Grundlagen Programmierung mit Python

**Inhalt**

[Anwenderprogramme (Software) 6](#_Toc185241537)

[Python Entwicklungsumgebung (user interface) 7](#_Toc185241538)

[Programmiersprachen 8](#_Toc185241539)

[Maschinensprache 8](#_Toc185241540)

[Assembler 8](#_Toc185241541)

[Hochsprache 8](#_Toc185241542)

[Grundlagen Programmierung (coding basics) 9](#_Toc185241543)

[Variablen 9](#_Toc185241544)

[Operatoren 9](#_Toc185241545)

[mathematische 9](#_Toc185241546)

[logische 9](#_Toc185241547)

[Zuweisungen 10](#_Toc185241548)

[= += -= \*= /= //= %= \*\*= 10](#_Toc185241549)

[Ausführungsregeln 10](#_Toc185241550)

[Bedingungen 10](#_Toc185241551)

[Schleifen 11](#_Toc185241552)

[Iterative Schleife (for loop) 11](#_Toc185241553)

[Bedingte Schleife (while loop) 11](#_Toc185241554)

[Funktionen 12](#_Toc185241555)

[Grafische Elemente 13](#_Toc185241556)

[Plot 13](#_Toc185241557)

[Beispiel Sinusfunktion: 13](#_Toc185241558)

[Beispiel: Funktion - QuadraticPolynomial 13](#_Toc185241559)

[Beispiel Funktion: CubicPolynomial 14](#_Toc185241560)

[Beispiel Funktion: NthRoot 14](#_Toc185241561)

[Linie (Line) 15](#_Toc185241562)

[Beispiel Funktion: DrawLines 15](#_Toc185241563)

[Rechteck (Rectangle) 15](#_Toc185241564)

[Beispiel Funktion: DrawRectangle 16](#_Toc185241565)

[Beispiel: DrawCircle 16](#_Toc185241566)

[Rotation von Grafikelementen 18](#_Toc185241567)

[Beispiel Funktion: RotateLine 18](#_Toc185241568)

[Aufgabe: Das Haus vom Nikolaus zeichnen 19](#_Toc185241569)

[Aufgabe: Hypnotische Kreise 20](#_Toc185241570)

[Klassen 24](#_Toc185241571)

[GUI erzeugen 26](#_Toc185241572)

[Beispiel Hello World 26](#_Toc185241573)

[Aufgabe: Kreis zwischen Schaltflächen 29](#_Toc185241574)

[Aufgabe: Lokomotive vorwärts rückwärts bewegen 32](#_Toc185241575)

[Aufgabe: Windrad 36](#_Toc185241576)

[Lösungsansatz 36](#_Toc185241577)

[Lösung Matlab Aufgabe: Windrad 37](#_Toc185241578)

[Aufgabe: Generischer Baum 42](#_Toc185241579)

[Lösungsansatz 42](#_Toc185241580)

[Lösung Matlab Aufgabe: Generischer Baum 43](#_Toc185241581)

[Aufgabe: Konfigurator für einfaches Haus 45](#_Toc185241582)

[Lösungsansatz 45](#_Toc185241583)

[Lösung Matlab Aufgabe: Konfigurator für einfaches Haus 45](#_Toc185241584)

[Opening Function 45](#_Toc185241585)

[StartButton-Callback 46](#_Toc185241586)

[Klasse: clsSimpleHouse 46](#_Toc185241587)

[Aufgabe: Eine Klasse für „Windrad“ anlegen 48](#_Toc185241588)

[Lösungsansatz 48](#_Toc185241589)

[Lösung Matlab Aufgabe: 49](#_Toc185241590)

[Aufgabe: Kisten-Stapel Konfigurator 60](#_Toc185241591)

[Lösungsansatz 60](#_Toc185241592)

[Lösung Matlab Aufgabe: 61](#_Toc185241593)

[Aufgabe: Quadrat auf Treppe Konfigurator 65](#_Toc185241594)

[Aufgabe: Pyramiden-Konfigurator 69](#_Toc185241595)

[Lösungsansatz 69](#_Toc185241596)

[Lösung Matlab Aufgabe: PyramidConfigurator 69](#_Toc185241597)

[Aufgabe: Treppen-Konfigurator 77](#_Toc185241598)

[Lösungsansatz 77](#_Toc185241599)

[Lösung Matlab Aufgabe: 77](#_Toc185241600)

[Aufgabe ShopFloorConfigurator 81](#_Toc185241601)

[Lösung Matlab Aufgabe: ShopFloorConfigurator 82](#_Toc185241602)

[Aufgabe: Haus-Konfigurator 86](#_Toc185241603)

[Lösungsansatz 86](#_Toc185241604)

[Lösung Matlab Aufgabe: HouseConfigurator 88](#_Toc185241605)

[Aufgabe Portal-Konfigurator 96](#_Toc185241606)

[Aufgabe Pontonbrücken-Konfigurator 97](#_Toc185241607)

[Aufgabe: Draw-Bridge 101](#_Toc185241608)

[Aufgabe: Slot-Machine 103](#_Toc185241609)

[Lösungsansatz 103](#_Toc185241610)

[Lösung Matlab Aufgabe: 103](#_Toc185241611)

[Aufgabe: GamblingGenerator 104](#_Toc185241612)

[Lösungsansatz 104](#_Toc185241613)

[Lösung Matlab Aufgabe: 104](#_Toc185241614)

[Aufgabe: Elevator 105](#_Toc185241615)

[Lösungsansatz 105](#_Toc185241616)

[Lösung Matlab Aufgabe: 105](#_Toc185241617)

[Aufgabe Labyrinth (Maze) 106](#_Toc185241618)

[Lösungsansatz 106](#_Toc185241619)

[Lösung Aufgabe Schatzsuche: 106](#_Toc185241620)

[Info 107](#_Toc185241621)

# Anwenderprogramme (Software)

Anwender haben es leicht, weil irgendjemand schon unterschiedliche Programme (Instruktionen) hineingestopft hat, er muss nur noch dasjenige herauspicken, dass er benutzen möchte: Textverarbeitung, Tabellenkalkulation, Navigation …

Computer werden zu dem Zweck gebaut, etwas für uns zu tun, aber sie sind nicht besonders intelligent, deshalb müssen wir in ihrer Sprache genau beschreiben, was wir tun wollen.

Programmierer müssen die Arbeitsweise des Computers und die Programmiersprache verstehen. Sie schreiben Programme, um Probleme zu lösen oder um Aufgaben zu automatisieren.

Sie haben Werkzeuge, um Programme zu schreiben. Sie schreiben Programme für viele Anwender oder Werkzeuge, um sich selbst die Arbeit zu erleichtern.

Unser Smartphone wartet auf die nächste Anforderung von uns, sobald wir es einschalten. Wir schauen auf das Smartphone von außen.

Als Programmierer erzeugt man etwas im Inneren (Memory) und seine Ergebnisse werden nach außen auf den Screen projiziert. Dabei ist man in Kontakt zu den inneren Komponenten wie Input-Devices (Keyboard, Mouse, Touch-Screen), Output-Devices (Screen, Speakers, Printer), CPU, Memory, Storage-System (primary RAM benötigt Strom, secondary HDD kein Stom), Network-Connection …

Code ist eine Sequenz von gespeicherten Instruktionen. Sie muss sehr präzise sein, Fehler werden vom Computer nicht toleriert. Das kann zunächst frustrieren.

Ein Bild, das Text, Screenshot, Diagramm, Schrift enthält.

Automatisch generierte Beschreibung

Komponenten befinden sich auf dem Motherboard.

Eine Programmiersprache wird in „Machine-Language“ übersetzt (Nullen und Einsen) und zur CPU gesendet.

# Python Entwicklungsumgebung (user interface)

Beschreibung der Oberfläche des verwendeten Programmier-Tools.

Ein Bild, das Screenshot, Text, Multimedia-Software, Grafiksoftware enthält.

Automatisch generierte Beschreibung

# Programmiersprachen

## Maschinensprache

## Assembler

## Hochsprache

# Grundlagen Programmierung (coding basics)

Python ist eine Skript-Programmiersprache für beliebig komplexe Aufgaben und die dazu gehörende grafische Visualisierung. Dieses Tutorial konzentriert sich jedoch nur auf die Möglichkeit der einfachen Programmierung.

Programme werden für Anwender geschrieben. Der Anwender bedient das Programm vor seinem Bildschirm. Das Programm befindet sich jedoch in dem Rechner und der Programmierer steuert den Anwender von innen heraus.

## Variablen

Variablen sind benannte Container, denen Informationen zugewiesen werden können. Die Inhalte können durch Eingabe des Variablennamens wieder zurückgeholt werden.

Stellen wir uns eine Box vor. Ich füge einen Inhalt hinzu, diesen USB-Stick, und stelle die Box beiseite. Benötige ich den Inhalt, so öffne ich die Box und finde meinen USB-Stick.

Beispiel eine Box mit Inhalt => myBox=“USB-Stick“

**Variablentypen**:

String (z.B. myText=”Breite”), Integer (z.B. i=1), Numeric (z.B. hight=2.5), Array(s.o.), List (z.B. startXY = [0 0])

myNumber = 10.5

myList = [10 20 30]

myList(2)

20

myArray(1) = 10

myArray(2) =20

myArray(3) = 30

hight = 12

width = 15

depth = 55

hight \* width \* depth

## Operatoren

### mathematische

***Mathematisch:***  + - \* / \*\* // %(a,b)

Multiplikation: hight \* width

Division: 24/6

Addition, Subtraktion: hight + width / depth

Potenzieren: 2\*\*4

### logische

Logische Operatoren ergeben als Ergebnisse wahr 1 (true) oder falsch 0 (false)

< <= > >= != == (identisch)

12<4 => 0 (false)

12>4 => 1 (true)

12>3&4>3 => 1 (true)

12>3|3>4 => 1 (true)

Not(12>3) => 0 (false)

*logisch* ==(equal) <(less than) >(greater than) and or not

### Zuweisungen

### = += -= \*= /= //= %= \*\*=

## Ausführungsregeln

Ausführungsregeln bestimmen die Reihenfolge der Abarbeitung innerhalb einer Programmzeile: (12/3) \* (12 \* 3 + 2\*\*4)

1. Klammern

2. Exponent

3. Multiplikation, Division

4. Addition, Subtraktion

5. links n. rechts

## Bedingungen

Bedingungen sind Verzweigung im Programmablauf, die von einem logischen Operator

abhängig sind. Die einfachste Bedingung ist die **if then else** Bedingung. (auch mit Script!)

**Beispiel**

x = 8

print

if x < 2 :

    print ("very small")

elif x < 5 :

    print ("still small")

elif x < 10 :

    print ("medium")

else :

    print("big")

Eine Bedingung für beliebige Verzweigungen ist die **match case** Bedingung.

**Beispiel Obstpreise**

term = 'Kiwi'

match term:

    case 'Orangen':

         print(f'{term} kosten xx')

    case 'Papaya'|'Kiwi':

         print(f'{term} kosten xx')

    case 'Bananen':

         print(f'{term} kosten xx')

    case \_:

        print('{term} haben wir heute nicht vorrätig')

end

## Schleifen

Schleifen sind sich wiederholende Programmabschnitte, entweder abhängig von einem Zähler oder einer logischen Bedingung.

Wir beginnen mit der zählerabhängigen Variante, der FOR LOOP.

### Iterative Schleife (for loop)

*for variable in list*

*print(variable)*

**Beispiele**

for x in range(2, 30, 3):

  print(x)

### Bedingte Schleife (while loop)

*while Bedingung*

*code*

**Beispiel**

i = 1

while i < 6:

  print(i)

  i += 1

## Funktionen

Funktionen sind separierte Programmteile, die Werte aufnehmen können, damit einen Programmabschnitt durchlaufen und ggf. ein Ergebnis zurückgeben.

Definition:

*function returnValue = functionName(attribute1, attribute2, …)*

*Code*

*End*

**Einfaches Beispiel “addValues”**

function summery = addValues(valueA, valueB)

summery = valueA + valueB;

end

*Aufruf*:

>>function addValues(12, 27)

**Beispiel “average”**

function y = average(x)

if ~isvector(x)

error('input must be a vector')

elseif ~isnumeric(x)

error('input must be a numeric value')

else

y = sum(x)/length(x);

end

end

*Aufruf*:

>>z = [10 20 30 40 75 98 22 12];

>>average(z)

Fehlermeldung Testen:

>>average(‚hello‘)

# Grafische Elemente

## Plot

x = x value or range of math. function

y = y value of math. function

**plot(x, y)**

Beispiele Linientypen: ‘--’, ‘:’, ‘\*’

**plot(x, y, ‘--’)**

Beispiele Farben: ‚g‘, ‚b‘, ‚y‘, ‚c‘

**plot(x, y, ‚g‘)**

### Beispiel Sinusfunktion:

from matplotlib import pyplot as plt

import numpy as np

x = np.linspace(0, 2\*np.pi, 100)

y = np.sin(x)

plt.plot(x,y)

Ein Bild, das Diagramm, Text, Reihe, Steigung enthält.

Automatisch generierte Beschreibung

### Beispiel: Funktion - QuadraticPolynomial

**Spezifikation**: Mit Hilfe einer Funktion (QuadraticPolynomial) soll ein quadratisches Polynom als Kurve erzeugt werden. Die Werte a, b und c sollen der Funktion übergeben werden.

def quadraticPolynomial(a, b, c):

    x = np.linspace(-10, 10, 100)

    y = a\*x\*\*2 + b\*x + c

    plt.plot(x,y)

    plt.show()

quadraticPolynomial(2, 2, 5)

Ein Bild, das Diagramm, Reihe, Text, Screenshot enthält.

Automatisch generierte Beschreibung

### Beispiel Funktion: CubicPolynomial

**Spezifikation**: Mit Hilfe einer Funktion (CubicPolynomial) soll ein kubisches Polynom als Kurve erzeugt werden. Die Werte a, b, c und d sollen der Funktion übergeben werden.

def qubicPolynomial(a, b, c, d):

    x = np.linspace(-10, 10, 100)

    y = a\*x\*\*3 + b\*x\*\*2 + c\*x + d

    plt.plot(x,y)

qubicPolynomial(2, 2, 5, 5)

Ein Bild, das Text, Diagramm, Reihe, Screenshot enthält.

Automatisch generierte Beschreibung

### Beispiel Funktion: NthRoot

**Spezifikation**: Mit Hilfe einer Funktion (NthRoot) soll die n’te Wurzel als Kurve erzeugt werden. Der Wert exponent sollen der Funktion übergeben werden.

def nthRoot(exponent):

    x = np.linspace(-10, 10, 100)

    y = np.power(x, (1/exponent))

    plt.plot(x,y)

nthRoot(3)

Ein Bild, das Reihe, Diagramm, Text, Screenshot enthält.

Automatisch generierte Beschreibung

## Linie (Line)

**Linie allgemein**

x = x value or xvalues of line

y = y value or yvalues of line

line color ‘Color’ = r g b => red, green, blue value between 0 – 1 or ‘b’(blue); ‘g’(green); ‘y’(yellow); ‘m’(magenta); ‘r’(red);

line width ‘LineWidth’ = decimal value (pixel)

### Beispiel Funktion: DrawLines

**Spezifikation**: Mit Hilfe einer Funktion (DrawLines) sollen 3 verbundene blaue Linien mit Linienstärke 3 in Form eines Galgens mit den folgenden festen Koordinaten erzeugt werden: 0,0 – 0,20 – 10,20 – 10,15

function DrawLines()

xValues = [0 0 10 10];

yValues = [0 20 20 15];

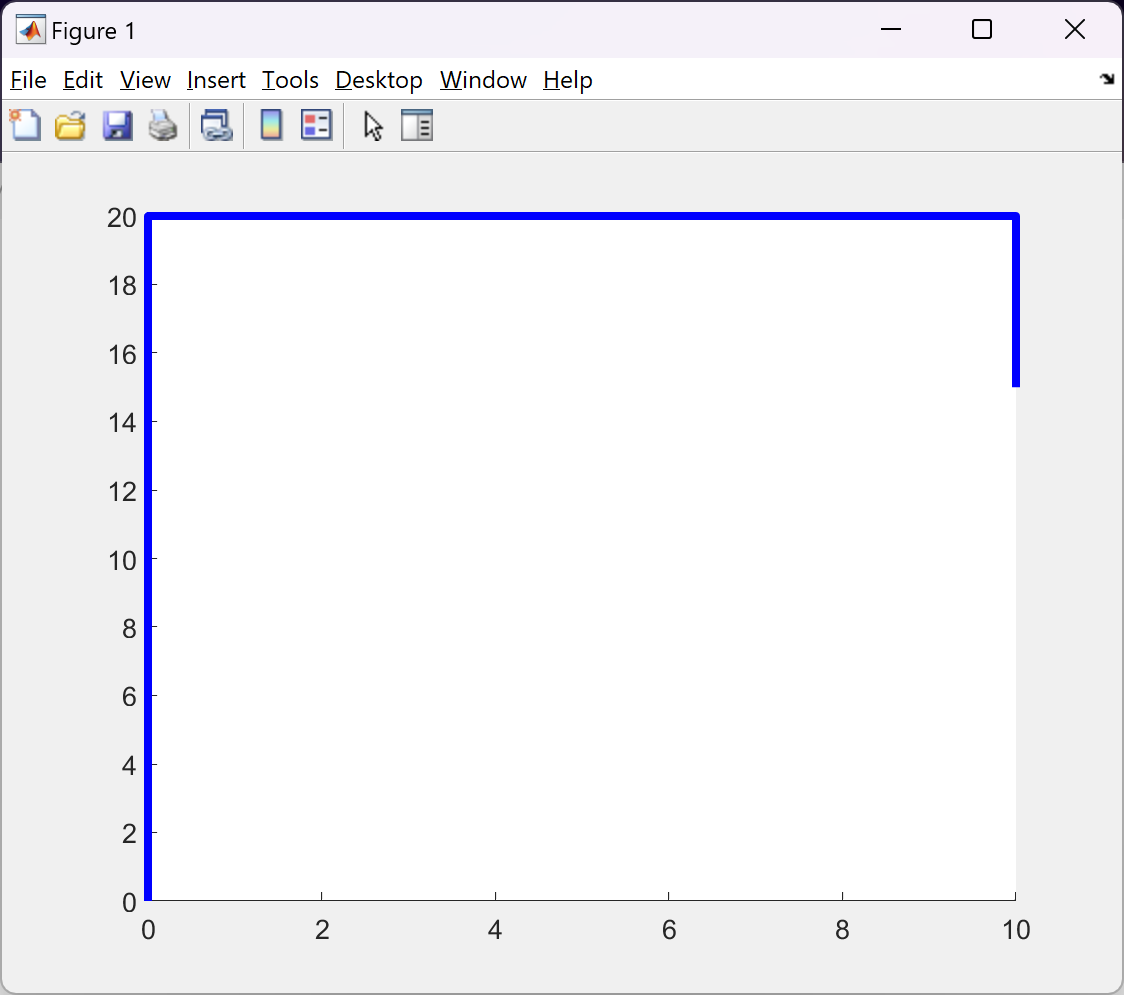
lineColor = 'b';

lineWidth = 3;

line(xValues, yValues, 'Color', lineColor, 'LineWidth', lineWidth);

end

*Aufruf*: DrawLines;



## Rechteck (Rectangle)

**Rechteck allgemein**

x = rectangle lower left x value

y = rectangle lower left y value

w = rectangle width

h = rectangle height

line width ‘LineWidth’ = decimal value (pixel)

rounding ‘Curvature’ = horizontal edge curving between 0 – 1, vertical edge curving between 0 – 1

edge color ‘EdgeColor’ = r g b => red, green, blue value between 0 – 1 or ‘b’(blue); ‘g’(green); ‘y’(yellow); ‘m’(magenta); ‘r’(red);

face color ‘FaceColor’ = r g b => red, green, blue value between 0 – 1 or ‘b’(blue); ‘g’(green); ‘y’(yellow); ‘m’(magenta); ‘r’(red);

rectangle(‘Position’, [x y w h], ‘Curvature’, [hor ver], ’EdgeColor‘, ‘red‘, ‘FaceColor’, [r g b], ‘LineWidth’, lw)

### Beispiel Funktion: DrawRectangle

**Spezifikation**: Mit Hilfe einer Funktion (DrawRectangle) soll ein gelbes Rechteck mit grünem Rand und Linienstärke 3 erzeugt werden. Die Breite und Höhe sollen der Funktion übergeben werden.

function DrawRectangle(width, height)

lowerLeftXY = [0 0];

deltaXY = [width height];

faceColor = 'y';

edgeColor = 'g';

lineWidth = 3;

rectangle( ...

'Position', [lowerLeftXY deltaXY], ...

'FaceColor', faceColor, ...

'EdgeColor', edgeColor, ...

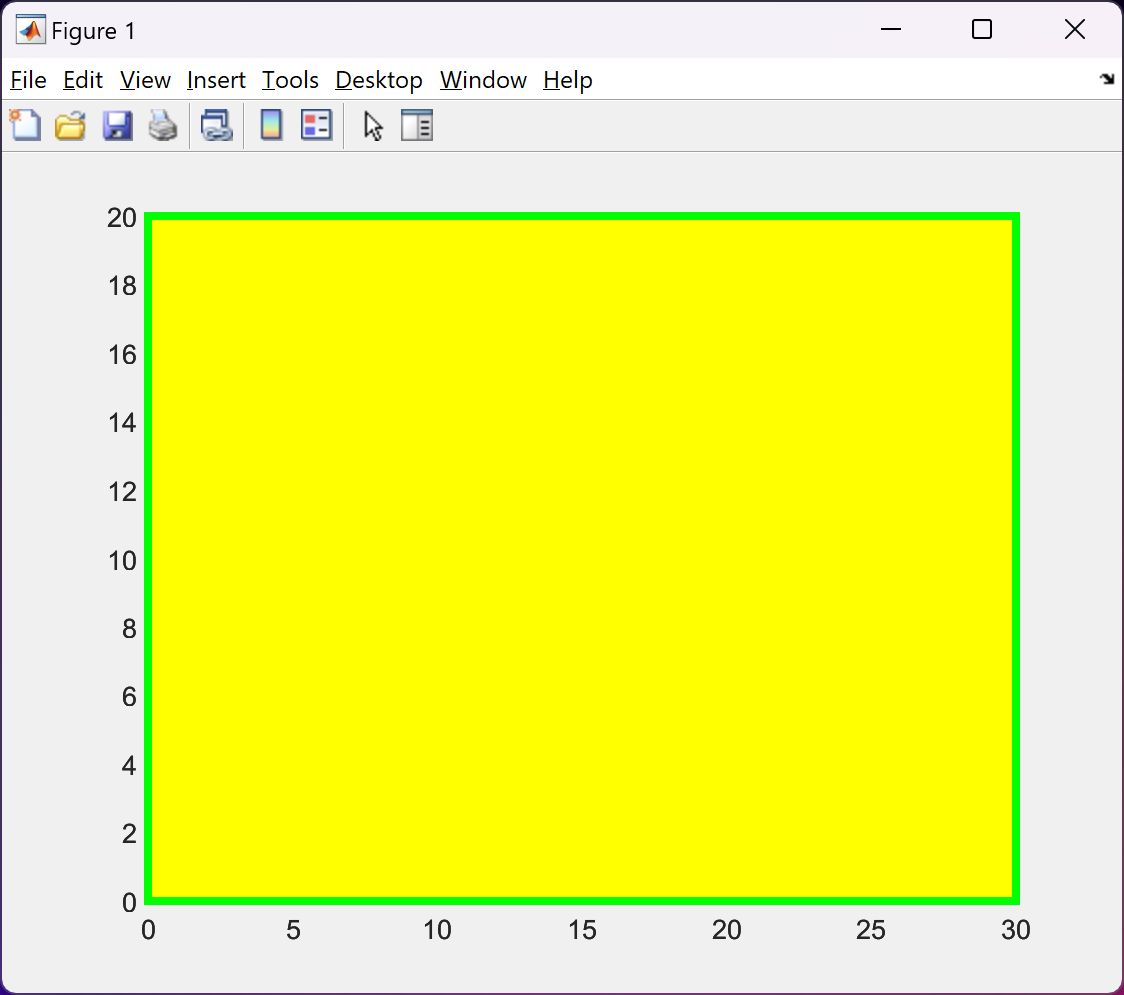
'LineWidth', lineWidth, ...

'Curvature', [0 0]);

end

*Aufruf*:

>>DrawRectangle(30, 20);



### Beispiel: DrawCircle

**Spezifikation**: Mit Hilfe einer Funktion (DrawCircle) soll ein gelber Kreis mit grünem Rand und Linienstärke 3 erzeugt werden. Der Durchmesser solle der Funktion übergeben werden.

function DrawCircle(diameter)

lowerLeftXY = [0 0];

deltaXY = [diameter diameter];

faceColor = 'y';

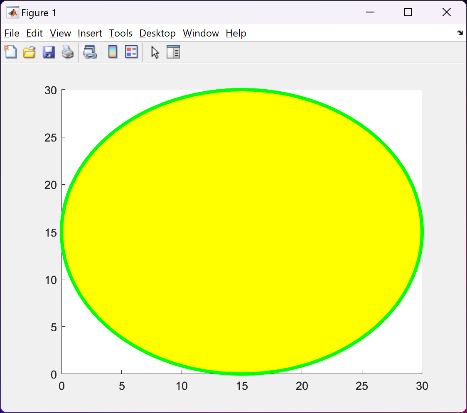
edgeColor = 'g';

lineWidth = 3;

rectangle( ...

'Position', [lowerLeftXY deltaXY], ...

'FaceColor', faceColor, ...

 'EdgeColor', edgeColor, ...

'LineWidth', lineWidth, ...

'Curvature', [1 1]);

End

*Aufruf*:

>>DrawCircle(30);

## Rotation von Grafikelementen

myLine = line(xValues, yValues, 'Color', lineColor, 'LineWidth', lineWidth);

zdir = [0 0 -1]; %axis clockwise

angleOffset = 20; %degree

centerXYZ = [50 50 0]; %!!!put in 3 dimensions

rotate(myLine, zdir, angleOffset, centerXYZ);

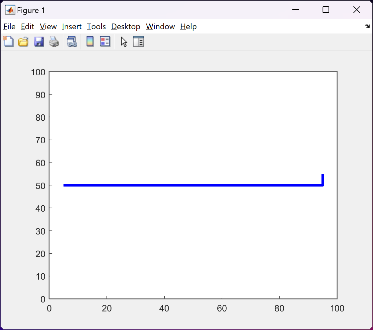
### Beispiel Funktion: RotateLine

**Spezifikation**: Mit Hilfe einer Funktion (RotateLine) soll eine horizontale blaue Linie mit Linienstärke 3 erzeugt werden und anschließend in einzelnen Schritten um 360 Grad gedreht werden. Der Offsetwinkel soll der Funktion übergeben werden können. Damit das Ende der Linie erkennbar ist, soll sie am Ende eine kleine Querlinie bekommen.

**Lösung**:

1. Damit sich die Achse bei der Drehung nicht ändert, wird vorab ein graues Rechteck (0,0 bis 100, 100) mit Linienstärke 1 erzeugt.

2. Linie erzeugen und Variable zuordnen: x von 5 bis 95; y auf 50; Farbe ‚b‘; Linienstärke 3

3. In einer Schleife (Anzahl 360/angleOffset) den Rotate-Befehl auf die Linie anwenden.

function RotateLine(angleOffset)

%create a canvas first

lowerLeftXY = [0 0];

deltaXY = [100 100];

edgeColor = [.4 .4 .4];

lineWidth = 1;

rectangle( ...

'Position', [lowerLeftXY deltaXY], ...

'EdgeColor', edgeColor, ...

'LineWidth', lineWidth);

%start with a line

lineColor = 'b';

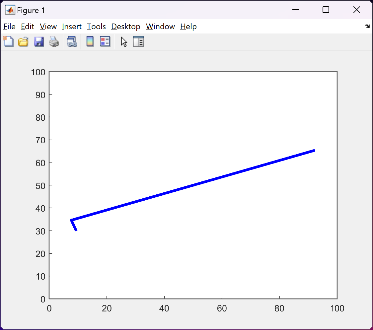
lineWidth = 3;

xValues = [5 95 95];

yValues = [50 50 55];

myLine = line(xValues, yValues, 'Color', lineColor, 'LineWidth', lineWidth);

pause(.2);



%define rotation

zdir = [0 0 -1]; %axis clockwise

%input: angleOffset in degree

centerXYZ = [50 50 0]; %!!!put in 3 dimensions

%now rotate the line

for i = 1: 360/angleOffset

rotate(myLine, zdir, angleOffset, centerXYZ);

pause(.2);

end

end

*Aufruf Funktion*:

>>rotateLine(20);

# Aufgabe: Das Haus vom Nikolaus zeichnen

**Spezifikation**: Mit Hilfe einer Funktion (DrawNicosHouse) soll das Haus des Nikolaus gezeichnet werden. Die Länge eines Segmentes und ob ein kleines (grün) oder großes (blau) Haus ist, soll übergeben werden können.

**Lösung**: Innerhalb einer Bedingung ein kleines (grün) oder großes (blau) Haus zeichnen.

function DrawNicosHouse(delta, isSmall)

lineWidth = 3;

sx = 0;

sy = 0;

d = delta;

if isSmall

lineColor = 'g';

xValues(1) = sx; yValues(1) = sy;

xValues(2) = sx; yValues(2) = sy+d;

xValues(3) = sx+d/2; yValues(3) = sy+2\*d;

xValues(4) = sx+d; yValues(4) = sy+d;

xValues(5) = sx; yValues(5) = sy+d;

xValues(6) = sx+d; yValues(6) = sy;

xValues(7) = sx; yValues(7) = sy;

xValues(8) = sx+d; yValues(8) = sy+d;

xValues(9) = sx+d; yValues(9) = sy;

else

lineColor = 'b';

xValues(1) = sx; yValues(1) = sy;

xValues(2) = sx; yValues(2) = sy+d;

xValues(3) = sx+d; yValues(3) = sy+2\*d;

xValues(4) = sx+2\*d; yValues(4) = sy+d;

xValues(5) = sx; yValues(5) = sy+d;

xValues(6) = sx+2\*d; yValues(6) = sy;

xValues(7) = sx; yValues(7) = sy;

xValues(8) = sx+2\*d; yValues(8) = sy+d;

xValues(9) = sx+2\*d; yValues(9) = sy;

end

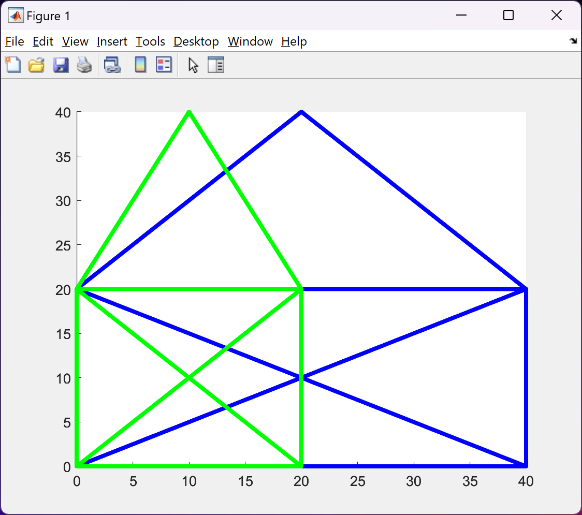
line(...

'XData', xValues, ...

'YData', yValues, ...

'Color', lineColor, ...

'LineWidth', lineWidth);

end

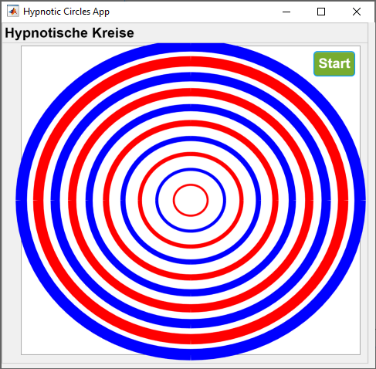
*Aufruf Funktion*:

>>DrawNicosHouse(30, true);

>>DrawNicosHouse(30, false);

**Zusatzaufgabe:** Zusätzliche Eingabe zur Verschiebung des Hauses in x-Richtung ermöglichen.

# Aufgabe: Hypnotische Kreise

**Spezifikation**: Mit Hilfe einer Funktion (DrawHynosisCircles) sollen 10 kleiner werdende Kreise (Start mit Radius 50) in abnehmender Strichstärke (Start mit 11) und den abwechselnden Farben rot und blau gezeichnet werden. Die Anzahl der Durchläufe soll der Funktion übergeben werden können.

Aufruf: >>DrawHypnosisCirlces(noOfCycles)

**Lösung**

1. Zunächst nur *einen* Kreis zeichnen!

function DrawHypnosisCircles(cycles)

%define attributes of circles

center = [50 50];

radius = 50;

lineWidth = 11;

color = 'r';

% draw circle

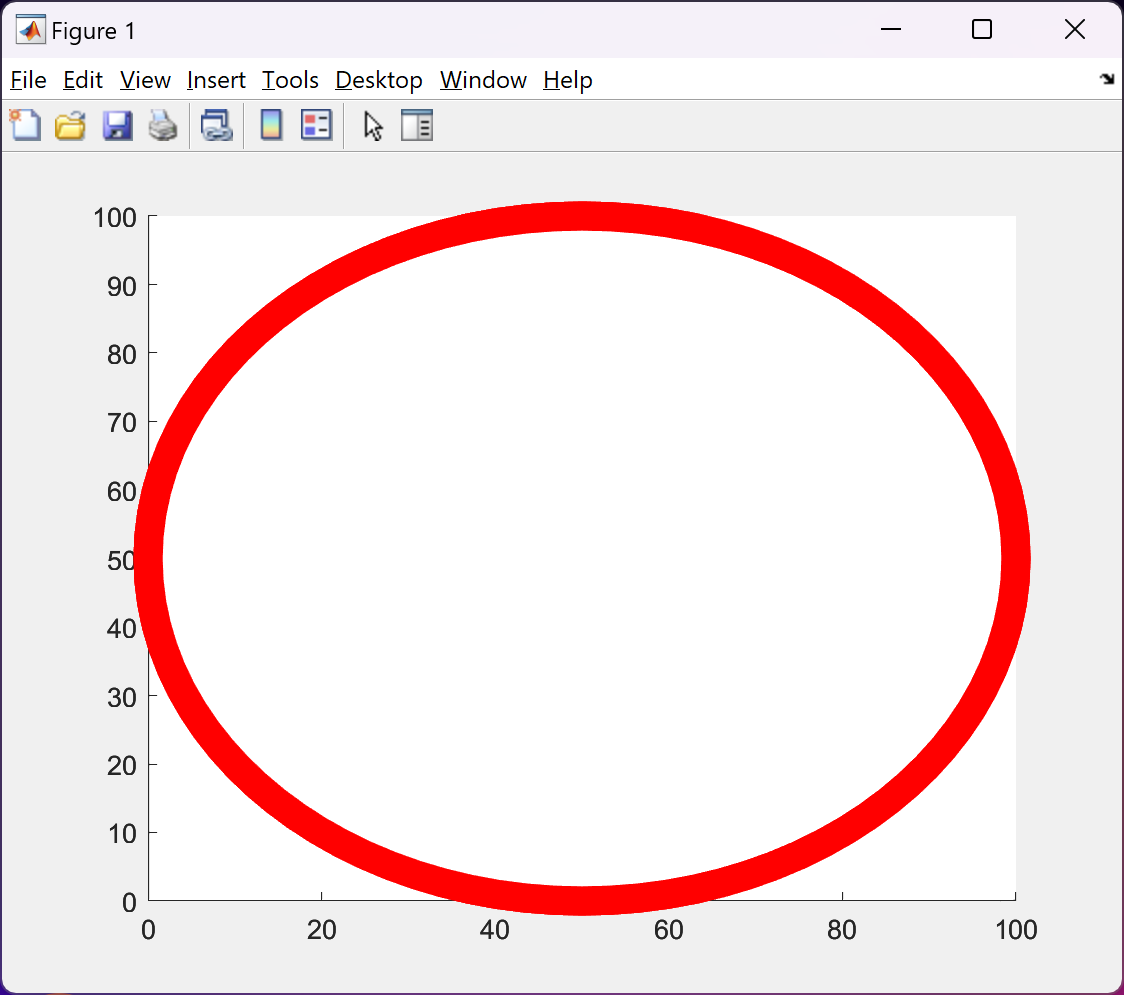
lowerLeftXY=[center(1)-radius center(2)-radius];

offSetXY = [radius\*2 radius\*2];

rectangle('Position',[lowerLeftXY offSetXY], ...

'EdgeColor', color, 'LineWidth', lineWidth, 'Curvature', [1 1]);

End



2. Mit einer „For-Schleife“ können im nächsten Schritt 10 Kreise hintereinander gezeichnet werden. Die Kreise sollen kleiner werden also muss sich der Radius nach jedem neuen Kreis verkleinern.

Damit das nicht so schnell geht, kann die Geschwindigkeit mit einer kleinen Pause reduziert werden.

function DrawHypnosisCircles1(cycles)

%define attributes of circles

center = [50 50];

radius = 50;

lineWidth = 11;

color = 'R';

for i=1:10

% draw circle

lowerLeftXY=[center(1)-radius center(2)-radius];

offSetXY = [radius\*2 radius\*2];

rectangle('Position',[lowerLeftXY offSetXY], ...

'EdgeColor', color, 'LineWidth', lineWidth, 'Curvature', [1 1]);

% reduce radius

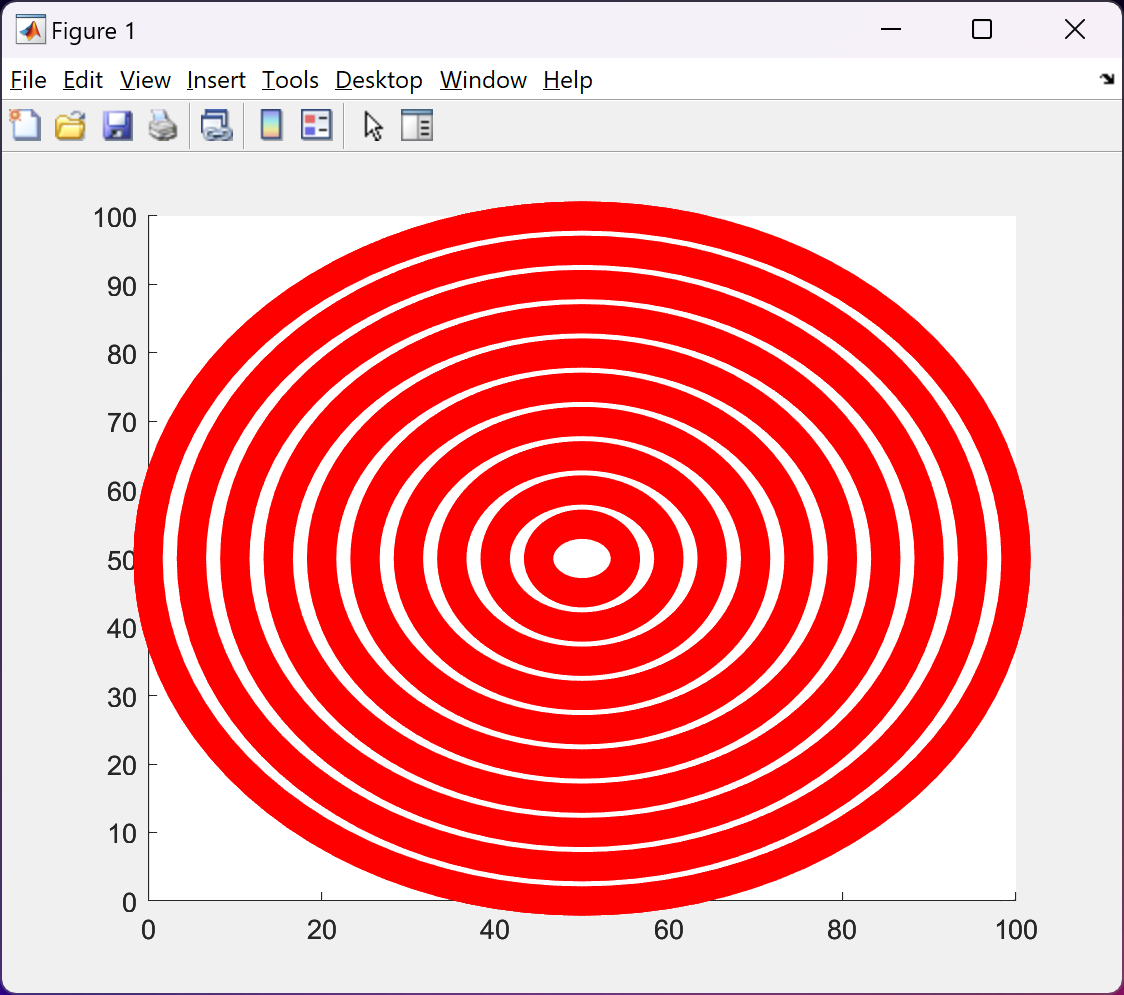
radius = radius - 5;;

% reduce speed

pause(.1);

end

end



3. Die Kreise sollen ihre Farbe bei jedem Durchlauf wechseln. Diese Bedingung wird jetzt hinzugefügt.

for …

%switch color

if color == 'r'

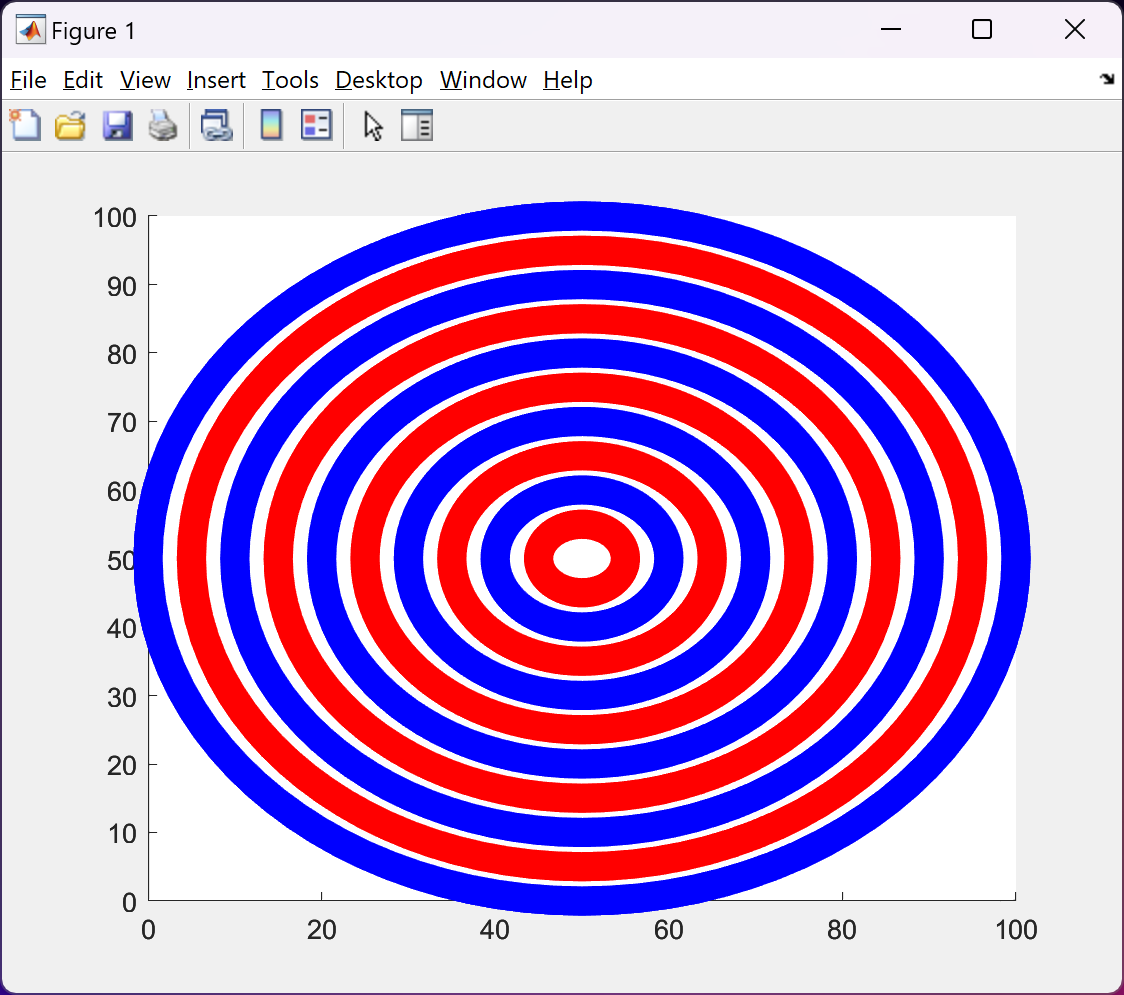
color = 'b';

else

color = 'r';

end

% draw circle ---

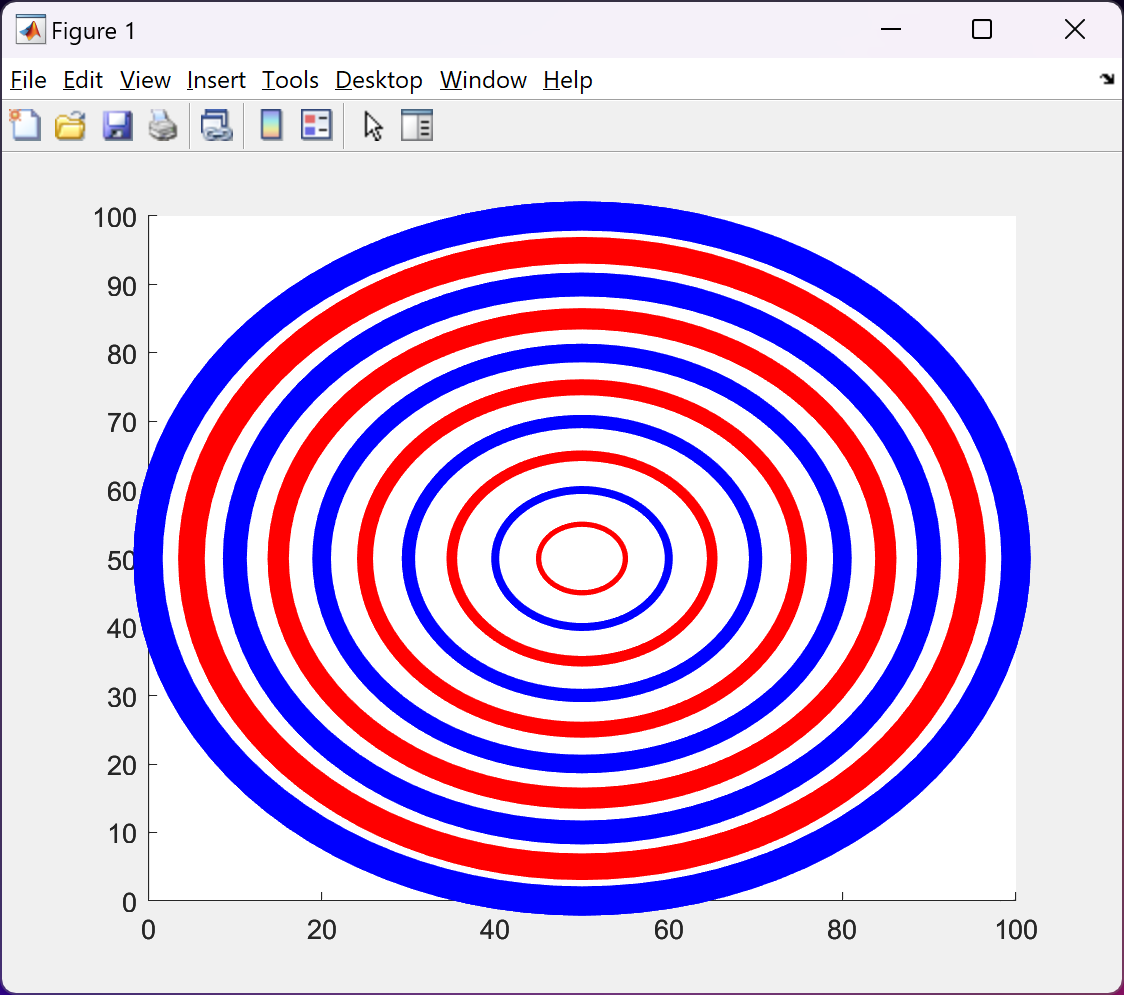


4. Schließlich wird noch die Linienstärke in jedem Durchlauf reduziert.

rectangle(…

% reduce line width

lineWidth = lineWidth-1;



5. Damit die Anzahl der Zyklen durchlaufen werden können, wird die „For-Schleife“ mit einer „While-Schleife“ umgeben. Hinweis: ursprünglicher Radius und Linienstärke innerhalb der Schleife setzen.

function DrawHypnosisCircles4(cycles)

%define attributes of circles

center = [50 50];

color = 'r';

while cycles > 0

radius = 50;

lineWidth = 11;

for i=1:10

%switch color

if color == 'r'

color = 'b';

else

color = 'r';

end

% draw circle

lowerLeftXY=[center(1)-radius center(2)-radius];

offSetXY = [radius\*2 radius\*2];

rectangle('Position',[lowerLeftXY offSetXY], ...

'EdgeColor', color, 'LineWidth', lineWidth, 'Curvature', [1 1]);

% reduce radius

radius = radius - 5;

% reduce line width

lineWidth = lineWidth-1;

% reduce speed

pause(.1);

end

color = 'b';

cycles = cycles - 1;

end

end

6. **Zusatzaufgabe**: Die Kreise sollen gelöscht werden bevor der nächste Gesamtdurchlauf erfolgt. Dazu müssen die erzeugten Kreise einer Variable zugewiesen werden. Im Anschluss der For-Schleife erfolgt dann eine weitere For-Schleife mit dem Löschbefehl „delete“ für jeden einzelnen Kreis.

function DrawHypnosisCircles5(cycles)

%define attributes of circles

center = [50 50];

color = 'r';

myRectangle = gObjects(10);

while cycles > 0

radius = 50;

lineWidth = 11;

for i=1:10

%switch color

if color == 'r'

color = 'b';

else

color = 'r';

end

% draw circle

lowerLeftXY=[center(1)-radius center(2)-radius];

offSetXY = [radius\*2 radius\*2];

myRectangle(i) = rectangle('Position',[lowerLeftXY offSetXY], ...

'EdgeColor', color, 'LineWidth', lineWidth, 'Curvature', [1 1]);

% reduce radius

radius = radius - 5;

% reduce line width

lineWidth = lineWidth-1;

% reduce speed

pause(.05);

end

for i = 10:-1:1

delete(myRectangle(i));

pause(.05);

end

cycles = cycles - 1;

end

end

*Aufruf Funktion*:

>>DrawHypnosisCircles(5);

# Klassen

Wir haben gelernt, mit Funktionen zu arbeiten. Eine Funktion erledigt eine Aufgabe mit Hilfe von Variablen, Bedingungen und Schleifen. Sie kann über einen Namen aufgerufen werden, ihr können Werte übergeben werden und sie hat gegebenenfalls einen Rückgabewert. Wenn aber mehrere Aufgaben zu demselben Thema abgearbeitet werden müssen, wäre es nützlich, sie irgendwie in einer gemeinsamen Hülle zusammenzufassen.

Die Funktionen, die zu einem Thema gehören, müssen vielleicht auch auf gemeinsame Variablen (Eigenschaften) zugreifen?

Es könnte auch nützlich sein, einige dieser Variablen von außerhalb der Hülle änderbar wären (öffentliche Eigenschaften).

Andere Variablen sind vielleicht nur innerhalb der Klasse von Bedeutung? (private Eigenschaften)

Diese Hülle wird in einer sogenannten „Klasse“ definiert.

In einer Klasse werden also Variablen, Funktionen und später auch Ereignisse zu einem ganz speziellen Thema zusammengefasst.

Klassen haben einen Bereich zur Definition der Eigenschaften (properties), einen Bereich zur Definition von Funktionen (methods) und einen Bereich zur Definition von Ereignissen (events).

**Konstruktor einer Klasse**

Der Konstruktor ist eine öffentliche Funktion innerhalb einer Klasse, die beim ersten Aufruf, der sogenannten Instanziierung, durchlaufen wird. Sie trägt den Namen der Klasse.

**Öffentliche, private und lokale Variablen einer Klasse:**

Öffentliche Variablen stehen in allen Methoden der Klasse zur Verfügung und können von außerhalb der Klasse gelesen oder verändert werden. Sie werden in der Property-Gruppe ohne oder mit dem Zusatz „Access=public“ zusammengefasst.

Private Variablen stehen in allen Methoden der Klasse zur Verfügung, können aber nicht von außen beeinflusst werden. Sie werden in der Property-Gruppe mit dem Zusatz „Access=private“ zusammengefasst.

Die öffentlichen und privaten Variablen einer Klasse sind in allen öffentlichen und privaten Funktionen der Klasse gültig. Um sie innerhalb der Klasse benutzen zu können, muss jedoch der Rückgabewert der Klasse, z.B. „app.Width“ vorangestellt werden.

**Öffentliche, private Funktionen einer Klasse**

Öffentliche Funktionen können von außerhalb der Klasse aufgerufen werden. Sie werden in der Method-Gruppe ohne oder mit dem Zusatz „Access=public“ zusammengefasst.

Private Funktionen können jedoch nicht von außerhalb der Klasse aufgerufen werden. Sie werden in der Method-Gruppe mit dem Zusatz „Access=private“ zusammengefasst.

Die öffentlichen und privaten Funktionen sind von allen anderen öffentlichen und privaten Funktionen der Klasse aufrufbar. Um sie innerhalb der Klasse benutzen zu können, muss jedoch der Rückgabewert der Klasse, z.B. „obj.Draw()“ vorangestellt werden.

**Einfaches Beispiel**

**Aufruf in Matlab-UI**: Lasche Home->Menü File->Dropdown New -> Schaltfeld Class

**Spezifikation**: In einer Klasse soll eine gelbe Box mit magenta Rand und den öffentlichen Eigenschaften Width (Breite) und Height (Höhe) erzeugt werden können. Die Box soll mit Aufruf der Funktion „Draw()“ erzeugt werden. Anschließend sollen mit weiteren Instanzen unterschiedlich große Boxen erzeugt werden. Die Funktion soll umschaltbar sein, sodass anstelle des Rechtecks auch Kreise oder ein Fenster in der Mitte erzeugt werden können.

**Lösung**:

classdef clsBox

properties (Access = private)

faceColor = 'y'

edgeColor = 'm'

lineWidth = 3

curvature = [0 0 ]

end

properties (Access = public)

Height = 20

Width = 20

IsWindow = false

IsCircle = false;

end

methods (Access = public)

% constructor

function obj = clsBox()

end

function obj = Draw(obj)

lowerLeftXY = [0 0];

deltaXY = [obj.Width obj.Height];

obj.drawBox(lowerLeftXY, deltaXY);

if obj.IsCircle

obj.curvature = [1 1];

end

if obj.IsWindow

obj.drawWindow();

end

end

end

methods (Access = private)

function obj = drawBox(obj, lowerLeftXY, deltaXY)

rectangle( ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor', obj.faceColor, ...

'EdgeColor', obj.edgeColor, ...

'LineWidth', obj.lineWidth, ...

'Curvature', obj.curvature);

end

function obj = drawWindow(obj)

width = obj.Width/4;

height = obj.Height/4;

lowerLeftXY = [obj.Width/2-width/2 obj.Height/2-height/2];

deltaXY = [width height];

obj.faceColor = 'c';

obj.edgeColor = 'b';

obj.drawBox(lowerLeftXY, deltaXY);

end

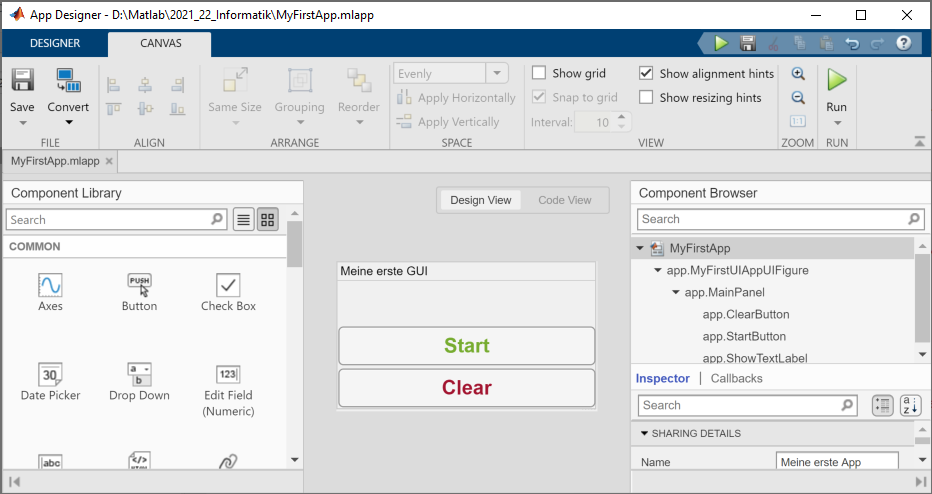
end

end

# GUI erzeugen

Erzeugung von grafischen Benutzeroberflächen (GUI, UI, Fenster, Window, Dialog)

Aufruf: appdesigner oder unter dem Tab: APPS ->aus Menüleiste: Design App



**Erklärung**: Steuerelemente, Properties, Ereignisse

## Beispiel Hello World

**GuiApp:** *Panel* erzeugen durch Hineinziehen des Steuerelementes „Panel“ aus der Component Library-> Bereich „Containers“ in die Zeichenfläche.

Im Component-Browser Werte ändern

* Text Feld leeren
* FontSize 20
* FontWeight B
* FontColor cyan

*Label* erzeugen durch Hineinziehen des Steuerelementes „Label“ in das Panel.

Im Component-Browser Werte ändern

* Text Feld leeren
* FontSize 20
* FontWeight B
* FontColor green

*Button* erzeugen durch Hineinziehen des Steuerelementes „Button“ das Panel.

Im Component-Browser Werte ändern

* Text Start
* FontSize 20
* FontWeight B
* FontColor red

*Button* erzeugen durch Hineinziehen des Steuerelementes „Button“ das Panel.

Ein Bild, das Tisch enthält.

Automatisch generierte BeschreibungIm Component-Browser Werte ändern

* Text Clear
* FontSize 20
* FontWeight B
* FontColor red

Grafikfläche bis auf die neuen Steuerelemente verkleinern.

Im Code-View des AppDesigners ein CallBack Ereignis für den Start- und ClearButton hinzufügen und ergänzen:

% Button pushed function: STARTButton

function STARTButtonPushed(app, event)

app.Label.Text = "Hello World!";

end

% Button pushed function: CLEARButton

function CLEARButtonPushed(app, event)

app.Label.Text = "";

end

**Aufgabe: Schleifen im Dialog**

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche die Schleifenaufgabe darstellen.

function STARTButtonPushed(app, event)

app.STARTButton.Enable = false;

loopStart = app.StartwertEditField.Value;

loopStep = app.StepwertEditField.Value;

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung loopEnd = app.EndwertEditField.Value;

for i = loopStart:loopStep:loopEnd

app.Label.Text = int2str(i);

pause(.2);

end

app.STARTButton.Enable = true;

end

function EnableButtons(app, isEnable)

if isEnable

app.FORButton.Enable = true;

app.WHILEButton.Enable = true;

else

app.FORButton.Enable = false;

app.WHILEButton.Enable = false;

end

end

function WHILEButtonPushed(app, event)

app.EnableButtons(false);

loopStart = app.StartwertEditField.Value;

loopStep = app.StepwertEditField.Value;

loopEnd = app.EndwertEditField.Value;

i = loopStart;

while i < loopEnd

app.Label.Text = int2str(i);

i = i + loopStep;

pause(.2);

end

app.EnableButtons(true);

end

**Aufgabe: Obstpreise im Dialog**

**Spezifikation**: Mit Hilfe einer Benutzeroberfläche soll der Preis von ausgewähltem Obst angezeigt werden können. Das zur Verfügung stehende Obst kann aus einem herunterklappbarem Steuerelement ausgewählt werden.

**Lösung**: 1. GUI mit DropDown und 2 Label erzeugen.

2. Funktion zur Erzeugung des Labelinhaltes.

3. Startfunktion für initialen Labelinhalt.

3. Änderungsfunktion der DrowDown aktivieren:

function fillPrice(app, fruit, price)

Ein Bild, das Text enthält.

Automatisch generierte Beschreibung costs = " kosten heute pro Kilo €";

app.FruitLabel.Text = strcat(fruit, costs);

app.PriceLabel.Text = price;

end

% Code that executes after component creation

function startupFcn(app)

app.fillPrice("Bananen", "2,90");

end

% Value changed function: DropDown

function DropDownValueChanged(app, event)

fruit = app.DropDown.Value;

price = "0,0";

switch fruit

case 'Äpfel'

price = "1,80";

case 'Birnen'

price = "2,40";

case 'Orangen'

price = "3,20";

case 'Bananen'

price = "2,90";

case {'Mangos', 'Papayas'}

price = "4,20";

case 'Ananas'

price = "3,90";

end

app.fillPrice(fruit, price);

end

# Aufgabe: Kreis zwischen Schaltflächen

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche und zwei Schaltflächen soll ein Kreis in einer Klasse nach rechts oder links verschoben werden können. (BallMove)

**Lösung**: Klasse mit öffentlichen Funktionen „MoveRight“ und „MoveLeft“ anlegen.

classdef clsBallMove

Ein Bild, das Text, Screenshot, Betriebssystem, Software enthält.

Automatisch generierte Beschreibung properties (Access = private)

sizeXY

ball

ballWidth=10

edgeColor='m'

faceColor='y'

lineWidth=3

curvature=[1 1]

UIAxes

end

properties (Access = public)

IsSquare

end

methods (Access = public)

% constructor

function app = clsBallMove(uIAxes, sizeXY)

app.UIAxes = uIAxes;

app.sizeXY = sizeXY;

app = app.draw();

end

function app = MoveLeft(app)

app = app.move(true);

end

function app = MoveRight(app)

app = app.move(false);

end

end

methods (Access = private)

function app = move(app, isLeft)

loopStart=app.ball.Position(1);

if isLeft

loopEnd = 0;

loopStep=-2;

else

loopEnd = app.sizeXY-app.ballWidth;

loopStep=2;

end

for i=loopStart:loopStep:loopEnd

app.ball.Position(1)=i;

pause(.05);

end

end

function app = draw(app)

lowerLeftXY = [app.sizeXY(1)/2 app.sizeXY(2)/2];

deltaXY = [app.ballWidth, app.ballWidth];

app.ball = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',app.lineWidth, ...

'Curvature',app.curvature);

end

end

end

**UI-Funktionen**

properties (Access = private)

Ball % Description

end

% Callbacks that handle component events

methods (Access = private)

% Code that executes after component creation

function startupFcn(app)

app.UIAxes.YAxis.Visible = 'off'; % remove y-axis

app.UIAxes.XAxis.Visible = 'off'; % remove x-axis

canvalOrigin = [0 0];

canvasSize = [100 100];

lowerLeftXY = canvalOrigin;

deltaXY = canvasSize;

rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'EdgeColor',[.4 .4 .4], ...

'LineWidth',1);

app.Ball = clsBallMove(app.UIAxes, canvasSize);

end

% Button pushed function: LButton

function LButtonPushed(app, event)

app.Ball = app.Ball.MoveLeft();

end

% Button pushed function: RButton

function RButtonPushed(app, event)

app.Ball = app.Ball.MoveRight();

end

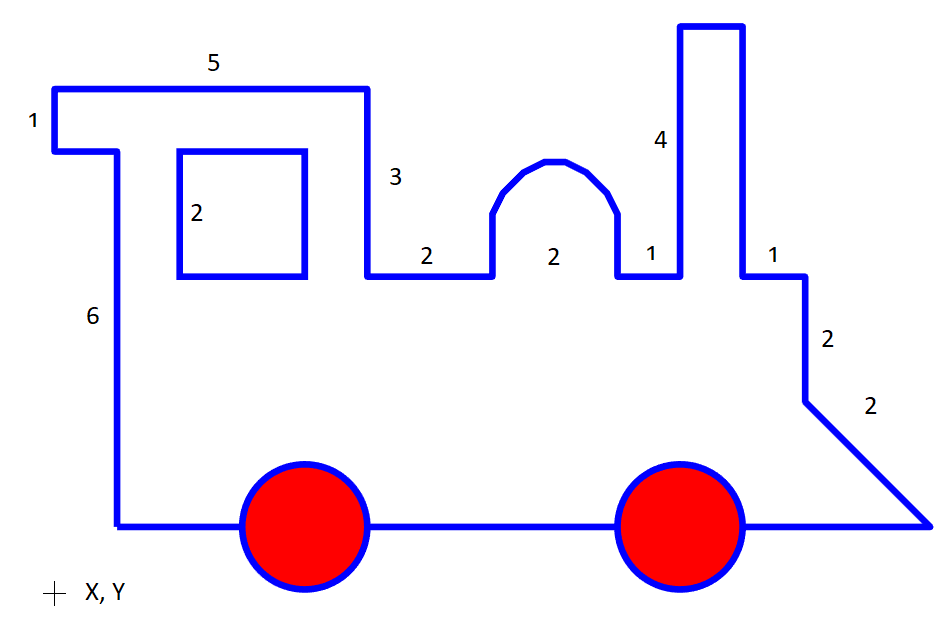
end

**Zusatzaufgabe**: Erweitern um Buttons und Funktionen für Reset/MoveUp/MoveDown und Umschaltmöglichkeit in ein Rechteck!

# Aufgabe: Lokomotive vorwärts rückwärts bewegen

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche und einer Start-Schaltfläche soll eine kleine Lokomotive gezeichnet und beweget werden. Sie soll sich bis zum Ende der Zeichenfläche bewegen und dann wieder zurück.

**Lösung**: Zunächst eine Klasse „clsLoco“ für die Lokomotive anlegen.



Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung

*In der GUI*

properties (Access = private)

Loc % Description

End

% Code that executes after component creation

function startupFcn(app)

app.UIAxes.YAxis.Visible = 'off'; % remove y-axis

app.UIAxes.XAxis.Visible = 'off'; % remove x-axis

size = 100;

lowerLeftXY = [0 0];

deltaXY = [size size/2];

rectangle( app.UIAxes, ...

'Position',[lowerLeftXY deltaXY], ...

'EdgeColor',[.4 .4 .4], ...

'LineWidth',1);

app.Loc = clsLoco(app.UIAxes, [0 0]);

end

% Button pushed function: STARTButton

function STARTButtonPushed(app, event)

app.Loc.Loops = app.LoopsSpinner.Value;

app.Loc = app.Loc.Move();

end

*Die Klasse*

classdef clsLoco

properties (Access = private)

startXY

UIAxes

body

wheel1

wheel2

edgeColor = 'b'

faceColor = 'c'

lineWidth = 2

end

properties (Access = public)

Loops = 3

end

methods (Access = public)

% constructor

function app = clsLoco(uIAxes, startXY)

app.UIAxes = uIAxes;

app.startXY = startXY;

app = app.draw();

end

function app = Move(app)

app = app.moveLoco();

end

end

methods (Access = private)

function app = moveLoco(app)

deltaXY = [2 2];

counter = app.Loops;

while counter > 0

loopStart = 1;

loopStep = 1;

loopEnd = 100-14;

for i=loopStart:loopStep:loopEnd

app.startXY(1) = app.startXY(1) + loopStep;

app.body.XData = app.getXValues();

lowerLeftXY = [app.startXY(1)+3 app.startXY(2)];

app.wheel1.Position = [lowerLeftXY, deltaXY];

lowerLeftXY = [app.startXY(1)+9 app.startXY(2)];

app.wheel2.Position = [lowerLeftXY, deltaXY];

pause(.1);

end

loopStart = 100-14;

loopStep = -1;

loopEnd = 1;

for i=loopStart:loopStep:loopEnd

app.startXY(1) = app.startXY(1) + loopStep;

app.body.XData = app.getXValues();

lowerLeftXY = [app.startXY(1)+3 app.startXY(2)];

app.wheel1.Position = [lowerLeftXY, deltaXY];

lowerLeftXY = [app.startXY(1)+9 app.startXY(2)];

app.wheel2.Position = [lowerLeftXY, deltaXY];

pause(.1);

end

counter = counter - 1;

end

end

function app = draw(app)

sY = app.startXY(2)+1;

xValues = app.getXValues();

yValues = [sY sY+6 sY+6 sY+7 sY+7 sY+4 sY+4 sY+5 sY+5 sY+4 sY+4 sY+8 sY+8 sY+4 sY+4 sY+2 sY sY];

app.body = line(app.UIAxes, ...

'XData', xValues, 'YData', yValues, ...

'Color',app.edgeColor, ...

'LineWidth',app.lineWidth);

lowerLeftXY = [app.startXY(1)+3 app.startXY(2)];

deltaXY = [2 2];

app.wheel1 = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',app.lineWidth, ...

'Curvature', [1 1]);

lowerLeftXY = [app.startXY(1)+9 app.startXY(2)];

app.wheel2 = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',app.lineWidth, ...

'Curvature', [1 1]);

end

function xValues = getXValues(app)

sX = app.startXY(1)+1;

xValues = [sX sX sX-1 sX-1 sX+4 sX+4 sX+6 sX+6 sX+8 sX+8 sX+9 sX+9 sX+10 sX+10 sX+11 sX+11 sX+13 sX];

end

end

end

# Aufgabe: Windrad

Es soll ein Windrad gezeichnet und seine Rotorblätter zum Drehen gebracht werden.

GUI-Name: WindWheel;

Zeichenfläche: 100x100,

WindWheelBase:

Drehachse im Zentrum der Zeichenfläche (Durchmesser 2)

Basisbreite des Mastes 8

RotorBlades:

Breite max. 5

Breite min. 2

Länge 80

Eingaben:

Anzahl der Umdrehungen

Schaltfläche zum Starten der Konfiguration

Klasse: clsWindWheel

Sie soll einen Canvas und einen Start-Button enthalten.

Funktionen: drawWindWheelBase und drawRotorBlades

## Lösungsansatz

* Zunächst mit GUIDE eine neue GUI mit dem Startknopf, einem Eingabefeld für die Anzahl der Umdrehungen und dem Canvas anlegen und unter dem gegebenen Namen speichern.
* In der Opening-Funktion die Beschriftungen ausschalten und eine globale Variable für den Drehpunkt (wheelCenter) anlegen. Dann einen grauen Rahmen zeichnen, der das Feld auf 100 Einheiten in X und Y Richtung begrenzt und schließlich die Funktion drawWindWheelBase und drawRotorBlades in der Ausgangsposition, beide mit Übergabe des Drehpunktes, aufrufen.
* Im StartButton\_Callback die Funktion drawRotorBlades aufrufen mit dem globalen Drehpunkt und einer Variablen für die Anzahl der Umdrehungen. Das Beispiel soll das Konzept von rotierenden Elementen vermitteln.
* Starten sie in der Funktion zunächst einfach: was ist erforderlich, um den Mühlenaufbau zu zeichnen? Dann fügen Sie die dazu notwendigen Variablen ein, lassen die Aufbau in der Funktion erzeugen und starten die GUI und drücken den Start-Button.
* Im nächsten Schritt überlegen Sie was erforderlich ist, um die Rotorblätter in einer Grundposition zu zeichnen? Fügen Sie auch hier die notwendigen Variablen ein und lassen das gesamte Windrad in der GUI aufbauen.
* Erst im letzten Schritt werden die Rotorblätter bewegt.

## Lösung Matlab Aufgabe: Windrad

Willkommen bei der Lösung der Matlab-Aufgabe „Windrad“. Wir starten mit GUIDE und legen eine neue GUI mit dem Startknopf und dem Canvas an und speichern sie unter dem Namen „WindWheel“.

In der „Opening-Function“ die Beschriftungen ausschalten, die graue rechteckige Begrenzung zeichnen.

% remove axis description

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

minX = 0;

minY = 0;

maxX = 100;

maxY = 100;

lowerLeftXY = [minX minY];

offSetXY = [maxX maxY];

rectangle('Position',[lowerLeftXY offSetXY], 'EdgeColor', [.4 .4 .4], 'LineWidth', 1);

Das globale Zentrum auf die Mitte des Feldes setzen und die Funktion „drawWindMillBase“ mit diesem Zentrum aufrufen.

global Center

Center = [maxX/2 maxY/2];

drawWindMillBase(Center);

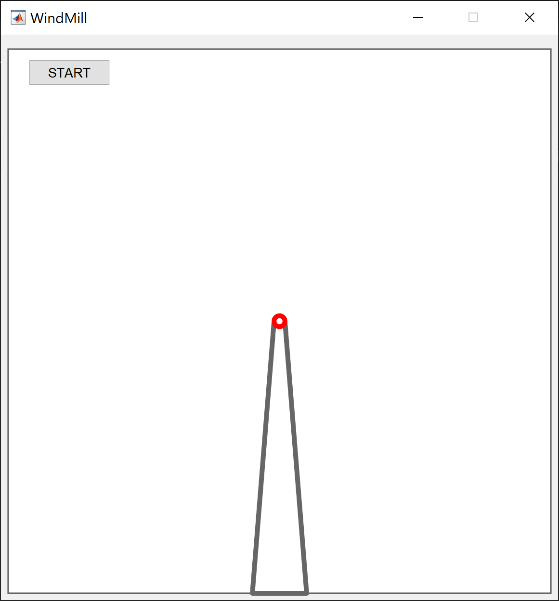
Anschließend die Funktion „drawWindMillBase“ anlegen und die grafischen Elemente Kreis und Linie anlegen.

function drawWindWheelBase(center)

% draw base

line(xValues, yValues, 'LineWidth',lineWidth, 'Color', baseColor);

% draw rotation point

 rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', rotationPointColor, 'LineWidth', lineWidth, 'Curvature', [1 1]);

end

Dann die dazu notwendigen Variablen anlegen. Ausprobieren!

function drawWindWheelBase(centerXY)

baseWidthTop = 2;

baseWidthButtom = 8;

lineColor = [.4 .4 .4];

lineWidth = 3;

%%axis

lowerLeftXY = [centerXY(1)-baseWidthTop/2 centerXY(2)-baseWidthTop/2];

deltaXY = [baseWidthTop baseWidthTop];

rectangle( ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor','c', ...

'EdgeColor','r', ...

'LineWidth',lineWidth, ...

'Curvature', [1 1]);

%%base

xValues = [centerXY(1)-baseWidthTop/2 centerXY(1)-baseWidthButtom/2 centerXY(1)+baseWidthButtom/2 centerXY(1)+baseWidthTop/2];

yValues = [centerXY(2) 0 0 centerXY(2)];

line(xValues, yValues, 'Color',lineColor, ...

'LineWidth',lineWidth);

end

Die Funktion „drawRotorBlades“ aus der Funktion WindWheelBase aufrufen.

nter

drawRotorBlades(Center);

Funktion „drawRotorBlades“ anlegen und das grafische Element einer Linie eintragen.

Notwendige Variablen anlegen und füllen und in Standposition ausprobieren.

function rotorBlades = drawRotorBlades(centerXY)

bladeMinWidth = 2;

bladeMaxWidth = 5;

bladeLength = 80;

lineColor = 'b';

lineWidth = 3;

xValues = [centerXY(1)-bladeMinWidth/2 centerXY(1)-bladeMaxWidth/2 centerXY(1) ...

centerXY(1)+bladeMaxWidth/2 centerXY(1)+bladeMinWidth/2 ...

centerXY(1)+bladeMaxWidth/2 centerXY(1) ...

centerXY(1)-bladeMaxWidth/2 centerXY(1)-bladeMinWidth/2];

yValues = [centerXY(2) centerXY(2)-bladeLength/4 centerXY(2)-bladeLength/2 ...

centerXY(2)-bladeLength/4 centerXY(2) ...

centerXY(2)+bladeLength/4 centerXY(2)+bladeLength/2 ...

centerXY(2)+bladeLength/4 centerXY(2)];

rotorBlades = line(xValues, yValues, 'Color',lineColor, ...

'LineWidth',lineWidth);

End

Die Funktion rotateRotorBlades erzeugen:

function rotorBlades = rotateRotorBlades(rotorBlades, centerXY, cycles)

if isgraphics(rotorBlades) && cycles > 0

angleOffset = 6;

zdir = [0 0 1];

for i = 1:cycles

angle = angleOffset;

while angle <= 360

center = [centerXY(1) centerXY(2) 0];

rotate(rotorBlades, zdir, angleOffset, center);

pause(.05);

angle = angle + angleOffset;

end

end

end

end

Im StartButton\_Callback die Funktion „drawRotorBlades“ mit z.B. 2 Umdrehungen aufrufen.

global Center

global RotorBlades

global Center

rotations = str2double(handles.Rotations.String);

RotorBlades = rotateRotorBlades(RotorBlades, Center, rotations);

In die Funktion „drawRotorBlades“ eine Bedingung einbauen, dass die statischen Rotorblätter nur bei 0 Umdrehungen gezeichnet werden sollen.

function drawRotorBlades(center, noOfRotations)

global rotorBlades

if noOfRotations > 0

%rotate blades

else

radius = 1;

lineWidth = 3;

bladeColor = 'b';

bladeWidth = 2.5;

bladeLength = 40;

%draw lower blade

xValues = [center(1)+radius center(1)+bladeWidth center(1) center(1)-bladeWidth center(1)-radius];

yValues = [center(2) center(2)-bladeLength/2 center(2)-bladeLength center(2)-bladeLength/2 center(2)];

rotorBlades(1)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

%draw upper blade

yValues = [centerXY(2) centerXY(2)+bladeLength/2 centerXY(2)+bladeLength center(2)+bladeLength/2 center(2)];

rotorBlades(2)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

end

end

Die Funktion „drawRotorBlades“ um eine For-Schleife für die Anzahl der Umdrehungen in der Bedingung erweitern.

…

if noOfRotations > 0

%rotate blades

for i=1:noOfRotations

end

else

…

Im nächsten Schritt wird innerhalb der For-Schleife eine While-Schleife für eine einzelne Umdrehung (360 Grad) erzeugt und der Befehl „rotate“ eingefügt.

…

%rotate blades

for i=1:noOfRotations

while angle <= 360

rotate(rotorBlades, zdir, angleOffset, center);

pause(.05);

end

end

…

Notwendige Variablen anlegen und füllen und das fertige Programm ausprobieren.

Hinweis: die Z-Direction kennzeichnet die Drehachse für die Rotation.

function drawRotorBlades(center, noOfRotations)

global rotorBlades

if noOfRotations > 0

%rotate blades

angleOffset = 6;

zdir = [0 0 1];

for i=1:noOfRotations

angle = angleOffset;

while angle <= 360

center = [center(1) center(2) 0];

rotate(rotorBlades, zdir, angleOffset, center);

pause(.05);

angle = angle + angleOffset;

end

end

else

radius = 1;

lineWidth = 3;

bladeColor = 'b';

bladeWidth = 2.5;

bladeLength = 40;

%draw lower blade

xValues = [center(1)+radius center(1)+bladeWidth center(1) center(1)-bladeWidth center(1)-radius];

yValues = [center(2) center(2)-bladeLength/2 center(2)-bladeLength center(2)-bladeLength/2 center(2)];

rotorBlades(1)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

%draw upper blade

yValues = [center(2) center(2)+bladeLength/2 center(2)+bladeLength center(2)+bladeLength/2 center(2)];

rotorBlades(2)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

end

end

# Aufgabe: Generischer Baum

**Spezifiaktion**: Es soll eine Baumstruktur erzeugt werden, bei der die jeweiligen Abzweige immer halb so breit sind wie zu Beginn. Begrenzt werden soll der Baum nur durch die Angabe einer Gesamtbreite. Die Größen von Zweighöhe, Radius des Abzweiges und die Linienstärken sollen sich abhängig von der jeweiligen Breite verkleinern.

Ein Bild, das Screenshot, Text, Display, Diagramm enthält.

Automatisch generierte Beschreibung

GUI-Name: GenericTree;

mit Canvas und StartButton

Funktionen: drawTree, drawBranch, drawFork

Minimale Breite: 1

Zeichenfläche: 200x200

Linienstärke und Durchmesser Astgabel: Zweigbreite/5

Länge Baumstamm: Zeichenfläche/4

## Lösungsansatz

* Zunächst mit GUIDE eine neue GUI mit dem Startknopf und dem Canvas anlegen und unter dem gegebenen Namen speichern.
* In der Opening-Funktion die Beschriftungen ausschalten und im StartButton Callback die Funktion „drawTree“ aufrufen.
* Denken Sie bitte wie immer daran, Programme bestehen im Wesentlichen aus Variablen, Bedingungen und Schleifen. In diesem Beispiel soll das Konzept von Rekursionen eingeführt werden, das bedeutet, das sich die Funktion selbst aufrufen kann.
* Starten sie in der Funktion zunächst einfach: was ist erforderlich, um einen Stamm zu zeichnen?
* Fügen Sie die dazu notwendigen Variablen ein, lassen den Stamm zeichnen.
* Dahinter wird einfach die Funktion „drawBranch“ aufgerufen.
* Die Funktion „drawBranch“ startet mit einem Kreis als Abzweig am Ende des Stammes. Von dem gehen nach links und rechts Zweige in Form von Linien ab, die nur halb so breit sind wie zu Beginn.
* Denken Sie sich eine Endbedingung aus, bei dem der Baum aufhören soll zu wachsen, z.B. eine minimale Zweiglänge. Solange diese Länge nicht erreicht ist, können sie die Position des Abzweiges auf das Ende des vorherigen Zweiges verschieben und die Funktion erneut für die rechte und linke Seite durchlaufen lassen. Probieren Sie es aus!
* Zuletzt benötigen wir noch eine Reduktion von Linienstärke, Radius und vertikaler Zweiglänge um die Aufgabe vollständig zu lösen.

## Lösung Matlab Aufgabe: Generischer Baum

Wir starten mit GUIDE und legen eine neue GUI mit dem Startknopf und dem Canvas an und speichern sie unter dem Namen „GenericTree“. In der Opening-Funktion die Beschriftungen ausschalten und im StartButton Callback die Funktion „drawTree“ aufrufen.

function GenericTree\_OpeningFcn(hObject, eventdata, handles, varargin)

% remove axis description

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

app.Tree = clsTree(UIAxes, center);

function startButton\_Callback(hObject, eventdata, handles)

app.Tree = app.Tree.Draw();

Wir starten in der Funktion zunächst einfach: ich füge den Befehl ein, der einen Stamm zeichnet, dann als Abzweig einen Kreis darstellt und schließlich je einen Zweig zu jeder Seite zeichnet. Damit kann ich sehen, welche Werte benötigt dieser Befehl? Dann füge ich die notwendigen Variablen ein. Nach Betätigung des Start-Button ergibt der Test nun den erwarteten Abzweig! So sieht die Funktion nur mit den grafischen Elementen aus:

classdef clsTree

properties (Access = private)

maxXY

UIAxes

lineColor = 'b'

faceColor = 'r'

edgeColor = 'r'

divider = 5;

trunkLineWidth = 8;

minWidth = 1;

branchHeight

end

properties (Access = public)

TreeWidth = 100

end

methods (Access = public)

% constructor

function app = clsTree(uIAxes, maxXY)

app.UIAxes = uIAxes;

app.maxXY = maxXY;

end

function app = Draw(app)

center = [app.maxXY(1)/2 0];

treeWidth = app.TreeWidth/2;

app = app.drawTrunk(center, treeWidth/2);

app.branchHeight = treeWidth/2;

center = [app.maxXY(1)/2 treeWidth/2];

app = app.drawBranch(center, treeWidth);

end

end

methods (Access = private)

function app = drawTrunk(app, center, treeWidth)

diameter = treeWidth/app.divider;

xValues = [center(1) center(1)];

yValues = [0 treeWidth-diameter/2];

line(app.UIAxes, ...

'XData', xValues, ...

'YData', yValues, ...

'Color', app.lineColor, ...

'LineWidth', app.trunkLineWidth);

end

function app = drawBranch(app, center, treeWidth)

treeWidth = treeWidth/2;

diameter = treeWidth/app.divider;

lineWidth = treeWidth/app.divider;

app = app.drawFork(center, diameter, lineWidth);

%left branch

xValues = [center(1)-diameter/2 center(1)-treeWidth center(1)-treeWidth];

yValues = [center(2) center(2) center(2)+app.branchHeight];

line(app.UIAxes, ...

'XData', xValues, ...

'YData', yValues, ...

'Color', app.lineColor, ...

'LineWidth', lineWidth);

%right branch

xValues = [center(1)+diameter/2 center(1)+treeWidth center(1)+treeWidth];

line(app.UIAxes, ...

'XData', xValues, ...

'YData', yValues, ...

'Color', app.lineColor, ...

'LineWidth', lineWidth);

if app.minWidth < treeWidth

%left

centerL = [center(1)-treeWidth center(2)+app.branchHeight];

app = app.drawBranch(centerL, treeWidth);

pause(.1);

%right

centerR = [center(1)+treeWidth center(2)+app.branchHeight];

app = app.drawBranch(centerR, treeWidth);

pause(.1);

end

end

function app = drawFork(app, center, diameter, lineWidth)

lowerLeftXY = [center(1)-diameter/2 center(2)-diameter/2];

deltaXY = [diameter diameter];

rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',lineWidth, ...

'Curvature', [1 1]);

end

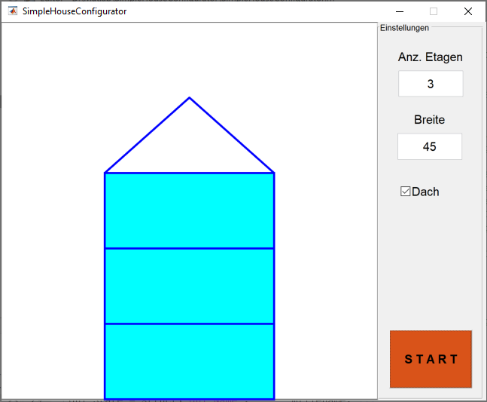
end

end

# Aufgabe: Konfigurator für einfaches Haus

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „einfacher Hauskonfigurator“. Es soll Konfigurator für ein einfaches Haus erzeugt werden, bei dem sich die Anzahl der Etagen, die Breite und ggf. die Erstellung eines Daches einstellen lässt.

Die Anfangskonfiguration: 1 Etage, Breite 20 und ohne Dach sein.

GUI-Name: SimpleHouseConfigurator; Canvas 100x100, Anzahl Etagen, Breite und StartButton

Klasse: clsSimpleHouse

## Lösungsansatz

* Zunächst mit GUIDE eine neue GUI mit dem Startknopf, einem Eingabefeld für die Anzahl Etagen und die Breite sowie dem Canvas anlegen und unter dem gegebenen Namen speichern.
* In der Opening-Funktion die Beschriftungen ausschalten und einen grauen Rahmen zeichnen, der das Feld auf 100 Einheiten in X und Y Richtung begrenzt. Danach eine globale Variable für das Haus (SimpleHouse) anlegen, die Werte für Anzahl Stockwerke und Breite auslesen und schließlich die Klasse Haus (clsSimpleHouse) mit diesen Werten instanziieren.
* Im StartButton-Callback ebenfalls die globale Variable für das Haus (SimpleHouse) anlegen, die Werte für Anzahl Stockwerke, Breite und ob ein Dach erforderlich ist auslesen und dem Hausobjekt zuweisen. Abschließend den „Draw“ Funktion des Objektes Haus (SimpleHouse) aufrufen.

## Lösung Matlab Aufgabe: Konfigurator für einfaches Haus

Willkommen bei der Lösung der Matlab-Aufgabe „einfacher Hauskonfigurator“. Wir starten mit GUIDE und legen eine neue GUI mit dem Canvas, Editfeld für Etagen und Breite sowie den Startknopf an und speichern sie unter dem Namen „SimpleHouseConfigurator“.

In der Opening-Funktion die Beschriftungen ausschalten und die Klasse „SimpleHouse“ instanziieren.

### Opening Function

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

lowerLeftXY = [0 0];

deltaXY = [100 100];

rectangle( ...

'Position',[lowerLeftXY deltaXY], ...

'EdgeColor',[.4 .4 .4], ...

'LineWidth',1);

global house

houseWidth = str2double(handles.Width.String);

noOfFloors = str2double(handles.FloorNo.String);

hasRoof = handles.RoofCheckbox.Value;

house = clsSimpleHouse(houseWidth, noOfFloors, hasRoof);

### StartButton-Callback

global house

house.Width = str2double(handles.Width.String);

house.NoFloors = str2double(handles.FloorNo.String);

house.HasRoof = handles.RoofCheckbox.Value;

house = house.Draw();

### Klasse: clsSimpleHouse

classdef clsSimpleHouse

properties (Access = private)

canvasWidth = 100;

house

roof

lineWidth = 2

lineColor = 'b'

floorHeight = 20

minWidth = 20.0

minNoFloors = 1

maxWidth

maxHeight

latestNoFloors

end

properties (Access = public)

Width

NoFloors

HasRoof = false

FaceColor = 'c'

end

methods (Access = public)

%constructor

function obj = clsSimpleHouse(width, noOfFloors, hasRoof)

obj.maxWidth = obj.canvasWidth;

obj.maxHeight = obj.canvasWidth;

obj.Width = width;

obj.NoFloors = noOfFloors;

obj.HasRoof = hasRoof;

obj.Draw();

end

function obj = Draw(obj)

obj = obj.deleteRoof();

obj = obj.deleteWalls();

obj = obj.drawWalls();

if obj.HasRoof

obj = obj.drawRoof();

end

end

end

methods (Access = private)

function obj = drawWalls(obj)

obj.latestNoFloors = obj.NoFloors;

for i=1:obj.NoFloors

lowerLeftXY = [obj.canvasWidth/2 - obj.Width/2 (i-1) \* obj.floorHeight];

deltaXY = [obj.Width obj.floorHeight];

obj.house(i) = rectangle( ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor',obj.FaceColor, ...

'EdgeColor',obj.lineColor, ...

'LineWidth',obj.lineWidth);

end

end

function obj = drawRoof(obj)

xValues = [obj.canvasWidth/2 - obj.Width/2 obj.canvasWidth/2 obj.canvasWidth/2 + obj.Width/2];

yValues = [obj.floorHeight \* obj.NoFloors obj.floorHeight \* (obj.NoFloors + 1) obj.floorHeight \* obj.NoFloors];

obj.roof = line(xValues, yValues, 'Color', obj.lineColor, 'LineWidth', obj.lineWidth);

end

function obj = deleteWalls(obj)

if ~isempty(obj.house)

for i=1:obj.latestNoFloors

delete(obj.house(i));

end

end

end

function obj = deleteRoof(obj)

if ~isempty(obj.roof)

delete(obj.roof);

end

end

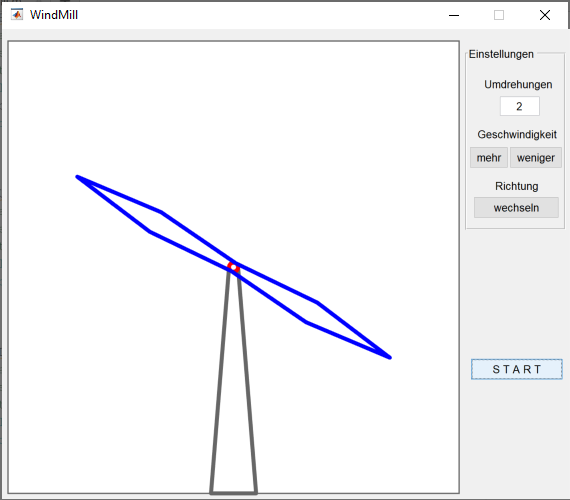
end

end

# Aufgabe: Eine Klasse für „Windrad“ anlegen

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Erweiterung für die Matlab Aufgabe „Windrad“. Dieses Mal soll eine Klasse für das Windrad aus der vorherigen Aufgabe erzeugt werden. Sie soll in der Lage sein, ihre Rotorblätter zum Drehen zu bringen. Außerdem soll die Anzahl der Umdrehungen, die Geschwindigkeit und die Drehrichtung im Dialog änderbar sein.

GUI-Name: WindMill

Klasse: clsWindMill(center)

Funktionen in der Klasse:

drawWindMillBase

drawRotorBlades

increaseSpeed

decreaseSpeed

changeDirection

## Lösungsansatz

* Es empfiehlt sich, das Ergebnis von dem ursprünglichen Programm „WindMill“ in einen neuen Ordner zu kopieren und von hier aus zu arbeiten.
* Zunächst die bestehende GUI „WindMill“ mit Hilfe des Befehls GUIDE um ein Seitenmenü erweitern. In einer Group-Box sollen ein Eingabefeld für die Anzahl der Umdrehungen, Schaltflächen zur Erhöhung oder Verringerung der Geschwindigkeit, sowie eine Schaltfläche für den Richtungswechsel angelegt werden. Der bestehende Startbutton wird einfach in das Seitenmenü verschoben.
* Mit der New Schaltfläche eine neue Klasse anlegen. Eine Klasse ist ein Container für viele Eigenschaften, *die Properties* und Methoden, *unsere Funktionen*. Eine der Funktionen kann ein Konstruktor sein, das ist eine Funktion, die aufgerufen wird, wenn das Objekte zum ersten Mal instanziiert wird.
* Es gibt private und öffentliche Eigenschaften und Methoden. Die privaten funktionieren nur innerhalb der Klasse, die öffentlichen können von außen geändert werden. Es sollten nur die notwendigsten Eigenschaften oder Funktionen öffentlich gemacht werden, denn wir müssen sie ständig auf Plausibilität überprüfen.
* Wir benötigen nur eine öffentliche Eigenschaft, die „NoOfRotations“. Als öffentliche Funktion benötigen wir „drawRotorBlades“, „increaseSpeed“, „decreaseSpeed“ und „changeDirection“. Dagegen sollte die Funktion „drawWindMillBase“ als private Funktion ausgeführt werden.
* Die Klasse wird als global gekennzeichnet und nur einmal initial angelegt, in unserem Fall schon in der Opening-Function. Wir können dann aus verschiedenen Callback-Funktionen immer wieder auf ihre Eigenschaften und Funktionen zugreifen.

classdef untitled11

%UNTITLED11 Summary of this class goes here

% Detailed explanation goes here

Properties (Access = private)

Property0

end

properties

Property1

end

methods

function obj = untitled11(inputArg1,inputArg2)

%UNTITLED11 Construct an instance of this class

% Detailed explanation goes here

obj.Property1 = inputArg1 + inputArg2;

end

function obj = method1(obj,inputArg)

%METHOD1 Summary of this method goes here

% Detailed explanation goes here

outputArg = obj.Property1 + inputArg;

end

end

methods (Access = private)

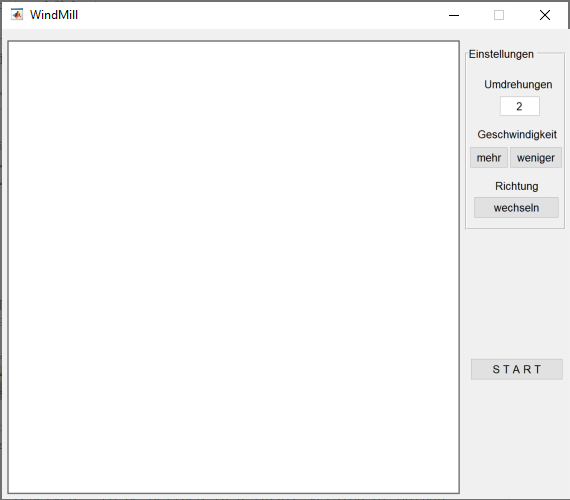
function obj = method2(obj,inputArg)

%METHOD1 Summary of this method goes here

% Detailed explanation goes here

obj = obj.Property1 + inputArg;

end

 end

end

## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der erweiterten Matlab-Aufgabe „Windrad“. Wir starten mit GUIDE und erweitern die GUI „WindMill“ mit einem Seitenmenü. Hier fügen wir zunächst eine Groupbox mit dem Tag „settingsGroup“ und dem Title „Einstellungen“ ein.

In diese Box werden eingefügt:

* Ein Static-Text mit Namen „rotationText“ und String „Umdrehungen“ sowie ein Edit-Feld mit Namen „rotations“ und dem String „2“ für die Umdrehungen
* Ein Static-Text mit Namen „velocityText“ und String „Geschwindigkeit“ sowie zwei Schaltflächen mit den Namen „speedUpButton“ String „mehr“ und „speedDownButton“ String „weniger“ für Beschleunigung oder Verringerung der Geschwindigkeit.
* Ein Static-Text mit Namen „directionText“ und String „Richtung“ sowie eine Schaltfläche mit den Namen „changeDirectionButton“ String „wechseln“ für den Richtungswechsel der Rotation.
* Unseren Start-Knopf können wir unter die Groupbox verschieben.

Die beiden bestehenden Funktionen in der Opening-Function werden ersetzt durch den globalen Namen und Aufruf der Klasse und Übergabe des Centers.

% drawWindMillBase(millCenter);

% drawRotorBlades(millCenter, 0);

global WindMill

WindMill = clsWindMill(millCenter)

Anschließend bauen wir die leere Hülle unserer neuen Klasse „clsWindMill“ auf:

classdef clsWindMill

%private properties

properties (Access = private)

end

%public properties

properties

end

%public methods

methods

%constructor

function obj=clsWindMill(centerXY)

end

%functions

% draw rotor blades

function obj = drawRotorBlades(obj)

end

% decrease rotation speed

function obj = decreaseSpeed(obj)

end

% increase rotation speed

function obj = increaseSpeed(obj)

end

% change rotation direction

function obj = changeDirection(obj)

end

end

%private methods

methods (Access = private)

% draw windmill base

function obj = drawWindMillBase(obj)

end

end

end

Wir beginnen mit der geforderten öffentlichen Variablen „NoOfRotations“ und geben ihr den Vorgabewert 0.

Die privaten Variablen sind alle weiteren auf unseren Funktionen und die neuen Anforderungen wie speed und direction. Der Block für die Properties sieht anschließend wie folgt aus:

%private properties

properties (Access = private)

lineWidth = 3;

radius = 1;

rotationSpeed = .05;

acw = "anticlockwise";

cw = "clockwise";

direction

rotorBlades

center

end

%public properties

properties

NoOfRotations = 0;

end

Füllen wir jetzt den Konstruktor aus, weisen das Center unser internen Variable zu und rufen die beiden gelöschten Funktionen aus der Opening-Function auf.

%constructor

function obj=clsWindMill(centerXY)

obj.direction = obj.acw;

obj.center=centerXY;

obj = obj.drawWindMillBase();

obj = obj.drawRotorBlades();

end

Hinweis: Das obj steht für die Klasse selbst!

Wenn das Programm jetzt gestartet würde, hätte sich noch nichts geändert, denn die eigentlichen Funktionen sind ja noch leer. Deshalb übertragen wir sie jetzt von unseren schon vorhanden Funktionen „drawWindMillBase“ und „drawRotorBlades“.

Beginnen wir im privaten Methodenbereich bei der Funktion „drawWindMillBase“:

%private methods

methods (Access = private)

% draw windmill base

function obj = drawWindMillBase(obj)

lineWidth = 3;

radius = 1;

% draw base

halfWidth = 4;

baseColor = [.4, .4, .4];

xValues = [center(1)+radius center(1)+halfWidth+radius center(1)-halfWidth-radius center(1)-radius];

yValues = [center(2) center(2)-center(2) center(2)-center(2) center(2)];

line(xValues, yValues, 'LineWidth',lineWidth, 'Color', baseColor);

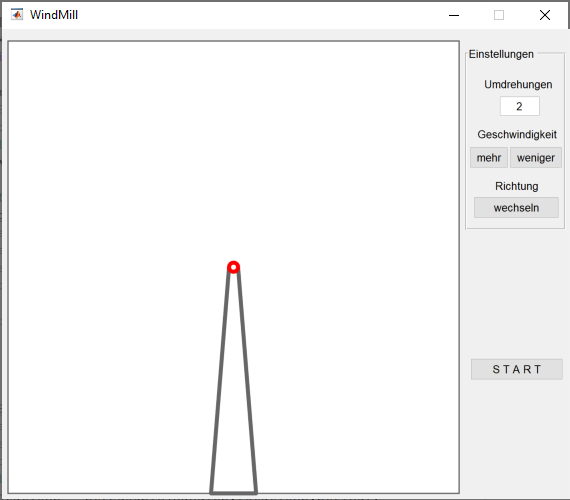
% draw rotation point

rotationPointColor = 'r';

lowerLeftXY = [center(1)-radius center(2)-radius];

deltaXY = [radius\*2 radius\*2];

rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', rotationPointColor, 'LineWidth', lineWidth, 'Curvature', [1 1]);

 end

end

Die Variablen lineWidth, radius und center sind jetzt die privaten Variablen der Klasse und müssen deshalb das Objekt der Klasse vorangestellt bekommen. Die bestehenden Zuweisungen können gelöscht werden und vor die genannten Variablen kann einfach das obj. kopiert werden.

Wenn das Programm gestartet wird, sollte die Basis gezeichnet sein.

% draw windmill base

function obj = drawWindMillBase(obj)

% draw base

halfWidth = 4;

baseColor = [.4, .4, .4];

xValues = [obj.center(1)+obj.radius obj.center(1)+halfWidth+obj.radius obj.center(1)-halfWidth-obj.radius obj.center(1)-obj.radius];

yValues = [obj.center(2) obj.center(2)-obj.center(2) obj.center(2)-obj.center(2) obj.center(2)];

line(xValues, yValues, 'LineWidth', obj.lineWidth, 'Color', baseColor);

% draw rotation point

rotationPointColor = 'r';

lowerLeftXY = [obj.center(1)-obj.radius obj.center(2)-obj.radius];

deltaXY = [obj.radius\*2 obj.radius\*2];

rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', rotationPointColor, 'LineWidth', obj.lineWidth, 'Curvature', [1 1]);

end

Wir können jetzt im öffentlichen Methodenbereich die Funktion „drawRotorBlades“ hinein kopieren:

% draw rotor blades

function obj = drawRotorBlades(obj)

global rotorBlades

if noOfRotations > 0

%rotate blades

angleOffset = 6;

zdir = [0 0 1];

for i=1:noOfRotations

angle = angleOffset;

while angle <= 360

center = [center(1) center(2) 0];

rotate(rotorBlades, zdir, angleOffset, center);

pause(.05);

angle = angle + angleOffset;

end

end

else

radius = 1;

lineWidth = 3;

bladeColor = 'b';

bladeWidth = 2.5;

bladeLength = 40;

%draw lower blade

xValues = [center(1)+radius center(1)+bladeWidth center(1) center(1)-bladeWidth center(1)-radius];

yValues = [center(2) center(2)-bladeLength/2 center(2)-bladeLength center(2)-bladeLength/2 center(2)];

rotorBlades(1)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

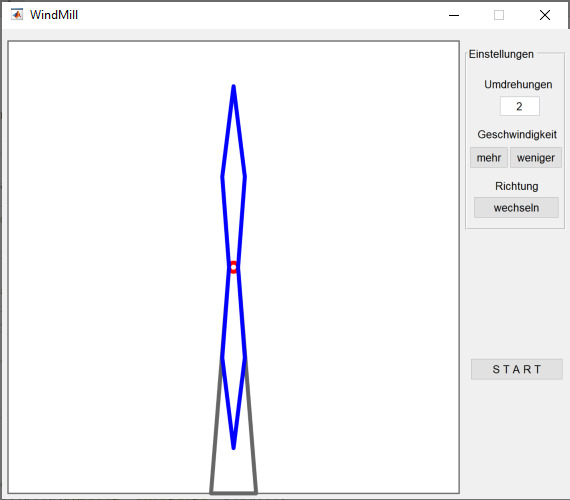
%draw upper blade

yValues = [center(2) center(2)+bladeLength/2 center(2)+bladeLength center(2)+bladeLength/2 center(2)];

rotorBlades(2)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', lineWidth);

end

end

Als erstes können wir die globale Variable entfernen, weil sie jetzt eine private Variable der Klasse geworden ist. Auch die Zuweisungen der Variablen lineWidth und radius können entfernt werden. Vor all diese Variablen und auch noOfRotations muss jetzt wieder das obj. kopiert werden. Zusätzlich muss der Anfangsbuchstabe von noOfRotations groß geschrieben werden, weil es sich um eine öffentliche Variablen handelt.

Wenn das Programm gestartet wird, sollten die Rotorblätter in der Ausgangsposition sichtbar sein.

%functions

% draw rotor blades

function obj = drawRotorBlades(obj)

if obj.NoOfRotations > 0

%rotate blades

angleOffset = 6;

zdir = [0 0 1];

for i=1:obj.NoOfRotations

angle = angleOffset;

while angle <= 360

obj.center = [obj.center(1) obj.center(2) 0];

rotate(obj.rotorBlades, zdir, angleOffset, obj.center);

pause(.05);

angle = angle + angleOffset;

end

end

else

bladeColor = 'b';

bladeWidth = 2.5;

bladeLength = 40;

%draw lower blade

xValues = [obj.center(1)+obj.radius obj.center(1)+bladeWidth obj.center(1) obj.center(1)-bladeWidth obj.center(1)-obj.radius];

yValues = [obj.center(2) obj.center(2)-bladeLength/2 obj.center(2)-bladeLength obj.center(2)-bladeLength/2 obj.center(2)];

obj.rotorBlades(1)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', obj.lineWidth);

%draw upper blade

yValues = [obj.center(2) obj.center(2)+bladeLength/2 obj.center(2)+bladeLength obj.center(2)+bladeLength/2 obj.center(2)];

obj.rotorBlades(2)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', obj.lineWidth);

end

end

Wenn wir den Code im StartButton Callback ändern, sollten sich die Rotorblätter auch wieder drehen. Wir benötigen die globale Variable millCenter nicht mehr, weil sie zum Bestandteil der Klasse geworden ist. Die Anzahl der Umdrehungen ist jetzt eine öffentliche Eigenschaft der Klasse, die zugewiesen werden muss. Da wir das Input-Feld „rotations“ mit der Anzahl der Umdrehungen angelegt haben, müssen wir diesen Wert in eine Zahl umwandeln und der Klassen-Eigenschaft zuweisen. Schließlich rufen wir die Funktion der WindMill-Klasse „drawRotorBlades“ einfach auf.

Vorher

global millCenter

noOfRotations = 2;

drawRotorBlades(millCenter, noOfRotations);

Nachher

global WindMill

WindMill.NoOfRotations = str2double(handles.rotations.String);

WindMill = WindMill.drawRotorBlades();

Das Rotorblatt sollte sich jetzt wieder drehen können, wenn der Startknopf gedrückt wurde.

Dann werden noch die Call-Back-Funktionen für Zu- und Abnahme von Geschwindigkeit und Änderung der Richtung einfach durch Aufruf der zugehörigen öffentlichen Methoden der Klasse implementiert.

function speedUpButton\_Callback(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.increaseSpeed();

function speedDownButton\_Callback(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.decreaseSpeed();

function changeDirectionButton\_Callback(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.changeDirection();

Die Implementierung der Funktion „increaseSpeed“ empfiehlt sich eine neue private Variable zu Festlegung eines Geschwindigkeits-Offsets „speedOffset“, z.B. von .01.

Die Geschwindigkeit wird größer, je kleiner der Wert für die Pause ist, deshalb wird das „speedOffset“ von der „rotationSpeed“ abgezogen. Es sollte jedoch einen minimalen Wert nicht unterschreiten.

% increase rotation speed

function obj = increaseSpeed(obj)

if obj.rotationSpeed > obj.speedOffset

obj.rotationSpeed = obj.rotationSpeed - obj.speedOffset;

end

end

Sehr ähnlich sieht dann die Funktion „decreaeSpeed“ aus. Nur sollte die Bedingung verhindern, dass ein maximaler Wert überschritten wird.

% decrease rotation speed

function obj = decreaseSpeed(obj)

if obj.rotationSpeed < .5

obj.rotationSpeed = obj.rotationSpeed + obj.speedOffset;

end

end

Dazu muss noch in der Funktion „drawRotorBlades“, in der While-Schleife die Pause in die Variable obj.rotationSpeed umgewandelt werden.

while angle <= 360

obj.center = [obj.center(1) obj.center(2) 0];

rotate(obj.rotorBlades, zdir, angleOffset, obj.center);

pause(obj.rotationSpeed);

angle = angle + angleOffset;

end

Zuletzt wird dann mit Hilfe einer Bedingung die Drehrichtung festgelegt.

% change rotation direction

function obj = changeDirection(obj)

if obj.direction == obj.acw

obj.direction = obj.cw;

else

obj.direction = obj.acw;

end

end

Dazu muss noch in der Funktion „drawRotorBlades“ die Z-Achse mit Hilfe einer Bedingung für die Drehung umgekehrt werden.

if obj.direction == obj.acw

zdir = [0 0 1];

else

zdir = [0 0 -1];

end

Alle Funktionen ausprobieren, fertig. Vielen Dank für’s zuschauen.

% remove axis description

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

minX = 0;

minY = 0;

maxX = 100;

maxY = 100;

lowerLeftXY = [minX minY];

offSetXY = [maxX maxY];

rectangle('Position',[lowerLeftXY offSetXY], 'EdgeColor', [.4 .4 .4], 'LineWidth', 1);

millCenter = [maxX/2 maxY/2];

global WindMill

WindMill = clsWindMill(millCenter);

function **startButton\_Callback**(hObject, eventdata, handles)

global WindMill

WindMill.NoOfRotations = str2double(handles.rotations.String);

WindMill = WindMill.Rotate();

function **speedUpButton\_Callback**(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.IncreaseSpeed();

function **speedDownButton\_Callback**(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.DecreaseSpeed();

function **changeDirectionButton\_Callback**(hObject, eventdata, handles)

global WindMill

WindMill = WindMill.ChangeDirection();

classdef **clsWindMill**

%private properties

properties (Access = private)

lineWidth = 3;

radius = 1;

rotationSpeed = .05;

speedOffset = .01;

acw = "anticlockwise";

cw = "clockwise";

direction

rotorBlades

center

end

%public properties

properties

NoOfRotations = 0;

end

%public methods

methods

%constructor

function obj=clsWindMill(centerXY)

obj.direction = obj.acw;

obj.center=centerXY;

obj = obj.drawWindMillBase();

obj = obj.drawRotorBlades();

end

% rotate rotor blades

function obj = Rotate(obj)

if obj.NoOfRotations > 0

%rotate blades

angleOffset = 6;

if obj.direction == obj.acw

zdir = [0 0 1];

else

zdir = [0 0 -1];

end

for i=1:obj.NoOfRotations

angle = angleOffset;

while angle <= 360

centerXYZ = [obj.center(1) obj.center(2) 0];

rotate(obj.rotorBlades, zdir, angleOffset, centerXYZ);

pause(obj.rotationSpeed);

angle = angle + angleOffset;

end

end

end

end

% decrease rotation speed

function obj = DecreaseSpeed(obj)

if obj.rotationSpeed < .5

obj.rotationSpeed = obj.rotationSpeed + obj.speedOffset;

end

end

% increase rotation speed

function obj = IncreaseSpeed(obj)

if obj.rotationSpeed > obj.speedOffset

obj.rotationSpeed = obj.rotationSpeed - obj.speedOffset;

end

end

% change rotation direction

function obj = changeDirection(obj)

if obj.direction == obj.acw

obj.direction = obj.cw;

else

obj.direction = obj.acw;

end

end

end

methods (Access = private)

% draw windmill base

function obj = drawWindMillBase(obj)

% draw base

baseColor = [.4, .4, .4];

xValues = [obj.center(1)+obj.radius obj.center(1)+4+obj.radius obj.center(1)-4-obj.radius obj.center(1)-obj.radius];

yValues = [obj.center(2) obj.center(2)-50 obj.center(2)-50 obj.center(2)];

line(xValues, yValues, 'LineWidth',obj.lineWidth, 'Color', baseColor);

% draw rotation point

rotationPointColor = 'r';

lowerLeftXY = [obj.center(1)-obj.radius obj.center(2)-obj.radius];

deltaXY = [obj.radius\*2 obj.radius\*2];

rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', rotationPointColor, 'LineWidth', obj.lineWidth, 'Curvature', [1 1]);

end

% draw rotor blades

function obj = drawRotorBlades(obj)

bladeColor = 'b';

bladeWidth = 2.5;

bladeLength = 40;

%draw lower blade

xValues = [obj.center(1)+obj.radius obj.center(1)+bladeWidth obj.center(1) obj.center(1)-bladeWidth obj.center(1)-obj.radius];

yValues = [obj.center(2) obj.center(2)-bladeLength/2 obj.center(2)-bladeLength obj.center(2)-bladeLength/2 obj.center(2)];

obj.rotorBlades(1)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', obj.lineWidth);

%draw upper blade

yValues = [obj.center(2) obj.center(2)+bladeLength/2 obj.center(2)+bladeLength obj.center(2)+bladeLength/2 obj.center(2)];

obj.rotorBlades(2)=line(xValues, yValues, 'Color', bladeColor, 'LineWidth', obj.lineWidth);

end

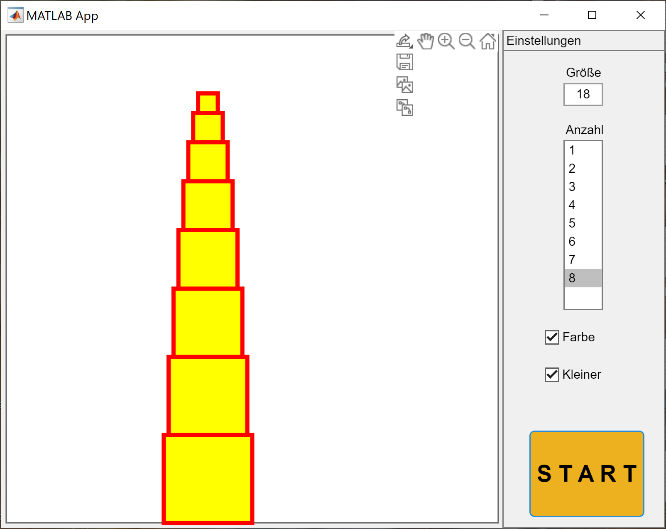
end

end

# Aufgabe: Kisten-Stapel Konfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „BoxStackConfigurator“. Es soll ein konfigurierbarer Stapel aus Quadraten erzeugt werden, bei der die Anzahl der Ebenen, ein Farbschema und die Möglichkeit der Verkleinerung der Quadrate in jeder Ebene eingestellt werden kann.

GUI-Name: BoxStackConfigurator

Klasse: clsBoxStack

Öffentliche Funktion: draw

## Lösungsansatz

* Zunächst die GUI „BoxStackConfigurator“ mit Hilfe des Befehls GUIDE anlegen und mit einem Seitenmenü versehen. In einer Group-Box sollen ein Editfeld für die Box-Breite, eine Liste für die Anzahl der Ebenen 1 -10, eine Checkbox für die Verkleinerung und eine für die Farbvariante. Unterhalb der Gruppe der übliche Start-Button.
* Die Klasse clsBoxStack anlegen, einen leeren Konstruktor und eine öffentliche Funktion „draw“ sowie die privaten Funktion „drawBox“ und „deleteBox“ anlegen, ebenfalls zunächst leer.
* Als öffentliche Eigenschaften benötigen wir die Anzahl der Ebenen, die Box-Breite, ob verkleinert werden soll und ob es farbig sein soll. Alle anderen Variablen können private angelegt werden.
* In der Opening-Funktion werden die Achsen ausgeschaltet und ein Rechteck mit xMax 100 und yMax 100 erzeugt. Danach wird der Boxstapel als global gekennzeichnet, mit einem Startpunkt 40, 0 instanziiert und die Funktion „draw“ gestartet.
* Im Start-Button Callback werden die Variablen aus dem Seitenmenü ausgelesen und der Klasse mitgegeben, anschließend wird die Funktion „draw“ erneut gestartet.
* In der Klasse selbst anschließend ein einfaches Rechteck in der „draw“ Funktion zeichnen und dann mit Hilfe von Bedingungen und Schleifen erweitern und in die private Funktion „drawBox“ überführen.
* Zuletzt die Möglichkeit der Verkleinerung und die Farben hinzufügen.

## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der Matlab-Aufgabe „BoxStackConfigurator“.

Wir starten mit GUIDE und erzeugen die zugehörige stGUI mit einem Canvas und einem Seitenmenü. Hier fügen wir zunächst eine Groupbox mit dem Tag „settingsGroup“ und dem Title „Einstellungen“ ein und bringen darin die geforderten Steuerelemente unter. Wie bekannt setzen wir darunter den Start-Button.

Ich gehe hier nicht mehr weiter auf den Grafik-Designer ein, wir werfen nur noch schnell einen Blick auf die Opening-Function und den Start-Button Callback.

Danach kümmern wir uns um den Aufbau der Klasse „clsBoxStack“.

Zunächst bauen wir die fast leere Hülle dieser Klasse auf:

* Mit den nicht öffentlichen Variablen für den Startpunkt, das Boxobjekt, die Breite, die Farbe der Fläche und der Umrandung, sowie die Linienstärke des Quadrates.
* Die öffentlichen Variablen sind die in der GUI änderbaren Werte der Ebenen, ob Verkleinerung und ob ein Farbschema gewünscht ist.
* Es folgt der Konstruktor mit Übergabe des Startpunktes und die öffentliche Funktion „draw“ mit dem Rechteck und den zugehörigen Variablen.

Nach dem Starten des Programms sollte jetzt das erste Quadrat zu sehen sein.

classdef clsBoxStack

properties (Access = private)

startXY

faceColor = 'w';

edgeColor = 'b';

lineWidth = 2;

end

properties

BoxWidth = 10;

Levels = 1;

IsReduce = false;

IsColor = false;

end

methods

function obj = clsBoxStack(startXY)

obj.startXY = startXY;

end

function obj = Draw(obj)

lowerLeftXY = [obj.startXY(1) obj.startXY(2)];

deltaXY = [obj.BoxWidth obj.BoxWidth];

rectangle('Position', [lowerLeftXY deltaXY], 'FaceColor', obj.faceColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

end

end

Als nächstes sollten wir die „draw“ Funktion um eine Schleife um das Rechteck erweitern, die so viele Quadrate zeichnet wie Ebenen angegeben wurden. Damit die Quadrate nicht alle übereinander gezeichnet werden, muss sich der Y-Wert der rechten unteren Ecke nach dem ersten Quadrat in der Schleife jeweils um die Quadratgröße erhöhen.

for i=1: obj.Levels

lowerLeftXY = [obj.startXY(1) obj.startXY(2) + obj.BoxWidth \* (i-1)];

deltaXY = [obj.BoxWidth obj.BoxWidth];

rectangle('Position', [lowerLeftXY deltaXY], 'FaceColor', obj.faceColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

Die Quadrate werden jetzt zwar gestapelt und die Breite kann verändert werden aber es können aber keine mehr entnommen werden, deshalb benötigen wir jetzt eine private Löschfunktion „deleteStack“. Bei dieser Gelegenheit könnte man das Zeichnen des Stapels in eine privaten Funktion „drawStack“ auslagern. Hinweis: Wir müssen die Rechtecke jetzt einem indizierten Objekt zuweisen, damit wir es auch löschen können.

function obj = deleteStack(obj)

for i=1: obj.lastLevel

delete(obj.box(i));

end

end

Private Variable “lastLevel” einführen und den Wert Null zuweisen, damit nicht zu Beginn gelöst wird.

Aufruf in der “draw” Funktion: obj = obj.deleteStack();

Auslagern der Schleife für den Stapel in die Funktion „drawStack“

function obj = drawStack(obj)

for i=1: obj.Levels

lowerLeftXY = [obj.startXY(1) obj.startXY(2) + obj.BoxWidth \* (i-1)];

deltaXY = [obj.BoxWidth obj.BoxWidth];

obj.box(i) = rectangle('Position', [lowerLeftXY deltaXY], 'FaceColor', obj.faceColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

end

Aufruf in der “draw” Funktion unterhalb der delete-Funktion: obj = obj.drawStack();

Speichern der aktuellen Anzahl der Ebenen in die Variable lastLevel:

obj.LastLevel = obj.Levels;

Damit die Quadrate in jeder Ebene verkleinert werden können, muss nicht nur die Größe des Quadrates geändert werden, sondern auch der X und der Y-Startpunkt. Beginnen wir mit einer Bedingung in der Schleife unterhalb der Zeichenfunktion, die die Quadratgröße um 2 Einheiten verkleinert und bei einem minimalen Wert damit aufhört.

%reduce size

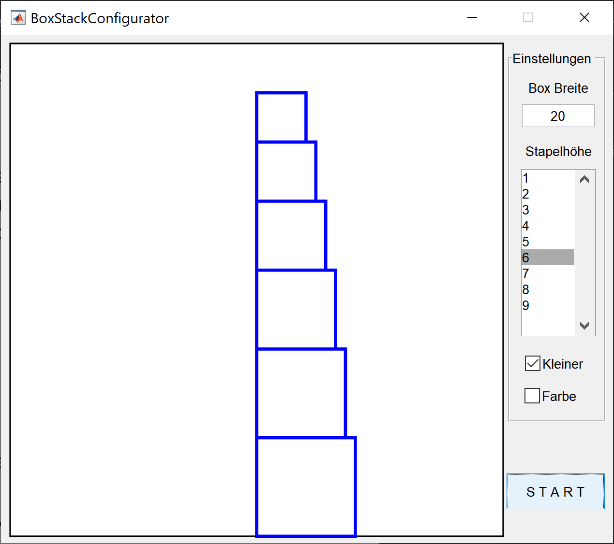
if obj.IsReduce

if obj.BoxWidth > 2

obj.BoxWidth = obj.BoxWidth - 2;

end

end

Würden wir das ausprobieren, sähe das merkwürdig aus, weil die Einfügepositionen nicht angepasst wurden. Deshalb beginnen wir bei dem Y-Wert und führen eine Variable „lastHeight“ mit dem Wert 0 am Anfang der Funktion ein und addieren sie anstelle des bisherigen Wertes zu dem Y-Startpunkt. Ist das Quadrat gezeichnet, wird zu der Variable der Wert der letzten Quadratgröße hinzugefügt.

Wenn wir es jetzt ausprobieren, haben wir ein linksbündigen Stapel.

Wenn wir erreichen wollen, dass das Quadrat mittig sitzt, dann müssen wir uns noch um den X-Wert kümmern.

function obj = drawStack(obj)

lastHeight = 0;

for i=1: obj.Levels

lowerLeftXY = [obj.startXY(1) obj.startXY(2) + lastHeight];

deltaXY = [obj.BoxWidth obj.BoxWidth];

obj.box(i) = rectangle('Position', [lowerLeftXY deltaXY], 'FaceColor', obj.faceColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

lastHeight = lastHeight + obj.BoxWidth;

%reduce size

if obj.IsReduce

if obj.BoxWidth > 2

obj.BoxWidth = obj.BoxWidth - 2;

end

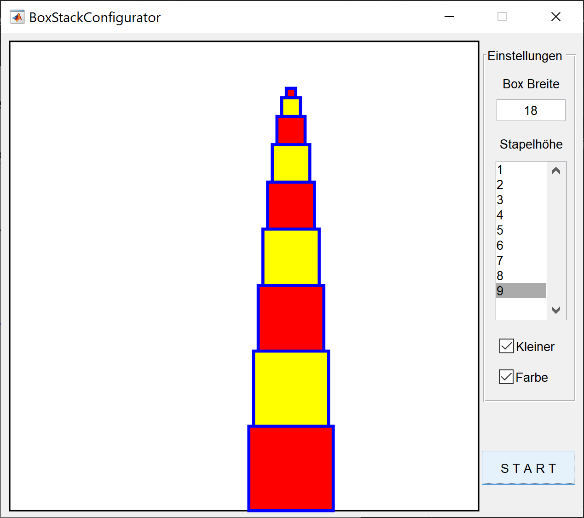
end

end

end

Dem X-Wert der linken unteren Ecke addieren wir einfach die aktuelle Anzahl der Ebenen, wenn der Stapel verkleinert werden soll:

lowerLeftXY = [obj.startXY(1) + obj.IsReduce \* i obj.startXY(2) + lastHeight];



Im letzten Schritt ergänzen wir noch eine Bedingung an den Anfang der Schleife für den Fall das Farbe angekreuzt ist.

%change color

if obj.IsColor

if obj.faceColor == 'w' || obj.faceColor == 'r'

obj.faceColor = 'y';

else

obj.faceColor = 'r';

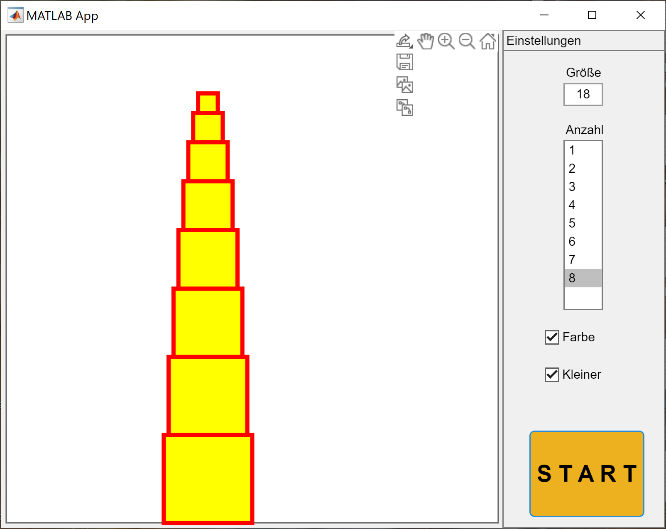
end

else

obj.faceColor = 'w';

end

Alle Funktionen ausprobieren, fertig. Vielen Dank für’s zuschauen.

classdef clsBoxStack

properties (Access = private)

UIAxes

boxStack

startXY

faceColor = 'white'

edgeColor = 'black'

lineWidth = 3

previousNo = 0

offset = 2

end

properties (Access = public)

BoxWidth = 10

NoOfBoxes = 1

IsColor = false

IsReduce = false

end

methods (Access = public)

% constructor

function obj = clsBoxStack(uiAxes, startX)

obj.UIAxes = uiAxes;

obj.startXY = [startX 0];

end

function obj = Start(obj)

obj = obj.deleteStack();

obj = obj.drawStack();

obj.previousNo = obj.NoOfBoxes;

end

end

methods (Access = private)

function obj = drawBox(obj, i, centerXY, boxWidth)

lowerLeftXY = [centerXY(1) - boxWidth/2 centerXY(2)];

deltaXY = [boxWidth boxWidth];

obj.boxStack(i) = rectangle( obj.UIAxes, ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor', obj.faceColor, ...

'EdgeColor', obj.edgeColor, ...

'LineWidth', obj.lineWidth);

end

function obj = deleteStack(obj)

if ~isempty(obj.boxStack)

for i = 1:obj.previousNo

delete(obj.boxStack(i));

end

end

end

function obj = drawStack(obj)

obj = obj.setColor();

boxWidth = obj.BoxWidth;

centerXY = [obj.startXY(1) - obj.BoxWidth/2 obj.startXY(2)];

for i = 1:obj.NoOfBoxes

obj = obj.drawBox(i, centerXY, boxWidth);

centerXY(2) = centerXY(2) + boxWidth;

if obj.IsReduce

boxWidth = boxWidth - obj.offset;

end

end

end

function obj = setColor(obj)

if obj.IsColor

obj.faceColor = 'y';

obj.edgeColor = 'r';

else

obj.faceColor = 'white';

obj.edgeColor = 'black';

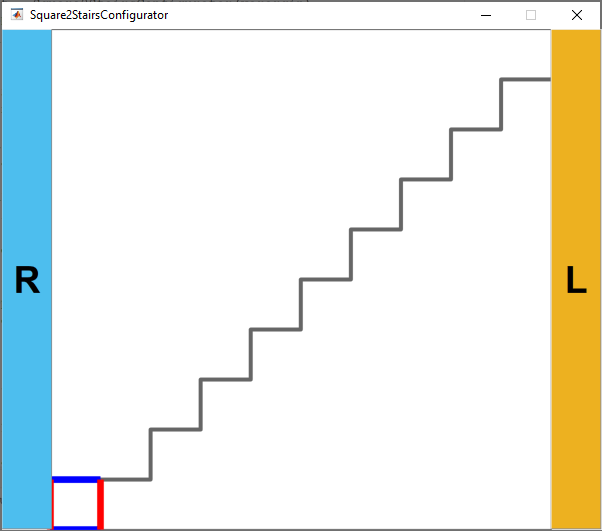
end

end

end

end

# Aufgabe: Quadrat auf Treppe Konfigurator

*Name: Square2StairsConfigurator*

*Opening Function*

function Square2StairsConfigurator\_OpeningFcn(hObject, eventdata, handles, varargin)handles.output = hObject;

…

% remove axis description

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

maxX = 100;

maxY = 100;

lowerLeft = [0 0];

upperRight = [maxX maxY];

% canvas border

rectangle('Position',[lowerLeft upperRight], 'EdgeColor', [.4, .4, .4], 'LineWidth',1);

global Stairs

Stairs = clsStairs(maxX);

*Callback-Functions*

function LeftButton\_Callback(hObject, eventdata, handles)

global Stairs

Stairs = Stairs.SquareUp();

function RightButton\_Callback(hObject, eventdata, handles)

global Stairs

Stairs = Stairs.SquareDown();

*Class*

classdef clsStairs

properties (Access = private)

maxX

minX = 0

delay = .2

%stairs

stairs

stairLength

stairLineWidth = 3

stairColor = [.4, .4, .4];

xDataB, xDataT, xDataL, xDataR

yDataB, yDataT, yDataL, yDataR

%square

squareLineWidth = 5

lineB, lineT, lineL, lineR

squareLength

squareColor = ['b' 'r']

isColorSwitch = false

isUp

lowerLeft = [0 0]

end

methods (Access = public)

% constructor

function obj = clsStairs(maxX)

obj.maxX = maxX;

obj.stairLength = maxX/10;

obj.squareLength = obj.stairLength;

obj = obj.drawStairs();

obj = obj.drawSquare();

end

function obj = SquareUp(obj)

obj.isUp = true;

obj = obj.upDownStairs();

end

function obj = SquareDown(obj)

obj.isUp = false;

obj = obj.upDownStairs();

end

end

methods (Access = private)

% draw stairs

function obj = drawStairs(obj)

xValues = [];

yValues = [];

for i = obj.minX:obj.stairLength:obj.maxX-obj.stairLength

xValues = [xValues i i+obj.stairLength];

yValues = [yValues i i];

end

obj.stairs = line(xValues,yValues, 'LineWidth',obj.stairLineWidth, 'Color', obj.stairColor);

end

% draw square

function obj = drawSquare(obj)

obj = obj.calculateSquare();

% buttom line

obj.lineB = line(obj.xDataB, obj.yDataB, 'Color', obj.squareColor(1),'LineWidth',obj.squareLineWidth);

% top line

obj.lineT = line(obj.xDataT, obj.yDataT, 'Color',obj.squareColor(1),'LineWidth',obj.squareLineWidth);

% left line

obj.lineL = line(obj.xDataL, obj.yDataL, 'Color',obj.squareColor(2),'LineWidth',obj.squareLineWidth);

% right line

obj.lineR = line(obj.xDataR, obj.yDataR, 'Color',obj.squareColor(2),'LineWidth',obj.squareLineWidth);

end

% move square

function obj = moveSquare(obj)

if ~isempty(obj.lineB)

obj = obj.calculateSquare();

% buttom line

obj.lineB.XData = obj.xDataB;

obj.lineB.YData = obj.yDataB;

obj.lineB.Color = obj.squareColor(1);

% top line

obj.lineT.XData = obj.xDataT;

obj.lineT.YData = obj.yDataT;

obj.lineT.Color = obj.squareColor(1);

% left line

obj.lineL.XData = obj.xDataL;

obj.lineL.YData = obj.yDataL;

obj.lineL.Color = obj.squareColor(2);

% right line

obj.lineR.XData = obj.xDataR;

obj.lineR.YData = obj.yDataR;

obj.lineR.Color = obj.squareColor(2);

end

end

% calculate square positions

function obj = calculateSquare(obj)

xL = obj.lowerLeft(1);

xR = obj.lowerLeft(1)+obj.squareLength;

yU = obj.lowerLeft(2);

yO = obj.lowerLeft(2)+obj.squareLength;

obj.xDataB = [xL xR];

obj.yDataB = [yU yU];

obj.xDataT = [xL xR];

obj.yDataT = [yO yO];

obj.xDataL = [xL xL];

obj.yDataL = [yU yO];

obj.xDataR = [xR xR];

obj.yDataR = [yU yO];

end

% set colors

function obj = setColors(obj)

% switch right left with top down color

if obj.isColorSwitch

obj.squareColor = ['b' 'r'];

else

obj.squareColor = ['r' 'b'];

end

obj.isColorSwitch = obj.isColorSwitch == false;

end

% square up- downstairs

function obj = upDownStairs(obj)

x = obj.lowerLeft(1);

if obj.isUp && x == 0

endY = obj.maxX - obj.squareLength;

step = obj.squareLength;

startY = obj.lowerLeft(2) + obj.squareLength;

else

endY = 0;

step = -obj.squareLength;

startY = obj.lowerLeft(2);

end

for y = startY: step: endY

obj.lowerLeft = [x y];

obj = obj.moveSquare();

pause(obj.delay);

obj = obj.setColors();

x = x + step;

if x > 0

obj.lowerLeft = [x y];

obj = obj.moveSquare();

pause(obj.delay);

end

end

end

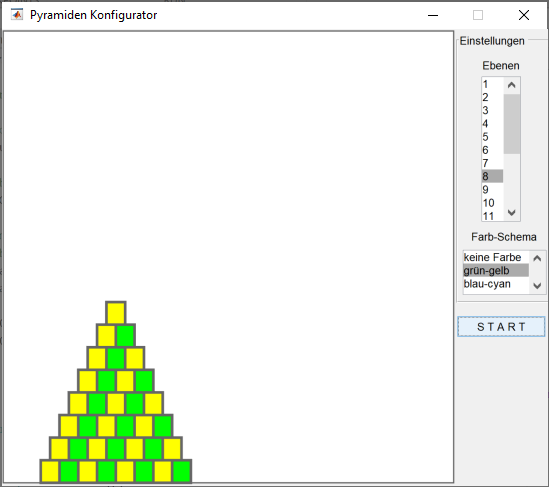
end

end

# Aufgabe: Pyramiden-Konfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „PyramidConfigurator“. Es soll eine konfigurierbare Pyramide aus Quadraten erzeugt werden, bei der die Anzahl der Ebenen und ein Farbschema eingestellt werden kann.

GUI-Name: PyramidConfigurator

Klasse: clsPyramid

Funktionen: draw

## Lösungsansatz

* Zunächst die GUI „PyramidConfigurator“ mit Hilfe des Befehls GUIDE anlegen und mit einem Seitenmenü versehen. In einer Group-Box sollen eine List für die Anzahl der Ebenen 1 -20 und eine List mit 3 verschiedenen Farb-Schemen sowie unterhalb der Gruppe der übliche Start-Button.
* Die Klasse clsPyramid anlegen, einen leeren Konstruktor und eine öffentliche Funktion „draw“ sowie die privaten Funktion „defineColorScheme“ anlegen, ebenfalls zunächst leer.
* Als öffentliche Eigenschaften benötigen wir die Anzahl der Ebenen und die Zahl des Farbschemas, die angibt welches Schema ausgewählt werden soll. Alle anderen Variablen können private angelegt werden.
* In der Opening-Funktion werden die Achsen ausgeschaltet und ein Rechteck mit xMax 120 und yMax 100 erzeugt. Danach wird die Pyramide als global gekennzeichnet, mit einem Startpunkt 10, 0 instanziiert und die Funktion „draw“ gestartet.
* Im Start-Button Callback werden die beiden Variablen aus dem Seitenmenü ausgelesen und der Klasse mitgegeben, anschließend wird die Funktion „draw“ erneut gestartet.
* Anschließend ein einfaches Rechteck in der „draw“ Funktion zeichnen und dann mit Hilfe von Bedingungen und Schleifen erweitern.
* Zuletzt die Auswahl des Farbschemas hinzufügen.

## Lösung Matlab Aufgabe: PyramidConfigurator

Willkommen bei der Lösung der Matlab-Aufgabe „PyramidConfigurator“.

Wir starten mit GUIDE und erzeugen die GUI „PyramidConfigurator“ mit einem Canvas und einem Seitenmenü. Hier fügen wir zunächst eine Groupbox mit dem Tag „settingsGroup“ und dem Title „Einstellungen“ ein.

In diese Box werden eingefügt:

* Ein Static-Text mit Namen „levelText“ und String „Ebenen“ sowie eine Liste mit Namen „levelList“ und den Werten „1-20“.
* Ein Static-Text mit Namen „colorSchemeText“ und String „Farb-Schema“ sowie eine Liste mit Namen „colorSchemeList“ und den Werten „Keine Farbe, grün-gelb, blau-cyan“.
* Den Start-Knopf legen wir unter die Groupbox an.

In der Opening-Function werden wie gehabt die Achsen ausgeschaltet, ein Rechteck bis 120, 100 gezeichnet, eine globale Variable für die Klasse „Pyramid“ angelegt und der Startpunkt auf 10, 0 festgelegt. Mit diesem Startpunkt wird die Klasse instanziiert und anschließend die „draw“ Funktion der Klasse aufgerufen.

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

xMinMax=[0 120];

yMinMax=[0 100];

lowerLeftXY = [xMinMax(1) yMinMax(1)];

deltaXY = [xMinMax(2) yMinMax(2)];

rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', [.4, .4, .4], 'LineWidth', 1);

global Pyramid

startXY = [10 0];

Pyramid = clsPyramid(startXY);

Pyramid = Pyramid.Draw();

Dann wird noch der StartButton Callback gefüllt mit:

global Pyramid

noOfLevels = handles.levelsListbox.Value;

colorScheme = handles.listboxColorScheme.Value;

Pyramid.NoOfLevels = noOfLevels;

Pyramid.NoOfColorScheme = colorScheme;

Pyramid = Pyramid.Draw();

Anschließend bauen wir die fast leere Hülle unserer neuen Klasse „clsPyramid“ auf:

* Mit den nicht öffentlichen Variablen für den Startpunkt, das Pyramidenobjekt, die Breite des Quadrates, die Farbe und die Linienstärke der Umrandung des Quadrates.
* Die öffentlichen Variablen sind die in der GUI änderbaren Werte der Ebenen und des Farbschemas.
* Den Konstruktor mit Übergabe des Startpunktes, die öffentliche Funktion „draw“.
* Die private Funktion „drawPyramid“ mit dem Rechteck und den zugehörigen Variablen.

classdef clsPyramid

properties (Access = private)

startXY

pyramid

squareSize = 5;

edgeColor = [.4, .4, .4];

lineWidth = 2;

end

properties (Access = public)

NoOfLevels = 1;

NoOfColorScheme = 1;

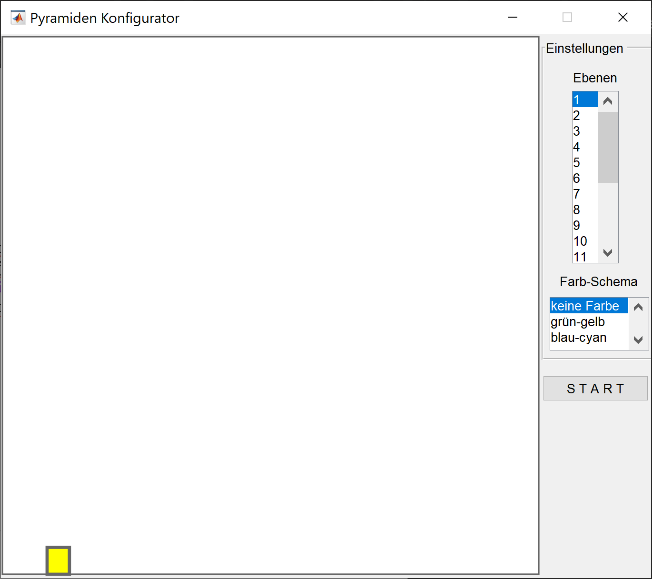
end

methods (Access = public)

% class constructor

function obj = clsPyramid(startXY)

obj.startXY = startXY;

 end

% draw function

function obj = Draw(obj)

obj = drawPyramid(obj);

end

end

methods (Access = private)

%draw new pyramid

function obj = drawPyramid(obj)

squareColor = 'y';

lowerLeft = [obj.startXY(1) obj.startXY(2)];

deltaXY = [obj.squareSize obj.squareSize];

obj.pyramid = rectangle('Position',[lowerLeft deltaXY], 'FaceColor', squareColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

end

end

Wenn das Programm gestartet wird, sollte das erste Rechteck in grauem Farbton auftauchen.

In einem weiteren Schritt fügen wir eine Bedingung in die „draw“ Funktion ein, um die Anzahl der Level zwischen 1 und 20 zu begrenzen und fügen dazu die privaten Variablen minLevel und maxLevel ein.

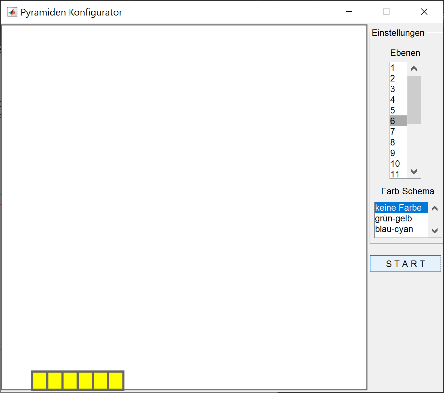
% draw function

function obj = Draw(obj)

if obj.NoOfLevels >= obj.minLevel && obj.NoOfLevels <= obj.maxLevels

obj = drawPyramid(obj);

end

 end

Nun überlegen wir uns eine Schleife für die erste Reihe von Quadraten. Sie könnte z.B. von 0 bis zur Anzahl von Ebenen-1 gehen, dann würde bei jedem Durchlauf zur linken unteren Ecke jeweils die mit dem Zähler multiplizierte Breite des Quadrates hinzugefügt. Das Objekt Pyramid muss jetzt mit dem Zähler indiziert werden:

for i = 0:obj.NoOfLevels-1

lowerLeft = [obj.startXY(1) + i\*obj.squareSize obj.startXY(2)];

deltaXY = [obj.squareSize obj.squareSize];

obj.pyramid(i+1) = rectangle('Position',[lowerLeft deltaXY], 'FaceColor', squareColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

Im nächsten Schritt wird noch eine Schleife für die einzelnen Ebenen benötigt, das bedeutet, dass die Endbedingung der gerade erzeugten Schleife nur beim ersten Mal der Anzahl der Ebenen entspricht, also benennen wir die Variable um und initialisieren sie mit der Anzahl der Ebenen-1. Hinter der Schleife muss dann die Anzahl der Quadrate reduziert werden. Das Objekt Pyramid muss jetzt mit beiden Zählern indiziert werden:

noOfSquares = obj.NoOfLevels - 1;

for i = 0:noOfSquares

lowerLeft = [obj.startXY(1) + i\*obj.squareSize obj.startXY(2)];

deltaXY = [obj.squareSize obj.squareSize];

obj.pyramid(i+1) = rectangle('Position',[lowerLeft deltaXY], 'FaceColor', squareColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

noOfSquares = noOfSquares - 1;

Am Programm sollte sich inhaltlich nichts geändert haben aber wir können jetzt eine zweite Schleife für die Ebenen darum herum bauen, den X-Wert der linken unteren Ecke um die halbe Breite und den Y-Wert um die Breite des Quadrates erhöhen.:

%draw new pyramid

function obj = drawPyramid(obj)

squareColor = 'y';

noOfSquares = obj.NoOfLevels - 1;

for j = 0:obj.NoOfLevels - 1

for i = 0:noOfSquares

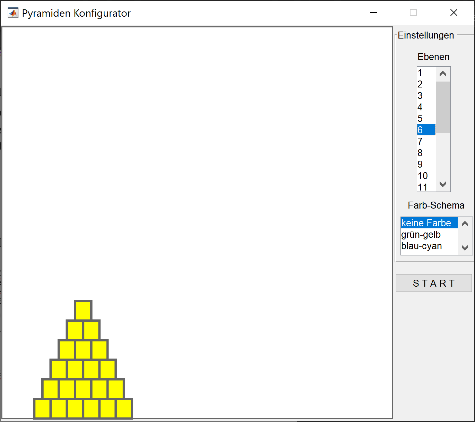
lowerLeft = [obj.startXY(1) + i\*obj.squareSize + j\*obj.squareSize/2 obj.startXY(2) + j\*obj.squareSize];

deltaXY = [obj.squareSize obj.squareSize];

obj.pyramid(j+1, i+1) = rectangle('Position',[lowerLeft deltaXY], 'FaceColor', squareColor, 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

end

noOfSquares = noOfSquares - 1;

 end

end

Das Programm zeichnet jetzt eine Pyramide, jedoch kann die Anzahl nicht mehr kleiner gemacht werden, weil die Quadrate nicht gelöscht werden. Wir müssen uns also die letzte Anzahl der Ebenen merken, dann bauen wir noch eine private Löschfunktion „deletePyramid“ ein und rufen sie in der „draw“ Funktion vorher auf.

function obj = draw(obj)

if obj.NoOfLevels >= obj.minLevel && obj.NoOfLevels <= obj.maxLevel

obj = deletePyramid(obj);

obj = drawPyramid(obj);

end

end

%delete latest pyramid

function obj.deletePyramid(obj)

noOfSquares = obj.latestLevel;

for i=1:obj.latestLevel

for j=1:noOfSquares

delete(obj.pyramid(i, j))

end

noOfSquares = noOfSquares-1;

end

end

Fehlt nur noch die Änderung des Farbschemas. Diese Funktion „defineColorScheme“ sollte ebenfalls private sein und wird in der Funktion „drawPyramid“ benötigt.

%define color scheme

function squareColors = defineColorScheme(obj)

switch obj.NoOfColorScheme

case 1

squareColors(1) = 'w';

squareColors(2) = 'w';

case 2

squareColors(1) = 'y';

squareColors(2) = 'g';

case 3

squareColors(1) = 'c';

squareColors(2) = 'b';

otherwise

sqaureColors(1) = 'w';

sqaureColors(2) = 'w';

end

end

Der Aufruf und die zugehörige Anpassung eines Color-Index in der „drawPyramid“ Funktion:

%draw new pyramid

function obj = drawPyramid(obj)

squareColors = defineColorScheme(obj);

obj.latestNoOfLevels = obj.NoOfLevels;

noOfSquares = obj.NoOfLevels - 1;

for j = 0:obj.NoOfLevels - 1

colorIndex = 1;

for i = 0:noOfSquares

lowerLeft = [obj.startXY(1) + i\*obj.squareSize + j\*obj.squareSize/2 obj.startXY(2) + j\*obj.squareSize];

deltaXY = [obj.squareSize obj.squareSize];

obj.pyramid(j+1, i+1) = rectangle('Position',[lowerLeft deltaXY], 'FaceColor', squareColors(colorIndex), 'EdgeColor', obj.edgeColor, 'LineWidth', obj.lineWidth);

if colorIndex == 1

colorIndex = 2;

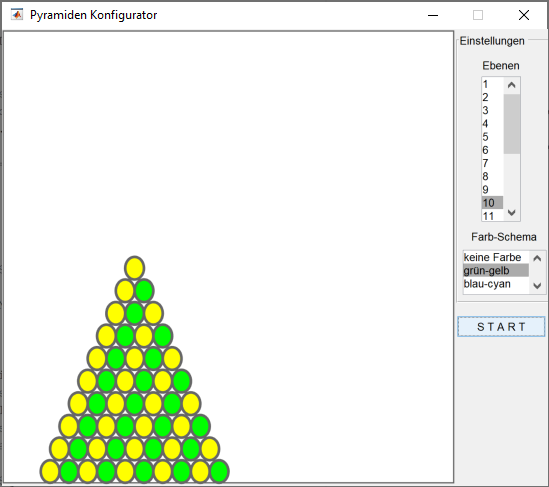
else

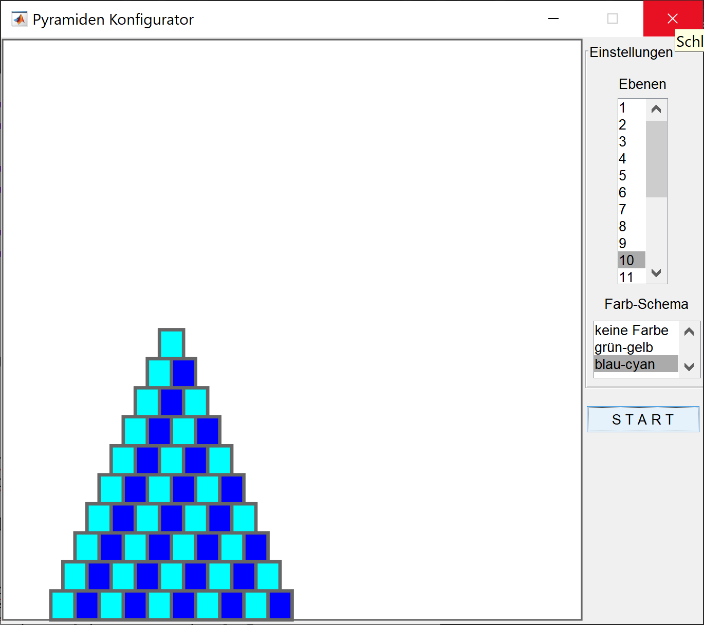
colorIndex = 1;

end

end

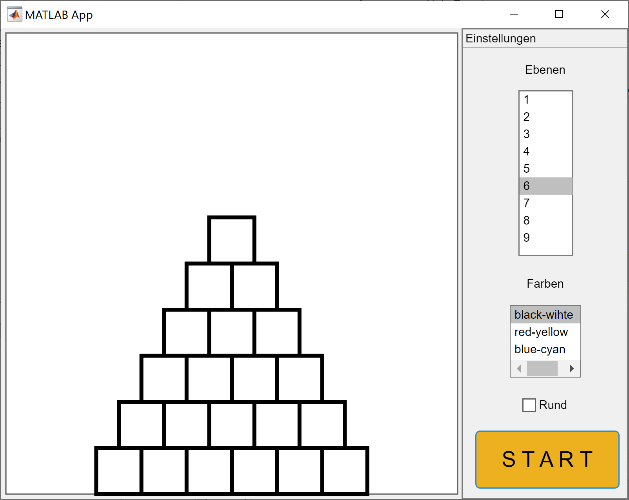
noOfSquares = noOfSquares - 1;

 end

 end

Alternative: mit Kreisen

Alle Funktionen ausprobieren, fertig. Vielen Dank für’s zuschauen.

classdef clsPyramid

properties (Access = private)

UIAxes

centerX

size = 10

pyramid

edgeColor = 'k'

faceColor = 'white'

lineWidth = 3

previousLevels

end

properties (Access = public)

Levels = 1

Color = 'black-white'

IsCircular = false

end

methods (Access = public)

% constructor

function app = clsPyramid(uiAxes, centerX)

app.UIAxes = uiAxes;

app.centerX = centerX;

end

function app = Draw(app)

app = app.deleteBlocks();

app = app.drawPyramid();

app.previousLevels = app.Levels;

end

end

methods (Access = private)

function app = drawPyramid(app)

blocks = app.Levels;

centerY = 0;

centerXY = [app.centerX-app.Levels\*app.size/2 centerY];

for i=1:app.Levels

for j=1:blocks

lowerLeftXY = ...

[centerXY(1) + app.size \* (j-1) + app.size/2 \* (i-1) ...

centerXY(2) + app.size \* (i-1)];

app = app.drawBlock(i,j,lowerLeftXY);

pause(.1);

end

blocks = blocks -1;

end

end

function app = drawBlock(app,i,j, lowerLeftXY)

deltaXY = [app.size app.size];

curvature = [0 0];

if app.IsCircular

curvature = [1 1];

end

app.pyramid(i,j) = rectangle(app.UIAxes, ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',app.lineWidth, ...

'Curvature', curvature);

end

function app = deleteBlocks(app)

if ~isempty(app.pyramid)

blocks = 1;

for i=app.previousLevels:-1:1

for j=blocks:-1:1

delete(app.pyramid(i, j));

pause(.1);

end

blocks = blocks + 1;

end

end

end

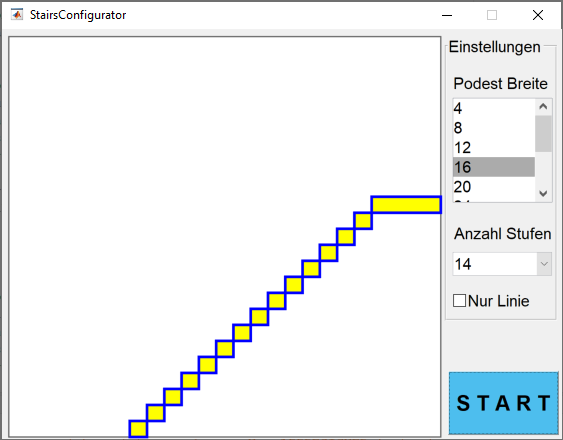
end

end

# Aufgabe: Treppen-Konfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „Treppenkonfigurator“. Es soll ein Konfigurator erzeugt werden, der zunächst ein Podest in der ausgewählten Höhe zeichnet und anschließend eine Treppe bis zu diesem Podest konstruiert. Die Treppe soll sowohl mit Hilfe von Rechtecken als auch mit Linien erzeugt werden können.

GUI-Name: StairsConfigurator

Funktionen: Draw, drawFloor und drawStairs

## Lösungsansatz

* Zunächst die GUI „StairsConfigurator“ mit Hilfe des Befehls GUIDE anlegen und mit einem Seitenmenü versehen. Es wird gefüllt mit einem Static-Text, einem Popup-Menü mit Werten einer 4er Reihe bis 40 und zusätzlich einer Checkbox.
* In der Opening-Funktion werden die Achsen ausgeschaltet, ein Maximalwert von 50 in einer Variablen gespeichert und damit ein Rechteck erzeugt. Danach wird das Objekt „Stairs“ als global gekennzeichnet, mit dem Maximalwert als „clsStairs“ instanziiert.
* Im Start-Button Callback werden die aktuellen Werte der Höhe und ob es als Linie erzeugt werden soll aus dem Seitenmenü ausgelesen, dem Objekt „Stairs“ zugewiesen und die Funktion „Draw“ des Objektes gestartet.
* Danach die Klasse clsStairs anlegen, einen Konstruktor mit Übergabe des Maximalwertes und die öffentliche Funktion „Draw“ sowie die privaten Funktionen „drawFloor“, „drawStairs“ anlegen, zunächst leer.
* Als öffentliche Eigenschaften benötigen wir die Höhe und eine logische Variablen, die angibt, ob es mit Linien erzeugt werden sollen.
* Anschließend ein einfaches Rechteck in der „drawStairs“ Funktion zeichnen und dann diese Funktion in der „Draw“ Funktion aufrufen.
* Programm mit Hilfe von Bedingungen und Schleifen um die angeforderten Darstellungen erzeugen.
* Zuletzt das Zeichnen der Treppe mit Linien hinzufügen.

## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der Matlab-Aufgabe „StairsConfigurator“.

Die GUI, die Opening-Function und der StartButton-Callback sollten jetzt gefüllt werden können und sehen wie folgt aus:

Beschreiben!

Gehen wir jetzt auf die Klasse „clsStairs“ näher ein und laden das vorbereitete leere Template. Hier ändern wir den Klassen- und Konstruktor-Namen und fügen die notwendigen leeren Funktionen ein: öffentlich „Draw“, privat „drawFloor“, „drawStairs“. Anschließend die öffentlichen Variablen mit Vorgabewerten:

Height = 0;

IsLine = false;

Dann können wir im Konstruktor den Maximalwert der Zeichenfläche als private Variable maxWidth in die Klasse übertragen.

Fügen wir als nächstes ein Rechteck in beide privaten draw-Funktionen ein und ergänzen dafür die privaten Variablen. Für das delta definieren wir die stairSize = 2 und das Podest soll eine Breite von 8 haben, floorWidth = 8. Jetzt haben wir die notwendigen Berechnungsgrundlagen für den X und Y-Wert:

lowerLeftXY = [obj.maxWidth - obj.floorWidth obj.Height - obj.stairSize];

deltaXY = [obj.floorWidth obj.stairSize];

Rufen wir damit zunächst nur die drawFloor Funktion in der öffentlichen Funktion draw auf und testen das Podest. Es kann angelegt werden, wird aber nicht gelöscht, deshalb fügen wir eine Löschfunktion in die draw-Funktion ein:

if ~isempty(obj.floor)

delete(obj.floor);

end

Bauen wir dann schon die Wert für die erste Stufe ein, wir können Teilwerte aus dem Podest übernehmen und ergänzen:

lowerLeftXY = [obj.maxWidth - obj.floorWidth - obj.stairSize

obj.Height - obj.stairSize \* 2];

deltaXY = [obj.stairSize obj.stairSize];

Der Variable stairs müssen wir den Index zuordnen:

obj.stairs(i) = rectangle( ...

Rufen wir damit jetzt auch die drawStairs Funktion in der öffentlichen Funktion draw auf und ergänzen es um einen Löschbefehl und testen es:

function obj = Draw(obj)

if ~isempty(obj.floor)

delete(obj.floor);

end

if ~isempty(obj.stairs)

delete(obj.stairs);

end

obj = obj.drawFloor();

obj = obj.drawStairs();

end

Es folgt die Schleife für die Stufen der Treppe. Berechnen wir dazu in der drawStairs-Funktion die Anzahl der Stufen:

noOfStairs = obj.Height/obj.stairSize - 1;

…und lassen die Schleife bis zu dieser Variable laufen und umfassen damit das Rechteck.

for i = 1:noOfStairs

Der X und der Y-Wert müssen beide in jedem Durchlauf um eine SquareSize reduziert werden!

lowerLeftXY = [obj.maxWidth - obj.floorWidth - obj.stairSize \* i obj.Height - obj.stairSize \* (i+1)];

Jetzt überprüfen, ob eine Treppe erzeugt wird.

Sicherheitshalber lagern wir das Löschen aus in die Funktion deleteStairs und löschen jede einzelne Stufe. Dabei müssen wir uns die letzte Anzahl Stufen am Ende der Funktion drawStairs merken! Nicht vergessen den Aufruf der Löschfunktion in der Draw-Funktion zu ändern:

function obj = deleteStairs(obj)

for i = 1:obj.latestNoOfStairs

delete(obj.stairs(i));

end

end

und

if ~isempty(obj.stairs)

delete(obj.stairs);

end

Da wir die X- und Y-Werte schon haben, ist der Einbau der Bedingung eine Linie zu verwenden, einfach. Wir können sie in die Schleife einfügen und das Rechteck in den Else-Zweig setzen.

if obj.IsLine

else

end

dann

if obj.IsLine

else

lowerLeftXY = [obj.maxWidth - obj.floorWidth - obj.stairSize \* i obj.Height - obj.stairSize \* (i+1)];

deltaXY = [obj.stairSize obj.stairSize];

obj.stairs(i) = rectangle( ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor', obj.faceColor, ...

'EdgeColor', obj.edgeColor, ...

'LineWidth', obj.lineWidth, ...

'Curvature', [0 0]);

end

In den If-Zweig setzen wir ein Linien-Template und füllen die x- und y-Werte. Für eine Stufe gibt es 3 Werte, deshalb eignen sich hier indizierte Variablen gut. Der erste X-Wert entspricht dem X-Wert des Rechtecks, der zweite ändert sich nicht und der dritte ist um eine Stufenbreite erweitert. Der erste Y-Wert entspricht dem Rechteck, die beiden weiteren müssen auch um eine Stufenbreite erweitert werden.

xValues(1) = obj.maxWidth - obj.floorWidth - obj.stairSize \* i;

xValues(2) = obj.maxWidth - obj.floorWidth - obj.stairSize \* i;

xValues(3) = obj.maxWidth - obj.floorWidth - obj.stairSize \* i + obj.stairSize;

yValues(1) = obj.Height - obj.stairSize \* (i+1);

yValues(2) = obj.Height - obj.stairSize \* i;

yValues(3) = obj.Height - obj.stairSize \* i;

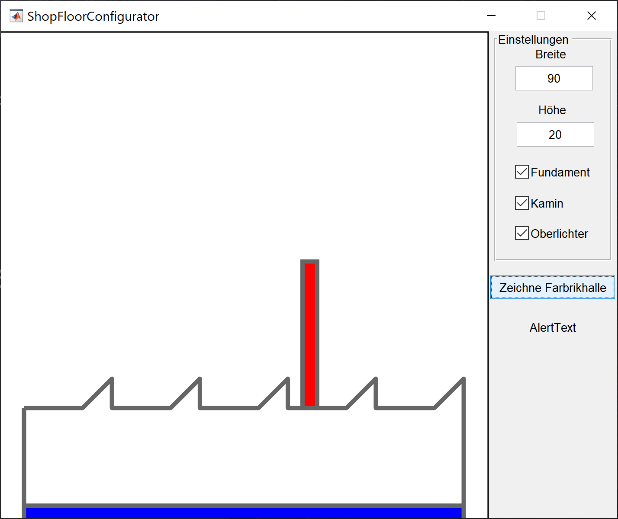
obj.stairs(i) = line(xValues, yValues, 'Color', obj.edgeColor, 'LineWidth', obj.lineWidth);

Und fertig ist der Konfigurator!

# Aufgabe ShopFloorConfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll eine Industriehalle gezeichnet werden.

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „ShopFloorConfigurator“. Es soll ein Konfigurator für die Darstellung einer Fabrikhalle erzeugt werden, bei dem folgende Angaben einstellbar oder aktivierbar sein sollen:

* die Breite und Höhe,
* Darstellung eines Fundamentes,
* eines Kamines
* Oberlichter.

GUI-Name: ShopFloorConfigurator

Klasse: clsShopFloor

Eigenschaften:

* Width = 40;
* Height = 20;
* IsBase = false;
* IsChimney = false;
* IsRoofLights = false;
* centerX - für den Startpunkt
* walls - für das Objekt der Wände
* base - für das Objekt des Fundamentes
* chimney - für das Objekt des Kamins
* roof - für das Objekt der Dachkonstruktion
* edgeColor = [.4 .4 .4];
* lineWidth = 3;
* baseFaceColor = 'b';
* baseHeight = 3;
* chimneyWidth = 3;
* chimneyFaceColor = 'r';

Funktionen:

* Draw
* delete
* drawWalls
* drawBase
* drawChimney
* drawRoof

Lösungsansatz ShopFloorConfigurator

* Zunächst die GUI „ShopFloorConfigurator“ mit Hilfe des Befehls GUIDE anlegen und mit einem Seitenmenü versehen und den angegebenen Steuerelementen füllen.
* In der Opening-Funktion werden die Achsen ausgeschaltet und ein Rechteck mit 100, erzeugt und der AlertText unsichtbar geschaltet. Danach wird das Objekt „ShopFloor“ als global gekennzeichnet, mit einem Startpunkt 5, 0 als „clsShopFloor“ instanziiert.
* Die aktuellen Werte Breite, Höhe, ob mit Fundament, ob mit Kamin oder ob mit Dachlichtern werden aus den „handles“ gelesen, dem Objekt „ShopFloor“ zugewiesen und die Funktion „Draw“ des Objektes gestartet.
* Im Start-Button Callback werden erneut die fünf Variablen aus dem Seitenmenü ausgelesen, der Klasse mitgegeben und anschließend die Funktion „Draw“ erneut gestartet. Man sollte den Aufruf noch mit einer Bedingung umfassen, damit die Hallenbreite den minimalen Wert von 10, maximalen Wert von 100, die Höhe den minimalen Wert von 10 und maximalen Wert von 50 nicht über- oder unterschreiten.
* Danach die Klasse clsShopFloor anlegen, einen Konstruktor mit Übergabe des Startpunktes und die öffentliche Funktion „Draw“ sowie die privaten Funktionen „drawWalls“, „drawBase“, „drawChimney“ und „drawRoof“ anlegen, zunächst leer.
* Als öffentliche Eigenschaften benötigen wir die Breite, die Höhe, logische Variablen, die angeben, ob ein Fundament, ein Kamin oder Dachlichter erzeugt werden sollen.
* Anschließend ein einfaches Rechteck in der „drawWall“ Funktion zeichnen und dann diese Funktion in der „Draw“ Funktion aufrufen.
* Programm mit Hilfe von Bedingungen und Schleifen um die angeforderten Darstellungen erzeugen.
* Zuletzt das Zeichnen des Daches mit Dachlichtern hinzufügen.

## Lösung Matlab Aufgabe: ShopFloorConfigurator

Willkommen bei der Lösung der Matlab-Aufgabe „ShopFloorConfigurator“

Die GUI, die Opening-Function und der StartButton-Callback sollten jetzt gefüllt werden können.

Gehen wir deshalb direkt auf die Klasse „clsShopFloor“ näher ein und laden das vorbereitete leere Template. Hier ändern wir den Klassen- und Konstruktor-Namen und fügen die notwendigen leeren Funktionen ein: öffentlich „Draw“, privat „drawWalls“, „drawBase“, „drawChimney“, „drawRoof“. Jetzt die öffentlichen Variablen mit Vorgabewerten:

* Width = 40;
* MinWidth = 20;
* MaxWidth = 100;
* Height = 20;
* MinHeight = 10;
* MaxHeight = 50;
* IsBase = false;
* IsChimney = false;
* IsRoofLights = false;

Die schon absehbaren privaten Variablen könnten sein:

* startXY für den Startpunkt
* walls für das Objekt der Wände
* base für das Objekt des Fundamentes
* chimney für das Objekt des Kamins
* roof für das Objekt der Dachkonstruktion

In der öffentlichen Funktion „Draw“ wird zunächst nur die Funktion „drawWalls“ aufgerufen. In dieser Fügen wir ein Linienobjekt ein und ergänzen die notwendigen Variablen.

Funktion: Draw

function obj = Draw(obj)

if ~isempty(obj.walls)

delete(obj.walls)

end

if ~isempty(obj.base)

delete(obj.base);

end

if ~isempty(obj.chimney)

delete(obj.chimney);

end

if ~isempty(obj.roof)

delete(obj.roof)

end

if obj.IsBase

obj = obj.drawBase();

obj.startXY = [obj.initialStartXY(1) obj.initialStartXY(2)+obj.baseHeight];

else

obj.startXY = obj.initialStartXY;

end

obj = obj.drawWalls();

if obj.IsChimney

obj = obj.drawChimney();

end

if obj.IsRoofLights

obj = obj.drawRoof();

end

end

Funktion: drawWalls

function obj = drawWalls(obj)

xValues(1) = obj.startXY(1);

xValues(2) = obj.startXY(1);

xValues(3) = obj.startXY(1) + obj.Width;

xValues(4) = obj.startXY(1) + obj.Width;

yValues(1) = obj.startXY(2) + obj.Height;

yValues(2) = obj.startXY(2);

yValues(3) = obj.startXY(2);

yValues(4) = obj.startXY(2) + obj.Height;

if ~obj.IsRoofLights

xValues(5) = obj.startXY(1);

yValues(5) = obj.startXY(2) + obj.Height;

end

obj.shopFloor = line(xValues, yValues, 'Color', obj.wallColor, 'LineWidth', obj.lineWidth);

end

Funktion: drawBase

function obj = drawBase(obj)

deltaXY = [obj.Width obj.baseHeight];

obj.base = rectangle('Position',[obj.initialStartXY deltaXY], 'FaceColor', obj.baseColor, ...

'EdgeColor', obj.wallColor, 'LineWidth', obj.lineWidth);

end

Funktion: drawChimney

function obj = drawChimney(obj)

lowerLeftXY = [obj.startXY(1) + obj.Width/3 \* 2 - obj.chimneyWidth obj.startXY(2) + obj.Height];

deltaXY = [obj.chimneyWidth obj.chimneyHeight];

obj.chimney = rectangle('Position',[lowerLeftXY deltaXY], 'FaceColor', ...

obj.chimneyColor, 'EdgeColor', obj.wallColor, 'LineWidth', obj.lineWidth);

end

Funktion: drawRoof

function obj = drawRoof(obj)

% calculate rooflight areas

noOfRoofLights = fix(obj.Width/(obj.roofLightWidth+obj.roofLightDistance));

% calculate rest of roof, roof minus rooflight areas

restRoof = obj.Width - (noOfRoofLights \* (obj.roofLightWidth+obj.roofLightDistance));

% calculate roof points

xValues(1) = obj.startXY(1);

yValues(1) = obj.startXY(2) + obj.Height;

xValues(2) = obj.startXY(1) + restRoof/2;

yValues(2) = yValues(1);

if restRoof/2 >= obj.roofLightWidth

xValues(3) = xValues(2) + obj.roofLightWidth;

yValues(3) = yValues(1) + obj.roofLightHeight;

xValues(4) = xValues(3);

yValues(4) = yValues(1);

else

xValues(3) = xValues(2);

yValues(3) = yValues(1);

xValues(4) = xValues(2);

yValues(4) = yValues(1);

end

counter = 5;

for i = 1:noOfRoofLights

%distance

xValues(counter) = xValues(counter-1) + obj.roofLightDistance;

yValues(counter) = yValues(1);

counter = counter+1;

%rooflight

xValues(counter) = xValues(counter-1) + obj.roofLightWidth;

yValues(counter) = yValues(1) + obj.roofLightHeight;

counter = counter+1;

xValues(counter) = xValues(counter-1);

yValues(counter) = yValues(1);

counter = counter+1;

end

xValues(counter) = obj.startXY(1) + obj.Width;

yValues(counter) = yValues(1);

obj.roof = line(xValues, yValues, 'Color', obj.wallColor, 'LineWidth', obj.lineWidth);

end

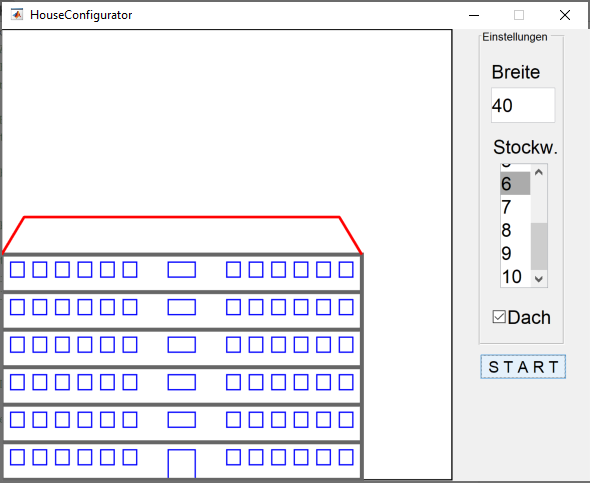
private Properties:

* startXY
* initialStartXY
* walls
* base
* roof
* chimney
* wallColor = [.4 .4 .4];
* lineWidth = 3;
* baseColor = 'b';
* baseHeight = 3;
* chimneyWidth = 3;
* chimneyHeight = 30;
* chimneyColor = 'r';
* roofLightWidth = 6;
* roofLightHeight
* roofLightDistance
* roofXValues
* roofYValues

# Aufgabe: Haus-Konfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „Haus Konfigurator“. Es soll ein konfigurierbares Haus in der Vorderansicht erzeugt werden, bei dem die Breite, Anzahl der Stockwerke und die Erzeugung eines Daches eingestellt werden kann.

GUI-Name: HouseKonfigurator

Klasse: clsHouse

mit den Funktionen: draw, drawWalls, drawRoof, drawOpenings

## Lösungsansatz

* Zunächst die GUI „HouseConfigurator“ mit Hilfe des Befehls GUIDE anlegen und mit einem Seitenmenü versehen. In einer Group-Box sollen ein Eingabefeld für die Breite des Houses, ein Dropdown für die Anzahl der Stockwerke 1 -10, eine Check-Box für die Auswahl eines Daches sowie unterhalb der Gruppe der übliche Start-Button. Dieses Mal soll noch ein Static Text unterhalb des Start-Buttons stehen als „alertText“, zunächst ohne Inhalt.
* In der Opening-Funktion werden die Achsen ausgeschaltet und ein Rechteck mit xMax 50 und yMax 30 erzeugt. Danach wird das Objekt „House“ als global gekennzeichnet, mit einem Startpunkt 0, 0 instanziiert.
* Die aktuellen Werte Breite und Stockwerk werden aus den „handles“ gelesen, dem Objekt „House“ zugewiesen und die Funktion „draw“ gestartet.
* Im Start-Button Callback werden die drei Variablen aus dem Seitenmenü ausgelesen und der Klasse mitgegeben, anschließend wird die Funktion „draw“ erneut gestartet. Man sollte den Aufruf noch mit einer Bedingung umfassen, damit die Hausbreite den minimalen Wert von 10 nicht unterschreitet und den maximalen Wert von 50 nicht überschreitet.
* Danach die Klasse clsHouse anlegen, einen Konstruktor mit Übergabe des Startpunktes und die öffentliche Funktion „draw“ sowie die privaten Funktionen „drawWall“, „drawRoof“ und „drawOpenings“ anlegen, zunächst leer.
* Als öffentliche Eigenschaften benötigen wir die Breite, die Anzahl der Stockwerke und eine logische Variable, die angibt, ob ein Dach erzeugt werden soll.
* Anschließend ein einfaches Rechteck in der „drawWall“ Funktion zeichnen und dann diese Funktion in der „draw“ Funktion aufrufen.
* Programm mit Hilfe von Bedingungen und Schleifen um Stockwerke und Dach erweitern.
* Zuletzt das Zeichnen der Öffnungen hinzufügen.

## Lösung Matlab Aufgabe: HouseConfigurator

Willkommen bei der Lösung der Matlab-Aufgabe „HouseConfigurator“.

Wir starten mit GUIDE und erzeugen die GUI „HouseConfigurator“ mit einem Canvas und einem Seitenmenü. Hier fügen wir eine Groupbox („settingsGroup“) und dem Title „Einstellungen“ ein. Sie soll eine Input-box, Listbox und Checkbox enthalten. Darunter wie immer der Start-Button. Darunter soll noch ein Statictext als Fehlertext gesetzt werden.

In diese Box werden eingefügt:

* Einen Static-Text mit der Fontsize 14, dem Namen „widthText“ und String „Breite“ sowie eine Inputbox mit dem Namen „widthInput“ und dem String 40.
* Einen Static-Text mit der Fontsize 14, dem Namen „floorsText“ und String „Stockw.“ und eine Liste mit der Fontsize 14, dem Namen „noOfFloorsList“ und den Werten „1-10“.
* Eine Checkbox mit der Fontsize 14, dem Namen „roofCheck“ und String „Dach“.
* Den Start-Knopf legen wir wieder unterhalb der Groupbox an und darunter noch den Static-Text mit der Fontsize 14, dem Namen „alertText“ aber ohne Inhalt.

In der Opening-Function werden wie gehabt die Achsen ausgeschaltet, ein Rechteck mit den Deltawerten 50, 30 gezeichnet und die Sichtbarkeit des „alertText“ ausgeschaltet.

Dann eine globale Variable für die Klasse „House“ anlegen, den Startpunkt auf 0, 0 festlegen und die Klasse instanziieren.

handles.Canvas.YAxis.Visible = 'off'; % remove y-axis

handles.Canvas.XAxis.Visible = 'off'; % remove x-axis

handles.alertText.Visible = 'off';

lowerLeftXY = [0 0];

deltaXY = [50 30];

rectangle('Position', [lowerLeftXY deltaXY], 'EdgeColor', [.4, .4, .4], 'LineWidth', 1);

global House

startXY = [0 0];

**House = clsHouse(startXY);**

**House.Width = str2double(handles.widthInput.String);**

**House.NoOfFloors = handles.noOfFloorsListBox.Value;**

**House.HasRoof = handles.roofCheckBox.Value;**

**House=House.draw();**

Wir lesen die Werte für die Breite und das Stockwerk aus den Steuerelementen der GUI und weisen sie der Klasse zu. Erst dann wir die „draw“ Funktion der Klasse aufgerufen.

Im Startbutton-Callback wird wieder das globale Objekt „House“ eingetragen, der „alertText“ ausgeschaltet und die Variablen für die Breite und Anzahl der Stockwerke ausgelesen.

Zusätzlich soll noch eine Bedingung eingefügt werden, damit die Breite von 10 nicht unterschritten und die Breite von 50 nicht überschritten wird. Für diesen Fall soll der „alertText“ mit einem geeigneten Warnhinweis versehen werden und anschließend eingeschaltet.

global House

handles.alertText.Visible = 'off';

houseWidth = str2double(handles.widthInput.String);

if houseWidth < House.MinWidth || houseWidth > House.MaxWidth

alertText = strcat("Minimale Hausbreite ist ", int2str(House.MinWidth), "Maximale Hausbreite ist ", int2str(House.MaxWidth));

handles.alertText.string = alertText;

handles.alertText.Visisble = 'on';

else

House.NoOfFloors = handles.noOfFloorsListBox.Value;

House.HasRoof = handles.roofCheckBox.Value;

House.Width = houseWidth;

House = House.Draw();

end

Stopp: Leere Klasse kopieren!!!

Danach die Klasse clsHouse anlegen, einen Konstruktor mit Übergabe des Startpunktes und die öffentliche Funktion „draw“ sowie die privaten Funktionen „drawWall“, „drawRoof“ und „drawOpenings“ anlegen, zunächst leer.

Als öffentliche Eigenschaften benötigen wir die Breite mit dem Vorgabewert 10, die Anzahl der Stockwerke mit dem Vorgabewert 1 und eine logische Variable mit dem Vorgabewert false, die angibt, ob ein Dach erzeugt werden soll. Die maximale und die minimale Breite hängt von der Größe des Canvas ab, deshalb sollten auch diese beiden Variablen öffentlich sein.

Die Höhe für ein Stockwerk sollte als private Variable „floorHeight“ mit dem Vorgabewert 2.5 angelegt werden.

Anschließend ein einfaches Rechteck in der „drawWall“ Funktion zeichnen, die zugehörigen Variablen ableiten und dann diese Funktion in der „draw“ Funktion aufrufen.

classdef clsHouse

properties (Access=private)

startXY

floorHeight = 2.5;

end

properties

Width = 10;

NoOfFloors = 1;

HasRoof = false;

MinWidth = 10.0

MaxWidth = 50.0

end

methods

function obj = clsHouse1(startXY)

obj.startXY = startXY;

end

function obj = draw(obj)

obj = obj.drawWalls();

end

end

methods (Access=private)

function obj = drawWalls(obj)

lowerLeftXY = obj.startXY;

deltaXY = [obj.Width obj.floorHeight];

rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.wallColor, 'LineWidth', obj.lineWidth);

end

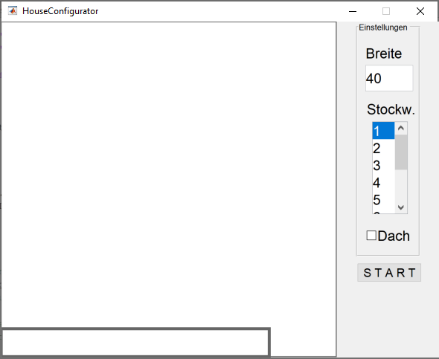
lowerLeftXY = [];

deltaXY = [];

function obj = drawRoof(obj)

end

function obj = drawOpenings(obj, floorNo)



end

end

end

Wenn das Programm jetzt gestartet wird, sollte ein Rechteck für die erste Etage erscheinen. Die Breite lässt sich zwischen dem Min – und Maxwert ändern, jedoch bleibt das vorherige Rechteck noch erhalten.

Testen!!!

Wir müssen also noch die Stockwerke übereinander stapeln und benötigen eine Löschfunktion.

Zum Löschen benötigen wir eine Variable „floor“, in der wir die Rechtecke speichern können.

Die Schleife um das Rechteck geht bis zur Anzahl der Stockwerke. Zu dem Y-Wert der unteren, rechten Ecke des Rechtecks wird die Stockwerkhöhe addiert, aber Achtung erst ab dem 2 Stock!

function obj = drawWalls(obj)

for i = 1:obj.NoOfFloors

lowerLeftXY = [obj.startXY(1) obj.startXY(2) + obj.floorHeight \* (i-1)];

deltaXY = [obj.Width obj.floorHeight];

obj.floor(i) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.wallColor, 'LineWidth', obj.lineWidth);

end

end

In der „Draw“ Funktion müssen wir noch die Anzahl der zuletzt gezeichneten Stockwerke speichern, z.B. in der privaten Variable „latestFloorNo“. Mit dieser Variable lässt sich die Löschfunktion, die eine Schleife enthält, aufrufen, aber vorher müssen wir prüfen, ob Wände überhaupt existieren mit if ~isempty …

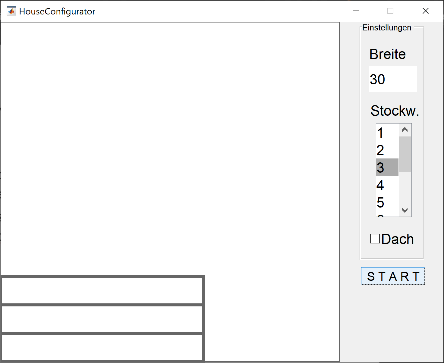
function obj = Draw(obj)

if ~isempty(obj.floor)

obj = obj.deleteFloor();

end

obj = obj.drawWalls();

 obj.latestFloorNo = obj.NoOfFloors;

end

Die Delete-Funktion:

function obj = deleteFloor(obj)

for i = 1:obj.latestFloorNo

delete(obj.floor(i));

end

end

Jetzt können wir diese Variante testen:

Testen!!!

Machen wir weiter mit dem Dach. Bevor die Wände gelöscht werden löschen wir zunächst das Dach, falls eines existiert mit if ~isempty …

if ~isempty(obj.roof)

delete(obj.roof);

end

Nachdem die Wände erzeugt wurden und wenn die Variable HasRoof aktiviert ist, rufen wir die Funktion „drawRoof“ auf.

if obj.HasRoof

obj = obj.drawRoof();

end

Für die Funktion „drawRoof“ benötigen wir noch eine private Variable, in der wir das Dach speichern.

Das Dach wird mit einer einfachen Linie erzeugt.

function obj = drawRoof(obj)

p1 = obj.startXY(1);

p2 = obj.startXY(1)+obj.Width/6;

p3 = obj.startXY(1)+obj.Width-obj.Width/6;

p4 = obj.startXY(1)+obj.Width;

xValues = [p1 p2 p3 p4];

p1 = obj.startXY(2)+(obj.NoOfFloors \* obj.floorHeight);

p2 = obj.startXY(2)+obj.floorHeight+(obj.NoOfFloors \* obj.floorHeight);

p3 = obj.startXY(2)+obj.floorHeight+(obj.NoOfFloors \* obj.floorHeight);

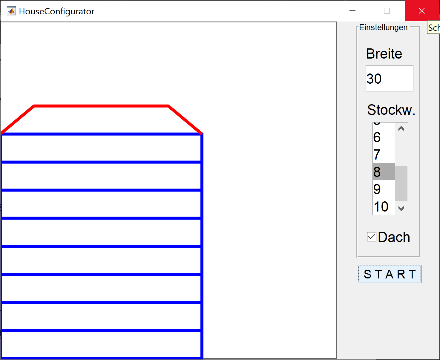
p4 = obj.startXY(2)+(obj.NoOfFloors \* obj.floorHeight);

yValues = [p1 p2 p3 p4];

obj.roof = line(xValues, yValues, 'Color', obj.roofColor, 'LineWidth', obj.lineWidth);

end

Testen!!!

Jetzt noch die Tür und die Fenster. Sie werden je Stockwerk gezeichnet, sollten deshalb in der Schleife der Wände aufgerufen werden und die aktuelle Ebene wird übergeben:

for i = 1:obj.NoOfFloors

lowerLeftXY = [obj.startXY(1) obj.startXY(2) + obj.floorHeight \* (i-1)];

deltaXY = [obj.Width obj.floorHeight];

obj.floor(i) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.wallColor, 'LineWidth', obj.lineWidth);

**obj = obj.drawOpenings(i);**

end

Die Funktion „drawOpenings“ bekommt den Übergabe Parameter des Stockwerks und es kann beginnen.

function obj = drawOpenings(obj, floorNo)

end

Zunächst benötigen wir einige private Variablen für die Größe und Anordnung der Fester: windowWidth = 1.5, windowHeight=1, windowSpacing=1.5, sillHeight=1 (Fensterbankhöhe), windowColor = ‚b‘ sowie eine Variable zur Speicherung der Fenster/Tür „windows“ und die letzte Anzahl von Fenster „latestWindowNo=0“.

In der Funktion „drawOpenings“ berechnen wir als erstes die Anzahl von Fenster (aus der Gesamtbreite geteilt durch Fensterbreite + Fensterabstand), die wir in der verfügbaren Breite des Hauses unterbringen können und speichern sie in der Variable „latestWindowNo“:

obj.latestWindowNo = fix(obj.Width / (obj.windowWidth + obj.WindowSpacing));

Da wir die rechten und linken Fenster in der Schleife gleichzeitig erzeugen können, braucht die Schleife nur bis zur Hälfte der Anzahl der Fenster gehen. Dafür benötigen wir aber einen Counter, damit der Index der Fenster fortlaufend ist.

counter = 1;

for i = 1:fix(obj.latestWindowNo/2)

…

counter = counter + 1;

End

Fügen wir jetzt die beiden Rechtecke in die Schleife ein und speichern sie in der privaten Variable „windows“ und vergessen den zweiten Counter nicht:

for i = 1:fix(obj.latestWindowsNo/2)

lowerLeftXY = [xValue yValue];

deltaXY = [obj.windowWidth obj.windowHeight];

obj.windows(floorNo, counter) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.windowColor, 'LineWidth', 1);

counter = counter + 1;

lowerLeftXY = [xValue yValue];

deltaXY = [obj.windowWidth obj.windowHeight];

obj.windows(floorNo, counter) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.windowColor, 'LineWidth', 1);

counter = counter + 1;

end

Weil die Berechnung der linken unteren Ecke etwas aufwendiger ist, können wir die X und Y-Werte zur besserem Lesbarkeit jeweils in eine zusätzliche Variable eintragen.

Ausgehend von der linken unteren Ecke des Hauses starten wir mit dem X-Wert in der Mitte, d.h. X-Startpunkt + halbe Breite. Für die linken Fenster müssen wir davon abziehen:

eine halbe Fensterbreite, der Abstand zwischen den Fenstern und noch eine Fensterbreite multipliziert mit Zähler der Schleife:

xValue = obj.startXY(1) + obj.Width/2 - obj.windowWidth/2 - (obj.windowSpacing + obj.windowWidth) \* i;

yValue = obj.startXY(2) + obj.sillHeight + obj.floorHeight \* (floorNo-1);

Der Y-Wert ergibt sich aus dem Y-Wert des Startpunktes und der Fensterbankhöhe und dazu addiert wird die Stockwerkhöhe multipliziert mit der aktuellen Stockwerks-Nr., jedoch nicht für das erste Stockwerk, also -1.

yValue = obj.startXY(2) + obj.sillHeight + obj.floorHeight \* (floorNo-1);

Für den X-Wert der rechten Fenster müssen wir zur Mitte addieren:

eine halbe Fensterbreite, den Abstand zwischen den Fenstern und zusätzlich abhängig vom Zähler der Schleife noch eine Fensterbreite plus Abstand zwischen den Fenstern.

xValue = obj.startXY(1) + obj.Width/2 + obj.windowWidth/2 + obj.windowSpacing + (obj.windowSpacing + obj.windowWidth) \* (i-1);

Tür/ mittleres Fenster hinzufügen:

if floorNo == 1

lowerLeftXY = [obj.startXY(1) + obj.Width/2 - obj.windowWidth/2 obj.startXY(2)];

deltaXY = [obj.windowWidth obj.windowHeight + obj.sillHeight];

obj.windows(floorNo, counter) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.windowColor, 'LineWidth', 1);

else

lowerLeftXY = [obj.startXY(1) + obj.Width/2 - obj.windowWidth/2 obj.startXY(2) + obj.sillHeight + obj.floorHeight \* (floorNo-1)];

deltaXY = [obj.windowWidth obj.windowHeight];

obj.windows(floorNo, counter) = rectangle('Position',[lowerLeftXY deltaXY], 'EdgeColor', obj.windowColor, 'LineWidth', 1);

end

Ein Bild, das Tisch enthält.

Automatisch generierte Beschreibungclassdef clsHouse

properties (Access = private)

centerX

UIAxes

previousLevels

previousRoof = false

house

roof

roofColor = 'r'

faceColor = 'w'

edgeColor = 'k'

lineWidth = 3

levelHeight = 10

windowSize = 5

end

properties (Access = public)

Width

Levels

HasRoof

end

methods (Access = public)

% constructor

function app = clsHouse(uIAxes, centerX)

app.UIAxes = uIAxes;

app.centerX = centerX;

end

function app = Draw(app)

if ~isempty(app.house)

app = app.deleteHouse();

end

for i=1:app.Levels

app = app.drawLevel(i);

end

if (app.HasRoof)

app = app.drawRoof();

end

app.previousLevels = app.Levels;

app.previousRoof = app.HasRoof;

end

end

methods (Access = private)

function app = drawLevel(app, i)

lowerLeftXY = [app.centerX - app.Width/2 app.levelHeight \* (i-1)];

deltaXY = [app.Width app.levelHeight];

app.house(i) = rectangle( app.UIAxes, ...

'Position',[lowerLeftXY deltaXY], ...

'FaceColor',app.faceColor, ...

'EdgeColor',app.edgeColor, ...

'LineWidth',app.lineWidth);

end

function app = drawRoof(app)

xValues = [app.centerX - app.Width/2 app.centerX - app.Width/2 + app.levelHeight app.centerX + app.Width/2 - app.levelHeight app.centerX + app.Width/2];

yValues = [app.Levels \* app.levelHeight app.Levels \* app.levelHeight + app.levelHeight app.Levels \* app.levelHeight + app.levelHeight app.Levels \* app.levelHeight];

app.roof = line(app.UIAxes, xValues, yValues, 'Color',app.roofColor, 'LineWidth',app.lineWidth);

end

function app = deleteHouse(app)

for i=app.previousLevels:-1:1

delete(app.house(i))

end

if (app.previousRoof)

delete(app.roof);

end

end

function app = drawWindows(app)

end

end

end

UI

properties (Access = private)

House % Description

end

function startupFcn(app)

app.UIAxes.YAxis.Visible = 'off'; % remove y-axis

app.UIAxes.XAxis.Visible = 'off'; % remove x-axis

lowerLeftXY = [0 0];

deltaXY = [100 100];

rectangle( app.UIAxes, ...

'Position',[lowerLeftXY deltaXY], ...

'EdgeColor',[.4 .4 .4], ...

'LineWidth',1);

centerX = deltaXY(1)/2;

app.House = clsHouse(app.UIAxes, centerX);

end

function STARTButtonPushed(app, event)

app.House.Width = app.WidthEditField.Value;

app.House.Levels = str2double(app.LevelListBox.Value);

app.House.HasRoof = app.RoofCheckBox.Value;

app.House = app.House.Draw();

end

# Aufgabe Portal-Konfigurator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

# Aufgabe Pontonbrücken-Konfigurator

**Spezifikation**: Mit Hilfe einer Benutzeroberfläche soll ein Konfigurator für eine Pontonbrücke erzeugt werden.

Für die Anzahl der Pontons kann zwischen 2, 3, 4, 5 und 6 gewählt werden. Dadurch verändert sich der Durchmesser des Pontons, bei 2-24, 3-20, 4-16, 5-12 und 6-8 Einheiten.

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte BeschreibungMit Hilfe einer Check-Box kann zwischen einer Farb- und einer Schwarz/Weiss- Darstellung umgeschaltet werden.

**App-Name**: BridgeConfigurator

**Name der Klasse**: clsBridge

1. Erzeugen Sie die Benutzeroberfläche mit den erforderlichen Steuerelementen und einer Zeichenfläche mit Hilfe der Designer-App.

2a. Richten Sie die notwendigen öffentlichen Variablen ein.

2b. Richten Sie die Startup-Funktion ein, schalten Sie die Achsen aus, erzeugen Sie ein Rechteck zur Fixierung der Zeichenoberfläche und initialisieren Sie die noch zu erzeugende Klasse.

Ein Bild, das Text, Screenshot, Display, Software enthält.

Automatisch generierte Beschreibung2c. Richten Sie die notwendigen Callback-Funktionen ein, leiten Sie die erforderlichen UI-Informationen an die noch zu erzeugende Klasse und starten Sie die öffentliche Funktion der Klasse.

3. Richten Sie die Klasse *clsBridge* mit einem Konstruktor, den notwendigen öffentlichen Variablen und den leeren Hüllen der geplanten öffentlichen und privaten Methoden, ein.

6. Füllen Sie in der Klasse die Funktion, die die Pontons zeichnet und stellen Sie sicher, dass die Pontons nach Veränderung der Ponton-Anzahl gelöscht werden. Achten Sie darauf, dass sich bei Änderung der Anzahl auch der Durchmesser ändert! Testen Sie dann das Programm!

7. Füllen Sie in der Klasse die Funktion, die die Plattform auf den Pontons zeichnet und stellen Sie sicher, dass die Plattform nach Veränderung der Ponton-Anzahl gelöscht wird.

8. Füllen Sie in der Klasse die Funktion, die die Treppen zeichnet und stellen Sie sicher, dass die Treppen nach Veränderung der Ponton-Anzahl gelöscht werden. Hinweis: beginnen Sie zunächst nur mit einer Treppe und testen Sie das Programm!

9 (Bonus). Schalten Sie die Farb-Darstellung ein falls die Farb-Checkbox aktiviert wurde, und schalten Sie zurück auf Schwarz/Weis Darstellung bei nicht aktivierter Checkbox.

|  |  |  |
| --- | --- | --- |
| **Zweck** | **private Variable** | **Vorgabewert** |
| Zeichenfläche | UIAxes |  |
| Größe der Zeichenfläche | maxXY |  |
| Die Pontons | pontoon |  |
| Ponton Flächenfarbe | pontoonFaceColor | ‚y‘ |
| Ponton Kantenfarbe | pontoonEdgeColor | ‚r‘ |
| Ponton Durchmesser | pontoonDiameter |  |
| Die Plattform | platform |  |
| Flächenfarbe Plattform, Treppen | faceColor | ‚c‘ |
| Kantenfarbe Plattform, Treppen | edgeColor | ‚b‘ |
| Linienstärke | lineWidth | 3 |
| Treppe links | stairsLeft |  |
| Treppe rechts | stairsRight |  |
| Stufen Breite/Höhe | stepSize | 4 |
| Vorherige Anzahl Pontons | lastCount | 2 |
| Vorheriger Durchmesser Pontons | lastDiameter | 24 |

**Lösungsvorschlag**

classdef clsBridge

properties (Access = private)

UIAxes

maxXY

pontoon

pontoonFaceColor = 'y'

pontoonEdgeColor = 'r'

pontoonDiameter

platform

faceColor = 'c'

edgeColor = 'b'

lineWidth = 3

stairsLeft

stairsRight

stepSize = 4

lastCount = 2

lastDiameter = 24

end

properties (Access = public)

PontoonCount = 2

IsColor = false;

end

methods (Access = public)

% constructor

function app = clsBridge(uIAxes, maxXY)

app.UIAxes = uIAxes;

app.maxXY = maxXY;

app = app.draw();

end

function app = Start(app)

app = app.delete();

app = app.draw();

app.lastCount = app.PontoonCount;

app.lastDiameter = app.pontoonDiameter;

end

end

methods (Access = private)

function app = draw(app)

app = app.setPontoonDiamenter();

app = app.setColor();

app = app.drawPontoon();

app = app.drawPlatform();

app = app.drawStairs();

end

function app = drawPontoon(app)

deltaXY = [app.pontoonDiameter, app.pontoonDiameter];

x = app.maxXY(1)/2-app.PontoonCount \* app.pontoonDiameter/2;

y = 0;

for i=1:app.PontoonCount

lowerLeftXY = [x y];

app.pontoon(i) = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor', app.pontoonFaceColor, ...

'EdgeColor', app.pontoonEdgeColor, ...

'LineWidth', app.lineWidth, ...

'Curvature', [1 1]);

x = x+app.pontoonDiameter;

end

end

function app = drawStairs(app)

stepNo = app.pontoonDiameter/app.stepSize;

deltaXY = [app.stepSize app.stepSize];

x = (app.maxXY(1)/2-app.PontoonCount \* app.pontoonDiameter/2)-app.stepSize \* stepNo;

y = 0;

for i=1:stepNo

lowerLeftXY = [x y];

app.stairsLeft(i) = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor', app.faceColor, ...

'EdgeColor', app.edgeColor, ...

'LineWidth', app.lineWidth);

x = x+app.stepSize;

y = y+app.stepSize;

end

x = (app.maxXY(1)/2+app.PontoonCount \* app.pontoonDiameter/2)+app.stepSize \* (stepNo-1);

y = 0;

for i=1:stepNo

lowerLeftXY = [x y];

app.stairsRight(i) = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor', app.faceColor, ...

'EdgeColor', app.edgeColor, ...

'LineWidth', app.lineWidth);

x = x-app.stepSize;

y = y+app.stepSize;

end

end

function app = drawPlatform(app)

w = app.pontoonDiameter\*app.PontoonCount;

x = app.maxXY(1)/2-app.PontoonCount \* app.pontoonDiameter/2;

y = app.pontoonDiameter;

lowerLeftXY = [x y];

deltaXY = [w app.stepSize];

app.platform = rectangle( ...

app.UIAxes, ...

'Position', ...

[lowerLeftXY deltaXY], ...

'FaceColor', app.faceColor, ...

'EdgeColor', app.edgeColor, ...

'LineWidth', app.lineWidth, ...

'Curvature', [0 0]);

end

function app = delete(app)

if ~isempty(app.pontoon)

for i = app.lastCount:-1:1

delete(app.pontoon(i));

end

end

if ~isempty(app.platform)

delete(app.platform);

end

if ~isempty(app.stairsLeft)

stepNo = app.lastDiameter/app.stepSize;

for i=stepNo:-1:1

delete(app.stairsLeft(i));

delete(app.stairsRight(i));

end

end

end

function app = setPontoonDiamenter(app)

switch app.PontoonCount

case 2

app.pontoonDiameter = 24;

case 3

app.pontoonDiameter = 20;

case 4

app.pontoonDiameter = 16;

case 5

app.pontoonDiameter = 12;

case 6

app.pontoonDiameter = 8;

end

end

function app = setColor(app)

if app.IsColor

app.pontoonFaceColor = 'y';

app.pontoonEdgeColor = 'r';

app.faceColor = 'c';

app.edgeColor = 'b';

else

app.pontoonFaceColor = 'w';

app.pontoonEdgeColor = 'k';

app.faceColor = 'w';

app.edgeColor = 'k';

end

end

end

end

# Aufgabe: Draw-Bridge

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

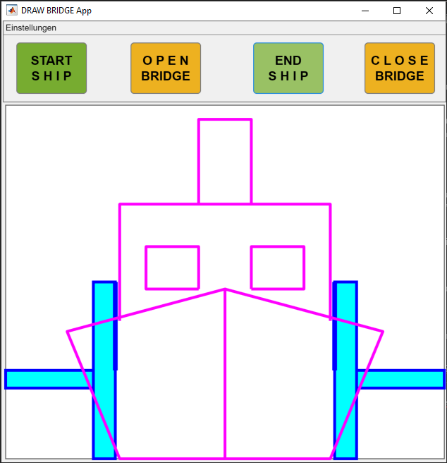
**Lösung**: Lore Ipsum tralala

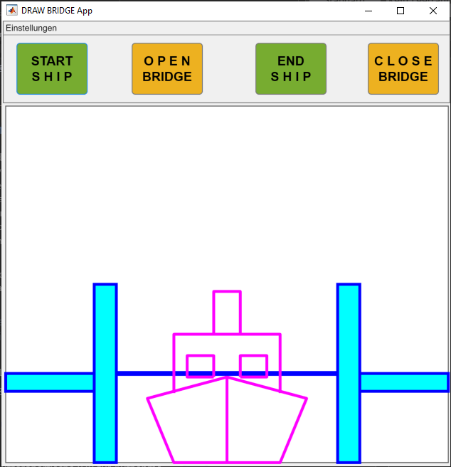
Es soll ein Programm mit dem Namen „DrawBridge“ entworfen werden, das beim Starten eine geschlossene Zugbrücke erzeugt.

Die Schaltfläche „START SHIP“ zeichnet das vorgegebene Schiff und vergrößert es in kleinen Schritten.

Mit der Schaltfläche „OPEN BRIDGE“ lässt sich die Zugbrücke bis zu einem 90 Grad-Winkel öffnen.

Die Schaltfläche „END SHIP“ vergrößert das vorgegebene Schiff weiter in kleinen Schritten und löscht es anschließend vollständig.

Ein Bild, das Text, Screenshot, Parkplatz enthält.

Automatisch generierte BeschreibungMit der Schaltfläche „CLOSE BRIDGE“ wird die Zugbrücke zurück in den Ursprungzustand geschlossen.

function app = drawShip(app, i)

%bow left

xy = [app.centerX, 0];

xValues = [xy(1) xy(1)-2\*i xy(1)-3\*i xy(1) xy(1)];

yValues = [xy(2) xy(2) xy(2)+3\*i xy(2)+4\*i xy(2)];

app.ship(1) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

%bow right

xValues = [xy(1) xy(1)+2\*i xy(1)+3\*i xy(1) xy(1)];

app.ship(2) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

%superstructures

xValues = [xy(1)-2\*i xy(1)-2\*i xy(1)+2\*i xy(1)+2\*i];

yValues = [xy(2)+3.25\*i xy(2)+6\*i xy(2)+6\*i xy(2)+3.25\*i];

app.ship(3) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

%chimney

xValues = [xy(1)-i/2 xy(1)-i/2 xy(1)+i/2 xy(1)+i/2];

yValues = [xy(2)+6\*i xy(2)+8\*i xy(2)+8\*i xy(2)+6\*i];

app.ship(4) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

%window left

xValues = [xy(1)-.5\*i xy(1)-1.5\*i xy(1)-1.5\*i xy(1)-.5\*i xy(1)-.5\*i];

yValues = [xy(2)+4\*i xy(2)+4\*i xy(2)+5\*i xy(2)+5\*i xy(2)+4\*i];

app.ship(5) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

%window right

xValues = [xy(1)+.5\*i xy(1)+1.5\*i xy(1)+1.5\*i xy(1)+.5\*i xy(1)+.5\*i];

yValues = [xy(2)+4\*i xy(2)+4\*i xy(2)+5\*i xy(2)+5\*i xy(2)+4\*i];

app.ship(6) = line(app.UIAxes, ...

xValues, yValues, ...

'Color', app.shipColor, ...

'LineWidth', app.lineWidth);

end

# Aufgabe: Slot-Machine

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „“. Es soll ein …

GUI-Name:

Funktionen:

## Lösungsansatz

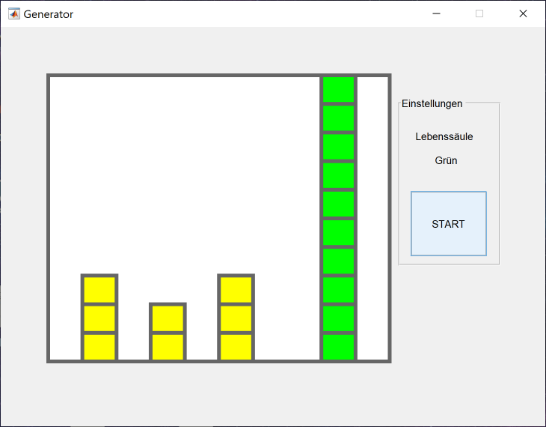
## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der Matlab-Aufgabe „“

# Aufgabe: GamblingGenerator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Es soll ein Gambling Generator mit einer optischen Anzeige erzeugt werden. Die Farbe der Kreditsäule soll dabei einstellbar sein. Nach Betätigung des Start-Buttons sollen sich die 3 Gambling-Säulen für eine kurze Zeit auf Basis von Zufallszahlen bewegen und der Reihe nach stehen bleiben. Bei 3 gleichhohen Säulen wir die Kreditsäule um 3 erhöht, bei 2 gleichen Säulen um 1. Sonst führt jeder Durchlauf zum Abzug von 1.

GUI-Name: GamblingGenerator

Klasse: clsGambler

Funktionen:

## Lösungsansatz

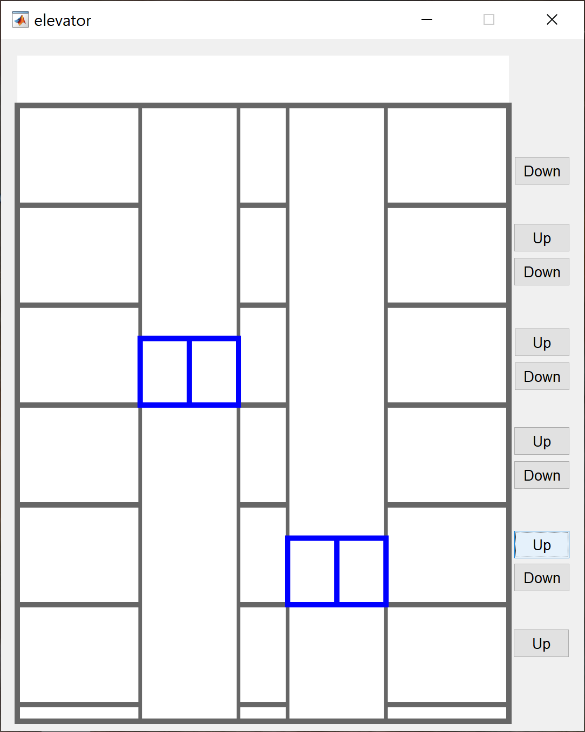
## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der Matlab-Aufgabe „“

# Aufgabe: Elevator

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „“. Es soll ein …

GUI-Name:

Funktionen:

## Lösungsansatz

## Lösung Matlab Aufgabe:

Willkommen bei der Lösung der Matlab-Aufgabe „“

# Aufgabe Labyrinth (Maze)

**Spezifiaktion**: Mit Hilfe einer Benutzeroberfläche soll

**Lösung**: Lore Ipsum tralala

Willkommen bei der Matlab Aufgabe „“. Es soll ein …

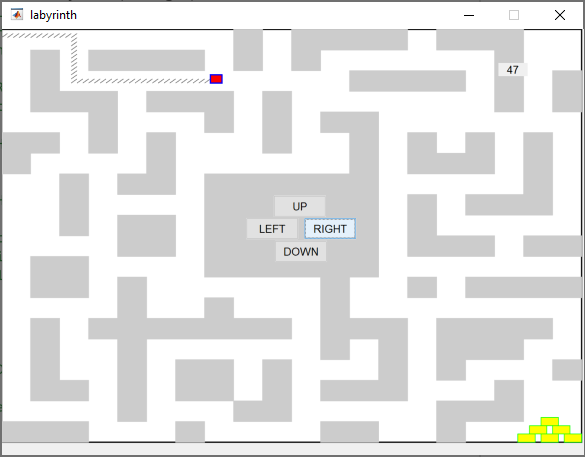
GUI-Name:

Funktionen:

## Lösungsansatz

## Lösung Aufgabe Schatzsuche:

Willkommen bei der Lösung der Aufgabe „Schatzsuche“



# Info

Studifix

https://studyflix.de/suche?query=Python