TALLER DE IMPLEMENTACIÓN – PARTE II
Aprendizaje por refuerzo

Carlos Garavito.
Escuela ingenieria, ciencia y tecnologia,
Universidad del Rosario.

0.1. Ejercicio 1

Realice un análisis de sensibilidad para los parámetros $n = 2, 4, 8$ y $\alpha = 0, 2/8, 0, 4/8, 0, 8/8$ de un agente n-Step sobre el entorno del Mountain Car.

El análisis sobre los parámetros α y n se hace implementando el método `load_agent_nStepCS` y `sweep_nStep` en el módulo `mountain_car.py`. El primero crea el agente, el segundo hace el barrido de parámetros.

```
def load_agent_nStepCS() -> nStepCS:
    '''
    Creates a nStepCS agent with a set
    of parameters determined inset
    '''
    # Define parameters
    parameters = {"numDims":2,\
                  "nA":3,\
                  "gamma":1,\
                  "epsilon":0.1,\
                  "alpha":0.1,\
                  "n":8,\
                  "numTilings":8,\
                  "numTiles":[10, 10],\
                  "scaleFactors":[\
                    {"min":-1.2,\
                    "max":0.6},
                    {"min":-0.07,\
                      "max":0.07}]
                    }
    # Create approximating function
    Q = TilesQ(parameters=parameters)
    # Create agent
    return nStepCS(parameters, Q)

def sweep_nStep():
    '''
    Runs a sweep over alpha and n
    '''
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_SarsaCS()
    # Create train-and-run object
```

```
    act = load_act(agent, 'Sarsa')
    # Sweep alpha
    print('Sweeping alpha...')
    alphas = [0.2/8, 0.4/8, 0.8/8]
    n = [2, 4, 8]
    act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n)
    print('Done!')
```

Listing 1: Implementación nStep en el ambiente mointain_car.py

Imagen arrojada por el método TrainRun.sweep(2) La imagen obtenida al ejecutar el código se muestra en la figura 2.

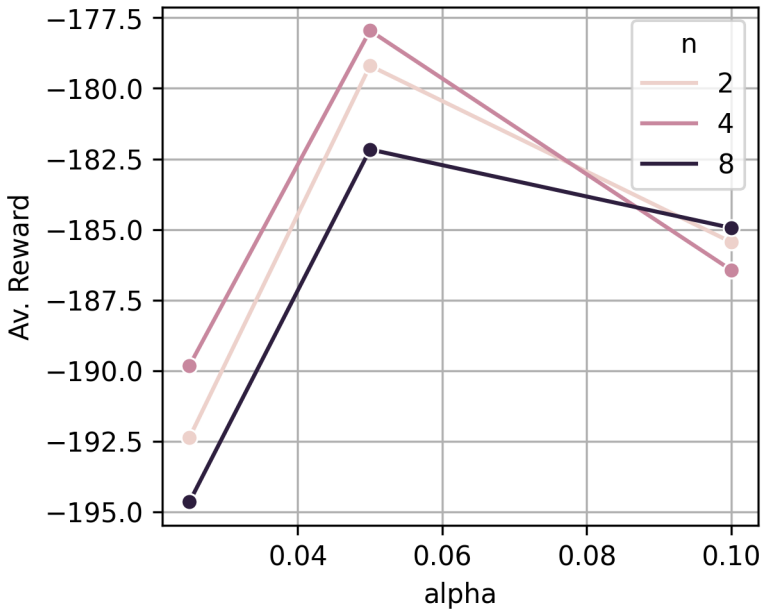

Figura 1: Comparación de reward para las diferentes combinaciones de valores para α y n, en el ambiente mountaun car.

Comentario sobre los parámetros α y n Se observa que, como se espera, la recompensa se ve afectada por la elección de los parámetros α y n. En particular, se puede ver que para todos los casos evaluados en n, la mejor recompensa se encuentra cuando $\alpha = 0{,}5$.

Parámetros con mejor desempeño Se observa que los parámetros con mejor desempeño son $n = 8$ y $\alpha = 0{,}1$ Se observa en la figura 2 que la combinación de valores para α y n que genera la mejor recompensa, corresponde a $\alpha = 0{,}50$ y $n = 4$.

0.2. Ejercicio 2

Entrene un agente sarsa sobre el entorno seleccionado. Para este propósito cree un nuevo módulo cart pole.py o lunar lander.py en la carpeta Examples.

La implementación del entorno `lunar_lander.py` se puede encontrar en el anexo y su documentación en [1].

Descripción del entorno El entorno `lunar_lander.py` consiste en optimizar la trayectoria de un cohete en proceso de aterrizaje, tal que la caida sea controlada por el encendido o apagado del motor.
Hay cuatro acciones posibles,

- No hacer nada
- Encender el motor que mueve el cohete hacia la izquierda,
- Encender el motor que mueve el cohete hacia arriba,
- Encender el motor que mueve el cohete hacia la derecha.

Los estados posibles son:

- Posición horizontal, limitado en el rango $(-90, 90)$;
- Posición vertical, limitado en el rango $(-90, 90)$;
- Velocidad horizontal, limitado en el rango $(-5, 5)$;
- Velocidad vertical, limitado en el rango $(-5, 5)$;
- Ángulo de inclinación del cohete, limitado en el rango (-3.1415927, 3.1415927);
- Velocidad angular, limitado en el rango $(-5, 5)$;
- Indicador booleano de aterrizaje en la pata 1 del cohete, limitado en el rango $(0, 1)$;
- Indicador booleano de aterrizaje en la pata 2 del cohete, limitado en el rango $(0, 1)$.

Imágenes arrojadas por TrainRun.train() En la figura 2 se muestra la evolución de la recompensa para los 100 episodios de entrenamiento. Se puede observar que el ambiente es más complejo y que la recompensa no converge.

Imágenes arrojadas por TrainRun.test() En la figura 3 se muestra la evolución de la recompensa para los 100 episodios de entrenamiento. Se puede observar que el promedio es aproximadamente -87 puntos.

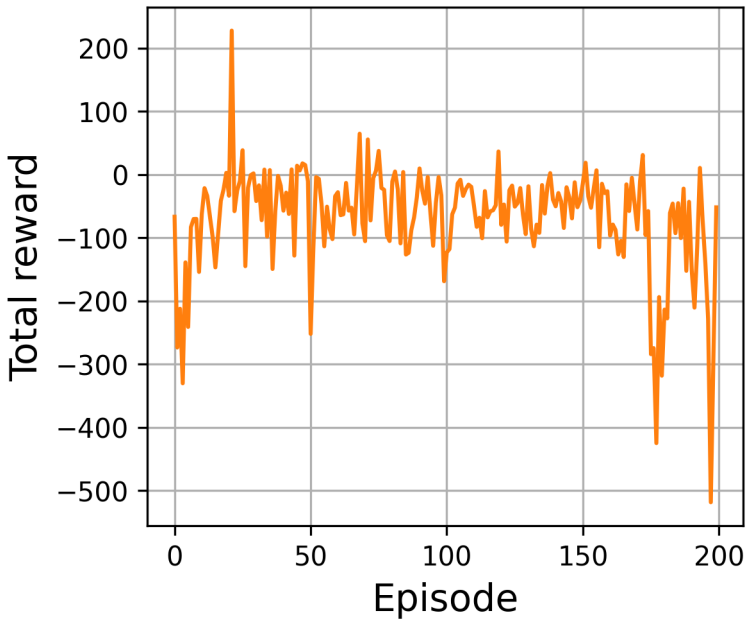

Figura 2: Evolución de la recompensa a lo largo de los episodios entrenados.

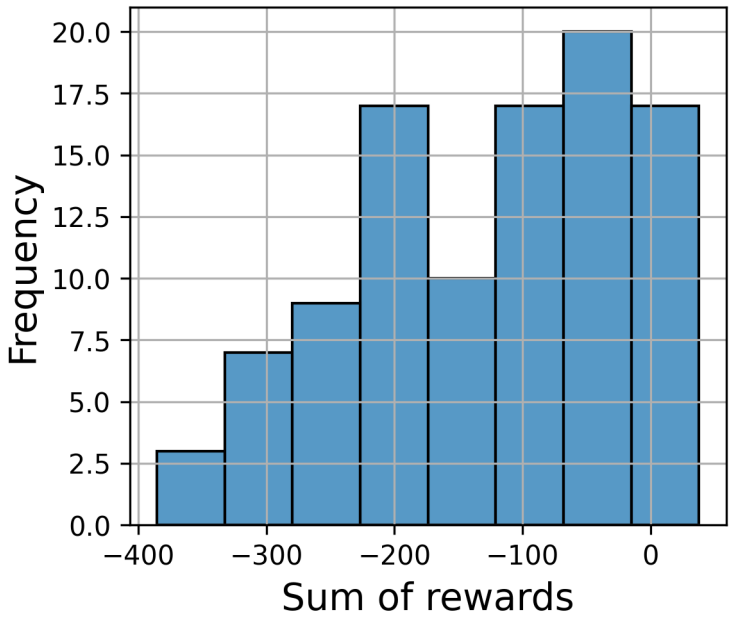

Figura 3: Histograma obtenido por el agente en el ambiente lunar lander, entrenado en 100 episodios.

Barrido de parámetros α Para realizar el barrido de parámetros alpha, se implementa el siguiente método, en el

```
1
2 def sweep_SARSA():
3     '''
4     Runs a sweep over alpha
```

```
    '''
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_SarsaCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Sweep alpha
    print('Sweeping alpha...')
    alphas = [0.2/8, 0.4/8, 0.8/8]
    act.sweep(parameter='alpha', values=alphas, num_simulations=10)
    print('Done!')
```

Listing 2: Implementación sweep_SARSA en el ambiente lunar_lander.py

Imagen arrojada por TrainRun.sweep() La gráfica obtenida se muestra en a figura 4. Se puede dar cuenta que el mejor valor de $\alpha = 0{,}025$, en el cual se alcanza una mejor recompensa. En particular, despues del episodio 150 la recompensa siempre es positiva para este valor de parámetro.

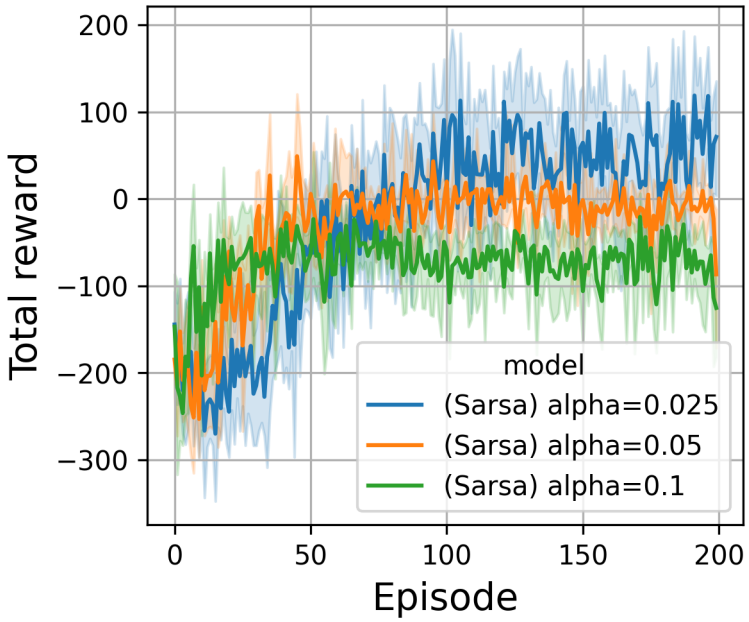

Figura 4: Comparación de reward para las diferentes combinaciones de valores para α, en el ambiente lunar lander.

0.3. Ejercicio 3

Repita el ejercicio 1 sobre el entorno seleccionado en el ejercicio 2.

El análisis sobre los parámetros α y n se hace implementando el método `sweep_nStep` en el módulo `lunar_lander.py`.

```
def load_agent_nStepCS() -> nStepCS:
    '''
    Creates a nStepCS agent with a set
    of parameters determined inset
    '''
    # Define parameters
    parameters = {"numDims":8,\
                  "nA":4,\
                  "gamma":1,\
                  "epsilon":0.1,\
                  "alpha":0.5,\
                  "numTilings":8,\
                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\
                  "n":4,\
                  "scaleFactors":[\
                    {"min":-90., "max":90.}, # x coordiantes
                    {"min":-90., "max":90.}, # y coordiantes
                    {"min":-5., "max":5.}, # x velocity
                    {"min":-5., "max":5.}, # y velocity
                    {"min":-3.1415927, "max":3.1415927}, # object angle
                    {"min":-5., "max":5.}, # angular velocity
                    {"min":0., "max":1.}, # boolean leg 1
                    {"min":-0., "max":1.}, # boolean leg 2
                    ]
                    }
    # Create approximating function
    Q = TilesQ(parameters=parameters)
    # Create agent
    return nStepCS(parameters, Q)

def sweep_nStep():
    '''
    Runs a sweep over alpha
    '''
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_nStepCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Sweep alpha
    print('Sweeping alpha...')
    alphas = [0.2/8, 0.4/8, 0.8/8]
    n = [2, 4, 8]
    act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n,
   num_simulations=5)
    print('Done!')
```

Listing 3: Implementación de sweep_nStep para el ambiente lunar lander.

Imagen arrojada por el método TrainRun.sweep(2) La imagen obtenida al ejecutar el código se muestra en la figura 5.

Comentario sobre los parámetros Se observa que el agente, cuando $n = 2, 8$, al incrementar el valor de α incrementa su rendimiento. En caso contario, con $n = 4$, en

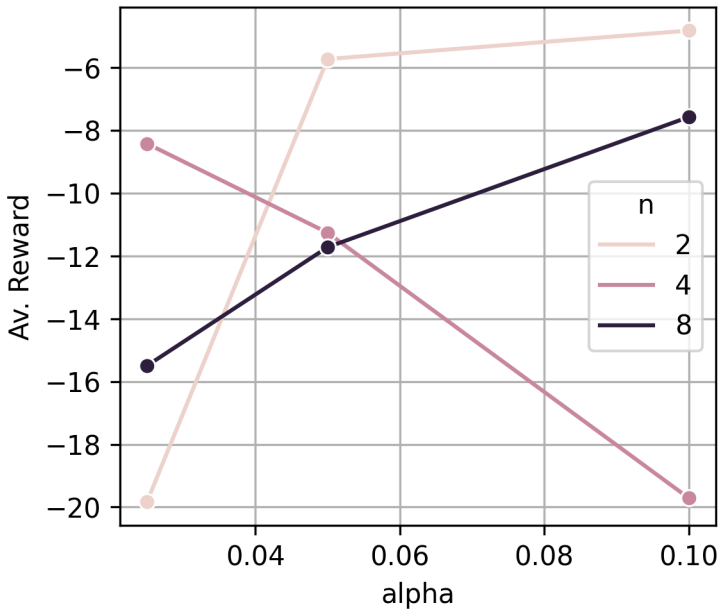

Figura 5: Comparación de reward para las diferentes combinaciones de valores para α y n, en el ambiente mountaun car.

donde la recompensa disminuye al aumentar α.

Parámetros con mejor desempeño Se observa que los parámetros con mejor desempeño son $n = 8$ y $\alpha = 0{,}1$

0.4. Ejercicio 4

Compare el desempeño de los dos agentes, sarsa y n-step, cada uno inicializado con sus mejores parámetros, en el entorno seleccionado en el ejercicio 2.

Histograma Para generar el histograma que compara ambos agentes, se implemento el método

```
def train_and_compare():
    # Create agent
    agent_SARSA = load_agent_SarsaCS()
    # Create train-and-run object
    act = load_act(agent_SARSA, 'Sarsa')
    # Train the SARSA agent
    print('Training SARSA agent...')
    act.train()
    # Testing the agent
    print('Testing SARSA agent...')
    act.num_episodes = 100
    act.test(to_df=True)
    df_sarsa = act.data
    # Create agent
```

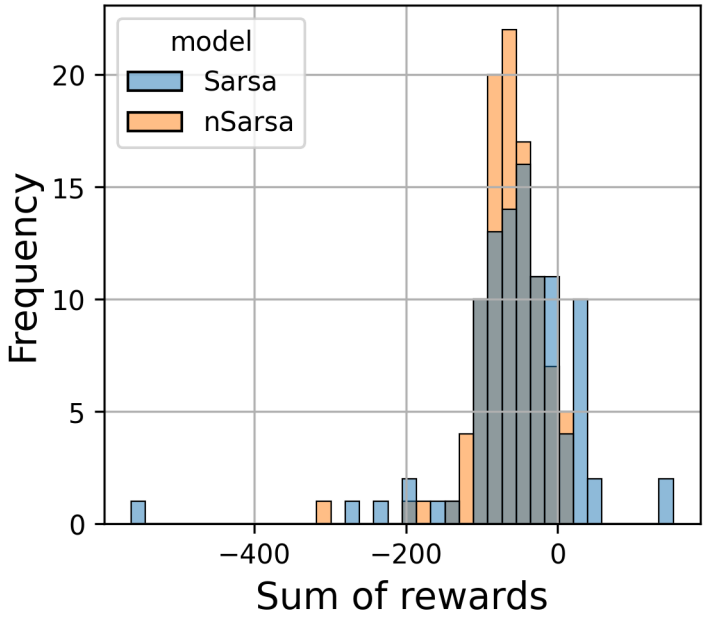

Figura 6: Histograma para la comparación de los dos métodos de entrenamiento.

```
    agent_nSARSA = load_agent_nStepCS()
    # Create train-and-run object
    act = load_act(agent_nSARSA, 'nSarsa')
    # Train the agent
    print('Training nSARSA agent...')
    act.train()
    # Testing the agent
    print('Testing nSARSA agent...')
    act.num_episodes = 100
    act.test(to_df=True)
    df_nsarsa = act.data
    # Compare performances
    df = pd.concat([df_sarsa, df_nsarsa], ignore_index=True)
    p = Plot(df)
    p.plot_histogram_rewards(act.file_compare_hist)
    print(f'Plot saved to {act.file_compare_hist}')
    p.plot_rewards(act.file_compare_rew)
    print(f'Plot saved to {act.file_compare_rew}')
```

Listing 4: Implementación del método train_and_compare() para el entorno lunar lander.

El histograma obtenido, se muestra en la figura 6.

Promedios de recompensa total El promedio de recompensas para cada modelo es:

- Sarsa: -51.197584
- nSarsa: -63.451483

Comentario Se puede dar cuenta que, bajo los mejores parámetros encontrados, ambos métodos tienen rendimiento similar para el entorno de lunar lander. En particular, con -51 puntos promedio, el Sarsa tiene mejor rendimiento que el nSarsa que tiene -63 puntos en promedio. Sin embargo, en ambos casos, no se encuentra una buena solución ya que en promedio no se tienen recompensas superiores a cero.

Finalmente, la implementación del presente taller se puede encontrar en `https://github.com/cgaravitoc/reinforcement_learning_projects`.

Referencias

[1] OpenAI Gym. Lunar Lander. `https://gym.openai.com/envs/LunarLander-v2/`. Accessed on 28th May 2023.

Anexo

Implementación del ambiente lunar_lander.py

```
from Utils.train import TrainRun
from Agents.agentsCS import SarsaCS, nStepCS
from Agents.linearQ import TilesQ
from Utils.interpreters import gym_interpreter1
import matplotlib.pyplot as plt
import pandas as pd
from Utils.utils import Plot



def try_env():
    '''
    Loads an agent and runs it
    without learning on the
    Lunar lander environment
    '''
    print('Loading agent and environment...')
     # Create agent
    agent = load_agent_SarsaCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Show the untrained agent
    print('Showing the untrained agent...')
    act.run(visual=True)
    print('Done!')


def train_and_run_SARSA():
    '''
    Trains a SARSA agent on the Mountain Car
    '''
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_SarsaCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Train the agent
    print('Training the agent...')
    act.train()
    # Show the trained agent
    print('Showing the trained agent...')
    act.run()
    # Testing the agent
    print('Testing the agent...')
    act.test()
    print('Done!')

def sweep_SARSA():
    '''
    Runs a sweep over alpha
    '''
```

```
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_SarsaCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Sweep alpha
    print('Sweeping alpha...')
    alphas = [0.2/8, 0.4/8, 0.8/8]
    act.sweep(parameter='alpha', values=alphas, num_simulations=50)
    print('Done!')


def load_agent_SarsaCS() -> SarsaCS:
    '''
    Creates a SarsaCS agent with a set
    of parameters determined inset
    '''
    # Define parameters
    parameters = {"numDims":8,\
                  "nA":4,\
                  "gamma":1,\
                  "epsilon":0.1,\
                  "alpha":0.1,\
                  "numTilings":8,\
                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\
                  "scaleFactors":[\
                    {"min":-90., "max":90.}, # x coordiantes
                    {"min":-90., "max":90.}, # y coordiantes
                    {"min":-5., "max":5.}, # x velocity
                    {"min":-5., "max":5.}, # y velocity
                    {"min":-3.1415927, "max":3.1415927}, # object angle
                    {"min":-5., "max":5.}, # angular velocity
                    {"min":0., "max":1.}, # boolean leg 1
                    {"min":-0., "max":1.}, # boolean leg 2
                    ]
                    }
    # Create approximating function
    Q = TilesQ(parameters=parameters)
    # Create agent
    return SarsaCS(parameters, Q)


def load_agent_nStepCS() -> SarsaCS:
    '''
    Creates a SarsaCS agent with a set
    of parameters determined inset
    '''
    # Define parameters
    parameters = {"numDims":8,\
                  "nA":4,\
                  "gamma":1,\
                  "epsilon":0.1,\
                  "alpha":0.1,\
                  "numTilings":8,\
                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\
```

```
                    "scaleFactors":[\
                      {"min":-90., "max":90.}, # x coordiantes
                      {"min":-90., "max":90.}, # y coordiantes
                      {"min":-5., "max":5.}, # x velocity
                      {"min":-5., "max":5.}, # y velocity
                      {"min":-3.1415927, "max":3.1415927}, # object angle
                      {"min":-5., "max":5.}, # angular velocity
                      {"min":0., "max":1.}, # boolean leg 1
                      {"min":-0., "max":1.}, # boolean leg 2
                      ]
                      }
    # Create approximating function
    Q = TilesQ(parameters=parameters)
    # Create agent
    return SarsaCS(parameters, Q)



def load_act(agent, model_name:str) -> TrainRun:
    '''
    Creates a train-and-run object with
    parameters given inset
    '''
    act = TrainRun(\
        env_name = 'LunarLander-v2',\
        state_interpreter=gym_interpreter1,\
        agent=agent,\
        model_name=model_name,\
        num_rounds=1000, # by default was in 1000
        num_episodes=200
        )
    return act

def sweep_nStep():
    '''
    Runs a sweep over alpha
    '''
    # Create agent
    print('Loading agent and environment...')
    agent = load_agent_nStepCS()
    # Create train-and-run object
    act = load_act(agent, 'Sarsa')
    # Sweep alpha
    print('Sweeping alpha...')
    alphas = [0.2/8, 0.4/8, 0.8/8]
    n = [2, 4, 8]
    act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n,
   num_simulations=5)
    print('Done!')


def train_and_compare():
    # Create agent
    agent_SARSA = load_agent_SarsaCS()
    # Create train-and-run object
```

```
    act = load_act(agent_SARSA, 'Sarsa')
    # Train the SARSA agent
    print('Training SARSA agent...')
    act.train()
    # Testing the agent
    print('Testing SARSA agent...')
    act.num_episodes = 100
    act.test(to_df=True)
    df_sarsa = act.data
    # Create agent
    agent_nSARSA = load_agent_nStepCS()
    # Create train-and-run object
    act = load_act(agent_nSARSA, 'nSarsa')
    # Train the agent
    print('Training nSARSA agent...')
    act.train()
    # Testing the agent
    print('Testing nSARSA agent...')
    act.num_episodes = 100
    act.test(to_df=True)
    df_nsarsa = act.data
    # Compare performances
    df = pd.concat([df_sarsa, df_nsarsa], ignore_index=True)
    p = Plot(df)
    p.plot_histogram_rewards(act.file_compare_hist)
    print(f'Plot saved to {act.file_compare_hist}')
    p.plot_rewards(act.file_compare_rew)
    print(f'Plot saved to {act.file_compare_rew}')
```

Listing 5: Implementación del ambiente lunar_lander.py