Aufgabe 1: Photo-Viewer

Den Bildbetrachter programmierte ich in der Skriptsprache Tcl/Tk. Er ist in der Lage, Bilder anzuzeigen, die in den Formaten JPEG, GIF oder PNG vorliegen. Sollten diese größer als der zur Verfügung stehende Anzeigebereich sein, so kann der Benutzer Scrollbars benutzen. Aus Geschwindigkeitsgründen wird immer nur das aktuelle Verzeichnis eingelesen, nicht das komplette Laufwerk. Das Programm erkennt selbstständig, welche Laufwerke auf dem System überhaupt zur Verfügung stehen.

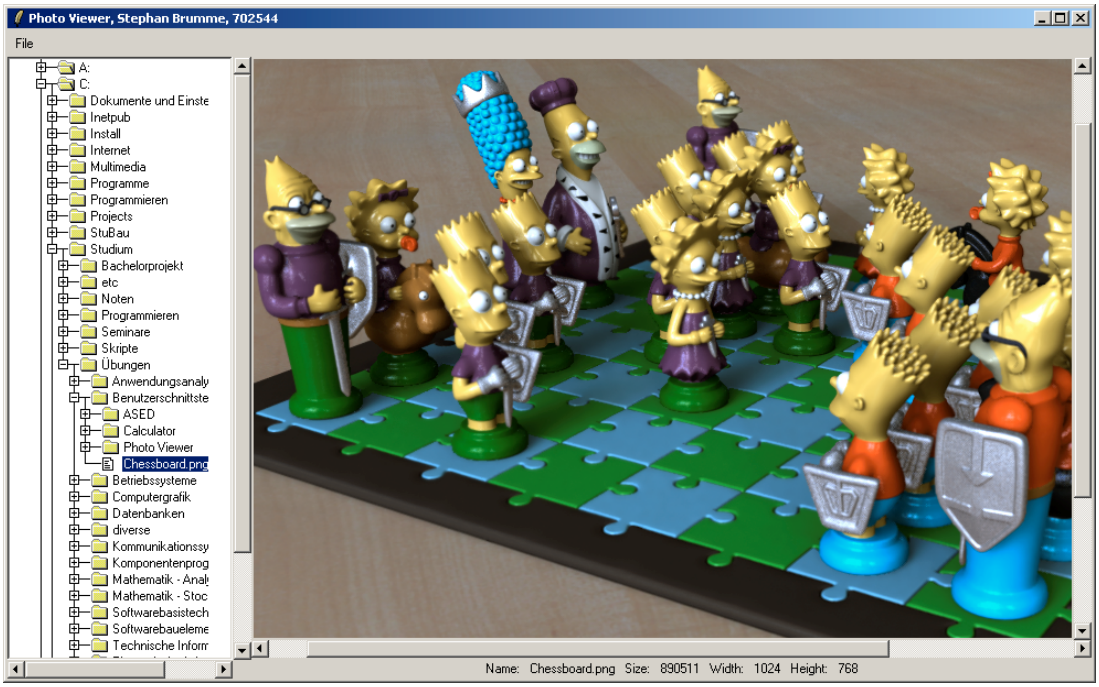

Abbildung 1: Screenshot Photo-Viewer

Grundsätzlich habe ich das Programm in zwei Teile untergliedert: die Dateinavigation auf der linken Seite (`FileTree` im Code genannt) und die Anzeuge eines Bildes auf der rechten Seite (`PictureCanvas`). Die Funktionalität beider Teile ist in Prozeduren enthalten, die jeweils mit den erwähnten Namen beginnen, z.B. `FileTree_Create`.

Beim Start des Programmes durchläuft die Tcl/Tk-Shell erstmal die Initialisierungs-Routinen FileTree_Create und `PictureCanvas_Create`. Sie sorgen dafür, dass die entsprechenden Widgets generiert und angezeigt werden. Dabei kommen auch die geforderten Features, wie Scrollbars und Bildinformationen, ins Spiel. Anschließend ist die Initialisierungs-Phase beendet und das Programm wartet nur noch auf Aktionen des Benutzers.

Die Suche nach Bilder in einem Verzeichnis erfolgt nur on-demand, d.h. erst wenn der Benutzer explizit ein Verzeichnis im Dateibaum öffnet. Die dafür zuständige Routine nennt sich `FileTree_ScanDirectory`. Sie sorgt u.a. auch dafür, dass Unterverzeichnisse und Bilder stets sortiert im Baum erscheinen, wobei aber alle Unterverzeichnisse vor allen Bildern dargestellt werden, was dem Standardverhalten des Windows Explorers entspricht.

Die Navigation durch den Benutzer ruft die Funktionen `FileTree_OnShowPicture` bzw. `FileTree_OnOpen` auf den Plan. Je nachdem, ob das zu öffnende Element ein Bild oder ein Unterzeichnis ist, verzweigt die Programmausführung dann weiter nach `PictureCanvas_Show` oder `FileTree_ScanDirectory`. Ersteres löscht ein eventuell vorher anzeigtes Bild vom Bildschirm, zeigt das neue an, kümmert sich um eine Anpassung der Scrollbalken an die neue Bildgröße und gibt Informationen über das Bild, wie z.B. Name, aus.

Quelltext

```
#####################################################
# Übung zur Vorlesung Benutzerschnittstellen
# Aufgabe 1: Photo-Viewer
#
# Autor: Stephan Brumme, 702544

# benoetigte Packages einbinden
package require BWidget
package require Img


#####################################################
# GUI-Elemente erzeugen und anordnen
wm title . "Photo Viewer, Stephan Brumme, 702544"
# Menü
menubutton .file
pack .file -side top -anchor w
# nur Eintrag "File"
.file configure -text File -menu .file.m
# nicht abreißbar
menu .file.m -tearoff 0
# "Programm beenden" erlauben
.file.m add command -label Quit -command { exit }


# ID zur Eindeutigkeit der Knoten im Dateibaum
set file_id 0

# Initialisierung und Anordnung
proc FileTree_Create { } {
    # Baum mit Scrollbars erzeugen
    ScrolledWindow .scrollabletree -relief sunken -borderwidth 2
    set tree [Tree .tree -borderwidth 0 -highlightthickness 0 \
            -opencmd "FileTree_OnOpen"]
    .scrollabletree setwidget $tree

    # links plazieren
    pack .scrollabletree -side left -fill y

    # alle Laufwerk erkennen
    foreach drive [file volume] {
        # und in den Baum einfügen
        .tree insert end root drive$drive \
                -text [string index $drive 0]: \
                -data $drive \
                -open 0 \
                -drawcross allways \
                -image [Bitmap::get openfold]
    }

    # Bild öffnen per Mausklick
    .tree configure -selectcommand "FileTree_OnShowPicture"
}


# ein Verzeichnis auslesen und in den Baum einfügen
proc FileTree_ScanDirectory { tree node path } {
    # nur neu lesen, wenn drawcross auf "allways"
    if {[$tree itemcget $node -drawcross] == "allways"} {
        # benötigte Variablen
        set list_of_directories {}
        set list_of_files {}
        global file_id

        # Verzeichnisse und Dateien erkennen
        foreach f [glob -nocomplain [file join $path "*"]] {
            set filename [file tail $f]
            if { [file isdirectory $f] } {
                lappend list_of_directories $filename
            } else {
                # nur gängige Bilder akzeptieren
                if { [regexp -nocase {(jpg$)|(jpeg$)|(gif$)|(png$)} $filename] } {
```

```
                    lappend list_of_files $filename
                }
            }
        }

        # Verzeichnisse sortiert ausgeben
        set sorted_directories [lsort -dictionary $list_of_directories]
        foreach filename $sorted_directories {
            $tree insert end $node node$file_id \
                        -text $filename \
                        -data "$path$filename\\" \
                        -image [Bitmap::get folder] \
                        -drawcross allways
            # neue eindeutige ID
            incr file_id
        }

        # Dateien sortiert ausgeben
        set sorted_files [lsort -dictionary $list_of_files]
        foreach filename $sorted_files {
            $tree insert end $node node$file_id \
                    -text $filename \
                    -data "$path$filename" \
                    -image [Bitmap::get file]
            # neue eindeutige ID
            incr file_id
        }

        # Teilbaum braucht nicht mehr neu eingelesen werden
        # markiert durch Änderung von drawcross in "auto""
        $tree itemconfigure $node -drawcross auto
    }
}


# Unterverzeichnis einlesen, wenn Teilbaum geöffnet wird
proc FileTree_OnOpen { node } {
    set subdir [.tree itemcget $node -data]
    FileTree_ScanDirectory .tree $node $subdir
}

# Bild anzeigen
proc FileTree_OnShowPicture { tree node } {
    set filename [$tree itemcget $node -data]
    # wenn Verzeichnis, dann wechseln
    if { [file isdirectory $filename] } {
        FileTree_OnOpen $node
        $tree opentree $node 0
    } else  {
    # sonst Bild anzeigen
        PictureCanvas_Show $filename
    }
}


# Initialisierung und Anordnung
proc PictureCanvas_Create { } {
    # Widgets erzeugen
    canvas .view
    label  .info

    # Scrollbars für die Bildanzeige (namens .view)
    scrollbar .viewhscrollbar -command ".view xview" -orient horiz
    scrollbar .viewvscrollbar -command ".view yview" -orient vertical
    .view configure -xscrollcommand ".viewhscrollbar set" \
                    -yscrollcommand ".viewvscrollbar set"

    # und alles anzeigen
    pack .info -side bottom -fill x
    pack .viewhscrollbar -side bottom -fill x
    pack .viewvscrollbar -side right -fill y
    pack .view -side top -fill both -expand 1
}

# Bild laden und anzeigen
proc PictureCanvas_Show { filename } {
```

```
    # Bild laden
    set handle [image create photo -file $filename]
    # anzeigen (vorher altes Bild löschen)
    .view delete all
    .view create image 0 0 -image $handle -anchor nw
    # Informationen extrahieren
    set img_width [image width $handle]
    set img_height [image height $handle]
    set img_filesize [file size $filename]
    set img_name [file tail $filename]
    # Scrollbars anpassen
    .view configure -scrollregion "0 0 $img_width $img_height"

    # und die Infos anzeigen
    .info configure -text \
            "Name:   $img_name   Size:   $img_filesize   Width:   $img_width   Height:
$img_height"
}


# Baum zur Navigation im Dateisystem erzeugen
FileTree_Create
# Vorschaufenster erzeugen
PictureCanvas_Create
```

Aufgabe 2: Virtueller Taschenrechner

Mein Taschenrechner verfügt über alle drei geforderten Modi, die Funktionalität orientiert sich an den im Aufgabenzettel dargestellten Abbildungen. Die Bildschirmfotos meiner Lösung sind den Originalen sehr ähnlich:

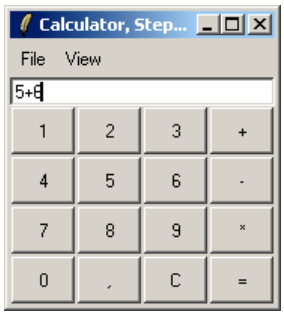

Abbildung 2: Einfacher Taschenrechner

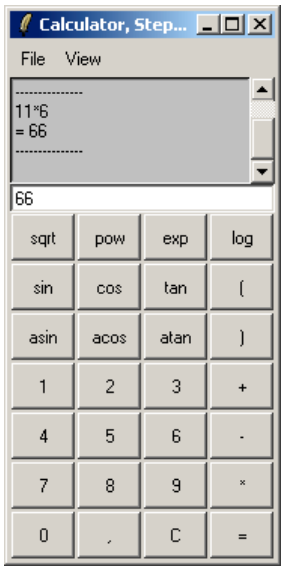

Abbildung 3: Erweiterter Taschenrechner

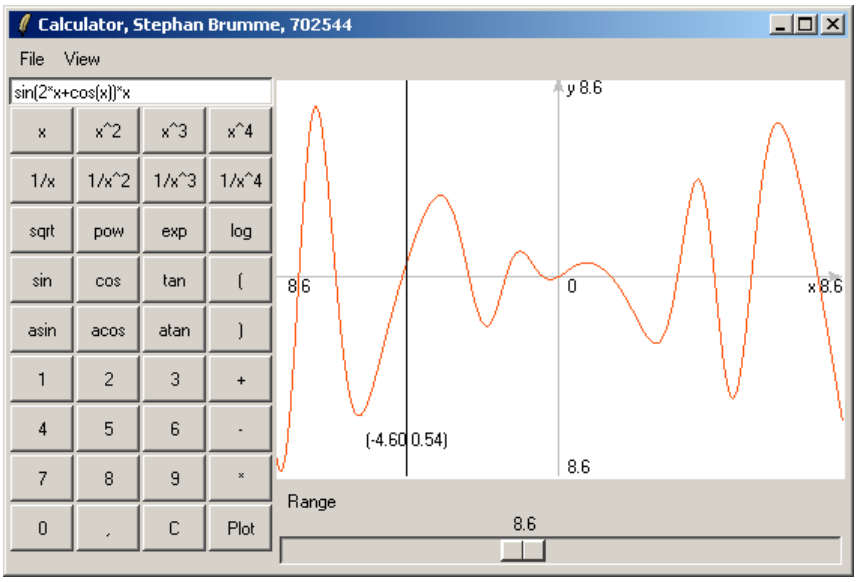

Abbildung 4: Funktionsplotter

Alle Funktionsmodi verfügen über eine Methode `Create`, z.B. `Simple_Create`. Diese ist verantwortlich für die Erstellung der Benutzeroberfläche. Dazu gehört neben dem Anlegen neuer Bedienelemente auch das Bereinigen der alten Widgets des vorherigen Funktionsmodus.

Das Menü enthält stets die Einträge „File/Load“, „File/Save“, „File/Quit“, „View/Simple“, „View/Advanced“ und „View/Plotter“. Allerdings ist das Laden und Speichern nur im erweiteren Modus möglich, sonst sind diese Einträge deaktiviert, also grau und nicht anklickbar. Ein kleines Häkchen symbolisiert im View-Menü den momentan gewählten Modus.

Die Eingabezeile ist stets mit der Variablen `$formula` verknüpft. Fehlerhafte Eingaben führen zu einem entsprechenden Hinweis. Der Funktionsplotter erlaubt eine interaktive Eingabe, d.h. er zeichnet die sofort die Funktion, wenn sie ein gültiger mathematischer Ausdruck ist. Eine explizite Betätigung der „Plot“-Taste ist nicht erforderlich. Zusätzlich darf statt `$x` auch nur `x` verwendet werden. Der interne Parser passt die Eingabe dann selbstständig an das für Tcl/Tk notwendige Format an.

Quelltext

```
##################################################
# Übung zur Vorlesung Benutzerschnittstellen
# Aufgabe 2: Virtueller Taschenrechner
#
# Autor: Stephan Brumme, 702544


# GUI aufräumen
proc Menu_Destroy {} {
    # alte Widgets zerstören
    destroy .input
    destroy .btn
    destroy .plot

    # Menüeintrage Load&Save sperren
    .menuframe.menufile.m entryconfigure 0 -state disabled
    .menuframe.menufile.m entryconfigure 1 -state disabled
}



proc Simple_Create {} {
    Menu_Destroy

    # Eingabefeld anlegen (Wert steht in $formula)
    entry .input -textvariable formula
    pack  .input -side top -fill x

    # Buttons anlegen
    frame .btn
    pack .btn -fill both

    set row 0
    foreach buttonrow {{1 2 3 +} {4 5 6 -} {7 8 9 *} {0 , C =}} {
        # zeilenweise erstellen
        set frm [frame .btn.frm$row]
        pack $frm
        incr row

        # Button erzeugen
        foreach singlebutton $buttonrow {
            button $frm.btn$singlebutton -text $singlebutton \
                    -command "append formula $singlebutton" \
                    -width 4 \
                    -height 1 \
                    -padx 5 \
                    -pady 5
            pack $frm.btn$singlebutton -side left -padx 1 -pady 1
        }
    }

    # Ausnahmen korrigieren
    $frm.btn, configure -command "append formula ."
    $frm.btn= configure -command {set formula [expr $formula]}
    $frm.btnC configure -command {set formula ""}
}


# History laden
proc Load {} {
    set hfile [open [tk_getOpenFile] r]
    .input.history insert end [read $hfile]
    close $hfile
}
```

```
# History speichern
proc Save {} {
    set hfile [open [tk_getSaveFile] w]
    puts $hfile [.input.history get 0.0 end]
    close $hfile
}

proc Advanced_Create {} {
    Menu_Destroy

    # Eingabefeld anlegen (Wert steht in $formula)
    frame .input
    pack  .input -side top -fill x

    # History soll schreibgeschützt sein => grau und Events abfangen
    text  .input.history -height 5 -width 25 -background gray
    bind  .input.history <Key> { break }
    entry .input.current -textvariable formula
    # Scrollbar für die History
    scrollbar .input.vscrollbar -command ".input.history yview" -orient vertical
    .input.history configure -yscrollcommand ".input.vscrollbar set"

    # alles darstellen
    pack  .input.current -side bottom -fill x
    pack  .input.vscrollbar -side right -fill y
    pack  .input.history -fill x

    # Menüeintrage Load&Save freischalten
    .menuframe.menufile.m entryconfigure 0 -state normal
    .menuframe.menufile.m entryconfigure 1 -state normal

    # Buttons anlegen
    frame .btn
    pack .btn -fill both

    set row 0
    foreach buttonrow {{sqrt pow exp log} {sin cos tan \(} {asin acos atan \)} \
                {1 2 3 +} {4 5 6 -} {7 8 9 *} {0 , C =}} {
        # zeilenweise erstellen
        set frm [frame .btn.frm$row]
        pack $frm
        incr row

        # Button erzeugen
        foreach singlebutton $buttonrow {
            button $frm.btn$singlebutton -text $singlebutton \
                    -command "append formula {$singlebutton}" \
                    -width 4 \
                    -height 1 \
                    -padx 5 \
                    -pady 5
            pack $frm.btn$singlebutton -side left -padx 1 -pady 1
        }
    }

    # Ausnahmen korrigieren
    $frm.btn, configure -command "append formula ."
    $frm.btn= configure -command {Advanced_Evaluate}
    $frm.btnC configure -command {set formula ""}
}


# Formelausdruck auswerten und in die History einfügen
proc Advanced_Evaluate {} {
    global formula

    # auswerten
    .input.history insert end $formula
    set formula [expr $formula]

    # Ergebnis zur History hinzufügen und ggf. Anzeige scrollen
    .input.history insert end "\n= $formula\n---------------\n"
    .input.history see end
}
```

```
set points ""
set datasize 100
set formulaplot ""

# Initialsierung
proc Plotter_Create {} {
    Menu_Destroy

    # Plotterbereich
    frame .plot
    pack .plot -side right -fill both
    canvas .plot.function -bg white
    scale .plot.range -label Range -orient horizontal \
            -variable range \
            -resolution 0.1 -from 0.1 -to 20 \
            -command "Plotter_Scale"
    .plot.range set 1.0
    pack .plot.range -side bottom -fill x
    pack .plot.function -fill both


    # Eingabefeld anlegen (Wert steht in $formula)
    global formulaplot
    set formula $formulaplot
    entry .input -textvariable formula \
            -validatecommand { Plotter_Update %P; return 1 } \
            -validate all
    pack  .input -side top -fill x

    # Buttons anlegen
    frame .btn
    pack .btn

    set row 0
    foreach buttonrow {{x x^2 x^3 x^4} {1/x 1/x^2 1/x^3 1/x^4} \
                {sqrt pow exp log} {sin cos tan \(} {asin acos atan \)} \
                {1 2 3 +} {4 5 6 -} {7 8 9 *} {0 , C Plot}} {
        # zeilenweise erstellen
        set frm [frame .btn.frm$row]
        pack $frm
        incr row

        # Button erzeugen
        foreach singlebutton $buttonrow {
            # Dollarzeichen nicht bei Button-Beschriftung zeigen
            button $frm.btn$singlebutton -text $singlebutton \
                    -command "append formula {$singlebutton}" \
                    -width 4 \
                    -height 1 \
                    -padx 5 \
                    -pady 5
            pack $frm.btn$singlebutton -side left -padx 1 -pady 1
        }
    }

    # Ausnahmen korrigieren
    $frm.btn, configure -command "append formula ."
    $frm.btnPlot configure -command {Plotter_Evaluate}
    $frm.btnC configure -command {set formula ""}
    .btn.frm0.btnx^2 configure -command "append formula pow(x,2)"
    .btn.frm0.btnx^3 configure -command "append formula pow(x,3)"
    .btn.frm0.btnx^4 configure -command "append formula pow(x,4)"
    .btn.frm1.btn1/x^2 configure -command "append formula 1/pow(x,2)"
    .btn.frm1.btn1/x^3 configure -command "append formula 1/pow(x,3)"
    .btn.frm1.btn1/x^4 configure -command "append formula 1/pow(x,4)"

    # Mausbewegung abfangen
    bind .plot.function <Motion> { Plotter_Tooltip %x %y }
}


# Formelausdruck auswerten
proc Plotter_Evaluate {} {
    global formulaplot
    global range
```

```
    # Offset zwischen zwei x-Werten (entspr. 1 Pixel)
    global datasize
    set plot_width  [lindex [.plot.function configure -width] 4]
    set datasize $plot_width
    set dataoffset [expr 2*$range/($datasize-1)]

    # Kurve ermitteln
    global points
    set points ""
    set x [expr -$range]
    for {set i 0} {$i < $datasize} {incr i} {
        # Punkt berechnen (ungültige Berechnungen abfangen)
        if { [catch { set y [expr $formulaplot] }] } {
            set y 0
        }
        lappend points $x $y

        set x [expr $x+$dataoffset]
    }

    Plotter_Draw
}


# und alles zeichnen
proc Plotter_Draw {} {
    global range

    # Canvas löschen
    .plot.function delete all

    # Ausmaße des Plot-Bereiches ermitteln
    set plot_width  [lindex [.plot.function configure -width] 4]
    set plot_height [lindex [.plot.function configure -height] 4]

    set left [expr -$range]
    set right $range
    set top $range
    set bottom [expr -$range]

    # Achsen zeichnen
    .plot.function create line $left 0 $right 0 \
            -fill gray -arrow last
    .plot.function create line 0 $top 0 $bottom \
            -fill gray -arrow first
    # Beschriftung
    .plot.function create text $range 0 -text "  x $range" -anchor ne
    .plot.function create text -$range 0 -text "   $range" -anchor nw
    .plot.function create text 0 $range -text "  y $range" -anchor nw
    .plot.function create text 0 -$range -text "  $range" -anchor sw
    .plot.function create text 0 0 -text "  0" -anchor nw

    # Kurve zeichnen
    global points
    global datasize
    eval .plot.function create line $points -fill red

    # alles skalieren und verschieben
    set scalex [expr 0.5*$plot_width/$range]
    set scaley [expr 0.5*$plot_height/$range]
    .plot.function scale all 0 0 $scalex -$scaley
    .plot.function move  all [expr $plot_width/2] [expr $plot_height/2]
}


# Wrapper für Skalierungsevents
proc Plotter_Scale {width} {
    Plotter_Evaluate
}


# Tooltip über der Zeichenfläche anzeigen
set cursor_id 0
set text_id 0
proc Plotter_Tooltip { eventx eventy } {
    global cursor_id
```

```
    global text_id
    global range
    global formulaplot

    # vertikale Linie
    .plot.function delete $cursor_id
    set cursor_id [.plot.function create line $eventx 0 $eventx [lindex [.plot.function
configure -height] 4] -fill black]

    # Funktionswert an aktueller Cursorposition
    set plot_width  [lindex [.plot.function configure -width] 4]
    set x [expr 2*$eventx*$range/$plot_width-$range]
    .plot.function delete $text_id
    if {![catch { set y [expr $formulaplot] }]} {
        set ausgabe [format "(%.2f %.2f)" $x $y]
        set eventy [expr $eventy-5]
        # ... ausgeben
        set text_id [.plot.function create text $eventx $eventy -text $ausgabe]
    }
}


# während der Formeleingabe Graphen aktualisieren
proc Plotter_Update { newformula } {
    global formulaplot
    # sowohl x als auch $x erlauben
    regsub -all {\$x} $newformula {x} formulaplot
    regsub -all {\$} $formulaplot {x} formulaplot
    regsub -all {($x|x)} $formulaplot {$x} formulaplot
    # Graph neu berechnen
    Plotter_Evaluate
}


# Initialisierung
proc Menu_Create { } {
    # Frame zur Positionierung des Menüs
    frame .menuframe
    pack .menuframe -side top -fill x
    # Menü "File"
    menubutton .menuframe.menufile -text File -menu .menuframe.menufile.m
    pack .menuframe.menufile -side left -anchor w
    # nicht abreißbar
    menu .menuframe.menufile.m -tearoff 0
    # "Programm beenden" erlauben, den Rest deaktivieren (da Simple bei Beginn)
    .menuframe.menufile.m add command -label Load -command { Load } -state disabled
    .menuframe.menufile.m add command -label Save -command { Save } -state disabled
    .menuframe.menufile.m add separator
    .menuframe.menufile.m add command -label Quit -command { exit }

    # Menü "View"
    menubutton .menuframe.menuview -text View -menu .menuframe.menuview.m
    pack .menuframe.menuview -side left -anchor w
    # nicht abreißbar
    menu .menuframe.menuview.m -tearoff 0
    # alle 3 Darstellungsarten
    .menuframe.menuview.m add radio -label Simple   -command { Simple_Create }
    .menuframe.menuview.m add radio -label Advanced -command { Advanced_Create }
    .menuframe.menuview.m add radio -label Plotter  -command { Plotter_Create }
    # Simple aktivieren
    .menuframe.menuview.m invoke 0
}



#####################################################
# GUI-Elemente erzeugen und anordnen
wm title . "Calculator, Stephan Brumme, 702544"
# Menü (startet automatisch Minimalfassung)
Menu_Create

set formula 0
```