```
# -*- coding: utf-8 -*-
import pandas as pd
import numpy  as np
import argparse
import sys
import os

if __name__ == '__main__':

    parser = argparse.ArgumentParser()
    parser.add_argument("--random_seed"  , help = "type random seed [987, 799, 100]"  ,
default = 0        , type=int  )
    parser.add_argument("--base_location", help = "project location on rni files"     ,
default = "./"     , type=str  )
    parser.add_argument("--experiment"   , help = "folder name specific to experiment",
default = "Outputs", type=str  )
    parser.add_argument("--agent_type"   , help = "van, cer, gen, per, her, cbe, ..." ,
default = "van"    , type=str  )
    parser.add_argument("--env_type"     , help = "pos, neg"                           ,
default = "pos"    , type=str  )
    parser.add_argument("--env_grid_size", help = "5, 8, ..."                        ,
default = 5        , type=int  )
    parser.add_argument("--epsilon_decay", help = "0.999, 0.9999"                    ,
default = 0.995    , type=float)
    parser.add_argument("--episodes"     , help = "1000"                             ,
default = 1000     , type=int  )
    parser.add_argument("--initial_path" , help = "True, False"                      ,
default = False    , type=bool )
    parser.add_argument("--subgoals"     , help = "0, 1, 2"                          ,
default = 1        , type=int  )
    parser.add_argument("--env_stoch"    , help = "0, 0.1"                           ,
default = 0.0      , type=float)
    args = parser.parse_args()

    '''
    Set parameters of the run
    '''
    episodes =   int(args.episodes)
    size     =   int(args.env_grid_size)
    seed     =   int(args.random_seed)
    subgoals =   int(args.subgoals)
    dtype    =   args.agent_type
    succ_rew =   10
    if args.env_type == "1d":
        timeout = int(5*size)                          # 5 times the number of bits that
need to be flipped
    else:
        timeout =   int(10*(size-1)*2*max(1,subgoals)) # 10 times the shortest path from
start to goal state
    memory   =   int(100*timeout)                      # Between 100 and 1000 episodes,
depending on how fast you reach the goal
    tsteprew =  -1/float(timeout)                      # Penalty per time step: Total of
-1 if you reach timeout
    mode     =  "Train"
    folder   =   os.path.join(args.base_location, args.experiment,
                              args.env_type+"-sub-"+str(subgoals)+"-size-"+str(size)+\
                                  "-eps-"+str(int(1000*args.epsilon_decay))+\
                                      "-stoch-"+str(int(100*args.env_stoch)))
    '''
    Load environment and agent, and confirm parameter settings are consistent
    '''
    eset = False
    if   args.env_type == "pos":
        from  env.env2dnav import Env2D
        assert (0 <= subgoals <= 2)
        eset = True
```

```
elif args.env_type == "neg":
    from  env.env2dneg import Env2D
    assert (subgoals == 2)
    eset = True
elif args.env_type == "1d":
    from  env.env1d    import Env1D
    assert (0 <= subgoals <= 1)
    eset = True
assert eset
if   args.agent_type == "van":
    from agent.dqn_van       import Agent
elif args.agent_type == "cer":
    from agent.dqn_cer       import Agent
elif args.agent_type == "per":
    from agent.dqn_per       import Agent
elif args.agent_type == "her":
    from agent.dqn_her       import Agent
elif args.agent_type == "cbe":
    from agent.dqn_cbe       import Agent
elif args.agent_type == "gen_fixed":
    from agent.dqn_gen_fixed import Agent
elif args.agent_type == "gen_05_00":
    from agent.dqn_gen_05_00 import Agent
elif args.agent_type == "gen_05_05":
    from agent.dqn_gen_05_05 import Agent
elif args.agent_type == "gen_10_05":
    from agent.dqn_gen_10_05 import Agent
elif args.agent_type == "gen_active":
    from agent.dqn_gen_active import Agent
elif args.agent_type == "cem_rl":
    from agent.dqn_cem_rl import Agent
else:
    print("Agent type not found")
    sys.exit()

if not os.path.exists(args.base_location):
    os.makedirs(args.base_location)

if not os.path.exists(os.path.join(args.base_location, args.experiment)):
    os.makedirs(os.path.join(args.base_location, args.experiment))

if not os.path.exists(folder):
    os.makedirs(folder)

if not os.path.exists(os.path.join(folder,"models")):
    os.makedirs(os.path.join(folder,"models"))

if args.initial_path:
    print("path created")
    logfile    = open(folder + "/logfile.log", 'w')
    logfile.close()
    print("logfile initialised")
    sys.exit()

logfile    = open(folder + "/logfile.log", 'a')
old_stdout = sys.stdout
sys.stdout = logfile

'''
Initialise class variables
'''
if args.env_type == "pos" or args.env_type == "neg":
    env      = Env2D(size, succ_rew, tsteprew, subgoals, seed, args.env_stoch)
    isize    = len(env.get_state())
    if dtype != "cbe":
        ag   = Agent(isize, mode, seed, args.epsilon_decay, folder, episodes, memory)
```

```
        else:
            ag   = Agent(isize, mode, seed, args.epsilon_decay, folder, episodes, memory,
size)
    else:
        env      = Env1D(size, succ_rew, tsteprew, subgoals, seed, args.env_stoch)
        isize    = len(env.get_state())
        if dtype != "cbe":
            ag   = Agent(isize, mode, seed, args.epsilon_decay, folder, episodes, memory,
size)
        else:
            ag   = Agent(isize, mode, seed, args.epsilon_decay, folder, episodes, memory,
size, size)

    results  = np.zeros((episodes, 10))
    results  =
pd.DataFrame(results,columns=["Episode","Exploration","PolicyNum","PolicyCountdown",

"SpawnReason","Length","TermReward","InternalReward",
                                            "TotalReward","Loss"])
    '''
    Run the loop
    '''
    for ep in range(episodes):
        env.reset()
        ag.reset()
        results["Episode"][ep]     = ep+1
        results["Exploration"][ep] = ag.exploration
        done   = False
        t      = 0
        reward = 0
        try:
            while not done:
                state  = env.get_state()
                action = ag.choose_action(state,reward,t)
                done, reward = env.step(action)
                t += 1
                if t == timeout:
                    done = True
            sys.stdout.flush()
            ag.terminate(reward, ep)
            results["Length"][ep]          = t
            results["TermReward"][ep]      = reward
            results["TotalReward"][ep]     = ag.tot_rew
            results["Loss"][ep]            = ag.loss
            results["SpawnReason"][ep]     = ag.spawn_by
            results["PolicyNum"][ep]       = ag.pol
            results["PolicyCountdown"][ep] = ag.cdown
            results["InternalReward"][ep]  = sum(ag.ep_reward)
        except:
            print("Episode run error, " + dtype + ", seed " + str(seed) + ", episode" +
str(ep+1) + ", done = " + str(done))
        if (ep+1)%250 == 0:
            results.to_csv(folder + "/results-" + dtype + "-" + str(seed) + ".csv",
index=False)
    results.to_csv(folder + "/results-" + dtype + "-" + str(seed) + ".csv", index=False)

    sys.stdout = old_stdout
    sys.stdout.flush()
    logfile.flush()
    logfile.close()
```