## Self-Organizing Map of RGB Colors

In [13]:

```
import numpy as np
import matplotlib.pyplot as plt
```

In [14]:

```
n_in = 3 # number of input values
nr, nc = 15, 15  # number of rows and columns of the output map
w = np.random.rand(nr, nc, 3)  # the weights of each node
nb = int(np.sqrt(nr))  # neighborhood --- sqrt of total nodes
lr = 0.05  # learning rate
n_iters = 10000 # number of training iterations
```

In [15]:

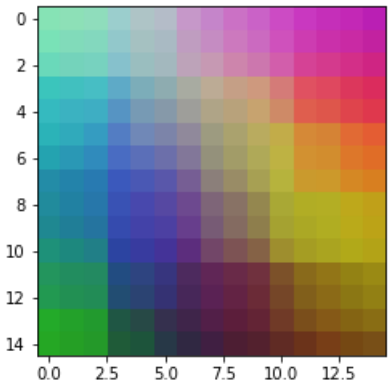
```
for i in range(3000):
    rand_in = np.random.rand(1, 1, 3)  # generate a random example
    dist = np.sum(np.absolute(w - rand_in), 2)  # distance between weights and input
    best = np.where(dist == np.amin(dist))  # lowest distance
    best_r, best_c = best[0][0], best[1][0]  # get row and column of lowest distance

    # needed to avoid going below lowest row or above highest row
    if best_r - nb < 0:
        rows = list(range(0, best_r+nb))
    elif best_r + nb > nr - 1:
        rows = list(range(best_r-nb, nr))
    else:
        rows = list(range(best_r-nb, best_r+nb))

    # needed to avoid going below lowest col or above highest col
    if best_c - nb < 0:
        cols = list(range(0, best_c+nb))
    elif best_c + nb > nc -1:
        cols = list(range(best_c-nb, nc))
    else:
        cols = list(range(best_c-nb, best_c+nb))

    # update the eligible neighborhood of the map
    eligible = np.ix_(rows, cols, list(range(3)))
    w_eligible = w[eligible]
    w_eligible += (lr * (rand_in - w_eligible))
    w[eligible] = w_eligible
```

In [16]:

```
plt.imshow(w)  # view the trained map
```

Out[16]:

```
<matplotlib.image.AxesImage at 0x7fa799eb0d68>
```



## IRIS Datset

In [17]:

```
data = np.genfromtxt('iris_data.csv', delimiter=',') # load in iris dataset
print(data.shape) # print the shape
```

```
(150, 4)
```

In [18]:

```
n_in = data.shape[1]  # define length of input vector
n_out = 10  # define number of nodes in map
w = np.random.rand(n_out, n_in)  # construct weight matrix
nb = 1  # define neighborhood size
lr = .1  # define learning rate
n_iters = 5000 # define the number of training iterations
```

In [19]:

```
for i in range(n_iters):
    # get a random example from the dataset
    rand_in = data[np.random.randint(0, data.shape[0], 1)[0], :]

    # get distances between weights and input and find the lowest distance node
    dist = np.sum(np.absolute(w - rand_in), 1)
    best = np.argmin(dist)

    # used to avoid going over the edge of the map
    if best - nb < 0:
        row = list(range(0, best+nb))
    elif best + nb > w.shape[0] - 1:
        row = list(range(best-nb, w.shape[0]))
    else:
        row = list(range(best-nb, best+nb))

    # update the neighborhood
    w_eligible = w[row, :]
    w_eligible += (lr * (rand_in - w_eligible))
    w[row, :] = w_eligible
```

In [22]:

```
energy = []

for i in range(w.shape[0]-1, -1, -1):
    energy.append(np.sqrt(np.sum(w[i, :] ** 2)))
```

In [23]:

```
fig = plt.figure()
subplot = fig.add_subplot(111)

subplot.set_ylabel("L2-Norm of Each Node's Weights")
subplot.axes.get_xaxis().set_visible(False)

plt.scatter([1]*len(distances), energy)
```

Out[23]:

```
<matplotlib.collections.PathCollection at 0x7fa799de59e8>
```


In [ ]: