```
import numpy as np # linear algebra
import pandas as pd # data processing, CSV file I/O (e.g. pd.read_csv)
import gym # for environment
from collections import deque
from keras.models import Sequential
from keras.layers import Dense
from keras.optimizers import Adam # adaptive momentum
import random

class DQNAgent:
    def __init__(self, env):

        self.state_size = env.observation_space.shape[0]
        self.action_size = env.action_space.n

        self.gamma = 0.95
        self.learning_rate = 0.001

        self.epsilon = 1
        self.epsilon_min = 0.01
        self.epsilon_decay = 0.995
        self.memory = deque(maxlen = 1000)

        self.model = self.build_model()

    def build_model(self):
        model = Sequential()
        model.add(Dense(48, input_dim = self.state_size, activation='tanh'))
        model.add(Dense(self.action_size, activation='linear'))
        model.compile(loss = 'mse', optimizer = Adam(learning_rate = self.learning_rate))
        return model

    def remember(self, state, action, reward, next_state, done):
        #storage
        self.memory.append((state, action, reward, next_state, done))

    def act(self, state):
        if random.uniform(0,1) <= self.epsilon:
            return env.action_space.sample()

        else:
            predicted_outputs = self.model.predict(state)
            return np.argmax(predicted_outputs[0])

    def replay(self, batch_size):

        if len(self.memory) < batch_size:
            return

        minibatch = random.sample(self.memory, batch_size)


        for state, action, reward, next_state, done in minibatch:
            if done:
                target = reward

            else:
                target = reward + self.gamma * np.amax(self.model.predict(next_state)[0])

            train_target = self.model.predict(state) # s --> NN --> Q(s,a)=train_target
            train_target[0][action] = target
            self.model.fit(state, train_target, verbose = 0)

    def adaptiveEGreedy(self):
        if self.epsilon > self.epsilon_min:
            self.epsilon *= self.epsilon_decay
```

```
env = gym.make('CartPole-v1', render_mode="human")
agent = DQNAgent(env)

batch_size = 16
episodes = 50
for e in range(episodes):
    # initialize environment
    state = env.reset()
    state = np.array(state)
    print("Before reshaping, state shape:", state.shape)
    state = np.reshape(state, [1, agent.state_size])

    time = 0 # each second I will get reward, because I want to sustain a balance forever
    while True:
        # act
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)[:4]
        next_state = np.reshape(next_state, [1, agent.state_size])

        # remember / storage
        agent.remember(state, action, reward, next_state, done)

        # update state
        state = next_state

        # replay
        agent.replay(batch_size)

        # adjust epsilon
        agent.adaptiveEGreedy()

        time += 1

        if done:
            print('episode: {}, time: {}'.format(e, time))
            break
```