```
import gym
import numpy as np
import tensorflow as tf
from tensorflow.keras import layers

# Define the Checkers Environment
class CheckersEnv(gym.Env):
    def __init__(self):
        # Define your environment here
        pass

    def step(self, action):
        # Execute one time step within the environment
        pass

    def reset(self):
        # Reset the state of the environment to an initial state
        pass

    def render(self, mode='human'):
        # Render the environment to the screen
        pass

# Define the RL Agent
class DQNAgent:
    def __init__(self, state_size, action_size):
        self.state_size = state_size
        self.action_size = action_size
        self.memory = []  # Experience Replay Memory
        self.gamma = 0.95  # Discount rate
        self.epsilon = 1.0  # Exploration rate
        self.epsilon_min = 0.01
```

```
    self.epsilon_decay = 0.995
    self.model = self._build_model()

def _build_model(self):
    model = tf.keras.Sequential([
        layers.Dense(24, input_shape=(self.state_size,), activation='relu'),
        layers.Dense(24, activation='relu'),
        layers.Dense(self.action_size, activation='linear')
    ])
    model.compile(loss='mse', optimizer=tf.keras.optimizers.Adam(lr=0.001))
    return model

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

def act(self, state):
    if np.random.rand() <= self.epsilon:
        return np.random.choice(self.action_size)
    act_values = self.model.predict(state)
    return np.argmax(act_values[0])

def replay(self, batch_size):
    minibatch = random.sample(self.memory, batch_size)
    for state, action, reward, next_state, done in minibatch:
        target = reward
        if not done:
            target = (reward + self.gamma * np.amax(self.model.predict(next_state)[0]))
        target_f = self.model.predict(state)
        target_f[0][action] = target
        self.model.fit(state, target_f, epochs=1, verbose=0)
    if self.epsilon > self.epsilon_min:
        self.epsilon *= self.epsilon_decay
```

```
# Initialize the environment and agent
env = CheckersEnv()
state_size = env.observation_space.shape[0]
action_size = env.action_space.n
agent = DQNAgent(state_size, action_size)

# Train the agent
EPISODES = 1000
for e in range(EPISODES):
    state = env.reset()
    state = np.reshape(state, [1, state_size])
    for time in range(500):
        # env.render()  # Uncomment if you want to render the environment
        action = agent.act(state)
        next_state, reward, done, _ = env.step(action)
        reward = reward if not done else -10
        next_state = np.reshape(next_state, [1, state_size])
        agent.remember(state, action, reward, next_state, done)
        state = next_state
        if done:
            print("episode: {}/{}, score: {}, e: {:.2}"
                .format(e, EPISODES, time, agent.epsilon))
            break
    if len(agent.memory) > batch_size:
        agent.replay(batch_size)
```