz für niedrige Frequenzen
   %     intensitaet   Rest-Intensitäten der niedrigen Frequenzen
   b06_data.hochpass = struct( 'frequenz', 0, 'intensitaet', 0  );

   %   Der Unter-struct tiefpass enthält folgende Komponenten:
   %     frequenz      Abschneide-Frequenz für hohe Frequenzen
   %     intensitaet   Rest-Intensitäten der hohen Frequenzen
   b06_data.tiefpass = struct( 'frequenz', 0, 'intensitaet', 0  );

   %   Der Unter-struct bw_col enthält folgende Komponenten:
   %     rbg           relativer rgb_Wert für grau
   b06_data.bw_col = struct( 'rgb', [1,1,1]  );

   %   Der Unter-struct brightness enthält folgende Komponenten:
   %     helligkeit    relativer Wert für die Helligkeit
   %     invert        Flag, ob invertiert werden soll, 1 = ja
   b06_data.brightness = struct( 'helligkeit', 1, 'invert', 0 );

   %   Der Unter-struct detail enthält folgende Komponenten:
   %     definiert     Flag, ob Detail definiert ist: 1 = ja, 0 = nein
   %     zeilen        relative Zeilen-Position des Details
   %     spalten       relative Spalten-Position des Details
   b06_data.detail  = struct( 'definiert', 0,  'zeilen', [0,1], 'spalten', [0,1]  );
return;
```

Aufgaben und Erweiterungen:
Speichern vervollständigen
Laden mit Check auf korrekten Bild-Typ, keine Schwarz-Weiß-BMP-Dateien, etc.

4.2 GUI, zeichnen, Vorschau

Zum Haupfenster gehören folgende Dateien:

- ***b06_gui.fig***
- ***b06_gui.m***

Aufgaben der Gruppe GUI:

- Axes für Original-Bild und modifiziertes Bild
- Context-Menu über den Axes
- evtl. weiterer Dialog für Optionen:
- Dialoge für Info, Hilfe

- Bilder zeichnen
- evtl. Detail zeichnen
- Vorschau + rückgängig

Zum Zeichnen der Bilder gibt es bereits eine Startfunktion ***b06_neu_zeichnen***, die u.a. die Funktionen ***b06_zeichne_orig*** und ***b06_zeichne_modif*** aufruft.

Ist das Flag ***b06_data.allg.detail*** auf 1 gesetzt, so soll im rechten Fenster nur ein Ausschnitt des Bildes angezeigt werden. im Unter-struct *b06_data.detail* stehen in den Komponenten *zeilen* und *spalten* die relativen Ausschnitte, z.B. bedeutet *zeilen = [0.25 ,0.50],* dass der Bildausschnitt ab dem Viertel der Zeilen beginnt und an der Hälfte endet.

Ist das Flag ***b06_data.allg.preview*** auf 1 gesetzt, dann werden die Vorschau-Daten an Stelle der modifizierten Daten angezeigt.

Änderungen werden immer **nacheinander** am modifizierten Bild ausgeführt

Die Dialoge für Info und Hilfe werden als eigenständige GUI-Dialoge über das b06_gui-Hauptmenü aufgerufen.

Aufgaben und Erweiterungen:
Kontext-Menü, Optionen, Hilfe, Info,
Schalter, ob Detail oder Modif. zeichnen

4.3 RGB-Farbfilter, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die relativen rgb-Parameter. Dieser Dialog wird über den Befehl ***b06_rgb_par*** aufgerufen.

- Farb-Filter, Dialog,
- Anwendung der Filter, vgl. Vorlesung
- Vorschau aktivieren

Änderungen werden immer **nacheinander** am modifizierten Bild ausgeführt

Aufgaben und Erweiterungen:
RBG-GUI + Filter, evtl. Vorschau,
als modalen Dialog

4.4 FFT, Filter, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die Tief- und Hochpass-Parameter. Diese Dialoge werden über die Befehle ***b06_tiefpass_par*** bzw. ***b06_hochpass_par*** aufgerufen.

- Berechnung der FFT, vgl. Vorlesung
- Definition von Filtern (Hochpass, Tiefpass, Restintensität)
- Anwendung der Filter, Rücktransformation, vgl. Vorlesung

Änderungen werden immer **nacheinander** am modifizierten Bild ausgeführt

Aufgaben und Erweiterungen:
GUI für Hoch-/Tiefpass, evtl. Vorschau,
als modalen Dialog

4.5 Schwarz-Weiß, Einfärbung, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die relativen Graustufen-Parameter und eine einfarbige Einfärbung des Bildes. Dieser Dialog wird über den Befehl ***b06_bw_col_par.m*** aufgerufen.

- in Schwarz-Weiß-Bild umwandeln, vgl. Vorlesung
- gleichmäßige Einfärbung des SW-Bildes, vgl. Vorlesung

Änderungen werden immer **nacheinander** am modifizierten Bild ausgeführt

Aufgaben und Erweiterungen:
GUI für BW + Einfärbung, evtl. Vorschau,
als modalen Dialog

4.6 Farben invertieren, Helligkeit, Dialog für Parameter, Original

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die relativen Helligkeits-Parameter und das Invertieren der Farben. Zusätzlich soll die Möglichkeit geschaffen werden, das mofifizierte Bild wieder mit den Daten des Original-Bildes zu versehen. Die Dialoge werden über die Befehle ***b06_brightness_par.m, b06_invert_par.m*** und ***b06_original_par.m*** aufgerufen:

- Farben invertieren, vgl. Vorlesung
- Helligkeit ändern, vgl. Vorlesung
- Original wieder herstellen

Änderungen werden immer **nacheinander** am modifizierten Bild ausgeführt

Aufgaben und Erweiterungen:
GUI für Helligkeit, evtl. Vorschau,
als modalen Dialog
Schalter: invertieren, Original

4.7 Detail

Die Hauptaufgabe dieser Gruppe besteht im Definieren eines Bildausschnittes durch Picken auf der Axes. Diese Aktion wird über den Befehl ***b06_detail.m*** aufgerufen.

- Picken des Detail-Bereichs,
- Identifizierung des Details in Bild-Array
- relative Position des Details berechnen

Darstellung am modifizierten Bild

Aufgaben und Erweiterungen:
Detail wählen, Ausschnitt identifizieren
Daten in modif. Bild kopieren

5. Musik (m06)

Teil-Projekte

1. Datenbasis (Stn)
2. GUI
3. Keyboard, Picken,
4. Spielen,Transposition
5. Hüllkurve, Dialog für Parameter
6. Hall, Dialog für Parameter
7. Vibrato/Tremolo, Dialog für Parameter
8. Zeichnen, Zoom

Weitere Ideen:
Samples einlesen, FFT, Filter, Hochpass, Tiefpass

5.1 Datenbasis

Start des Programms über die MATLAB-function ***m06*** im M-File 'm06.m':

```
function m06

   % globale Variable m06_data anlegen:
   % muss in jeder function, die m06_data verwendet, ebenfalls so deklariert werden
   global m06_data;

   % Daten-struct anlegen
   m06_data = m06_init_data( m06_data );

   % Initialisierung des structs m06_data mit Dummy-Start-Daten:
   m06_dummy_data;

   % Aufruf der GUI-Oberfläche + sämtliche weiteren Aktionen:
   m06_gui;

return;
```

Das Anlegen der Datenbasis wurde hierbei in die Funktion *m06_init_data* ausgelagert, da die Initilalisierung der Daten im Laufe des Programms noch öfter vorkommt, z.B. wenn ein neues Sample angelegt wird.

In dieser Test-Umgebung werden durch die Funktion ***m06_dummy_data*** auch Test-Daten für die anderen Arbeitsgruppen erzeugt. Am Ende wird durch den Aufruf von *m06_gui* die grafische Oberfläche gestartet, die alle weiteren Benutzer-Aktionen steuert.

Wir wollen uns die Funktion ***m06_init_data*** etwas genauer ansehen, da hier die Datenbasis definiert wird. Die Inline-Dokumentation erklärt die Komponenten des structs:

```
function  m06_data = m06_init_data( m06_data )

   clear m06_data;   % Variable zur Sicherheit löschen

   % Bedeutung der Komponenten des structs m06_data:
   %   oberste Ebene des structs m06_data mit den Komponenten:
   %     allg          Unter-struct mit allgemeine Daten zur Konfiguration
   %     oberton       Unter-struct mit Daten zu den Obertönen
   %     effekte       Unter-struct mit Daten zu den Effekten
   %     original      Array mit der Original-Amplituden-Funktion
   %     extern        0: aus Obertönen, 1: externe wav-Datei
   %     tfeld         Array mit den Zeitpunkten
   %     ffeld         Array mit den Frequenzpunkten
   %     schwingung    Array mit der Amplituden-Funktion nach Effekten
   %     tausschnitt   welcher Ausschnitt von schwingung in axes
   %     frequenzen    Array mit den Musik-Frequenz-Intensitäten
   %     fausschnitt   welcher Ausschnitt von frequenzen in axes
   %     ton           aktuell gespielter Ton, 1 = untransponiert
   %     transposition Array mit Zeit-Funktionen der transp. Töne
   m06_data = struct( 'allg', [], 'oberton', [], 'effekte', [], 'original', [], 'extern', 0, ...
                   'tfeld', [], 'ffeld', [], 'schwingung', [], 'tausschnitt', [0,1], ...,
                   'frequenzen', [], 'fausschnitt', [0,1], 'ton', 10, 'transposition', [] );

   %   Der Unter-struct allg enthält folgende Komponenten:
   %     version       Version des Musik-Projekts
   %     samplerate    Sample-Rate, für CDs 44100 / s
   %     frequenz      Test-Frequenz
   %     dauer         Zeitdauer des Tons in s
   %     intensitaet   Intensitaet des Tons
   %     zoomAmp       Zoomfaktor für Zeit-Amplitude
   %     zoomFrequ     Zoomfaktor für Frequenz-Amplitude
   %     keyaktiv      ob die Eingabe über Tastatur aktiv ist
   %     angelegt      ob bereits Daten definiert sind
   %     filename      Dateiname zum Speichern des Tons
   %     handles       handles-struct des GUI für Text-Ausgaben, etc.
   m06_data.allg  = struct( 'version', 1.0, 'samplerate', 44100, 'frequenz', 220, ...
                    'dauer', 3, 'intensitaet', 1, 'zoomAmp', 1.2, 'zoomFrequ', 1.2, ...
                    'keyaktiv', 0, 'angelegt', 0, 'filename', '',  'handles', []  );

   %   Der Unter-struct transposition enthält folgende Komponenten:
   %     frequenz      Frequenz des transponierten Zons
   %     schwingung    Array mit der Amplituden-Funktion nach Effekten
   m06_data.transposition = struct( 'frequenz', 0, 'schwingung', [] );

   %   Der Unter-struct oberton enthält folgende Komponenten:
   %     anzahl        Zahl der Obertöne + Grundton
   %     frequenz      Vektor mit relativen Frequenzen der Obertöne
   %     intensitaet   Vektor mit relativen Intensitäten der Obertöne
   m06_data.oberton = struct( 'anzahl', 9, 'frequenz', [], 'intensitaet', []  );

   %   Der Unter-struct effekte enthält folgende Komponenten:
   %     vibrato       Unter-struct mit Daten des Vibratos
   %     anzahl        Zahl der Effekte nacheinander
   %     reihenfolge   Reihenfolge der Effekte nacheinander
   %     tremolo       (1) Unter-struct mit Daten des Tremolos
   %     huellkurve    (2) Unter-struct mit Daten der Huellkurve
   %     hall          (3) Unter-struct mit Daten des Halls
   m06_data.effekte = struct( 'vibrato', [], 'anzahl', 8, 'reihenfolge', [], ...
                    'huellkurve', [], 'hall', [], 'tremolo', []  );

   %   Der Unter-struct vibrato enthält folgende Komponenten:
   %     frequenz      Frequenz des Vibratos, 0 = kein Vibrato
   %     amplitude     relative Amplitude des Vibratos in Prozent
   m06_data.effekte.vibrato = struct( 'frequenz', 5, 'amplitude', 5 );
```

```
    %   Der Unter-struct tremolo enthält folgende Komponenten:
    %     frequenz      Frequenz des Tremolos
    %     amplitude     relative Amplitude des Tremolos in Prozent
    m06_data.effekte.tremolo = struct( 'frequenz', 5, 'amplitude', 20 );

    %   Der Unter-struct huellkurve enthält folgende Komponenten:
    %     tattack       Attack-Zeit
    %     hattack       Attack-Höhe
    %     tsustain      Sustain-Zeit
    %     trelease      Release-Zeit
    %     hrelease      Release-Höhe
    m06_data.effekte.huellkurve = struct( 'tattack', 0.1, 'hattack', 1, ...
                        'tsustain', 2.0, 'trelease', 0.3, 'hrelease', 0.4 );

    %   Der Unter-struct hall enthält folgende Komponenten:
    %     hallzeit      Zeit bis Hall zurückkommt in s
    %     anzahl        Anzahl der Hall-Wiederholungen
    %     amplitude     rel. Amplitide des Halls in Prozent
    m06_data.effekte.hall = struct( 'hallzeit', 0.2, 'anzahl', 3, 'amplitude', 30 );

return;
```

Struktur der Musik-Dateien:

Zum Speichern und Laden von Musik-Daten verwenden wir ein eigenes Datei-Format. Die Musik-Dateien haben die Endung ***.m06***. Als Beispiel für den Aufbau dieser Dateien mag ***dummy.m06*** dienen:

```
m06
%
% Musik-Zeichnung - Projekt im SS 2006
% erzeugt am 25-Apr-2006

% Versions-Nummer: V nummer
V 1
% Sample-Rate: S wert
S 44100
% Frequenz: f frequenz
f 220
% Dauer: d dauer
d 1
% Intensität: i intensitaet
i 1

% Obertöne: o anzahl
o 13
% Oberton: O nr frequenz intensitaet
O 1 1 1
% Oberton: O nr frequenz intensitaet
O 2 2 0
% Oberton: O nr frequenz intensitaet
O 3 3 0.333333
% Oberton: O nr frequenz intensitaet
O 4 4 0
% Oberton: O nr frequenz intensitaet
O 5 5 0.2
% Oberton: O nr frequenz intensitaet
O 6 6 0
% Oberton: O nr frequenz intensitaet
O 7 7 0.142857
% Oberton: O nr frequenz intensitaet
O 8 8 0
% Oberton: O nr frequenz intensitaet
O 9 9 0.111111
```

```
% Oberton: O nr frequenz intensitaet
O 10 10 0
% Oberton: O nr frequenz intensitaet
O 11 11 0.0909091
% Oberton: O nr frequenz intensitaet
O 12 12 0
% Oberton: O nr frequenz intensitaet
O 13 13 0.0769231

% Vibrato: v frequenz  amplitude (in Prozent)
v 0.1 0.7

% Effekte: e anzahl
e 3
% Effekt: E nr effekt
E 1 1
% Effekt: E nr effekt
E 2 2
% Effekt: E nr effekt
E 3 3
% Tremolo: t frequenz amplitude (in Prozent)
t 2 5
% Hüllkurve: k tattack hattack tsustain trelease hrelease
k 0.1 1 0.7 0.15 0.6
% Hall: h hallzeit anzahl amplitude (in Prozent)
h 0.2 5 1
```

Alle Zeilen, die mit % beginnen, sind Kommentar-Zeilen und werden beim Einlesen ignoriert.

Als ein Beispiel für das Einlesen dieser Datei liegt eine Test-Funktion ***m06_lese_datei*** im Musik-Verzeichnis.

5.2 GUI, Obertöne, Hilfe, Info

Zum Hauptfenster gehören folgende Dateien:

- ***m06_gui.fig***
- ***m06_gui.m***

Aufgaben der Gruppe GUI:

- Axes für Funktion und Frequenz-Bereich
- Eingabefelder für Grundton, Intensität, Dauer
- Eingabefelder für Obertöne (relative Intensitäten / Frequenzen)
- Context-Menu über den Axes
- weiterer Dialog für Optionen: Sample-Rate, ...
- Dialoge für Info, Hilfe

Im **Hauptfenster** sollen für den additiven Synthesizer folgende Eingaben möglich sein (vgl. auch die oben angegebene Datei dummy.m06 bzw. den M-File *m06_dummy_data*):
* Grundton (Standard 220 Hz)
* Sample-Rate (Standard 44.100 Hz)
* Dauer (Standard z.B. 2 sec)
* Intensität (Standard 1)
* Ca. 15 Obertöne mit den Frequenzen mit den zugehörigen rel. Intensitäten
* Zoom-Bereich für Funktion und Frequenzen, siehe entsprechende Variable in *m06_data*.

Aufgaben und Erweiterungen:
GUI-Eingabefelder, etc.

5.3 Keyboard, Picken

Zum Spielen mit einem virtuellen Keyboard öffnet sich ein separates GUI-Fenster, in dem eine Klaviatur abgebildet ist (kann zB. auch ein Foto aus einer bmp-Datei sein). Der Start-Befehl im Hauptfenster lautet: ***m06_keyboard***.

Mit der Maus kann man nun einzelne Tasten picken. Dazu soll der zugehörige Ton berechnet werden und auch erklingen.

Der Spielbereich umfasst genau eine Oktave, d.h. die 13 Töne:
C, Cis, D, Dis, E, F, Fis, G, Gis, A, Ais, H, C,
am besten die "**Kleine Oktave**" unterhalb des mittleren c'.

Anmerkung: Bei der temperierten Stimmung ist das Frequenz-Verhältnis zweier aufeinander folgender Halbtöne die 12. Wurzel aus 2. Der Oktav-Abstand (12 Halbtöne) verdoppelt gerade die Frequenz.

Die Zuordnung und die Frequenzen finden Sie im M-File *m06_transposition.m*. Die Töne werden dort auch bereits berechnet und unter *m06_data.transposition(tr).schwingung* gespeichert (für tr = 1, 2…, 13).

Mit folgendem Befehl wird der Ton mit der Nummer *ton* gespielt:

```
wavplay( m06_data.transposition(ton).schwingung, m06_data.allg.samplerate );
```

Aufgaben und Erweiterungen:
Separates GUI mit Keyboard,
picken + Identifikation des zugehörigen Tons
Ton aufrufen, analog der Zuordnung in 5.4

5.4 Spielen, Transposition

Die Gruppe Spielen, Transposition kann evtl. mit 5.3 zusammenarbeiten.

Im Unterschied zu 5.3 sollen die Töne hier über die Tastatur angesteuert werden. Ausgangspunkt ist die Datei ***m06_spiele_ton.m***, die über die KeyPressFcn des Hauptfensters aufgerufen wird und als Parameter die gedrückte Taste übergeben bekommt.

Um das Spielen ähnlich einer Klaviatur zu halten, sollten folgende Tasten verwendet werden:

```
  f g     j   k   l        entsprechend den     2  4      7  9   11
c v  b  n  m  ,  .  -       Ton-Nummern:       1  3  5  6   8  10  12  13
```

Mit folgendem Befehl wird der Ton mit der Nummer *ton* gespielt:

```
wavplay( m06_data.transposition(ton).schwingung, m06_data.allg.samplerate );
```

Bevor man Spielen kann sollten einmalig die transponierten Amplituden-Zeitfunktionen berechnet werden. Hierzu dient der Befehl ***m06_spiele_tonleiter***, der vom Menü im Hauptfenster aufgerufen wird und das Spielen mit der Tastatur aktiviert.

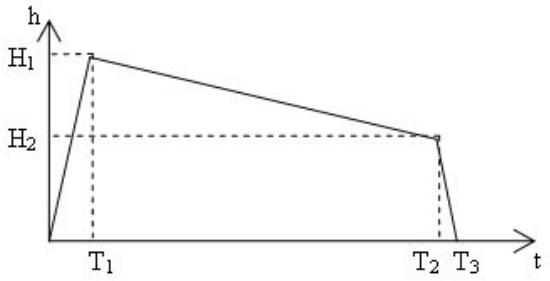
Aufgaben und Erweiterungen:
Töne zuordnen und Spiel aktivieren,
zur Zeit reagiert das System sehr träge – Analyse des Codes: Woran könnte das liegen?
Gibt es eine Möglichkeit, die Ton-Ausgabe zu beschleunigen?

5.5 Hüllkurve, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die Hüllkurven-Parameter. Dieser Dialog wird über den Befehl ***m06_huellkurven_par*** aufgerufen.

Dialog für Hüllkurven-Parameter:

- T1 Attack-Zeit
- T2 Sustain-Zeit
- T3 Release-Zeit
- H1 Attack-Höhe
- H2 Release-Höhe



Wenn die Hüllkurven-Daten in das globale struct ***m06_data*** eingetragen sind, werden sie automatisch zum Zeichnen und Spielen verwendet, falls die Hüllkurve als Effekt in der Datei ***m06_effekt_reihenfolge.m*** eingetragen ist. Die Hüllkurve hat die Effekt-Nummer 2.

Die Implementation dieses Effektes finden Sie im M-File ***m06_add_huellkurve.m***.

Aufgaben und Erweiterungen:
GUI für Hüllkurven-Parameter
modaler Dialog

5.6 Hall, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen des Dialogs für die Hall-Parameter:

- Hallzeit (Standard: 0.2 sec)
- wie viele Wiederholungen (Standard: 1)
- relative Amplitude (in Prozent, Standard: 1 %)

Dieser Dialog wird über den Befehl ***m06_hall_par*** aufgerufen.

Wenn die Hall-Daten in das globale struct ***m06_data*** eingetragen sind, werden sie automatisch zum Zeichnen und Spielen verwendet, falls der Hall als Effekt in der Datei ***m06_effekt_reihenfolge.m*** eingetragen ist. Die Hall hat die Effekt-Nummer 3.

Hall ist eine zeitliche Wiederholung des Signal mit schwächerer Intensität.

Man kopiert dazu das Array mit der Amplituden-Funktion, verkleinert die Amplitude um einen prozentualen Faktor und addiert dieses Array zeitlich versetzt zum Original-Signal. Wenn der Hall zum Beispiel um 10 Einheiten später kommt, dann lautet das einmal verhallte Signal für die Zeit t: A(t) = A(t) + h(t-10), natürlich erst ab t=11.

Die Implementation dieses Effektes finden Sie im M-File ***m06_add_hall.m***.

Aufgaben und Erweiterungen:
GUI für Hall-Parameter
modaler Dialog

5.7 Vibrato / Tremolo, Dialog für Parameter

Die Hauptaufgabe dieser Gruppe besteht im Erzeugen der beiden Dialoge für die Vibrato- und Tremolo-Parameter.

Diese Dialoge wird über die Befehle ***m06_vibrato_par*** bzw. ***m06_tremolo_par*** aufgerufen und haben beide die zwei Parameter:

- Frequenz (Standard: V=0.1, T=2.0)
- relative Amplitude (in Prozent, Standard: V=1 %, T=5 %)

Diie Vibrato- und Tremolo-Daten werden in das globale struct ***m06_data*** eingetragen.

Das Vibrato wird direkt beim Erzeugen der Grund-Schwingung in m06_additiver_synthie.m eingebaut, falls die Vibrato-Frequenz und –Amplitude größer als Null sind.

Das Tremolo wird als Effekt beim Spielen verwendet, falls es als Effekt in der Datei ***m06_effekt_reihenfolge.m*** eingetragen ist. Das Tremolo hat die Effekt-Nummer 1.

Ein **Vibrato** ist eine zeitlich periodische Variation der Grundfrequenz f_0, also z.B.

$$f(t) = f_0 * (1 + v_1 * \sin(2*pi*f_1*t))$$

Die Amplitude der Variation v1 ist normalerweise klein gegen 1. Die Vibrato-Frequenz f_1 liegt normalerweise im Berich weniger Hertz. Die Amplituden-Funktion ergibt sich damit zu

$$A(t) = A0 * \sin(2*pi* f(t) * t)$$

Die Implementation des Vibratos finden Sie im M-File ***m06_vibrato_par.m***.

Ein **Tremolo** ist eine zeitlich periodische Variation der Amplitude A_0, also z.B.

$$A(t) = A_0 * (1 + a_1 * \sin(2*pi*f_2*t))$$

Die Amplitude der Variation v1 ist normalerweise klein gegen 1. Die Implementation des Tremolo-Effektes finden Sie im M-File ***m06_add_tremolo.m***.

Aufgaben und Erweiterungen:
GUI für Vibrato- und Tremolo-Parameter
modaler Dialog

5.8 Zeichnen, Zoom

Das Zeichnen der Grafiken für die Amplitude und die Frequenzen wird über den Befehl ***m06_neu_zeichnen*** gestartet.

```
function  m06_neu_zeichnen()

   global m06_data;
   if( m06_data.allg.angelegt == 0 )
      fprintf( 'Debug: Noch keine Daten angelegt \n' );
      return;
   end

   m06_berechne_schwingung;
   m06_berechne_frequenzen;

   m06_zeichne_amplitude;
   m06_zeichne_frequenzen;

return;
```

Vor dem Zeichnen wird unter Berücksichtigung des Grundtons und der Reihenfolge der Effekte (Hall, Tremolo, Hüllkurve) die Amplituden-Zeitfunktion und die Frequenz-Verteilung berechnet.

Anschließend werden die Amplituden-Zeitfunktion und Frequenz-Verteilung gezeichnet, der Zoom-Bereich festgelegt und die Achsen beschriftet.

Bis auf die Beschriftung der Achsen dürften die Funktionen bereits fertig sein. Sie sollten Sie aber noch ausgiebig testen.

Als weiteres sollten Sie einen Ausschnitt als **Zoom-Bereich** wählbar machen, indem Sie auf die Position der Slider unterhalb der Fenster reagieren.

Zum Slider der Amplituden-Zeitfunktion gehört der Callback ***slider1_Callback*** im M-File *m06_gui.m*, zum Slider zur Frequenz-Funktion gehört der Callback ***slider2_Callback.***

Im struct *m06_data* werden die Zoom-Werte in den Komponenten ***tausschnitt(1)*** und ***(2)*** für die Zeitfunktion bzw. in ***fausschnitt(1)*** und ***(2)*** für die Frequenzen erfasst.

Aufgaben und Erweiterungen:
Slider-Steuerung, Zoom-Bereich zeichnen, Achsen beschriften

5.9 Erweiterung: FFT / Filter

Berechnung der FFT, Definition von Filtern, Anwendung der Filter, Rücktransformation

6. Strategiespiel: Sokoban (s06)

Teil-Projekte

1. Datenbasis, Start des Spiels, Zug zurück
2. GUI, Menüs, Tastatur-Steuerung des Spielers + switch + beep, Optionen-, Info-Dialog
3. Ablauf: Spiel auswählen + laden
4. Funktionen zum Zeichnen der Figuren, Steine, Kisten, etc., mehrere zur Auswahl
5. bmp erzeugen aus Figuren-Funktionen, Spielfeld zeichnen, Ablauf des Zeichnens
6. Verschiebungs-Logik, Ablauf des Ziehens

6.1 Datenbasis / Zug zurück

Start des Programms über die MATLAB-function ***s06*** im M-File 's06.m':

```
function s06()

    % globale Variable s06_data anlegen:
    % muss in jeder function, die s06_data verwendet,
    % ebenfalls so deklariert werden
    global s06_data;

    % globale Datenstruktur initialisieren
    s06_data = s06_init_data( s06_data );

    % GUI des Spiels starten
    s06_gui;
end
```

Das Anlegen der Datenbasis wurde hierbei in die Funktion *s06_init_data* ausgelagert, da die Initilalisierung der Daten im Laufe des Programms noch öfter vorkommt, z.B. wenn ein neues Spiel geladen wird.

Wir wollen uns die Funktion ***s06_init_data*** etwas genauer ansehen, da hier die Datenbasis definiert wird. Die Inline-Dokumentation erklärt die Komponenten des structs:

Hauptsächlich sind folgende Komponenten im struct s06_data vorhanden:

- Flag, ob ein Spiel geladen ist
- Zahl der Zeilen und Spalten
- Zwei-dimensionales Array des Labyrinths, jeweils mit Zahl für Belegung der Felder: Mauer 0, leer 1, Kiste 2
- Backup des Arrays für Zug zurück
- Position Spieler
- Optionen zur Spielfigur als Unter-struct
- Position der Zielpunkte als Unter-struct
- Array für das bmp-Image

```
function s06_data = s06_init_data( s06_data )

   clear s06_data;    % Variable löschen

   % Bedeutung der Komponenten des structs s06_data
   %   version      Version des Spiel-Projekts
   %   geladen     ob ein Spiel aktiv ist
   %   ende         ob ein Spiel beendet wurde
   %   pixel        Spielfiguren-Größe in Pixel, fest auf [16,20]
   %   zeilen       wie viele Zeilen das Spielfeld hat
   %   spalten      wie viele Spalten das Spielfeld hat
   %   p            aktuelle Position der Spielfigur
   %   p_vor        vorherige Position der Spielfigur
   %   dir          Laufrichtung der Figur = Cursor-Taste
   %   figur        Unter-struct mit Optionen zur Figur
   %   anzziele     wie viele Ziele gibt es = ebenfalls Stein-Zahl
   %   ziel         struct-Array mit den Ziel-Positionen
   %   feld         Belegung des Spielfeldes: 0 = Mauer,
   %                                  1 = freies Feld, 2 = Kiste
   %   feld_vor     Feld-Belegung vor dem letzten Zug für Zurück
   %   imag         bmp-Bild-Array des Spielfeldes
   %   handles      handles-struct des GUI für Text-Ausgaben, etc.
   %
   s06_data = struct( 'version', 1.0, 'geladen', 0, 'ende', 0, 'pixel', [16,20], ...
                      'zeilen', 0, 'spalten', 0, ...
                      'p', [], 'p_vor', [], 'dir', 0, 'figur', [], ...
                      'anzziele', 0, 'ziel', [], ...
                      'feld', [], 'feld_vor', [], 'imag', [], 'handles', [] );

   %  Der Unter-struct figur enthält folgende Komponenten:
   %    nr       Nummer der Spiel-Figur (Auswahl aus mehreren)
   %    col      RGB-Farbe der Spiel-Figur
   %    back     1 = Zug zurück erlaubt, 0 = nicht erlaubt
   s06_data.figur = struct( 'nr', 1, 'col', [255,0,0], 'back', 1 );

   %  Der Unter-struct ziel enthält folgende Komponenten:
   %    x       x-Position des Ziels
   %    y       y-Position des Ziels
   s06_data.ziel = struct( 'x', [], 'y', [] );

   s06_data.geladen = 0;
   s06_data.zeilen  = 0;
   s06_data.spalten = 0;

   % feste Größe der Spielfiguren in Pixel
   s06_data.pixel = [16,20];
end
```

Funktionen zum Ablauf:

- Datenbasis löschen, initialisieren,
- Start des Spiels

Aufgaben und Erweiterungen:

Zug zurück, Datei "***s06_zurueck.m***", nach Eingabe von 'z' bzw. über Context-Menü
Datenbasis erweitern: Spielfeld-Farbe, Stein-Typ,
evtl. Spielstand speichern

6.2 GUI / Tastatur-Steuerung / Optionen- und Info-Dialog

Zum Hauptfenster gehören folgende Dateien:

- ***s06_gui.fig***
- ***s06_gui.m***

Aufgaben der Gruppe GUI:

- GUI erweitern, Axes für Spielfeld verwalten, Tastatur verwalten
- Menü: neu zeichnen, Optionen, Hilfe, Ende, ...
- Context-Menü für zurück, neu zeichnen, ...
- Info-Dialog, evtl. Kurz-Hilfe
- Optionen-Dialog: modal, Figurenart/Farbe, ob zurück erlaubt ist, etc.

Aufgaben und Erweiterungen:
Kontext-Menü, Optionen-GUI modal, Info, Hilfe

6.3 Ablauf: Spiel auswählen + laden

Aufgaben dieser Gruppe:

- Datei-Dialog
- Spiele-Datei einladen, analysieren und in Datenbasis eintragen
- Optionen-Dialog (welche Figur/Farbe etc.) von 6.2 übernehmen
- Laden erweitern um: Figurtyp und –Farbe, Startrichtung, Version, ...
- Beispiel-Dateien erzeugen, am besten aus dem Original-Sokoban (z.B. unter www.pimpernel.com/sokoban/sokoban.html)

Aufbau der Level-Dateien am Beispiel ***Level1.s06***:

```
s06
% Sokoban Level 1 für MATLAB-Spiel
% Zahl der Zeilen und Spalten
z 11
s 19
% Start-Position der Figur:  p z s
p 9 12
% Zahl der Zielpunkte:  E anz
E 6
% Zielpunkt:  e nr z s
e 1 7 17
e 2 7 18
e 3 8 17
e 4 8 18
e 5 9 17
e 6 9 18
% Spielfeld:  n nr xxxxxxxxxx
n 1 0000000000000000000
n 2 0000011100000000000
n 3 0000021100000000000
n 4 0000011200000000000
n 5 0001121210000000000
n 6 0001010010000000000
n 7 0111010010000011110
n 8 0121121111111111110
```

```
n 9 0000010001010011110
n 10 0000011111000000000
n 11 0000000000000000000
```

Folgende **Kenner** für den Beginn einer Zeile in der Datei sind bereits vergeben:

- s06 in 1. Zeile der Datei
- % für Kommentar-Zeilen
- z Zahl der Zeilen des Spiels (z zeilen)
- s Zahl der Spalten des Spiels (s spalten)
- p Startposition der Figur (p zeile spalte)
- E Anzahl der Zielpunkte = Anzahl der Kisten (E anzahl)
- e Zeile mit Definition eines Ziels (e nummer zeile spalte)
- n Zeile mit Definition einer Spielfeld-Zeile (0=Mauer, 1=frei, 2=Kiste)

Folgende **Kenner** sollen noch eingebaut werden:

- v Versionsnummer zu Beginn der Datei, nach s06 (v nummer, z.B. v 1.0)
- d Start-Blickrichtung der Figur (d richtung), richtung = Cursortasten-Code
- c RGB-Farbe der Spielfigur (c r g b)
- t Typ der Spielfigur (t nummer)

Hauptsächlich folgende M-Files werden von dieser Gruppe gepflegt:

- s06_lade_spiel.m
- s06_lese_spiel.m

Ein Großteil des Codes ist bereits fertig. Folgende Erweiterungen sind noch einzubauen:

- kleinere Checks im Spielablauf
- neue Level-Dateien erzeugen, Typ s06, vgl. Verzeichnis Levels oder obigen Link
- Lesen der s06 erweitern: Figurtyp und –Farbe, Startrichtung, Version, ...

Aufgaben und Erweiterungen:
Einlesen erweitern: Version, Blickrichtung, Farben, ...
weitere Spiel-Dateien erzeugen

6.4 Funktionen: Figuren zeichnen, ...

Aufgaben dieser Gruppe:

- Funktionen zum Zeichnen der Figuren, Steine, Kisten, etc.,
- mehrere Varianten zur Auswahl, mehrere Farben (frei mit Dialog wählbar)
- vier Versionen der Figur, um Lauf-Richtung zu markieren

Startversionen für das Zeichnen liegt in den M-Files:

- s06_zeichne_figur.m
- s06_zeichne_stein.m
- s06_zeichne_kiste.m (Code noch implementieren)
- s06_zeichne_ziel.m (andere Form der Ziel-Positzion!)
- s06_leeres_feld.m

Speziell für die Spielfiguren sollen mehrere Versionen erstellt werden:

- unterschiedliche Figur-Typen, am besten von oben
- Farbe der Kleidung frei wählbar (über RGB-Feld s06_data.figur.col)
- 4 Typen für die Laufrichtung der Figur, Auswahl nach Cursor-Code

Aufgaben und Erweiterungen:
Figuren entwerfen, Laufrichtung, ...

6.5 Spielfeld zeichnen

Aufgaben dieser Gruppe:

- bmp erzeugen aus Figuren-Funktionen,
- Kisten zeichnen, Laufrichtung und Figur-Farbe und –Typ berücksichtigen
- Spielfeld zeichnen,
- Ablauf des Zeichnens

Eine Startversion für das Zeichnen liegt in "***s06_draw.m***", von dem aus weitere M-Files aufgerufen werden.

Aufgaben und Erweiterungen:
Aufruf zum Kisten-Zeichnen
Highlight einbauen, wenn Spiel gewonnen wurde, z.B. Blinken der Kisten
evtl. weitere Spiel-Dateien erzeugen

6.6 Verschiebungs-Logik

Aufgaben dieser Gruppe ist die Implementation der Verschiebungs-Logik:

- Feld in Laufrichtung frei: Ok, Position verschieben, letzte Position merken
- Stein auf nächstem Feld: verboten
- Kiste auf nächstem Feld: Check ob übernächstes Feld frei, wenn Nein -> verboten
    - wenn ja: Kistenfeld mit 1 (frei) belegen,
    - übernächstes Feld mit 2 (Kiste) belegen,
    - Figur –Position auf nächstes Feld setzen

Am Ende: Check auf Spielende: ob alle Zielpunkte mit Kisten belegt sind

Eine Startversion für das Zeichnen liegt in "***s06_ziehe.m***".

Aufgaben und Erweiterungen:
Logik implementieren, Check auf Spiel-Ende

6.7 Erweiterung: Level-Editor

Grafische Oberfläche zum Aufbau eines Spielfeldes und
Abspeichern als neue Sokoban-Datei

7. Tic-Tac-Toe (t06)

Teil-Projekte

1. Start-Dialog, Figuren, Farben
2. Daten, Ablauf, Teil-Analyse
3. GUI, Zeichnen (Figuren, Feld), Analyse
4. Strategie 3x3
5. Strategie1 4x4
6. Strategie2 4x4

7.1 Start-Dialog, Figuren

Über die GUI-Funktion ***t06_option_gui*** wird der Options-Dialog gestartet, in dem man die Einstellungen für ein Spiel festlegen kann. Die gewählten Einstellungen werden im globalen struct t06_data abgelegt, siehe unten. Folgende Optionen werden angeboten:

- Schalter: 3x3 vs. 4x4
- Wahl: Spieler1 / Spieler2 als Name bzw. Strategie (1/2)
- Figuren-Wahl für Spieler 1 und 2

Aufgaben und Erweiterungen:
Dialog zur Figurenwahl, Spielernamen, 3x3 oder 4x4

7.2 Datenbasis, Ablauf, Teil-Analyse

Datenbasis:
globales struct ***t06_data***, das in der Startfunktion t06 angelegt wird:

```
%  Startfunktion t06() für das Spiel TicTacToe
function  t06()

   global t06_data;     % globale Variable t06_data:
   % Bedeutung der Komponenten des structs t06_data:
   %   geladen     ob ein Spiel aktiv ist
   %   ende        ob ein Spiel beendet wurde
   %   zeilen      wie viele Zeilen das Spielfeld hat, erlaubt sind 3 oder 4
   %   name1       Name des 1. Spielers
   %   strategie1  Strategie des 1. Spielers:
   %                 0 : manuelle Eingabe der Spielzüge
   %                 1 : Computer mit Strategie 1  /   2 : Computer mit Strategie 2, nur für 4x4
   %   fig1        Nummer der Spielfigur für Spieler 1 /   analog die Daten für Spieler 2
   %   amzug       Merker, welcher Spieler am Zug ist, 1 oder 2
   %   pixel       Pixelzahl für die Spielfigur, fest [16,16]
   %   feld        Belegung des Spielfeldes: 0 = freies Feld, 1 = Stein von Spieler 1, 2 analog
   %   imag        bmp-Bild-Array des Spielfeldes mit den Steinen
   %   handles     handles-struct des GUI für Text-Ausgaben, etc.
   t06_data = struct(    'geladen', 0, 'ende', 0, 'zeilen', 3, ...
                         'name1', '', 'strategie1', 0, 'fig1', 1, ...
                         'name2', '', 'strategie2', 1, 'fig2', 2, ...
                         'amzug', 1,  'pixel', [], 'feld', [],  'imag', [], 'handles', [] );
   t06_data.geladen = 0;       % noch ist kein Spiel definiert:

   t06_gui;                    % TicTacToe-GUI starten
return;
```

Ablauf:
Als Ausgangspunkt für den Ablauf kann die Funktion ***t06_play*** (in Ansätzen zu finden im M-File t06_play.m) dienen:

```
function  t06_play()

   global t06_data;            % Zugriff auf globale Variable t06_data

   t06_draw;                   % zu Beginn das leere Spielfeld zeichnen

   % Flag, ob noch Felder frei sind, bzw, ob jemand gewonnen hat
   t06_data.ende = 0;

   % in Schleife, bis zum Ende
   while( t06_data.ende == 0 )
      ende = t06_felder_frei;   % Check, ob noch Felder frei sind
      if( ende )
         fprintf( 'Debug: Spiel zu Ende \n' );  % Ende-Meldung
         return;
      end

      t06_ablauf;     % Ablauf: Wer ist dran? 1 oder 2

      t06_draw;       % Spielfeld wieder neu zeichnen, mit Check, ob jemand gewonnen hat
   end
return;
```

Das Spiel läuft in einer Schleife. Zu Beginn wird getestet, ob es überhaupt noch freie Felder gibt. Dann kommt der eigentliche Zug in t06_ablauf. Am Ende des Zuges wird das Spielfeld neu gezeichnet und analysiert, ob der Spieler gewonnen hat. Dies ist aber bereits Aufgabe der Gruppe GUI / Zeichnen / Analyse.

In der Unterfunktion ***t06_ablauf*** (in Ansätzen zu finden im M-File t06_play.m) wird zuerst nachgeschaut, wer als nächstes am Zug ist, Spieler 1 oder 2, und welche Strategie dieser Spieler hat – entweder manuelles Picken oder eine Computer-Strategie (siehe weiter unten).

Beim Picken, teilweise realisiert in der Unterfunktion ***t06_pick*** (zu finden im M-File t06_play.m) wird die MATLAB-Funktion *ginput* aufgerufen, die die Koordinaten des gepickten Punktes [x,y] zurückgibt. Diese Koordinaten beziehen sich auf das gezeichnete bmp-Bild (mit der Größe 3*16 x 3*16 für das 3x3-Spiel und der Pixelzahl 16 für eine Zelle) und müssen noch auf die Zelle [zeile,spalte] umgerechnet werden:

```
   % Zellen-Daten: ganzzahlig gerundete Werte aus x und y
   spalte = int32(x/16 + 0.5);
   zeile    = int32(y/16 + 0.5);
```

Nach dem Picken muss noch gecheckt werden, ob dieses Feld auch leer ist. Wenn ja, wird der aktuelle Spieler (steht in *t06_data.amzug*) in die Zelle des Feldes eingetragen.

```
    % Feld war frei und wird mit dem Spieler belegt, der am Zug ist:
    t06_data.feld( zeile, spalte ) = t06_data.amzug;
```

Wenn nicht, wird in einer Schleife weiter gepickt.

Aufgaben und Erweiterungen:
Zug-Mensch/Computer richtig einbauen, Meldungen einbauen

7.3 GUI, Zeichnen, Analyse

Menü:

- Datei: Neues Spiel / Beenden
- Hilfe: Info

Aufgaben von ***t06_draw***:

- Feld und Figuren zeichnen, Art je nach Option
- Analyse, ob nach dem Zeichnen ein Spieler gewonnen hat

```
function  t06_draw()

   global t06_data;    % Zugriff auf globale Variable t06_data

   t06_draw_lines();  % Feld-Linien in Image eintragen

   t06_draw_figures();  % alle Figuren in Image eintragen

   % vor dem image zur Sicherheit nach uint8 wandeln
   t06_data.imag = uint8( t06_data.imag );
   image( t06_data.imag )  % Und das Image anzeigen
   axis image

   % Check, ob es einen Sieger gibt
   fprintf('Debug: Hier fehlt Check, ob es einen Sieger gibt \n' );
return;
```

Figuren-Zeichnen-Funktion:
In einer Schleife für alle Zellen des Spielfeldes werden die Figuren der Spieler 1 oder 2 immer relativ zur logischen Zelle (m,n) gezeichnet. Dadurch kommt man mit nur einer einzigen Zeichnen-Funktion für alle Figuren aus. Hier als Beispiel eine ganz einfache Figur-Zeichnen-Funktion, die die Zellen nur einfarbig einfärbt:

```
function  t06_draw_figure( m, n, spieler )

   global t06_data;     % Zugriff auf globale Variable t06_data

   % als Dummy nur mit unterschiedlichen Farben belegen
   for( z=1 : 16 )
     for( s =1 : 16 )
       if( spieler == 1 )
         t06_data.imag( (m-1)*16 +z ,(n-1)*16 +s, 3) = 0;
       else
         t06_data.imag( (m-1)*16 +z ,(n-1)*16 +s, 2) = 0;
       end
     end
   end
return;
```

Vorschlag für Standard-Figuren: Spieler 1 als Kreuz / Spieler 2 als Raute
Farben und weitere Figuren sollten aber im Options-Dialog einstellbar sein.

Aufgaben und Erweiterungen:
Analyse, ob Spiel gewonnen ist,
Figuren-Typ etc zeichnen, Feld zeichnen,
Hilfe-Dialog mit Info über das Spiel und kurzer Erklärung der Regeln

7.4 Strategie 3x3

Strategie für den Computer-Zug im 3x3-Spielfeld, Analyse des Spielfeldes

```
function [z,s,ok] = t06_strategie_3_3()
```

Zurückgegeben werden Zeile z und Spalte s der automatisch bestimmten Zelle [z,s] und ob die Strategie überhaupt etwas liefern konnte: ok = 1. Bei ok = 0 hat die Strategie versagt.

Im M-File t06_strategie_3_3.m ist als Beispiel eine recht dumme Strategie implementiert, die die erste freie Zelle zurückgibt, die im Feld gefunden wurde.

Aufgaben und Erweiterungen:
Strategien entwickeln, es geht um das Prinzip, nicht um die Weltmeisterschaft

7.5 Strategie A 4x4

Strategie A für den Computer-Zug im 4x4-Spielfeld,
analog t06_strategie_3_3, nur jetzt für den Fall 4x4

```
function [z,s,ok] = t06_strategieA_4_4()
```

Aufgaben und Erweiterungen:
Strategien entwickeln, es geht um das Prinzip, nicht um die Weltmeisterschaft

7.6 Strategie B 4x4

Strategie B für den Computer-Zug im 4x4-Spielfeld,
analog t06_strategie_3_3, nur jetzt für den Fall 4x4

```
function [z,s,ok] = t06_strategieB_4_4()
```

Für den Fall 4x4 sollen zwei Strategien A und B entwickelt werden, die auch gegeneinander konkurrieren können.

Aufgaben und Erweiterungen:
Strategien entwickeln, es geht um das Prinzip, nicht um die Weltmeisterschaft

Hinweis:

In den Beispiel-Dateien zum Spiel sind einige Debug-Ausgaben eingebaut, die beim endgültigen Spiel natürlich entfernt werden müssen.