```python
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 to grayscale and
crop the image


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
    ):
        """
        Gym Wrapper for CarRacing-v3 that adds frame-skipping, frame-stacking,
        and an initial no-op period to facilitate training.

        Args:
            env (gym.Env): the original environment
            skip_frames (int, optional): the number of frames to skip, in other words we will
repeat the same action for `skip_frames` steps.
            stack_frames (int, optional): the number of frames to stack as a state, we stack
            `stack_frames` frames to form the state and allow agent understand the motion of
the car. Defaults to 4.
            initial_no_op (int, optional): the initial number of no-op steps to do nothing at
the beginning of the episode. Defaults to 50.
            do_nothing_action (int, optional): the action index for doing nothing. Defaults to
0, which should be correct unless you have modified the
            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):
        """
        Reset the environment and perform a sequence of no-op actions before
        returning the initial stacked state.

        Returns:
            stacked_state (np.ndarray): Array shape (stack_frames,84,84).
            info (dict): Env-provided info dict.
        """
        # ========== YOUR CODE HERE ==========
        # TODO:
        # 1. call the enviroment reset
        # 2. do nothing for the next self.initial_no_op` steps
        # 3. crop and resize the final frame
        # 4. stack the frames to form the initial state
        # ====================================

        # step 1
        obs, info = self.env.reset(**kwargs)

        # step 2
        for _ in range(self.initial_no_op):
            obs, _, terminated, truncated, _ = self.env.step(self.do_nothing_action)
            if terminated or truncated:
                obs, info = self.env.reset(**kwargs)

        # step 3
        processed_frame = preprocess(obs)  # Assume this returns (84, 84) grayscale float32
np.array

        # step 4
        frames = []
        for _ in range(self.stack_frames):
            frames.append(processed_frame)

        self.stacked_state = np.stack(frames, axis=0)

        # ========== YOUR CODE ENDS ==========

        return self.stacked_state, info

    def step(self, action):
        """
        Apply the given action with frame-skipping, accumulate rewards,
        and update the frame stack.

        Args:
            action (int): Discrete action index to execute.

        Returns:
            stacked_state (np.ndarray): Updated state with recent `stack_frames` frames.
            reward (float): Sum of rewards over skipped frames.
            terminated (bool): True if env episode ended.
            truncated (bool): True if env truncated.
            info (dict): Env-provided info dict.
        """
        # ========== YOUR CODE HERE ==========
        # TODO:
        # 1. take step(action) on underlying env for `self.skip_frames` steps.
        # 2. sum the immediate rewards
        # 3. preprocess the final observed frame.
        # 4. append new frame to `self.stacked_state` and remove oldest.
        # ====================================

        total_reward = 0.0
        terminated = False
        truncated = False
        info = {}

        # step 1 and step 2
        for _ in range(self.skip_frames):
            obs, reward, terminated, truncated, info = self.env.step(action)
            total_reward += reward
            if terminated or truncated:
                break

        # step 3
        processed_frame = preprocess(obs)

        # step 4
        self.stacked_state = np.concatenate(
            [self.stacked_state[1:], np.expand_dims(processed_frame, axis=0)],
            axis=0
        )
        # ========== YOUR CODE ENDS ==========
        return self.stacked_state, total_reward, terminated, truncated, info
```