Mosaik Generator Projekt

Betreuer: Prog. Dr. Christof Rezk-Salama
Bearbeiter: Mohamad Sahyouni
Matrikel Nr.:974101

Abbildung 1: Mosaik Kunst.

❖ Einleitung:

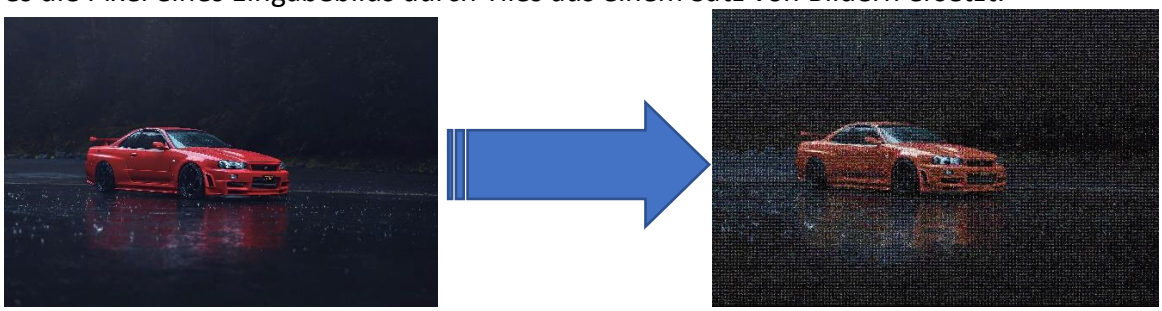
Mosaik ist einer der alten Künste, die seit der Antike existieren. Es basiert auf dem Mechanismus der Dekoration durch sehr viele kleine Stücke, die miteinander gestapelt werden, um ein Gemälde zu bilden. Künstler können verschiedene Materialien in Mosaiken verwenden, darunter Stein, Metall, Glas, usw. Diese kleinen Objekte, aus denen das Mosaikgemälde besteht, sind zufällige Stücke, die allein keinen Wert darstellen und möglicherweise keine konsistenten Dimensionen enthalten. Das Gemälde wird gezeichnet, indem die kleinen Stücke richtig zusammengesetzt werden. Mosaikkunst spielte eine wichtige Rolle in der westlichen Kunst, obwohl es überall auf der Welt zu sehen ist und es wurde als eine Art Dekoration in vielen alten Zivilisationen wie, islamischen, und byzantinischen Zivilisationen verwendet.

Um ein Mosaik zu bilden, muss zuerst die Oberfläche vorbereitet werden. D.h. Klebstoffe hinzufügen und dann Stücke aller Art darauf zu kleben. Diese Vorgehensweise wird in dem Bau von kleinen Mosaiken verwendet. In den großen Mosaiken legt der Künstler die Vorderseite der Steinstücke oder das Material, aus dem das Mosaik besteht, mit einem nicht starken Klebstoff auf Geschenkpapier und dann wird das Projekt zu dem Ort des Kunstwerks übertragen und folgend werden die Steine auf die Hauptoberfläche des Kunstwerks geklebt.

Abbildung 2: römische Mosaik.

❖ Mosaik Generator Programm:

Das Ziel dieses Projektes ist, Erstellen eines Programms, das ein Mosaikbild generiert, indem es die Pixel eines Eingabebilds durch Tiles aus einem Satz von Bildern ersetzt.



Eingabebild

Mosaikbild

Dieses Mosaik-Projekt umfasst vier Python-Dateien. Die Dateien "run.py" und "run1.py" sind für die Steuerung des Projekts verantwortlich und enthalten fast den gleichen Code. Der Hauptunterschied zwischen ihnen besteht darin, dass "run.py" nur von der Kommandozeile aus ausgeführt werden kann, während "run1.py" entweder von der Kommandozeile oder in einem Interpreter wie VScode oder PyCharm direkt ausgeführt werden kann.

```python
def main():
    input_img = input('input image path:')
    output_img = input('output image path:')
    tiles_dir = input('type tiles directory path:')
    xinput = input('insert width of tiles:')
    yinput  = input('insert height of tiles:')

    '''defualt inputs'''
    output='mosaic.jpg'
    tiles ='tiles'
    x=10
    y=10

    '''set giving input'''
    if str(input_img) != '':
        input2 = str(input_img)
    else:
        print('you have to to give at least the target image path...')
    if str(output_img) != '':
        output = str(output_img)
    if str(tiles_dir) != '':
        tiles = str(tiles_dir)
    if str(xinput) != '':
        x = int(str(xinput))
    if str(yinput) != '':
        y = int(str(yinput))

    if not os.path.isfile(input2):
        print('! ERROR input Image not found'.format(input2))
    elif not os.path.isdir(tiles):
        print('! ERROR tiles Dir not found'.format(tiles))
    else:
        print('Starting...')
        Mosaic = MosaicGenerator(input2, output, tiles, x, y)
        resizedTiles = Tiles.getresizedTiles(Mosaic.tiles + '/*', (x, y))
        inputImgResized = Mosaic.resize_input(Mosaic.input, x, y)
        pixelsMatches = Mosaic.matchingTiles(inputImgResized, resizedTiles)
        OutputImg = Image.new('RGB', Image.open(Mosaic.input).size)
        newImage = Mosaic.replace_pixels(inputImgResized.size[0],
                                         inputImgResized.size[1], OutputImg,
                                         x, y, resizedTiles, pixelsMatches)
        Mosaic.save_result(newImage, Mosaic.output)
        print('finished.')

if __name__ == '__main__':
    main()
```

Abbildung 4: run1.py

```python
def main():
    parser = argparse.ArgumentParser()
    parser.add_argument('-i', dest='input_img')
    parser.add_argument('-o', dest='output_img',default='mosaic.jpg')
    parser.add_argument('-t', dest='tiles_dir',default='tiles')
    parser.add_argument('-x', dest='x', default=10)
    parser.add_argument('-y', dest='y', default=10)
    args = parser.parse_args()

    input2 = str(args.input_img)
    output = str(args.output_img)
    tiles = str(args.tiles_dir)
    x = int(str(args.x))
    y = int(str(args.y))

    if len(sys.argv) > 1:
        if not os.path.isfile(input2):
            print('! ERROR input Image not found'.format(input2))
        elif not os.path.isdir(tiles):
            print('! ERROR tiles directory not found'.format(tiles))
        else:
            print('Starting...')
            Mosaic = MosaicGenerator(input2, output, tiles, x, y)
            resizedTiles = Tiles.getresizedTiles(Mosaic.tiles + '/*', (x, y))
            inputImgResized = Mosaic.resize_input(Mosaic.input, x, y)
            pixelsMatches = Mosaic.matchingTiles(inputImgResized, resizedTiles)
            OutputImg = Image.new('RGB', Image.open(Mosaic.input).size)
            newImage = Mosaic.replace_pixels(inputImgResized.size[0],
                                             inputImgResized.size[1], OutputImg,
                                             x, y, resizedTiles, pixelsMatches)
            Mosaic.save_result(newImage, Mosaic.output)
            print('finished.')
    else:
        print('! ERROR no inputs were detected')
        print('you have to to give at least the target image path...')
        print('for example: python run.py -i target.jpg ')


if __name__ == '__main__':

    main()
```

Abbildung 3: run.py

In run.py wird argparse Modul verwendet, um die Argumente aus der Kommandozeile zu lesen. Es gibt 5 Argument, die aus der Kommandozeile gelesen werden können. Die Argumente sind:

1. input_img „-i “: Target-image path.
2. output_img „-o “: Mosaikbild Name.
3. tiles_dir „-t “: Path des Tiles-verzeichnis.
4. x „-x “: Breite der Tiles in Pixel.
5. y „-y “: Höhe der Tiles in Pixel.

Run.py kann mit dem folgenden Befehl ausgeführt werden:

```
python run.py -i target.jpg -o out.jpg -t tiles -x 40 -y 40
```

Auf jeden Fall kann der Benutzer alle Argumente angeben oder stattdessen Default verwenden,

indem er die Eingabe ignoriert. Zum Beispiel hier werden die Breite und Höhe nicht angegeben, deshalb werden die Standardwerte „default“ für x und y verwendet:

```
python run.py -i target.jpg -o out.jpg -t tiles
```

Alle Argumente haben Default-Werte, außer input_img. Da dieses Argument keinen Default-Wert hat, ist es notwendig, dass der Nutzer einen Wert für input_img angibt, da dieser nicht ignoriert werden kann. Ein Beispiel dafür wäre, wenn nur -i angegeben wird, um dem Argument input_img einen Wert zu geben:

```
python run.py -i target.jpg
```

Die Argumente input_img, output_img und tiles_dir sind Pfade zum Target-image, Mosaikbild und Tiles-Verzeichnis, deshalb werden sie als String-Objekte in den Variablen input2, output und tiles gespeichert. Die Argumente x und y repräsentieren die Breite und Höhe der Tiles im Mosaikbild und werden als Integer-Objekte in den Variablen x und y gespeichert.

In Zeile 23 in run.py wird überprüft, ob Eingaben in der Kommandozeile angegeben wurden. Falls nicht, werden Zeilen 41, 42 und 43 ausgeführt.

Um das Programm durch run1.py zu steuern, kann der Nutzer entweder die Datei in einem Interpreter öffnen und durch Drücken auf den run Taste ausführen, oder durch die Kommandozeile mit dem Befehl „python run1.py“ ausführen.

Nachdem run1.py ausgeführt wurde, dann kann der Nutzer die Eingaben in der Konsole bzw. Kommandozeile angeben. Dies wird durch Verwendung der Funktion input in den Zeilen 7, 8, 9, 10 und 11 ermöglicht. In den Zeilen 14 bis 17 werden die variablen output, tiles, x und y definiert und die Default-Werte angegeben. Diese werden durch die in den Variablen input_img, output_img, tiles_dir, xinput und yinput gespeicherte Eingaben ersetzt.

Um einen Default-Wert zu ersetzen, wird die Variable, in der die Eingabe gespeichert wird, geprüft und falls die leer ist (enthalt nur ein leerer String), wird der Default-Wert nicht ersetzt.

Es fällt sofort auf, dass es keinen Default-Wert für die Variable input2 gibt. D.h. genauso wie in run.py der Pfad des Target-image muss angegeben werden.

Die Zeilen 33 bis 48 in run1.py und die Zeilen 24 bis 39 in run.py sind gleich. Daher werden die Zeilen nur einmal betrachtet.

In der Zeilen 33 und 34 aus run1.py wird geprüft, ob der Pfad des Target-image zu einer Datei führt, falls nicht, wird ein Error Nachricht gezeigt. In der nächsten Zeile wird der Pfad des Tiles-verzeichnis geprüft, ob er zu einem Verzeichnis führt.

Ein Objekt der Klasse MosaicGenerator wird in der Zeile 39 definiert und die Variablen input2, output, tiles, x und y als Parametern des Konstruktors angegeben.

Eine Variable resizedTiles wird in der nächsten Zeile definiert. Die Variable enthält ein Array, im Array sind alle Tiles aus dem Tiles-Verzeichnis und alle sind resized.

Die Zeile 41 enthält die Variable inputImgResized. In dieser Variable wird ein Image gespeichert, das Image ist die Ergebnisse der Methode resized_input.

Die Variable PixelsMatches in der Zeile 42 ist ein Array, in dem die passenden Tiles gespeichert sind.

In der Zeile 43 wird ein neues Image Objekt mit der Größe der Variable inputImgResized erzeugt und in der Variable OutputImg gespeichert.

Zunächst wird die Methode replace_pixels ausgeführt, die Methode wechselt die Teilbilder durch das passende Objekt aus dem resizedTiles Array. Das Resultat ist ein Image Objekt, das in der Variable newImage gespeichert wird.

Zuletzt wird das Bild aus der Variable newImage in dem Pfad aus der Variable output gespeichert und die Nachricht ‚finished.‘ angezeigt.

```
 7  class MosaicGenerator:
 8      def __init__(self, input, output, tiles, x, y):
 9          self.input = input
10          self.output = output
11          self.tiles = tiles
12          self.x = x
13          self.y = y
```

Abbildung 5: MosaicGen.py Konsturktor der Klasse MosaicGenerator.

In MosaicGen.py ist die Klasse MosaicGenerator definiert. Die Methode __init__ ist der Konstruktor dieser Klasse. Der Konstruktor erzeugt ein Objekt der Type MosaicGenerator. Das Objekt hat 5 Attribute „input, output, tiles, x und y “, in diesen Attributen werden die Werte aus den Argumenten des Konstruktors gespeichert.

```
80      @staticmethod
81      def resize_input(img, x, y):
82          print('resizing input image...')
83          toResizeImg = Image.open(img)
84          inputWidth = toResizeImg.size[0]
85          inputHeight = toResizeImg.size[1]
86          xx = 0
87          yy = 0
88          imgx = 0
89          imgy = 0
90          while (xx + x) <= inputWidth:
91              xx += x
92              imgx+=1
93          while (yy + y) <= inputHeight:
94              yy += y
95              imgy+=1
96          newImg = toResizeImg.resize((imgx, imgy))
97          return newImg
```

Abbildung 6: die Methode resize_input der Klasse MosaicGenerator.

In der Zeile 81 ist die Methode resize_input definiert. Diese Methode hat drei Argumenten img, x und y. die Argumente repräsentieren die Target-image, Breite und Höhe. Hier wird gerechnet wie viel Unterbilder die Größe (x, y) horizontal und vertikal passen in der Target-image.

Zum Beispiel: Target-image der Größe = (1920,1080), x=10 und y = 10, dann ist der neuen Größe gleich (192, 108), da 192 Tiles der Größe (10,10) horizontal und 108 vertikal passen.

In der ersten while-Schleife wird die horizontale Anzahl der Unterbilder gerechnet und in der zweiten die vertikale Anzahl. Dann wird die Größe der Target-image in der Zeile 96 geändert.

```
69      @staticmethod
70      def colorsSimilar(c1,c2):
71          x1 = c1[0]
72          y1 = c1[1]
73          z1 = c1[2]
74          x2 = c2[0]
75          y2 = c2[1]
76          z2 = c2[2]
77          v = math.sqrt(math.pow((x1 - x2),2) + math.pow((y1 - y2),2)
78                        +math.pow((z1 - z2),2))
79          return v
```

Abbildung 7: die Mehthode colorSimilar der Klasse MosaicGenerator.

Diese Methode nimmt zwei Argumente, die rgb Werte sind, und rechnet die Distanz dazwischen durch Verwendung der folgenden Rechnung:

C1= (R1, G1, B1), C2= (R2, G2, B2). Hinwies: sqrt = Quadratwurzel

D = sqrt ((R1-R2)^2 + (G1-G2)^2 + (B1-B2)^2)

Je kleiner D, desto besser die Übereinstimmung.

```python
@staticmethod
def matchingTiles(img, tiles):
    print('finding average colors...')
    avgColors = []
    for img2 in tiles:
        colors = np.asarray(img2)
        imgColors = np.average(colors, axis=0)
        avgColor = np.average(imgColors, axis=0)
        avgColors.append(avgColor)

    print('finding best matches...')
    w = img.size[0]
    h = img.size[1]
    Matches = np.zeros((w, h), dtype=np.uint32)
    for i in range(w):
        for j in range(h):
            pxl = img.getpixel((i, j))
            similarity=[]

            for n in range(len(avgColors)):
                similarity.append(MosaicGenerator.colorsSimilar(pxl, avgColors[n]))

            bestmatch1 = 0
            bestmatch2 = 1
            bestmatch3 = 2
            bestmatch4 = 3
            for n in range(len(similarity)):
                if similarity[n]<=similarity[bestmatch1]:
                    bestmatch4 = bestmatch3
                    bestmatch3 = bestmatch2
                    bestmatch2 = bestmatch1
                    bestmatch1 = n
                elif similarity[n]<=similarity[bestmatch2]:
                    bestmatch4 = bestmatch3
                    bestmatch3 = bestmatch2
                    bestmatch2 = n
                elif similarity[n]<=similarity[bestmatch3]:
                    bestmatch4 = bestmatch3
                    bestmatch3 = n
                elif similarity[n] <= similarity[bestmatch4]:
                    bestmatch4 = n

            bestmatch = []
            bestmatch.append(bestmatch1)
            bestmatch.append(bestmatch2)
            bestmatch.append(bestmatch3)
            bestmatch.append(bestmatch4)
            rnd = random.randint(0, 3)
            Matches[i][j] = bestmatch[rnd]
    return Matches
```

Die statische Methode matchingTiles nimmt Zwei Argumenten (img: Image, tiles: list) und liefert ein NDArray zurück. Das Argument img soll die Target-image sein und das zweite Argument tiles soll eine Liste, die alle Tiles enthält, sein. Bevor diese als Argumente angegeben werden können, muss die Größe geändert und angepasst werden.

Abbildung 8: die Methode matchingTiles der Klasse MosaicGenerator.

In dieser Methode wird zuerst die average colors der Tiles mithelfe der Module Numpy gefunden. Die folgenden Schritte werden für jeder Tile aus der Tiles-Liste gemacht:

1. rgp Werte für alle Pixels von dem Tile „img2“ erhalten.
2. Average Color für jedes Pixel des Tiles rechnen.
3. Average Color für das gesamte Tile rechnen.
4. Rgb Wert aus dem Schritt 3 zu der Liste avgColors addieren.

Da die Tiles nacheinander und in der richtigen reinfolge bearbeitet wurden, ist die Reihenfolge der average Colors wie die Reihenfolge der Tiles. Also avgColors[i] enthalt die average Colors von dem Tile tiles[i]

Nachdem die average Colors aller Tiles gefunden werden, wird die passende Tile für jeder Unterbild der Target-image gesucht.

In den Zeilen 29 und 30 gibt es zwei Variablen w und h, dessen Werte sind die Breite und Höhe des Arguments img.

In der nächsten Zeile wird ein NDArray Matches der Größe (w, h), da w mal h ist gleich der Anzahl aller Unterbilder der Target-image, erzeugt, dies Array enthält w Listen, die aus h Nullen bestehen.

Das Mosaikbild soll aus w horizontalen und vertikalen h Unterbilder bestehen, deshalb gibt es zwei for-Schleifen, die bis w und h laufen. So es ist möglich für jeder Unterbild die passende Tile in der richtigen Stelle zu speichern.

In der Zeile 34 wird die average rbg color des Unterbildes in der Variable pxl gespeichert. Danach eine leere liste similarity wird erzeugt, um die Ähnlichkeiten zu speichern.

Es gibt eine weitere for-Schleife in der Zeile 37, in der pxl mit aller Objekte der Liste avgColors mithilfe der Methode colorsSimilar verglichen wird und den Ergebnissen in der Liste simislarity gespeichert werden.

In den Zeilen 40,41,42 und 43 werden 4 Variablen definiert, um die am besten passenden Tiles zu speichern. In der for-Schleife aus der Zeile 44 werden die vier kleinere Werte aus der Liste similarity gesucht und denen Indexe gespeichert. Danach wird zufällig einen dieser Indexe ausgewählt und in dem NDArray Matches gespeichert.

```
99      @staticmethod
100     def replace_pixels(w1, h1, img, w2, h2, tiles, nearst_tiles):
101         print('replacing pixels with tiles...')
102         for i in range(w1):
103             for j in range(h1):
104                 x = i * w2
105                 y = j * h2
106                 index = nearst_tiles[i][j]
107                 tile = tiles[index]
108                 img.paste(tile, (x, y))
109         return img
```

Abbildung 9: die Methode replace_pixels aus der Klasse MosaicGenerator.

Die Methode replace_pixels nimmt die Argumente w1, h1, img, w2, h2, tiles und nearst_tiles. w1 und h1 repräsentieren die geäderte Größe der Target-image. img soll das Mosaikbild mit der originalen Größe der Target-image sein. x und y sind die Größe der Tiles. Argument tiles soll ein Tiles der Größe (x, y) enthaltende Array sein. nearst_tiles ist die Liste der am besten passenden Tiles Indexe.

In der Zeilen 102 und 103 gibt es zwei for-Schleifen, die bis w1 und h1 laufen. In der nächsten zwei Zeilen wird das Pixel der oberen linken Ecke des Unterbildes bestimmt. Dann wird der Index der passenden Tile für die Stelle [i][j] aus der Array nearst_tiles gelesen und weiterverwendet, um die Tile aus der Variable tiles zu bekommen und in das Mosaikbild zu setzen.

> Zum Beispiel sei i = 0, j=1, w2=10 und h2=10 dann ist x = 0 und y = 10. Und das ist richtig, da hier wird die zweite Tile (tiles[nearst_tiles[0][1]]) in der ersten Spalte addiert und die ersten 10 horizontalen und vertikalen Pixels in dieser Spalte enthalten schon die erste Tile.

```
111     @staticmethod
112     def save_result(img, name):
113         img.save(name)
114         print('mosaic image saved to '+ name)
115         print('Displaying result image...')
116         img.show(img)
117
```

Abbildung 10: die Methode save_result der Klasse MosaicGenerator.

Die Methode save_result bekommt zwei Argumente. Einer der Argumente heißt img und es soll das Mosaikbild sein. Das Andere Argument heißt name und es ist der Pfad, in dem das Mosaikbild gespeichert werden soll.

Nachdem das Mosaikbild gespeichert wird, wird es auf dem Bildschirm gezeigt.

```
4   class Tiles:
5       def import_tiles(path):
6           print('getting tiles paths...')
7           pathsArr = []
8           for imgName in glob.iglob(path):
9               pathsArr.append(imgName)
10
11          print('importing tiles...')
12          tilesArr=[]
13          for path in pathsArr:
14              tile = Image.open(path)
15              tilesArr.append(tile)
16          return tilesArr
17
18      def getresizedTiles(tiles, size):
19          resized = []
20          tilesArr = Tiles.import_tiles(tiles)
21          print('resizing Tiles...')
22          for tile in tilesArr:
23              tile = tile.resize(size)
24              resized.append(tile)
25          return resized
26
```

Abbildung 11: die Klasse Tiles aus dem Datei TILES.py

Die Datei TLES.py enthält einer Klasse Tiles, diese klasse besteht aus wie Methoden

1. Die Methode import_tiles:
   Diese nimmt nur ein Argument, das Argument ist der Pfad des Tiles-verzeichnis. Das Argument wird verwendet, um die Pfade aller Dateien dieses Verzeichnis zu erhalten und zu speichern. Danach werden die Pfade weiterverwendet, um alle Dateien als Image Objekte zu haben und in der Liste tilesArr zu speichern.
2. Die Methode getresizedTiles:
   Diese hat zwei Argumente, eine ist der Pfad des Tiles-verzeichnis und die zweite ist ein Tupel, die aus zwei Integres besteht und die Größe der Tiles in dem Mosaikbild repräsentiert. Zuerst werden die Tiles durch Verwendung der Methode import_tiles importiert und dann wird die Größe aller Tiles geändert und in einer neuen List gespeichert. Zuletzt wird die List zurückgeliefert

---

❖ Ausführen des Programms

Zuerst müssen drei Module mit den folgenden Befehlen installiert werden:

1. pip3 install numpy
2. pip3 install pillow
3. pip3 install glob

Das Programm kann auf drei Arten ausgeführt werden. Als der Nutzer das Verzeichnis des Projekts geöffnet hat, kann er dann das Programm mit einer der folgenden 3 Vorgehensweisen ausführen:

1. Die Kommandozeile öffnen und der folgende Befehl angeben:

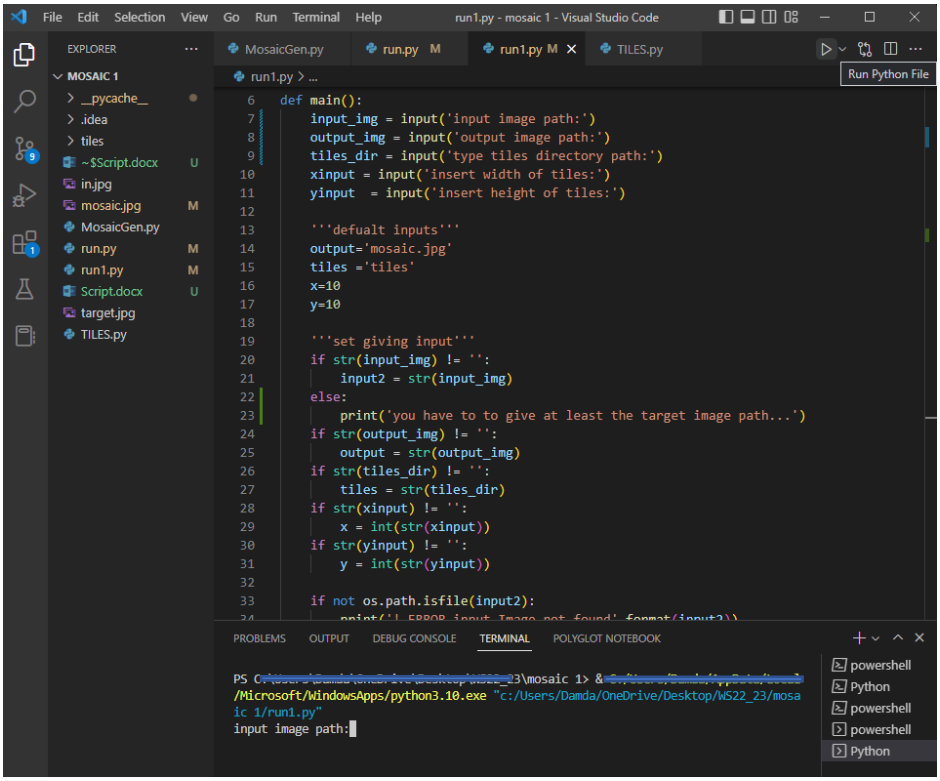
python run.py -i <Target-image> -o <output img> -t <tiles directory> -x <Breite> -y <Höhe>

2. Die Kommandozeile öffnen und der folgende Befehl angeben:
   Dann die Eingaben angeben.

python run1.py

```
PS C:\Users\Damda\OneDrive\Desktop\WS22_23\mosaic 1> python run1.py
input image path: <Target-image>
output image path: <output image>
type tiles directory path: <tiles directory>
insert width of tiles: <Breite>
insert height of tiles: <Höhe>
```

3. Der Datei run1.py auf einen Interpreter öffnen und auf der run Taste drücken



Auch wie in der Vorgehensweise Nr. 2 müssen die Eingaben in der Konsole angegeben werden.