In [3]:

```
from IPython.display import clear_output
import gym as g
import random as rnd
import numpy as np
import time
from sklearn.preprocessing import KBinsDiscretizer
import math
import matplotlib.pyplot as plt
```

In [12]:

```
env = g.make("CartPole-v1").env
```

In [13]:

```
print(env.action_space)
print(env.observation_space)
print(env.observation_space.high)
print(env.observation_space.low)
```

```
Discrete(2)
Box([-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38], [4.8000002e+00 3.
4028235e+38 4.1887903e-01 3.4028235e+38], (4,), float32)
[4.8000002e+00 3.4028235e+38 4.1887903e-01 3.4028235e+38]
[-4.8000002e+00 -3.4028235e+38 -4.1887903e-01 -3.4028235e+38]
```

In [14]:

```
# Funtion to run the episode using passed parameters.
def run_episode(env, parameters):
    observation = env.reset()
    totalreward = 0

    for _ in range(200):
        action = 0 if np.matmul(parameters,observation) < 0 else 1
        observation, reward, done, info = env.step(action)
        totalreward += reward

        if done:
            break

    return totalreward
```

In [15]:

```
# Function to train the agent.
def train(env, useRandom):

    counter = 0
    bestparams = None
    bestreward = 0
    reward = 0

    for i in range(300):
        counter += 1
        parameters = np.random.rand(4) * 2 - 1

        if useRandom == True:
            # Attempt to reach 200 steps using completely random actions.
            reward = rnd_episode(env)

        else:
            # Attempt to reach 200 steps using Q-Learning.
            reward = run_episode(env,parameters)

        clear_output(wait=True)
        print(f"episode: {_+1} / {training_size}, Rewards: {reward}")

        if reward > bestreward:
            bestreward = reward
            bestparams = parameters

            if reward == 200:
                bestWeights.append(parameters)

                break

    return counter
```

In [16]:

```
training_size = 100
useRandom = False

results = []
bestWeights = [] #stores weights achieving 200 steps

for _ in range(training_size):
    trainResults = train(env, useRandom)
    results.append(trainResults)
```

```
episode: 100 / 100, Rewards: 200.0
```

In [17]:

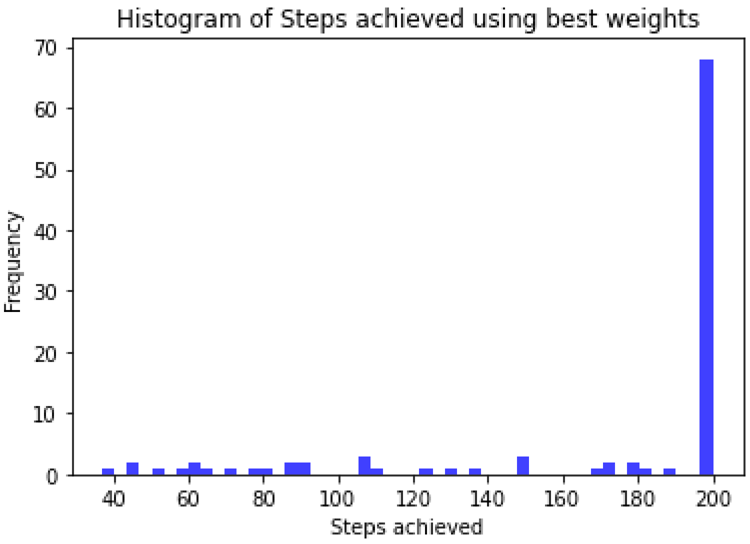
```
# Run again using bestWeights value
rewardsReturned = []

print("Testing best weights again..")
for i in range(len(bestWeights)):
    rewardsReturned.append(run_episode(env, bestWeights[i]))

print(rewardsReturned)

# Plot graph of results given using the best found weights
plt.hist(rewardsReturned,50, facecolor='b', alpha=0.75)
plt.xlabel('Steps achieved')
plt.ylabel('Frequency')
plt.title('Histogram of Steps achieved using best weights')
plt.show()
```

```
Testing best weights again..
[172.0, 200.0, 200.0, 72.0, 200.0, 108.0, 200.0, 200.0, 200.0, 200.0, 200.0, 200.0,
200.0, 58.0, 200.0, 200.0, 200.0, 200.0, 200.0, 200.0, 200.0, 200.0, 200.0, 86.0, 20
0.0, 200.0, 200.0, 200.0, 200.0, 111.0, 200.0, 200.0, 178.0, 200.0, 200.0, 200.0, 8
2.0, 200.0, 200.0, 200.0, 131.0, 200.0, 200.0, 45.0, 200.0, 200.0, 200.0, 86.0, 150.
0, 45.0, 107.0, 52.0, 200.0, 200.0, 65.0, 200.0, 200.0, 200.0, 200.0, 200.0, 179.0,
62.0, 200.0, 151.0, 200.0, 200.0, 200.0, 200.0, 200.0, 187.0, 92.0, 200.0, 200.0, 16
8.0, 200.0, 200.0, 37.0, 200.0, 200.0, 151.0, 200.0, 136.0, 200.0, 200.0, 79.0, 200.
0, 182.0, 122.0, 92.0, 173.0, 200.0, 200.0, 63.0, 200.0, 106.0, 200.0, 200.0, 200.0,
200.0, 200.0]
```



In [ ]:

In [ ]: