```
#!/usr/bin/env python3

import statistics
import os
import pickle
import glob
import sys

import gym
import matplotlib.pyplot as plt
from yacs.config import CfgNode as CN
import tensorflow.keras as keras

from agents import DQN, TargetDQN
from policies import epsilon_episode_decay, random_policy, epsilon_greedy_policy_generato
r, acrobot_epsilon_decay

### Hyperparameter options
# gamma = [.99, 1]
# n_units = [[16, 8], [32, 16], [40]]
# learning_rate = [.01, .001]
# target_freq = [25, 50]

### Experiments: 1000 episodes; epsilon decay 300; batch size 2000; learning delay 50;
# --Gamma--
# 1: DQN; gamma=.99; n_units=[16, 8]; learning_rate=.01
# 2: DQN; gamma=1; n_units=[16, 8]; learning_rate=.01
# 3: DQN; gamma=.98; n_units=[16, 8]; learning_rate=.01
# 4: DQN; gamma=.97; n_units=[16, 8]; learning_rate=.01
# 5: DQN; gamma=.96; n_units=[16, 8]; learning_rate=.01 <--
# 6: DQN; gamma=.95; n_units=[16, 8]; learning_rate=.01
# 7: DQN; gamma=.94; n_units=[16, 8]; learning_rate=.01
# 8: DQN; gamma=.93; n_units=[16, 8]; learning_rate=.01
# --> Pick best
# --Learning rate--
# 9: DQN; gamma=.96; n_units=[16, 8]; learning_rate=.001 <--
# --> Pick best
# --Network--
# 10: DQN; gamma=.96; n_units=[32, 16]; learning_rate=.001
# 11: DQN; gamma=.96; n_units=[40]; learning_rate=.001 <--
# --> Pick best
# Target update frequency
# 12: TargetDQN; gamma=.96; n_units=[40]; learning_rate=.001; target_freq=25 <--
# 13: TargetDQN; gamma=.96; n_units=[40]; learning_rate=.001; target_freq=50
# --> Pick best
# Batch size :::: TRASHED BC TARGET FREQ 50
# 14: TargetDQN; gamma=.96; n_units=[40]; learning_rate=.001; target_freq=25; batch size
64
# 15: TargetDQN; gamma=.96; n_units=[30]; learning_rate=.001; target_freq=25
# 16: TargetDQN; gamma=.96; n_units=[20]; learning_rate=.001; target_freq=25
# 17: TargetDQN; gamma=.96; n_units=[60]; learning_rate=.001; target_freq=25
# 18: Experiment 12 for 3000 episodes
# 19: TargetDQN; gamma=.99; n_units=[40]; learning_rate=.001; target_freq=25 <--
# 20: Experiment 19 for 3000 episodes
# 21: TargetDQN; gamma=1; n_units=[40]; learning_rate=.001; target_freq=25
# 22: TargetDQN; gamma=.99; n_units=[40]; learning_rate=.001; target_freq=25; Epsilon dec
ay to .01 <--
# 23: TargetDQN; gamma=1; n_units=[40]; learning_rate=.001; target_freq=25; Epsilon decay
 to .01
# 24: TargetDQN; gamma=1; n_units=[40]; learning_rate=.001; target_freq=50; Epsilon decay
 to .01
# 25: TargetDQN; gamma=.99; n_units=[40]; learning_rate=.001; target_freq=50; Epsilon dec
ay to .01
# 26: Experiment 22 with L2 .1
# 27: Experiment 22 with L2 .01
# 28: Experiment 22 with L2 .001
# 29: Experiment 22 with L2 .0001
# Final choice: Experiment 22. Before final test, might need to remove the regularizer st
uff
```

```
# Figure 1: 5 independent runs with experiment 22
# Figure 2: 5 independent runs with experiment 22 modified with learning rate of .01

def save_results_and_models(agent, agent_folder, trial_name):
    fbase = "{}/".format(agent_folder)
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    fbase = "{}/".format(fbase + trial_name)
    if not os.path.exists(fbase):
        os.mkdir(fbase)

    results = {}
    results["rewards"] = agent.reward_log
    results["losses"] = agent.loss_log
    print("Reward log length: {}".format(len(results["rewards"])))
    print("Loss log length: {}".format(len(results["losses"])))

    # Save full results binary
    with open("{}results_dict.pkl".format(fbase), "wb") as f:
        pickle.dump(results, f)

    if agent.type == "DQN":
        agent.model.save("{}model.h5".format(fbase))
    elif agent.type == "TargetDQN":
        agent.model.save("{}model.h5".format(fbase))
        agent.target_model.save("{}target_model.h5".format(fbase))

def main():

    agent_folder = sys.argv[1]
    trial_name = sys.argv[2]

    keras.backend.clear_session()

    # Create environment
    env = gym.make('Acrobot-v1')
    print("State space: {}".format(env.observation_space))
    print("Action space: {}".format(env.action_space))

    # Create agent configuration
    agent_class = DQN
    state_size = env.observation_space.shape[0]
    action_size = env.action_space.n
    policy = epsilon_greedy_policy_generator(-1, 2)
    loss_fn = keras.losses.mean_squared_error
    epsilon = epsilon_episode_decay(1, .1, 300)
    gamma = .99
    buffer_size = 10000
    n_units = [16, 8]
    l2 = 0
    learning_rate = .01
    learning_delay = 50
    learning_freq = 1
    verbose = True
    target_update_freq = 25

    # Create silent episode configuration
    silent_episodes = CN()
    silent_episodes.n_episodes = 1000
    silent_episodes.n_steps = 500
    silent_episodes.render_flag = False
    silent_episodes.batch_size = 2000
    silent_episodes.verbose = True

    # Create visible episodes configuration
    visible_episodes = CN()
    visible_episodes.n_episodes = 1
    visible_episodes.n_steps = 500
    visible_episodes.render_flag = False
    visible_episodes.batch_size = 2000
    visible_episodes.verbose = True

    # Build agent
    agent = agent_class(
        state_size=state_size,
        action_size=action_size,
        policy=policy,
        loss_fn=loss_fn,
        epsilon=epsilon,
        gamma=gamma,
        buffer_size=buffer_size,
        n_units=n_units,
        l2=l2,
        learning_rate=learning_rate,
        learning_delay=learning_delay,
        learning_freq=learning_freq,
        verbose=verbose,
        target_update_freq=target_update_freq
        )

    print("--Training--")
    print("\tAgent type: {}".format(agent.type))

    # Run silent episodes
    agent.execute_episodes(
        env=env,
        n_episodes=silent_episodes.n_episodes,
        n_steps=silent_episodes.n_steps,
        render_flag=silent_episodes.render_flag,
        batch_size=silent_episodes.batch_size,
        verbose=silent_episodes.verbose
        )

    # Run visible episodes
    agent.execute_episodes(
        env=env,
        n_episodes=visible_episodes.n_episodes,
        n_steps=visible_episodes.n_steps,
        render_flag=visible_episodes.render_flag,
        batch_size=visible_episodes.batch_size,
        verbose=visible_episodes.verbose,
        train=False
        )

    save_results_and_models(agent, agent_folder, trial_name)

if __name__ == "__main__":
    main()
```