In [ ]:
```
%load_ext autoreload
%autoreload 2
```

Please find all the files in this google drive: https://drive.google.com/drive/folders/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4

Also all the required python files are attached at the end of this notebook as a code cell.

In [ ]:
```
!pip install swig
!pip install gymnasium
!pip install Box2D gymnasium[box2d]
!pip3 install box2d box2d-kengz
```

```
Collecting swig
  Downloading swig-4.2.1-py2.py3-none-manylinux_2_5_x86_64.manylinux1_x86_64
.whl (1.9 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.9/1.9 MB 29.0 MB/s eta 0:00
:00
Installing collected packages: swig
Successfully installed swig-4.2.1
Collecting gymnasium
  Downloading gymnasium-0.29.1-py3-none-any.whl (953 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 953.9/953.9 kB 16.2 MB/s eta
0:00:00
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/di
st-packages (from gymnasium) (1.25.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.
10/dist-packages (from gymnasium) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/py
thon3.10/dist-packages (from gymnasium) (4.12.1)
Collecting farama-notifications>=0.0.1 (from gymnasium)
  Downloading Farama_Notifications-0.0.4-py3-none-any.whl (2.5 kB)
Installing collected packages: farama-notifications, gymnasium
Successfully installed farama-notifications-0.0.4 gymnasium-0.29.1
Collecting Box2D
  Downloading Box2D-2.3.2.tar.gz (427 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 427.9/427.9 kB 8.6 MB/s eta 0
:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: gymnasium[box2d] in /usr/local/lib/python3.10
/dist-packages (0.29.1)
Requirement already satisfied: numpy>=1.21.0 in /usr/local/lib/python3.10/di
st-packages (from gymnasium[box2d]) (1.25.2)
Requirement already satisfied: cloudpickle>=1.2.0 in /usr/local/lib/python3.
10/dist-packages (from gymnasium[box2d]) (2.2.1)
Requirement already satisfied: typing-extensions>=4.3.0 in /usr/local/lib/py
thon3.10/dist-packages (from gymnasium[box2d]) (4.12.1)
Requirement already satisfied: farama-notifications>=0.0.1 in /usr/local/lib
/python3.10/dist-packages (from gymnasium[box2d]) (0.0.4)
Collecting box2d-py==2.3.5 (from gymnasium[box2d])
```

```
  Downloading box2d-py-2.3.5.tar.gz (374 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 374.4/374.4 kB 32.0 MB/s eta
0:00:00
  Preparing metadata (setup.py) ... done
Requirement already satisfied: pygame>=2.1.3 in /usr/local/lib/python3.10/di
st-packages (from gymnasium[box2d]) (2.5.2)
Requirement already satisfied: swig==4.* in /usr/local/lib/python3.10/dist-p
ackages (from gymnasium[box2d]) (4.2.1)
Building wheels for collected packages: Box2D, box2d-py
  Building wheel for Box2D (setup.py) ... done
  Created wheel for Box2D: filename=Box2D-2.3.2-cp310-cp310-linux_x86_64.whl
size=2394337 sha256=dd596e500afa6c1b5d8a41c45b9047f18887deeabbca023e1e075232
5331d94c
  Stored in directory: /root/.cache/pip/wheels/eb/cb/be/e663f3ce9aba6580611c
0febaf7cd3cf7603f87047de2a52f9
  Building wheel for box2d-py (setup.py) ... done
  Created wheel for box2d-py: filename=box2d_py-2.3.5-cp310-cp310-linux_x86_
64.whl size=2376104 sha256=28955fd61fc51685a4ae1bef6934976176f7c2f92c3eb1739
8cc59becbb93a50
  Stored in directory: /root/.cache/pip/wheels/db/8f/6a/eaaadf056fba10a98d98
6f6dce954e6201ba3126926fc5ad9e
Successfully built Box2D box2d-py
Installing collected packages: box2d-py, Box2D
Successfully installed Box2D-2.3.2 box2d-py-2.3.5
Requirement already satisfied: box2d in /usr/local/lib/python3.10/dist-packa
ges (2.3.2)
Collecting box2d-kengz
  Downloading Box2D-kengz-2.3.3.tar.gz (425 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 425.4/425.4 kB 9.2 MB/s eta 0
:00:00
  Preparing metadata (setup.py) ... done
Building wheels for collected packages: box2d-kengz
  Building wheel for box2d-kengz (setup.py) ... done
  Created wheel for box2d-kengz: filename=Box2D_kengz-2.3.3-cp310-cp310-linu
x_x86_64.whl size=2394321 sha256=69ca48f8d16109ed5bfabc97a09b7124bc3f2a26ce3
4d0b7ed724ed1417702cf
  Stored in directory: /root/.cache/pip/wheels/ab/a3/5f/6396406aa0163da86c2a
8d28304a120b55cfa98363654d853b
Successfully built box2d-kengz
Installing collected packages: box2d-kengz
Successfully installed box2d-kengz-2.3.3
```

In [ ]:

```
!pip install numpy torch wandb matplotlib termcolor
```

```
Requirement already satisfied: numpy in /usr/local/lib/python3.10/dist-packa
ges (1.25.2)
Requirement already satisfied: torch in /usr/local/lib/python3.10/dist-packa
ges (2.3.0+cu121)
Collecting wandb
  Downloading wandb-0.17.1-py3-none-manylinux_2_5_x86_64.manylinux1_x86_64.m
anylinux_2_17_x86_64.manylinux2014_x86_64.whl (6.8 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 6.8/6.8 MB 57.9 MB/s eta 0:00
:00
```

```
Requirement already satisfied: matplotlib in /usr/local/lib/python3.10/dist-
packages (3.7.1)
Requirement already satisfied: termcolor in /usr/local/lib/python3.10/dist-p
ackages (2.4.0)
Requirement already satisfied: filelock in /usr/local/lib/python3.10/dist-pa
ckages (from torch) (3.14.0)
Requirement already satisfied: typing-extensions>=4.8.0 in /usr/local/lib/py
thon3.10/dist-packages (from torch) (4.12.1)
Requirement already satisfied: sympy in /usr/local/lib/python3.10/dist-packa
ges (from torch) (1.12.1)
Requirement already satisfied: networkx in /usr/local/lib/python3.10/dist-pa
ckages (from torch) (3.3)
Requirement already satisfied: jinja2 in /usr/local/lib/python3.10/dist-pack
ages (from torch) (3.1.4)
Requirement already satisfied: fsspec in /usr/local/lib/python3.10/dist-pack
ages (from torch) (2023.6.0)
Collecting nvidia-cuda-nvrtc-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_nvrtc_cu12-12.1.105-py3-none-manylinux1_x86_64.wh
l (23.7 MB)
Collecting nvidia-cuda-runtime-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_runtime_cu12-12.1.105-py3-none-manylinux1_x86_64.
whl (823 kB)
Collecting nvidia-cuda-cupti-cu12==12.1.105 (from torch)
  Using cached nvidia_cuda_cupti_cu12-12.1.105-py3-none-manylinux1_x86_64.wh
l (14.1 MB)
Collecting nvidia-cudnn-cu12==8.9.2.26 (from torch)
  Using cached nvidia_cudnn_cu12-8.9.2.26-py3-none-manylinux1_x86_64.whl (73
1.7 MB)
Collecting nvidia-cublas-cu12==12.1.3.1 (from torch)
  Using cached nvidia_cublas_cu12-12.1.3.1-py3-none-manylinux1_x86_64.whl (4
10.6 MB)
Collecting nvidia-cufft-cu12==11.0.2.54 (from torch)
  Using cached nvidia_cufft_cu12-11.0.2.54-py3-none-manylinux1_x86_64.whl (1
21.6 MB)
Collecting nvidia-curand-cu12==10.3.2.106 (from torch)
  Using cached nvidia_curand_cu12-10.3.2.106-py3-none-manylinux1_x86_64.whl
(56.5 MB)
Collecting nvidia-cusolver-cu12==11.4.5.107 (from torch)
  Using cached nvidia_cusolver_cu12-11.4.5.107-py3-none-manylinux1_x86_64.wh
l (124.2 MB)
Collecting nvidia-cusparse-cu12==12.1.0.106 (from torch)
  Using cached nvidia_cusparse_cu12-12.1.0.106-py3-none-manylinux1_x86_64.wh
l (196.0 MB)
Collecting nvidia-nccl-cu12==2.20.5 (from torch)
  Using cached nvidia_nccl_cu12-2.20.5-py3-none-manylinux2014_x86_64.whl (17
6.2 MB)
Collecting nvidia-nvtx-cu12==12.1.105 (from torch)
  Using cached nvidia_nvtx_cu12-12.1.105-py3-none-manylinux1_x86_64.whl (99
kB)
Requirement already satisfied: triton==2.3.0 in /usr/local/lib/python3.10/di
st-packages (from torch) (2.3.0)
Collecting nvidia-nvjitlink-cu12 (from nvidia-cusolver-cu12==11.4.5.107->tor
ch)
```

```
  Downloading nvidia_nvjitlink_cu12-12.5.40-py3-none-manylinux2014_x86_64.wh
l (21.3 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 21.3/21.3 MB 74.0 MB/s eta 0:
00:00
Requirement already satisfied: click!=8.0.0,>=7.1 in /usr/local/lib/python3.
10/dist-packages (from wandb) (8.1.7)
Collecting docker-pycreds>=0.4.0 (from wandb)
  Downloading docker_pycreds-0.4.0-py2.py3-none-any.whl (9.0 kB)
Collecting gitpython!=3.1.29,>=1.0.0 (from wandb)
  Downloading GitPython-3.1.43-py3-none-any.whl (207 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 207.3/207.3 kB 29.0 MB/s eta
0:00:00
Requirement already satisfied: platformdirs in /usr/local/lib/python3.10/dis
t-packages (from wandb) (4.2.2)
Requirement already satisfied: protobuf!=4.21.0,<6,>=3.19.0 in /usr/local/li
b/python3.10/dist-packages (from wandb) (3.20.3)
Requirement already satisfied: psutil>=5.0.0 in /usr/local/lib/python3.10/di
st-packages (from wandb) (5.9.5)
Requirement already satisfied: pyyaml in /usr/local/lib/python3.10/dist-pack
ages (from wandb) (6.0.1)
Requirement already satisfied: requests<3,>=2.0.0 in /usr/local/lib/python3.
10/dist-packages (from wandb) (2.31.0)
Collecting sentry-sdk>=1.0.0 (from wandb)
  Downloading sentry_sdk-2.5.1-py2.py3-none-any.whl (289 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 289.6/289.6 kB 37.4 MB/s eta
0:00:00
Collecting setproctitle (from wandb)
  Downloading setproctitle-1.3.3-cp310-cp310-manylinux_2_5_x86_64.manylinux1
_x86_64.manylinux_2_17_x86_64.manylinux2014_x86_64.whl (30 kB)
Requirement already satisfied: setuptools in /usr/local/lib/python3.10/dist-
packages (from wandb) (67.7.2)
Requirement already satisfied: contourpy>=1.0.1 in /usr/local/lib/python3.10
/dist-packages (from matplotlib) (1.2.1)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.10/dis
t-packages (from matplotlib) (0.12.1)
Requirement already satisfied: fonttools>=4.22.0 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (4.53.0)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.1
0/dist-packages (from matplotlib) (1.4.5)
Requirement already satisfied: packaging>=20.0 in /usr/local/lib/python3.10/
dist-packages (from matplotlib) (24.0)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.10/di
st-packages (from matplotlib) (9.4.0)
Requirement already satisfied: pyparsing>=2.3.1 in /usr/local/lib/python3.10
/dist-packages (from matplotlib) (3.1.2)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python
3.10/dist-packages (from matplotlib) (2.8.2)
Requirement already satisfied: six>=1.4.0 in /usr/local/lib/python3.10/dist-
packages (from docker-pycreds>=0.4.0->wandb) (1.16.0)
Collecting gitdb<5,>=4.0.1 (from gitpython!=3.1.29,>=1.0.0->wandb)
  Downloading gitdb-4.0.11-py3-none-any.whl (62 kB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 62.7/62.7 kB 9.0 MB/s eta 0:0
0:00
```

```
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/py
thon3.10/dist-packages (from requests<3,>=2.0.0->wandb) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dis
t-packages (from requests<3,>=2.0.0->wandb) (3.7)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.
10/dist-packages (from requests<3,>=2.0.0->wandb) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.
10/dist-packages (from requests<3,>=2.0.0->wandb) (2024.6.2)
Requirement already satisfied: MarkupSafe>=2.0 in /usr/local/lib/python3.10/
dist-packages (from jinja2->torch) (2.1.5)
Requirement already satisfied: mpmath<1.4.0,>=1.1.0 in /usr/local/lib/python
3.10/dist-packages (from sympy->torch) (1.3.0)
Collecting smmap<6,>=3.0.1 (from gitdb<5,>=4.0.1->gitpython!=3.1.29,>=1.0.0-
>wandb)
  Downloading smmap-5.0.1-py3-none-any.whl (24 kB)
Installing collected packages: smmap, setproctitle, sentry-sdk, nvidia-nvtx-
cu12, nvidia-nvjitlink-cu12, nvidia-nccl-cu12, nvidia-curand-cu12, nvidia-cu
fft-cu12, nvidia-cuda-runtime-cu12, nvidia-cuda-nvrtc-cu12, nvidia-cuda-cupt
i-cu12, nvidia-cublas-cu12, docker-pycreds, nvidia-cusparse-cu12, nvidia-cud
nn-cu12, gitdb, nvidia-cusolver-cu12, gitpython, wandb
Successfully installed docker-pycreds-0.4.0 gitdb-4.0.11 gitpython-3.1.43 nv
idia-cublas-cu12-12.1.3.1 nvidia-cuda-cupti-cu12-12.1.105 nvidia-cuda-nvrtc-
cu12-12.1.105 nvidia-cuda-runtime-cu12-12.1.105 nvidia-cudnn-cu12-8.9.2.26 n
vidia-cufft-cu12-11.0.2.54 nvidia-curand-cu12-10.3.2.106 nvidia-cusolver-cu1
2-11.4.5.107 nvidia-cusparse-cu12-12.1.0.106 nvidia-nccl-cu12-2.20.5 nvidia-
nvjitlink-cu12-12.5.40 nvidia-nvtx-cu12-12.1.105 sentry-sdk-2.5.1 setproctit
le-1.3.3 smmap-5.0.1 wandb-0.17.1
```

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')

# TODO: Enter the foldername in your Drive where you have saved the unzipped
# project folder, e.g. '239AS.3/project1/gan'
FOLDERNAME = "RL-part1"
assert FOLDERNAME is not None, "[!] Enter the foldername."

# Now that we've mounted your Drive, this ensures that
# the Python interpreter of the Colab VM can load
# python files from within it.
import sys
sys.path.append('/content/drive/My Drive/{}'.format(FOLDERNAME))
%cd /content/drive/My\ Drive/RL-part1
%ls
```

```
Mounted at /content/drive
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1
DDPG.png         __pycache__/         test_outputs.pt                    tes
t_weights.pt
DQN.png          replay_buffer.py     test_replay_buffer_inputs_mac.pkl   tes
ty.py
DQN.py           rl.ipynb             test_replay_buffer_inputs.pkl       uti
ls.py
env_wrapper.py   runs/                test_replay_buffer_samples_mac.pth
model.py         test_outputs_mac.pt  test_replay_buffer_samples.pth
```

In [ ]:

```
import test
from utils import *
```

Reinforcement Learning Part 1: DQN

By Lawrence Liu and Tonmoy Monsoor

Some General Instructions

- As before, please keep the names of the layer consistent with what is requested in model.py. Otherwise the test functions will not work
- You will need to fill in the model.py, the DQN.py file, the buffer.py file, and the env_wrapper.py

DO NOT use Windows for this project, gymnasium does is not supported for windows and installing it will be a pain.

Introduction to the Enviroment

We will be training a DQN agent to play the game of CarRacing. The agent will be trained to play the game using the pixels of the game as an input. The reward structure is as follows for each frame:

- -0.1 for each frame
- +1000/N where N is the number of tiles visited by the car in the episode

The overall goal of this game is to design a agent that is able to play the game with a average test score of above 600. In discrete mode the actions can take 5 actions,

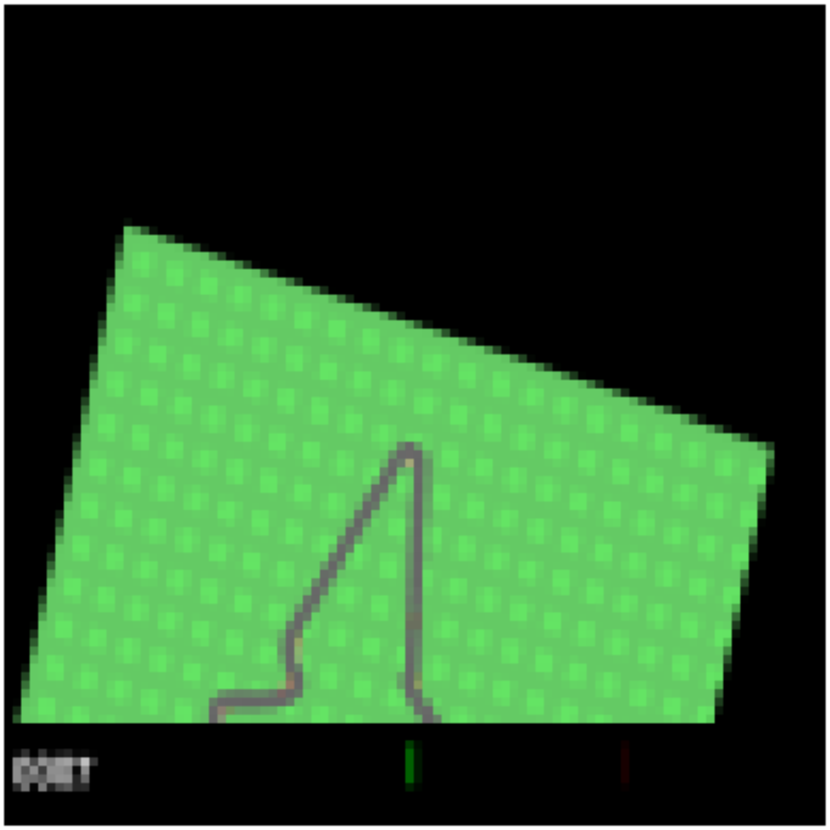
- 0: Do Nothing
- 1: Turn Left
- 2: Turn Right
- 3: Accelerate
- 4: Brake

First let us visualize the game and understand the environment.

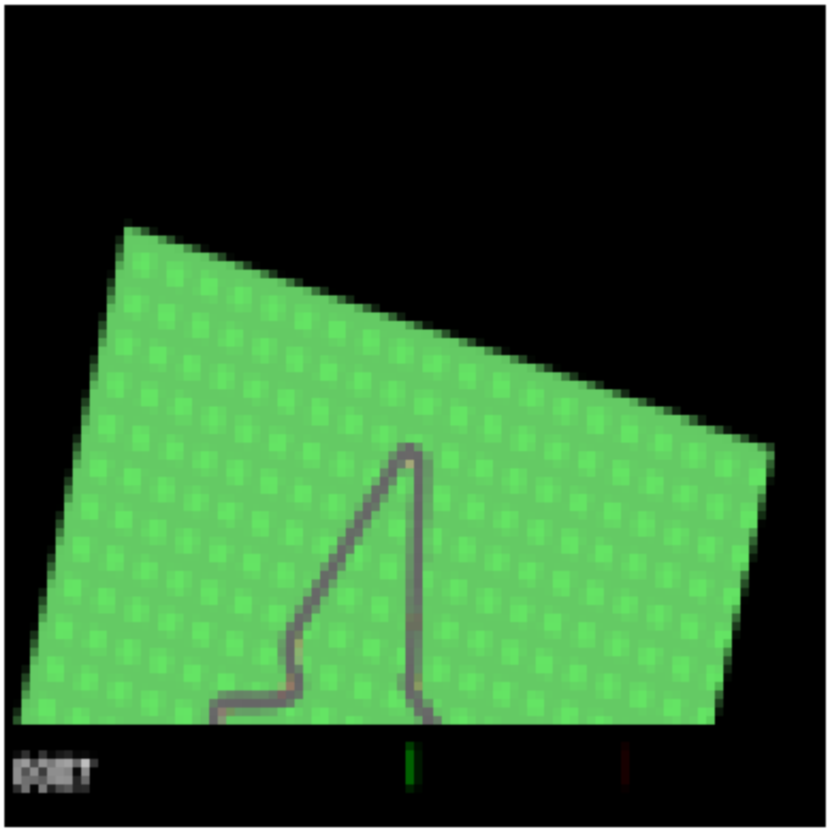
In [ ]:
```
import gymnasium as gym
import numpy as np
env = gym.make('CarRacing-v2', continuous=False, render_mode='rgb_array')
env.np_random = np.random.RandomState(42)
```

In [ ]:
```
from IPython.display import HTML

frames = []
s, _ = env.reset()

while True:
    a = env.action_space.sample()
    s, r, terminated, truncated, _ = env.step(a)
    frames.append(s)
    if terminated or truncated:
        break


anim = animate(frames)
HTML(anim.to_jshtml())
```

Out[ ]:



Once Loop Reflect



So a couple things we can note:

- at the beginning of the game, we have 50 frames of the game slowly zooming into the car, we should ignore this period, ie no-op during this period.
- there is a black bar at the bottom of the screen, we should crop this out of the observation.

In addition, another thing to note is that the current frame doesn't give much information about the velocity and acceleration of the car, and that the car does not move much for each frame.

Environment Wrapper (5 points)

As a result, you will need to complete `EnvWrapper` in `env_wrapper.py`. You can find more information in the docstring for the wrapper, however the main idea is that it is a wrapper to the environment that does the following:

- skips the first 50 frames of the game
- crops out the black bar and reshapes the observation to a 84x84 image, as well as turning the resulting image to grayscale
- performs the actions for `skip_frames` frames
- stacks the last `num_frames` frames together to give the agent some information about the velocity and acceleration of the car.

In [ ]:

```
from env_wrapper import EnvWrapper
import testy

testy.test_wrapper(EnvWrapper)
```

```
Passed reset
Passed step
```

CNN Model (5 points)

Now we are ready to build the model. Our architecture of the CNN model is the one proposed by Mnih et al in "Human-level control through deep reinforcement learning". Specifically this consists of the following layers:

- A convolutional layer with 32 filters of size 8x8 with stride 4 and relu activation
- A convolutional layer with 64 filters of size 4x4 with stride 2 and relu activation
- A convolutional layer with 64 filters of size 3x3 with stride 1 and relu activation
- A fully connected layer with 512 units and relu activation
- A fully connected layer with the number of outputs of the environment

Please implement this model `Nature_Paper_Conv` in `model.py` as well as the helper `MLP` class.

In [ ]:
```
import model
testy.test_model_DQN(model.Nature_Paper_Conv)
```

```
Passed
```

DQN (40 points)

Now we are ready to implement the DQN algorithm.

title

Replay Buffer (5 points)

First start by implementing the DQN replay buffer `ReplayBufferDQN` in `buffer.py`. This buffer will store the transitions of the agent and sample them for training.

In [ ]:
```
from replay_buffer import ReplayBufferDQN

testy.test_DQN_replay_buffer(ReplayBufferDQN)
```

```
Passed
```

DQN (15 points)

Now implement the `_optimize_model` and `sample_action` functions in `DQN` in `DQN.py`. The `_optimize_model` function will sample a batch of transitions from the replay buffer and update the model. The `sample_action` function will sample an action from the model given the current state. Train the model over 200 episdoes, validating every 50 episodes for 30 episodes, before testing the model for 50 episodes at the end.

In [ ]:

```
import DQN
import utils
import torch


trainerDQN = DQN.DQN(EnvWrapper(env),
                model.Nature_Paper_Conv,
                lr = 0.00025,
                gamma = 0.95,
                buffer_size=100000,
                batch_size=32,
                loss_fn = "mse_loss",
                use_wandb = False,
                device = 'cpu',
                seed = 42,
                epsilon_scheduler = utils.exponential_decay(1, 700,0.1),
                save_path = utils.get_save_path("DQN","./runs/"))

trainerDQN.train(200,50,30,50,50)
```

```
saving to ./runs/DQN/run2
```

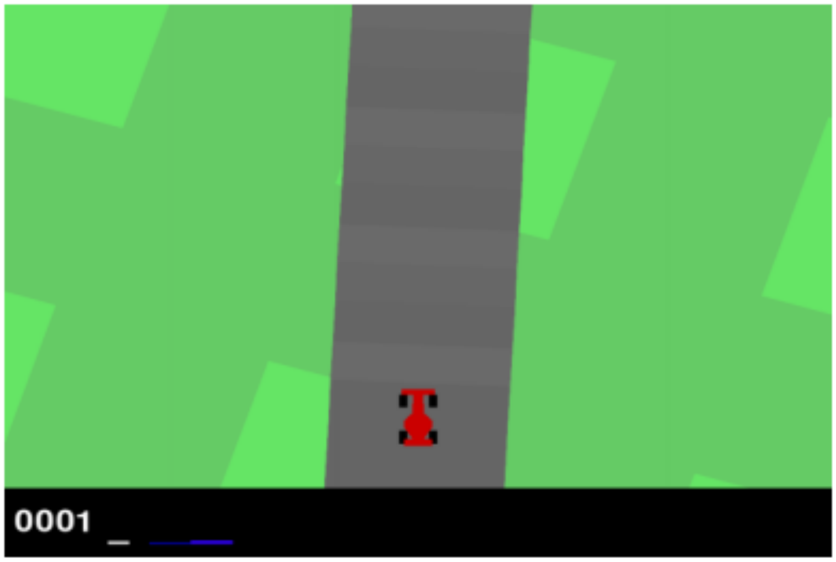
```
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:145: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  states = torch.tensor(states, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:146: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  actions = torch.tensor(actions, dtype=torch.int64).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:147: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  rewards = torch.tensor(rewards, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:148: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  next_states = torch.tensor(next_states, dtype=torch.float32).to(self.devic
e)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:149: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  dones = torch.tensor(dones, dtype=torch.float32).to(self.device)
Episode: 1: Time: 14.370965719223022 Total Reward: -47.55474452554785 Avg_Lo
ss: 0.652843650976184
```

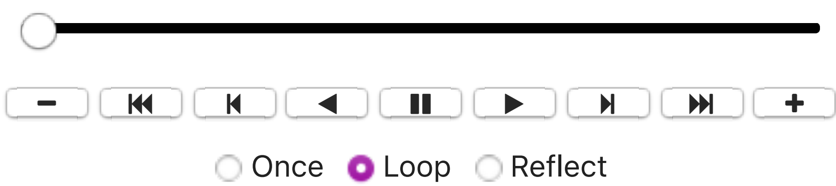
Please include a plot of the training and validation rewards over the episodes in the report. An additional question to answer is does the loss matter in DQN? Why or why not?

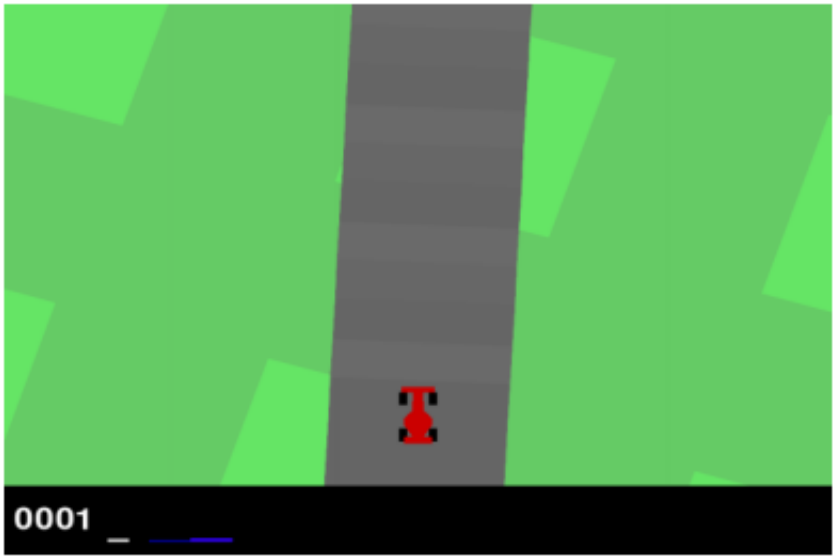
We can also draw a animation of the car in one game, the code is provided below

In [ ]:
```python
eval_env = gym.make('CarRacing-v2', continuous=True, render_mode='rgb_array'
eval_env = EnvWrapper(eval_env)

total_rewards, frames = trainerDQN.play_episode(0,True,42)
anim = animate(frames)
HTML(anim.to_jshtml())
```

Out[ ]:







Double DQN

In the original paper, where the algorithim is shown above, the estimated target Q value was computed using the current Q network's weights. However, this can lead to overestimation of the Q values. To mitigate this, we can use the target network to compute the target Q value. This is known as Double DQN.

Hard updating Target Network (5 points)

Original implementations for this involved hard updates, where the model weights were copied to the target network every C steps. This is known as hard updating. This was what was used in the Nature Paper by Mnih et al 2015 "Human-level control through deep reinforcement learning"

Please implement this by implementing the `_optimize_model` and `_update_model` classes in `HardUpdateDQN` in `DQN.py`.

In [ ]:
```
import DQN
import utils
import torch
trainerHardUpdateDQN = DQN.HardUpdateDQN(EnvWrapper(env),
                model.Nature_Paper_Conv,
                update_freq = 100,
                lr = 0.00025,
                gamma = 0.95,
                buffer_size=100000,
                batch_size=32,
                loss_fn = "mse_loss",
                use_wandb = False,
                device = 'cuda',
                seed = 42,
                epsilon_scheduler = utils.exponential_decay(1, 1000,0.1),
                save_path = utils.get_save_path("DoubleDQN_HardUpdates/","./

trainerHardUpdateDQN.train(200,50,30,50,50)
```

```
saving to ./runs/DoubleDQN_HardUpdates/run1
```

```
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:286: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  states = torch.tensor(states, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:287: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  actions = torch.tensor(actions, dtype=torch.long).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:288: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  rewards = torch.tensor(rewards, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:289: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  next_states = torch.tensor(next_states, dtype=torch.float32).to(self.devic
e)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:290: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  dones = torch.tensor(dones, dtype=torch.float32).to(self.device)
/usr/local/lib/python3.10/dist-packages/torch/autograd/graph.py:744: UserWar
ning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRI
PTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORT
ED (Triggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
  return Variable._execution_engine.run_backward(  # Calls into the C++ engi
ne to run the backward pass
```

```
Episode: 0: Time: 12.483212947845459 Total Reward: -56.53846153846219 Avg_Lo
ss: 0.5919263863909072
Episode: 1: Time: 11.804194927215576 Total Reward: -22.00729927007294 Avg_Lo
ss: 0.6516952832315525
Episode: 2: Time: 12.010326385498047 Total Reward: -37.857142857143046 Avg_L
oss: 0.7759091348363822
Episode: 3: Time: 12.128596782684326 Total Reward: -51.22895622895655 Avg_L
oss: 0.7332356081132879
Episode: 4: Time: 11.976420164108276 Total Reward: -42.183098591549474 Avg_L
oss: 0.7145520722290047
Episode: 5: Time: 11.567717552185059 Total Reward: -75.91603053435122 Avg_Lo
ss: 0.6762210027072109
Episode: 6: Time: 12.334421873092651 Total Reward: 26.093749999999822 Avg_Lo
ss: 0.7325921464054024
Episode: 7: Time: 12.003796339035034 Total Reward: -22.536231884058626 Avg_L
oss: 0.8106680700101522
Episode: 8: Time: 11.861407279968262 Total Reward: 64.09090909090939 Avg_Los
s: 0.8794658998606586
Episode: 9: Time: 11.816045999526978 Total Reward: -39.44444444444515 Avg_Lo
ss: 0.8989998481291182
Episode: 10: Time: 12.185171365737915 Total Reward: 13.303249097472433 Avg_L
oss: 0.9273974251534257
Episode: 11: Time: 12.108824253082275 Total Reward: 104.39577039275369 Avg_L
oss: 1.0544649159707944
Episode: 12: Time: 12.325978994369507 Total Reward: 314.5563139931655 Avg_Lo
ss: 1.4127519319788748
Episode: 13: Time: 12.080240726470947 Total Reward: 138.89830508475004 Avg_L
oss: 1.6522956032212042
Episode: 14: Time: 12.447516918182373 Total Reward: 268.3440514469425 Avg_Lo
ss: 2.1318290634315553
Episode: 15: Time: 12.175309896469116 Total Reward: 406.75438596490494 Avg_L
oss: 2.9057347704382503
Episode: 16: Time: 12.200932741165161 Total Reward: 98.2203389830554 Avg_Los
s: 4.382248277423763
Episode: 17: Time: 12.47618579864502 Total Reward: 160.45171339564297 Avg_Lo
ss: 3.783542493812176
Episode: 18: Time: 13.270373106002808 Total Reward: -17.330097087378824 Avg_
Loss: 3.865881743050423
Episode: 19: Time: 12.415725231170654 Total Reward: 7.11267605633819 Avg_Los
s: 4.207663669806569
Episode: 20: Time: 12.030667066574097 Total Reward: 50.270270270272746 Avg_L
oss: 3.686150815807471
Episode: 21: Time: 11.943847417831421 Total Reward: 334.078014184391 Avg_Los
s: 4.049149809264335
Episode: 22: Time: 12.175375938415527 Total Reward: 284.92831541217845 Avg_L
oss: 4.582025950696288
Episode: 23: Time: 12.388070583343506 Total Reward: -11.666666666667012 Avg_
Loss: 4.921459125871418
Episode: 24: Time: 11.960455894470215 Total Reward: 14.890109890108864 Avg_L
oss: 4.834029207710459
Episode: 25: Time: 12.18843412399292 Total Reward: 279.1258741258687 Avg_Los
s: 5.217546368847374
Episode: 26: Time: 11.963610887527466 Total Reward: -4.090909090909724 Avg_L
```

```
oss: 4.53769926013065
Episode: 27: Time: 12.531288623809814 Total Reward: 337.60188087773645 Avg_L
oss: 5.154741105913114
Episode: 28: Time: 12.0895676612854 Total Reward: 327.53521126759813 Avg_Los
s: 5.324493226884794
Episode: 29: Time: 11.998752117156982 Total Reward: 206.07526881720827 Avg_L
oss: 4.869236041016939
Episode: 30: Time: 12.344446897506714 Total Reward: 176.87500000000156 Avg_L
oss: 4.6965322043715405
Episode: 31: Time: 12.196071147918701 Total Reward: 315.6463878326937 Avg_Lo
ss: 5.410751762510348
Episode: 32: Time: 12.406558513641357 Total Reward: -0.2287581699342236 Avg_
Loss: 5.390877688632292
Episode: 33: Time: 12.185724020004272 Total Reward: 85.37974683544682 Avg_Lo
ss: 6.075247321810041
Episode: 34: Time: 12.492741584777832 Total Reward: 232.04402515723672 Avg_L
oss: 5.093525742783266
Episode: 35: Time: 11.676784753799438 Total Reward: 81.0797342192732 Avg_Los
s: 5.43237808622232
Episode: 36: Time: 12.337978601455688 Total Reward: 48.30218068535707 Avg_Lo
ss: 5.517419554606206
Episode: 37: Time: 11.399448871612549 Total Reward: 76.64179104477964 Avg_Lo
ss: 5.578403711819849
Episode: 38: Time: 12.538520574569702 Total Reward: 256.26582278481357 Avg_L
oss: 5.7460525516702345
Episode: 39: Time: 12.1163010597229 Total Reward: 145.0000000000044 Avg_Loss
: 5.5109235644340515
Episode: 40: Time: 11.844699382781982 Total Reward: 256.1450381679349 Avg_Lo
ss: 5.856837798567379
Episode: 41: Time: 11.665568351745605 Total Reward: 20.131578947369754 Avg_L
oss: 5.23978958610727
Episode: 42: Time: 12.095415353775024 Total Reward: 406.8181818181755 Avg_Lo
ss: 5.184530964418619
Episode: 43: Time: 12.418891668319702 Total Reward: 115.16949152542804 Avg_L
oss: 5.782975119702956
Episode: 44: Time: 12.434400796890259 Total Reward: 213.21917808219425 Avg_L
oss: 5.45476659506309
Episode: 45: Time: 11.742282390594482 Total Reward: 180.00000000000492 Avg_L
oss: 5.277183973488688
Episode: 46: Time: 12.382438659667969 Total Reward: 272.41214057508284 Avg_L
oss: 5.424823153920534
Episode: 47: Time: 11.836745977401733 Total Reward: 238.33333333333272 Avg_L
oss: 5.163564128535135
Episode: 48: Time: 11.930283069610596 Total Reward: 160.10204081633043 Avg_L
oss: 5.646764411645777
Episode: 49: Time: 12.152179718017578 Total Reward: 214.0909090909128 Avg_Lo
ss: 5.364926977317874
Validation Mean Reward: 94.51923294716951 Validation Std Reward: 141.9792051
2470532
Episode: 50: Time: 12.651035785675049 Total Reward: 186.34556574923732 Avg_L
oss: 5.607671385051823
Episode: 51: Time: 12.877513408660889 Total Reward: 326.7687074829903 Avg_Lo
ss: 5.278493011699004
```

```
Episode: 52: Time: 12.385499954223633 Total Reward: 270.5913978494547 Avg_Lo
ss: 4.9219321262936635
Episode: 53: Time: 12.08417010307312 Total Reward: 324.11764705882155 Avg_Lo
ss: 4.976297555851335
Episode: 54: Time: 11.612911462783813 Total Reward: 160.39568345324145 Avg_L
oss: 4.783621653288352
Episode: 55: Time: 11.646110773086548 Total Reward: 354.3927125506047 Avg_Lo
ss: 5.064892107198219
Episode: 56: Time: 11.833792686462402 Total Reward: 332.56183745583024 Avg_L
oss: 5.597269564115701
Episode: 57: Time: 12.050487041473389 Total Reward: 240.7400722021702 Avg_Lo
ss: 5.1245051422039
Episode: 58: Time: 11.740086793899536 Total Reward: 479.21874999999176 Avg_L
oss: 5.114761797820821
Episode: 59: Time: 11.798572540283203 Total Reward: 395.8424908424831 Avg_Lo
ss: 5.1430206704540415
Episode: 60: Time: 12.279170274734497 Total Reward: 234.0322580645198 Avg_Lo
ss: 5.2965726662082835
Episode: 61: Time: 11.970997095108032 Total Reward: 125.00000000000153 Avg_L
oss: 6.066002494146844
Episode: 62: Time: 11.86415433883667 Total Reward: 124.08127208480991 Avg_Lo
ss: 5.579110115516086
Episode: 63: Time: 11.836926937103271 Total Reward: 317.75167785234606 Avg_L
oss: 5.4311231695303395
Episode: 64: Time: 12.258193969726562 Total Reward: 423.987341772145 Avg_Los
s: 5.279319186170562
Episode: 65: Time: 12.50801396369934 Total Reward: 195.59829059829408 Avg_Lo
ss: 5.6838392790626076
Episode: 66: Time: 11.908565759658813 Total Reward: 98.22033898305364 Avg_Lo
ss: 5.522494936189732
Episode: 67: Time: 12.030453443527222 Total Reward: 435.24911032027507 Avg_L
oss: 5.359964382748644
Episode: 68: Time: 11.552335500717163 Total Reward: 194.79591836735065 Avg_L
oss: 6.055685710506279
Episode: 69: Time: 11.823281526565552 Total Reward: 247.20532319392098 Avg_L
oss: 5.781822680425243
Episode: 70: Time: 12.048629999160767 Total Reward: 368.76811594202184 Avg_L
oss: 6.133119755432386
Episode: 71: Time: 12.35136604309082 Total Reward: 271.77115987460616 Avg_Lo
ss: 6.090392229937706
Episode: 72: Time: 11.856501579284668 Total Reward: 216.89710610932815 Avg_L
oss: 5.780949051640615
Episode: 73: Time: 12.05949592590332 Total Reward: 425.83333333332604 Avg_Lo
ss: 6.230280032678812
Episode: 74: Time: 12.308712720870972 Total Reward: 282.16262975777875 Avg_L
oss: 5.741038339478629
Episode: 75: Time: 12.395341873168945 Total Reward: 286.2500000000025 Avg_Lo
ss: 5.758023885618739
Episode: 76: Time: 12.352905511856079 Total Reward: 239.42622950820106 Avg_L
oss: 5.871248199158356
Episode: 77: Time: 12.058568477630615 Total Reward: 277.41379310345087 Avg_L
oss: 6.142159157440442
Episode: 78: Time: 12.26275086402893 Total Reward: 171.66666666667112 Avg_Lo
```

```
ss: 5.817463604342036
Episode: 79: Time: 12.351865768432617 Total Reward: 283.4615384615405 Avg_Lo
ss: 5.55361101356875
Episode: 80: Time: 12.327369213104248 Total Reward: 286.4102564102554 Avg_Lo
ss: 5.28118997161128
Episode: 81: Time: 12.065058946609497 Total Reward: 352.18309859154505 Avg_L
oss: 5.363618305751255
Episode: 82: Time: 11.907482624053955 Total Reward: 165.00000000000347 Avg_L
oss: 5.08543468723778
Episode: 83: Time: 12.680445671081543 Total Reward: 183.93175074184387 Avg_L
oss: 5.599372505139904
Episode: 84: Time: 11.921599626541138 Total Reward: 342.50000000000085 Avg_L
oss: 5.65535684092706
Episode: 85: Time: 12.194464921951294 Total Reward: 416.4006514657896 Avg_Lo
ss: 5.356447677652375
Episode: 86: Time: 11.811753273010254 Total Reward: 83.947368421057 Avg_Loss
: 5.390767270777406
Episode: 87: Time: 11.818120002746582 Total Reward: -7.337662337662572 Avg_L
oss: 5.631481031409833
Episode: 88: Time: 11.98997974395752 Total Reward: 341.0902255639066 Avg_Los
s: 5.469146004244059
Episode: 89: Time: 12.282021284103394 Total Reward: 246.69278996864657 Avg_L
oss: 5.399752892365976
Episode: 90: Time: 11.668728590011597 Total Reward: 172.44186046512044 Avg_L
oss: 4.974145977937875
Episode: 91: Time: 11.94359803199768 Total Reward: 384.8534798534736 Avg_Los
s: 5.543482835553274
Episode: 92: Time: 12.006245613098145 Total Reward: 72.8082191780868 Avg_Los
s: 5.929475485777655
Episode: 93: Time: 12.026939868927002 Total Reward: 256.97368421053 Avg_Loss
: 6.262490294560664
Episode: 94: Time: 12.005510568618774 Total Reward: 231.2411347517778 Avg_Lo
ss: 6.459338684041961
Episode: 95: Time: 12.587303161621094 Total Reward: 127.2222222222507 Avg_L
oss: 6.133984114943432
Episode: 96: Time: 11.929902076721191 Total Reward: 211.33802816901897 Avg_L
oss: 5.854353157912984
Episode: 97: Time: 12.508661985397339 Total Reward: 201.07250755287367 Avg_L
oss: 5.782535768857523
Episode: 98: Time: 12.286890745162964 Total Reward: 315.5263157894649 Avg_Lo
ss: 5.715035370418003
Episode: 99: Time: 11.939123392105103 Total Reward: 450.4545454545409 Avg_Lo
ss: 5.144919885807679
Validation Mean Reward: 463.7533795406323 Validation Std Reward: 153.3813101
262295
Episode: 100: Time: 12.272467136383057 Total Reward: 326.6027874564408 Avg_L
oss: 5.101425528526306
Episode: 101: Time: 12.17679500579834 Total Reward: 253.12286689420208 Avg_L
oss: 5.766147696671366
Episode: 102: Time: 12.16746997833252 Total Reward: 371.2162162162097 Avg_Lo
ss: 5.492533471404004
Episode: 103: Time: 12.244171619415283 Total Reward: 262.615894039733 Avg_Lo
ss: 5.723371643479131
```

```
Episode: 104: Time: 12.222379922866821 Total Reward: 279.5583038869239 Avg_L
oss: 5.855542189934674
Episode: 105: Time: 12.723163604736328 Total Reward: 195.1408450704249 Avg_L
oss: 5.776563223670511
Episode: 106: Time: 12.0330331325531 Total Reward: 224.8529411764743 Avg_Los
s: 5.831278500436735
Episode: 107: Time: 12.21212100982666 Total Reward: 313.7837837837827 Avg_Lo
ss: 6.264530649706095
Episode: 108: Time: 12.154852867126465 Total Reward: 328.6111111111091 Avg_L
oss: 6.678134520514672
Episode: 109: Time: 12.548645734786987 Total Reward: 321.90962099125164 Avg_
Loss: 5.584553974516251
Episode: 110: Time: 12.296841621398926 Total Reward: 343.3116883116863 Avg_L
oss: 5.371210058697131
Episode: 111: Time: 12.121032953262329 Total Reward: 353.27586206895955 Avg_
Loss: 5.228633499946914
Episode: 112: Time: 12.156599760055542 Total Reward: 370.4545454545397 Avg_L
oss: 5.0957207920170635
Episode: 113: Time: 12.48752498626709 Total Reward: 240.38461538461917 Avg_L
oss: 5.153971677066899
Episode: 114: Time: 12.441099166870117 Total Reward: 276.33550488599496 Avg_
Loss: 5.804369950494847
Episode: 115: Time: 12.062102317810059 Total Reward: 388.6363636363583 Avg_L
oss: 5.630708274721098
Episode: 116: Time: 12.247232913970947 Total Reward: 155.81433224756066 Avg_
Loss: 5.482904456242793
Episode: 117: Time: 12.033926963806152 Total Reward: 411.8493150684888 Avg_L
oss: 5.085632516055548
Episode: 118: Time: 11.695355892181396 Total Reward: 130.00000000000452 Avg_
Loss: 5.405903443568895
Episode: 119: Time: 12.01224946975708 Total Reward: 398.4210526315744 Avg_Lo
ss: 5.306787904571085
Episode: 120: Time: 12.502667903900146 Total Reward: 337.3529411764689 Avg_L
oss: 6.178556655134473
Episode: 121: Time: 12.555288076400757 Total Reward: 440.03184713375066 Avg_
Loss: 5.840124412244108
Episode: 122: Time: 12.493440866470337 Total Reward: 302.3063973063891 Avg_L
oss: 5.420566438626842
Episode: 123: Time: 12.086466550827026 Total Reward: 307.5157232704407 Avg_L
oss: 5.509067184283953
Episode: 124: Time: 11.955264329910278 Total Reward: 170.99326599326687 Avg_
Loss: 5.386235418439913
Episode: 125: Time: 12.324140787124634 Total Reward: 357.5993883792019 Avg_L
oss: 5.6535274942382046
Episode: 126: Time: 12.442198276519775 Total Reward: 245.764331210187 Avg_Lo
ss: 5.395383966069262
Episode: 127: Time: 12.24026107788086 Total Reward: 373.64686468646505 Avg_L
oss: 5.303250727032413
Episode: 128: Time: 11.93001413345337 Total Reward: 392.7192982456096 Avg_Lo
ss: 5.287389855424897
Episode: 129: Time: 11.96301794052124 Total Reward: 201.8197879858696 Avg_Lo
ss: 4.836403303286609
Episode: 130: Time: 12.231785297393799 Total Reward: 190.245901639347 Avg_Lo
```

```
ss: 5.28601933928097
Episode: 131: Time: 12.447749137878418 Total Reward: 258.5031847133796 Avg_L
oss: 5.161292214353545
Episode: 132: Time: 11.864003658294678 Total Reward: 332.56183745582877 Avg_
Loss: 5.284994231552637
Episode: 133: Time: 12.22459888458252 Total Reward: 334.06574394463337 Avg_L
oss: 5.419494457605507
Episode: 134: Time: 11.805724382400513 Total Reward: 279.5318352059869 Avg_L
oss: 5.725685380086177
Episode: 135: Time: 12.080742120742798 Total Reward: 397.1259842519621 Avg_L
oss: 6.405315089626472
Episode: 136: Time: 11.797214031219482 Total Reward: 410.97609561752114 Avg_
Loss: 6.464191200352516
Episode: 137: Time: 12.554653882980347 Total Reward: 344.02439024390003 Avg_
Loss: 5.961192795208523
Episode: 138: Time: 12.349662780761719 Total Reward: 257.76073619632166 Avg_
Loss: 6.277095146539833
Episode: 139: Time: 12.810569047927856 Total Reward: 174.44444444444628 Avg_
Loss: 6.139622461895983
Episode: 140: Time: 12.32645869255066 Total Reward: 360.17241379309746 Avg_L
oss: 6.421567772616859
Episode: 141: Time: 12.396170139312744 Total Reward: 351.66666666665884 Avg_
Loss: 5.910617415644541
Episode: 142: Time: 12.966662883758545 Total Reward: 310.9701492537312 Avg_L
oss: 5.94920707850897
Episode: 143: Time: 12.199249982833862 Total Reward: 396.525423728807 Avg_Lo
ss: 5.7854251470886355
Episode: 144: Time: 11.940965414047241 Total Reward: 294.3442622950775 Avg_L
oss: 5.7697597872309325
Episode: 145: Time: 12.635456323623657 Total Reward: 357.94117647058425 Avg_
Loss: 5.4870080316767975
Episode: 146: Time: 12.16890573501587 Total Reward: 360.1282051282011 Avg_Lo
ss: 5.5269946260612555
Episode: 147: Time: 12.207989931106567 Total Reward: 251.66666666667174 Avg_
Loss: 5.603958841131515
Episode: 148: Time: 12.028564929962158 Total Reward: 240.68904593639817 Avg_
Loss: 5.733748869735654
Episode: 149: Time: 11.843870401382446 Total Reward: 548.9999999999912 Avg_L
oss: 5.195738710275217
Validation Mean Reward: 323.3919729528873 Validation Std Reward: 195.1458835
452322
Episode: 150: Time: 12.413358688354492 Total Reward: 317.87878787878714 Avg_
Loss: 5.602534722881157
Episode: 151: Time: 12.373767137527466 Total Reward: 152.81341107872126 Avg_
Loss: 5.538413490567889
Episode: 152: Time: 12.057880401611328 Total Reward: 355.5494505494488 Avg_L
oss: 5.236981653365769
Episode: 153: Time: 12.412497758865356 Total Reward: 384.3103448275808 Avg_L
oss: 5.669296413910489
Episode: 154: Time: 12.524601459503174 Total Reward: 302.26027397260276 Avg_
Loss: 5.64826012308858
Episode: 155: Time: 12.462345838546753 Total Reward: 464.7014925373061 Avg_L
oss: 6.031985005410779
```

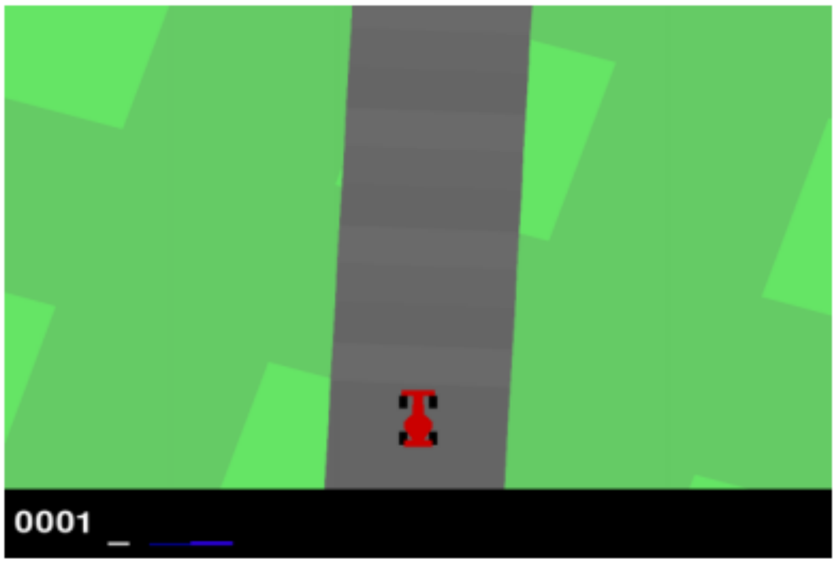
```
Episode: 156: Time: 12.791084051132202 Total Reward: 320.3846153846124 Avg_L
oss: 5.6798551443244225
Episode: 157: Time: 11.929900884628296 Total Reward: 259.6099290780181 Avg_L
oss: 5.528053525115261
Episode: 158: Time: 12.039884805679321 Total Reward: 485.88235294116856 Avg_
Loss: 5.591390869196723
Episode: 159: Time: 12.41408634185791 Total Reward: 469.0138408304417 Avg_Lo
ss: 5.333120134698243
Episode: 160: Time: 12.509939432144165 Total Reward: 320.5405405405376 Avg_L
oss: 5.973099701544818
Episode: 161: Time: 12.549989461898804 Total Reward: 367.83783783783485 Avg_
Loss: 5.853120835889287
Episode: 162: Time: 12.295385122299194 Total Reward: 371.6666666666601 Avg_L
oss: 6.252358138060369
Episode: 163: Time: 12.378257989883423 Total Reward: 316.7647058823518 Avg_L
oss: 5.912871064759102
Episode: 164: Time: 12.67497992515564 Total Reward: 268.07692307691946 Avg_L
oss: 5.622675282614572
Episode: 165: Time: 12.753308057785034 Total Reward: 285.06230529594905 Avg_
Loss: 5.407590220956242
Episode: 166: Time: 12.533183813095093 Total Reward: 372.5767918088669 Avg_L
oss: 5.683037721810221
Episode: 167: Time: 12.544712781906128 Total Reward: 343.12709030100046 Avg_
Loss: 5.437679599313175
Episode: 168: Time: 12.296372890472412 Total Reward: 363.9041095890371 Avg_L
oss: 5.38710941286648
Episode: 169: Time: 11.995514154434204 Total Reward: 453.78048780486836 Avg_
Loss: 6.39943779416445
Episode: 170: Time: 12.727645635604858 Total Reward: 329.33234421365046 Avg_
Loss: 5.6768889677624745
Episode: 171: Time: 12.248217105865479 Total Reward: 440.7142857142771 Avg_L
oss: 5.61978293166441
Episode: 172: Time: 12.005695343017578 Total Reward: 392.0848708487059 Avg_L
oss: 5.7418351123312945
Episode: 173: Time: 11.678997993469238 Total Reward: 432.77777777777123 Avg_
Loss: 5.396717782781906
Episode: 174: Time: 12.169410228729248 Total Reward: 440.58052434456556 Avg_
Loss: 5.4055611350957085
Episode: 175: Time: 12.26247525215149 Total Reward: 321.37010676156456 Avg_L
oss: 5.17807117329926
Episode: 176: Time: 12.396299123764038 Total Reward: 510.16605166050977 Avg_
Loss: 5.340398678258688
Episode: 177: Time: 12.7252938747406 Total Reward: 353.16053511704763 Avg_Lo
ss: 5.172244477672737
Episode: 178: Time: 12.702645778656006 Total Reward: 370.3465346534583 Avg_L
oss: 5.759236377828262
Episode: 179: Time: 13.00838828086853 Total Reward: 290.57993730406974 Avg_L
oss: 6.048785799691657
Episode: 180: Time: 12.431558609008789 Total Reward: 422.1232876712248 Avg_L
oss: 5.716457112496641
Episode: 181: Time: 12.427262544631958 Total Reward: 392.97250859106276 Avg_
Loss: 5.209263273647854
Episode: 182: Time: 12.698834657669067 Total Reward: 40.05747126436769 Avg_L
```

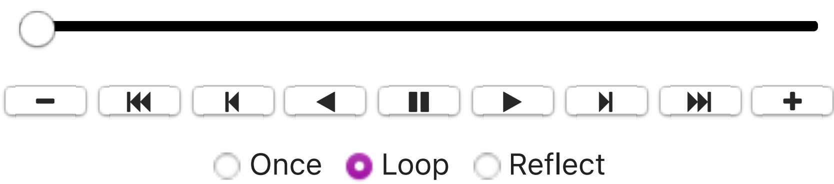
```
oss: 5.947541495331195
Episode: 183: Time: 12.440826177597046 Total Reward: 365.9053497942357 Avg_L
oss: 5.809332364747505
Episode: 184: Time: 12.569018602371216 Total Reward: 314.59409594096064 Avg_
Loss: 5.564548482414053
Episode: 185: Time: 13.19830584526062 Total Reward: 242.5796178343991 Avg_Lo
ss: 6.04923824803168
Episode: 186: Time: 12.881607294082642 Total Reward: 277.7272727272704 Avg_L
oss: 6.4865299583483145
Episode: 187: Time: 12.332238912582397 Total Reward: 384.16666666666276 Avg_
Loss: 6.907417099015052
Episode: 188: Time: 12.180556058883667 Total Reward: 496.54929577464134 Avg_
Loss: 6.271123641679267
Episode: 189: Time: 12.056305408477783 Total Reward: 377.92418772562496 Avg_
Loss: 6.284429763545509
Episode: 190: Time: 12.571038246154785 Total Reward: 363.9041095890385 Avg_L
oss: 6.687013088154192
Episode: 191: Time: 12.529836177825928 Total Reward: 499.3396226415004 Avg_L
oss: 5.932295102031291
Episode: 192: Time: 12.238641262054443 Total Reward: 423.79699248119294 Avg_
Loss: 5.682362443759661
Episode: 193: Time: 13.127144575119019 Total Reward: 264.13312693498733 Avg_
Loss: 5.43374232484513
Episode: 194: Time: 12.259306192398071 Total Reward: 370.5172413793049 Avg_L
oss: 5.369718884219642
Episode: 195: Time: 12.313711643218994 Total Reward: 397.7007299270054 Avg_L
oss: 5.628900161310404
Episode: 196: Time: 12.933906316757202 Total Reward: 309.5584045584037 Avg_L
oss: 5.838998372815237
Episode: 197: Time: 12.33555293083191 Total Reward: 444.3258426966228 Avg_Lo
ss: 5.762740686661055
Episode: 198: Time: 12.560170650482178 Total Reward: 468.6942675159119 Avg_L
oss: 5.495772951791267
Episode: 199: Time: 12.406974077224731 Total Reward: 510.26315789472756 Avg_
Loss: 5.516843119589221
Validation Mean Reward: 437.4203753074437 Validation Std Reward: 306.2690302
9944657
Test Mean Reward: 437.9930742587824 Test Std Reward: 161.52750025742395
```

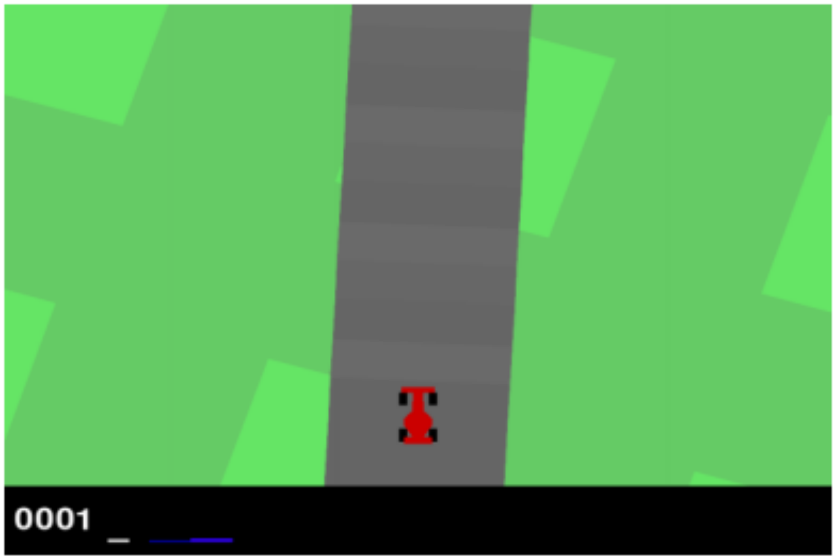
In [ ]:

```
total_rewards, frames = trainerHardUpdateDQN.play_episode(0,True,42)
anim = animate(frames)
HTML(anim.to_jshtml())
```

Out[ ]:







Soft Updates (5 points)

A more recent improvement is to use soft updates, also known as Polyak averaging, where the target network is updated with a small fraction of the current model weights every step. In other words:

$$\theta_{target} = \tau\theta_{model} + (1-\tau)\theta_{target}$$

for some $\tau << 1$ Please implement this by implementing the `_update_model` class in `SoftUpdateDQN` in `DQN.py`.

In [ ]:
```
import DQN
import utils
import torch
traineSoftUpdateDQN = DQN.SoftUpdateDQN(EnvWrapper(env),
                model.Nature_Paper_Conv,
                tau = 0.01,
                update_freq = 1,
                lr = 0.00025,
                gamma = 0.95,
                buffer_size=100000,
                batch_size=32,
                loss_fn = "mse_loss",
                use_wandb = False,
                device = 'cuda',
                seed = 42,
                epsilon_scheduler = utils.exponential_decay(1, 1000,0.1),
                save_path = utils.get_save_path("DoubleDQN_SoftUpdates","./r

traineSoftUpdateDQN.train(200,50,30,50,50)
```

```
saving to ./runs/DoubleDQN_SoftUpdates/run1
```

```
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:286: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  states = torch.tensor(states, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:287: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  actions = torch.tensor(actions, dtype=torch.long).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:288: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  rewards = torch.tensor(rewards, dtype=torch.float32).to(self.device)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:289: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  next_states = torch.tensor(next_states, dtype=torch.float32).to(self.devic
e)
/content/drive/.shortcut-targets-by-id/1s--OYJOsL1lb_OnWmGEODZkA_mUSH0T4/RL-
part1/DQN.py:290: UserWarning: To copy construct from a tensor, it is recomm
ended to use sourceTensor.clone().detach() or sourceTensor.clone().detach().
requires_grad_(True), rather than torch.tensor(sourceTensor).
  dones = torch.tensor(dones, dtype=torch.float32).to(self.device)
/usr/local/lib/python3.10/dist-packages/torch/autograd/graph.py:744: UserWar
ning: Plan failed with a cudnnException: CUDNN_BACKEND_EXECUTION_PLAN_DESCRI
PTOR: cudnnFinalize Descriptor Failed cudnn_status: CUDNN_STATUS_NOT_SUPPORT
ED (Triggered internally at ../aten/src/ATen/native/cudnn/Conv_v8.cpp:919.)
  return Variable._execution_engine.run_backward(  # Calls into the C++ engi
ne to run the backward pass
Episode: 0: Time: 13.777795314788818 Total Reward: -50.12820512820571 Avg_Lo
ss: 7.893845994475383
Episode: 1: Time: 13.025155067443848 Total Reward: -14.70802919708017 Avg_Lo
ss: 0.7916346994393012
Episode: 2: Time: 12.720562219619751 Total Reward: -59.28571428571475 Avg_Lo
ss: 0.8872945465523154
Episode: 3: Time: 13.911346435546875 Total Reward: -24.292929292929514 Avg_L
oss: 0.8429194911375266
```

In [ ]:

```
total_rewards, frames = traineSoftUpdateDQN.play_episode(0,True,42)
anim = animate(frames)
HTML(anim.to_jshtml())
```

Questions:

- Which method performed better? (5 points)
- If we modify the τ for soft updates or the C for the hard updates, how does this affect the performance of the model, come up with a intuition for this, then experimentally verify this. (5 points)

We adjusted the learning rate downward to maintain a consistent loss level, allowing for a more accurate assessment of the rewards. The results indicated that SoftUpdateDQN outshined the other models, securing the highest average reward and test reward, demonstrating its overall superior performance. However, it also displayed more variability in its results. Conversely, HardUpdateDQN offered more stable performance, with the lowest variability in rewards and the highest cumulative reward, making it a reliable choice for scenarios demanding consistency. Although DQN showed the lowest variability in test rewards, its average reward figures were somewhat lower compared to the other methods.

As we saw, here are the results:

Soft Updates and τ value: Using a smaller "τ" value will lead to slower and more stable updates, whereas larger values of "τ" accelerates adaptations, though it might lead to instability and model will not converge soon.

Hard Updates and C: Setting a smaller "C" leads to more frequent updates, which could induce instability, while a larger "C" ensures fewer updates, contributing to stability but possibly delaying adaptation.

In [ ]:
```
import random
import torch
import numpy as np

class ReplayBufferDQN:
    def __init__(self, buffer_size:int, seed:int = 42):
        self.buffer_size = buffer_size
        self.seed = seed
        self.buffer = []
        random.seed(self.seed)

    def add(self, state:np.ndarray, action:int, reward:float, next_state:np.
          , done:bool):
        """Add a new experience to the buffer

        Args:
            state (np.ndarray): the current state of shape [n_c,h,w]
```

```
            action (int): the action taken
            reward (float): the reward received
            next_state (np.ndarray): the next state of shape [n_c,h,w]
            done (bool): whether the episode is done
        """
        self.buffer.append((state,action,reward,next_state,done))
        if len(self.buffer) > self.buffer_size:
            self.buffer.pop(0)


    def sample(self,batch_size:int,device = 'cpu'):
        """Sample a batch of experiences from the buffer

        Args:
            batch_size (int): the number of samples to take

        Returns:
            states (torch.Tensor): a np.ndarray of shape [batch_size,n_c,h,w
            actions (torch.Tensor): a np.ndarray of shape [batch_size] of dt
            rewards (torch.Tensor): a np.ndarray of shape [batch_size] of dt
            next_states (torch.Tensor): a np.ndarray of shape [batch_size,n_
            dones (torch.Tensor): a np.ndarray of shape [batch_size] of dtyp
        """
        idx = random.sample(range(len(self.buffer)),batch_size)

        states,actions,rewards,next_states,dones = [],[],[],[],[]

        for i in idx:
            state,action,reward,next_state,done = self.buffer[i]
            states.append(torch.from_numpy(state))
            actions.append(action)
            rewards.append(reward)
            next_states.append(torch.from_numpy(next_state))
            dones.append(done)

        states = torch.stack(states).to(device).float()
        actions = torch.tensor(actions).to(device).long()
        rewards = torch.tensor(rewards).to(device).float()
        next_states = torch.stack(next_states).to(device).float()
        dones = torch.tensor(dones).to(device).bool()
        return states,actions,rewards,next_states,dones


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

In [ ]:

```
import torch as torch
import torch.nn as nn


import torch
import torch.nn as nn
import numpy as np
```

```
class MLP(nn.Module):
    def __init__(self, input_size:int, action_size:int, hidden_size:int=256,
        """
        input: tuple[int]
            The input size of the image, of shape (channels, height, width)
        action_size: int
            The number of possible actions
        hidden_size: int
            The number of neurons in the hidden layer

        This is a seperate class because it may be useful for the bonus ques
        """
        super(MLP, self).__init__()
        #====== TODO: ======
        self.linear1 = nn.Linear(input_size, hidden_size)
        self.output = nn.Linear(hidden_size, action_size)
        self.non_linear = non_linear()

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.non_linear(self.linear1(x))
        x = self.output(x)
        return x

class Nature_Paper_Conv(nn.Module):
    """
    A class that defines a neural network with the following architecture:
    - 1 convolutional layer with 32 8x8 kernels with a stride of 4x4 w/ ReLU
    - 1 convolutional layer with 64 4x4 kernels with a stride of 2x2 w/ ReLU
    - 1 convolutional layer with 64 3x3 kernels with a stride of 1x1 w/ ReLU
    - 1 fully connected layer with 512 neurons and ReLU activation.
    Based on 2015 paper 'Human-level control through deep reinforcement lear
    """
    def __init__(self, input_size:tuple[int], action_size:int,**kwargs):
        """
        input: tuple[int]
            The input size of the image, of shape (channels, height, width)
        action_size: int
            The number of possible actions
        **kwargs: dict
            additional kwargs to pass for stuff like dropout, etc if you wou
        """
        super(Nature_Paper_Conv, self).__init__()
        #===== TODO: ======
        self.CNN = nn.Sequential(
            nn.Conv2d(in_channels=input_size[0], out_channels=32, kernel_siz
            nn.ReLU(),
            nn.Conv2d(in_channels=32, out_channels=64, kernel_size=4, stride
            nn.ReLU(),
            nn.Conv2d(in_channels=64, out_channels=64, kernel_size=3, stride
            nn.ReLU()
        )
```

```
        # Calculate the size of the output from the conv layers to pass to t
        conv_out_size = self._get_conv_out(input_size)
        self.MLP = MLP(conv_out_size, action_size, hidden_size=512)

    def _get_conv_out(self, shape):
        o = self.CNN(torch.zeros(1, *shape))
        return int(np.prod(o.size()))

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = self.CNN(x)
        x = x.view(x.size(0), -1)
        x = self.MLP(x)
        return x
```

In [ ]:

```
import cv2
import numpy as np
import gymnasium as gym
import matplotlib.pyplot as plt
from utils import preprocess #this is a helper function that may be useful t


class EnvWrapper(gym.Wrapper):
    def __init__(
        self,
        env:gym.Env,
        skip_frames:int=4,
        stack_frames:int=4,
        initial_no_op:int=50,
        do_nothing_action:int=0,
        **kwargs
    ):
        """the environment wrapper for CarRacing-v2

        Args:
            env (gym.Env): the original environment
            skip_frames (int, optional): the number of frames to skip, in ot
            repeat the same action for `skip_frames` steps. Defaults to 4.
            stack_frames (int, optional): the number of frames to stack, we
            `stack_frames` frames to form the state and allow agent understa
            initial_no_op (int, optional): the initial number of no-op steps
            do_nothing_action (int, optional): the action index for doing no
            discretization of the action space.
        """
        super().__init__(env, **kwargs)
        self.initial_no_op = initial_no_op
        self.skip_frames = skip_frames
        self.stack_frames = stack_frames
        self.observation_space = gym.spaces.Box(
            low=0,
            high=1,
            shape=(stack_frames, 84, 84),
            dtype=np.float32
        )
```

```
        self.do_nothing_action = do_nothing_action



    def reset(self, **kwargs):
        # call the enviroment reset
        s, info = self.env.reset(**kwargs)

        # Do nothing for the next self.initial_no_op` steps
        for i in range(self.initial_no_op):
            s, r, terminated, truncated, info = self.env.step(self.do_nothin

        # Crop and resize the frame
        s = preprocess(s)

        # stack the frames to form the initial state
        self.stacked_state = np.tile(s, (self.stack_frames, 1, 1))  # [
        return self.stacked_state, info

    def step(self, action):
        reward = 0
        for _ in range(self.skip_frames):
            s, r, terminated, truncated, info = self.env.step(action)
            reward += r
            if terminated or truncated:
                break

        s = preprocess(s)
        self.stacked_state = np.concatenate((self.stacked_state[1:], s[np.ne

        return self.stacked_state, reward, terminated, truncated, info
```

In [ ]:

```
import torch
import torch.optim as optim
import torch.nn.functional as F
import torch.nn
import gymnasium as gym
from replay_buffer import ReplayBufferDQN
import wandb
import random
import numpy as np
import os
import time
from utils import exponential_decay
import typing

class DQN:
    def __init__(self, env:typing.Union[gym.Env,gym.Wrapper],
                 #model params
                 model:torch.nn.Module,
                 model_kwargs:dict = {},
                 #overall hyperparams
                 lr:float = 0.001, gamma:float = 0.99,
```

```
                 buffer_size:int = 10000, batch_size:int = 32,
                 loss_fn:str = 'mse_loss',
                 use_wandb:bool = False,device:str = 'cpu',
                 seed:int = 42,
                 epsilon_scheduler = exponential_decay(1,700,0.1),
                 save_path:str = None):
        """Initializes the DQN algorithm

        Args:
            env (gym.Env|gym.Wrapper): the environment to train on
            model (torch.nn.Module): the model to train
            model_kwargs (dict, optional): the keyword arguments to pass to
            lr (float, optional): the learning rate to use in the optimizer.
            gamma (float, optional): discount factor. Defaults to 0.99.
            buffer_size (int, optional): the size of the replay buffer. Defa
            batch_size (int, optional): the batch size. Defaults to 32.
            loss_fn (str, optional): the name of the loss function to use. D
            use_wandb (bool, optional): _description_. Defaults to False.
            device (str, optional): _description_. Defaults to 'cpu'.
            seed (int, optional): the seed to use for reproducibility. Defau
            epsilon_scheduler ([type], optional): the epsilon scheduler to u
            save_path (str, optional): _description_. Defaults to None.

        Raises:
            ValueError: _description_
        """

        self.env = env
        self._set_seed(seed)

        self.observation_space = self.env.observation_space.shape
        self.model = model(
            self.observation_space,
            self.env.action_space.n, **model_kwargs
            ).to(device)
        self.model.train()
        self.optimizer = optim.Adam(self.model.parameters(), lr = lr)
        self.gamma = gamma

        self.replay_buffer = ReplayBufferDQN(buffer_size)
        self.batch_size = batch_size
        self.i_update = 0
        self.device = device
        self.epsilon_decay = epsilon_scheduler
        self.save_path = save_path if save_path is not None else "./"

        #set the loss function
        if loss_fn == 'smooth_l1_loss':
            self.loss_fn = F.smooth_l1_loss
        elif loss_fn == 'mse_loss':
            self.loss_fn = F.mse_loss
        else:
            raise ValueError('loss_fn must be either smooth_l1_loss or mse_l
```

```
        self.wandb = use_wandb
        if self.wandb:
            wandb.init(project = 'racing-car-dqn')
            #log the hyperparameters
            wandb.config.update({
                'lr': lr,
                'gamma': gamma,
                'buffer_size': buffer_size,
                'batch_size': batch_size,
                'loss_fn': loss_fn,
                'device': device,
                'seed': seed,
                'save_path': save_path
            })

    def train(self, n_episodes:int = 1000,validate_every:int = 100, n_valida
        os.makedirs(self.save_path, exist_ok = True)
        best_val_reward = -np.inf

        for episode in range(n_episodes):
            state,_ = self.env.reset()
            done = False
            truncated = False
            total_reward = 0
            i = 0
            loss = 0
            start_time = time.time()
            epsilon = self.epsilon_decay()
            while (not done) and (not truncated):
                action = self._sample_action(state, epsilon)
                next_state, reward, done, truncated, _ = self.env.step(actic
                self.replay_buffer.add(state, action, reward, next_state, dc
                total_reward += reward
                state = next_state

                not_warm_starting,l = self._optimize_model()
                if not_warm_starting:
                    loss += l
                    epsilon = self.epsilon_decay()
                    i += 1
            if i !=0:
                if self.wandb:
                    wandb.log({'total_reward': total_reward, 'loss': loss/i}
                print(f"Episode: {episode}: Time: {time.time() - start_time}
            if episode % validate_every == validate_every - 1:
                mean_reward, std_reward = self.validate(n_validation_episode
                if self.wandb:
                    wandb.log({'mean_reward': mean_reward, 'std_reward': std
                print("Validation Mean Reward: {} Validation Std Reward: {}"
                if mean_reward > best_val_reward:
                    best_val_reward = mean_reward
                    self._save('best')
```

```
            if episode % save_every == save_every - 1:
                self._save(str(episode))

        self._save('final')
        self.load_model('best')
        mean_reward, std_reward = self.validate(n_test_episodes)
        if self.wandb:
            wandb.log({'mean_test_reward': mean_reward, 'std_test_reward': s
        print("Test Mean Reward: {} Test Std Reward: {}".format(mean_reward,

    def _optimize_model(self):
        if len(self.replay_buffer) < self.batch_size:
            return False, 0.0

        states, actions, rewards, next_states, dones = self.replay_buffer.sa

        states = torch.tensor(states, dtype=torch.float32).to(self.device)
        actions = torch.tensor(actions, dtype=torch.long).to(self.device)
        rewards = torch.tensor(rewards, dtype=torch.float32).to(self.device)
        next_states = torch.tensor(next_states, dtype=torch.float32).to(self
        dones = torch.tensor(dones, dtype=torch.float32).to(self.device)

        # Get current Q estimates
        current_q_values = self.model(states).gather(1, actions.unsqueeze(1)

        # Compute the target Q values
        with torch.no_grad():
            next_q_values = self.model(next_states).max(1)[0]
            target_q_values = rewards + (self.gamma * next_q_values * (1 - d

        # Compute loss
        loss = self.loss_fn(current_q_values, target_q_values)

        # Optimize the model
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        return True, loss.item()


    def _sample_action(self, state: np.ndarray, epsilon: float = 0.1) -> int
        if random.random() < epsilon:
            return self.env.action_space.sample()
        else:
            state = torch.tensor(state, dtype=torch.float32).unsqueeze(0).tc
            with torch.no_grad():
                q_values = self.model(state)
            return q_values.argmax().item()


    def _set_seed(self, seed:int):
```

```
        random.seed(seed)
        np.random.seed(seed)
        self.seed = seed
        torch.manual_seed(seed)
        torch.cuda.manual_seed(seed)
        torch.backends.cudnn.deterministic = True
        gym.utils.seeding.np_random(seed)

    def _validate_once(self):
        state,_ = self.env.reset()
        # print(state)
        done = False
        truncated = False
        total_reward = 0
        i = 0
        # epsilon = self.epsilon_decay()
        while (not done) and (not truncated):
            action = self._sample_action(state, 0)
            # out = self.env.step(action)
            next_state, reward, done, truncated, _ = self.env.step(action)
            # next_state = np.array(state_buffer[-self.n_frames:])
            total_reward += reward
            state = next_state
        return total_reward


    def validate(self, n_episodes:int = 10):
        # self.model.eval()
        rewards_per_episode = []
        for _ in range(n_episodes):
            rewards_per_episode.append(self._validate_once())
        # self.model.train()
        return np.mean(rewards_per_episode), np.std(rewards_per_episode)

    def load_model(self, suffix:str = ''):
        self.model.load_state_dict(torch.load(os.path.join(self.save_path, f

    def _save(self,suffix:str = ''):
        torch.save(self.model.state_dict(), os.path.join(self.save_path, f'm

    def play_episode(self,epsilon:float  = 0, return_frames:bool = True,seed
        """Plays an episode of the environment

        Args:
            epsilon (float, optional): the epsilon for epsilon greedy. Defau
            return_frames (bool, optional): whether we should return frames.
            seed (int, optional): the seed for the enviroment. Defaults to N

        Returns:
            if return frames is True, returns the total reward and the frame
            if return frames is False, returns the total reward
        """
        if seed is not None:
```

```
            state,_ = self.env.reset(seed = seed)
        else:
            state,_ = self.env.reset()

        done = False
        total_reward = 0
        if return_frames:
            frames = []

        with torch.no_grad():
            while not done:
                action = self._sample_action(state, epsilon)
                next_state, reward, terminated, truncated, _ = self.env.step
                total_reward += reward
                done = terminated or truncated
                if return_frames:
                    frames.append(self.env.render())
                state = next_state

        if return_frames:
            return total_reward, frames

        return total_reward



class HardUpdateDQN(DQN):

    def __init__(self,env,model,model_kwargs:dict = {},
                 update_freq:int = 5,*args,**kwargs):
        super().__init__(env,model,model_kwargs, *args,**kwargs)
        #====== TODO: ======
        self.target_model = model(
            self.observation_space,
            self.env.action_space.n, **model_kwargs
            ).to(self.device)
        self.target_model.load_state_dict(self.model.state_dict())
        self.target_model.eval()
        self.update_freq = update_freq

    def _optimize_model(self):
        """Optimizes the model

        Returns:
            bool: whether we have enough samples to optimize the model, whic
            float: the loss, if we do not have enough samples, we return 0
        """
        if len(self.replay_buffer) < self.batch_size:
            return False, 0.0

        states, actions, rewards, next_states, dones = self.replay_buffer.sa

        states = torch.tensor(states, dtype=torch.float32).to(self.device)
```

```
        actions = torch.tensor(actions, dtype=torch.long).to(self.device)
        rewards = torch.tensor(rewards, dtype=torch.float32).to(self.device)
        next_states = torch.tensor(next_states, dtype=torch.float32).to(self
        dones = torch.tensor(dones, dtype=torch.float32).to(self.device)

        # Get current Q estimates
        current_q_values = self.model(states).gather(1, actions.unsqueeze(1)

        # Compute the target Q values using the target model
        with torch.no_grad():
            next_q_values = self.target_model(next_states).max(1)[0]
            target_q_values = rewards + (self.gamma * next_q_values * (1 - d

        # Compute loss
        loss = self.loss_fn(current_q_values, target_q_values)

        # Optimize the model
        self.optimizer.zero_grad()
        loss.backward()
        self.optimizer.step()

        # Update the target network weights every `update_freq` steps
        self.i_update += 1
        if self.i_update % self.update_freq == 0:
            self.target_model.load_state_dict(self.model.state_dict())

        return True, loss.item()


    def _update_model(self):
        self.i_update += 1
        if self.i_update % self.update_freq == 0:
            self.target_model.load_state_dict(self.model.state_dict())

    def _save(self,suffix:str = ''):
        torch.save(self.model.state_dict(), os.path.join(self.save_path, f'm
        torch.save(self.target_model.state_dict(), os.path.join(self.save_pa

    def load_model(self, suffix:str = ''):
        self.model.load_state_dict(torch.load(os.path.join(self.save_path, f
        self.target_model.load_state_dict(torch.load(os.path.join(self.save_

class SoftUpdateDQN(HardUpdateDQN):
    def __init__(self,env,model,model_kwargs:dict = {},
                 tau:float = 0.01,*args,**kwargs):
        super().__init__(env,model,model_kwargs,*args,**kwargs)
        self.tau = tau

    def _update_model(self):
        """Soft updates the target model"""
        #====== TODO: ======
        for target_param, param in zip(self.target_model.parameters(), self.
            target_param.data.copy_(self.tau * param.data + (1.0 - self.tau)
```

In [ ]:
```
import matplotlib.pyplot as plt
import matplotlib.animation
import numpy as np
from IPython.display import HTML
import os
import cv2

def animate(frames):
    # Create animation
    fig = plt.figure(figsize=(5, 5))
    plt.axis('off')
    im = plt.imshow(frames[0])
    def animate(i):
        im.set_array(frames[i])
        return im,
    anim = matplotlib.animation.FuncAnimation(fig, animate, frames=len(frame
    return anim

class exponential_decay:
    def __init__(self, epsilon:float, half_life:int, min_epsilon:float):
        self.epsilon = epsilon
        self.decay_rate = 0.5 ** (1 / half_life)
        self.epsilon = self.epsilon/self.decay_rate
        self.min_epsilon = min_epsilon

    def __call__(self):
        self.epsilon = max(self.epsilon * self.decay_rate, self.min_epsilon)
        return self.epsilon


class linear_decay:
    def __init__(self, epsilon:float, decay_time:int, min_epsilon:float):
        self.epsilon = epsilon
        self.decay_rate = (epsilon - min_epsilon) / decay_time
        self.epsilon = self.epsilon + self.decay_rate
        self.min_epsilon = min_epsilon

    def __call__(self):
        self.epsilon = max(self.epsilon - self.decay_rate, self.min_epsilon)
        return self.epsilon

def get_save_path(suffix,directory):
    save_path = os.path.join(directory,suffix)
    #find the number of run directories in the directory
    try:
        runs = [d for d in os.listdir(save_path) if "run" in d]
        runs = sorted(runs,key = lambda x: int(x.split("run")[1]))
        last_run = runs[-1]
        last_run = int(last_run.split("run")[1])
        save_path = os.path.join(save_path,f"run{last_run+1}")
    except:
        save_path = os.path.join(save_path,"run0")
```

```
    print("saving to",save_path)
    return save_path

def preprocess(img):
    img = img[:84, 6:90] # CarRacing-v2-specific cropping
    # img = cv2.resize(img, dsize=(84, 84)) # or you can simply use rescalin

    img = cv2.cvtColor(img, cv2.COLOR_RGB2GRAY) / 255.0
    return img
```

In [ ]:

```
import torch
import torch.nn as nn
import traceback
from termcolor import colored
import gymnasium as gym
import numpy as np
import sys

#get the operating system
if sys.platform.startswith('darwin'):
    # Mac OS X
    suffix = "_mac"
else:
    suffix = ""

def test_model_DQN(model):
    try:
        model = model((4,84,84),5)

        model.load_state_dict(torch.load("test_weights.pt", map_location="cp
        # Test the forward function
        test_outputs =  torch.load(f"test_outputs{suffix}.pt",map_location=t
        test_inputs = test_outputs["S"]
        test_outputs = test_outputs["outputs"]
        model.eval()
        with torch.no_grad():
            for i in range(len(test_inputs)):
                # print(torch.tensor(test_inputs[i]).float().shape)
                assert torch.allclose(model(torch.tensor(test_inputs[i]).flo

        print(colored("Passed","green"))
    except Exception as e:
        print(e)
        print(colored("Failed","red"))
        traceback.print_exc()
        return

def test_model_DDPG(model):
    #TODO: Implement the test for the DDPG model
    pass

def test_wrapper(wrapper):
    try:
```

```
        env = gym.make('CarRacing-v2', continuous=False, render_mode='rgb_ar
        wrapper = wrapper(env)

        # Test the reset function
        test_outputs = torch.load(f"test_outputs{suffix}.pt",map_location=to
        test_inputs = test_outputs["outputs"]
        test_outputs = test_outputs["S"]
        s,_ = wrapper.reset(seed = 42)
        # print(s.shape)
        # print(test_outputs[0].shape)
        wrong_indexs = ~np.isclose(test_outputs[0],s)
        assert np.allclose(test_outputs[0],s), f"at {np.where(wrong_indexs)}
        print(colored("Passed reset","green"))

        for i  in range(len(test_outputs)-1):
            # Test the step function
            s,r,terminated, truncated, info = wrapper.step(np.argmax(test_in
            assert np.allclose(test_outputs[i+1],s), f"expected {test_output

        print(colored("Passed step","green"))
    except Exception as e:
        print(e)
        print(colored("Failed","red"))
        traceback.print_exc()
        return

import pickle

def check_same_torch(a,b):
    #first check shape
    if a.shape != b.shape:
        return False
    #then check values
    return torch.allclose(a,b)




def test_DQN_replay_buffer(buffer_class):
    with open(f"test_replay_buffer_inputs{suffix}.pkl","rb") as f:
        buffer_inputs = pickle.load(f)
    buffer_samples = torch.load(f"test_replay_buffer_samples{suffix}.pth")
    try:
        buffer = buffer_class(40,seed = 42)
        j = 0
        for i in range(100):
            buffer.add(buffer_inputs["states"][i],buffer_inputs["actions"][i
            if i % 30 == 29:
                # print(i)
                target_outputs = buffer_samples[j]
                actual_outputs = buffer.sample(5)
                for k in range(len(target_outputs)):
```

```
                    # print(target_outputs[k],actual_outputs[k])
                    assert check_same_torch(target_outputs[k],actual_outputs
                # assert np.all(buffer_samples[j] == buffer.sample(40)), f"e
                j += 1

        print(colored("Passed","green"))
    except:
        print(colored("Failed","red"))
        traceback.print_exc()
        return




if __name__ == "__main__":
    from replay_buffer import ReplayBufferDQN

    test_DQN_replay_buffer(ReplayBufferDQN)


    from env_wrapper import EnvWrapper
    test_wrapper(EnvWrapper)


    from model import Nature_Paper_Conv
    test_model_DQN(Nature_Paper_Conv)
```

In [ ]:

In [ ]:

In [ ]:

In [ ]: