Author: Ivar Holmlund

Report for mandatory excercise 2

In [1]:
```
import time
from skimage import io
import matplotlib.pyplot as plt
from skimage.morphology import dilation
from skimage.morphology import area_closing, area_opening
from skimage.segmentation import watershed
from skimage.filters import threshold_triangle
from skimage.util import invert
import numpy as np
from skimage import measure
from skimage.feature import peak_local_max
start_time = time.time() #Time the script
```

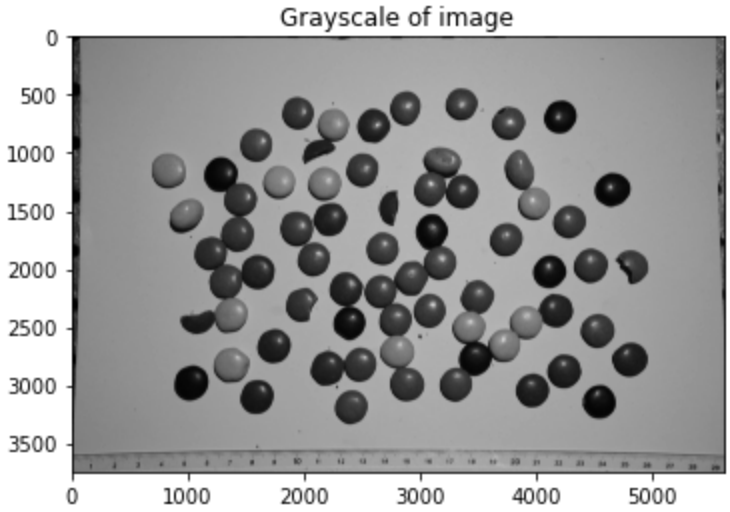
Using threshold function from mandatory exc 1

- Changed values for foreground to 1 instead of 255
- Makes masking later easier

In [2]:
```
def threshold(image, th=None):
    shape = np.shape(image)
    binarised = np.zeros(shape)
    if len(shape) == 3:
        if th is None:
            th = otsu(image)
            print(th)
        image = image.mean(axis=2)
    elif len(shape) > 3:
        raise ValueError('Must be at 2D image')
    for i, row in enumerate(image):
        for j, value in enumerate(row):
            if value >= th:
                image[i][j] = 0
            else:
                image[i][j] = 1
    return image
```

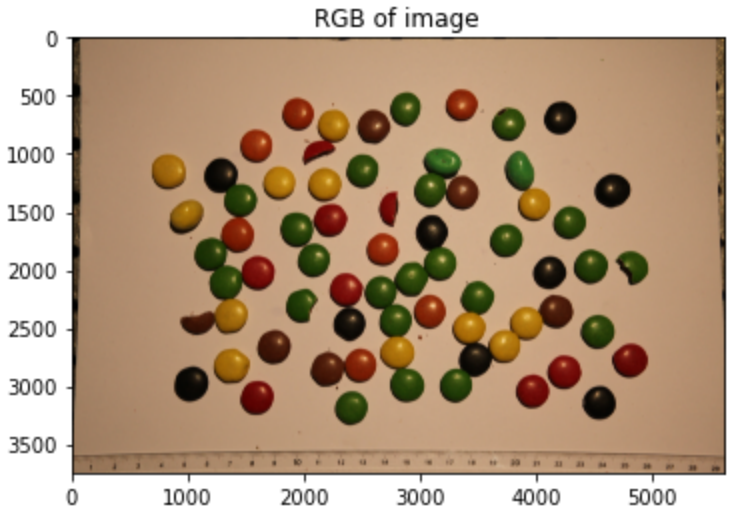
Loading in figure

In [3]:
```
ch = io.imread('IMG_2754_nonstop_alltogether.jpg', as_gray = True)
ch_rgb = io.imread('IMG_2754_nonstop_alltogether.jpg', as_gray = False)
plt.figure()
plt.title('Grayscale of image')
plt.imshow(ch, 'gray')
plt.show()
```



In [4]:
```
plt.title('RGB of image')
plt.imshow(ch_rgb)
```

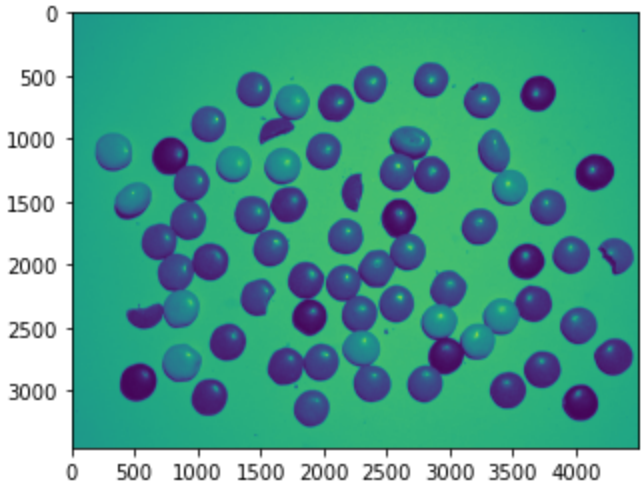
Out[4]:
```
<matplotlib.image.AxesImage at 0x20480034700>
```



Minimize image

- To reduce noise/unnescescary information around the edge
- Increase computation time

In [5]:
```
minimized_image = []
x1 = 50
x2 = 3500
y1 = 500
y2 = 5000

minimized_image = ch[x1:x2, y1:y2]
minimized_image_rgb = ch_rgb[x1:x2, y1:y2]

plt.figure()
plt.imshow(minimized_image)
plt.show()
```

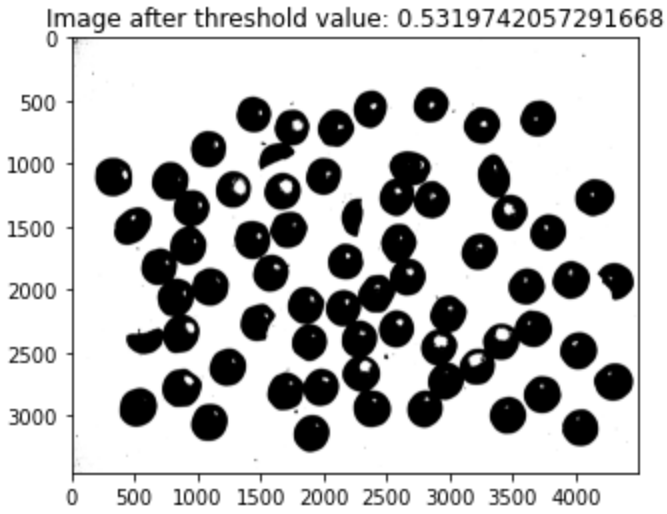


Threshold value and inverting

- Using triangle threshold for value
- Foreground values are 1, so will not invert

In [6]:
```
triangle_threshold = threshold_triangle(minimized_image)
image = threshold(minimized_image, triangle_threshold)
#image = invert(image)
plt.title(f'Image after threshold value: {triangle_threshold}')
plt.imshow(image, 'binary')
```

Out[6]:
```
<matplotlib.image.AxesImage at 0x20483df8b20>
```

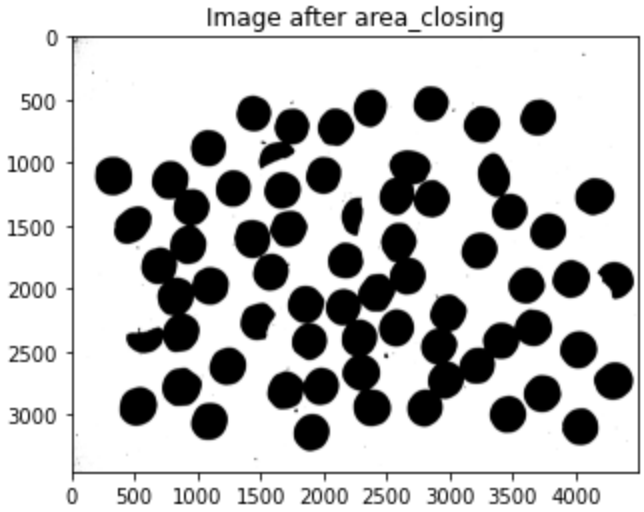


Area_closing

- Using area closing to close all foreground objects

In [7]:
```
from skimage.morphology import area_closing
image_ac = area_closing(image, area_threshold=12000)

plt.figure()
plt.title('Image after area_closing')
plt.imshow(image_ac, 'binary')
```

Out[7]:
```
<matplotlib.image.AxesImage at 0x20480056820>
```

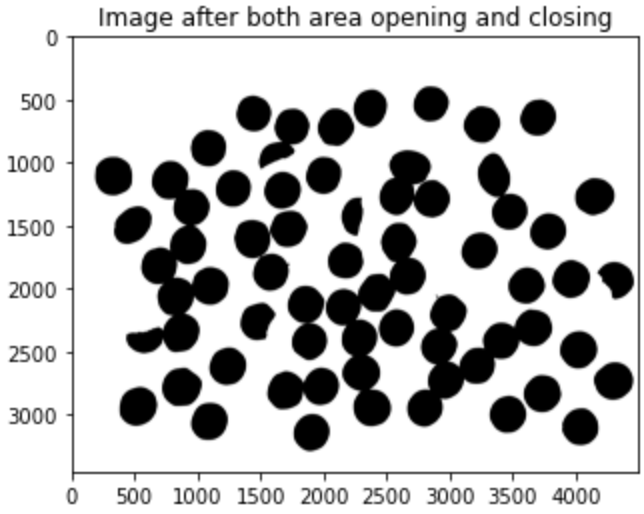


Using area_opening

- Remove noise and objects less than area_threshold

In [8]:
```
image_f = area_opening(image_ac, area_threshold = 12000)
plt.figure()
plt.title('Image after both area opening and closing')
plt.imshow(image_f, 'binary')
```

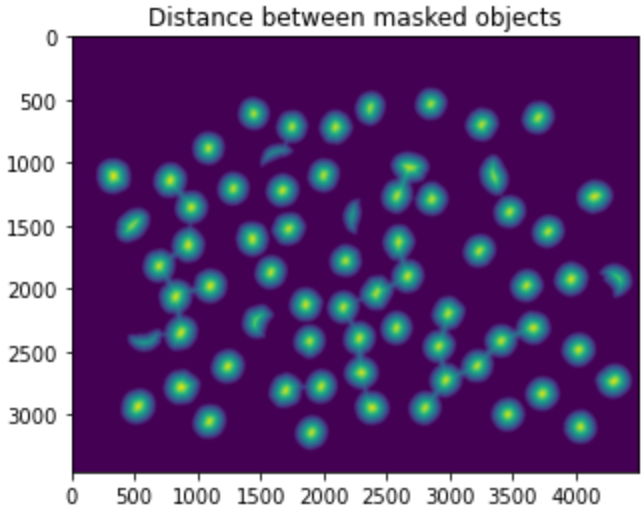
Out[8]:
```
<matplotlib.image.AxesImage at 0x20483e0ef40>
```



Finding distance between masked objects

In [9]:
```
from scipy import ndimage as ndi
distance = ndi.distance_transform_edt(image_f)
plt.title('Distance between masked objects')
plt.imshow(distance)
```

Out[9]:
```
<matplotlib.image.AxesImage at 0x2048005ab50>
```



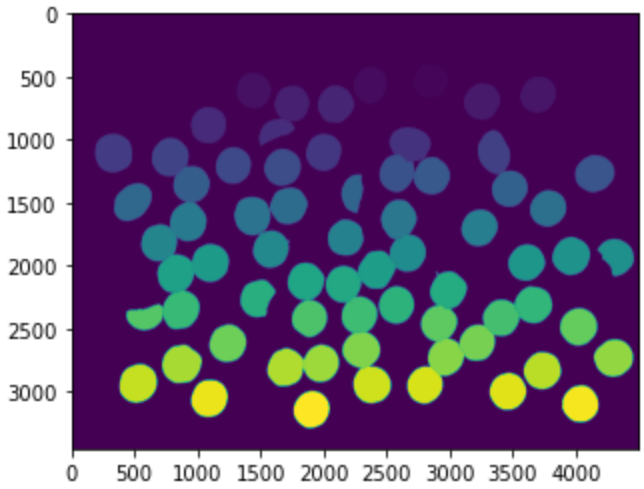
Using watershed to differentiate objects better

- Using watershed_line = True to see where the watershed lines are set
- Using min_distance in peak_local_max to ensure that local peaks are not to close
- Iterating over with a 30x30 sqaure footprint to ensure capturing large areas
  - This is to avoid one object, being split into two
- Can use image_f as labls since it already is only 1 and 0's

In [10]:

```
import warnings #To ignore a futurewarning about indices
warnings.simplefilter(action='ignore', category=FutureWarning)

local_max = peak_local_max(distance,
                           indices = False,
                           footprint=np.ones((30, 30)),
                           labels = image_f.astype(int),
                           min_distance = 150)

markers, _ = ndi.label(local_max)
labels = watershed(-distance, markers, mask = image_f, watershed_line = True)
plt.imshow(labels)
print(f'Amount of objects = {len(np.unique(labels))}')
```

```
Amount of objects = 69
```



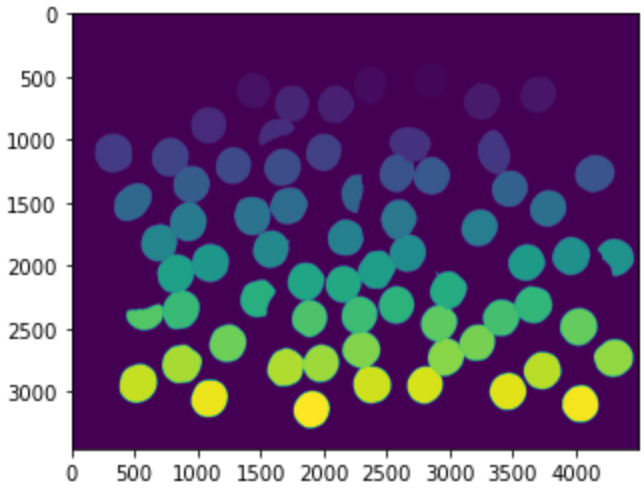
Creating a mask constisting of objects location and name/value

In [11]:

```
labels2 = measure.label(labels)
plt.imshow(labels2)
```

```
properties = measure.regionprops(labels2)
print(f'Amount of objects in image: {len(properties)}')
```

```
Amount of objects in image: 68
```



Removing outliers and unwanted objects

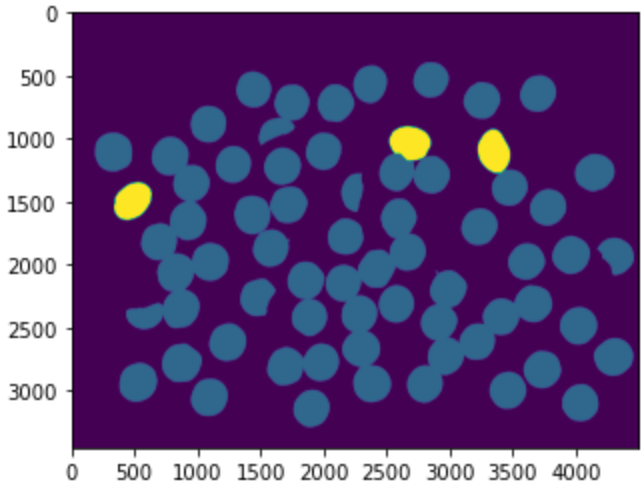
- Removing based on size and eccentricity
- Will remove objects, that does not fullfill set criterias.
- Will use list comprehension with several if-test to find the different suitable areas'

Values used to find MM's

- *Eccentricity* >= 0.30
  - Measure how round the object is
    - 0 is a perfect circle, 1 is an ellipse
- *Area* >= 60000
  - Counts amount of pixels object contains.
- *Major axis length* >= 330
  - Longest straight line possible to create over object.

In [12]:
```
truncated_properties_MM = [prop for prop in properties if prop.eccentricity>=0.30 and prop
                       and prop.major_axis_length >= 330]
image_MM = dilation(image_f, footprint = np.ones((2,2)))
for obj in truncated_properties_MM:
    for coord in obj.coords:
        image_MM[coord[0], coord[1]] = 3
plt.imshow(image_MM)
```

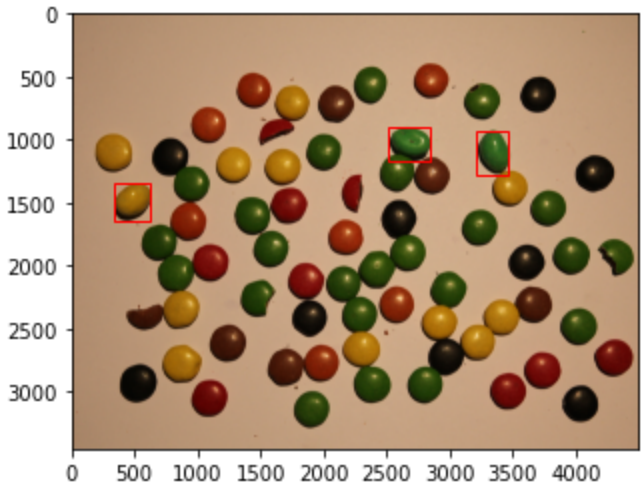
Out[12]:
```
<matplotlib.image.AxesImage at 0x20484697c10>
```



Adding red rectangles around MM's

In [13]:

```
from matplotlib.patches import Rectangle
rectangles = []
for obj in truncated_properties_MM:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))

plt.imshow(minimized_image_rgb)
for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                      edgecolor='red',
                      facecolor='none',
                      lw=1))
stop_time = time.time()
```



Time to run script

In [14]:

```
print(f'The script used: {stop_time - start_time} seconds')
```

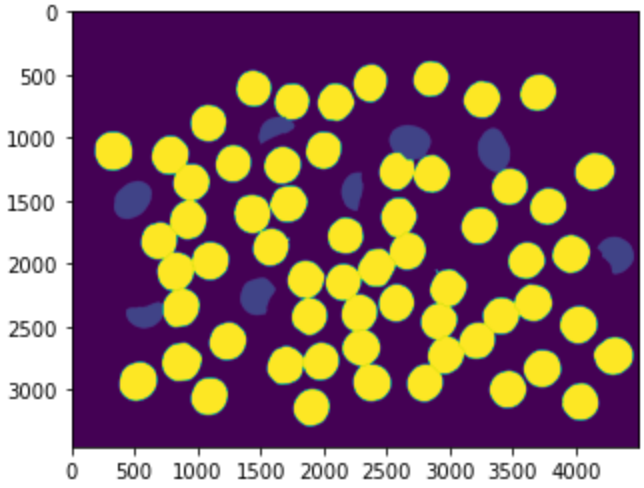
```
The script used: 51.92501187324524 seconds
```

Extra

Finding whole and broken nonstops in image

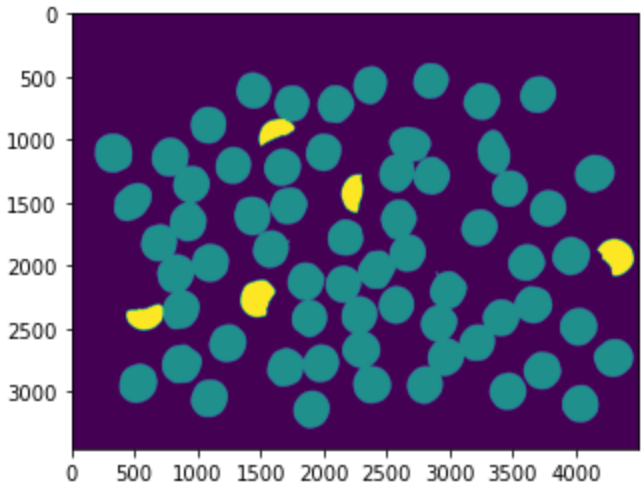
In [15]:
```
truncated_properties_whole_nn = [prop for prop in properties if prop.eccentricity<=0.49 ar
                        and prop.major_axis_length >= 200]
image_MM_whole = dilation(image_f, footprint = np.ones((2,2)))

for obj in truncated_properties_whole_nn:
    for coord in obj.coords:
        image_MM_whole[coord[0], coord[1]] = 5
plt.imshow(image_MM_whole)
```

Out[15]:
```
<matplotlib.image.AxesImage at 0x20484b8aca0>
```



In [16]:
```
truncated_properties_broken_nn = []
truncated_properties_broken_nn = [prop for prop in properties if prop.eccentricity>=0.55 a
                        and prop.major_axis_length >= 280]
image_MM_broken = dilation(image_f, footprint = np.ones((2,2)))

for obj in truncated_properties_broken_nn:
    for coord in obj.coords:
        image_MM_broken[coord[0], coord[1]] = 2
plt.imshow(image_MM_broken)
```

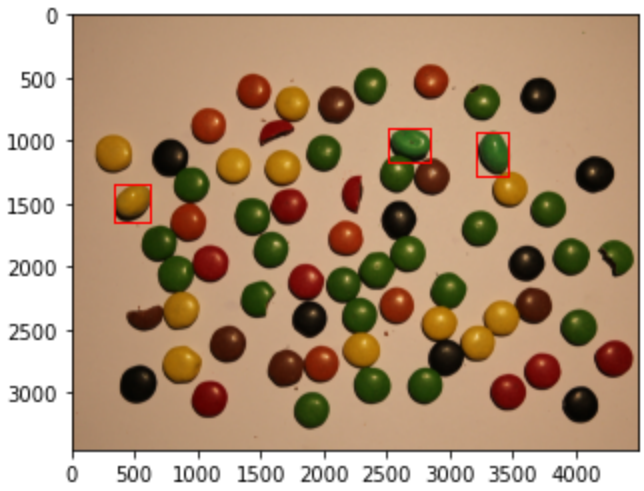
Out[16]:
```
<matplotlib.image.AxesImage at 0x20484c3ba90>
```



In [17]:

```
rectangles1 = []
for obj in truncated_properties_MM:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))

for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                        edgecolor='red',
                        facecolor='none',
                        lw=1))

plt.imshow(minimized_image_rgb)
```

Out[17]: <matplotlib.image.AxesImage at 0x20483f95640>



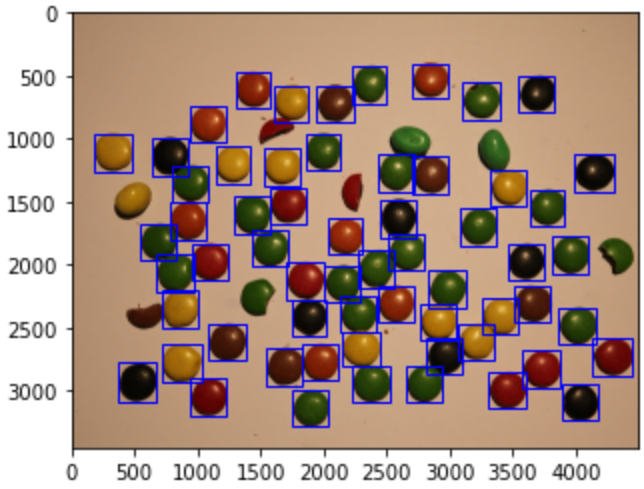
Finding whole nonstops in image

In [18]:

```
fig = plt.figure()
rectangles = []
for obj in truncated_properties_whole_nn:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))


for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                        edgecolor='blue',
                        facecolor='none',
```

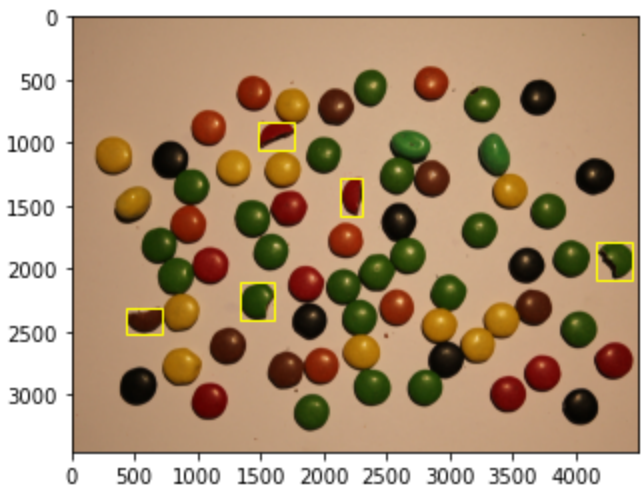
```
                    lw=1))
plt.imshow(minimized_image_rgb)
```

Out[18]:
```
<matplotlib.image.AxesImage at 0x20483f95a30>
```



Finding broken nonstops

In [19]:
```
rectangles = []
for obj in truncated_properties_broken_nn:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))

plt.figure()

for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                        edgecolor='yellow',
                        facecolor='none',
                        lw=1))
plt.imshow(minimized_image_rgb)
stop_time = time.time()
```

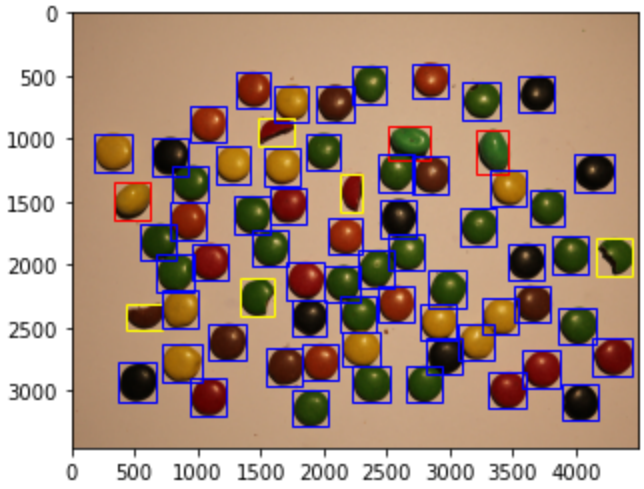


In [20]:

```
plt.figure()
rectangles1 = []
for obj in truncated_properties_MM:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))

for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                    edgecolor='red',
                    facecolor='none',
                    lw=1))

rectangles = []
for obj in truncated_properties_broken_nn:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))

for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                    edgecolor='yellow',
                    facecolor='none',
                    lw=1))

rectangles = []
for obj in truncated_properties_whole_nn:
    X_values = []
    Y_values = []
    for coord in obj.coords:
        X_values.append(coord[0])
        Y_values.append(coord[1])

    min_x = min(X_values)
    min_y = min(Y_values)
    max_x = max(X_values)
    max_y = max(Y_values)
    rectangles.append((min_y, min_x, max_y-min_y, max_x-min_x,))


for rec in rectangles:
    plt.gca().add_patch(Rectangle((rec[0],rec[1]),
                                  rec[2],rec[3],
                    edgecolor='blue',
                    facecolor='none',
```

```
                        lw=1))
plt.imshow(minimized_image_rgb)
```

Out[20]:
```
<matplotlib.image.AxesImage at 0x20484d58d00>
```



In [21]:
```
print(f'The script included extra used: {stop_time - start_time} seconds')
```

```
The script included extra used: 62.86484622955322 seconds
```