Capítulo 7

Aprendizagem Profunda em Unity com ML-Agents

Vitor Azevedo Silva, Brenno Yves Damasceno Morais,
Suzana Matos França de Oliveira e Danilo Borges da Silva.

Abstract

Deep Learning (DL) approach is applied in different research areas, mainly in games. However, to use algorithms present in these techniques can be relatively complex. These chapter goals on practicality, made possible by the creation of virtual environments on Unity, and the training of agents with ML-Agents. Two learning environments are showed to demonstrate how simple it is to train an agent, with a graphical interface, and use advanced DL approaches such as Proximal Policy Optimization. Lastly, is shown how to produce training charts and interpret it, and tips for advancing studies on the topic.

Resumo

As técnicas de Aprendizagem Profunda (AP) estão sendo utilizadas em diferentes áreas de pesquisa, principalmente em jogos. Porém, o uso dos algoritmos presentes nessas técnicas pode ser relativamente complexo. Este capítulo é focado na praticidade possibilitado pela criação de ambientes virtuais na Unity e pelo treinamento de agentes com o ML-Agents. Dois ambientes de aprendizagem são exibidos para demonstrar como é simples realizar o treinamento de gum agente com interface gráfica e uso de técnicas avançadas de AP como o Proximal Policy Optimization. Por fim, é mostrado como produzir gráficos de treinamento e interpretá-los, e dicas para avançar nos estudos sobre a temática.

7.1. Introdução

A extensão dos campos de Aprendizagem Profunda permite o florescimento de inúmeras ideias e motivações para novas pesquisas. Com o pacote ML-Agents da Unity, as possibilidades são facilitadas. Explorar e desafiar os limites dessa área, realizar testes realísticos e desenvolver agentes que podem se tornar prelúdio para algo maior, são apenas algumas das possibilidades. Este trabalho tem como objetivo explorar esse poderoso pacote à medida que apresenta a Unity, plataforma de modelagem ideal para realizar com fidelidade grandes ideias, incluindo simulações e jogos.

Visto que o pacote ML-Agents possui inerentemente intimidade com áreas da Inteligencia Artificial é recomendada uma noção básica dos conceitos que o compõem, tais como redes neurais e aprendizado de máquina, de modo que o usuário possa compreender com totalidade suas funções e fundamentos.

As redes neurais podem ser entendidas como modelos computacionais inspirados na abstração de um neurônio real, cuja finalidade é simular o processo de aprendizado e resolução de problemas complexos. Com essas redes é possível reconhecer padrões, relacionar, agrupar e classificar dados e informações, com o objetivo de aprimorar continuamente sua busca por melhores resultados. Sua arquitetura é baseada no sistema neurológico humano, onde possui "receptores" dispostos em uma ou várias camadas. Para cada receptor é atribuído um peso que é recalculado a medida que a rede aprende [Poole and Mackworth 2010].

O aprendizado de máquina *(Machine Learning)* é considerado uma ramificação da inteligência artificial, sendo assim, um subcampo da ciência da computação. Tem como principal finalidade o desenvolvimento de técnicas computacionais sobre armazenamento de "experiência" e sistemas com a habilidade de adquirir conhecimento sem intermédio do desenvolvedor. Destacam-se os seguites tipos de treinamento [Lanham 2018]:

- ***Unsupervised Training*** (Treinamento Não Supervisionado): é um método que examina os dados de forma autônoma e desenvolve uma classificação. A classificação é baseada em certas métricas que podem ser descobertas pelo próprio treinamento.

- ***Supervised Training*** (Treinamento Supervisionado): é um método comum em que a maioria das abordagens de *Machine Learning* utilizam para predizer ou classificar. Requer dados de entrada e dados de saída rotulados, necessitando um conjunto de dados de treinamento para construir um modelo.

- ***Imitation Learning*** (Aprendizagem por Imitação): é um método em que o treino é executado observando ações designadas e imitando-as.

- ***Curriculum Learning*** (Aprendizado por Currículo): é um método avançado de aprendizado que funciona subdividindo o problema em níveis de complexidade.

- ***Reinforcement Learning*** (Aprendizagem por Reforço): é um método baseado na Teoria do Controle [Doyle et al. 2013], permite que a rede treine sem dados iniciais ou modelo pré-determinados. O treinamento é modelado com base em recompensar ações que aumentam a probabilidade de sucesso no objetivo determinado.

Esse trabalho destaca a técnica *Reinforcement Learning* (RL) que é frequentemente utilizada para treinamento de agentes, os quais tem a habilidade de perceber o estado do ambiente e emitir uma resposta conforme os estímulos. Os agentes tornam-se inteligentes por meio da modelagem dos dados extraídos dos sensores, recompensando os acertos e punindo os erros, aumentando sua acurácia na tarefa de atingir o objetivo determinado. Este processo pode ser observado na Figura 7.1. Aqui, o ambiente será produzido com o uso da Unity.

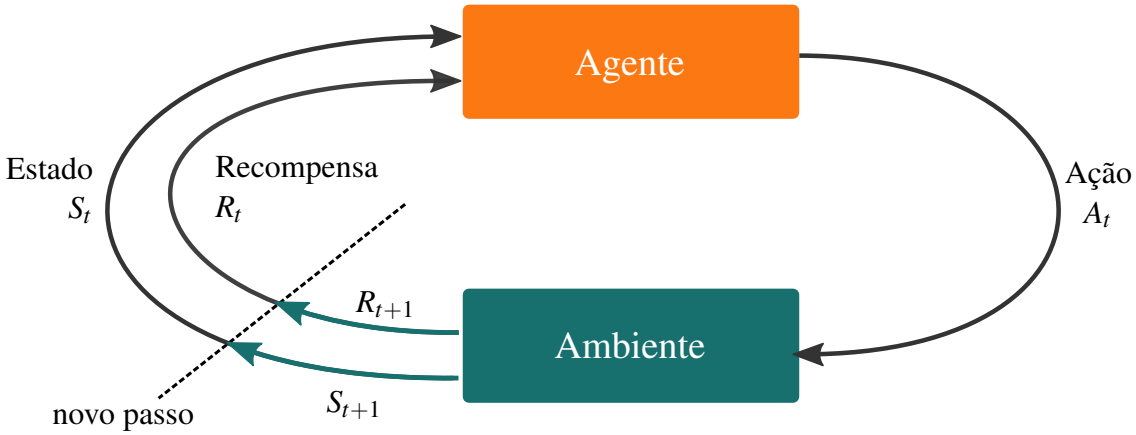


Figura 7.1. Ciclo de Aprendizagem por Reforço. O agente interage com o ambiente e recebe uma recompensa (R_t), positiva ou negativa, norteando seus passos e definindo seus objetivos de acordo com seu estado (S_t) e suas ações (A_t) em cada passo do treinamento [Sutton and Barto 2018].

A **Unity** é um Motor de Jogo (*Game Engine*) em um ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) com a finalidade de facilitar a construção dos jogos.

Um motor de jogo é um programa ou um conjunto de bibliotecas que têm a capacidade de agrupar e produzir com totalidade os elementos de um jogo em tempo real, ou seja, reúne recursos de renderização, animação, sons, dinâmicas, física, etc, de modo que o desenvolvimento dos jogos seja simples para pessoas de diversas áreas como programadores ou artistas.

A Unity possui suporte para o desenvolvimento de jogos em duas ou três dimensões, simulações realísticas, podendo incluir funcionalidades como uso de Realidade Aumentada, Realidade Virtual e Inteligência Artificial É uma ferramenta bastante eficiente para criar ambientes e com o uso do ML-Agents os agentes podem ser treinados na Unity [Mills and Stufflebeam 2005].

As demais seções deste capítulo estão divididas em mostrar as componentes necessárias para criação do ambiente e treinamento do agente, Seções 7.2 e 7.3. Na realização da configuração dos ambientes de aprendizagem, Seção 7.4, e na observação de gráficos do treinamento, Seção 7.5. Por fim, a Seção 7.6 destina-se a apresentação de dicas de estudos em aprendizagem profunda com Unity e ML-Agents.

7.2. Unity

Nesta seção é apresentado como instalar o Unity e como identificar alguns elementos observados em sua IDE com sua devida finalidade. Para melhor identificar elementos de interface e os arquivos utilizados optou-se por sublinhar esses elementos.

Para utilizar a Unity acesse o site oficial[1], faça o download e instale o UnityHub. Abra esse programa e vá até a seção Installs (instalações), clique em Add (adicionar) e

[1] https://store.unity.com/download?ref=personal

em seguida escolha a versão compatível mais nova da Unity. Neste trabalho utilizou-se a versão 2019.4.2f1.

Para criar um projeto, abra o UnityHub, acesse a aba Projects, e selecione NEW no canto superior direito. Com isso, uma nova janela se abrirá, ao lado esquerdo localize 3D para criar um projeto com suporte 3D. Escolha um nome de sua preferência e selecione Create. A plataforma de edição é subdividida em janelas (Figure 7.2), cada qual com suas funções práticas, desde modificar o comportamento, física, lógica e aparência do cenário, até organizar os objetos do jogo, a posição de cada um, etc.

Hierarchy(hierarquia): é uma janela que lista todos os objetos da cena atual. Na Unity todo objeto é um GameObject (Figura 7.2a).

Scene (cena): é uma janela que apresenta uma perspectiva interativa do cenário em desenvolvimento. É possível utilizar esse painel para selecionar, reposicionar ou redimensionar objetos como: cenários, personagens, câmera e iluminação (Figura 7.2b).

Game (jogo): é onde aparece o resultado final do jogo, pelo ponto de vista das câmeras definidas na cena. É possível selecionar a escala do jogo, o tamanho, resolução e outros aspectos em relação à imagem (Figura 7.2c). Há também opções de executar e pausar a execução do jogo para observar cada *frame* em diferentes resoluções.

Project (projeto): é a janela que organiza e apresenta todos os arquivos relacionados ao projeto. Do lado esquerdo, Figura 7.2d, é mostrado um painel organizado de forma hierárquica em que é possível navegar pelas pastas que estruturam o projeto. Ao selecionar uma pasta a Unity mostra todos os seus arquivos.

Inspector (inspetor): contém informações detalhadas acerca dos GameObject da cena. É possível alterar a propriedade e inserir componentes ao GameObject, como a característica de ser um objeto rígido ou scripts escritos em linguagem C# (Figura 7.2e).

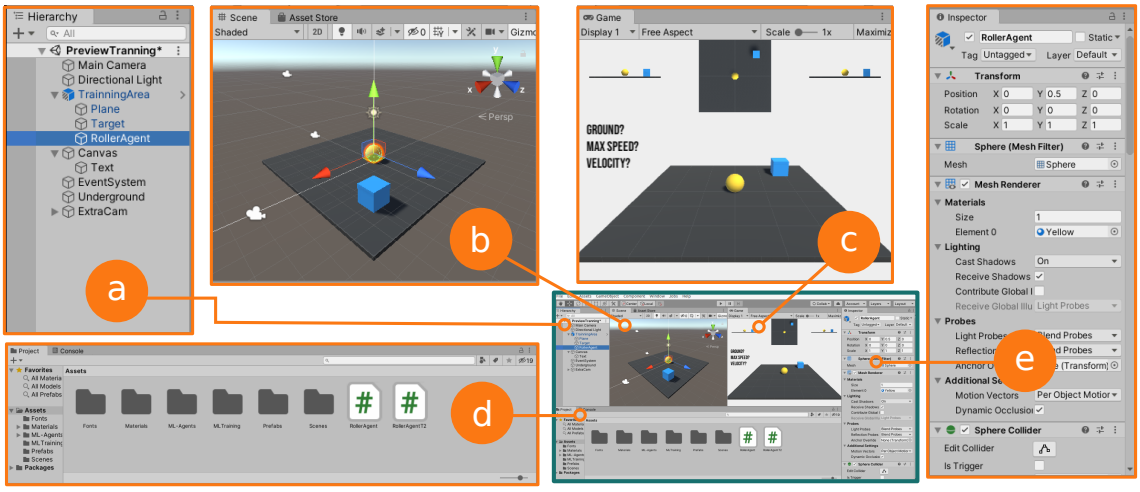


Figura 7.2. Elementos da interface da Unity. (a) *Hierarchy*, onde contém todos os objetos do cenário. (b) *Scene*, onde apresenta uma perspectiva do ambiente. (c) *Game*, onde é apresentado o resultado final do jogo. (d) *Project*, onde apresenta os arquivos relacionados ao projeto. (e) *Inspector*, onde contém informações acerca de um objeto selecionado.

7.3. ML-Agents

O **Unity Machine Learning Agents Toolkit** (ML-Agents) é um projeto de código aberto que permite a utilização de ambientes virtuais e simulações para o treinamento de agentes inteligentes [Juliani et al. 2018]. Os agentes podem ser treinados com o uso de técnicas de *Machine Learning* (ML) como *Reinforment Learning*, *Imitation Learning*, *Curriculum Learning* e outros métodos disponibilizados em bibliotecas matemáticas.

A comunicação entre esses ambientes é feita por meio de uma *Application Programming Interface* (API) – interface de programação de aplicativos – que consta de um conjunto de rotinas e padrões estabelecidos por um software para a utilização das suas funcionalidades por outros aplicativos [Pereira 2019]. O ML-Agents disponibiliza uma API desenvolvida na linguagem Python de fácil usabilidade, fornecendo assim códigos implementados de do estados-da-arte compostos no TensorFlow[Community 2020a].

O TensorFlow é um *framework open-source* (código aberto) com suporte para Python, Javascript, C/C++, Go e Ruby desenvolvido pelo Google Brain, subgrupo do Google voltado para pesquisas na área de Inteligência Artificial com o objetivo de auxiliar no aprendizado de máquina. Por ser um dos pacotes pré-requisitados pelo ML-Agents, o TensorFlow é instalado automaticamente em versão compatível com a versão instalada do ML-Agents.

A união da Unity, ML-Agents e o Tensorflow permite que desenvolvedores projetem, treinem e visualizem suas aplicações em duas ou três dimensões. Os agentes treinados podem ser utilizados para inúmeros propósitos, incluindo o controle de comportamento de NPCs (personagens não jogáveis), testes automáticos na versão beta em jogos recém construídos, desenvolvimento de protótipos de robôs que atuaram no mundo real, além de avaliar diferentes designs antes do lançamento [Community 2020a]. Os componentes principais do ML-Agents podem ser estruturados seguindo o diagrama na Figura 7.3 e são caracterizados por:

- **Ambiente de Aprendizado**: contém o Scene da Unity e todos os objetos. Este será o ambiente na qual os agentes irão observar, agir e aprender.
- **API Python de baixo nível**: contém uma interface Python simples que permite interagir e manipular um ambiente de aprendizado.
- **Comunicador externo**: conecta o ambiente de aprendizado com a API Python de baixo nível.
- **Treinadores Python**: contém os algoritmos de *Machine Learning* responsáveis por treinar os agentes. Os algoritmos se encontram no pacote Python `mlagents`, a sua execução se deve ao comando `ml-agents-learn` que permite como parâmetros selecionar o método de treinamento e as opções em um documento formatado.

O Agente do ML-Agents consiste de um conjunto de componentes anexado a um GameObject, denominado Agente, que é responsável pela manipulação e gerenciamento de observações que, ao realizar a ação, recebe uma recompensa positiva ou negativa e

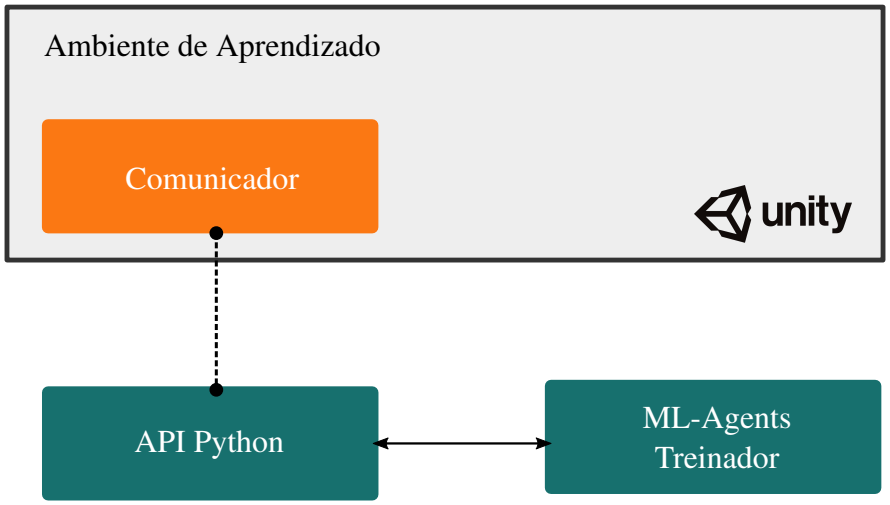


Figura 7.3. Diagrama de blocos simplificado do ML-Agents.

sempre será conectado a um comportamento (*behavior*). O comportamento define atributos específicos do Agente, como por exemplo o número de ações que ele irá executar. O Agente é identificado unicamente por um campo chamado *Behavior name*. Um comportamento também pode ser compreendido como uma função que recebe observações e recompensas do agente e retorna ações, podendo ser classificada em três tipos distintos [Community 2020a]:

- **Comportamento de aprendizagem**: é aquele que ainda não está definido, mas está prestes a ser treinado.
- **Comportamento heurístico**: é aquele que é definido por um conjunto de regras codificadas implementadas no código.
- **Comportamento de inferência**: é aquele que inclui um arquivo de rede neural treinado, o qual criado em decorrência do treinamento do comportamento de aprendizagem, tornando-se um comportamento de inferência.

O treinamento realizado em cada comportamento é associado a um conjunto de vários episódios. É denominado episódio (no contexto de RL) uma sequência de estados do ambiente, ações e recompensas, que terminam em um estado final. Por exemplo, do início de um jogo da velha até o momento em que o jogador perde, empata ou vence. O comportamento é responsável pela a definição de três importantes fases de treino, que ocorrem ciclicamente em cada episódio simulado (Figura 7.1):

- **Observações** (*observations*): é aquilo que o agente percebe sobre o ambiente. As observações podem ser numéricas e/ou visuais. As numéricas medem os atributos do ambiente do ponto de vista do agente. Para os ambientes mais interessantes, um agente exigirá várias observações numéricas contínuas. As observações visuais, por outro lado, são imagens geradas a partir das câmeras conectadas ao agente e representam o seu campo de visão.

- **Ações** (*actions*): são as ações que o agente pode executar. Semelhante às observações, as ações podem ser contínuas ou discretas, dependendo da complexidade do ambiente e do agente. Se for um ambiente de grade simples, em que apenas a localização do agente é importante (norte, sul, leste, oeste), é desejável utilizar observações discretas. Se o ambiente for mais complexo e o agente puder se mover livremente, será mais apropriado usar ações contínuas para descrever o seu posicionamento de acordo com a dimensão do ambiente.

- **Recompensa** (*reward*): é um valor escalar que sintetiza o desempenho do agente. O sinal de recompensa não precisa ser fornecido a todo momento, mas apenas quando o agente realiza uma ação boa ou ruim.

7.3.1. Uso do ML-Agents e suas possibilidades

O uso do ML-Agents promove possibilidades interessantes para diferentes áreas. A utilização do ambiente realístico da Unity é propício inclusive para testes de agentes que irão atuar no mundo real, como carros autônomos e estabilidade de drones. No ambiente de desenvolvimento de jogos, um fator importante é a jogabilidade que, com o uso de *Machine Learning*, pode oferecer uma experiência personalizada para o jogador. Alguns exemplos de ambientes de treino são:

- **Agente único**: apenas um agente, que recebe a própria recompensa.

- **Agente único simultâneo**: múltiplos agentes independentes que recebem recompensas independentes mas com os mesmos *Behavior Parameters* (parâmetros do comportamento).

- **Agente jogando contra si**: em um jogo que permite dois jogadores, dois agentes com as mesmas recompensas e mesmos *Behavior Parameters* jogam contra si.

- **Multiagentes cooperativo**: múltiplos agentes interagindo com ou sem o mesmo *Behavior Parameter* mas possuindo o mesmo sistema de recompensa.

- **Multiagentes competitivos**: múltiplos agentes interagindo com ou sem o mesmo *Behavior Parameter* e possuindo o sistema de recompensa inverso.

Nos ambientes produzidos neste trabalho utilizou-se como ambiente de treino o primeiro caso: **Agente único**.

7.3.2. Instalação do ML-Agents

O kit de ferramentas do ML-Agents armazena vários componentes, entre eles o devkit da Unity C#, que será integrado nas cenas do jogo, assim como três pacotes Python: `mla gents`, que contém algoritmos de ML, permitindo treinar o Agente no ambiente criado facilitando a comunicação entre o cenário da Unity e esses algoritmos; `mlagents_envs`, que contém a API python que irá interagir com o cenário da Unity; e `gym_unity`, que oferece uma função de encapsulamento para a cena suportando a interface OpenIA Gym. A OpenIA Gym pode ser definida como um *toolkit* que testa e compara diferentes métodos

de aprendizagem por reforço em um ambiente de simulação, selecionando o melhor, com o objetivo de maximizar os ganhos da interação com o ambiente.

O kit do ML-Agents contém uma pasta chamada Projects que possui vários exemplos de ambientes com a finalidade de ajudar o usuário a iniciar o seu aprendizado, demonstrando algumas possibilidades, além da quantidade de recursos do kit e das ferramentas da Unity, a versão do kit utilizada neste minicurso será a `release_3`, porém existem novas versões. Para baixar o kit basta clonar o repositório do projeto hospedado no Github.

Clonagem de Repositório. É preciso ter o git instalado, caso o contrário, acesse o site[2] para instalação. E execute a seguinte linha de comando:

```
$ git clone --branch release_3 https://github.com/Unity-Technologies/ml-agents.git
```

Instalar o Pacote ML-Agents na Unity. Para fazer uso do ML-Agents deve-se importar o pacote com.unity.ml-agents e associá-lo ao projeto. Na Unity, navegue até o a aba Window, situado na barra superior, e logo após em Package Manager. Em seguida, localize o botão + e selecione Add package from disk..., navegue da pasta raiz do ML-Agents (instalado pelo git clone) até a pasta com.unity.ml-agents, abra-a e selecione o arquivo package.json (Figura 7.4).

Concluindo esse procedimento de instalação do ML-Agents em um projeto da Unity, pode-se prosseguir para criação dos projetos de aprendizagem.

7.4. Projetos de Aprendizagem Profunda

Nesta seção são apresentados dois Ambientes de Aprendizado: A (Seção 7.4.1) e B (Seção 7.4.2). No Ambiente de Aprendizado A, destina-se a criação de um ambiente inspirado no tutorial apresentado pelo ML-Agents [Community 2020a], com pequenas modificações.

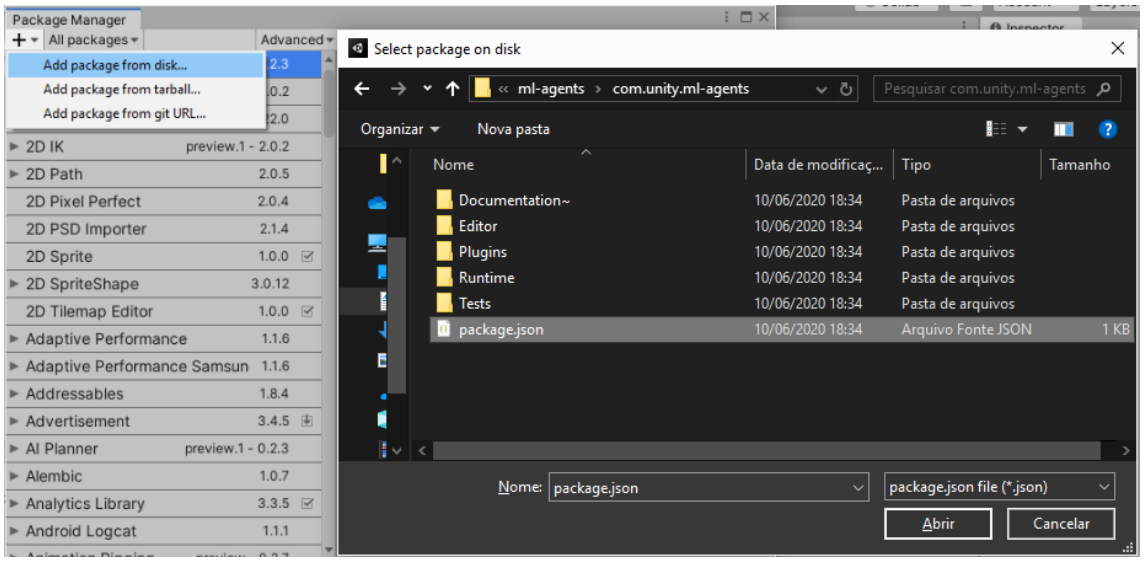


Figura 7.4. Adicionando pacote com.unity.ml-agents no editor Unity

[2] https://git-scm.com/

O Ambiente de Aprendizado B teve como base Ambiente A, porém com modificações na cena e restrições das ações do Agente com possibilidade de intervenção do usuário em tempo real. Em ambos os ambientes o objetivo do agente é alcançar um alvo.

Os códigos citados na nesta seção estão disponíveis no repositório hospedado no Github[3] para consulta. Em cada Ambiente de Aprendizado foram realizadas as seguintes etapas:

1. Criação do ambiente de treinamento na Unity (*scene*) com a configuração dos componentes necessários;
2. Implementação do script C# do Agente; e
3. Realização do treinamento do Agente com a visualização do resultado.

Considera-se que para prosseguir na criação e configuração dos ambientes descritos o projeto na Unity possua o pacote ML-Agents instalado, como foi mostrado na Seção 7.3. Para melhor visualizar as informações e procedimentos tratados nesta seção adotaram-se marcações diferenciadas para relacionar nomes e parâmetros que pertencem a Unity com sublinhado, e texto referente a código com fonte diferenciada.

Para começar, abra o UnityHub e crie um novo projeto 3D chamado LearningProject. A imagem padrão exibido pelo Unity terá como nome da *scene* SampleScene. Com o propósito de organização, foram criados mais dois *scenes*, para isso deve-se selecionar nas opções da aba de *scene* a opção Save Scene As..., ao abrir a janela especifique o nome de cada uma das *scenes* como: AAmbient e BAmbient, que corresponderão aos ambientes citados nas próximas subseções (7.4.1 e 7.4.2). Ambos devem ser colocados na pasta Scene presente na pasta Assets.

Existem GameObjects criados automaticamente que pertencem a ambos ambientes: Main Camera e Direcional Light. Altere os parâmetros do Main Camera pelo Inspector como segue: Position para $(0, 2.5, -8)$, Rotation para $(20, 0, 0)$, Clear Flags para Solid Color, e em Background coloque na paleta o código hexadecimal #FFFFFF que corresponde a cor branca.

Na construção dos Ambientes também foram utilizados materiais – configuração de cor e textura – presentes no repositório do ML-Agents. Para isso, foi copiada a pasta presente no subdiretório do SharedAssets do ML-Agents com o nome Materials e colocado dentro da pasta Assets do ProjetoAprendizagem.

7.4.1. Ambiente de Aprendizado A

O Ambiente de Aprendizado A é simples (Figura 7.6b), sendo composto por três tipos de GameObject: Plane, um solo limitado; Sphere, uma esfera de raio unitário; e Cube, uma caixa também de raio unitário. Esses GameObjects ficarão agrupados em um GameObject vazio e criado um Prefab para que possamos posteriormente replicar a mesma estrutura para o treinamento posterior.

[3] `https://github.com/devcavi/mlagentscourse`

Criação do Prefab: Crie um GameObject vazio dentro da aba Hierarchy, para isso acesse a aba GameObject a opção Create Empty, e renomeie o objeto criado com o nome TrainingAreaA. Esse será o nome do Prefab. Para prosseguir, deve-se transformar esse objeto em Prefab e posteriormente serão adicionados os elementos do ambiente de treinamento. Crie na pasta Assets uma subpasta com o nome Prefabs e abra essa pasta. Para criar um Prefab basta arrastar o objeto TrainingAreaA para a pasta Prefabs. Feito isso, clique no Prefab TrainingAreaA, dentro da pasta Prefabs, para continuar configurando o ambiente.

Agora, irá aparecer na aba Hierarchy somente o objeto TrainingAreaA. Clique sobre este objeto com o botão direito e selecione dentro da opção 3D Object os objetos Sphere, Cube e Plane (uma de cada vez). O próximo passo será renomear esses objetos: Sphere $\rightarrow$ RollerAgent, Cube $\rightarrow$ Target e Plane $\rightarrow$ Floor. Para renomear, selecione o objeto com o botão direito e siga para a opção Rename. O próximo passo será a configuração de cada um desses três objetos via interface da Unity para terem uma estrutura similar a apresentada nas Figuras 7.5a e 7.5b.

- Floor. A modificação feita será a inserção de uma textura (material), para isso acesse a pasta Materials e localize o arquivo GridMatFloor. Selecione esse arquivo e arraste até o objeto Floor. Ao fazer isso, verá que o material foi aplicado ao objeto.
- Target. Inicialmente, realize a inserção do material com nome AgentBlue, da mesma forma que realizado no Floor. No Inspector do objeto, modifique a Position para $(3.5, 0.5, 2.5)$.
- RollerAgent. Aplique também um material ao objeto, nome Yellow. Altere sua Position para $(0, 0.5, 0)$. Neste objeto será adicionado uma componente chamada Rigidbody. Para isso, via Inspector, clique no botão Add Component e procure por essa componente, selecionando-a. Isso possibilita ao objeto interagir de acordo com as leis físicas dentro da Unity.

Para fazer com que o RollerAgent seja um Agente usado para treinamento, outras três componentes deverão ser inseridas ao mesmo. Da mesma forma que foi adicionado a componente Rigidbody, insira a primeira componente da seguinte forma: selecione New script e coloque como nome AgentA e clique em Create and Add. Abra o script AgentA e insira o Código 7.1.

```
1  using System.Collections;
2  using System.Collections.Generic;
3  using UnityEngine;
4  using Unity.MLAgents;
5  using Unity.MLAgents.Sensors;
6
7  public class RollerAgentA : Agent
8  {
9      Rigidbody rBody;
10     public Transform target;
11     public float speed = 10;
12
13     void Start () {
14         rBody = GetComponent<Rigidbody>();
15     }
16
17     public override void OnEpisodeBegin()
18     {
```

```
19          if (this.transform.localPosition.y < 0)
20          {
21              this.rBody.angularVelocity = Vector3.zero;
22              this.rBody.velocity = Vector3.zero;
23              this.transform.localPosition = new Vector3(0, 0.5f, 0);
24          }
25
26          target.localPosition = new Vector3(Random.value * 8 - 4,
27                                             0.5f, Random.value * 8 - 4);
28      }
29
30      public override void CollectObservations(VectorSensor sensor)
31      {
32          sensor.AddObservation(target.localPosition);
33          sensor.AddObservation(this.transform.localPosition);
34          sensor.AddObservation(rBody.velocity.x);
35          sensor.AddObservation(rBody.velocity.z);
36      }
37
38      public override void OnActionReceived(float[] vectorAction)
39      {
40          Vector3 controlSignal = Vector3.zero;
41          controlSignal.x = vectorAction[0];
42          controlSignal.z = vectorAction[1];
43
44          rBody.AddForce(controlSignal * speed);
45
46          float distanceTotarget = Vector3.Distance(this.transform.localPosition,
47                                                    target.localPosition);
48
49          if (distanceTotarget < 1.42f)
50          {
51              SetReward(1.0f);
52              EndEpisode();
53          }
54
55          if (this.transform.localPosition.y < 0)
56          {
57              EndEpisode();
58          }
59      }
60  }
```

Código 7.1. Descrição do script de interação do Agente com o ambiente.

Os elementos do Código 7.1 serão discutidos a seguir, porém objetivo do RollerAgent, será alcançar o Target a partir de qualquer posição dos limites determinados pelo Floor. Para isso a rede terá que aprender como gerar as forças que serão aplicadas no agente, que possui propriedades de corpo rígido, para que o objetivo seja alcançado.

Inicializando o agente: O processo de treinamento baseia-se em episódios em que o agente tenta alcançar seu objetivo. Cada episódio encerra no momento em que o RollerAgent atinge o Target (linha 49) ou fracassa, caindo do Floor (linha 55). Toda vez que o RollerAgent atinge o Target, o episódio encerra e o Target é movido para uma nova posição aleatória (linha 26); se o Target cair da plataforma, ele será movido novamente para cima do Floor.

Atributos da classe: O `RollerAgentA` possui três atributos, um privado (`rBody`) e os demais públicos (`Target` e `speed`). O fato de serem públicos permite com que possam ser atribuídos em tempo de execução pela interface da Unity. A funcionalidade de cada parâmetro pode ser sintetizada:

- `rBody`: é responsável pela interação física do agente no cenário. Por meio desse parâmetro que as forças serão aplicadas (linha 44) e a velocidade do agente será rastreada (linhas 33 e 34). Na inicialização, o `rBody` é capturado por meio do RollerAgent que possui uma componente Rigidbody (linha 14).

- `target`: é o objetivo do agente, ou seja, o Target. Ao executar o jogo, deve-se setar esse elemento na componente Roller Agent A via Inspector.

- `speed`: é um atributo do tipo real (`float`), responsável por armazenar o módulo da velocidade máxima no qual o RollerAgent irá se movimentar durante o treinamento. Será inicializado com o valor 10.

Pode-se observar que a classe `RollerAgentA` herda funções e propriedades pertencentes à classe `Agent` (linha 7), logo a descrição dos métodos que serão comentados à seguir são sobrecarregamentos (*override*) de funções herdadas:

- `Start()`: nesta função o atributo `rBody` é inicializado adquirindo a componente Rigidbody do RollerAgent com o método `GameObject.getComponent<T>()` (linha 14).

- `OnEpisodeBegin()`: esta função é chamada no início de cada episódio, será responsável por mudar os atributos para que o Target seja colocado em uma posição aleatória (linha 26), e que o RollerAgent seja movido para o centro da plataforma, resetando suas velocidades angular e linear no começo de cada episódio (linhas 21 a 23).

- `CollectObservations(VectorSensor sensor)`: esta função é responsável pelas coletas de dados da observação do ambiente. O tamanho deste vetor é determinado pela componente Behavior Paramenters que informa a quantidade de observações. A quantidade de dados que serão adicionados deverão ser do tamanho especificado nessa componente. Esse exemplo contém 8 (oito) observações: três para posição do Target (linha 31), três para posição do RollerAgent e dois para a velocidade do agente nos eixos x e z (linhas 33 e 34). Observa-se que o personagem não se desloca no eixo y, por isso a observação deste eixo não se faz relevante para o treinamento.

- `OnActionReceived(float[] vectorAction)`: esta função é responsável pela as ações do agente. Como o objetivo é o deslocamento no plano então será necessário aplicar uma força para o deslocamento no eixo x e z. As ações também estão associadas às saídas da rede. Neste caso, cada saída é um valor contínuo (real) no intervalo entre -1 e 1, isso é determinado também na componente Behavior Parameters. As forças serão adicionadas por meio da função `AddForce()` (linha 43), na qual a força aplicada corresponde a um vetor, criado à partir da saída da rede com a magnitude da velocidade máxima permitida (`speed`). Nesta função também é criado o sistema de recompensas caso o RollerAgent atinja o alvo (linha 49), sendo baseado na distância entre o RollerAgent e o Target (linha 45).

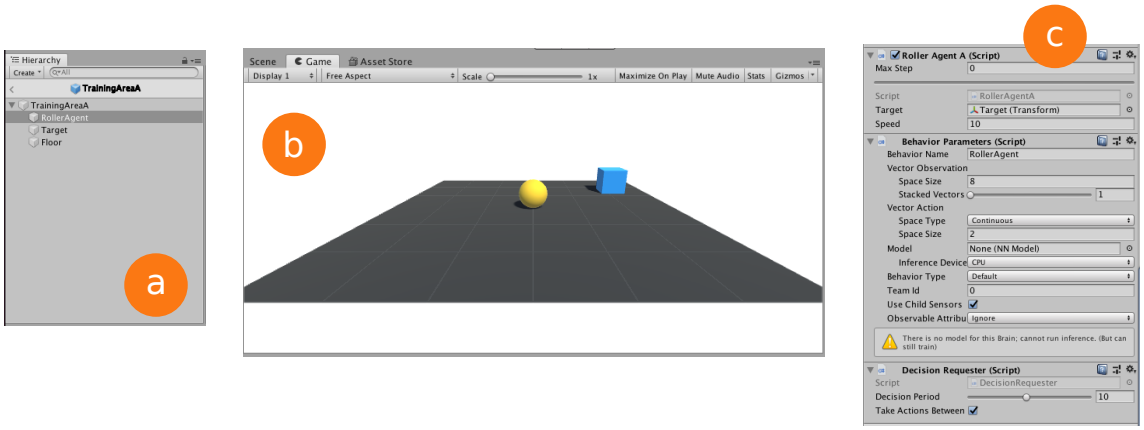


Figura 7.5. Configuração do ambiente de treinamento. a) Estrutura dos objetos. b) Visualização do Jogo pela câmera do Unity. c) Configuração das componentes necessárias do agente para a aprendizagem com ML-Agents.

Os dois componentes que faltam ao RollerAgent são responsáveis pela criação, configuração e comunicação para o treinamento da Rede Neural. Adicione as componentes Behavior Parameters e Decision Requester. Configure-os de acordo com dados da Figura 7.5c. Observe na imagem a configuração da quantidade de observações (Vector Observation) e ações (Vector Action), além do tipo de saída da rede (Continuous). A próxima etapa é definir a configuração do treinamento pretendido, para isso é necessário criar um arquivo de configuração com extensão yaml.

A última peça para concluir a configuração do agente para o treinamento ocorre de forma externa à Unity. Dentro da raiz do projeto LearningProject, crie uma pasta com nome config e dentro dela crie um arquivo chamado roller_agent.yaml. O conteúdo deste arquivo é disposto no Código 7.2.

```
1  behaviors:
2    RollerAgent:
3      trainer_type: ppo
4      hyperparameters:
5        batch_size: 10
6        buffer_size: 100
7        learning_rate: 3.0e-4
8        beta: 5.0e-4
9        epsilon: 0.2
10       lambd: 0.99
11       num_epoch: 3
12       learning_rate_schedule: linear
13     network_settings:
14       normalize: false
15       hidden_units: 128
16       num_layers: 2
17     reward_signals:
18       extrinsic:
19         gamma: 0.99
20         strength: 1.0
21     max_steps: 500000
22     time_horizon: 64
23     summary_freq: 10000
```

Código 7.2. Configuração de treinamento.

Alguns dados do Treinamento: No Código 7.2, o Behavior Name é definido na linha 2, que deve ser idêntico ao definido na componente Bahavior Parameters. É a partir

dessa informação que a configuração da rede e os dados para treinamento, definidos no arquivo yaml, serão cruzados. Na linha 3, foi definido o tipo de algoritmo utilizado no treinamento; o *Proximal Policy Optimization* (PPO) [Schulman et al. 2017] é o método indicado pelos desenvolvedores do ML-Agents, pois demonstrou ser mais geral e estável do que outros algoritmos de RL. Os parâmetros associados ao PPO, como seus limites, podem ser consultados na referência [AurelianTactics 2018a].

A principal contribuição do PPO é garantir que uma nova atualização da política não mude muito da política anterior. Isso leva a uma menor variação no treinamento ao custo de algum viés, mas garante um treinamento mais suave e também certifica que o agente não siga um caminho irrecuperável de ações sem sentido [Trivedi 2019].

Outras informações sobre a rede se encontram nas linhas 13 a 16 (Código 7.2). Além das camadas de entrada e saída, definidas na componente Behavior Parameters, essa rede possui duas camadas ocultas com 128 nós cada (linha 15). Outro parâmetro importante é o `max_steps` (linha 21) que define o limite de passos de treinamento, quando esse número é alcançado o algoritmo de treinamento é finalizado. Agora, o agente está com todas as configurações necessárias para o treinamento, a seguir serão definidos os passos realizados para executar o algoritmo de treinamento.

Antes de começar o treinamento, deve-se criar um ambiente virtual em Python, cuja finalidade é separar projetos, dependências e bibliotecas em um único lugar. Aqui utilizou-se o ambiente Linux, porém os mesmos passos podem ser realizados para outro sistema operacional. Abra o terminal e execute os comandos a seguir, dentro da pasta do projeto (LearningProject):

```
$ sudo apt-get install python3-venv
$ python3 -m venv env
```

Ative o ambiente virtual e instale as atualizações dos pacotes `pip` e `setuptools` e a seguir instale o pacote `ml-agents 0.17` (release_3):

```
$ source env/bin/activate
(env) $ pip install --upgrade pip
(env) $ pip install --upgrade setuptools
(env) $ pip install mlagents==0.17
```

Para realizar o treinamento, com o ambiente virtual ativado (`env`), deve ser inserido o seguinte comando:

```
(env) $ mlagents-learn config/roller_agent.yaml --run-id=AgentA
```

a opção `--run-id` determina um nome dado ao treinamento que será executado. O arquivo de configuração também deve ser informado (roller_agent.yaml). Esses são os parâmetros obrigatórios para execução de qualquer treinamento.

Ao executar o comando aparecerá no terminal algo similar a Figura 7.6a, onde será necessário dar um "play" no Unity para iniciar o treinamento (Figura 7.6b). Vale ressaltar que o comando de treinamento (`mlagents-learn`) possui diversas opções, onde destacam-se:

- `--env`: responsável por informar o executável do ambiente, fazendo com que não seja preciso clicar em "play" na Unity.

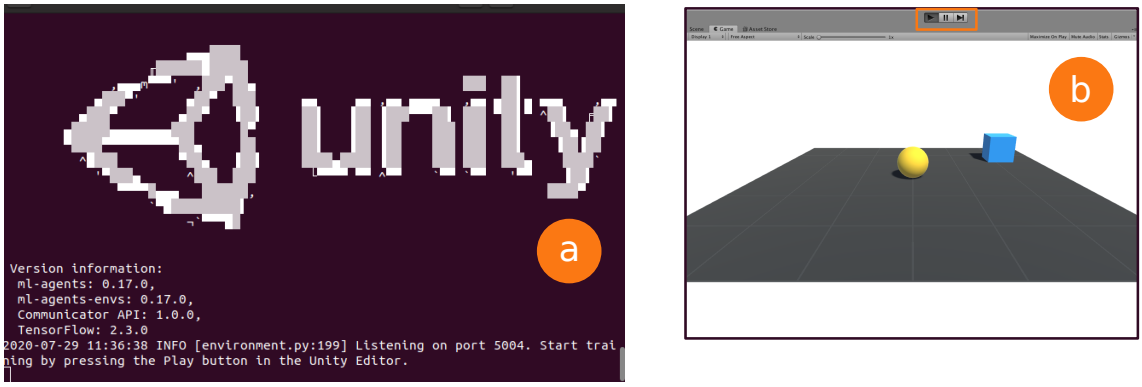


Figura 7.6. Iniciando o treinamento do agente. a) Tela de espera para execução do treinamento vinculado ao Unity. b) Acionamento do "Play" (botão pressionado do retângulo em destaque) para iniciar o treinamento.

- `--resume`: continua o treinamento de um determinado `--run-id`.
- `--force`: faz com que um treinamento seja executado sobrescrevendo um `--run-id` existente.

O treinamento deste Agente demorou cerca de 2h11" em um computador Intel® Core™ i7-6500U CPU 2.50GHz×4. Para minimizar esse tempo pode ser inserido vários agentes na Scene replicando-se o Prefab do TrainingAreaA, inserindo cada um em uma posição diferente. Com 16 áreas de treinamento minimizou-se esse tempo para 17".

O comando de treinamento cria uma pasta no projeto chamado results onde ficarão os arquivos relacionados ao treinamento do agente. Ao final do treinamento acesse a pasta results/AgentA. O arquivo RollerAgent.nn é o resultado do treinamento, que deve ser inserido no Behavior Paramenters no campo Model na Unity. Com isso, é possível visualizar o resultado do treinamento ao acionar botão "play". Essa inserção deve ser via o Prefab do TrainingAreaA para que todos os agentes criados a partir dele possam "enxergar" o treinamento realizado. O resultado pode ser visto na Figura 7.7, onde toda vez que o agente acerta o objetivo a posição do objetivo é alterada e reaparece no ambiente.

7.4.2. Ambiente de Aprendizado B

Este é o Ambiente de Aprendizado B, onde pequenas alterações foram realizadas em relação ao Ambiente de Aprendizado A. Agora o Target pode mudar sua posição aleatoriamente também no eixo y e o usuário tem a possibilidade de mudar a velocidade máxima (`speed`) do RollerAgent em tempo de execução do jogo pela interface da Unity.

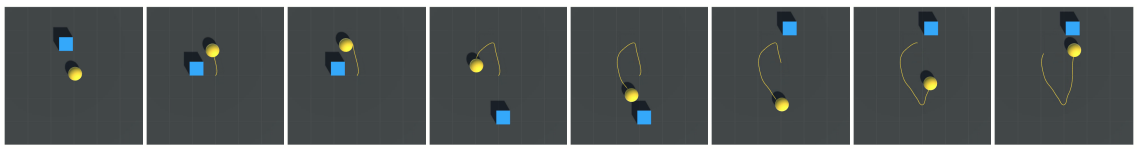


Figura 7.7. Animação do agente treinado. *Frames* da visão superior para melhor observar o movimento do agente em conjunto com um rastro (leitura da esquerda para direita).

Além dessas mudanças, o Agente é restito a ficar o menor tempo possível no ar, ou seja, as forças aplicadas em direção ao eixo y seriam limitadas de acordo com a necessidade, isso é o Target se encontrar fora do alcance na direção y. Para que tudo isso seja possível acontecer, foram introduzidas mudanças na função de recompensa e nas configurações das componentes do Agente e em suas observações e ações.

Para configuração do BAmbient, execute as mesmas etapas descritas na criação do AAmbient, alterando as nomenclaturas do Prefab de treinamento para TrainingAreaB e do script do Agente para RollerAgentB. Concluída a criação do ambiente, serão realizadas as alterações.

Alteração no Prefab RollerAgentB: Selecione o GameObject Plane e localize o atributo Tag no Inspector (abaixo do nome do objeto). Crie uma nova *tag* com o nome Floor. Essa marcação será útil para saber quando o Agente está no solo a partir do script.

Alteração do script do RollerAgentB: Foram inseridos três parâmetros no script: `randomVel`, indicando se haverá (ou não) aleatoriedade na velocidade máxima; e `speedRand`, indicando o valor da velocidade máxima em cada episódio. A variável `speed`, que já existia no Ambiente de Aprendizado A, guardará agora o valor máximo permitido para a velocidade máxima, neste caso foi determinado 250. Todos os atributos do RollerAgentB estão descritos no Código 7.3. Por fim, a variável `touchingGround` será responsável por informar se o RollerAgentB está ou não tocando o Floor. Para que isso aconteça de forma correta deve-se especificar o nome da Tag associado ao Floor que é uma constante (linha 7).

```
1  Rigidbody rBody;
2  public Transform target;
3  public float speed = 250;
4  public bool randomVel = false;
5  float speedRand;
6  bool touchingGround;
7  const string k_Ground = "Floor";
```

Código 7.3. Atributos da script do Agente para interação com o ambiente.

A função `OnEpisodeBegin()` deve ser modificada de acordo com o Código 7.4. Na linha 3, caso o Agente alcance uma altura superior a 10 ele é reinicializado. Na linha 9, o `speedRand` recebe uma nova velocidade máxima que varia no intervalo $[30, \texttt{speed}]$. Na linha 12, foi adicionado um intervalo randômico no eixo y do `target`, variando no intervalo $[0.5, 2.5]$.

```
1  public override void OnEpisodeBegin()
2  {
3   if (this.transform.localPosition.y < 0 || this.transform.localPosition.y > 10)
4   {
5       this.rBody.angularVelocity = Vector3.zero;
6          this.rBody.velocity = Vector3.zero;
7          this.transform.localPosition = new Vector3(0, 0.5f, 0);
8          if (randomVel){
9              speedRand = Random.value*(speed-30.0f) + 30.0f;
10         }
11  }
12  Target.localPosition = new Vector3(Random.value * 8 - 4,
13                                     Random.value*2.0f + 0.5f, Random.value * 8 - 4);

14 }
```

Código 7.4. Função que determina o início de cada novo episódio do Agente.

Será necessário adicionar dois novos métodos no Agente: `OnCollisionEnter()` `OnCollisionExit()`; descritos no Código 7.5. O primeiro verifica se houve o contato do Agente no objeto que possui uma Tag com nome definido na variável `k_Ground`, isso é, o objeto Floor (linha 3). A segunda indica se o Agente perdeu o contato com o objeto com essa Tag (linha 9). Essas ações informam, respectivamente, se o `touchingGround` é verdadeiro ou falso. Para melhorar a resposta dessas duas funções, a componente Rigidbody do Agente deve ter o atributo Collision Detection alterada para Continuous.

```
1  void OnCollisionEnter(Collision col)
2  {
3      if (col.transform.CompareTag(k_Ground))
4       touchingGround = true;
5  }
6
7  void OnCollisionExit(Collision col)
8  {
9      if (col.transform.CompareTag(k_Ground))
10      touchingGround = false;
11 }
```

Código 7.5. Funções responsáveis por tratar o toque do Agente no solo.

Na função `CollectObservations(VectorSensor sensor)` foram adicionadas outras observações, como é possível observar no Código 7.6. O total de novas observações somam 3 (três): informação se o agente toca ou não o solo (linha 3), velocidade no eixo *y* do agente (linha 6) e valor da velocidade máxima do Agente no episódio (linha 8 ou linha 10). As duas opções de velocidade dependem da informação se a velocidade é aleatória ou não. Neste trabalho, a variável `randomVel` é verdadeira no momento do treinamento do Agente, porém será trocada para falso uma vez que estiver treinado, dependendo somente da velocidade atribuída pelo usuário.

```
1  public override void CollectObservations(VectorSensor sensor)
2  {
3      sensor.AddObservation(touchingGround ? 1 : 0);
4      sensor.AddObservation(target.localPosition);
5      sensor.AddObservation(this.transform.localPosition);
6      sensor.AddObservation(rBody.velocity);
7      if(randomVel){
8          sensor.AddObservation(speedRand/speed);
9      } else {
10         sensor.AddObservation(speed/250.0f);
11     }
12 }
```

Código 7.6. Função reponsável por capturar as observações do Agente.

Por fim, a função responsável pelas ações foi bastante modificada, principalmente em termos de recompensa (`OnActionReceived()`). O Código 7.7 contém as alterações realizadas. Observa-se que agora são três ações recebidas, indicando a direção da aplicação da força no Agente. A magnitude da velocidade depende se está em modo randômico (linha 9) ou fixo (linha 7). Outra alteração foi na recompensa, onde a função utilizada agora é a `AddReward()`, que é responsável por acumular as recomensas durante o treinamento até sua finalização (`EndEpisode()`). Na linha 13, foi inserida uma penalização ao Agente, com a finalidade de que ele chegue no objetivo o mais rápido possível. De forma semelhante, se o Agente ficar muito tempo fora do solo (no ar), ele é penalizado (linha 15). Somente quando o Agente alcançar o objeto, que ele somará uma recompensa

positiva (linha 18). Caso o personagem caia do solo ou alcance uma altura superior a 10 é fixada uma recompensa ao episódio (linha 23).

```
1  public override void OnActionReceived(float[] vectorAction)
2  {
3      Vector3 controlSignal = Vector3.zero;
4      controlSignal.x = vectorAction[0];
5      controlSignal.y = vectorAction[1];
6      controlSignal.z = vectorAction[2];
7      Vector3 force_app = (controlSignal * speed);
8      if(randomVel){
9          force_app = (controlSignal * speedRand);
10     }
11     rBody.AddForce(force_app);
12     float distanceTotarget = Vector3.Distance(this.transform.localPosition,
13                                               target.localPosition);
14     AddReward(-0.0001f);
15     if(!touchingGround)
16         AddReward(-0.005f);
17     if (distanceTotarget < 1.42f)
18     {
19         AddReward(1.0f);
20         EndEpisode();
21     }
22     if (this.transform.localPosition.y < 0 || this.transform.localPosition.y > 10)
23     {
24         SetReward(-0.5f);
25         EndEpisode();
26     }
27 }
```

Código 7.7. Função reponsável por realizar as acões do Agente e pelo cálculo da recompensa.

Essas inserções de valores à recompensa têm como objetivo moldar o comportamento do Agente, fazendo com que ele alcance o objetivo de forma rápida e mais realista possível, ou seja, sem ficar flutuando no ar o tempo todo. Para utilizar o `AddReward()` é necessário limitar a quantidade de passos (*steps*) de um episódio. Isso deve ser realizado na componente Roller Agent B na interface da Unity no campo Max Steps, atribua esse valor para 2500. Como foi alterado também a quantidade de observações e ações, altere-as respectivamente para 11 e 3 na componente Behavior Parameters.

Realizadas essas últimas modificações, pode-se partir para o treinamento do Agente. Para isso, o ambiente virtual deve ser ativado e dentro do diretório raiz do projeto deve ser executado o comando:

```
(env) $ mlagents-learn config/roller_agent.yaml --run-id=AgentB
```

Pode-se observar que foi utilizado o mesmo arquivo de do treinamento feito no Agente do Ambiente A.

O tempo de treinamento foi de 19” considerando 16 Agentes ao mesmo tempo. Na Figura 7.8, pode-se observar a sequência de frames descrevendo a ação do personagem no Ambiente B, nesta animação foi escolhida uma velocidade máxima de 45.

Na próxima seção, serão apresentados os gráficos produzidos por cada um dos treinamentos relativos aos identificadores `AgentA` e `AgentB`, descritos no comando `mlagents-learn` (`--run-id`).

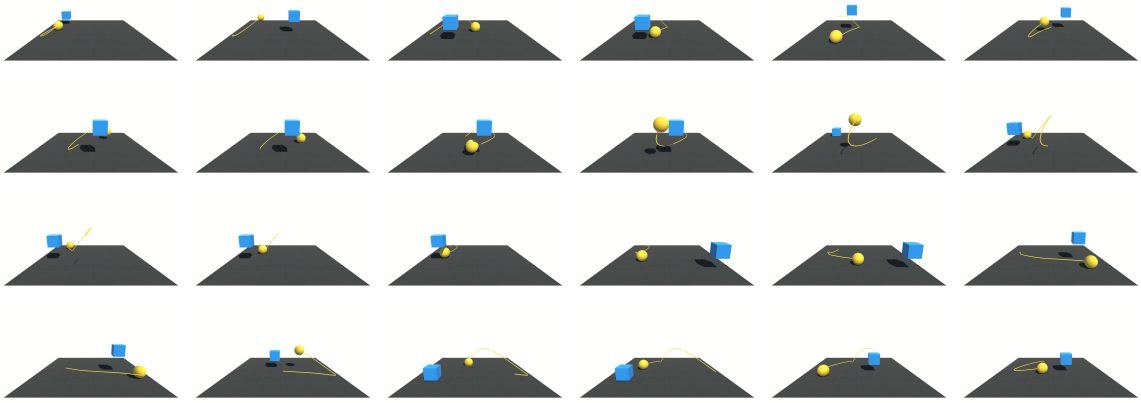


Figura 7.8. Animação do Agente treinado (leitura da esquerda para direita de cima para baixo). A visão escolhida foi em perspectiva para observar a alteração no eixo y da caixa.

7.5. Análise gráfica do Tensorboard

O kit de ferramentas do ML-Agents armazena informações adquiridas durante a sessão de aprendizado, assim os desenvolvedores podem acompanhar o treinamento da rede por meio de gráficos. A ferramenta responsável por mostrar essas informações pertence ao Tensorflow e chama-se Tensorboard [Community 2020b]. Nesta seção é realizada uma análise de alguns gráficos do treinamento, que encontram-se na íntegra no Tensorboard-Dev[4] .

O comando `mlagents-learn` salva os arquivos com os dados de treinamento em uma pasta chamada results, organizada pelo nome especificado no `--run-id` que foi designada para na seção de treinamento. Para observar o processo de treinamento antes, durante ou depois, deve-se iniciar o Tensorboard pelo terminal:

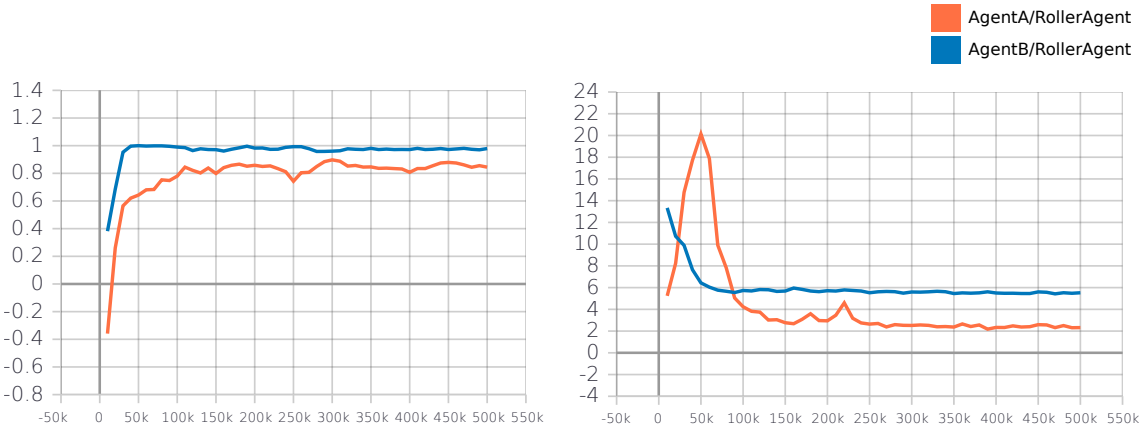
```
(env) $ tensorboard --logdir=results
```

Para executar esse comando, o ambiente virtual deve estar ativado. A localização da pasta deve ser informada na opção `--log-dir`. Por padrão, o Tensorboard opera na porta 6006, podendo ser alterado com a opção `--port` seguido do número da porta desejada. Ademais, os dados serão exibidos no navegador por meio da porta especificada, neste caso `http://localhost:6006/`.

O Tensorboard divide em os gráficos gerados em quatro categorias: *Environment* (Ambiente), dados relacionados aos valores de recompensa adquirida entre os episódios de treinamento; *Is Training* (está treinando), dados que informam se o agente está treinando ainda ou finalizou o treinamento; *Losses* (Perdas), dados que indicam como está se comportando o algoritmo de aprendizagem do agente; e *Policy* (Política), dados que exibem o decaimento dos parâmetros da função de aprendizado durante o treinamento. Destacam-se a seguir alguns gráficos produzidos e sua interpretação.

Na Figura 7.9, tem-se os gráficos que mostram se o personagem conseguiu performar bem o treinamento. Observa-se que o `AgentA` após um curto espaço de tempo atingiu

[4] `https://tensorboard.dev/experiment/Zhm1zVjNTQWrDJ1TPDY0jA/`

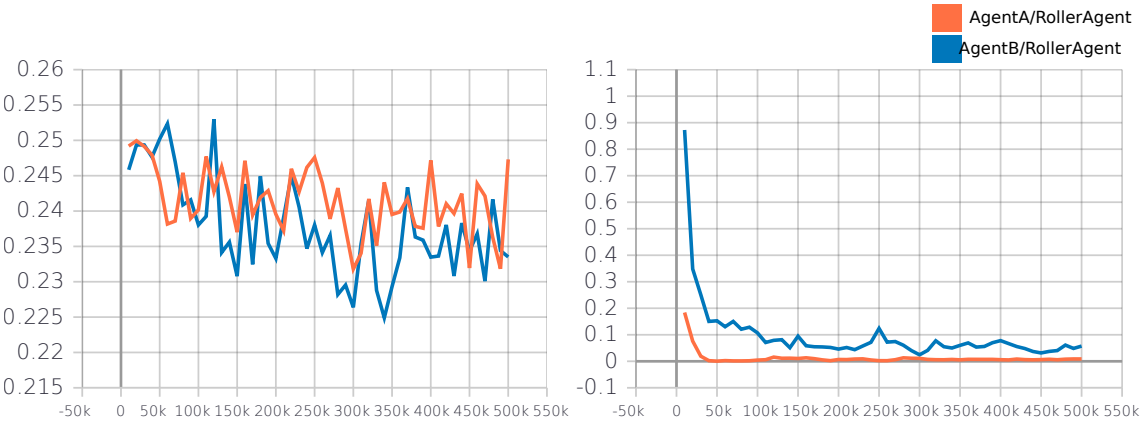


(a) Environment/Cumulative Reward (b) Environment/Episode Length

Figura 7.9. Gráficos relacionados à categoria *Environment*. O eixo x apresenta a quantidade de passos de treinamento, onde é no máximo 500K (mil). (a) exibe a recompensa alcançada pelo Agente durante o aprendizado. (b) indica a duração média de cada episódio.

frequentemente a recompensa máxima (1.0), contudo o `AgentB` não consegue alcançar o valor máximo devido aos descontos aplicados durante cada passo do treinamento (Figura 7.9a). O desconto aplicado ao `AgentB`, influencia também o tempo gasto no treinamento, levando a uma duração de episódio relativamente menor do que em relação ao `AgentA`, pois ele é forçado a chegar o mais rápido possível no objetivo (Figura 7.9b).

Na Figura 7.10, pode-se observar gráficos relacionados ao *Losses*. Pode-se mensurar o quanto função de perda de política está se modificando, em média esses valores tendem a ser menores que 1 (Figura 7.10a). Essas variações estão relacionadas também à aleatoriedade do modelo de treinamento. Percebe-se que o gráfico ao lado, Figura 7.10b, indica a perda média da atualização da função de valor, quado inicia-se o treinamento o valor é grande e com o treinamento bem sucedido tende a diminuir e se estabilizar.



(a) Losses/Value (b) Losses/Policy

Figura 7.10. Gráficos relacionados à categoria *Losses*. (a) Correlaciona o quanto a política (processo para decidir ações) está mudando. (b) indica o quanto a função de aprendizado está na direção correta.

Existem outros gráficos que possuem a tendência em comum de decrescerem ao longo do tempo, esses são: Epison, Beta e Learning Rate. Para uma descrição completa do significado desses e de outros parâmetros consulte [AurelianTactics 2018b].

7.6. Conclusão e Considerações Finais

O pacote ML-Agents, em conjunto com a plataforma Unity, fazem com que a criação de cenários, bem como de desenvolvimento e treinamento de Agentes Inteligentes, seja feita de forma simples para pessoas que são da área de Inteligência Artificial, ou não. Dois exemplos foram apresentados a fim de ilustrar o uso dessas ferramentas, onde foi visto que o algoritmo para treinamento do Agente pode ser utilizado como uma caixa preta.

Com a finalidade de observar a Unity sob novos parâmetros e se aprofundar no assunto, é possível analisar diversos outros exemplos de ambiente. São recomendados três canais relacionados, da plataforma YouTube: *Bot Academy*[5], Sebastian Schuchmann[6] e *Immersive Limit*[7]. A plataforma *Medium*[8] também dispõe diversos artigos acerca do assunto.

Todos os arquivos dispostos no projeto de aprendizagem e mencionados no Tensorboard, assim como os tutoriais em vídeo estão disponíveis no repositório git do Grupo de Estudos CAVI (*Computer Graphics, Animation, Visualization and Inteligence*)[9].

[5] `https://www.youtube.com/BotAcademy`
[6] `https://www.youtube.com/SebastianSchuchmannAI`
[7] `https://www.youtube.com/ImmersiveLimit`
[8] `https://medium.com/`
[9] `https://github.com/devcavi/mlagentscourse`

Referências

[AurelianTactics 2018a] AurelianTactics (2018a). Ppo hyperparameters and ranges. https://medium.com/aureliantactics/ppo-hyperparameters-and-ranges-6fc2d29bccbe. Acesso em: 30 de jul. de 2020.

[AurelianTactics 2018b] AurelianTactics (2018b). Understanding ppo plots in tensorboard. https://medium.com/aureliantactics/understanding-ppo-plots-in-tensorboard-cbc3199b9ba2. Acesso em: 30 de jul. de 2020.

[Community 2020a] Community, U. (2020a). Overview ml-agents. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/ML-Agents-Overview.md. Acesso em: 30 de jul. de 2020.

[Community 2020b] Community, U. (2020b). Using tensorboard to observe training. https://github.com/Unity-Technologies/ml-agents/blob/master/docs/Using-Tensorboard.md. Acesso em: 30 de jul. de 2020.

[Doyle et al. 2013] Doyle, J. C., Francis, B. A., and Tannenbaum, A. R. (2013). Feedback control theory. Courier Corporation.

[Juliani et al. 2018] Juliani, A., Berges, V.-P., Teng, E., Cohen, A., Harper, J., Elion, C., Goy, C., Gao, Y., Henry, H., Mattar, M., and Lange, D. (2018). Unity: A general platform for intelligent agents.

[Lanham 2018] Lanham, M. (2018). Learn Unity ML-Agents – Fundamentals of Unity Machine Learning. Packt Publishing Ltd., Livery Place 35 Livery Street Birmingham B3 2PB, UK.

[Mills and Stufflebeam 2005] Mills, F. and Stufflebeam, R. (2005). Introduction to intelligent agents. http://www.mind.ilstu.edu/curriculum/ants_nasa/intelligent_agents.php. Acesso em: 30 de jul. de 2020.

[Pereira 2019] Pereira, W. (2019). Api: conceito, exemplos de uso e importância da integração para desenvolvedores. https://take.net/blog/devs/api-conceito-e-exemplos. Acesso em: 30 de jul. de 2020.

[Poole and Mackworth 2010] Poole, D. L. and Mackworth, A. K. (2010). Artificial Intelligence: foundations of computational agents. Cambridge University Press.

[Schulman et al. 2017] Schulman, J., Wolski, F., Dhariwal, P., Radford, A., and Klimov, O. (2017). Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347.

[Sutton and Barto 2018] Sutton, R. S. and Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

[Trivedi 2019] Trivedi, C. (2019). Proximal policy optimization tutorial (actor-critic method). https://towardsdatascience.com/proximal-policy-optimization-tutorial-part-1-actor-critic-method-d53f9afffbf6. Acesso em: 30 de jul. de 2020.