



Instituto Superior de Engenharia de Lisboa

Ambientes Virtuais Interativos e Inteligentes

Mestrado Engenharia Informática e Multimédia

Projeto 3

Aprendizagem por Reforço com ML-Agents

Semestre de Verão, 2022/2023

Nome: Gonçalo Fonseca | Número: A50185

Nome: Ruben Santos | Número: A49063

Índice

1	Introdução	1
2	Criação do ambiente e agente	2
2.1	Agente	3
2.1.1	Ações	3
2.1.2	Observações	3
2.1.3	Recompensas	6
2.1.4	Validação do setup de treino usando o modo Heurístico	8
3	Treino do Agente	9
4	Avaliação do desempenho	13
5	Conclusão	15
	Referências	16
6	ANEXOS	17

1 Introdução

O objetivo deste relatório é aplicar os conceitos de **Aprendizagem por Reforço** utilizando a biblioteca **ML-Agents** num ambiente criado no **Unity**. O trabalho consiste na criação de um ambiente de aprendizagem virtual no qual um agente será treinado para executar uma determinada tarefa.

Os ML-Agents são um projeto *open source* que permite que os jogos e as simulações sirvam de ambiente para treinar agentes inteligentes utilizando a aprendizagem por reforço profundo e a aprendizagem por imitação [1]. Com os *ML-Agents*, é possível criar uma jogabilidade mais apelativa e uma experiência de jogo melhorada, ensinando os agentes inteligentes a "aprender" através de uma combinação das aprendizagens referidas anteriormente [2].

A Aprendizagem por reforço é um método baseado na recompensa de comportamentos desejados e/ou punição de comportamentos indesejados. Em geral, um agente de aprendizagem por reforço é capaz de perceber e interpretar o seu ambiente, tomar ações e aprender através de tentativa e erro.

O método para recompensar o comportamento desejado atribui valores positivos às ações desejadas para encorajar o agente e valores negativos aos comportamentos indesejados. Tal faz com que o agente procure pela maior recompensa a longo prazo a fim de encontrar uma solução ótima. [6]

Em traços gerais, a figura 1 que se segue, representa o modo de funcionamento da aprendizagem por reforço.

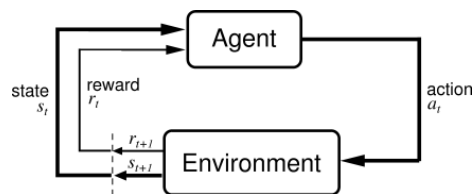


Figura 1: Funcionamento da Aprendizagem por Reforço

2 Criação do ambiente e agente

Para que fosse possível criar a simulação e treinar um agente usando os modelos de ML-Agents, seria necessário criar um ambiente para o qual o agente navega-se. Para tal, usou-se algumas ideias de labirintos simples, e decidiu-se criar o ambiente que se apresenta na Figura 2.

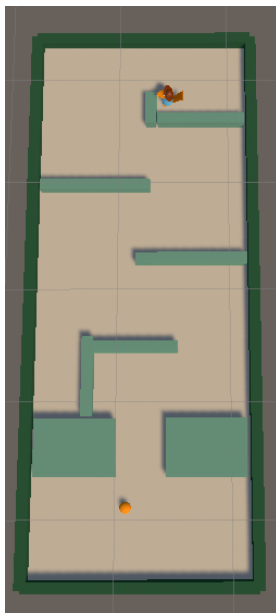


Figura 2: Ambiente

Este labirinto, embora simplificado (para um pessoa), apresenta uma configuração que requer que o agente faça várias mudanças de direção ao longo do caminho para percorrê-lo com sucesso. Essas mudanças serão observadas em maior detalhe posteriormente.

Além disso, para realizar a simulação, é essencial ter um agente e um alvo definidos. Nesse contexto, foi criado o agente, conforme apresentado na figura 3 (localizado no topo da figura 2). O objetivo do agente é alcançar o alvo, representado pelo objeto na figura 4 (localizado na parte inferior da figura 2). Essa configuração estabelece a dinâmica da interação entre o agente e o alvo, onde o agente deve tomar decisões e navegar pelo ambiente para atingir o objetivo desejado.



Figura 3: Agente



Figura 4: Alvo

2.1 Agente

Um agente que utiliza *reinforcement learning* para aprender as ações corretas, efetua observações, toma decisões e realiza ações, recebendo recompensas em consequência dessas ações.

2.1.1 Ações

No processo de aprendizagem do agente, foi adotada a abordagem de fornecer ao agente duas ações contínuas. Essas ações são representadas pelas variáveis **move** (para mover para frente e para trás) e **rotate** (para mover para a esquerda e para a direita), conforme ilustrado na figura 5 seguinte.

```
public override void OnActionReceived(ActionBuffers actions)
{
    float movespeed = 30f;
    float rotatespeed = 250f;
    float move = actions.ContinuousActions[0];
    float rotate = actions.ContinuousActions[1];
    transform.Rotate(new Vector3(0, rotate * Time.fixedDeltaTime * rotatespeed, 0));
    transform.localPosition += transform.forward * move * Time.fixedDeltaTime * movespeed;
}
```

Figura 5: Ações do agente

Essa escolha pelas ações contínuas permite uma maior flexibilidade no comportamento do agente, permitindo-o ajustar a sua velocidade de movimento e direção em resposta às observações e objetivos do ambiente.

2.1.2 Observações

As observações no contexto de *reinforcement learning* são informações recebidas pelo agente sobre o estado do ambiente. Essas observações podem ser parciais ou completas e são utilizadas pelo agente para tomar decisões em relação à sua próxima ação. O tipo e a quantidade de informações contidas nas observações variam de acordo com o problema específico de aprendizagem por reforço.[5]

Para que o agente realize as observações, foi utilizado o componente *Ray Perception Sensor 3D* com os parâmetros ilustrados na figura 6 seguinte:

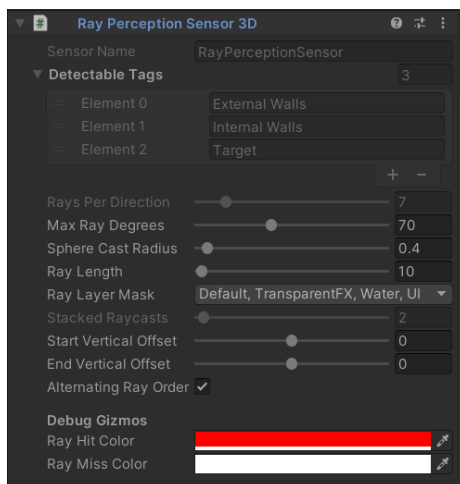


Figura 6: Observações da componente Ray Perception Sensor 3D

O agente utiliza a componente *Ray Perception Sensor 3D* para realizar as observações, pois esta componente possui diversos parâmetros configuráveis para otimizar a detecção de objetos. Para definir quais os objetos o agente deve distinguir, foi criada uma lista denominada *Detectable Tags*, que contém as *tags* específicas atribuídas aos objetos relevantes para a tarefa do agente.

Para determinar a resolução e a precisão das observações realizadas pelo agente, utilizou-se o parâmetro **Rays Per Direction** que se refere ao número de raios que são lançados em cada direção pelo componente *Ray Perception Sensor 3D*. Um valor maior de *Rays Per Direction* resulta num maior número de raios lançados em cada direção, o que permite uma percepção mais detalhada do ambiente, capturando informações mais detalhadas sobre a presença e a localização dos objetos.

Outro parâmetro relevante é o *Observation Stacks*, que determina a quantidade de resultados anteriores que são utilizados como *stack* para realizar a projeção. Optou-se por utilizar 2 stacks, visando equilibrar a complexidade computacional sem comprometer significativamente o desempenho do agente.

Essas configurações foram definidas com o objetivo de melhorar a performance do agente no processo de aprendizagem por reforço, permitindo identificar os objetos alvo (*target*) e distinguir entre paredes internas (*Internal walls*) e paredes externas (*External walls*) no ambiente.

Para calcular o total de observações do *Ray Perception Sensor 3D*, usa-se a seguinte fórmula [4]:

$$(\text{Observation Stacks}) * (1 + 2 * \text{Rays Per Direction}) * (\text{Num Detectable Tags} + 2)$$

Figura 7: Fórmula do cálculo do número total de observações

Então, usando os dados da figura 6 para calcular o número total de observações, obtém-se: $(2) * (1 + 2 * 7) * (3 + 2) = 150$.

Um exemplo visual que ilustra a utilização dos raios sensoriais do *Ray Perception Sensor 3D* no Unity está apresenta na figura 8 abaixo.

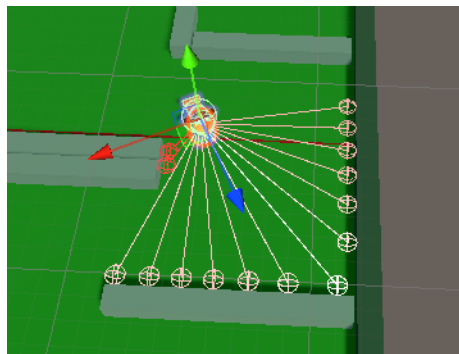


Figura 8: Raios sensoriais do *Ray Perception Sensor 3D*

Além do *Ray Perception Sensor 3D*, também foi utilizado o método **Agent.CollectObservations()** para permitir que o agente realize observações vetoriais. Esta função é especialmente útil quando o agente precisa de analisar os aspectos do ambiente que são numéricos e não visuais, como determinar se já entrou numa determinada área ou se saiu dela.

```
public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(targetRoom);
}
```

Figura 9: Função Agent.CollectObservations()

Ao utilizar essa observação vetorial, estamos a indicar ao agente que deve prestar atenção à variável *targetRoom* e às suas mudanças.

2.1.3 Recompensas

Por fim, temos as recompensas (*rewards*), que são um mecanismo de incentivo que indica ao agente o que é correto e o que é errado, através da recompensa e do castigo de valores [3]. Existem várias ocasiões onde essa recompensa (positiva ou negativa) acontece.

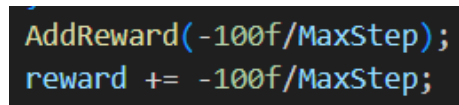
Neste projeto, foram fornecidos ao agente dois tipos de recompensas: **as recompensas parciais** e **as recompensas desejadas**.

As **recompensas parciais** são recompensas imediatas concedidas ao agente com base nas suas ações e na interação com o ambiente. Essas recompensas não impedem o agente de prosseguir com as suas ações, mas têm como função fornecer informações que ajudam a indicar um melhor “caminho” para o agente.

Por outro lado, **as recompensas desejadas** são recompensas que o agente procura ativamente alcançar durante o treino, sendo estas definidas com base nos objetivos específicos do projeto e podem incluir recompensas positivas para alcançar determinadas metas ou recompensas negativas para evitar comportamentos indesejados. Essas recompensas desejadas são fundamentais para direcionar o agente em direção ao comportamento desejado e ajudá-lo a aprender a tomar decisões que maximizem a sua performance no ambiente.

Em seguida, encontram-se as várias recompensas.

1. **Recompensa parcial negativa:** Para cada ação tomada pelo agente, o mesmo é incentivado a mover-se e a tomar decisões mais rapidamente, evitando que fique parado no mesmo local.



```
AddReward(-100f/MaxStep);  
reward += -100f/MaxStep;
```

Figura 10: Recompensa parcial negativa

2. **Recompensa parcial targetRoom:** Foi atribuída uma recompensa positiva para incentivar o agente a atravessar uma área específica no ambiente, como por exemplo, o plano Z em -1. No entanto, como esse local gerava uma recompensa positiva para o agente, foi necessário criar uma recompensa negativa adicional para evitar que o agente ficasse preso nessa região. A figura 11 apresenta as recompensas dadas, e a figura 12 apresenta as áreas construídas para essas mesmas recompensas como foi referido.


```

if(!targetRoom && transform.localPosition.z < -1)
{
    AddReward(50f);
    targetRoom = true;
    reward += 50f;
}
if(targetRoom && transform.localPosition.z > 1)
{
    AddReward(-100f);
    targetRoom = false;
    reward += -100f;
}

```

Figura 11: Recompensa parcial targetRoom

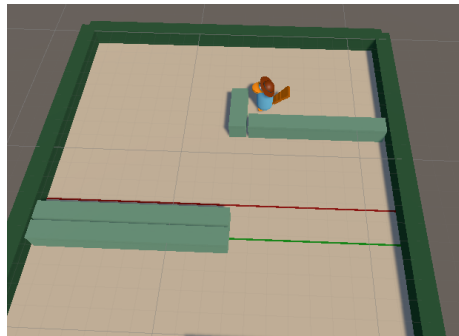


Figura 12: Demonstração do local da recompensa parcial targetRoom

3. Por fim, temos **as recompensas desejadas**, que, como o próprio nome sugere, representam os objetivos que desejamos que o agente compreenda plenamente. Nesse sentido, recompensamos positivamente quando o agente atinge o alvo desejado e recompensamos negativamente quando ele colide com as paredes.

```

private void OnTriggerEnter(Collider other) {
    // string objetoNome = other.gameObject.name;
    // print("Trigger collision:" + objetoNome);
    if (other.gameObject.CompareTag("Target"))
    {
        AddReward(+200f);
        renderFloor.material = winMaterial;
        reward += 200f;
        EndEpisode();
        // print("Reward: " + reward );
    }
    if(other.gameObject.CompareTag("External Walls") || other.gameObject.CompareTag("Internal Walls"))
    // if(other.gameObject.CompareTag("Wall"))
    {
        AddReward(-100f);
        renderFloor.material = loseMaterial;
        reward += -100f;
        EndEpisode();
        // print("Reward: " + reward );
    }
}

```

Figura 13: Recompensas desejadas

2.1.4 Validação do setup de treino usando o modo Heurístico

Antes de iniciar o treino do agente, foi realizado um processo heurístico para verificar e ajustar manualmente o comportamento do agente, a fim de identificar possíveis erros no ambiente e no próprio agente.

Um exemplo em que a validação do *setup* de treino usando o modo Heurístico foi útil foi na verificação de colisões. Durante o desenvolvimento do projeto, antes de treinar o agente no ambiente, optou-se por habilitar o modo Heurístico devido a ocorrências estranhas, como o agente encerrando o episódio prematuramente, mesmo sem ter concluído todos os *steps* da época. Após investigação, constatou-se que o *Floor* localizado abaixo do agente estava configurado com uma componente de colisão ativado, o que acionava a função **OnTriggerEnter(Collider other)** e resultava no término antecipado do episódio.

Outro exemplo em que a validação do *setup* de treino usando o modo Heurístico foi útil foi ao criar um agente com características mais complexas. Ao adicionar objetos como olhos, boca e chapéu ao agente, os raios do sensor de percepção eram raios que colidiam com esses objetos, resultando em dificuldades de aprendizado do ambiente. Por meio do modo Heurístico, foi possível observar diretamente como os raios de percepção interagem com os objetos adicionados ao agente. Essa validação revelou que a presença desses objetos estava a interferir na percepção adequada do ambiente, afetando negativamente a aprendizagem do agente.

3 Treino do Agente

Após a criação do ambiente e do agente, foram realizados diferentes treinos para permitir que o agente aprendesse e se familiarizasse com o ambiente, assim como os locais que desejávamos que ele aprendesse a visitar.

Inicialmente, foi utilizado um ambiente mais simples para ensinar conceitos básicos ao agente, e posteriormente foram gradualmente introduzidas complexidades ao ambiente, como se pode observar na figura 14, sendo que a imagem 1 é a versão mais antiga e simples, e a versão 5 é a versão mais recente e complexa.

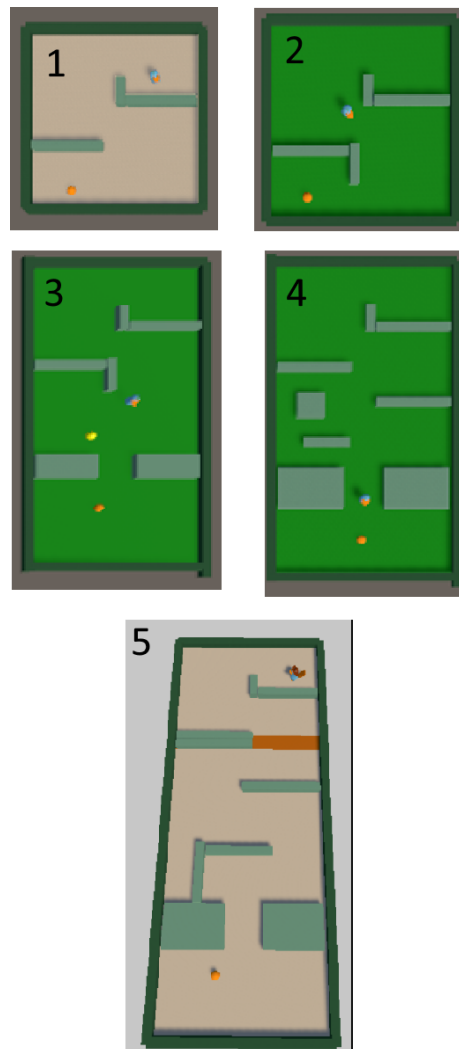


Figura 14: Evolução do ambiente de aprendizagem

Para iniciar o treino, utilizou-se um ficheiro de configuração chamado **RayAgent.yaml** para configurar e controlar determinados parâmetros específicos do treino.

```
default_settings: null
behaviors:
  RayAgent:
    trainer_type: ppo
    hyperparameters:
      batch_size: 10
      buffer_size: 100
      learning_rate: 0.0003
      beta: 0.005
      epsilon: 0.2
      lambda: 0.99
      num_epoch: 3
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 128
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
      # gail:
      #   strength: 0.1
      #   demo_path: Demos\AgentDemo_1.demo
    init_path: null
    keep_checkpoints: 5
    checkpoint_interval: 4000
    max_steps: 500000
    time_horizon: 64
    summary_freq: 5000
    threaded: false
    self_play: null
    behavioral_cloning: null
```

Figura 15: Parâmetros no ficheiro **RayAgent.yaml**

No ficheiro de configuração, podemos encontrar informações importantes, como o tipo de algoritmo utilizado, os hiperparâmetros de treino e outros parâmetros relevantes.

No treino do agente, foi utilizado o algoritmo chamado **Proximal Policy Optimization (PPO)**, que é uma técnica de aprendizagem por reforço pertencente à família de algoritmos de gradiente de política. O **PPO** é especialmente projetado para treinar agentes em ambientes complexos, onde a política do agente é atualizada iterativamente para melhorar o desempenho. O **PPO** utiliza otimização baseada em gradiente e tem como objetivo maximizar a função de recompensa acumulada ao longo do tempo. Essa abordagem permite ao algoritmo explorar de maneira eficiente o ambiente e encontrar as melhores ações para alcançar um desempenho otimizado. Com a sua capacidade de maximizar a recompensa cumulativa, o **PPO** demonstra uma excelente exploração e eficiência na aprendizagem do agente. [7]

O ficheiro **YAML** fornece ainda uma variedade de parâmetros para controlar o treino do agente usando o algoritmo **PPO**. Esses parâmetros incluem configurações como o **batch_size** (usado para atualizar os pesos da rede neural), **learning_rate** (taxa de aprendizado usada na otimização dos pesos), **network_settings** (configurações relacionadas à estrutura da rede neural), **reward_signals** (configurações relacionadas aos sinais de recompensa usados durante o treino), entre muitos outros. Cada um desses parâmetros desempenha um papel fundamental na configuração e controlo do treino do agente, afetando aspetos como a atualização dos pesos da rede neural, a exploração do ambiente e a melhoria do desempenho ao longo do tempo.

O *reinforcement learning* possui algumas limitações, entre elas a necessidade de recompensas adequadas e o longo do tempo da aprendizagem inicial.

Para superar essas limitações, foi desenvolvida a abordagem **GAIL** (*Generative Adversarial Imitation Learning*) que utiliza demonstrações para treinar agentes, permitindo a imitação de comportamentos e superando a necessidade de treino por tentativa e erro. Essa abordagem tem se mostrado promissora, capacitando agentes a aprender tarefas complexas e adquirir habilidades, o que demonstra uma alternativa vantajosa ao *reinforcement learning* tradicional.

O **GAIL** (*Generative Adversarial Imitation Learning*) é um algoritmo de aprendizagem por imitação que utiliza a abordagem *Adversarial* para treinar agentes de aprendizagem por reforço a partir de demonstrações. Diferente da aprendizagem por reforço tradicional, onde o agente aprende a partir de recompensas projetadas, o **GAIL** permite que o agente aprenda diretamente com as ações demonstradas.

Neste trabalho, adotou-se a estratégia GAIL, começando com o uso do modo heurístico no ambiente para gravar dados usando o método *Demonstration Recorder*. Estes metadados, incluindo observações e ações registradas, foram utilizados como ponto de partida para o treino do agente.

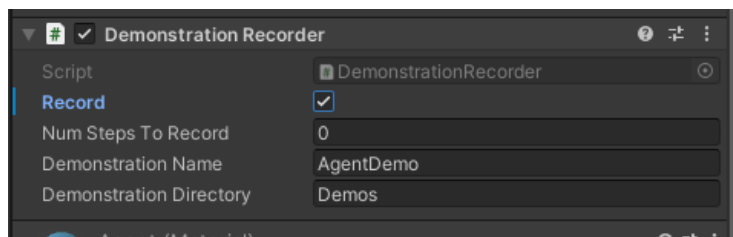


Figura 16: Componente *Demonstration Recorder*

No entanto, é importante destacar que o **GAIL** também apresenta as suas próprias limitações, como a necessidade de acesso a demonstrações e o risco de imitar comportamentos sub-ótimos se as demonstrações não forem representativas ou inadequadas. Para mitigar este risco, no ficheiro de configuração representado na figura 15, iniciou-se o treino com um valor alto para o parâmetro *strength* (0.8), permitindo que o agente aprendesse mais rapidamente. Gradualmente, reduzimos este parâmetro para valores mais baixos (0.3 e 0.1), de modo a evitar que o agente imitasse os comportamentos apresentados no ficheiro **Demos/AgentDemo_1.demo** criado por nós anteriormente.

Esta abordagem foi adotada para garantir que o agente se beneficiasse do conhecimento inicial fornecido pelas demonstrações, mas também tivesse a capacidade de aprender e melhorar a sua política ao longo do treino. Através do ajuste cuidadoso dos parâmetros, procuramos maximizar o desempenho e evitar possíveis ações associados à imitação de comportamentos sub-ótimos.

Em seguida, podemos examinar alguns dados sobre o ficheiro **AgentDemo_1.demo** (com pares de observações e ações), no qual o agente se baseou para iniciar o treino. Este ficheiro contém informações valiosas sobre as nossas demonstrações e servirá como referência para o agente durante o seu processo de aprendizagem.

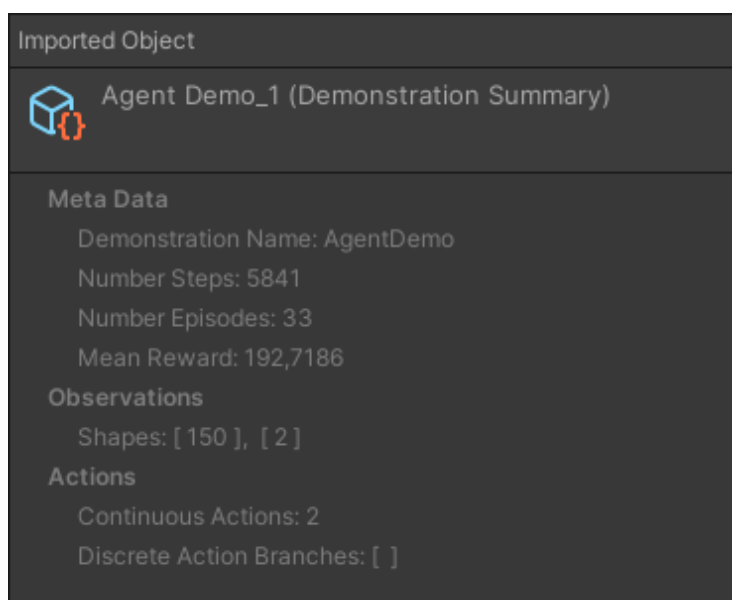


Figura 17: Estimativas do ficheiro *AgentDemo_1.demo*

Através do uso do TensorBoard, foi possível acompanhar a progressão do desempenho do agente durante o treino, como são apresentados nas figuras 19 a 26 presente nos Anexos.

O treino do ambiente final deste projeto é representado como **”Treino T7C RayAgent”** nas figuras em anexo, destacado a cor cinzenta. Observa-se uma evolução positiva ao longo das **500.000** épocas de treino, conforme evidenciado pelas mudanças nas recompensas acumuladas, no tamanho dos episódios entre outras métricas.

Na figura 20, pode-se observar que as recompensas acumulativas aumentaram ao longo do treino, aproximando-se do valor máximo de **250** em cada época. Isso indica que o agente está a aprender a obter recompensas mais altas e a realizar a tarefa de maneira mais eficiente.

Além disso, na figura 21, é possível notar uma diminuição no tamanho dos episódios ao longo do treino. Isso sugere que o agente está a ser mais eficaz a executar a tarefa de forma rápida e precisa, acertando nas ações necessárias com maior agilidade.

Estas tendências positivas nas recompensas acumulativas e no tamanho dos episódios indicam que o agente está a progredir e a melhorar as suas habilidades ao longo do treino. É um sinal encorajador de que as estratégias de aprendizagem implementadas levam a um desempenho mais eficiente e eficaz do agente no ambiente em questão.

4 Avaliação do desempenho

Para concluir a fase experimental do projeto, foi realizada uma avaliação do modelo a fim de determinar a sua performance. Para tal, o modelo foi testado utilizando um comportamento de agente no estado de inferência, obtendo-se os seguintes resultados:

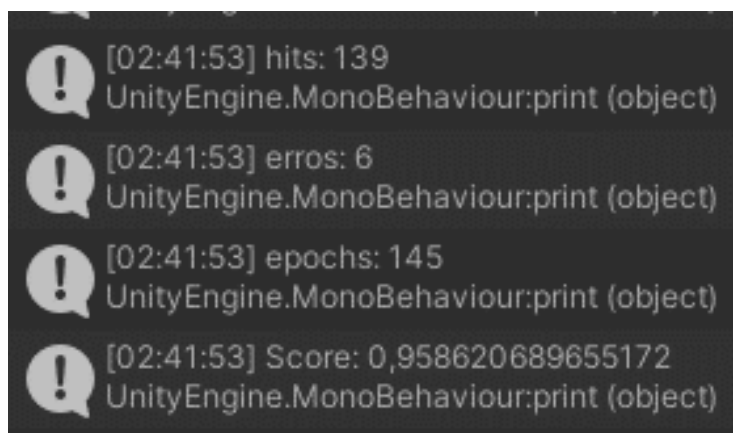


Figura 18: Resultados da avaliação do desempenho

Ao analisar os resultados apresentados na Figura 18, observa-se um desempenho positivo do agente, com os seguintes resultados:

- Número de épocas: 145
- Número de erros: 6
- Número de acertos: 139
- Percentagem de *score*: 95%

Com o **Número de erros** baixo (6 erros), ao longo das 145 épocas de treino, o agente demonstrou uma capacidade consistente de realizar a tarefa corretamente na maioria das vezes. Isso indica que o agente aprendeu a evitar ações que levariam a resultados indesejados, demonstrando uma compreensão eficaz das políticas necessárias para alcançar o sucesso.

Uma vez que o número de erros é baixo, a **Percentagem de Score** é alta. Com o *score* de **95%**, o agente apresentou uma taxa de sucesso significativamente alta em relação à tarefa em questão. Isso significa que o agente obteve um desempenho próximo da perfeição, realizando a tarefa corretamente na maior parte das vezes.

Os resultados positivos podem ser atribuídos a uma combinação de fatores-chave no treino do agente:

1. **Progressão gradual do ambiente:** O agente começou o seu treino num ambiente simples e gradualmente passou para versões mais complexas. Essa abordagem permite ao agente aprender progressivamente, começando com tarefas mais fáceis e avançando para desafios mais difíceis à medida que desenvolve suas habilidades. Essa progressão gradual permite que o agente se adapte e generalize o seu conhecimento para lidar com cenários mais complexos.
2. **Distribuição de recompensas:** A configuração das recompensas é crucial para orientar o comportamento do agente. A utilização de recompensas pequenas negativas a cada passo incentiva o agente a procurar ações eficientes e evitar ações desnecessárias. A recompensa positiva ao atingir o alvo e a penalização ao atingir as paredes também ajudam o agente a aprender a movimentar-se em direção ao objetivo desejado. A distribuição cuidadosa das recompensas cria um ambiente de aprendizagem favorável, onde o agente é incentivado a procurar por ações que levem ao sucesso.
3. **Utilização do GAIL:** A estratégia do **GAIL**, que utiliza demonstrações realizadas por pessoas para iniciar o treino do agente, pode ter contribuído para os bons resultados. Ao fornecer ao agente um conhecimento prévio, o agente começou o treino com uma base sólida, aprendendo a imitar comportamentos eficazes desde o início. A redução gradual do parâmetro *strength* ao longo do treino permitiu que o agente desenvolvesse gradualmente a sua própria política de ações, combinando a imitação inicial com a exploração e a aprendizagem contínua.

5 Conclusão

Com base neste trabalho, foi possível concluir que o uso de *reinforcement learning* num ambiente não tão simples, através do Unity com ML-Agents, é uma abordagem eficaz para treinar agentes autônomos, com algumas limitações. Através do uso de recompensas parciais e desejadas, o agente foi capaz de aprender a tomar decisões adequadas para navegar no nosso ambiente.

A utilização do algoritmo **PPO** permitiu otimizar o desempenho do agente ao longo do tempo, maximizando a função de recompensa acumulada. Além disso, a incorporação de técnicas como o *Ray Perception Sensor 3D* e observações vetoriais permitiram ao agente obter informações detalhadas sobre o ambiente, contribuindo para uma melhor tomada de decisão. Outro fator importante foi a exploração da abordagem GAIL da imitação, demonstrou ser uma estratégia promissora para superar as limitações do *reinforcement learning*.

No geral, este trabalho destacou a importância de considerar diferentes técnicas e abordagens no treino de agentes autônomos, visando alcançar um desempenho superior em ambientes mais complexos.

Referências

- [1] Unity-Technologies/ml-agents: The Unity Machine Learning Agents Toolkit (ML-Agents) is an open-source project that enables games and simulations to serve as environments for training intelligent agents using deep reinforcement learning and imitation learning. - [Em linha] [Consult. 27 mai. 2023]. Disponível em <https://github.com/Unity-Technologies/ml-agents>.
- [2] TECHNOLOGIES, Unity - Build More Engaging Games with ML Agents — Unity [Em linha] [Consult. 27 mai. 2023]. Disponível em <https://unity.com/products/machine-learning-agents>.
- [3] HANG, Alina - How to Design a Reinforcement Learning Reward Function for a Lunar Lander [Em linha], atual. 12 nov. 2021. [Consult. 2 jun. 2023]. Disponível em <https://towardsdatascience.com/how-to-design-reinforcement-learning-reward-function-for-a-lunar-lander-562a24c393f6>.
- [4] Unity ML-Agents Toolkit - [Em linha]. [S.l.] : Unity Technologies, 2023 [Consult. 2 jun. 2023]. Disponível em <https://github.com/Unity-Technologies/ml-agents>.
- [5] Machine Learning Glossary: Reinforcement Learning - [Em linha] [Consult. 2 jun. 2023]. Disponível em <https://developers.google.com/machine-learning/glossary/rl>
- [6] What is Reinforcement Learning? A Comprehensive Overview - [Em linha] [Consult. 27 jan. 2023]. Disponível em <https://www.techtarget.com/searchenterpriseai/definition/reinforcement-learning>.
- [7] Proximal Policy Optimization - Em Wikipedia [Em linha] [Consult. 3 jun. 2023]. Disponível em https://en.wikipedia.org/w/index.php?title=Proximal_Policy_Optimization&oldid=1126985391.

6 ANEXOS

<input checked="" type="checkbox"/>	T2C\RayAgent	●
<input type="checkbox"/>	T3C\RayAgent	●
<input type="checkbox"/>	T4C\RayAgent	●
<input type="checkbox"/>	T5C\RayAgent	●
<input checked="" type="checkbox"/>	T6C\RayAgent	●
<input checked="" type="checkbox"/>	T7C\RayAgent	●

Figura 19: Legenda dos gráficos do Tensorboard

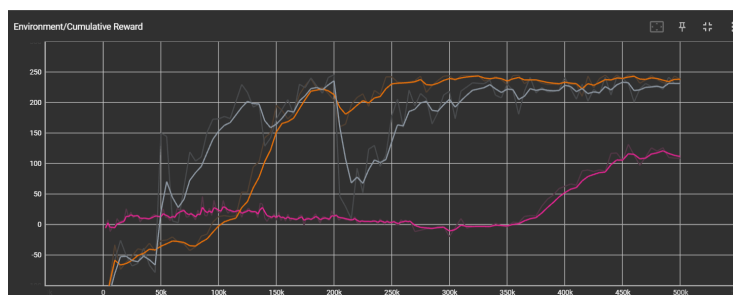


Figura 20: Environment/Cumulative Reward

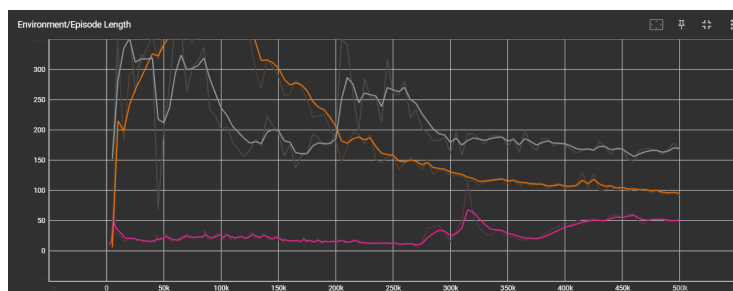


Figura 21: Environment/Episode Length

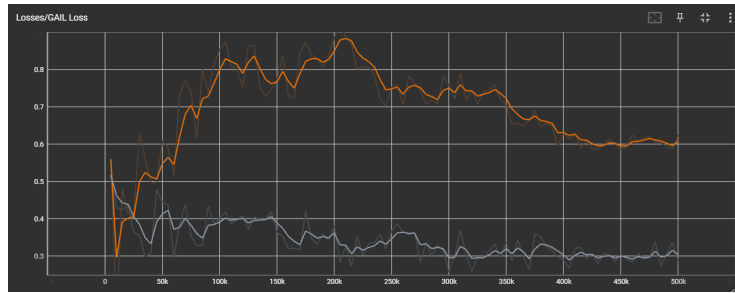


Figura 22: Losses/GAIL Loss

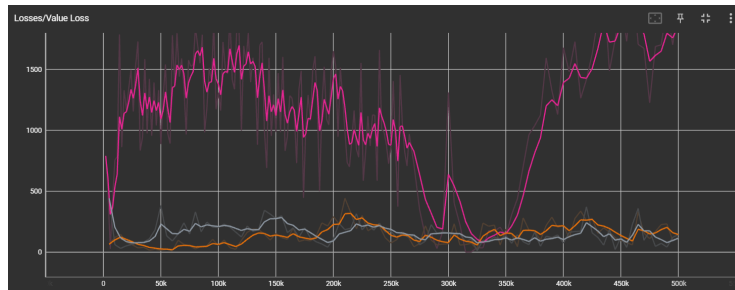


Figura 23: Losses/Value Loss

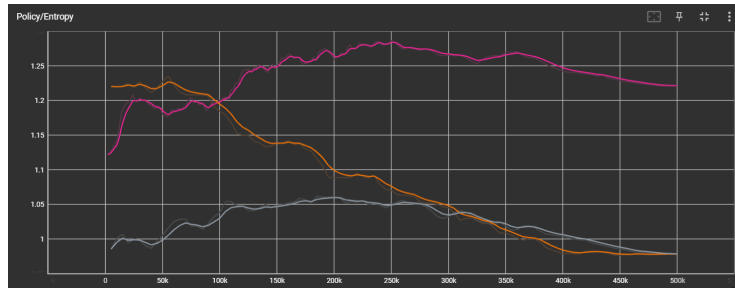


Figura 24: Policy/Entropy

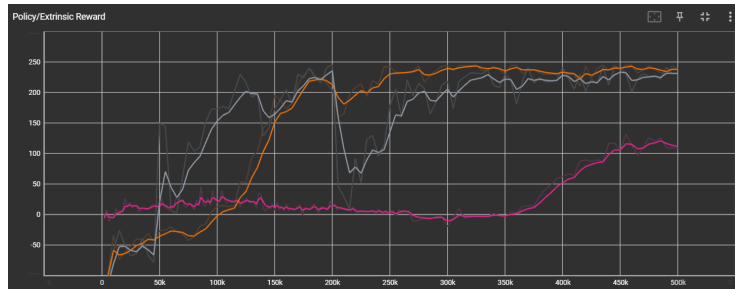


Figura 25: Policy/Extrinsic Reward

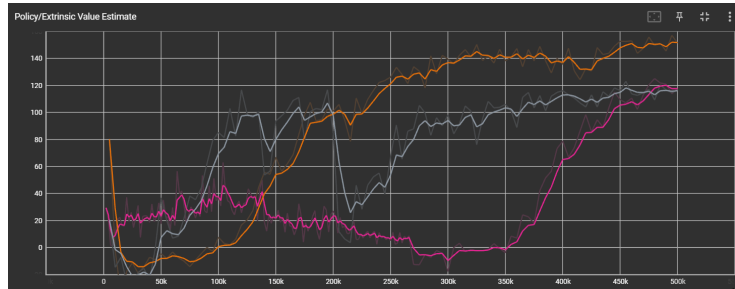


Figura 26: Policy/Extrinsic Value Estimate