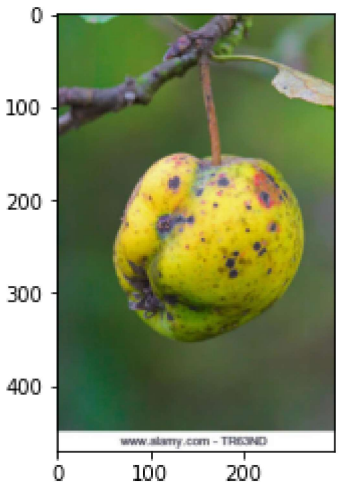
In [1]:
```
#importing the libraries..
#NumPy is a Python library used for working with arrays.
#Matplotlib is a cross-platform,used for data visualization and graphical plotting.
#scikit-image (a.k.a. skimage ) is a collection of algorithms for image processing and

import numpy as np
import os
import matplotlib.pyplot as plt
from skimage.io import imread
from skimage.transform import resize
```

In [2]:
```
# Load training images and preprocessing it..
target = []
train_image = []
flat_data = []

data_directory= 'Train'
categories = ['Blotch_Apple', 'Normal_Apple','Rot_Apple','Scab_Apple']

for category in categories:
  class_num=categories.index(category)# label Encoding the values
  path=os.path.join(data_directory, category)# create path to use  all the images
  for img in os.listdir(path):
    img_array = imread(os.path.join(path, img))
    #print(img_array.shape)
    plt.imshow(img_array)
    img_resized = resize(img_array,(150,150,3))
    flat_data.append(img_resized.flatten())
    train_image.append(img_resized)
    target.append(class_num)

flat_data = np.array(flat_data)
target = np.array(target)
train_image=np.array(train_image)
```



In [3]:
```
len(flat_data[0])
```

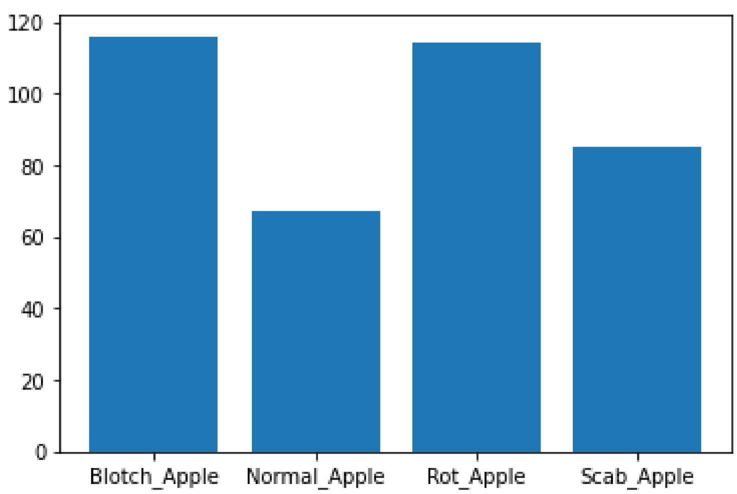
Out[3]:
```
67500
```

In [4]:
```
target
```

Out[4]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3])
```

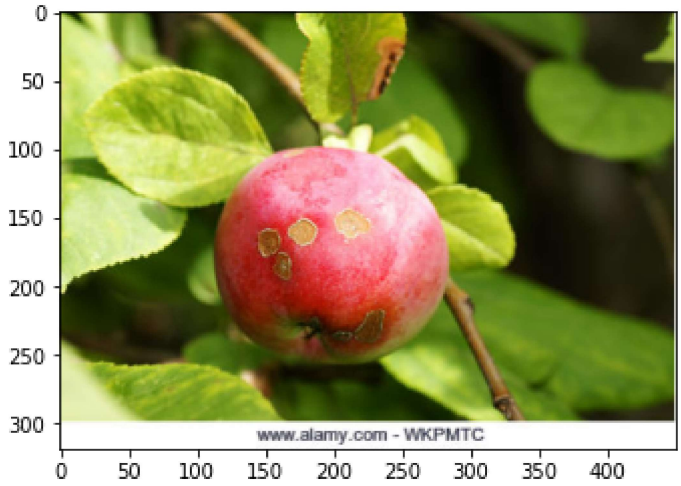
In [5]:
```
#BAR GRAPH REPRESENTING THE COUNT OF APPLE DISEASE IMAGES OF TRAINED IMAGES...
unique, count = np.unique(target, return_counts=True)
plt.bar(categories, count)
```

Out[5]: <BarContainer object of 4 artists>



In [6]:
```
# Load test images
target = []
test_image = []
flat_data = []

data_directory= 'Test'
categories = ['Blotch_Apple', 'Normal_Apple','Rot_Apple','Scab_Apple']

for category in categories:
  class_num=categories.index(category)
  path=os.path.join(data_directory, category)
  for img in os.listdir(path):
    img_array = imread(os.path.join(path, img))
    #print(img_array.shape)
    plt.imshow(img_array)
    img_resized = resize(img_array,(150,150,3))
    flat_data.append(img_resized.flatten())
    test_image.append(img_resized)
    target.append(class_num)
```

```
flat_data = np.array(flat_data)
target = np.array(target)
test_image=np.array(test_image)
```



In [7]:
```
len(flat_data[0])
```

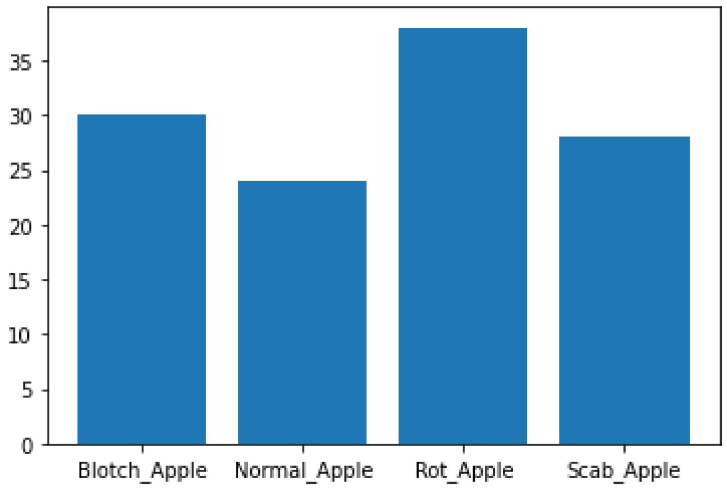
Out[7]:
```
67500
```

In [8]:
```
target
```

Out[8]:
```
array([0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2, 2,
       2, 2, 2, 2, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3, 3,
       3, 3, 3, 3, 3, 3, 3, 3, 3, 3])
```

In [9]:
```
#BAR GRAPH REPRESENTING THE COUNT OF APPLE DISEASE IMAGES OF TESTED IMAGES...
unique, count = np.unique(target, return_counts=True)
plt.bar(categories, count)
```

Out[9]:
```
<BarContainer object of 4 artists>
```



In [10]:
```
feature = np.ndarray(shape = (len(train_image), 150,150 ,3),
                     dtype = np.float32)
```

```
for i in range(len(train_image)):
    feature[i] = train_image[i]

feature = feature / 382
```

In [11]:
```
feature
```

Out[11]:
```
array([[[[1.7823797e-03, 1.4538714e-03, 1.2177558e-03],
         [1.7905786e-03, 1.4620703e-03, 1.2259549e-03],
         [1.8048363e-03, 1.4763279e-03, 1.2402125e-03],
         ...,
         [1.8240173e-03, 1.5231936e-03, 1.3013851e-03],
         [1.8080394e-03, 1.5205538e-03, 1.2947173e-03],
         [1.8141499e-03, 1.5267051e-03, 1.3008555e-03]],

        [[1.7929646e-03, 1.4644563e-03, 1.2283409e-03],
         [1.8094497e-03, 1.4809412e-03, 1.2448259e-03],
         [1.8298622e-03, 1.5013539e-03, 1.2652385e-03],
         ...,
         [1.8299794e-03, 1.5362604e-03, 1.3122959e-03],
         [1.8287443e-03, 1.5412839e-03, 1.3154394e-03],
         [1.8128854e-03, 1.5254406e-03, 1.2995910e-03]],

        [[1.8001815e-03, 1.4716731e-03, 1.2355577e-03],
         [1.8233942e-03, 1.4948859e-03, 1.2587705e-03],
         [1.8476285e-03, 1.5191202e-03, 1.2830048e-03],
         ...,
         [1.8312862e-03, 1.5442120e-03, 1.3180743e-03],
         [1.8234642e-03, 1.5360210e-03, 1.3101706e-03],
         [1.8058923e-03, 1.5184483e-03, 1.2925984e-03]],

        ...,

        [[1.6306615e-03, 1.2472487e-03, 9.8600029e-04],
         [1.6372026e-03, 1.2613080e-03, 9.5537625e-04],
         [1.6426544e-03, 1.2624324e-03, 9.8947017e-04],
         ...,
         [2.0635985e-05, 1.7822900e-06, 4.4812523e-06],
         [2.8971766e-05, 4.8390784e-06, 1.1655227e-05],
         [4.4465292e-04, 3.1652747e-04, 2.6503662e-04]],

        [[1.6314524e-03, 1.2446538e-03, 1.0071405e-03],
         [1.6472708e-03, 1.2670337e-03, 9.8411995e-04],
         [1.6520717e-03, 1.2650723e-03, 1.0159756e-03],
         ...,
         [1.6323818e-05, 3.6877950e-06, 4.0617735e-05],
         [1.8052200e-05, 1.1898218e-05, 5.6747303e-05],
         [4.0519849e-04, 3.2884808e-04, 3.0417499e-04]],

        [[1.6521943e-03, 1.2634028e-03, 1.0432417e-03],
         [1.6546273e-03, 1.2734508e-03, 1.0044051e-03],
         [1.6536522e-03, 1.2611627e-03, 1.0310287e-03],
         ...,
         [6.0964312e-04, 4.4082062e-04, 3.9923171e-04],
         [5.9502025e-04, 4.5419007e-04, 4.1035376e-04],
         [8.5136702e-04, 6.6401891e-04, 5.4304651e-04]]],


       [[[2.0389862e-03, 1.7926048e-03, 1.3203740e-03],
         [2.0439255e-03, 1.7975442e-03, 1.3253135e-03],
         [2.0528694e-03, 1.8064883e-03, 1.3342573e-03],
         ...,
         [1.6085292e-05, 1.6085292e-05, 1.6085292e-05],
         [8.2257593e-06, 8.2257593e-06, 8.2257593e-06],
         [5.3805154e-04, 5.3805154e-04, 5.3805154e-04]],

        [[2.0429119e-03, 1.7965307e-03, 1.3259425e-03],
```

```
  [2.0443641e-03, 1.7979828e-03, 1.3273946e-03],
  [2.0534880e-03, 1.8071069e-03, 1.3365187e-03],
  ...,
  [1.6085292e-05, 1.6085292e-05, 1.6085292e-05],
  [8.2257593e-06, 8.2257593e-06, 8.2257593e-06],
  [5.3805154e-04, 5.3805154e-04, 5.3805154e-04]],

 [[2.0430302e-03, 1.7966490e-03, 1.3449499e-03],
  [2.0499432e-03, 1.8035619e-03, 1.3518630e-03],
  [2.0608450e-03, 1.8144639e-03, 1.3627649e-03],
  ...,
  [1.6085292e-05, 1.6085292e-05, 1.6085292e-05],
  [8.2257593e-06, 8.2257593e-06, 8.2257593e-06],
  [5.3805154e-04, 5.3805154e-04, 5.3805154e-04]],

 ...,

 [[1.2348962e-03, 1.2348962e-03, 1.2348962e-03],
  [1.5450871e-03, 1.5450871e-03, 1.5450871e-03],
  [1.2449991e-03, 1.2449991e-03, 1.2449991e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.4506745e-03, 2.4506745e-03, 2.4506745e-03],
  [2.4747676e-03, 2.4747676e-03, 2.4747676e-03],
  [1.7842756e-03, 1.7842756e-03, 1.7842756e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6162954e-03, 2.6162954e-03, 2.6162954e-03],
  [2.5881536e-03, 2.5881536e-03, 2.5881536e-03],
  [2.1137332e-03, 2.1137332e-03, 2.1137332e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]]],


[[[2.4022174e-03, 2.2995586e-03, 2.1763679e-03],
  [2.4022174e-03, 2.2995586e-03, 2.1763679e-03],
  [2.4022174e-03, 2.2995586e-03, 2.1763679e-03],
  ...,
  [2.2997286e-03, 2.2689309e-03, 2.1149428e-03],
  [2.2968899e-03, 2.2660922e-03, 2.1121039e-03],
  [2.2893662e-03, 2.2585685e-03, 2.1045802e-03]],

 [[2.4022004e-03, 2.2995414e-03, 2.1763507e-03],
  [2.4022004e-03, 2.2995414e-03, 2.1763507e-03],
  [2.4022004e-03, 2.2995414e-03, 2.1763507e-03],
  ...,
  [2.2983365e-03, 2.2675388e-03, 2.1135504e-03],
  [2.2895462e-03, 2.2587485e-03, 2.1047601e-03],
  [2.2840663e-03, 2.2532686e-03, 2.0992805e-03]],

 [[2.3919633e-03, 2.2893045e-03, 2.1661138e-03],
  [2.3919633e-03, 2.2893045e-03, 2.1661138e-03],
```

```
  [2.3919633e-03, 2.2893045e-03, 2.1661138e-03],
  ...,
  [2.2851066e-03, 2.2543089e-03, 2.1003205e-03],
  [2.2798264e-03, 2.2490288e-03, 2.0950404e-03],
  [2.2747905e-03, 2.2439929e-03, 2.0900045e-03]],

 ...,

 [[2.4535351e-03, 2.3508761e-03, 2.2584831e-03],
  [2.4528620e-03, 2.3502028e-03, 2.2578100e-03],
  [2.4441089e-03, 2.3414502e-03, 2.2490572e-03],
  ...,
  [1.7952289e-03, 1.7644076e-03, 1.7130901e-03],
  [1.7776099e-03, 1.7467887e-03, 1.6954710e-03],
  [1.7698719e-03, 1.7390507e-03, 1.6877331e-03]],

 [[2.4432982e-03, 2.3406392e-03, 2.2482462e-03],
  [2.4432610e-03, 2.3406022e-03, 2.2482092e-03],
  [2.4344490e-03, 2.3317903e-03, 2.2393973e-03],
  ...,
  [1.7967802e-03, 1.7454850e-03, 1.7044044e-03],
  [1.7962502e-03, 1.7449552e-03, 1.7038744e-03],
  [1.7863443e-03, 1.7350491e-03, 1.6939685e-03]],

 [[2.4432810e-03, 2.3406220e-03, 2.2482292e-03],
  [2.4432437e-03, 2.3405850e-03, 2.2481920e-03],
  [2.4344318e-03, 2.3317731e-03, 2.2393800e-03],
  ...,
  [1.7967825e-03, 1.7454531e-03, 1.7043895e-03],
  [1.7962814e-03, 1.7449521e-03, 1.7038885e-03],
  [1.7863719e-03, 1.7350425e-03, 1.6939790e-03]]],


...,


[[[9.7976578e-04, 8.4511086e-04, 3.3979895e-04],
  [1.0375311e-03, 8.7798969e-04, 4.3712294e-04],
  [1.0656248e-03, 9.0562197e-04, 3.7120935e-04],
  ...,
  [1.4304646e-03, 1.6050314e-03, 9.0491702e-04],
  [1.4244020e-03, 1.5992889e-03, 8.9944777e-04],
  [1.2939094e-03, 1.5242764e-03, 8.2443532e-04]],

 [[8.9751487e-04, 7.0950040e-04, 3.8990469e-04],
  [9.6758263e-04, 8.3393004e-04, 2.8182770e-04],
  [9.8842673e-04, 8.5093465e-04, 3.4568572e-04],
  ...,
  [1.3915887e-03, 1.5661556e-03, 8.4721920e-04],
  [1.4104372e-03, 1.5852378e-03, 8.6648331e-04],
  [1.3454231e-03, 1.5626504e-03, 8.4315264e-04]],

 [[8.4599189e-04, 6.0961692e-04, 3.6199941e-04],
  [9.3447999e-04, 7.8821456e-04, 3.1289074e-04],
  [9.0308802e-04, 8.0095330e-04, 2.8176143e-04],
  ...,
  [1.3561660e-03, 1.5320060e-03, 7.9943379e-04],
  [1.3587368e-03, 1.5347539e-03, 8.0227788e-04],
  [1.3520395e-03, 1.5610269e-03, 8.2429737e-04]],
```

```
 ...,

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]]],


[[[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 ...,

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],
```

```
 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]]],


[[[5.5910257e-04, 1.2680003e-03, 4.9536087e-04],
  [6.1928475e-04, 1.2471754e-03, 5.0223776e-04],
  [6.4837903e-04, 1.2450638e-03, 5.0980301e-04],
  ...,
  [1.0927051e-03, 1.5222664e-03, 1.2216634e-03],
  [1.1192380e-03, 1.5091100e-03, 1.2234624e-03],
  [1.1305152e-03, 1.4783698e-03, 1.2219559e-03]],

 [[5.9774134e-04, 1.2652605e-03, 5.0698355e-04],
  [6.0020853e-04, 1.2724772e-03, 5.1226938e-04],
  [6.2520819e-04, 1.2721191e-03, 5.1910768e-04],
  ...,
  [1.0617931e-03, 1.5197891e-03, 1.0820357e-03],
  [1.0817772e-03, 1.4994300e-03, 1.0637704e-03],
  [1.0949619e-03, 1.4729322e-03, 1.0482962e-03]],

 [[6.5343903e-04, 1.2417033e-03, 5.3875812e-04],
  [6.4034248e-04, 1.2556299e-03, 5.3686253e-04],
  [6.6558766e-04, 1.2602720e-03, 5.3891889e-04],
  ...,
  [1.0908296e-03, 1.4659220e-03, 9.3169603e-04],
  [1.0558187e-03, 1.4338703e-03, 8.8211254e-04],
  [1.0241795e-03, 1.4085108e-03, 8.5325120e-04]],

 ...,

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],

 [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  ...,
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
  [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]],
```

```
        [[2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
         [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
         [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
         ...,
         [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
         [2.6178011e-03, 2.6178011e-03, 2.6178011e-03],
         [2.6178011e-03, 2.6178011e-03, 2.6178011e-03]]], dtype=float32)
```

In [12]:
```
x_test = np.ndarray(shape = (len(test_image), 150, 150, 3),
                    dtype = np.float32)

for i in range(len(test_image)):
    x_test[i] = test_image[i]

x_test = x_test / 120
```

In [13]:
```
#splitting data into train and test

from sklearn.model_selection import train_test_split
x_train, x_test, y_train, y_test = train_test_split(flat_data, target, test_size=0.20,
```

In [14]:
```
from sklearn.preprocessing import StandardScaler #It normalises
sc = StandardScaler()
x_train = sc.fit_transform(x_train)#fit() method is used while working with model to c
#method uses these parameters/weights on the test data to predict the output.
x_test = sc.transform(x_test)
```

In [15]:
```
x_train
```

Out[15]:
```
array([[-0.6071266 , -1.27999838, -1.07889575, ..., -1.23932348,
        -1.87908636, -1.59728432],
       [ 0.39890579, -0.89537811, -1.03716048, ...,  0.20426687,
        -1.61784698, -1.33435985],
       [ 1.1096276 ,  1.20621754,  1.2596338 , ...,  0.86831548,
         0.91343834,  0.92730999],
       ...,
       [ 0.05312729,  0.49521508, -0.05602006, ...,  0.86831548,
         0.91343834,  0.92730999],
       [-0.13289778,  0.01545322,  0.10133574, ..., -0.77942151,
        -0.697005  , -0.62074795],
       [-0.28172596, -0.43011998, -0.67118   , ...,  0.86831548,
         0.91343834,  0.92730999]])
```

In [16]:
```
x_test
```

Out[16]:
```
array([[-1.42825841, -1.34234033, -1.05207471, ...,  0.86831548,
         0.91343834,  0.92730999],
       [ 1.1096276 ,  1.20621754,  1.2596338 , ...,  0.86831548,
         0.91343834,  0.92730999],
       [ 1.1096276 ,  1.20621754,  1.2596338 , ...,  0.86831548,
         0.91343834,  0.92730999],
       ...,
       [ 1.1096276 ,  1.20621754,  1.2596338 , ...,  0.86831548,
         0.91343834,  0.92730999],
       [ 0.95811262,  1.06830289,  1.18632962, ..., -1.19018647,
        -1.37068636, -1.77007124],
       [ 1.08631761,  1.20621754,  1.24916177, ..., -0.48915842,
        -0.47104511,  0.36498171]])
```

In [17]:
```
#FITTING THE NAIVE BAYES CLASSIFIER..
```

In [18]:
```
from sklearn.naive_bayes import GaussianNB
nvclassifier = GaussianNB()
nvclassifier.fit(x_train,y_train)
```

Out[18]:
```
GaussianNB()
```

In [19]:
```
#predicting the Test set results.
y_predicting = nvclassifier.predict(x_test)
print(y_predicting)
```

```
[3 2 2 1 2 0 2 1 1 1 2 2 3 0 2 0 0 2 0 0 2 2 1 1]
```

In [20]:
```
#we can check accuracy of naive bayes usong confusion matrix
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y_test, y_predicting)
print(cm)
```

```
[[4 1 0 0]
 [0 4 1 0]
 [0 0 8 1]
 [2 1 1 1]]
```

In [21]:
```
print(cm)
```

```
[[4 1 0 0]
 [0 4 1 0]
 [0 0 8 1]
 [2 1 1 1]]
```

In [22]:
```
a = cm.shape
corrPred = 0
falsePred = 0
for row in range(a[0]):
    for c in range(a[1]):
        if row==c:
            corrPred+= cm[row,c]

        else:
            falsePred = cm[row,c]

print("correct prediction: ",corrPred)
print("False prediction: ",falsePred)
print("\n\n Accuracy of the naive bayes classification is  ",corrPred/(cm.sum()))
```

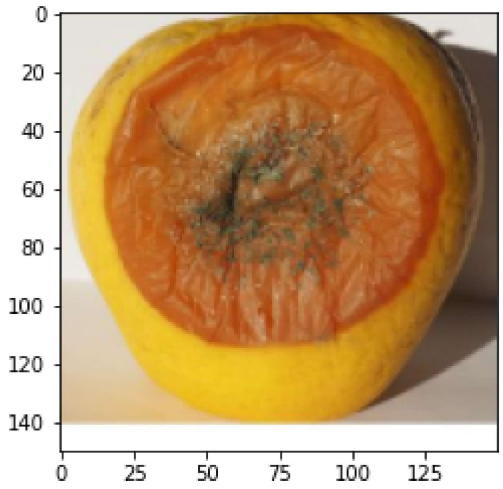
```
correct prediction:  17
False prediction:  1


 Accuracy of the naive bayes classification is   0.7083333333333334
```

In [23]:
```
# SAVE THE MODEL USING PICKLE LIIBRARY
import pickle
pickle.dump(nvclassifier,open('img_model.p','wb'))
```

In [24]:
```
model = pickle.load(open('img_model.p','rb'))
```

In [25]:
```
# testing a  Diseased images.

flat_data = []
url = str("Test/Rot_Apple/2apbkfh.jpg")
img = imread(url)
img_resized = resize(img,(150,150,3))
flat_data.append(img_resized.flatten())
flat_data = np.array(flat_data)
print(img.shape)
plt.imshow(img_resized)
y_out = model.predict(flat_data)
y_out = categories[y_out[0]]
print(f'Predicted Output : {y_out}')
```

```
(320, 308, 3)
Predicted Output : Rot_Apple
```



In [ ]: