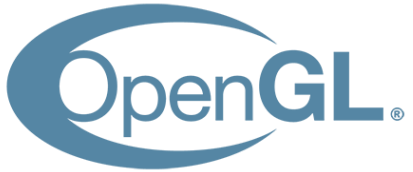
Computergrafik IV

Kantonsschule Sursee
Dr. Philipp Hurni

Agenda

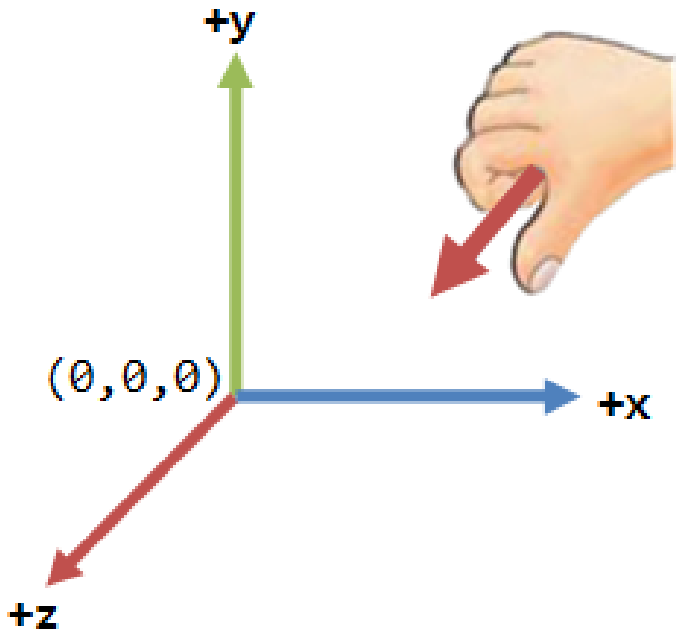
- OpenGL mit Python
- Projektionen
- Übungen:
  - der Würfel in der 3D Welt
  - Würfel generieren
  - Würfel herumfliegen lassen
- Prüfungsanforderungen
- Übungsaufgaben

OpenGL mit Python Pygame

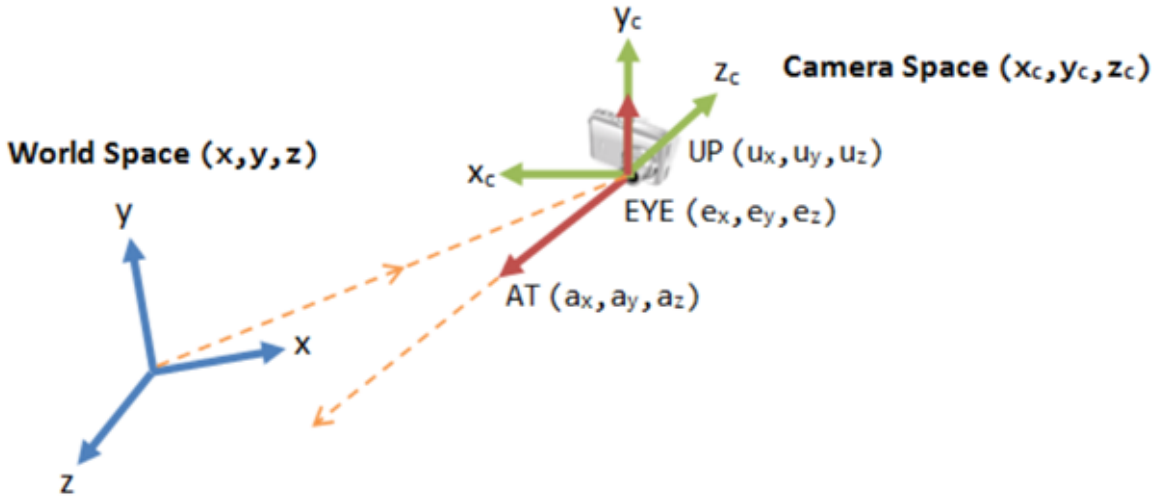


- OpenGL (Open Graphics Library) ist eine plattform- und sprachunabhängige API (Application Programming Interface) zur Darstellung von 2D- und 3D-Vektorgrafiken.
- Pygame ist eine Reihe von Python-Modulen, die für interaktive Spieleentwicklung entwickelt wurden. Es bietet einfache Funktionen für Fensterverwaltung, Ereignisbehandlung (Tastatur, Maus), Timer und grundlegendes 2D-Rendering (Sprites, Oberflächen).
- Pygame wird verwendet, um das Zeichenfenster zu initialisieren und zu verwalten (pygame.display.set_mode). Beim Initialisieren wird innerhalb von Pygame OpenGL die Kontrolle übergeben.

X,Y,Z in der Computergrafik

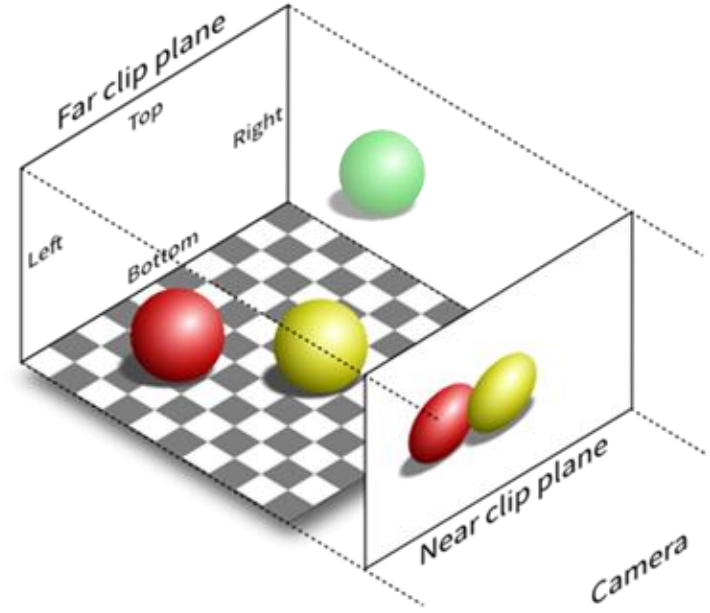


Right-Hand System (RHS) or Counter-Clockwise (CCW) System

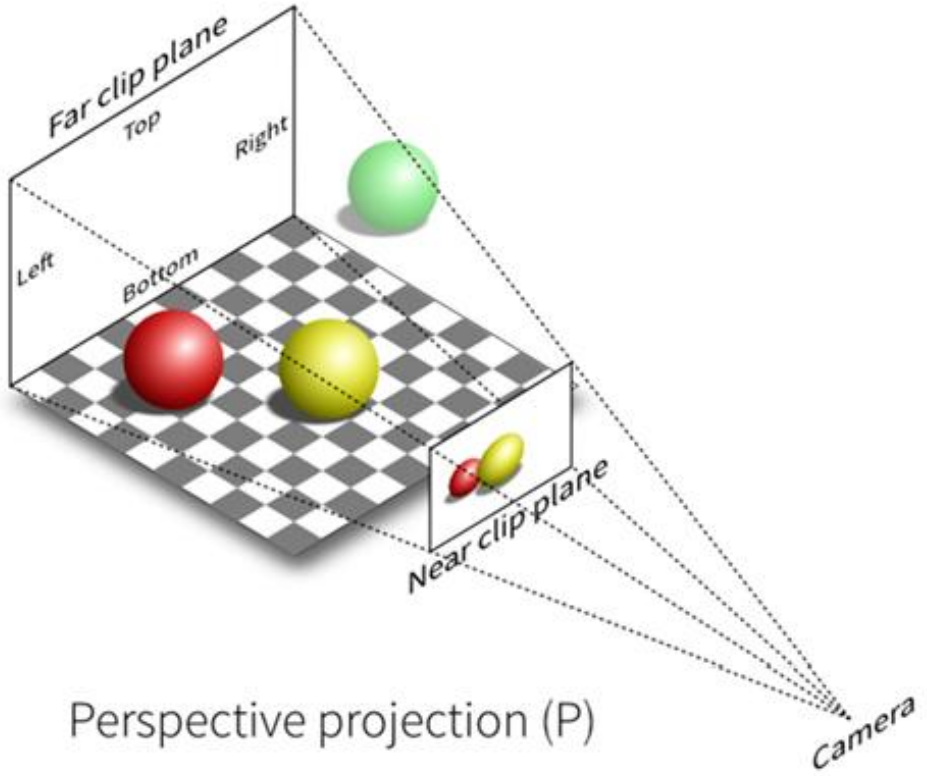


In OpenGL gibt es das Konzept einer "virtuellen Kamera", aus welcher die Szene aufgenommen wird

Arten der Projektion



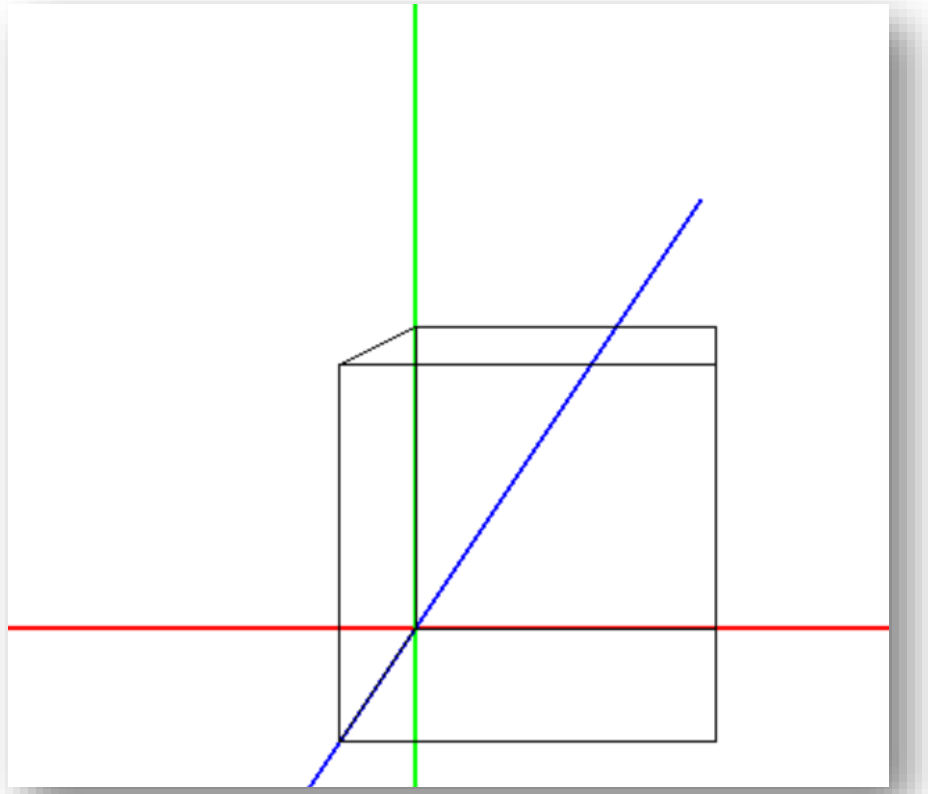
Orthographic projection (O)



Perspective projection (P)

Unsere Welt mit dem Würfel

- Öffne "Wuerfel – einzeln.py"
- Koordinatensystem (x,y,z)
- Nullpunkt am Schneidepunkt der Linien
- Kamera ist an Position
  - x=100
  - y=150
  - z=500
- Würfel ist an Position (hintere linke untere Ecke)
  - x = 0
  - y = 0
  - z = 0



Wuerfel - einzeln.py

Unsere Welt mit dem Würfel

Funktionsdefinition

```
119 def generiereWuerfel(wuerfel_startpunkt_x, wuerfel_startpunkt_y, wuerfel_startpunkt_z, seitenlaenge):
120
121     SEITE = seitenlaenge
122
123     linien = [
124         # Vorderes Quadrat
125         [[0, 0, 0], [SEITE, 0, 0]],
126         [[SEITE, 0, 0], [SEITE, SEITE, 0]],
127         [[SEITE, SEITE, 0], [0, SEITE, 0]],
128         [[0, SEITE, 0], [0, 0, 0]],
129
130         # Hinteres Quadrat
131         [[0, 0, SEITE], [SEITE, 0, SEITE]],
132         [[SEITE, 0, SEITE], [SEITE, SEITE, SEITE]],
133         [[SEITE, SEITE, SEITE], [0, SEITE, SEITE]],
134         [[0, SEITE, SEITE], [0, 0, SEITE]],
135
136         # Verknüpfung der Quadrate
137         [[0, 0, 0], [0, 0, SEITE]],
138         [[SEITE, 0, 0], [SEITE, 0, SEITE]],
139         [[SEITE, SEITE, 0], [SEITE, SEITE, SEITE]],
140         [[0, SEITE, 0], [0, SEITE, SEITE]],
141     ]
```

Aufruf

```
61         # hier generieren wir den Würfel - am Nullpunkt
62         wuerfellinien = generiereWuerfel(0, 0, 0, 100)
63
64         # Zeichne den Würfel
65         glBegin(GL_LINES)
66         for linie in wuerfellinien:
67             for koordinaten in linie:
68                 glVertex3fv(koordinaten)
69         glEnd()
```

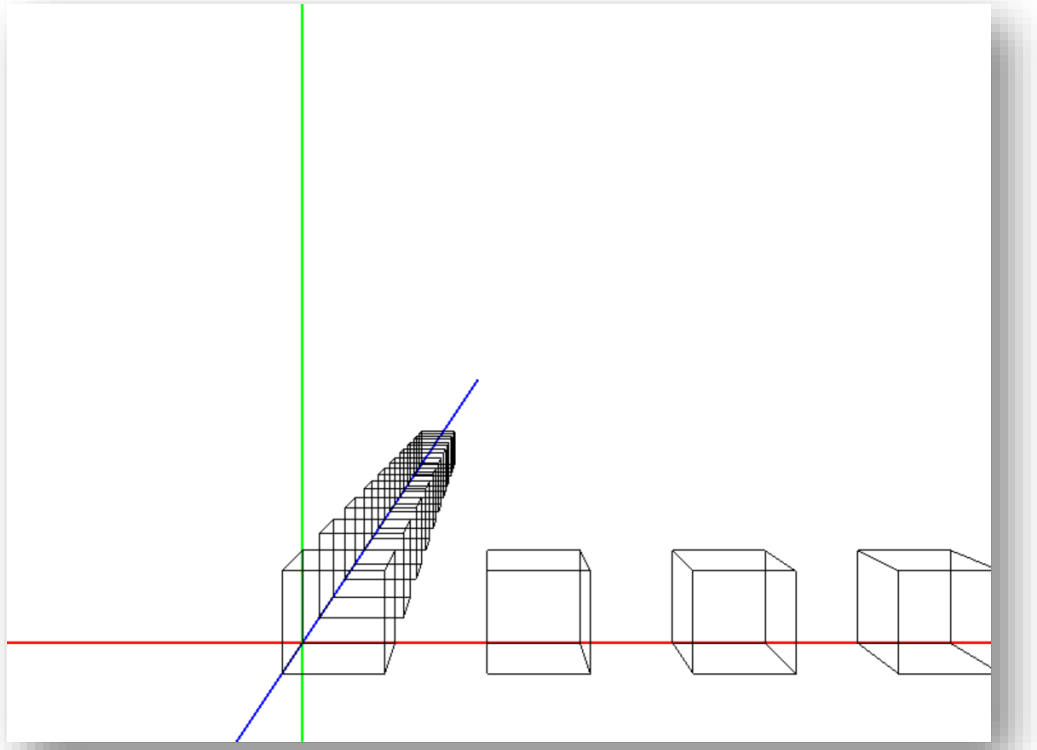
Übung Generierung von Objekten

Schreibe zwei Schlaufen

- Eine Schlaufe generiert Würfel in die positive x-Achse
- Eine Schlaufe generiert Würfel in die negative z-Achse

Hinweis: einfach das hier mit einer Schlaufe mehrfach machen

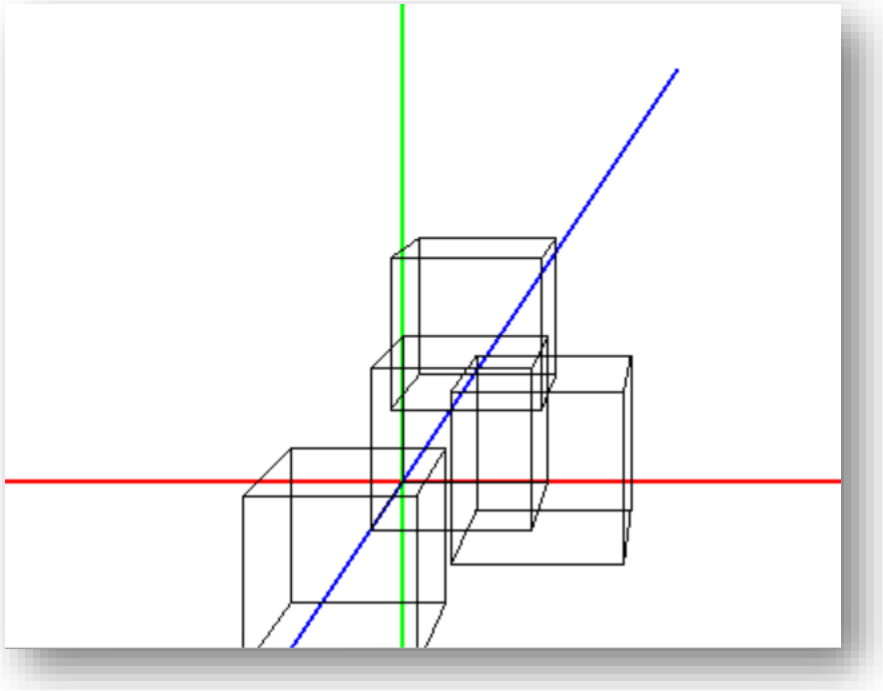
```
61          # hier generieren wir den Würfel - am Nullpunkt
62          wuerfellinien = generiereWuerfel(0, 0, 0, 100)
63
64          # Zeichne den Würfel
65          glBegin(GL_LINES)
66          for linie in wuerfellinien:
67              for koordinaten in linie:
68                  glVertex3fv(koordinaten)
69          glEnd()
```



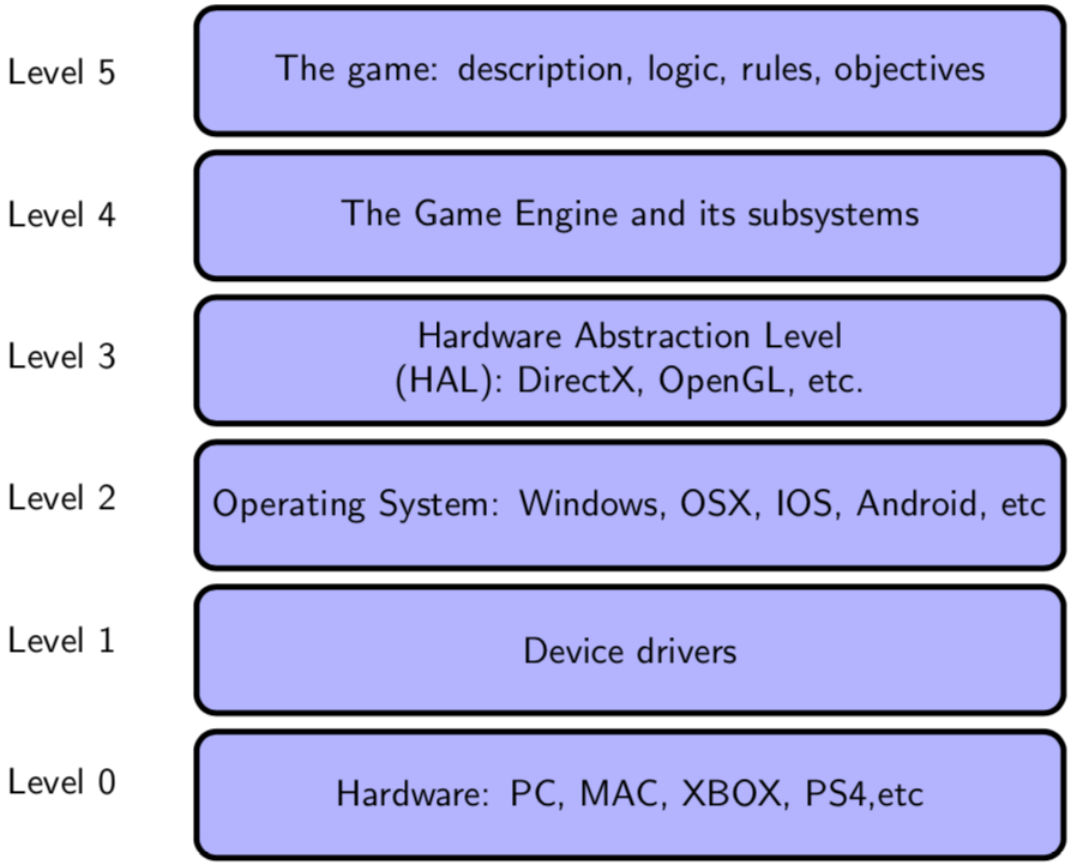
Übung Herumfliegende Würfel

Versuche den Code nachzuvollziehen

Mach Änderungen, probier aus was passiert



Schichtenmodell bei Grafik-Anwendungen



Prüfungsanforderungen Algorithmen

- Eigenschaften von Algorithmen
  - Verstehen
  - Anwenden können
- Algorithmen-Komplexität
  - Komplexitätsklassen
  - Probleme in die Komplexitätsklassen einordnen können (Komplexitätsanalyse)
- Code lesen und interpretieren
  - siehe Übungen
- Algorithmische Lösungen
  - Formulieren können
  - Umsetzen können mit Python
  - siehe Übungen

Prüfungsanforderungen Computergrafik

- Bilder
  - Pixelgrafik vs. Vektorgrafik
  - Farbmodell RGB
- Dateiformate
  - Vektorgrafik vs. Bitmap
  - Konzept verlustlose vs. verlustbehaftete Kompression
  - Formate SVG vs BMP
- Grafikalgorithmen
  - Floodfill mittels Rekursion
  - Bresenham-Algorithmus
- Basistransformationen der Computergrafik
  - Rotation
  - Translation
  - Skalierung
  - Scherung
- Kavaliers-Projektion
  - Grundprinzip / Anwendung

- Kein Data Science
- Kein Programmieren bei Computergrafik

Total maximal 25% Anteil "selbst programmieren"

Unterlagen auf Exam.net vorhanden
Notizen mitnehmen erlaubt

Übungsaufgaben frühere Prüfung

3. Floodfill Rekursion – Untenstehend eine in schwarze Blöcke eingerahmte Tetris-Figur. Du hast nun den Flood Fill Algorithmus und führst ihn auf das Pixel (1,1) aus. Du ersetzt die alte Farbe Weiss mit der neuen Farbe Rot.

[4 Punkte]


```
def floodfill(x, y, alteFarbe, neueFarbe):
  print("floodfill(", x,",",y,",alteFarbe, neueFarbe))")
  if getpixelcolor(x, y) == alteFarbe:
      drawpixel(x, y, neueFarbe)
      floodfill(x, y + 1, alteFarbe, neueFarbe)      # oben
      floodfill(x, y - 1, alteFarbe, neueFarbe)      # unten
      floodfill(x + 1, y, alteFarbe, neueFarbe)      # rechts
      floodfill(x - 1, y, alteFarbe, neueFarbe)      # links
  else:
    print("stop floodfill(",x,",",y,")")
    # anhalten!
```

Übungsaufgaben frühere Prüfung


```
def floodfill(x, y, alteFarbe, neueFarbe):
    print("floodfill(", x,",",y,",",alteFarbe, neueFarbe))")
    if getpixelcolor(x, y) == alteFarbe:
        drawpixel(x, y, neueFarbe)
        floodfill(x, y + 1, alteFarbe, neueFarbe)      # oben
        floodfill(x, y - 1, alteFarbe, neueFarbe)      # unten
        floodfill(x + 1, y, alteFarbe, neueFarbe)      # rechts
        floodfill(x - 1, y, alteFarbe, neueFarbe)      # links
    else:
        print("stop floodfill(",x,",",y,")")
        # anhalten!
```

- Floodfill(1,1) – zeichnet (1,1)!
  - Floodfill(1,2) # oben von (1,1) - bricht ab
  - Floodfill(1,0) # unten von (1,1) – bricht ab
  - Floodfill(2,1) # rechts von (1,1) – zeichnet (2,1)!
    - Floodfill(2,2) #oben von (2,1) – bricht ab
    - Floodfill(2,0) # unten von (2,1) -bricht ab
    - Floodfill(3,1) #rechts von (2,1) – zeichnet (3,1)!
      - Floodfill(3,2) #oben von (3,1) – bricht ab
      - Floodfill(3,0) #unten von (3,1) – bricht ab
      - Floodfill(4,1) #rechts von (3,1) – zeichnet (4,1)!
        - Floodfill(4,2) #oben von (4,1)- bricht ab
        - Floodfill (4,0) #unten von (4,1) -bricht ab
        - Floodfill(5,1) #rechts von (4,1) -bricht ab
        - Floodfill (4,1) #links von (4,1) – bricht ab
      - Floodfill(2,1) #links von (3,1) – bricht ab
    - Floodfill(1,1) #links von (2,1) – bricht ab
  - Floodfill(0,1) #links von (1,1) – bricht ab

Übungsaufgaben frühere Prüfung

```
zahlenliste = [20, 10, 35, 5]
minimum = min(zahlenliste) # sucht die kleinste Zahl in der Liste

valid = True
for x in zahlenliste:
    if (x % minimum) != 0:
        valid = False

if valid == False:
    print("Liste erfüllt die Bedingung nicht")

else:
    print("Liste erfüllt die Bedingung")
```