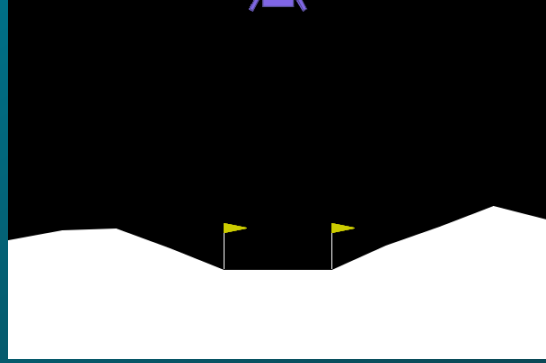
ISIA



Customizing OpenAi environment

Projeto realizado por:
Leonardo Regadas, up202108144
João Silva, up202107600
Diogo Miranda, up202106743

Descrição do ambiente

Este ambiente discreto trata-se de um problema de otimização da trajetória de uma nave. Neste problema, o motor da nave pode-se encontrar em dois estados: ligado e desligado.
A nave tem como objetivo pousar numa plataforma (cujo centro tem coordenadas (0,0)), com um combustível infinito.

A nave pode tomar uma das seguintes decisões discretas:
- 0: não fazer nada;
- 1: ligar o motor “esquerdo”;
- 2: ligar o motor principal;
- 3: ligar o motor “direito”.

Descrição do ambiente

O espaço de observação é constituído por vetor com 8 dimensões: as coordenadas x e y da nave, as velocidades lineares correspondentes, o ângulo, a velocidade angular, e dois valores booleanos responsáveis por ver se cada uma das pernas da nave se encontra em contacto com uma superfície ou não.

Para cada passo, o valor de recompensa (reward):
- altera tendo em conta a distância até à plataforma de pouso;
- altera de acordo com a velocidade da nave;
- diminui quanto mais inclinada a nave estiver;
- aumenta em 10 pontos para cada perna que pouse no chão;
- aumenta em 0.03 em cada frame em que algum motor lateral é usado;
- aumenta em 0.3 em cada frame em que o motor principal é usado.

Aterrar de forma certa fornece + 100 pontos e de forma errada -100 pontos.

Mudanças no ambiente

Como objetivo de melhorar a eficácia da nave e também considerando situações de vida real, decidimos alterar alguns valores das recompensas, entre eles:

Aterragem soft da nave

Reward responsável por penalizar de uma forma proporcional à velocidade vertical no momento da aterragem.

Distância ao centro da plataforma

Reward responsável por penalizar de uma forma proporcional à distância da nave ao centro da plataforma

```python
# Create a custom reward wrapper
class CustomRewardWrapper(gym.Wrapper):
    def step(self, action):
        obs, reward, done, info, truncated = super().step(action)
        center_position = [0, 0] # Central position of the landing pad
        reward = self.modify_reward(reward, obs, center_position) # Modify the reward
        return obs, reward, done, info, truncated

    # Modify the reward function as needed
    @staticmethod
    def modify_reward(reward, obs, center_position):
        agent_position = obs[:2] # Get the agent's position
        distance = np.linalg.norm(np.array(agent_position) - np.array(center_position)) # Calculate the distance from the center

        # Get the vertical velocity of the lander
        vertical_velocity = obs[3]

        # Modify the reward to account for distance and vertical velocity
        return reward - distance - 0.1 * np.abs(vertical_velocity)
```

Algoritmos de RL escolhidos

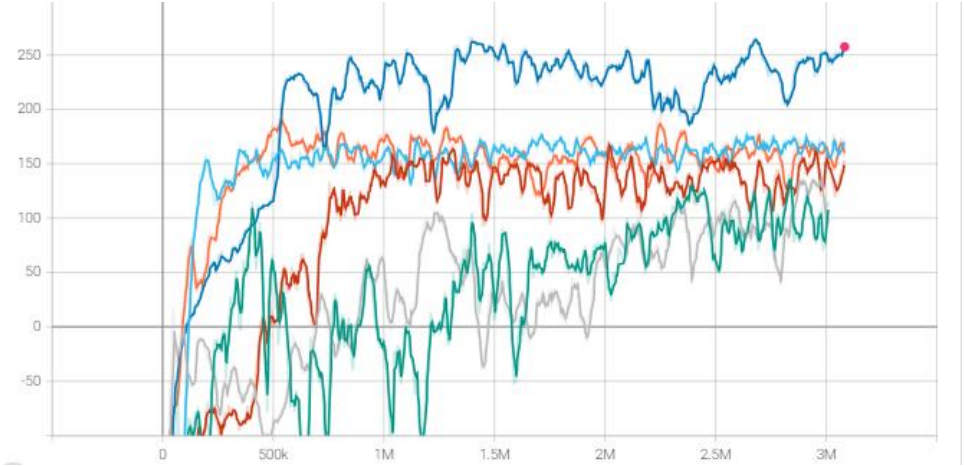
PPO

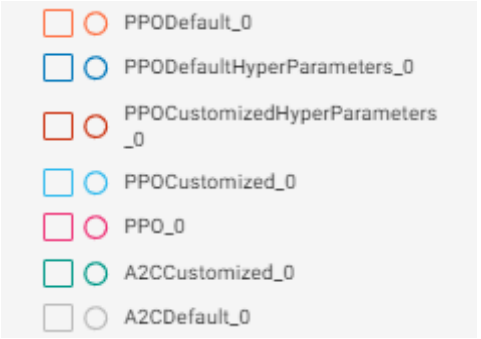
Recolhe um conjunto de experiências com o objetivo de dar update à política de decision-making. Após dar update, as experiências já usadas são trocadas por outro conjunto de forma a dar outro update ao seu decision-making, sem que altere muito em relação ao modelo anterior.

A2C

Implementação determinística que espera pelo término da tarefa específica do agente antes de dar update. É uma alternativa do A3C policy gradient.

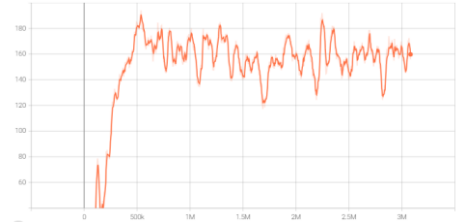
Resultados experimentais



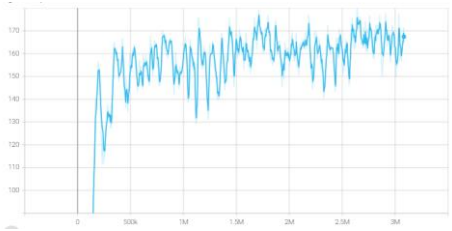


Resultados experimentais

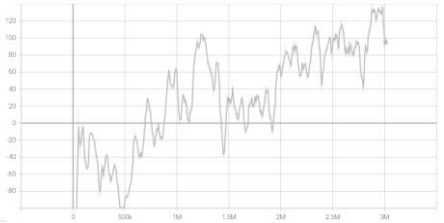
PPO default:



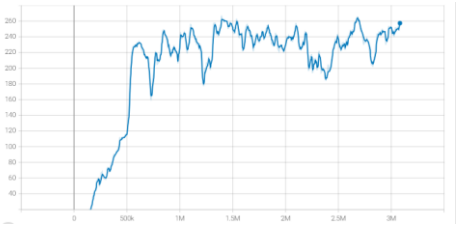
PPO customized:



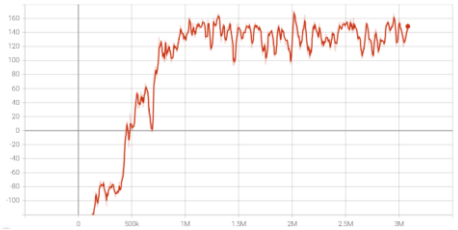
A2C default



PPO default with hyperparametes:



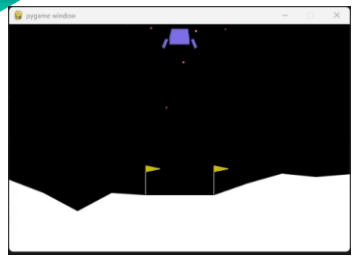
PPO customized with hyperparametes:



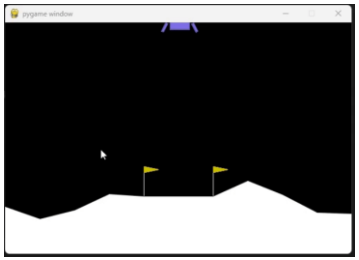
A2C customized

Resultados experimentais

PPO default:

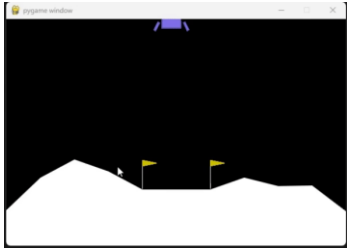


PPO customized:

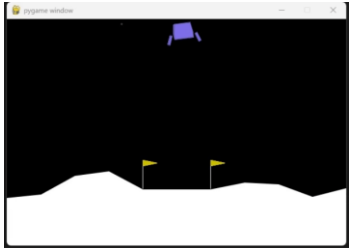


A2C default

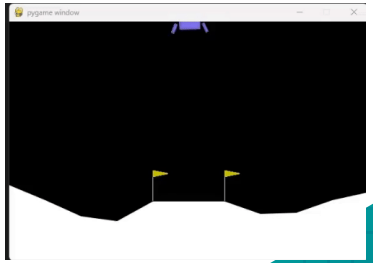
PPO default with hyperparametes:



PPO customized with hyperparametes:

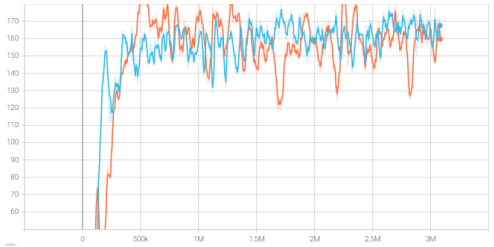


A2C customized

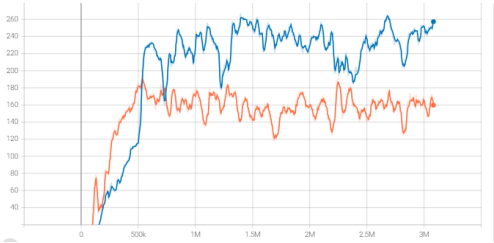


Resultados experimentais

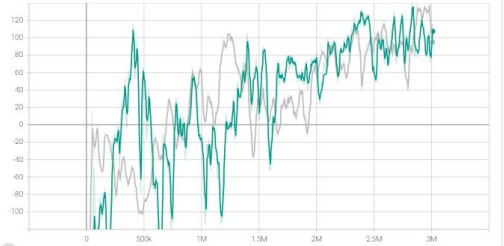
PPO default x
PPO customized

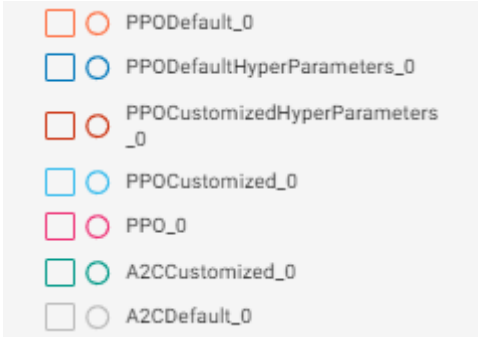


PPO default x
PPO hyperparams



A2C default x
A2C customized





Conclusão

As alterações dos rewards que realizamos impactaram a forma como o nave foi aprendendo a pousar na plataforma ao longo das iterações. Além disso, verificamos a importância dos hyper-parameters neste tipo de problemas de reinforcement learning.

Acreditamos que o algoritmo PPO tenha sido mais eficiente no treino do modelo, uma vez que os gráficos de A2C permaneceram sempre inferiores em termos de valores de rewards.

Em trabalhos futuros pretendemos explorar ainda mais alguns algoritmos, alterando sempre os valores de rewards e os hyperparameters para ver o impacto de cada na aprendizagem dos modelos. Um maior número de iterações também seria fulcral para criar resultados mais precisos. Poderiam também ser adicionados obstáculos no ambiente em questão para conferir uma complexidade extra ao problema em questão.