

TALLER DE IMPLEMENTACIÓN – PARTE II

Aprendizaje por refuerzo



Carlos Garavito.

Escuela ingeniería, ciencia y tecnología,
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.

```
1 def load_agent_nStepCS() -> nStepCS:
2     '''
3     Creates a nStepCS agent with a set
4     of parameters determined inset
5     '''
6     # Define parameters
7     parameters = {"numDims":2,\
8                   "nA":3,\
9                   "gamma":1,\
10                  "epsilon":0.1,\
11                  "alpha":0.1,\
12                  "n":8,\
13                  "numTilings":8,\
14                  "numTiles":[10, 10],\
15                  "scaleFactors":[\
16                      {"min":-1.2,\
17                      "max":0.6},\
18                      {"min":-0.07,\
19                      "max":0.07}]\
20                  ]
21     # Create approximating function
22     Q = TilesQ(parameters=parameters)
23     # Create agent
24     return nStepCS(parameters, Q)
25
26 def sweep_nStep():
27     '''
28     Runs a sweep over alpha and n
29     '''
30     # Create agent
31     print('Loading agent and environment...')
32     agent = load_agent_SarsaCS()
33     # Create train-and-run object
```

```

34 act = load_act(agent, 'Sarsa')
35 # Sweep alpha
36 print('Sweeping alpha...')
37 alphas = [0.2/8, 0.4/8, 0.8/8]
38 n = [2, 4, 8]
39 act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n)
40 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.

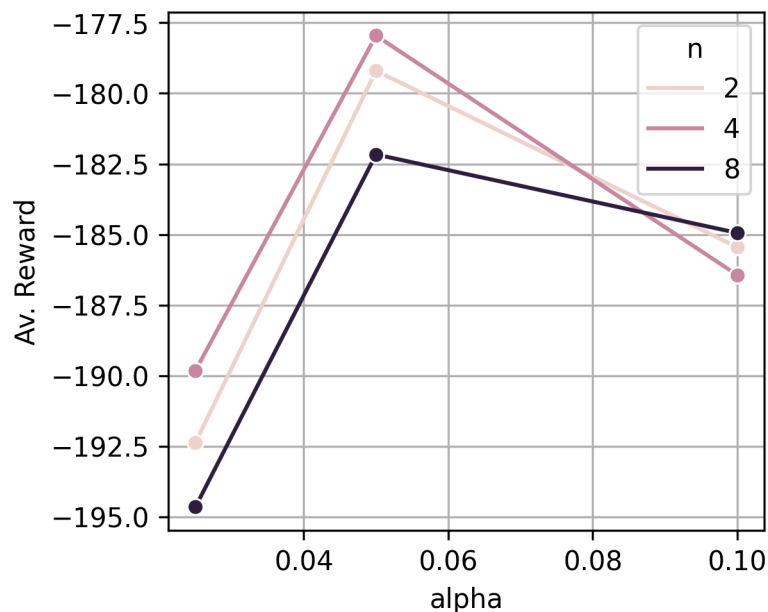


Figura 1: Comparación de reward para las diferentes combinaciones de valores para α y n , en el ambiente mountaun car.

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 caída 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.

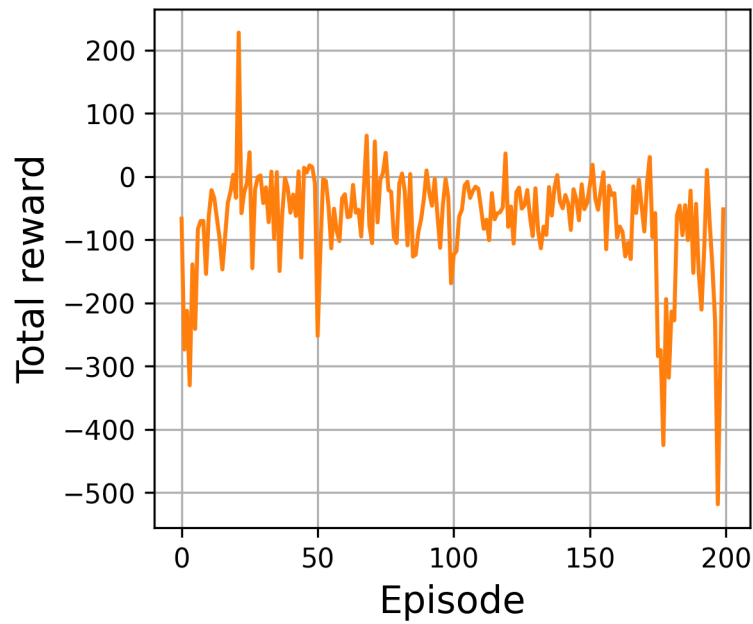


Figura 2: Evolución de la recompensa a lo largo de los episodios entrenados.

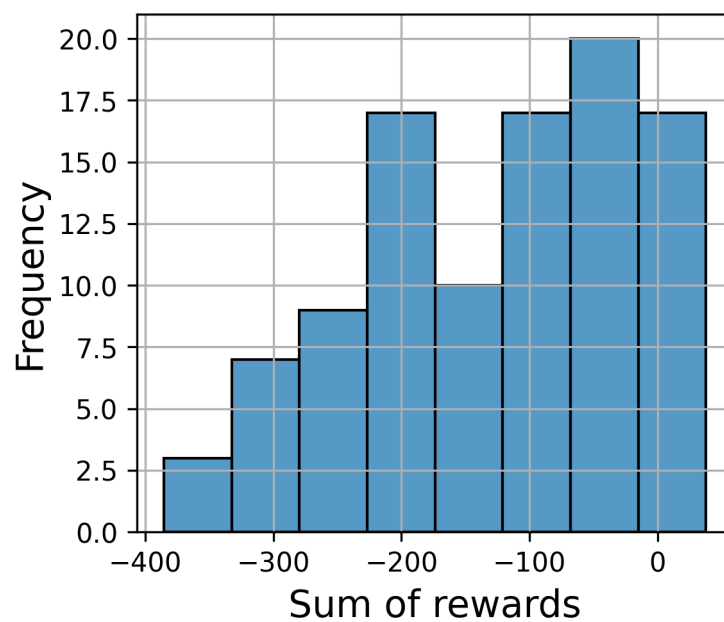


Figura 3: Histograma obtenido por el agente en el ambiente lunar lander, entrenado en 100 episodios.

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

```

```

5      '''
6      # Create agent
7      print('Loading agent and environment...')
8      agent = load_agent_SarsaCS()
9      # Create train-and-run object
10     act = load_act(agent, 'Sarsa')
11     # Sweep alpha
12     print('Sweeping alpha...')
13     alphas = [0.2/8, 0.4/8, 0.8/8]
14     act.sweep(parameter='alpha', values=alphas, num_simulations=10)
15     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.

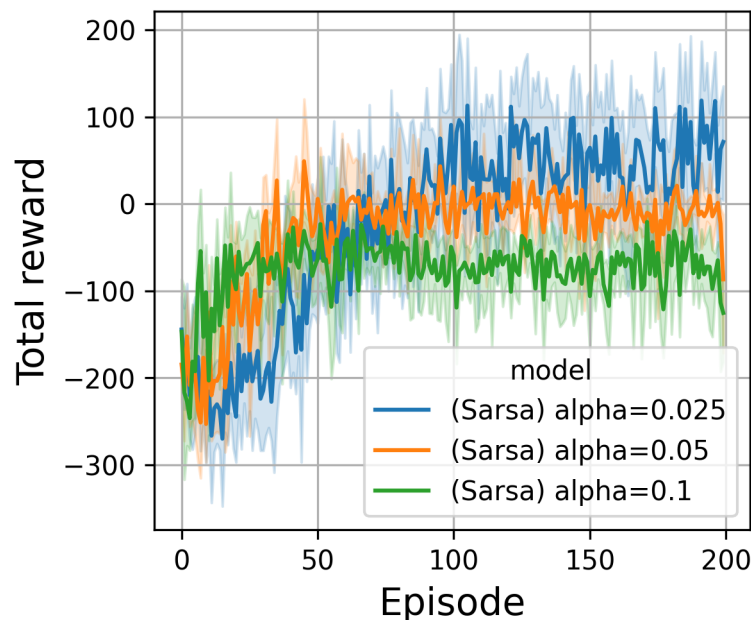


Figura 4: Comparación de reward para las diferentes combinaciones de valores para α , en el ambiente lunar lander.

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`.

```

1 def load_agent_nStepCS() -> nStepCS:
2     '''
3     Creates a nStepCS agent with a set
4     of parameters determined inset
5     '''
6     # Define parameters
7     parameters = {"numDims":8,\
8                   "nA":4,\
9                   "gamma":1,\
10                  "epsilon":0.1,\
11                  "alpha":0.5,\
12                  "numTilings":8,\
13                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\
14                  "n":4,\
15                  "scaleFactors":[\
16                      {"min":-90., "max":90.}, # x coordiantes
17                      {"min":-90., "max":90.}, # y coordiantes
18                      {"min":-5., "max":5.}, # x velocity
19                      {"min":-5., "max":5.}, # y velocity
20                      {"min":-3.1415927, "max":3.1415927}, # object angle
21                      {"min":-5., "max":5.}, # angular velocity
22                      {"min":0., "max":1.}, # boolean leg 1
23                      {"min":-0., "max":1.}, # boolean leg 2
24                  ]
25            }
26
27     # Create approximating function
28     Q = TilesQ(parameters=parameters)
29     # Create agent
30     return nStepCS(parameters, Q)
31
32 def sweep_nStep():
33     '''
34     Runs a sweep over alpha
35     '''
36     # Create agent
37     print('Loading agent and environment...')
38     agent = load_agent_nStepCS()
39     # Create train-and-run object
40     act = load_act(agent, 'Sarsa')
41     # Sweep alpha
42     print('Sweeping alpha...')
43     alphas = [0.2/8, 0.4/8, 0.8/8]
44     n = [2, 4, 8]
45     act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n,
46               num_simulations=5)
47     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 contrario, con $n = 4$, en

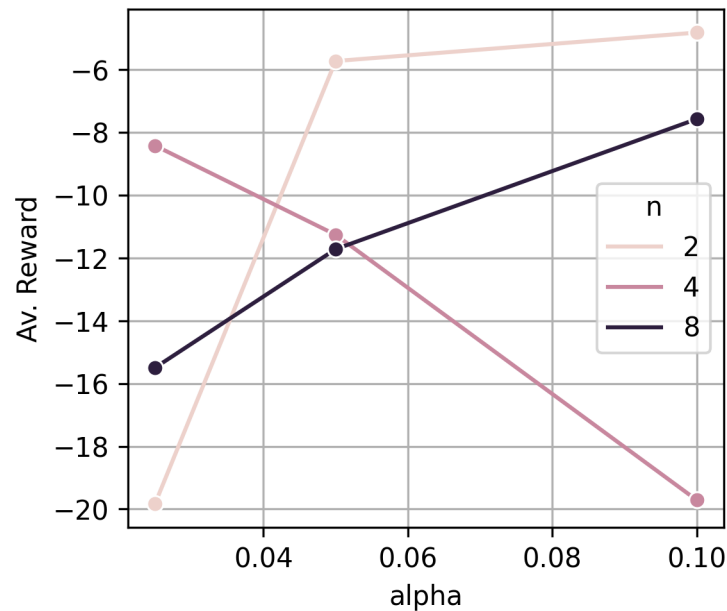


Figura 5: Comparación de reward para las diferentes combinaciones de valores para α y n , en el ambiente mountaun car.

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

```

1 def train_and_compare():
2     # Create agent
3     agent_SARSA = load_agent_SarsaCS()
4     # Create train-and-run object
5     act = load_act(agent_SARSA, 'Sarsa')
6     # Train the SARSA agent
7     print('Training SARSA agent...')
8     act.train()
9     # Testing the agent
10    print('Testing SARSA agent...')
11    act.num_episodes = 100
12    act.test(to_df=True)
13    df_sarsa = act.data
14    # Create agent

```

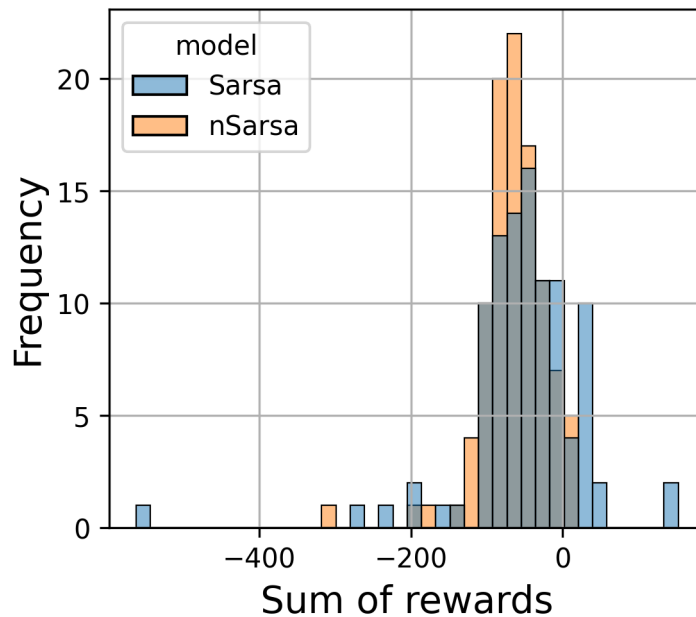


Figura 6: Histograma para la comparación de los dos métodos de entrenamiento.

```

15 agent_nSARSA = load_agent_nStepCS()
16 # Create train-and-run object
17 act = load_act(agent_nSARSA, 'nSarsa')
18 # Train the agent
19 print('Training nSARSA agent...')
20 act.train()
21 # Testing the agent
22 print('Testing nSARSA agent...')
23 act.num_episodes = 100
24 act.test(to_df=True)
25 df_nsarsa = act.data
26 # Compare performances
27 df = pd.concat([df_sarsa, df_nsarsa], ignore_index=True)
28 p = Plot(df)
29 p.plot_histogram_rewards(act.file_compare_hist)
30 print(f'Plot saved to {act.file_compare_hist}')
31 p.plot_rewards(act.file_compare_rew)
32 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

```
1 from Utils.train import TrainRun
2 from Agents.agentsCS import SarsaCS, nStepCS
3 from Agents.linearQ import TilesQ
4 from Utils.interpreters import gym_interpreter1
5 import matplotlib.pyplot as plt
6 import pandas as pd
7 from Utils.utils import Plot
8
9
10
11 def try_env():
12     '''
13     Loads an agent and runs it
14     without learning on the
15     Lunar lander environment
16     '''
17     print('Loading agent and environment...')
18     # Create agent
19     agent = load_agent_SarsaCS()
20     # Create train-and-run object
21     act = load_act(agent, 'Sarsa')
22     # Show the untrained agent
23     print('Showing the untrained agent...')
24     act.run(visual=True)
25     print('Done!')
26
27
28 def train_and_run_SARSA():
29     '''
30     Trains a SARSA agent on the Mountain Car
31     '''
32     # Create agent
33     print('Loading agent and environment...')
34     agent = load_agent_SarsaCS()
35     # Create train-and-run object
36     act = load_act(agent, 'Sarsa')
37     # Train the agent
38     print('Training the agent...')
39     act.train()
40     # Show the trained agent
41     print('Showing the trained agent...')
42     act.run()
43     # Testing the agent
44     print('Testing the agent...')
45     act.test()
46     print('Done!')
47
48 def sweep_SARSA():
49     '''
50     Runs a sweep over alpha
51     '''
```

```

52 # Create agent
53 print('Loading agent and environment...')
54 agent = load_agent_SarsaCS()
55 # Create train-and-run object
56 act = load_act(agent, 'Sarsa')
57 # Sweep alpha
58 print('Sweeping alpha...')
59 alphas = [0.2/8, 0.4/8, 0.8/8]
60 act.sweep(parameter='alpha', values=alphas, num_simulations=50)
61 print('Done!')
62
63
64 def load_agent_SarsaCS() -> SarsaCS:
65     '''
66     Creates a SarsaCS agent with a set
67     of parameters determined inset
68     '''
69     # Define parameters
70     parameters = {"numDims":8,\
71                  "nA":4,\
72                  "gamma":1,\
73                  "epsilon":0.1,\
74                  "alpha":0.1,\
75                  "numTilings":8,\
76                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\
77                  "scaleFactors":[\
78                      {"min":-90., "max":90.}, # x coordiantes
79                      {"min":-90., "max":90.}, # y coordiantes
80                      {"min":-5., "max":5.}, # x velocity
81                      {"min":-5., "max":5.}, # y velocity
82                      {"min":-3.1415927, "max":3.1415927}, # object angle
83                      {"min":-5., "max":5.}, # angular velocity
84                      {"min":0., "max":1.}, # boolean leg 1
85                      {"min":-0., "max":1.}, # boolean leg 2
86                  ]
87     }
88     # Create approximating function
89     Q = TilesQ(parameters=parameters)
90     # Create agent
91     return SarsaCS(parameters, Q)
92
93
94 def load_agent_nStepCS() -> SarsaCS:
95     '''
96     Creates a SarsaCS agent with a set
97     of parameters determined inset
98     '''
99     # Define parameters
100     parameters = {"numDims":8,\
101                  "nA":4,\
102                  "gamma":1,\
103                  "epsilon":0.1,\
104                  "alpha":0.1,\
105                  "numTilings":8,\
106                  "numTiles":[10, 10,10, 10,10, 10,10, 10],\

```

```

1107         "scaleFactors":[\
1108             {"min":-90., "max":90.}, # x coordiantes
1109             {"min":-90., "max":90.}, # y coordiantes
1110             {"min":-5., "max":5.}, # x velocity
1111             {"min":-5., "max":5.}, # y velocity
1112             {"min":-3.1415927, "max":3.1415927}, # object angle
1113             {"min":-5., "max":5.}, # angular velocity
1114             {"min":0., "max":1.}, # boolean leg 1
1115             {"min":-0., "max":1.}, # boolean leg 2
1116         ]
1117     }
1118     # Create approximating function
1119     Q = TilesQ(parameters=parameters)
1120     # Create agent
1121     return SarsaCS(parameters, Q)
1122
1123
1124
1125 def load_act(agent, model_name:str) -> TrainRun:
1126     '''
1127     Creates a train-and-run object with
1128     parameters given inset
1129     '''
1130     act = TrainRun(\
1131         env_name = 'LunarLander-v2',\
1132         state_interpreter=gym_interpreter1,\
1133         agent=agent,\
1134         model_name=model_name,\
1135         num_rounds=1000, # by default was in 1000
1136         num_episodes=200
1137     )
1138     return act
1139
1140 def sweep_nStep():
1141     '''
1142     Runs a sweep over alpha
1143     '''
1144     # Create agent
1145     print('Loading agent and environment...')
1146     agent = load_agent_nStepCS()
1147     # Create train-and-run object
1148     act = load_act(agent, 'Sarsa')
1149     # Sweep alpha
1150     print('Sweeping alpha...')
1151     alphas = [0.2/8, 0.4/8, 0.8/8]
1152     n = [2, 4, 8]
1153     act.sweep2(parameter1='alpha', values1=alphas, parameter2='n', values2=n,
1154               num_simulations=5)
1155     print('Done!')
1156
1157 def train_and_compare():
1158     # Create agent
1159     agent_SARSA = load_agent_SarsaCS()
1160     # Create train-and-run object

```

```

161 act = load_act(agent_SARSA, 'Sarsa')
162 # Train the SARSA agent
163 print('Training SARSA agent...')
164 act.train()
165 # Testing the agent
166 print('Testing SARSA agent...')
167 act.num_episodes = 100
168 act.test(to_df=True)
169 df_sarsa = act.data
170 # Create agent
171 agent_nSARSA = load_agent_nStepCS()
172 # Create train-and-run object
173 act = load_act(agent_nSARSA, 'nSarsa')
174 # Train the agent
175 print('Training nSARSA agent...')
176 act.train()
177 # Testing the agent
178 print('Testing nSARSA agent...')
179 act.num_episodes = 100
180 act.test(to_df=True)
181 df_nsarsa = act.data
182 # Compare performances
183 df = pd.concat([df_sarsa, df_nsarsa], ignore_index=True)
184 p = Plot(df)
185 p.plot_histogram_rewards(act.file_compare_hist)
186 print(f'Plot saved to {act.file_compare_hist}')
187 p.plot_rewards(act.file_compare_rew)
188 print(f'Plot saved to {act.file_compare_rew}')

```

Listing 5: Implementación del ambiente lunar_lander.py