MVA - Homework 2 - Reinforcement Learning (2022/2023)

Name: DELLIAUX Thomas

Email:thomas.delliaux59@gmail.com

Instructions

- The deadline is **December 16 (2022) at 11:59 pm (Paris time).**
- By doing this homework you agree to the late day policy, collaboration and misconduct rules reported on Piazza (https://piazza.com/class/l4y5ubadwj64mb?cid=6).
- **Mysterious or unsupported answers will not receive full credit**. A correct answer, unsupported by calculations, explanation, or algebraic work will receive no credit; an incorrect answer supported by substantially correct calculations and explanations might still receive partial credit.
- Answers should be provided in **English**.

Colab setup

In [33]:
```
! jupyter nbconvert --to html /content/drive/MyDrive/MVA/S1/RL/HW2.ipynb
```

```
[NbConvertApp] Converting notebook /content/drive/MyDrive/MVA/S1/RL/HW2.ipynb to html
[NbConvertApp] Writing 1152839 bytes to /content/drive/MyDrive/MVA/S1/RL/HW2.html
```

In [1]:
```
from IPython import get_ipython

if 'google.colab' in str(get_ipython()):
  # install rlberry library
  !pip install git+https://github.com/rlberry-py/rlberry.git@mva2021#egg=rlberry[default] > /dev/null 2>&1
  !pip install gym==0.22 > /dev/null 2>&1
  !pip install pyglet==1.3.2 > /dev/null 2>&1

  # pytorch
  !pip install torch > /dev/null 2>&1

  # install ffmpeg-python for saving videos
  !pip install ffmpeg-python > /dev/null 2>&1

  # packages required to show video
  !pip install pyvirtualdisplay > /dev/null 2>&1
  !apt-get install -y xvfb python-opengl ffmpeg > /dev/null 2>&1

  print("Libraries installed, please restart the runtime!")
```

```
Libraries installed, please restart the runtime!
```

In [2]:
```
# Create directory for saving videos
!mkdir videos > /dev/null 2>&1

# Initialize display and import function to show videos
import rlberry.colab_utils.display_setup
from rlberry.colab_utils.display_setup import show_video
```

In [3]:
```
# Useful imports
import gym
import torch
import numpy as np
import matplotlib.pyplot as plt
from copy import deepcopy
from gym.wrappers import Monitor
from rlberry.agents import Agent
from rlberry.manager import AgentManager, plot_writer_data

import torch.nn as nn
import torch.optim as optim
from torch.distributions import Categorical

# torch device
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
```

```
[INFO] OpenGL_accelerate module loaded
[INFO] Using accelerated ArrayDatatype
[INFO] Generating grammar tables from /usr/lib/python3.8/lib2to3/Grammar.txt
[INFO] Generating grammar tables from /usr/lib/python3.8/lib2to3/PatternGrammar.txt
[INFO] NumExpr defaulting to 2 threads.
```

Preparation

Environment

In the coding exercises, you will use the CartPole environment from Gym (https://gym.openai.com/) library. The cells below show how to interact with this MDP and how to visualize it.

In [4]:
```
def get_env():
  """Creates an instance of a CartPole-v0 environment."""
  return gym.make('CartPole-v0')

def render_policy(env, agent):
  env = deepcopy(env)
  env = Monitor(env, './videos', force=True, video_callable=lambda episode: True)
  for episode in range(1):
    done = False
    state = env.reset()
    env.render()
    while not done:
        action = agent.select_action(state, evaluation=True)
        state, reward, done, info = env.step(action)
        env.render()
    env.close()
    show_video()
```

Running experiments with `rlberry`

In order to compare different algorithms, you'll need to run experiments several times and plot the results. For that, we use the rlberry (https://github.com/rlberry-py/rlberry) library here. The code below gives a quick introduction to what you will need from this library, but you can also check its documentation here (https://rlberry.readthedocs.io/en/latest/).

Basically, all you have to do is to write your agents using the `rlberry.agents.Agent` interface, and `rlberry` provides an `AgentManger` class that allows you to run your agents in parallel and plot the results.

In [5]:

```python
from rlberry.agents import Agent

class MyAgent(Agent):
  name = "MyAgent"
  def __init__(self, env, param1, param2, **kwargs):
    """
    The base class (Agent) initializes:
      self.env : instance of the environment used for training (in fit() method)
      self.eval_env : instance of the environment used for evaluation (in eval() meth
od)
      self.rng : random number generator (https://numpy.org/doc/stable/reference/rand
om/generator.html)
      self.writer : use self.writer.add_scalar(tag, value, global_step) to log traini
ng data

    For reproducibility, use ONLY self.rng if you need random numbers in you agent!
    To be able to visualize plots with AgentManager, log data using self.writer (see
below)
    """
    Agent.__init__(self, env, **kwargs)
    self.param1 = param1
    self.param2 = param2
    self.total_steps = 0
    self.total_episodes = 0

  def select_action(self, state, evaluation=False):
    """
    If evaluation=True, run evaluation policy (e.g., greedy with respect to Q)
    If evaluation=False, run exploration policy (e.g., epsilon greedy)
    """
    return self.env.action_space.sample()  # random action for this example

  def fit(self, budget):
    """budget = number of timesteps to train your agent"""
    state = self.env.reset()
    episode_reward = 0.0
    for tt in range(budget):
      self.total_steps += 1
      action = self.select_action(state, evaluation=False)
      next_state, reward, done, _ = self.env.step(action)
      episode_reward += reward

      # Log data
      self.writer.add_scalar('rewards', reward, global_step=self.total_steps)

      state = next_state
      if done:
        self.total_episodes += 1
        # Log episode data
        self.writer.add_scalar('episode_rewards', episode_reward, global_step=self.to
tal_steps)
        self.writer.add_scalar('episode', self.total_episodes, global_step=self.total
_steps)

        state = self.env.reset()
        episode_reward = 0.0


  def eval(self, **kwargs):
    """
    Here, you can run Monte-Carlo policy evaluation
    with self.eval_env and return the result.
    Returning zero for this example.
    """
    return 0.0
```

In [6]:

```
#
# Initialize and train a single instance of MyAgent
#
my_agent = MyAgent(
    env=(get_env, {}),       # tuple (constructor, kwargs)
    param1=10,               # extra params your agent might need
    param2=15)
# train the agent for 100 timesteps
my_agent.fit(100)
# pandas DataFrame containing data stored with my_agent.writer.add_scalar(tag, value,
global_step)
print(my_agent.writer.data)

#
# Run several instances of MyAgent in parallel and plot the results
#
manager_kwargs = dict(
    agent_class=MyAgent,
    train_env=(get_env, dict()),
    eval_env=(get_env, dict()),
    fit_budget=100,                    # Number of total timesteps
    n_fit=2,                           # Number of agent instances to fit
    parallelization='thread',          # Use 'thread' in the notebook!
    seed=456,                          # Seed
    default_writer_kwargs=dict(maxlen=None,log_interval=10),
)
my_agent_manager = AgentManager(
    init_kwargs=dict(param1=10, param2=20),
    agent_name='MyAgent',
    **manager_kwargs
)
my_agent_manager.fit()   # Train 'n_fit' instances in parallel
```

```
INFO: Making new env: CartPole-v0
/usr/local/lib/python3.8/dist-packages/gym/envs/registration.py:505: UserWarning: WAR
N: The environment CartPole-v0 is out of date. You should consider upgrading to versi
on `v1` with the environment ID `CartPole-v1`.
  logger.warn(
INFO: Making new env: CartPole-v0
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for MyAgent...
INFO: Making new env: CartPole-v0
/usr/local/lib/python3.8/dist-packages/gym/envs/registration.py:505: UserWarning: WAR
N: The environment CartPole-v0 is out of date. You should consider upgrading to versi
on `v1` with the environment ID `CartPole-v1`.
  logger.warn(
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

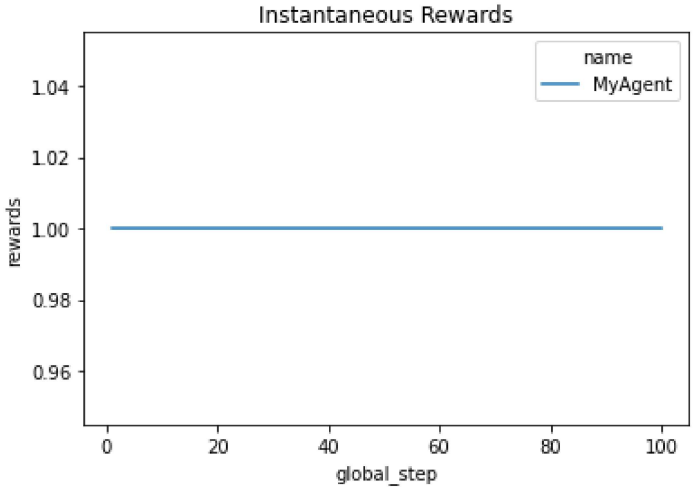
	name	tag	value	global_step
0	MyAgent	rewards	1.0	1
1	MyAgent	rewards	1.0	2
2	MyAgent	rewards	1.0	3
3	MyAgent	rewards	1.0	4
4	MyAgent	rewards	1.0	5
..	...	...	...	...
211	MyAgent	episode_rewards	13.0	78
212	MyAgent	episode	1.0	18
213	MyAgent	episode	2.0	55
214	MyAgent	episode	3.0	65
215	MyAgent	episode	4.0	78

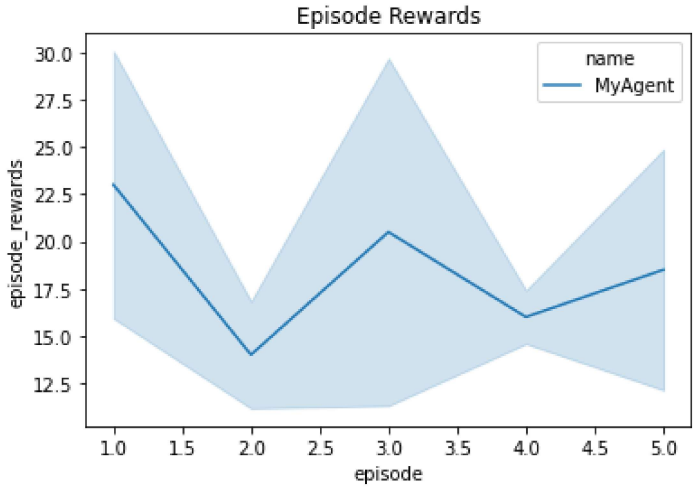
[216 rows x 4 columns]

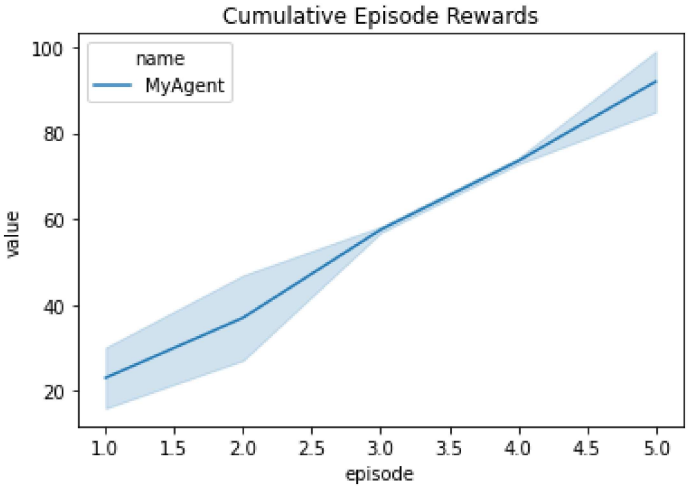
```
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

In [7]:

```
# Plot the results
example_managers = []
example_managers.append(my_agent_manager)    # You could add more managers here, for
other agents/parameters
_ = plot_writer_data(example_managers, tag='rewards', title='Instantaneous Rewards')
_ = plot_writer_data(example_managers, tag='episode_rewards', xtag='episode', title
='Episode Rewards')
_ = plot_writer_data(example_managers, tag='episode_rewards', xtag='episode', title
='Cumulative Episode Rewards', preprocess_func=np.cumsum)

# Render the policy of one of the trained agents
agent_instance = my_agent_manager.get_agent_instances()[0]
render_policy(agent_instance.eval_env, agent_instance)
```








```
INFO: Clearing 4 monitor files from previous run (because force=True was provided)
INFO: Starting new video recorder writing to /content/videos/openaigym.video.0.2436.v
ideo000000.mp4
INFO: Finished writing results. You can upload them to the scoreboard via gym.upload
('/content/videos')
```

0:00 / 0:00

In [8]:
```
get_env().action_space.n
```

```
INFO: Making new env: CartPole-v0
/usr/local/lib/python3.8/dist-packages/gym/envs/registration.py:505: UserWarning: WAR
N: The environment CartPole-v0 is out of date. You should consider upgrading to versi
on `v1` with the environment ID `CartPole-v1`.
  logger.warn(
```

Out[8]: 2

Part 1: DQN

The goal of this exercise is to compare different variants of Deep Q-Learning (a.k.a. DQN).

DQN. Recall from the class the DQN aims to minimize the following loss function

$$L(\theta) = \mathbb{E}_{s,a,r,s' \sim \mathcal{D}} \left[\left(r + \gamma \max_{a' \in \mathcal{A}} Q(s', a'; \theta') - Q(s, a; \theta) \right)^2 \right]$$

where θ' is the parameter of the target network updated every C iterations from θ.

Question 1.1 (written)

(a) The DQN objective resembles a classical supervised learning problem. Could you highlight the differences?

(b) Could you describe the role of C and the trade-off at play in choosing a good value for C?

(a) The first difference is that we don't have a dataset at the begining, we have to collect the data with exploration. The second difference is that the target will move during the training.

(b) If C is too small, the target will change too fast. So our NN will not be able to converge toward the target. If C is too high, we will converge towards the wrong "Q-function", that is, not towards the Q-function which is the fixed point of Bellman operator.

Question 2.2 (implementation)

We would like to evaluate DQN with newer variants of DQN, namely Double DQN, Dueling DQN and Double Dueling DQN. In class we have seen the principle of DQN, please refer to the following paper for Double and Dueling DQN (https://arxiv.org/abs/1511.06581 (https://arxiv.org/abs/1511.06581)).

The difference between DQN and Double DNQ is how the target network is built. In Double DQN we use the greedy action suggested by $Q(s, a; \boldsymbol{\theta})$ to evaluate the target (i.e., θ'), see the appendix of https://arxiv.org/abs/1511.06581 (https://arxiv.org/abs/1511.06581).

In Dueling DQN, the Q function is estimated as

$$Q(s, a) = V(s) + A(s, a) - \frac{1}{|\mathcal{A}|} \sum_{a'} A(s, a')$$

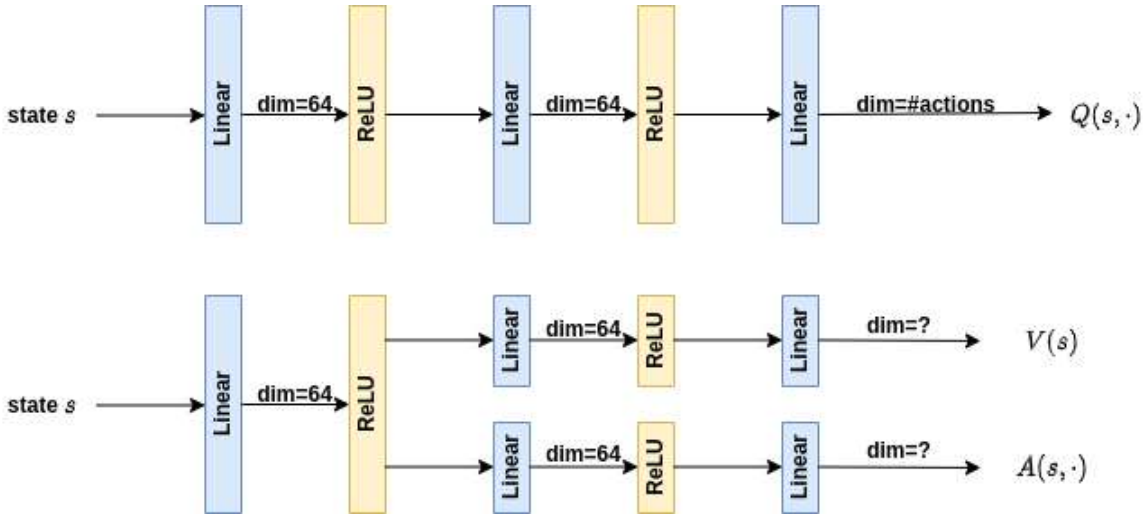
V and A share parameters. Dueling DQN can be implemented "standard" or "double".

Starting from the provided code:

- (code) Implement DQN and Double DQN. Use the network in the figure below **(top)**.
- (code) Implement Dueling DQN and Dueling Double DQN. See the figure below **(bottom)** for the network with shared parameters.
- Compare the performance of the algorithms.

More precisely, you'll have to:

- complete the code for the networks `QNet` and `DuelingQNet` ;
- compute the loss in the fit() method of `DQNAgent`
- define the exploration and evaluation policy in the `select_action` method of `DQNAgent`



In [9]:

```python
class ReplayBuffer:
  def __init__(self, capacity, rng):
    """
    Parameters
    ----------
    capacity : int
      Maximum number of transitions
    rng :
      instance of numpy's default_rng
    """
    self.capacity = capacity
    self.rng = rng  # random number generator
    self.memory = []
    self.position = 0

  def push(self, sample):
    """Saves a transition."""
    if len(self.memory) < self.capacity:
      self.memory.append(None)
    self.memory[self.position] = sample
    self.position = (self.position + 1) % self.capacity

  def sample(self, batch_size):
    indices = self.rng.choice(len(self.memory), size=batch_size)
    samples = [self.memory[idx] for idx in indices]
    return map(np.asarray, zip(*samples))

  def __len__(self):
    return len(self.memory)
```

In [10]:
```
class QNet(nn.Module):
  def __init__(self, obs_size, n_actions):
    super(QNet, self).__init__()
    self.fc1 = nn.Linear(obs_size, 64)
    self.ReLU = torch.relu
    self.fc2 = nn.Linear(64,64)
    self.fc3 = nn.Linear(64,n_actions)

  def forward(self, state):
    Q = self.fc1(state)
    Q = self.ReLU(Q)
    Q = self.fc2(Q)
    Q = self.ReLU(Q)
    Q = self.fc3(Q)
    return Q


class DuelingQNet(nn.Module):
  def __init__(self, obs_size, n_actions):
    super(DuelingQNet, self).__init__()
    self.fc1 = nn.Linear(obs_size, 64)
    self.ReLU = torch.relu
    self.fc2_val = nn.Linear(64,64)
    self.fc2_adv = nn.Linear(64,64)
    self.fc3_val = nn.Linear(64,1)
    self.fc3_adv = nn.Linear(64,n_actions)

  def forward(self, state):
    Q = self.fc1(state)
    Q = self.ReLU(Q)
    V = self.fc2_val(Q)
    A = self.fc2_adv(Q)
    V = self.fc3_val(V)
    A = self.fc3_adv(A)

    Q = V.expand(A.size()) + (A-A.mean().expand(A.size()))

    return Q
```

In [11]:
```
# Parameters
DQN_TRAINING_TIMESTEPS = 10000  # number of timesteps for training. You might change
this!

DQN_PARAMS = dict(
    dueling_dqn=False,          # set to true to use dueling DQN
    double_dqn=False,           # set to true to use double DQN
    gamma=0.99,
    batch_size=256,             # batch size (in number of transitions)
    eval_every=250,             # evaluate every ... steps
    buffer_capacity=30000,      # capacity of the replay buffer
    update_target_every=500,    # update target net every ... steps
    epsilon_start=1.0,          # initial value of epsilon
    epsilon_min=0.05,           # minimum value of epsilon
    decrease_epsilon=5000,      # parameter to decrease epsilon
    learning_rate=0.001,        # learning rate
)
DUELING_DQN_PARAMS = deepcopy(DQN_PARAMS)           # dueling DQN
DOUBLE_DQN_PARAMS = deepcopy(DQN_PARAMS)            # double DQN
DOUBLE_DUELING_DQN_PARAMS = deepcopy(DQN_PARAMS)    # double & dueling DQN

DUELING_DQN_PARAMS.update(dict(dueling_dqn=True))
DOUBLE_DQN_PARAMS.update(dict(double_dqn=True))
DOUBLE_DUELING_DQN_PARAMS.update(dueling_dqn=True, double_dqn=True)
```

In [12]:

```
class DQNAgent(Agent):
  name = 'DQN'
  def __init__(
      self,
      env,
      dueling_dqn: bool,                # Set to true for dueling DQN
      double_dqn: bool,                 # Set to true for double DQN
      gamma: float = 0.99,
      batch_size: int = 256,
      eval_every: int = 500,
      buffer_capacity: int = 30000,
      update_target_every: int = 500,
      epsilon_start: float = 1.0,
      epsilon_min: float = 0.05,
      decrease_epsilon: int = 200,
      learning_rate: float = 0.001,
      **kwargs):
    Agent.__init__(self, env, **kwargs)
    env = self.env
    self.dueling_dqn = dueling_dqn
    self.double_dqn = double_dqn
    self.gamma = gamma
    self.batch_size = batch_size
    self.eval_every = eval_every
    self.update_target_every = update_target_every
    self.epsilon_start = epsilon_start
    self.epsilon_min = epsilon_min
    self.decrease_epsilon = decrease_epsilon
    self.total_timesteps = 0
    self.total_episodes = 0
    self.total_updates = 0

    # initialize epsilon
    self.epsilon = epsilon_start

    # initialize replay buffer
    self.replay_buffer = ReplayBuffer(buffer_capacity, self.rng)

    # select network class
    if self.dueling_dqn:
      net_class = DuelingQNet
    else:
      net_class = QNet

    # update name according to params
    if self.dueling_dqn:
      self.name = 'Dueling' + self.name
    if self.double_dqn:
      self.name = 'Double' + self.name

    # create network and target network
    obs_size = env.observation_space.shape[0]
    n_actions = env.action_space.n
    self.q_net = net_class(obs_size, n_actions)
    self.target_net = net_class(obs_size, n_actions)
    self.target_net.load_state_dict(self.q_net.state_dict())
    self.target_net.eval()

    # objective and optimizer
    self.optimizer = optim.Adam(
        params=self.q_net.parameters(), lr=learning_rate)
    self.loss_fn = nn.MSELoss()

  def select_action(self, state, evaluation=False):
    """
    If evaluation=False, get action according to exploration policy.
    Otherwise, get action according to the evaluation policy.
```

```
    """
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    # TODO: implement action selection strategy
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    if self.rng.uniform() < self.epsilon and (not evaluation):
        action = np.random.randint(0,self.env.action_space.n)# this happens with prob
ability epsilon
    else:
      state = torch.from_numpy(state)
      action = torch.argmax(self.q_net(state)).item()
    return action # replace by the action you computed

  def fit(self, budget):
    """
    budget : number of training timesteps
    """
    state = self.env.reset()
    done = False
    episode_reward = 0.0
    for tt in range(budget):
      self.total_timesteps += 1
      action = self.select_action(state, evaluation=False)
      next_state, reward, done, _ = self.env.step(action)
      episode_reward += reward
      self.replay_buffer.push((state, next_state, action, reward, done))

      if len(self.replay_buffer) > self.batch_size:
        #
        # Update model
        #
        self.total_updates += 1

        # get batch
        (batch_state, batch_next_state,
         batch_action, batch_reward,
         batch_done) = self.replay_buffer.sample(self.batch_size)
        # convert to torch tensors
        batch_state = torch.FloatTensor(batch_state).to(device)
        batch_next_state = torch.FloatTensor(batch_next_state).to(device)
        batch_action = torch.LongTensor(batch_action).unsqueeze(1).to(device)
        batch_reward = torch.FloatTensor(batch_reward).unsqueeze(1).to(device)
        batch_done = torch.FloatTensor(batch_done).unsqueeze(1).to(device)

        mask_terminal = (1-batch_done)

        # decrease epsilon
        if self.epsilon > self.epsilon_min:
            self.epsilon -= (self.epsilon_start
                             - self.epsilon_min) / self.decrease_epsilon

        #
        #  TO DO: compute loss and update networks
        #
        if not self.double_dqn:
          # !!!!!!!!!!!!!!!!!!!!!!!!!
          # TO DO: compute DQN targets
          # !!!!!!!!!!!!!!!!!!!!!!!!!


          # values = Q(s_t, a_t), for t in batch
          values = self.q_net(batch_state).gather(1, batch_action.long())
          with torch.no_grad():
            Q_max_target = torch.max(self.target_net(batch_next_state),axis = 1)[0].u
nsqueeze(1)
            targets = batch_reward + torch.mul(self.gamma,Q_max_target)
            targets = targets*mask_terminal

        else:
```

```
          # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          # TO DO: compute Double DQN targets
          # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
          values = self.q_net(batch_state).gather(1, batch_action.long())
          with torch.no_grad():
            Q_argmax_value = torch.argmax(self.q_net(batch_next_state),axis=1).unsque
eze(1)
            targets = batch_reward + torch.mul(self.gamma,self.target_net(batch_next_
state).gather(1,Q_argmax_value))
            targets = targets*mask_terminal


        # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        # TO DO: compute loss and take gradient step
        # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        loss = torch.tensor(0.0)
        loss = self.loss_fn(values, targets)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()
        self.writer.add_scalar('loss', loss.item(), self.total_timesteps)

      # evaluate agent
      if self.total_timesteps % self.eval_every == 0:
          mean_rewards = self.eval(n_sim=5)
          self.writer.add_scalar(
              'eval_rewards', mean_rewards, self.total_timesteps)

      # update target network
      if self.total_updates % self.update_target_every == 0:
          self.target_net.load_state_dict(self.q_net.state_dict())
          self.target_net.eval()

      # check end of episode
      state = next_state
      if done:
        state = self.env.reset()
        self.total_episodes += 1
        self.writer.add_scalar(
                'episode_rewards', episode_reward, self.total_timesteps)
        self.writer.add_scalar(
                'episode', self.total_episodes, self.total_timesteps)
        episode_reward = 0.0

  def eval(self, n_sim=1, **kwargs):
    rewards = np.zeros(n_sim)
    eval_env = self.eval_env     # evaluation environment
    # Loop over number of simulations
    for sim in range(n_sim):
      state = eval_env.reset()
      done = False
      while not done:
        action = self.select_action(state, evaluation=True)
        next_state, reward, done, _ = eval_env.step(action)
        # update sum of rewards
        rewards[sim] += reward
        state = next_state
    return rewards.mean()
```

In [13]:

```
# # Training one instance of DQN
# dqn_agent = DQNAgent(
#     env=(get_env, dict()),  # we can send (constructor, kwargs) as an env
#     **DQN_PARAMS
# )
# dqn_agent.fit(DQN_TRAINING_TIMESTEPS)

#
# Training several instances using AgentManager
#
manager_kwargs = dict(
    agent_class=DQNAgent,
    train_env=(get_env, dict()),
    eval_env=(get_env, dict()),
    fit_budget=DQN_TRAINING_TIMESTEPS,
    n_fit=2,                    # NOTE: You may increase this parameter (number of age
nts to train)
    parallelization='thread',
    seed=456,
    default_writer_kwargs=dict(maxlen=None,log_interval=10),
)
```

In [14]:

```
# DQN
dqn_manager = AgentManager(
    init_kwargs=DQN_PARAMS,
    agent_name='DQN',
    **manager_kwargs
)
dqn_manager.fit()
```

```
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for DQN...
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
[INFO] [DQN[worker: 1]] | max_global_step = 1302 | episode_rewards = 16.0 | dw_time_e
lapsed = 10.005862207000064 | episode = 61 | eval_rewards = 15.0 | loss = 0.169133856
89258575 |
[INFO] [DQN[worker: 0]] | max_global_step = 1309 | episode_rewards = 11.0 | dw_time_e
lapsed = 10.005029861999901 | episode = 55 | eval_rewards = 11.6 | loss = 0.115003682
67297745 |
[INFO] [DQN[worker: 1]] | max_global_step = 2291 | episode_rewards = 11.0 | dw_time_e
lapsed = 20.00911070500024 | episode = 96 | eval_rewards = 179.6 | loss = 0.076796799
89814758 |
[INFO] [DQN[worker: 0]] | max_global_step = 2500 | episode_rewards = 45.0 | dw_time_e
lapsed = 20.470901520000552 | episode = 98 | eval_rewards = 144.8 | loss = 0.08432719
856500626 |
[INFO] [DQN[worker: 1]] | max_global_step = 3250 | episode_rewards = 33.0 | dw_time_e
lapsed = 30.031645852999645 | episode = 115 | eval_rewards = 189.2 | loss = 0.0417992
10011959076 |
[INFO] [DQN[worker: 0]] | max_global_step = 3549 | episode_rewards = 162.0 | dw_time_
elapsed = 30.48700776500027 | episode = 116 | eval_rewards = 177.4 | loss = 0.0460720
8073139191 |
[INFO] [DQN[worker: 1]] | max_global_step = 4500 | episode_rewards = 148.0 | dw_time_
elapsed = 40.25432298800024 | episode = 126 | eval_rewards = 191.6 | loss = 0.1339519
0238952637 |
[INFO] [DQN[worker: 0]] | max_global_step = 4797 | episode_rewards = 154.0 | dw_time_
elapsed = 40.48829508400013 | episode = 124 | eval_rewards = 166.6 | loss = 0.0595018
08136701584 |
[INFO] [DQN[worker: 0]] | max_global_step = 6000 | episode_rewards = 91.0 | dw_time_e
lapsed = 50.57298209500004 | episode = 133 | eval_rewards = 177.4 | loss = 0.55119621
75369263 |
[INFO] [DQN[worker: 1]] | max_global_step = 5750 | episode_rewards = 175.0 | dw_time_
elapsed = 50.65234710799996 | episode = 133 | eval_rewards = 186.4 | loss = 0.5187461
376190186 |
[INFO] [DQN[worker: 0]] | max_global_step = 7208 | episode_rewards = 178.0 | dw_time_
elapsed = 60.57984113800012 | episode = 140 | eval_rewards = 192.8 | loss = 0.6377927
06489563 |
[INFO] [DQN[worker: 1]] | max_global_step = 7000 | episode_rewards = 106.0 | dw_time_
elapsed = 60.93972701700022 | episode = 142 | eval_rewards = 156.4 | loss = 1.2930579
18548584 |
[INFO] [DQN[worker: 0]] | max_global_step = 8415 | episode_rewards = 200.0 | dw_time_
elapsed = 70.58467513400046 | episode = 147 | eval_rewards = 180.6 | loss = 0.8224681
01978302 |
[INFO] [DQN[worker: 1]] | max_global_step = 8250 | episode_rewards = 145.0 | dw_time_
elapsed = 71.14888452200012 | episode = 153 | eval_rewards = 117.4 | loss = 1.5538091
659545898 |
[INFO] [DQN[worker: 0]] | max_global_step = 9562 | episode_rewards = 200.0 | dw_time_
elapsed = 80.58489762299996 | episode = 153 | eval_rewards = 129.8 | loss = 1.3864306
211471558 |
[INFO] [DQN[worker: 1]] | max_global_step = 9497 | episode_rewards = 116.0 | dw_time_
elapsed = 81.15279630199984 | episode = 164 | eval_rewards = 105.4 | loss = 0.2159555
2563667297 |
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

In [15]:
```
# Double DQN
double_dqn_manager = AgentManager(
    init_kwargs=DOUBLE_DQN_PARAMS,
    agent_name='DoubleDQN',
    **manager_kwargs
)
double_dqn_manager.fit()
```

```
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for DoubleDQN...
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 1561 | episode_rewards = 29.0 | dw_
time_elapsed = 10.001297075000366 | episode = 61 | eval_rewards = 200.0 | loss = 0.03
531479090452194 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 1588 | episode_rewards = 15.0 | dw_
time_elapsed = 10.003514205999636 | episode = 73 | eval_rewards = 194.6 | loss = 0.11
30024865269661 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 2784 | episode_rewards = 43.0 | dw_
time_elapsed = 20.005542308999793 | episode = 104 | eval_rewards = 187.6 | loss = 0.1
1145141720771 79 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 2750 | episode_rewards = 54.0 | dw_
time_elapsed = 20.350198130000535 | episode = 87 | eval_rewards = 194.2 | loss = 0.06
431448459625244 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 3975 | episode_rewards = 42.0 | dw_
time_elapsed = 30.006292193000263 | episode = 117 | eval_rewards = 166.0 | loss = 0.0
804305151104927 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 3944 | episode_rewards = 197.0 | dw
_time_elapsed = 30.352987920000487 | episode = 97 | eval_rewards = 200.0 | loss = 0.2
9474759101867676 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 5110 | episode_rewards = 185.0 | dw
_time_elapsed = 40.01289975399959 | episode = 123 | eval_rewards = 170.8 | loss = 0.0
52753448486328125 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 5067 | episode_rewards = 161.0 | dw
_time_elapsed = 40.35519256900079 | episode = 103 | eval_rewards = 153.4 | loss = 0.5
7461613416671 75 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 6248 | episode_rewards = 200.0 | dw
_time_elapsed = 50.01653609099958 | episode = 129 | eval_rewards = 200.0 | loss = 0.0
96347071230411 53 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 6227 | episode_rewards = 171.0 | dw
_time_elapsed = 50.35964855600014 | episode = 110 | eval_rewards = 154.4 | loss = 0.3
9303445816040 04 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 7327 | episode_rewards = 196.0 | dw
_time_elapsed = 60.02129135300038 | episode = 135 | eval_rewards = 179.2 | loss = 0.7
0649588108062 74 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 7339 | episode_rewards = 198.0 | dw
_time_elapsed = 60.359823135000624 | episode = 115 | eval_rewards = 168.6 | loss = 0.
1678003966808319 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 8496 | episode_rewards = 157.0 | dw
_time_elapsed = 70.02202776800004 | episode = 142 | eval_rewards = 169.0 | loss = 0.3
0460131168365 48 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 8500 | episode_rewards = 188.0 | dw
_time_elapsed = 70.59099778900054 | episode = 122 | eval_rewards = 155.4 | loss = 0.2
4449001252651215 |
[INFO] [DoubleDQN[worker: 1]] | max_global_step = 9660 | episode_rewards = 149.0 | dw
_time_elapsed = 80.02649985500011 | episode = 150 | eval_rewards = 162.2 | loss = 0.1
3908353447914124 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 9687 | episode_rewards = 146.0 | dw
_time_elapsed = 80.59361045700007 | episode = 129 | eval_rewards = 144.8 | loss = 0.3
839982748031616 |
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

In [16]:
```
# Dueling DQN
dueling_dqn_manager = AgentManager(
    init_kwargs=DUELING_DQN_PARAMS,
    agent_name='DuelingDQN',
    **manager_kwargs
)
dueling_dqn_manager.fit()
```

```
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for DuelingDQN...
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 1425 | episode_rewards = 29.0 | dw
_time_elapsed = 10.005527737000193 | episode = 63 | eval_rewards = 58.8 | loss = 0.08
321225643157959 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 1482 | episode_rewards = 12.0 | dw
_time_elapsed = 10.004625216000022 | episode = 75 | eval_rewards = 21.8 | loss = 0.09
917256981134415 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 2446 | episode_rewards = 43.0 | dw
_time_elapsed = 20.00929226900007 | episode = 86 | eval_rewards = 103.0 | loss = 0.15
414246916770935 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 2606 | episode_rewards = 13.0 | dw
_time_elapsed = 20.010944462000225 | episode = 118 | eval_rewards = 83.2 | loss = 0.0
7086402922868729 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 3404 | episode_rewards = 183.0 | d
w_time_elapsed = 30.01388360400051 | episode = 96 | eval_rewards = 168.8 | loss = 0.1
0898501425981522 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 3531 | episode_rewards = 200.0 | d
w_time_elapsed = 30.018955123000524 | episode = 131 | eval_rewards = 200.0 | loss =
0.3682723343372345 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 4404 | episode_rewards = 200.0 | d
w_time_elapsed = 40.023223230999974 | episode = 136 | eval_rewards = 198.4 | loss =
0.21331265568733215 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 4250 | episode_rewards = 187.0 | d
w_time_elapsed = 40.28163850800047 | episode = 103 | eval_rewards = 195.6 | loss = 0.
209097221493721 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 5245 | episode_rewards = 200.0 | d
w_time_elapsed = 50.02926848300012 | episode = 140 | eval_rewards = 200.0 | loss = 0.
17052534222602844 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 5125 | episode_rewards = 200.0 | d
w_time_elapsed = 50.28793422700073 | episode = 108 | eval_rewards = 197.6 | loss = 0.
16717448830604553 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 6045 | episode_rewards = 200.0 | d
w_time_elapsed = 60.03690498300057 | episode = 144 | eval_rewards = 200.0 | loss = 0.
17374354600906372 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 5993 | episode_rewards = 200.0 | d
w_time_elapsed = 60.29364048000025 | episode = 112 | eval_rewards = 200.0 | loss = 0.
181975319981575 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 6932 | episode_rewards = 200.0 | d
w_time_elapsed = 70.03897952900024 | episode = 149 | eval_rewards = 200.0 | loss = 0.
3505435287952423 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 6802 | episode_rewards = 178.0 | d
w_time_elapsed = 70.2948439390002 | episode = 116 | eval_rewards = 200.0 | loss = 0.7
87834644317627 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 7750 | episode_rewards = 200.0 | d
w_time_elapsed = 80.08654569600003 | episode = 153 | eval_rewards = 200.0 | loss = 0.
27858206629753113 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 7705 | episode_rewards = 200.0 | d
w_time_elapsed = 80.29714858900024 | episode = 121 | eval_rewards = 196.0 | loss = 0.
3346664607524872 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 8623 | episode_rewards = 200.0 | d
w_time_elapsed = 90.08841947500059 | episode = 157 | eval_rewards = 200.0 | loss = 0.
19313400983810425 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 8535 | episode_rewards = 192.0 | d
w_time_elapsed = 90.305790677 | episode = 125 | eval_rewards = 157.0 | loss = 0.21961
674094200134 |
[INFO] [DuelingDQN[worker: 1]] | max_global_step = 9475 | episode_rewards = 200.0 | d
w_time_elapsed = 100.09692104200076 | episode = 162 | eval_rewards = 200.0 | loss =
3.85347318649292 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 9385 | episode_rewards = 200.0 | d
w_time_elapsed = 100.3110876540004 | episode = 130 | eval_rewards = 200.0 | loss = 0.
6641786098480225 |
```

```
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

In [17]:

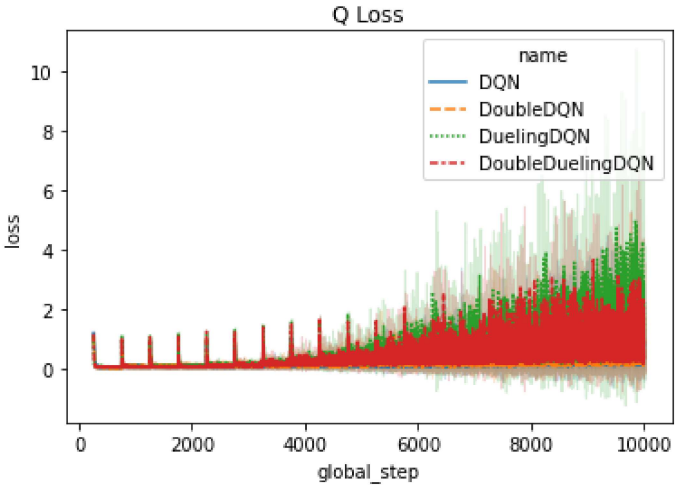
```
# Double+Dueling DQN
double_dueling_dqn_manager = AgentManager(
    init_kwargs=DOUBLE_DUELING_DQN_PARAMS,
    agent_name='DoubleDuelingDQN',
    **manager_kwargs
)
double_dueling_dqn_manager.fit()
```

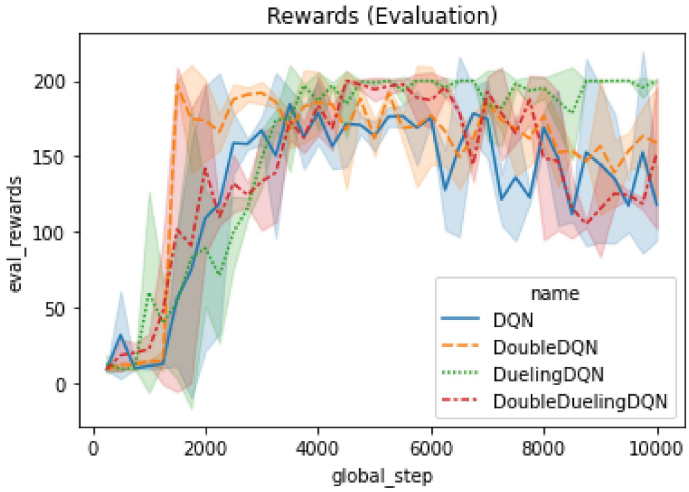
```
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for DoubleDuelingDQN...
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 1361 | episode_rewards = 42.
0 | dw_time_elapsed = 10.006089262000387 | episode = 63 | eval_rewards = 13.8 | loss
= 0.05334196984767914 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 1320 | episode_rewards = 17.
0 | dw_time_elapsed = 10.005854771000486 | episode = 59 | eval_rewards = 82.0 | loss
= 0.06321326643228531 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 2250 | episode_rewards = 69.
0 | dw_time_elapsed = 20.0248539840004 | episode = 89 | eval_rewards = 113.8 | loss =
0.07876498252153397 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 3294 | episode_rewards = 30.
0 | dw_time_elapsed = 30.017039512000338 | episode = 107 | eval_rewards = 116.4 | los
s = 0.1632828265428543 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 3205 | episode_rewards = 14
2.0 | dw_time_elapsed = 30.02521093300038 | episode = 105 | eval_rewards = 156.2 | lo
ss = 0.22887545824050903 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 4191 | episode_rewards = 18
5.0 | dw_time_elapsed = 40.01790362599968 | episode = 113 | eval_rewards = 170.0 | lo
ss = 0.11239340901374817 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 4057 | episode_rewards = 10
1.0 | dw_time_elapsed = 40.02811527799986 | episode = 111 | eval_rewards = 197.6 | lo
ss = 0.13230405747890472 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 4927 | episode_rewards = 20
0.0 | dw_time_elapsed = 50.03290741700039 | episode = 116 | eval_rewards = 195.0 | lo
ss = 0.36881372332572937 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 5000 | episode_rewards = 20
0.0 | dw_time_elapsed = 50.08756600699962 | episode = 117 | eval_rewards = 199.8 | lo
ss = 0.15566711127758026 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 5830 | episode_rewards = 20
0.0 | dw_time_elapsed = 60.088666984000156 | episode = 121 | eval_rewards = 200.0 | l
oss = 0.12665626406669617 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 5750 | episode_rewards = 19
2.0 | dw_time_elapsed = 60.433171370999844 | episode = 120 | eval_rewards = 177.8 | l
oss = 0.3068544268608093 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 6664 | episode_rewards = 20
0.0 | dw_time_elapsed = 70.09360035200007 | episode = 126 | eval_rewards = 180.4 | lo
ss = 1.0298560857772827 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 6608 | episode_rewards = 17
5.0 | dw_time_elapsed = 70.44197034199988 | episode = 124 | eval_rewards = 178.0 | lo
ss = 1.7890489101409912 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 7482 | episode_rewards = 18
1.0 | dw_time_elapsed = 80.45058295699982 | episode = 129 | eval_rewards = 156.8 | lo
ss = 0.9220632910728455 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 7500 | episode_rewards = 20
0.0 | dw_time_elapsed = 80.57636201300011 | episode = 130 | eval_rewards = 161.0 | lo
ss = 0.14570944011211395 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 8326 | episode_rewards = 12
1.0 | dw_time_elapsed = 90.4567782539998 | episode = 136 | eval_rewards = 114.8 | los
s = 0.6797748804092407 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 8368 | episode_rewards = 17
5.0 | dw_time_elapsed = 90.57961428300041 | episode = 134 | eval_rewards = 179.2 | lo
ss = 0.37469393014907837 |
[INFO] [DoubleDuelingDQN[worker: 1]] | max_global_step = 9250 | episode_rewards = 13
1.0 | dw_time_elapsed = 100.65445842600002 | episode = 143 | eval_rewards = 119.4 | l
oss = 0.2632215619087219 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 9250 | episode_rewards = 15
3.0 | dw_time_elapsed = 100.69671326200023 | episode = 140 | eval_rewards = 131.6 | l
oss = 1.6002917289733887 |
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

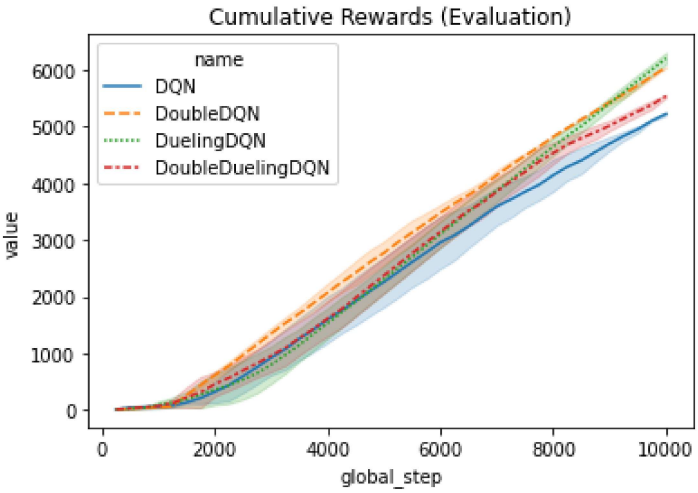
```
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

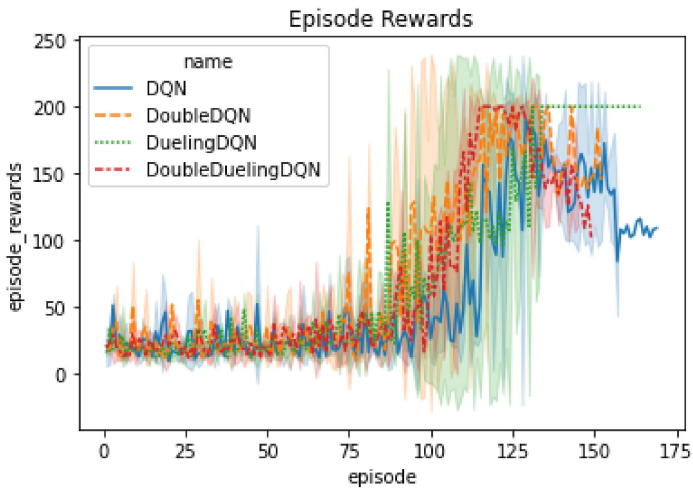
In [18]:

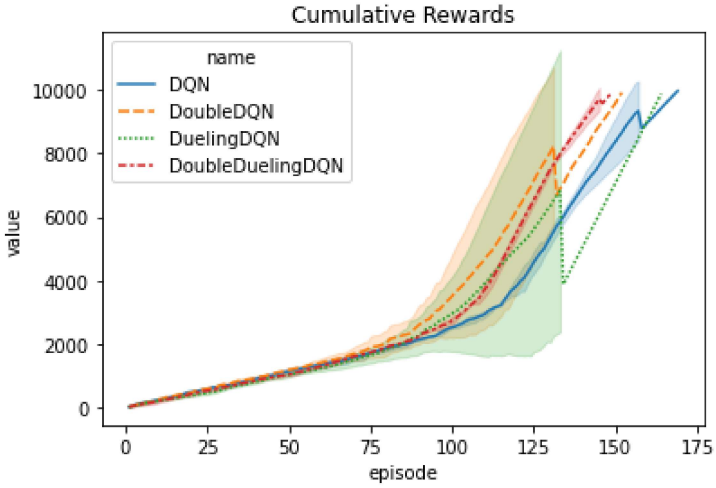
```
all_dqn_managers = []
all_dqn_managers.append(dqn_manager)
all_dqn_managers.append(double_dqn_manager)
all_dqn_managers.append(dueling_dqn_manager)
all_dqn_managers.append(double_dueling_dqn_manager)

# We can plot the data that was
# stored by the agent with self.writer.add_scalar(tag, value, global_step):
_ = plot_writer_data(all_dqn_managers, tag='loss', title='Q Loss')
_ = plot_writer_data(all_dqn_managers, tag='eval_rewards', title='Rewards (Evaluatio
n)')
_ = plot_writer_data(all_dqn_managers, tag='eval_rewards', title='Cumulative Rewards
(Evaluation)', preprocess_func=np.cumsum)

# rewards in each episode
# Warning: not all runs will have the same number of episodes. Can you see why?
_ = plot_writer_data(all_dqn_managers, tag='episode_rewards', xtag='episode', title
='Episode Rewards')
_ = plot_writer_data(all_dqn_managers, tag='episode_rewards', xtag='episode', title
='Cumulative Rewards', preprocess_func=np.cumsum)
```


Q Loss
name
DQN
DoubleDQN
DuelingDQN
DoubleDuelingDQN
loss
global_step


Rewards (Evaluation)
name
DQN
DoubleDQN
DuelingDQN
DoubleDuelingDQN
eval_rewards
global_step


Cumulative Rewards (Evaluation)
name
DQN
DoubleDQN
DuelingDQN
DoubleDuelingDQN
value
global_step


Episode Rewards
name
DQN
DoubleDQN
DuelingDQN
DoubleDuelingDQN
episode_rewards
episode



In [28]:
```
DQN_instance = all_dqn_managers[2].get_agent_instances()[1]
render_policy(DQN_instance.eval_env, DQN_instance)
```

```
INFO: Clearing 4 monitor files from previous run (because force=True was provided)
INFO: Starting new video recorder writing to /content/videos/openaigym.video.3.2436.v
ideo000000.mp4
INFO: Finished writing results. You can upload them to the scoreboard via gym.upload
('/content/videos')
```

0:00 / 0:04

Part 2: REINFORCE

Question 2.1 (written)

In class we have seen the derivation of the policy gradient theorem in the "Monte-Carlo style". Recall that the policy gradient is given by

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\cdot|\pi_\theta)} \left[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta) R(\tau) \right]$$

where $R(\tau) = \sum_{t=1}^{|\tau|} r_t$. By construction the policy gradient is on-policy, we need to estimate the gradient using the samples collected through the current policy π_θ.

1. *Derive an off-policy variant* by assuming to collect samples from a behavioral policy $\mu(s, a)$. The target policy, i.e., the policy for which we want to compute the gradient is π_θ. Write explicitly the gradient log probability in your derivation.
2. What are the properties that μ needs to satisfy to obtain a meaningful gradient estimate?

1.One have :

$$\begin{aligned}
J(\pi_\theta) &= \mathbb{E}_{\tau \sim \mathbb{P}(.|\pi_\theta)}[R(\tau)] \\
&= \int R(\tau)\mathbb{P}(\tau|\pi_\theta)d\tau \\
&= \int \frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} R(\tau)\mathbb{P}(\tau|\mu)d\tau \\
&= \mathbb{E}_{\tau \sim \mathbb{P}(.|\mu)}[\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} R(\tau)]
\end{aligned}$$

From that we can derive an off-policy variant :

$$\begin{aligned}
\nabla_\theta J(\pi_\theta) &= \nabla_\theta \mathbb{E}_{\tau \sim \mathbb{P}(.|\mu)}[\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} R(\tau)] \\
&= \nabla_\theta \int \frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} R(\tau)\mathbb{P}(\tau|\mu)d\tau \\
&= \int \frac{\nabla_\theta \mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} R(\tau)\mathbb{P}(\tau|\mu)d\tau \\
&= \int \frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} \nabla_\theta log(\mathbb{P}(\tau|\pi_\theta)) R(\tau)\mathbb{P}(\tau|\mu)d\tau \\
&= \mathbb{E}_{\tau \sim \mathbb{P}(.|\mu)}[\frac{\mathbb{P}(\tau|\pi_\theta)}{\mathbb{P}(\tau|\mu)} \nabla_\theta log(\mathbb{P}(\tau|\pi_\theta)) R(\tau)]
\end{aligned}$$

With $\mathbb{P}(\tau|\mu) = \mu_0(s_0)\mu(s_0, a_0)\mathbb{P}(s_1|s_0, a_0)\mu(s_1, a_1)\dots$

1. To have a meaningful gradient, μ need to have the same support that π_θ, if it's not the case, we may end up with a gradient equal to zero. Moreover, μ need to be close from π_θ, otherwise our gradient can take very large values ($\approx \infty$) or very small values (≈ 0), and so our gradient will be not meaningful.

We may also want a policy that is good to explore the environement. Indeed, if we sample with a policy that has a bad exploration this can lead to a bad algorithm. For example, if the policy samples always the same trajectory, we will have more or less the same gradient at each step, and our chances to converge towards the optimal policy are low.

Question 2.2 (implementation)

Starting from the provided code, implement on-policy policy gradient

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau \sim \mathbb{P}(\cdot|\pi_\theta)} \left[\nabla_\theta \log \mathbb{P}(\tau|\pi_\theta) R(\tau) \right]$$

Use the most efficient version of the gradient estimator without baseline (i.e., GPOMDP).

In [20]:
```
class PolicyNet(nn.Module):
  def __init__(self, obs_size, n_actions):
    super(PolicyNet, self).__init__()
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    #  TO DO : Implement network & Forward method
    #  You can use the same architecture as DQN
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    self.fc1 = nn.Linear(obs_size, 64)
    self.ReLU = torch.relu
    self.fc2 = nn.Linear(64,64)
    self.fc3 = nn.Linear(64,n_actions)
    self.softmax = nn.Softmax(dim=-1)

  def forward(self, state):
    action_logits = self.fc1(state)
    action_logits = self.ReLU(action_logits)
    action_logits = self.fc2(action_logits)
    action_logits = self.ReLU(action_logits)
    action_logits = self.fc3(action_logits)
    # compute softmax of the output of the net, to get probabilities over actions
    action_probs = self.softmax(action_logits)
    dist = Categorical(action_probs)
    return action_logits, dist
```

In [21]:
```
# Parameters
REINFORCE_TRAINING_TIMESTEPS = 5*DQN_TRAINING_TIMESTEPS  # number of timesteps for tr
aining. You might change this!

REINFORCE_PARAMS = dict(
    gamma=0.99,
    batch_size=200,            # batch size
    eval_every=250,            # evaluate every ... steps
    learning_rate=0.001,       # learning rate
)
```

In [22]:

```
class REINFORCEAgent(Agent):
  name = 'REINFORCE'
  def __init__(
      self,
      env,
      gamma: float = 0.99,
      batch_size: int = 200,
      eval_every: int = 250,
      learning_rate: float = 0.1,
      **kwargs):
    Agent.__init__(self, env, **kwargs)
    env = self.env
    self.gamma = gamma
    self.batch_size = batch_size
    self.eval_every = eval_every
    self.total_timesteps = 0
    self.total_episodes = 0
    self.total_updates = 0

    # create network
    obs_size = env.observation_space.shape[0]
    n_actions = env.action_space.n
    self.policy_net = PolicyNet(obs_size, n_actions)

    # optimizer
    self.optimizer = optim.Adam(
        params=self.policy_net.parameters(), lr=learning_rate)

  def select_action(self, state, evaluation=False):
    """
    If evaluation=False, get action according to exploration policy.
    Otherwise, get action according to the evaluation policy.
    """
    tensor_state = torch.FloatTensor(state).to(device)
    with torch.no_grad():
      _, dist = self.policy_net(tensor_state)
      action = dist.sample().item()
    return action

  def fit(self, budget):
    """
    budget : number of training timesteps
    """
    state = self.env.reset()
    done = False
    episode_reward = 0.0

    trajectory_states = []
    trajectory_actions = []
    trajectory_rewards = []
    trajectory_dones = []

    for tt in range(budget):
      self.total_timesteps += 1
      action = self.select_action(state, evaluation=False)
      next_state, reward, done, _ = self.env.step(action)
      episode_reward += reward

      trajectory_states.append(state)
      trajectory_actions.append(action)
      trajectory_rewards.append(reward)
      trajectory_dones.append(done)

      if ((tt + 1) % self.batch_size) == 0:
        #
        # Update model
        #
```

```
        self.total_updates += 1

        # convert to torch tensors
        batch_state = torch.FloatTensor(
            np.array(trajectory_states, dtype=float)
            ).to(device)
        batch_action = torch.LongTensor(
             np.array(trajectory_actions, dtype=int)
             ).unsqueeze(1).to(device)
        batch_reward = torch.FloatTensor(
             np.array(trajectory_rewards, dtype=float)
            ).unsqueeze(1).to(device)
        batch_done = torch.FloatTensor(
            np.array(trajectory_dones, dtype=bool)
            ).unsqueeze(1).to(device)

        # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        #  TO DO: compute loss for REINFORCE
        # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
        assert len(trajectory_actions) == self.batch_size

        action_logits, _ = self.policy_net(batch_state)
        logprobs = nn.functional.log_softmax(action_logits, dim=-1).gather(1, batch_a
ction.long())

        prev_T = 0
        q = torch.zeros_like(batch_reward)
        nb_episode = len(batch_done[batch_done==1])
        index_done = (batch_done).nonzero(as_tuple=True)[0]

        for T in index_done :
            #We add 1 because arange(a,b) dans Tensor[a:b] don't include b but I
            #want the reward of the terminale state in my calculations.
            gamma_power = (self.gamma**torch.arange(T-prev_T+1)).unsqueeze(1)
            discounted_rewards = gamma_power*batch_reward[prev_T:T+1]
            #we avoid index T in order to have q[T]=0.
            for i in range(prev_T,T):
              q[i]=torch.cumsum(discounted_rewards[i-prev_T:],axis=0)[-1]

            #T+1 be cause we start the next epsiode at index T+1 not T.
            prev_T=T+1

        #Check if it remain an incomplete episode.
        gamma_power = (self.gamma**torch.arange(len(batch_done)-prev_T)).unsqueeze(1)
        discounted_rewards = gamma_power*batch_reward[prev_T:]
        for i in range(prev_T,len(batch_done)):
          q[i]=torch.cumsum(discounted_rewards[i-prev_T:],axis=0)[-1]


        loss = torch.sum(-logprobs*q)/(nb_episode+1)
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # log info about loss
        self.writer.add_scalar('loss', loss.item(), self.total_timesteps)

        # clear data
        trajectory_states = []
        trajectory_actions = []
        trajectory_rewards = []
        trajectory_dones = []

      # evaluate agent
      if self.total_timesteps % self.eval_every == 0:
          mean_rewards = self.eval(n_sim=5)
          self.writer.add_scalar(
              'eval_rewards', mean_rewards, self.total_timesteps)
```

```
    # check end of episode
    state = next_state
    if done:
      state = self.env.reset()
      self.total_episodes += 1
      self.writer.add_scalar(
              'episode_rewards', episode_reward, self.total_timesteps)
      self.writer.add_scalar(
              'episode', self.total_episodes, self.total_timesteps)
      episode_reward = 0.0

def eval(self, n_sim=1, **kwargs):
  rewards = np.zeros(n_sim)
  eval_env = self.eval_env     # evaluation environment
  # Loop over number of simulations
  for sim in range(n_sim):
    state = eval_env.reset()
    done = False
    while not done:
      action = self.select_action(state, evaluation=True)
      next_state, reward, done, _ = eval_env.step(action)
      # update sum of rewards
      rewards[sim] += reward
      state = next_state
  return rewards.mean()
```

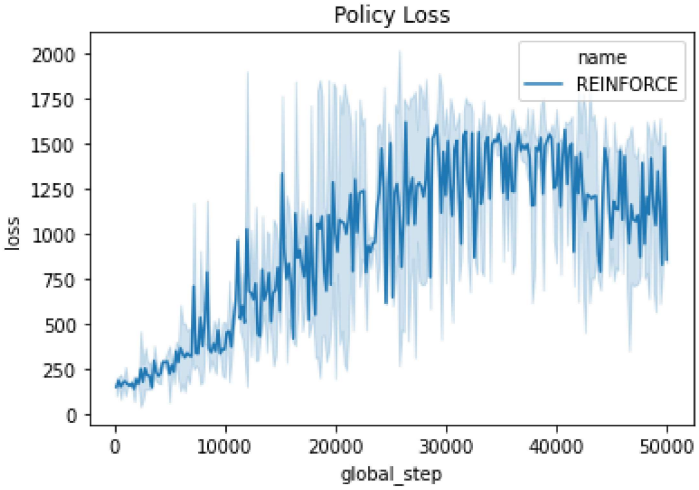
In [23]:

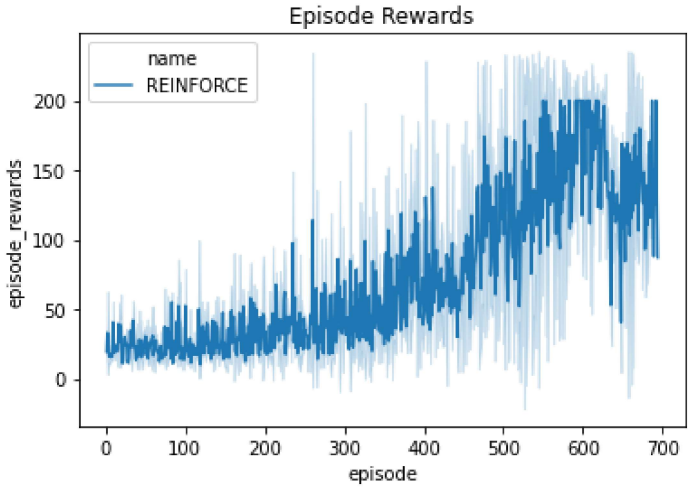
```
# # Training one instance of REINFORCE
# reinforce_agent = REINFORCEAgent(
#     env=(get_env, dict()),  # we can send (constructor, kwargs) as an env
#     **REINFORCE_PARAMS
# )
# reinforce_agent.fit(REINFORCE_TRAINING_TIMESTEPS)

#
# Training several instances using AgentManager
#
manager_kwargs = dict(
    agent_class=REINFORCEAgent,
    train_env=(get_env, dict()),
    eval_env=(get_env, dict()),
    fit_budget=REINFORCE_TRAINING_TIMESTEPS,
    n_fit=2,                # NOTE: You may increase this parameter (number of agents
to train)
    parallelization='thread',
    seed=456,
    default_writer_kwargs=dict(maxlen=None,log_interval=10),
)

# REINFORCE
reinforce_manager = AgentManager(
    init_kwargs=REINFORCE_PARAMS,
    agent_name='REINFORCE',
    **manager_kwargs
)

# Train and append to previous managers
reinforce_manager.fit()
```

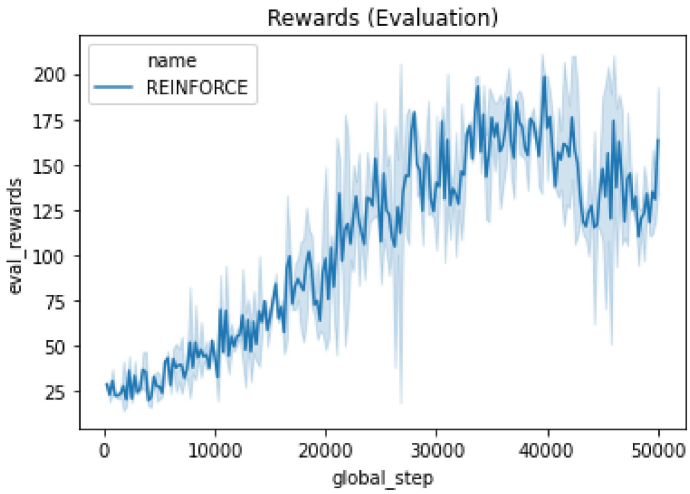
```
[WARNING] (Re)defining the following DefaultWriter parameters in AgentManager: ['maxl
en', 'log_interval']
[INFO] Running AgentManager fit() for REINFORCE...
INFO: Making new env: CartPole-v0
/usr/local/lib/python3.8/dist-packages/gym/envs/registration.py:505: UserWarning: WAR
N: The environment CartPole-v0 is out of date. You should consider upgrading to versi
on `v1` with the environment ID `CartPole-v1`.
  logger.warn(
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
[INFO] [REINFORCE[worker: 0]] | max_global_step = 6459 | episode_rewards = 28.0 | dw_
time_elapsed = 10.01504894000027 | episode = 219 | loss = 448.4236755371094 | eval_re
wards = 47.0 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 6500 | episode_rewards = 47.0 | dw_
time_elapsed = 10.130625645999316 | episode = 259 | loss = 178.16693115234375 | eval_
rewards = 28.6 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 11846 | episode_rewards = 72.0 | dw
_time_elapsed = 20.0438506520004 | episode = 332 | loss = 631.726806640625 | eval_rew
ards = 48.0 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 12000 | episode_rewards = 147.0 | d
w_time_elapsed = 20.14407704899986 | episode = 383 | loss = 1643.5623779296875 | eval
_rewards = 52.2 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 16133 | episode_rewards = 89.0 | dw
_time_elapsed = 30.202018210999995 | episode = 440 | loss = 542.435546875 | eval_rewa
rds = 68.6 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 16250 | episode_rewards = 51.0 | dw
_time_elapsed = 30.38112532500054 | episode = 401 | loss = 402.8597106933594 | eval_r
ewards = 49.2 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 19469 | episode_rewards = 200.0 | d
w_time_elapsed = 40.2233190879997 | episode = 473 | loss = 1634.138427734375 | eval_r
ewards = 90.0 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 20250 | episode_rewards = 45.0 | dw
_time_elapsed = 40.58971553200081 | episode = 462 | loss = 814.3150024414062 | eval_r
ewards = 60.4 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 22500 | episode_rewards = 139.0 | d
w_time_elapsed = 50.38855419099946 | episode = 497 | loss = 971.2975463867188 | eval_
rewards = 141.4 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 24000 | episode_rewards = 178.0 | d
w_time_elapsed = 51.01003046900041 | episode = 501 | loss = 1499.937255859375 | eval_
rewards = 122.4 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 25500 | episode_rewards = 200.0 | d
w_time_elapsed = 60.85947604299963 | episode = 523 | loss = 1576.749267578125 | eval_
rewards = 142.4 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 27500 | episode_rewards = 157.0 | d
w_time_elapsed = 61.17481564800073 | episode = 540 | loss = 1048.54150390625 | eval_r
ewards = 117.2 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 28250 | episode_rewards = 200.0 | d
w_time_elapsed = 71.03682566999942 | episode = 538 | loss = 1694.61328125 | eval_rewa
rds = 137.8 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 30400 | episode_rewards = 166.0 | d
w_time_elapsed = 71.30368482000085 | episode = 561 | loss = 1608.48046875 | eval_rewa
rds = 140.8 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 31000 | episode_rewards = 108.0 | d
w_time_elapsed = 81.31386207899959 | episode = 556 | loss = 1584.070068359375 | eval_
rewards = 189.6 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 33250 | episode_rewards = 200.0 | d
w_time_elapsed = 81.49533664400042 | episode = 582 | loss = 1427.9794921875 | eval_re
wards = 151.8 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 33750 | episode_rewards = 200.0 | d
w_time_elapsed = 91.666597077 | episode = 574 | loss = 1525.2735595703125 | eval_rewa
rds = 188.8 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 35750 | episode_rewards = 200.0 | d
w_time_elapsed = 91.71992992800006 | episode = 597 | loss = 1390.48486328125 | eval_r
ewards = 179.6 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 36478 | episode_rewards = 200.0 | d
w_time_elapsed = 101.71589459499955 | episode = 590 | loss = 1519.9921875 | eval_rewa
```

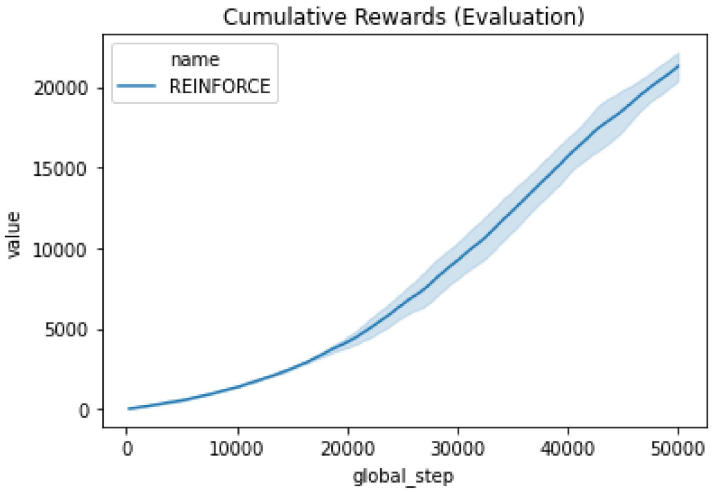
```
rds = 154.6 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 38250 | episode_rewards = 200.0 | d
w_time_elapsed = 102.13199726400035 | episode = 611 | loss = 1449.6046142578125 | eva
l_rewards = 168.8 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 39000 | episode_rewards = 197.0 | d
w_time_elapsed = 111.78778041699934 | episode = 604 | loss = 1559.928466796875 | eval
_rewards = 179.0 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 40750 | episode_rewards = 200.0 | d
w_time_elapsed = 112.29923017400051 | episode = 626 | loss = 1379.41162109375 | eval_
rewards = 139.0 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 43800 | episode_rewards = 120.0 | d
w_time_elapsed = 122.32267159399998 | episode = 651 | loss = 781.3135375976562 | eval
_rewards = 140.4 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 41500 | episode_rewards = 200.0 | d
w_time_elapsed = 122.37038374999975 | episode = 619 | loss = 1607.730712890625 | eval
_rewards = 193.4 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 44335 | episode_rewards = 144.0 | d
w_time_elapsed = 132.43924881699968 | episode = 638 | loss = 1000.986328125 | eval_re
wards = 77.8 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 46500 | episode_rewards = 102.0 | d
w_time_elapsed = 133.10311604300023 | episode = 667 | loss = 803.1593627929688 | eval
_rewards = 181.0 |
[INFO] [REINFORCE[worker: 1]] | max_global_step = 47500 | episode_rewards = 162.0 | d
w_time_elapsed = 142.5216965720001 | episode = 667 | loss = 1369.220703125 | eval_rew
ards = 140.8 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 49500 | episode_rewards = 150.0 | d
w_time_elapsed = 143.20834174100037 | episode = 690 | loss = 763.7030639648438 | eval
_rewards = 118.4 |
[INFO] ... trained!
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
INFO: Making new env: CartPole-v0
```

In [24]:
```
# Plots for REINFORCE
_ = plot_writer_data(reinforce_manager, tag='loss', title='Policy Loss')
_ = plot_writer_data(reinforce_manager, tag='episode_rewards', xtag='episode', title
='Episode Rewards')
_ = plot_writer_data(reinforce_manager, tag='eval_rewards', title='Rewards (Evaluatio
n)')
_ = plot_writer_data(reinforce_manager, tag='eval_rewards', title='Cumulative Rewards
(Evaluation)', preprocess_func=np.cumsum)
```


Policy Loss
name
REINFORCE
loss
global_step
0
250
500
750
1000
1250
1500
1750
2000
0
10000
20000
30000
40000
50000


Episode Rewards
name
REINFORCE
episode_rewards
episode
0
50
100
150
200
0
100
200
300
400
500
600
700


Rewards (Evaluation)
name
REINFORCE
eval_rewards
global_step
25
50
75
100
125
150
175
200
0
10000
20000
30000
40000
50000



In [29]:
```
reinforce_instance = reinforce_manager.get_agent_instances()[0]
render_policy(reinforce_instance.eval_env, reinforce_instance)
```

```
INFO: Clearing 4 monitor files from previous run (because force=True was provided)
INFO: Starting new video recorder writing to /content/videos/openaigym.video.4.2436.v
ideo000000.mp4
INFO: Finished writing results. You can upload them to the scoreboard via gym.upload
('/content/videos')
```

0:00 / 0:02

Part 3: Model Selection Using UCB

Question 3.1: Model Selection for RL

In RL, it is hard to define in advance what is the best method to use and/or which architecture to choose for representing the policy or the value function.

Normally, several algorithms are run in parallel and the final performance is used to select the best algorithm. Can we do this online?

In this exercise, we present a simple idea for doing online model-selection by leveraging a multi-armed bandit (MAB) algorithm.

You will start from a list containing one instance of each agent that you trained above (different versions of DQN and REINFORCE), and you'll adaptively allocate computational resources to train those algorithms for more timesteps.

Implement below the UCB algorithm such that, at reach round t:

- UCB selects the action a_t, which corresponds to one of the algorithms in the list;
- You train the algorithm for a few timesteps;
- UCB receives the reward r_t which is equal to the Monte-Carlo policy evaluation of the chosen algorithm.

The goal of this strategy is to allocate more computational resources to the algorithms that are yielding more rewards.

In [26]:

```
# Get one (pre-)trained agent from each manager
all_managers = all_dqn_managers + [reinforce_manager]
arms = [manager.get_agent_instances()[0] for manager in all_managers]
print(f"Arms/algorithms: {[alg.name for alg in arms]}")

# Select number of UCB rounds
UCB_N_ROUNDS = 200

# Number of training steps every time an agent is selected
N_STEPS_TO_TRAIN = 250

# Run UCB
all_episode_rewards = []
arm_counts = np.zeros(len(arms))
arm_cumulative_rewards = np.zeros(len(arms))
for tt in range(UCB_N_ROUNDS):
  if tt < len(arms):
    arm_index = tt
  else:
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    # TODO: Implement UCB arm selection
    # !!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!!
    arm_index = np.argmax((arm_cumulative_rewards/arm_counts)+np.sqrt(np.log(2*(tt)**
2)/arm_counts) )

  # train the selected algorithm for more timesteps
  arms[arm_index].fit(N_STEPS_TO_TRAIN)

  # evaluate arm
  episode_reward = arms[arm_index].eval()

  # update stats
  arm_counts[arm_index] += 1
  arm_cumulative_rewards[arm_index] += episode_reward

  # store reward
  all_episode_rewards.append(episode_reward)

  if (tt+1) % 10 == 0:
    print("----------------------------------------------------------------")
    print(f"UCB round {tt+1}, "
          f"chosen algorithm: {arms[arm_index].name}, "
          f"eval reward = {episode_reward}.")
    print("----------------------------------------------------------------")
```

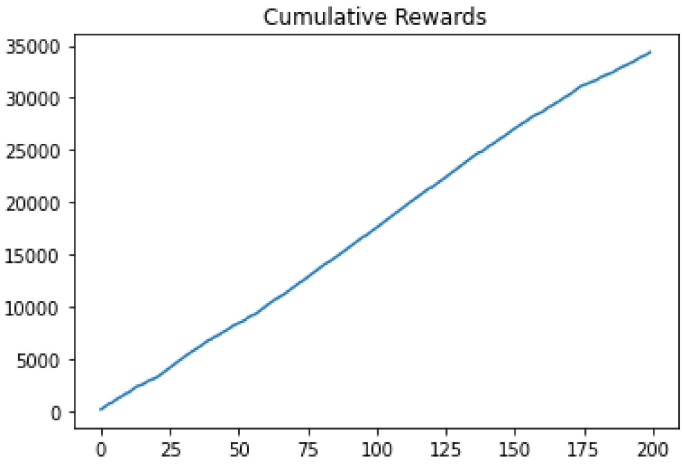
```
[INFO] [DQN[worker: 0]] | max_global_step = 10001 | episode_rewards = 55.0 | dw_time_
elapsed = 541.7214535479998 | episode = 157 | eval_rewards = 135.0 | loss = 0.2195590
7344818115 |

Arms/algorithms: ['DQN', 'DoubleDQN', 'DuelingDQN', 'DoubleDuelingDQN', 'REINFORCE']

[INFO] [DoubleDQN[worker: 0]] | max_global_step = 10001 | episode_rewards = 200.0 | d
w_time_elapsed = 456.72406695600057 | episode = 131 | eval_rewards = 133.2 | loss =
0.06767435371875763 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 10001 | episode_rewards = 200.0 |
dw_time_elapsed = 373.1590343000007 | episode = 133 | eval_rewards = 200.0 | loss =
0.4267362356185913 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 10001 | episode_rewards = 15
0.0 | dw_time_elapsed = 266.04926593599976 | episode = 145 | eval_rewards = 187.4 | l
oss = 2.2961864471435547 |
[INFO] [REINFORCE[worker: 0]] | max_global_step = 50200 | episode_rewards = 87.0 | dw
_time_elapsed = 153.55373554700054 | episode = 694 | loss = 1413.1202392578125 | eval
_rewards = 184.0 |

------------------------------------------------------------
UCB round 10, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 11643 | episode_rewards = 200.0 |
dw_time_elapsed = 383.1613918140001 | episode = 139 | eval_rewards = 109.4 | loss =
4.843062877655029 |
[INFO] [DoubleDQN[worker: 0]] | max_global_step = 10251 | episode_rewards = 155.0 | d
w_time_elapsed = 472.03230820299996 | episode = 132 | eval_rewards = 157.0 | loss =
0.1702594757080078 |
[INFO] [DoubleDuelingDQN[worker: 0]] | max_global_step = 10501 | episode_rewards = 12
8.0 | dw_time_elapsed = 282.24721127500015 | episode = 147 | eval_rewards = 156.8 | l
oss = 0.8191589117050171 |

------------------------------------------------------------
UCB round 20, chosen algorithm: DoubleDuelingDQN, eval reward = 119.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 13745 | episode_rewards = 200.0 |
dw_time_elapsed = 393.16366132200073 | episode = 148 | eval_rewards = 125.0 | loss =
3.810828685760498 |

------------------------------------------------------------
UCB round 30, chosen algorithm: DuelingDQN, eval reward = 184.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 16000 | episode_rewards = 200.0 |
dw_time_elapsed = 403.18512723799995 | episode = 157 | eval_rewards = 181.6 | loss =
9.769478797912598 |

------------------------------------------------------------
UCB round 40, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 18462 | episode_rewards = 200.0 |
dw_time_elapsed = 413.187913924 | episode = 167 | eval_rewards = 200.0 | loss = 4.316
808223724365 |

------------------------------------------------------------
UCB round 50, chosen algorithm: DuelingDQN, eval reward = 117.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 21000 | episode_rewards = 200.0 |
dw_time_elapsed = 423.2606400250006 | episode = 178 | eval_rewards = 126.8 | loss =
0.7078510522842407 |

------------------------------------------------------------
UCB round 60, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------
```

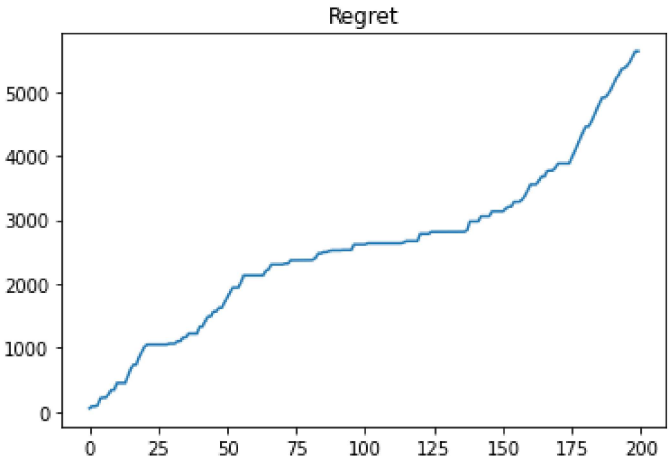
```
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 23501 | episode_rewards = 200.0 |
dw_time_elapsed = 433.30722746000083 | episode = 188 | eval_rewards = 200.0 | loss =
7.144288063049316 |

------------------------------------------------------------
UCB round 70, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 26000 | episode_rewards = 200.0 |
dw_time_elapsed = 443.4508916570003 | episode = 198 | eval_rewards = 200.0 | loss =
8.06419849395752 |
[INFO] [DuelingDQN[worker: 0]] | max_global_step = 27978 | episode_rewards = 200.0 |
dw_time_elapsed = 453.4524928610008 | episode = 206 | eval_rewards = 200.0 | loss =
1.059225082397461 |

------------------------------------------------------------
UCB round 80, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 30333 | episode_rewards = 184.0 |
dw_time_elapsed = 463.45518211800027 | episode = 216 | eval_rewards = 190.2 | loss =
2.3702287673950195 |

------------------------------------------------------------
UCB round 90, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 32750 | episode_rewards = 168.0 |
dw_time_elapsed = 473.4711893530002 | episode = 227 | eval_rewards = 191.0 | loss = 1
6.9434871673584 |

------------------------------------------------------------
UCB round 100, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 35131 | episode_rewards = 200.0 |
dw_time_elapsed = 483.4715760600002 | episode = 236 | eval_rewards = 200.0 | loss = 2
9.05024528503418 |

------------------------------------------------------------
UCB round 110, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 37500 | episode_rewards = 200.0 |
dw_time_elapsed = 493.50219972300056 | episode = 246 | eval_rewards = 200.0 | loss =
13.821199417114258 |

------------------------------------------------------------
UCB round 120, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 39938 | episode_rewards = 200.0 |
dw_time_elapsed = 503.5027934590007 | episode = 255 | eval_rewards = 198.6 | loss = 2
5.70690155029297 |

------------------------------------------------------------
UCB round 130, chosen algorithm: DuelingDQN, eval reward = 198.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 42251 | episode_rewards = 200.0 |
dw_time_elapsed = 513.5540728020005 | episode = 265 | eval_rewards = 200.0 | loss =
1.0615870952606201 |

------------------------------------------------------------
UCB round 140, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 44750 | episode_rewards = 200.0 |
dw_time_elapsed = 523.7141687650001 | episode = 275 | eval_rewards = 188.4 | loss =
1.1659635305404663 |
```

```
------------------------------------------------------------
UCB round 150, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 47250 | episode_rewards = 181.0 |
dw_time_elapsed = 533.8208036550004 | episode = 286 | eval_rewards = 156.6 | loss =
0.7931839227676392 |

------------------------------------------------------------
UCB round 160, chosen algorithm: DuelingDQN, eval reward = 118.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 49791 | episode_rewards = 32.0 | d
w_time_elapsed = 543.8217086070008 | episode = 303 | eval_rewards = 180.4 | loss = 1.
0031845569610596 |

------------------------------------------------------------
UCB round 170, chosen algorithm: DuelingDQN, eval reward = 153.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 52305 | episode_rewards = 133.0 |
dw_time_elapsed = 553.8242112190001 | episode = 316 | eval_rewards = 108.0 | loss =
2.220078468322754 |

------------------------------------------------------------
UCB round 180, chosen algorithm: DuelingDQN, eval reward = 104.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 55000 | episode_rewards = 104.0 |
dw_time_elapsed = 563.9055892900005 | episode = 334 | eval_rewards = 95.0 | loss = 1.
8926453590393066 |

------------------------------------------------------------
UCB round 190, chosen algorithm: DuelingDQN, eval reward = 132.0.
------------------------------------------------------------

[INFO] [DuelingDQN[worker: 0]] | max_global_step = 57606 | episode_rewards = 200.0 |
dw_time_elapsed = 573.906377151 | episode = 349 | eval_rewards = 150.0 | loss = 2.932
6164722442627 |

------------------------------------------------------------
UCB round 200, chosen algorithm: DuelingDQN, eval reward = 200.0.
------------------------------------------------------------
```

In [27]:
```
# Print stats
print(f"Arms: {[alg.name for alg in arms]}")
print(f"Counts: {arm_counts}")
print(f"Mean arm rewards: {arm_cumulative_rewards/arm_counts}")

# Plot rewards and regret
all_episode_rewards = np.array(all_episode_rewards)
plt.figure()
plt.plot(np.cumsum(all_episode_rewards))
plt.title("Cumulative Rewards")
plt.figure()
plt.plot(np.cumsum(200.0 - all_episode_rewards))  # maximum reward in CartPole-v0 is 200.0
plt.title("Regret")
plt.show()
```

```
Arms: ['DQN', 'DoubleDQN', 'DuelingDQN', 'DoubleDuelingDQN', 'REINFORCE']
Counts: [  1.   2. 193.   3.   1.]
Mean arm rewards: [146.         152.         172.97927461 147.          88.        ]
```





Question 3.2: Is UCB well-suited for online model-selection?

At every round of UCB, one algorithm is selected and trained for a few timesteps. Does this break any assumption made by the UCB algorithm? If so, how would you handle this issue?

In order to estimate a confidence interval for r(a), the distributions ($\nu(a)$) from which the sample must be stationnary. Indeed, if the law change over the time, we don't necessarily have $\mathbb{E}[\hat{r}_t(a)] = r(a)$ and so we can't use correctly the Hoeffding inequality.

When we train an algorithm, we change the distribution before we draw a reward. So the mean r(a) will move over the time, that's why UCB may have sub optimal perfomance on this situation.

For handle this issue, we can try to use Thompson Sampling. Indeed, in this alogrithm we don't try to build an confidence interval, we use an a posteriori distribution that we udpate after each step.

In [27]: