ISEL
INSTITUTO SUPERIOR DE ENGENHARIA DE LISBOA

Aprendizagem por Reforço com ML-Agents

Ambiente Virtuais Interativos e Inteligentes

Trabalho realizador por:

Duarte Valente | A47657

João Valido | A51090

Docente:

Eng. Arnaldo Abrantes

Curso: MEIM

2023

Índice

1. Introdução 4

1.1. Descrição 4

1.2. Âmbito 4

1.3. Documentação 4

A. Implementação Geral 5

B. Implementação Detalhada 6

1. Criação do Ambiente de aprendizagem do Unity 6

2. Criação do Agente 7

2.1. Observações 7

2.2. Ações 8

2.3. Recompensas 10

3. Validação do setup de treino 11

4. Treino do Agente 12

4.1. Algoritmo 12

4.2. Parâmetros 12

4.3. Desempenho do agente 14

5. Avaliação do desempenho do Agente 16

C. Conclusão 17

Índice de ilustrações

Figura 1 - Ambiente de aprendizagem 6
Figura 2 - RayPerceptionSensor3D 7
Figura 3 - Observação - Distância entre obstáculo e Agente 8
Figura 4 - Behavior Parameters 9
Figura 5 - Ação de saltar 9
Figura 6 – Parede para acionar recompensa 10
Figura 7 - Método 'Heuristic()' 11
Figura 8 – Hiper Parâmetros usados 12
Figura 9 - Treino com 8 enviroments 14
Figura 10 - Treino com 16 enviroments 14
Figura 11 - Reward ao longo do tempo 15
Figura 12 - Highscore ao longo do tempo 15
Figura 13 - Highscore em 10 minutos 16

1. Introdução

1.1. Descrição

Este trabalho tinha como objetivo, usando os conhecimentos adquiridos nas aulas, construir um ambiente de aprendizagem no Unity e usar a toolbox ML-Agents para treinar um agente a realizar uma tarefa à nossa escolha, isto é, a aprender a ter um comportamento específico.

1.2. Âmbito

Este relatório insere-se no âmbito da realização da ficha laboratorial unidade curricular de Ambientes Virtuais Interativos e Inteligentes, do Mestrado em Engenharia Informática e Multimédia do DEETC do ISEL.

1.3. Documentação

Documentação de apoio à unidade curricular de Ambientes Virtuais Interativos e Inteligentes e o tutorial disponível em: https://towardsdatascience.com/ultimate-walkthrough-for-ml-agents-in-unity3d-5603f76f68b

A. Implementação Geral

Neste projeto, explorámos as capacidades do Unity, uma plataforma de desenvolvimento de jogos, e da ML-Agents, uma toolbox de aprendizado por reforço desenvolvida pela Unity Technologies, para criar um ambiente virtual onde um agente possa aprender a executar uma tarefa de maneira autônoma. A aprendizagem por reforço é uma abordagem de aprendizado de máquina que permite que um agente tome decisões em um ambiente complexo através da interação contínua com o mesmo, recebendo feedback em forma de recompensas ou penalidades.

Para iniciar o projeto, definimos que a tarefa que o agente deverá aprender a realizar seria “conduzir” num circuito desenhado, começando num ponto de partida, com o objetivo de chegar à meta. Com base na tarefa escolhida, projetámos um ambiente virtual detalhado no Unity, estabelecendo as regras e condições necessárias para a execução da tarefa.

Em seguida, utilizamos a ML-Agents para configurar o ambiente de aprendizagem, criando uma interface de comunicação entre o agente e o ambiente. A ML-Agents fornece uma estrutura poderosa para a criação e treino de agentes no Unity, oferecendo recursos como simulação física e renderização gráfica de alta qualidade.

A etapa de treino envolve a interação contínua do agente com o ambiente, onde ele toma ações com base em sua política de aprendizado e recebe feedback na forma de recompensas ou penalidades, neste caso específico, o agente será recompensado assim que cruzar a meta do percurso, e será penalizado cada vez que for contra as paredes do circuito. Ao longo do tempo, o agente ajusta sua política de ações para diminuir as penalizações recebidas e, assim, aprende a executar a tarefa de maneira mais eficiente e consistente.

Ao finalizar o treino, avaliamos o desempenho do agente em relação à tarefa definida. Analisamos métricas como a taxa de sucesso na execução da tarefa, o tempo necessário para concluir a tarefa e a qualidade do comportamento aprendido. Além disso, discutimos as limitações encontradas durante o processo de implementação e identificamos possíveis melhorias.

B. Implementação Detalhada

1. Criação do Ambiente de aprendizagem do Unity

O primeiro passo foi criar um ambiente virtual no Unity, onde o agente poderia interagir e aprender a partir das observações e ações disponíveis. O ambiente foi projetado levando em consideração os objetivos específicos do projeto, garantindo que o agente tivesse informações relevantes para tomar decisões.

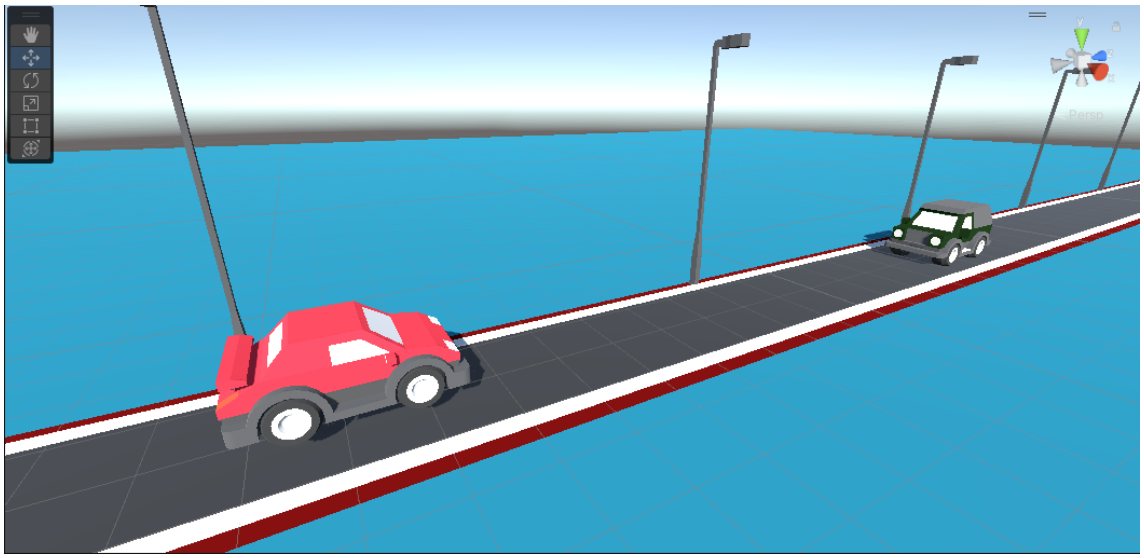

Figura 1 - Ambiente de aprendizagem

Como podemos observar pela Figura 1, o ambiente é constituído por um cenário, onde estão presentes alguns elementos decorativos como os candeeiros, a estrada e a relva, os mesmo vão avançando ao longo do tempo para criar a sensação de que os elementos principais estão em andamento. Como elementos principais do nosso ambiente temos o carro mais à esquerda, que é o nosso agente, e o carro mais à direita, que é o obstáculo que o agente tem de aprender a ultrapassar.

2. Criação do Agente

Após a criação do ambiente, um agente foi implementado para interagir com o mesmo. O agente foi definido usando a estrutura do ML-Agents, que permite especificar a observação do agente, as ações possíveis e a política de aprendizagem. O agente foi configurado para aprender a partir de recompensas recebidas com base no seu desempenho.

2.1. Observações

As observações são as informações que o agente recebe do ambiente. No nosso caso, como observação, o agente utiliza a distância até ao próximo objeto, também conhecido como obstáculo. Isto é possível através do componente *RayPerceptionSensor3D,* disponível através do pacote dos ML-Agents, demonstrado na Figura 2.

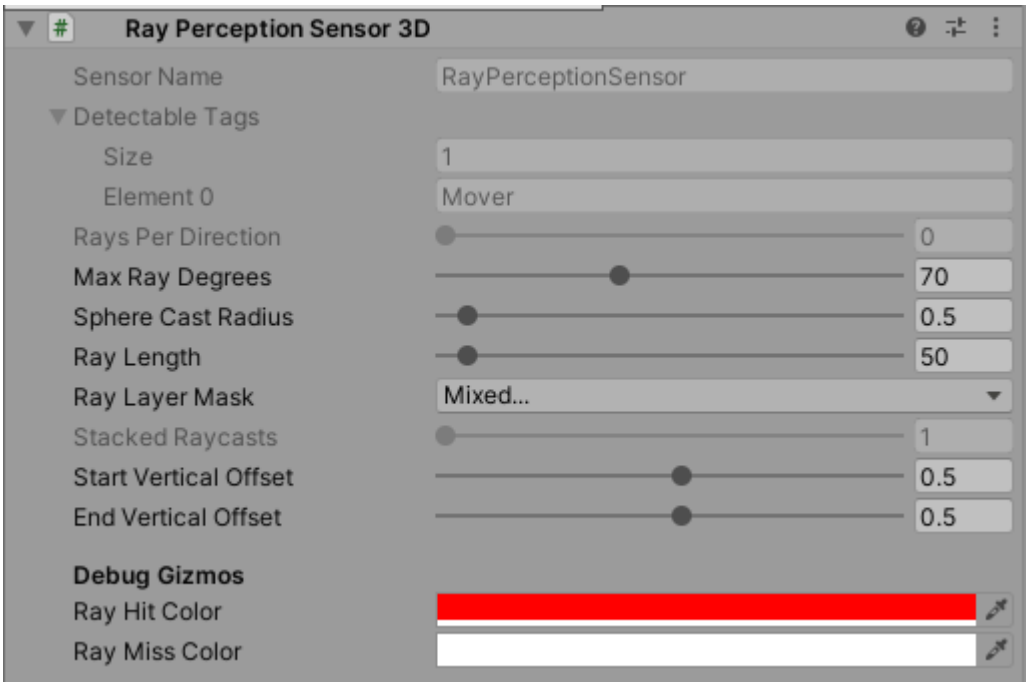

Figura 2 - RayPerceptionSensor3D

Primeiro, temos de adicionar as tags que serão detetadas. No nosso caso, os obstáculos gerados têm a tag ‘Mover’, então vamos adicioná-la. Em seguida, ‘Rays Per Direction’ descreve o número de raios à esquerda e à direita do centro. Como precisamos apenas de um único raio central, podemos definir esse valor como 0. O próximo valor relevante é o ‘Ray Length’. No nosso caso deve ir pelo menos até o final da estrada, por segurança vamos usar 50 aqui. Para garantir que o raio não atinge o solo, vamos dar-lhe um deslocamento vertical de 0,5 no início e no fim.

Adicionando então o componente ao nosso agente, podemos ver um raio vermelho a ser disparado na visualização da cena, como demonstrado na Figura 3.

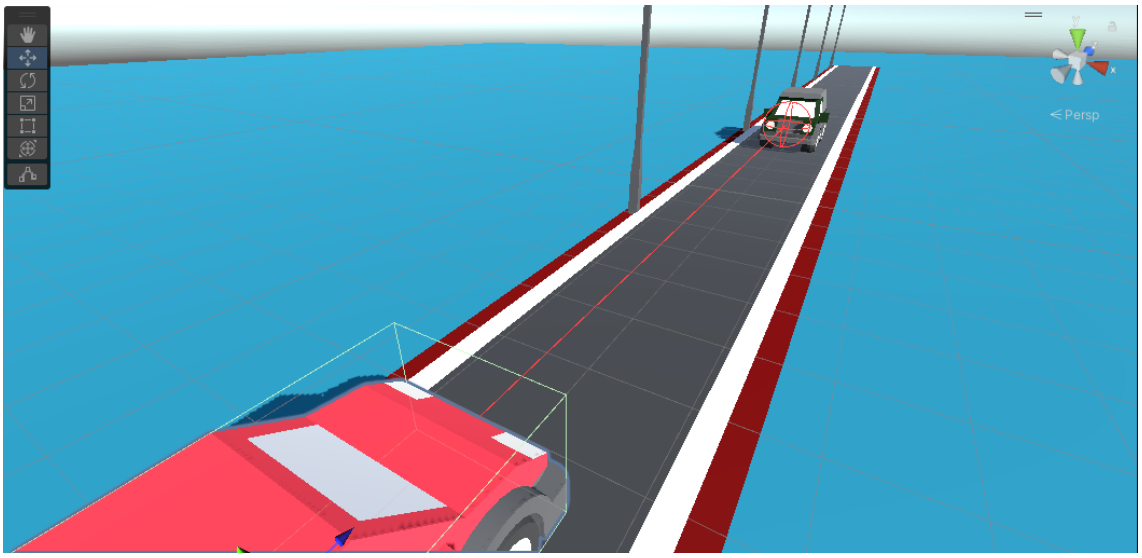

Figura 3 - Observação - Distância entre obstáculo e Agente

2.2. Ações

As ações foram definidas como as ações que o agente pode executar no ambiente. As ações foram projetadas de forma a permitir ao agente explorar e interagir efetivamente com o ambiente para maximizar sua recompensa. Neste projeto o agente tem duas ações, a ação de saltar e a ação de não fazer nada, tendo assim um ‘*Discrete Action Space’* de tamanho dois, opção essa que definimos num componente associado ao agente, chamada ‘*Behavior Parameters’,* demonstrado na Figura 4.

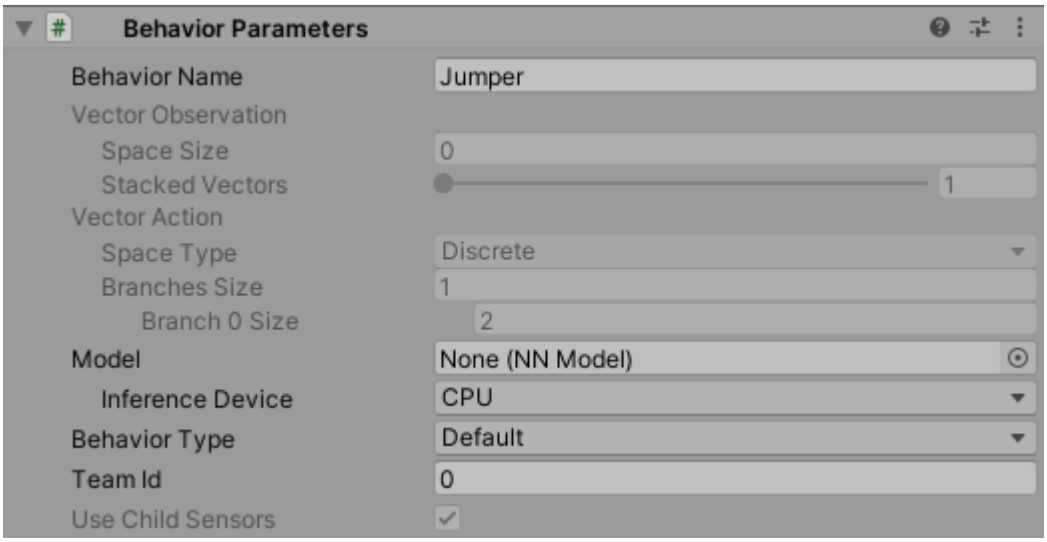

Figura 4 - Behavior Parameters

Tendo apenas duas ações, o valor das mesmas será 0 e 1, sendo 0 atribuído ao agente não fazer nada e 1 a saltar. Inicialmente damos a possibilidade de um utilizador desviar-se dos obstáculos, carregando na tecla 'espaço' sempre que desejar dar um salto, demonstrado na Figura 5.

Figura 5 - Ação de saltar

2.3. Recompensas

Neste momento estamos quase prontos para treinar o agente, mais ainda falta uma parte importante, definir as recompensas. As recompensas foram estabelecidas para fornecer feedback ao agente sobre o seu desempenho. Foram projetadas para incentivar o agente a realizar a tarefa específica desejada. As recompensas foram configuradas de forma a reforçar o comportamento desejado e desencorajar o comportamento indesejado.

Uma regra básica é uma punição de -1 ao perder e uma recompensa de +1 ao vencer. Como este é um jogo de 'Highscore', não há vitória, então decidimos adicionar 0,1 de recompensa por cada obstáculo ultrapassado. A pontuação neste jogo funciona da seguinte maneira: Cada obstáculo tem uma parede escondida atrás de si que aciona o método 'OnTriggerEnter()' em caso de colisão, demonstrado na Figura 6. De seguida, no método 'OnCollisionEnter()', encontramos o caso em que o agente é atingido pelo obstáculo, caso o mesmo aconteça, o agente volta à posição inicial. De maneira que o agente perceba se está a desempenhar um bom ou mau desempenho, por cada obstáculo que ultrapassa é recompensado positivamente, e negativamente cada vez que colidir com o mesmo.

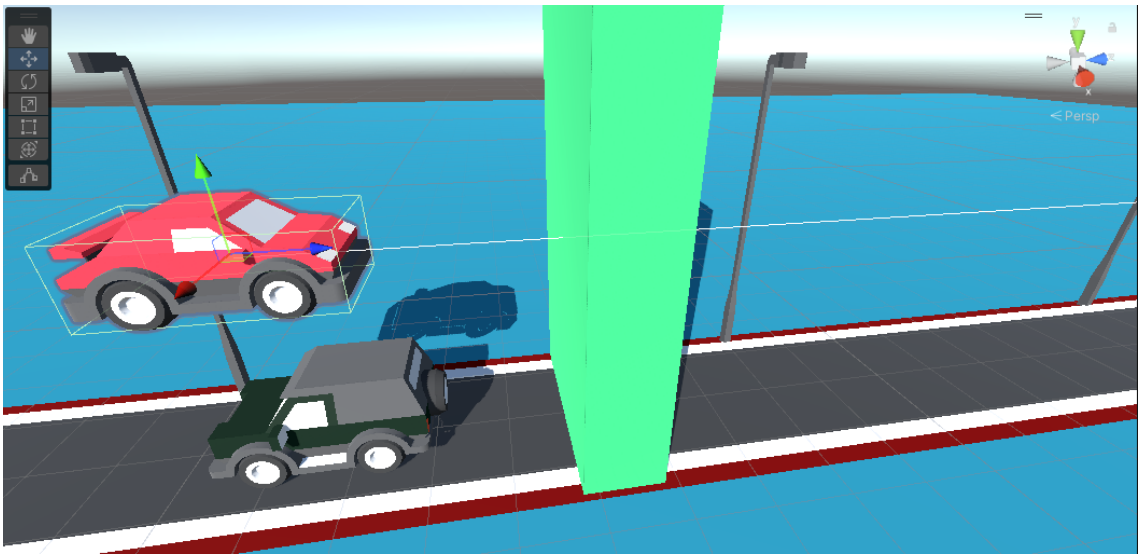

Figura 6 – Parede para acionar recompensa

3. Validação do setup de treino

Para validar o setup de treinamento do agente foi utilizado o modo heurístico, permitindo-nos assim controlar o agente manualmente para demonstrar a tarefa desejada. Isso permitiu verificar se o ambiente e as configurações do agente estavam corretos e se o agente era capaz de aprender com base no comportamento do especialista. Sendo isto tudo possível devido ao método 'Heuristic()', demonstrado na Figura 7.

Após alguns testes chegámos à conclusão de que o ambiente está preparado para que o agente começasse a treinar no mesmo.

```
public override void Heuristic(float[] actionsOut)
    {
        actionsOut[0] = 0;
        //Player Input

        if (Input.GetKey(jumpKey))
            actionsOut[0] = 1;
    }
```

Figura 7 - Método 'Heuristic()'

4. Treino do Agente

4.1. Algoritmo

O algoritmo PPO (Proximal Policy Optimization) foi o algoritmo de aprendizagem de reforço utilizado, pois é baseado em política, isto é, o algoritmo atualiza a política do agente para melhorar o desempenho do mesmo. É amplamente utilizado para treinar agentes em ambientes complexos, como jogos e simulações.

O PPO tem como objetivo otimizar a política do agente, que é uma função que mapeia os estados para as ações a serem tomadas. A política é representada por uma rede neural, onde os estados são usados como entrada e as ações são geradas como saída. O objetivo do algoritmo é atualizar a política para maximizar a recompensa esperada ao longo do tempo. Uma característica distintiva do PPO é o uso de uma técnica chamada Clipped Surrogate Objective. Esta técnica é usada para controlar o tamanho das atualizações da política, evitando mudanças muito grandes que possam prejudicar o desempenho do agente. Limitando as atualizações para ficarem dentro de uma "região de confiança" em relação à política anterior.

4.2. Parâmetros

No algoritmo Proximal Policy Optimization (PPO), os hiper parâmetros desempenham um papel crucial no desempenho e na eficiência do treino do agente. Os hiper parâmetros usados estão representados na Figura 8.

```
hyperparameters:
 batch_size: 128
 beta: 0.005
 buffer_size: 2048
 epsilon: 0.2
 lambd: 0.95
 learning_rate: 0.0003
 learning_rate_schedule: linear
 num_epoch: 3
```

Figura 8 – Hiper Parâmetros usados

- batch_size: É o número de amostras de experiência coletadas antes de realizar uma atualização da política. No caso mencionado, é definido como 128, o que significa que a cada iteração do treino, o algoritmo coleta 128 amostras de experiência para atualizar a política.

- beta: Controla a taxa de penalização aplicada na função de perda do algoritmo PPO. Uma penalização é aplicada para limitar a magnitude das atualizações da política, garantindo que as mudanças não sejam muito grandes. Um valor de beta menor, como 0.005, resulta em uma penalização mais branda, permitindo atualizações maiores da política.

- buffer_size: O tamanho do buffer refere-se ao número máximo de amostras de experiência que podem ser armazenadas em um buffer de replay durante o treino. É definido como 2048, o que significa que o algoritmo mantém um buffer de replay com até 2048 amostras de experiência.

- epsilon: O epsilon define a "região de confiança", isto é, controla o tamanho máximo permitido para a diferença relativa entre a política atualizada e a política anterior. Um valor de epsilon menor, como 0.2, indica uma região de confiança mais restrita, limitando as atualizações da política a mudanças relativamente pequenas.

- lambd: É o fator de desconto utilizado para calcular a função de vantagem. A função de vantagem estima a vantagem de uma determinada ação em relação a outras ações possíveis. Um valor de lambd de 0.95 indica um fator de desconto de longo prazo, considerando recompensas futuras com mais peso.

- learning_rate: Determina o tamanho das atualizações da política durante o treino. Um valor de 0.0003 indica uma taxa de aprendizagem relativamente baixa, o que significa que as atualizações da política serão mais sutis.

- learning_rate_schedule: Define como a taxa de aprendizagem é ajustada ao longo do tempo durante o treino. No caso mencionado, é definida como linear, o que implica que a taxa de aprendizagem diminui linearmente ao longo do tempo.

- num_epoch: Indica quantas vezes os dados de experiência são usados para atualizar a política em cada iteração do treino. O algoritmo realiza três épocas de atualização da política com os dados coletados antes de avançar para a próxima tentativa.

4.3. Desempenho do agente

Tendo o algoritmo escolhido e os respetivos hiper parâmetros configurados, podemos finalmente começar a treinar o agente. Dividimos o treino em duas fases. Na primeira fase utilizámos 8 agentes nos respetivos ambientes, como demonstrado na Figura 9, com o propósito de realizar uma segunda fase onde treinámos 16 agentes nos seus ambientes, como demonstrado na Figura 10, isto com o propósito de acelerar e estabilizar o treino. No nosso caso, isso é tão fácil quanto copiar o ambiente várias vezes. Isto também para conseguirmos comparar as curvas de aprendizagem dos agentes nas duas situações.

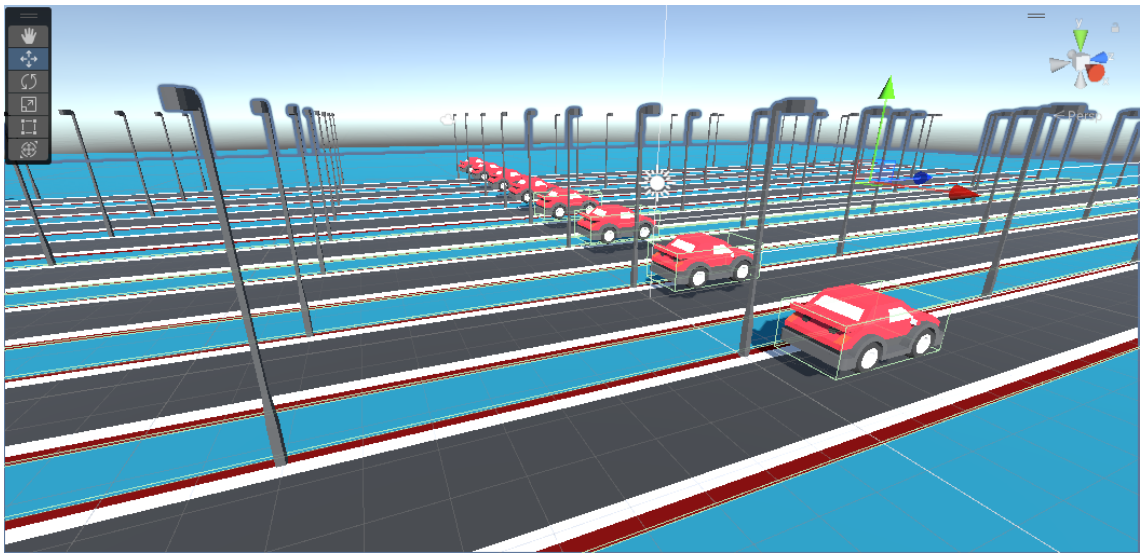

Figura 9 - Treino com 8 enviroments

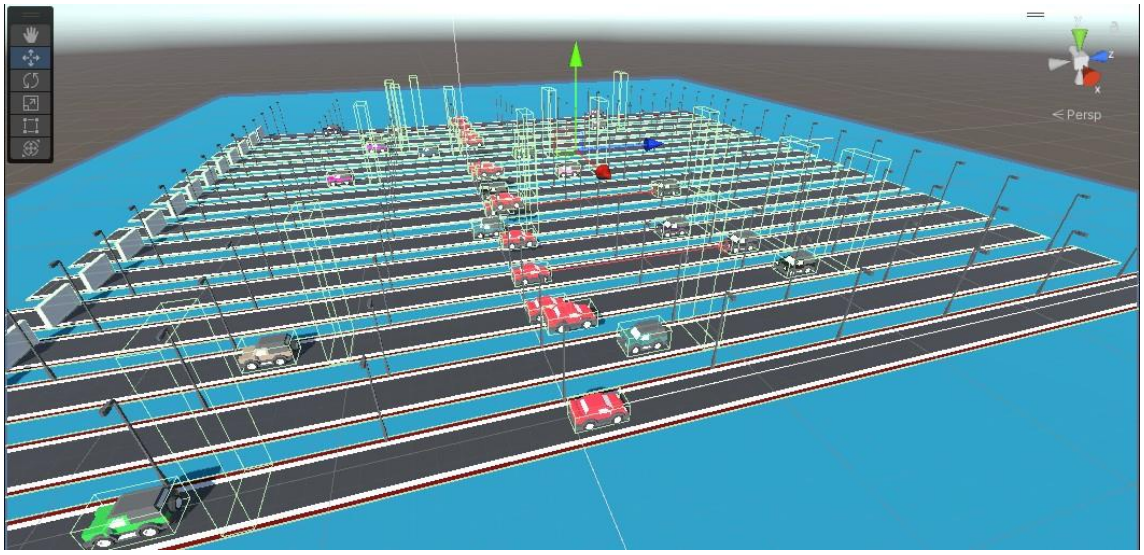

Figura 10 - Treino com 16 enviroments

Durante ambos os treinos, com a duração de aproximadamente 10 minutos cada, o desempenho dos agentes foi monitorizado e registado. Foram utilizados gráficos do TensorBoard para visualizar e analisar a evolução do desempenho ao longo do tempo.

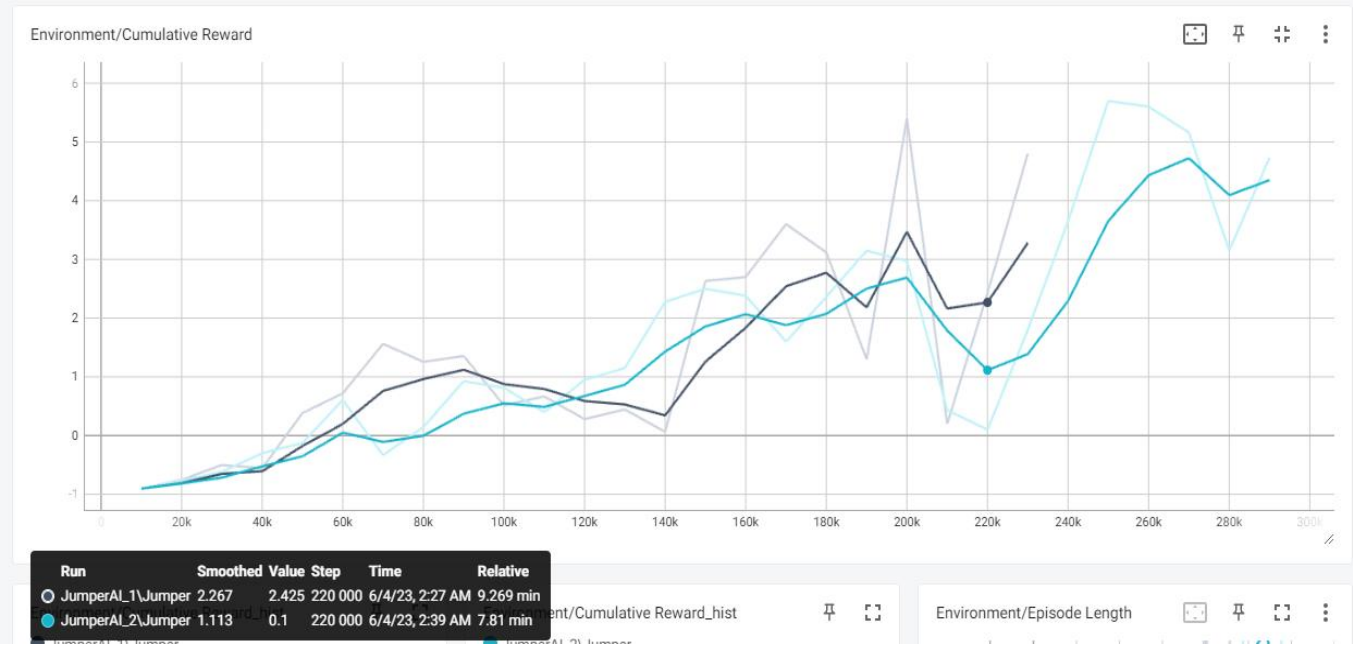

Figura 11 - Reward ao longo do tempo

Na Figura 11 podemos comparar a média do valor das recompensas, primeiramente podemos verificar que ambos os treinos precisaram aproximadamente de 60k a 80k steps para ficarem com uma média positiva de recompensas, isto é, só a partir dessa quantidade de steps é que o agente começou a ultrapassar mais vezes o obstáculo do que a chocar com o mesmo.

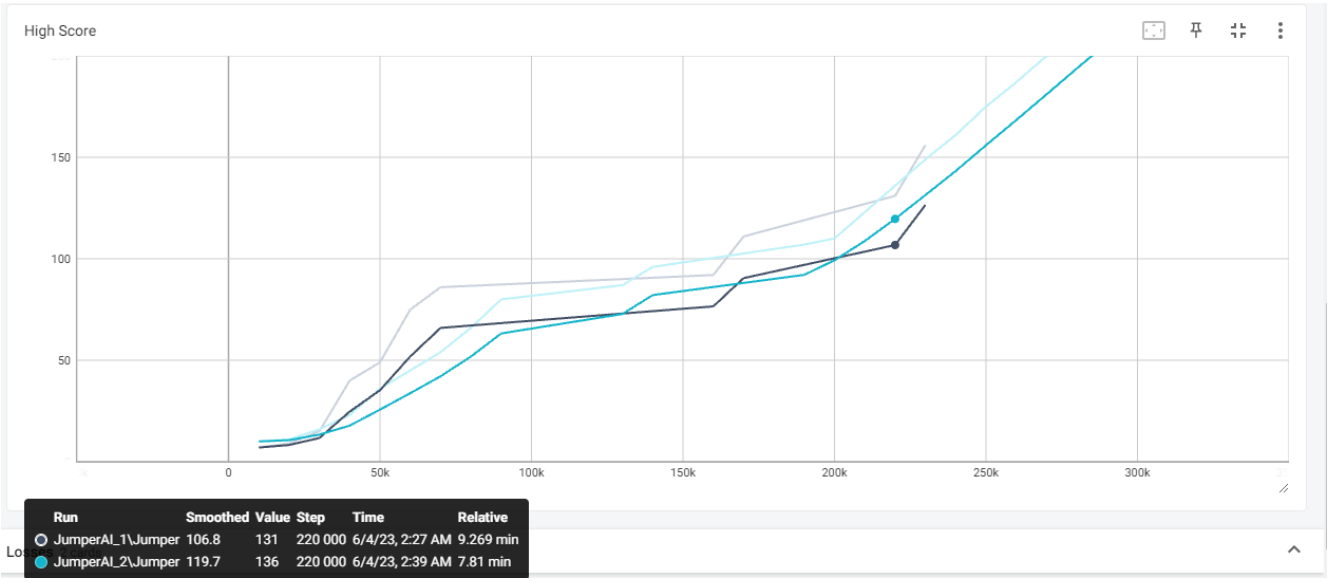

Figura 12 - Highscore ao longo do tempo

Também é possível verificar o mesmo na Figura 12, desta vez estamos a observar a evolução do highscore ao longo do tempo, isto é, ao fim de X steps, o agente foi capaz de chegar a um total de Y obstáculos ultrapassados.

Analisando ambos os gráficos, podemos concluir que apesar de ambos os treinos terem uma duração de 10 minutos, os gráficos do segundo treino são um pouco maiores, e isso deve-se ao facto de processarmos muito mais etapas no mesmo período devido ao número de agentes que temos. Podemos ainda concluir que em termos de steps, o facto de termos mais ou menos agentes não influencia muito o desempenho dos mesmos, pois com os mesmos steps ambos os treinos apresentaram valores muito semelhantes.

5. Avaliação do desempenho do Agente

Após o treino, o desempenho do agente foi avaliado fazendo inferência usando a rede neural resultante do treino. O agente foi testado no ambiente de aprendizagem para verificar a sua capacidade de realizar a tarefa com sucesso. Como podemos verificar na Figura 13, o agente em certa de 10 minutos conseguiu chegar a um Highscore de 205. Se deixássemos mais tempo os resultados também seriam melhores, como pudemos verificar no capítulo anterior, o que indica que o mesmo aprendeu com sucesso a tarefa que tínhamos como objetivo deste projeto ensinar-lhe, a desviar-se dos obstáculos que vão aparecendo de x em x tempo.

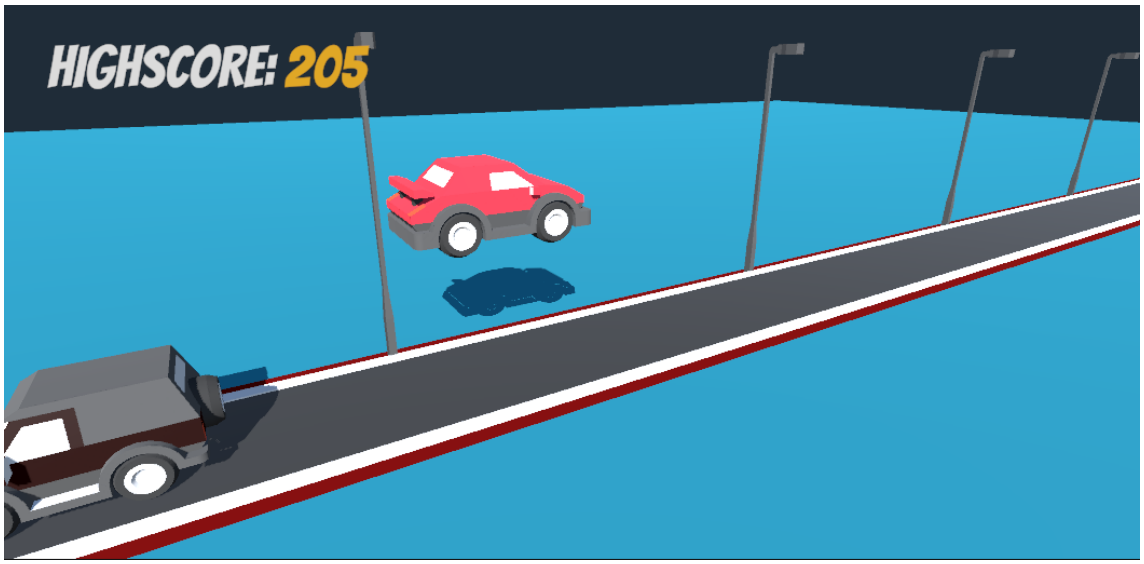

Figura 13 - Highscore em 10 minutos

C. Conclusão

A criação de um ambiente de aprendizagem no Unity usando o ML-Agents, combinada com o treino de agentes utilizando algoritmos de aprendizagem por reforço, oferece uma abordagem poderosa para desenvolver agentes capazes de realizar tarefas específicas em ambientes virtuais. A avaliação do desempenho durante o treino e após o mesmo permite ajustar e melhorar o agente, garantindo que ele alcança um desempenho ótimo para a tarefa em questão.

A nossa ideia inicial era ensinar os ml-agents a percorrerem um percurso em formato de pista de corrida. Contudo, tendo em conta a nossa experiência anterior em unity, que era praticamente nula, conseguimos realizar este jogo de aprendizagem de um ml-agent, um pouco mais simples e ficar a perceber um pouco mais sobre este tema.