BRUNO GONÇALVES ROCHA

UM EXPERIMENTO DE SIMULAÇÃO ROBÓTICA COM MACHINE LEARNING

Ji-paraná
2021

BRUNO GONÇALVES ROCHA

UM EXPERIMENTO DE SIMULAÇÃO ROBÓTICA COM MACHINE LEARNING

Monografia apresentada à Banca Examinadora do Centro Universitário São Lucas Afia, como requisitos de aprovação para obtenção do Título de Bacharel em Sistema de Informação, sob a orientação do Prof. Pablo Nascimento e coorientação pelo Prof. Maigon N. Pontuschka.

Ji-Paraná
2021

Dados Internacionais de Catalogação na Publicação - CIP

R672e Rocha, Bruno Gonçalves.

Um experimento de simulação robótica com Machine Learning. / Bruno Gonçalves Rocha. – Ji-Paraná, 2021.
56 p.; il.

Monografia (Curso de Sistemas de Informação) – Centro Universitário São Lucas Ji-Paraná, 2021.

Orientador: Prof. Pablo Henrique Gonçalves Nascimento

1. Robótica. 2. Simulação. 3. Inteligência artificial. 4. Machine Learning. I. Nascimento, Pablo Henrique Gonçalves. II. Título.

CDU 004.89

Ficha Catalográfica Elaborada pelo Bibliotecário Giordani Nunes da Silva CRB 11/1125

RESUMO

Este trabalho surgiu com base em um estudo realizado por pesquisadores do ITA/INPE em que dois robôs com propulsores de jatos de ar se deslocavam sobre uma superfície de baixo atrito com o objetivo de se aproximarem e se acoplarem de forma autônoma. As limitações impostas pelos recursos físicos do experimento realizado no mundo real nos fizeram pensar que seria possível realizar com vantagens estes mesmos testes em um ambiente simulado. Neste trabalho recriamos os robôs do experimento do ITA/INPE no *Blender* e os inserimos em uma simulação no motor de jogos *Unity3D*. Este experimento de inteligência artificial utilizou técnicas de Machine Learning, em especial o método de aprendizado por reforço com o pacote *Unity ML-Agents Toolkit* para que os robôs possam se mover e alcançar algumas metas, como a de um robô encostar no outro. Durante esse treinamento da inteligência artificial, foram realizados diversos treinamentos para aumentar gradualmente a dificuldade e melhorar o desempenho do robô. O resultado obtido foi satisfatório pois foi possível treinar os robôs a se encontrarem, mesmo quando aparecessem em pontos aleatórios do espaço do experimento.

Palavras-chave: Robótica. Simulação. Inteligência artificial. Aprendizado por reforço Machine Learning.

Abstract

The present work was carried out based on a study by researchers from ITA/INPE in which two robots with air jet thrusters moved on a low friction surface in order to come closer and dock autonomously. The limitations imposed by the physical resources of the experiment in the real world made us think that it would be possible to perform these same tests with advantages in a simulated environment. In this work we recreated the robots from the ITA/INPE experiment in Blender and inserted them into a simulation in the Unity3D game engine. Our artificial intelligence experiment used Machine Learning techniques, in particular the Reinforcement Learning method with the Unity ML-Agents Toolkit package so that the robots might move and meet small goals such as touching each other. During this artificial intelligence training, several training sessions were carried out to gradually increase the difficulty and improve the robot's performance. The result obtained was satisfactory because it was possible to train the robots to meet, even when they appeared at random points in the space of the experiment.

Keywords: Robotics. Simulation. Artificial Intelligence. Reinforcement Learning. Machine Learning.

LISTA DE ILUSTRAÇÕES

LISTA DE ABREVIATURAS E SIGLAS

3D	Tridimensional.
bit	Binary digit (dígito binário).
EAP	Estrutura Analítica do Projeto.
GHz	GigaHertz.
IA	Inteligência Artificial.
ITA	Instituto Tecnológico da Aeronáutica.
INPE	Instituto Nacional de Pesquisas Espaciais.
RAM	Random Access Memory (Memória de Acesso Aleatório).
UDP	User Datagram Protocol.
INPE	Instituto de Pesquisas Espaciais.
UML	Unified Modeling Language (Linguagem de Modelagem Unificada).
WIFI	Wireless Fidelity – Termo utilizado para descrever conexão sem fio.

SUMÁRIO

1 INTRODUÇÃO

O Grupo de Pesquisa de Simulação Robótica do Centro Universitário São Lucas Ji-Paraná, criado em junho de 2019, se reúne semanalmente aos sábados e procura despertar o prazer pela pesquisa científica em uma parceria entre professores e acadêmicos do Departamento de Sistemas de Informação do Centro Universitário São Lucas Ji-Paraná e um professor do curso de Ciência da Computação da PUC-SP e pós doutorado pelo INPE. Um dos primeiros projetos realizados pelo grupo foi o estudo sobre um braço robótico programado em um *game engine* (ou motor de jogos) com a finalidade de que esta mesma programação pudesse ser utilizada em um braço robótico real. O grupo estuda possíveis aplicações de braços robóticos tanto para naves espaciais, como para operação em oceano profundo. O grupo de pesquisa se divide em dois subgrupos: um que se encarrega de fazer a modelagem 3D e o outro que se dedica ao estudo da programação em *game engine,* desenvolvendo ambientes virtuais e robôs virtuais.

Ao longo dos trabalhos do grupo, conhecemos o trabalho desenvolvido pelo ITA – Instituto Tecnológico da Aeronáutica em parceria com INPE – Instituto de Pesquisas Espaciais no estudo de equipamentos robóticos para uso no espaço. O interesse específico é desenvolver equipamentos que auxiliem no acoplamento de naves espaciais no espaço. Como não é possível fazer pesquisas diretamente no espaço, o grupo realiza simulações em mesas de baixo atrito para simular como os objetos se deslocam no espaço com propulsão por meio de jatos de ar e como seria possível fazer com que dois robôs conseguissem se aproximar e acoplarem-se com o uso de braços robóticos utilizando algoritmos de inteligência artificial.

O grupo do ITA/INPE enfrentou alguns problemas com limitações físicas nestes experimentos, pois necessitam de cilindros de ar comprimido que duram apenas 15 minutos. Deste modo os testes ficam muito caros e demorados.

Neste sentido, nosso grupo percebeu que seria possível recriar virtualmente o ambiente do experimento do ITA/INPE sem as limitações físicas, de modo a poder testar os algoritmos de aproximação e acoplamento dos robôs.

Assim, neste trabalho, iremos desenvolver um simulador em ambiente virtual na *game engine Unity*, juntamente com o pacote *Unity Machine Learning Agents Toolkit (ML-Agents)*, para simular, desenvolver e treinar um robô que fica em uma

base não inercial, ou seja, que simula o deslizamento do experimento do ITA/IMPE em uma mesa de vidro com baixíssimo atrito e monitorar seu comportamento no ambiente de maneira autônoma, de modo que possa se acoplar com outro robô virtual usando os mesmos parâmetros utilizados no experimento real do ITA/INPE.

A meta é que o robô seja capaz, de forma autônoma, de se deslocar pelo ambiente e encostar com o outro robô da maneira mais otimizada possível, economizando combustível.

1.2 PROBLEMATIZAÇÃO

O experimento realizado pelos professores da PUC-SP e do ITA (PONTUSCHKA; FONSECA, LIMA, 2016) é composto por duas unidades idênticas de manipuladores robóticos flutuantes sobre uma plataforma aerossuportada com o objetivo de realizar uma manobra de acoplamento das duas unidades. Por manipulador robótico entende-se um robô que possui um braço robótico que pode manipular elementos no ambiente. Uma plataforma aerossuportada é uma base que se movimenta sobre um colchão de ar criado por meio de ar comprimido. Os manipuladores robóticos deslocam-se por meio de propulsores de ar comprimido colocados em seus lados que soltam jatos de ar, fazendo com que se desloquem no sentido contrário ao lado que os propulsores soltam os jatos de ar. Outros equipamentos foram utilizados para este experimento que realizam a captura do posicionamento de cada unidade, e processam o algoritmo de acoplamento e coordenam o acionamento dos propulsores e dos braços robóticos dos manipuladores robóticos para que possam se acoplar.

O grande problema com este experimento é que o tempo de utilização dos cilindros de ar comprimido dura em média cinco minutos. O custo para recarregar os cilindros é alto e a necessidade de levá-los a um local em que isso possa ser feito faz com que seja um processo demorado além de custoso.

No contexto do grupo de pesquisa de Simulação Robótica, pensou-se a possibilidade de recriar este experimento em um mundo virtual 3D em que as condições de deslocamento inercial pudessem ser simuladas. Isto resolveria o problema da limitação imposta pelos cilindros de ar comprimido e permitiria estudar os algoritmos de posicionamento das unidades de modo a conseguir que se acoplem.

Um primeiro passo seria recriar o ambiente do experimento em um mundo virtual, para poder, em seguida, desenvolver um algoritmo que permita que cada robô possa se deslocar de modo a encontrar o outro e, finalmente, que consigam acoplar seus manipuladores.

Neste sentido o presente trabalho pretende fazer os dois primeiros passos desta tarefa de recriar o experimento dos professores da PUC-SP e do ITA, ou seja, criar o ambiente virtual 3D com um manipulador robótico propelido por propulsores e desenvolver um algoritmo que permita que este manipulador robótico se desloque até uma posição determinada, em que o outro manipulador robótico se encontraria.

1.3 OBJETIVO GERAL

Desenvolver um ambiente virtual 3D com o *game engine Unity*, para a recriação da parte inicial do experimento realizado por Pontuschka, Fonseca e Lima (2016), ou seja, o deslocamento do manipulador robótico até a posição do segundo manipulador de modo a tocá-lo.

1.3.1 Objetivos específicos

- Modelar uma réplica do manipulador robótico em *software* 3D;
- Emular o funcionamento do manipulador robótico real no manipulador robótico virtual na *Game Engine;*
- Desenvolver o ambiente de testes (cenário) para o robô no ambiente virtual;
- Realizar testes de locomoção com o uso de inteligência artificial de modo que o manipulador robótico se desloque com o uso dos propulsores e que toque o outro.

1.4 JUSTIFICATIVA

O presente tema para esta pesquisa surgiu como ideia no grupo de pesquisa Simulação Robótica para baixar os custos e aumentar opções de testes de robôs. Com a utilização de um ambiente virtual em uma *game engine* 3D, o desenvolvimento pode ser muito mais barato, possibilitando a implementação de mais componentes, além de ter a possibilidade de se compartilhar os arquivos com os participantes do grupo para que cada um possa fazer seus testes individualmente. Nesse contexto, o robô poderá utilizar Inteligência Artificial, para realizar ações básicas como ir de um ponto A até B. A simulação pode depois ser compartilhada com a equipe que realiza os testes no mundo real, diminuindo o tempo de teste bem como os custos e contribuindo com o avanço da pesquisa com os robôs reais.

2 REFERENCIAL TEÓRICO

2.1 GAME ENGINE

Uma *game engine,* de acordo com Gregory (2015), é um termo que surgiu nos anos 90 para se referir a um jogo em primeira pessoa muito popular da época chamado *Doom*. A arquitetura utilizada nele foi inovadora no fato de fazer uma separação entre os componentes do software, como por exemplo o sistema de renderização gráfica, sistema de detecção de colisão e sistema de áudio dos demais elementos de arte, regras de jogo e os mundos do jogo. Essa separação ganhou valor quando os criadores de jogos começaram a modificar os jogos, seja na arte, som ou equipamentos novos fazendo somente uma mudança mínima no *software* “motor”.

2.3 INTELIGENCIA ARTIFICIAL

Nas palavras de Silva, Lenz, Freitas e Bispo (2019), inteligência artificial é um termo que diferencia um *software* dos demais por ele ser inteligente e que tende a realizar atividades que antes eram exclusivamente humanas como, por exemplo, a tradução e reconhecimento facial.

2.3.1 O início da inteligência artificial

Coppin (2010), considera Aristóteles (384 a 322 A.C), como precursor da Inteligência artificial, pois nossos sistemas para raciocínio lógico são baseados na lógica que ele inventou e formam a base do pensamento científico moderno. Salienta que a principal contribuição de Aristóteles para a Inteligência Artificial foi o estudo da lógica, quando inventou a ideia do silogismo definiu como: “Um discurso no qual certas coisas tendo sido estabelecidas, alguma outra coisa segue da necessidade de elas assim serem.”

Os estudos sobre Inteligência Artificial começaram em 1950, com o Projeto de Pesquisas de Verão em Inteligência Artificial de *Dartmouth*, no *Dartmouth College*, em *Hanover*, *New Hampshire*, Estados Unidos.

Segundo Coppin “Inteligência Artificial envolve utilizar métodos baseados no comportamento inteligente de humanos e outros animais para solucionar problemas complexos.” (COPPIN,2010, pg.4).

2.3.2 Categorias da inteligência artificial

A inteligência artificial pode ser dividida em três categorias, cada uma com uma função diferente.

Inteligência artificial fraca: Também conhecida como Inteligência artificial limitada, utiliza uma grande quantidade de dados armazenados para realizar tarefas complexas com eles. Consegue realizar cálculos complexos de uma forma muito rápida, porém não pode ser programada para fazer algo além disso.

Silva et al. (2019, pg.17) definem a inteligência artificial fraca da seguinte forma:

> A inteligência artificial fraca é uma corrente de pesquisa e desenvolvimento que defende que nunca será possível construir máquinas inteligentes no real sentido da palavra, pois, para ela, a inteligência demanda consciência e autopercepção, habilidades impossíveis de serem recriadas. Tudo que se pode fazer envolve imitar comportamentos inteligentes e emoções, bem como resolver problemas, mas nunca a consciência, considerando que isso se resume a um conjunto de cálculos. (Silva et al., 2019, pg.17).

Inteligência artificial forte: Esse tipo de inteligência artificial, também conhecida como "nível humano”, é o tipo de inteligência artificial que consegue aprender utilizando técnicas de aprendizado de máquina. Com essas técnicas pode aprender a enxergar e reconhecer objetos e reagir a eles.

Sobre a inteligência artificial forte, Coppin, menciona que:

> Os seguidores da IA forte acreditam que, dispondo de um computador com capacidade de processamento e fornecendo a ele suficiente inteligência, pode-se criar um computador que possa literalmente pensar e ser consciente do mesmo modo que um humano é consciente. (COPPIN, 2010, pg.4).

Superinteligência: É uma área da inteligência artificial que se especula que será superior à inteligência humana. Poderá tomar decisões e armazenar os dados. Para definir o termo superinteligência, Silva cita a definição do filósofo sueco Nick Bostrom que a define como “um intelecto que é muito mais inteligente do que o melhor

cérebro humano em praticamente todas as áreas, incluindo criatividade científica, conhecimentos gerais e habilidades sociais” (BOSTROM, 2003, p. 12–17).

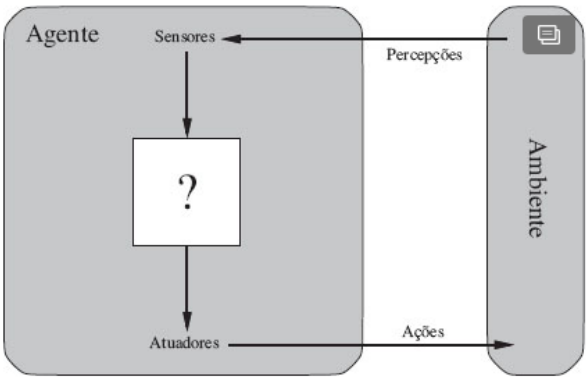
É nessa área que as pesquisas de hoje se concentram e debatem sobre a possibilidade de um futuro guiado por Inteligências Artificiais que, por pensarem de uma maneira muito mais inteligente que um ser humano, possam trazer um assustador futuro em que cause a extinção da raça humana.

2.3.3 Agentes inteligentes

É considerado um agente aquele que consegue perceber seu ambiente por meio de sensores e ainda agir nesse ambiente com o uso de atuadores. Por exemplo, nós, agentes humanos, possuímos olhos, ouvidos, mãos entre outros. Um agente robótico que é o foco desta pesquisa, possui câmera e detectores que funcionam como sensores e vários motores como sendo atuadores. Além disso, um agente de *software* pode receber comandos por teclas digitadas por uma pessoa, ou por meio de arquivos ou pacotes de dados em redes.

A percepção que o agente tem através das entradas percebidas pelos sensores formam uma sequência de percepções que é o histórico de tudo que é percebido. As ações que o agente pode tomar, vêm dessa sequência de percepções, e, por isso, é importante mapear as percepções para executar uma ação específica.

Figura 1 - Interação entre agente e ambiente



Fonte: NORVIG, **Inteligência artificial,** 2013. p.30.

Um exemplo simples são os aspiradores de pó autônomos, através dos sensores se movem entre os móveis limpando a casa. Para recarregarem, alguns

modelos possuem uma base em algum local da casa e o aspirador deve voltar para ela, o que é semelhante ao objetivo do presente trabalho.

Como demonstrado na imagem abaixo, o agente percebe que há um obstáculo com seus sensores e, de acordo com sua programação, realiza uma ação que pode ser de desviar do objeto para os lados ou ativar a função de limpeza.

Figura 2 - Um exemplo de agente inteligente

Fonte: iRobot, disponível em: <https://www.irobotloja.com.br>.

Todas as possibilidades de ações possíveis tomadas pelo agente só podem ser realizadas pelo uso da técnica de aprendizado de máquina, que através de intensos treinos com diferentes ambientes se fez possível existir um grande banco de dados para que a inteligência artificial, que opera os atores, possa tomar a melhor decisão possível, no caso do exemplo, realizar uma limpeza eficiente.

2.4 MACHINE LEARNING

Uma definição de aprendizado de máquina apresentada por Monard e Baranauskas (2005) é:

> Aprendizado de Máquina é uma área de IA cujo objetivo é o desenvolvimento de técnicas computacionais sobre o aprendizado bem como a construção de sistemas capazes de adquirir conhecimento de forma automática. Um sistema de aprendizado é um programa de computador que toma decisões baseado em experiências acumuladas através da solução bem-sucedida de problemas anteriores. (MONARD; BARANAUSKAS, 2005).

No que diz respeito ao aprendizado de máquina, não existe apenas uma maneira de executá-la, os vários métodos existentes acabam por compartilhar alguns pontos, seja na linguagem de programação ou até na forma de aprendizado.

2.4.1 Categorias do aprendizado de máquina

Existem três dos principais tipos de aprendizado de máquina:

Aprendizado Supervisionado: Segundo Breve (2010), neste tipo de aprendizado, a máquina irá aprender através de dados separados para o seu treinamento, sendo eles em pares de dados de entrada e saídas desejadas. O valor de saída pode ser predição ou contínuo. Seu objetivo é ter uma capacidade de realizar a predição como saída, para qualquer entrada.

Aprendizado Não Supervisionado: Breve (2010) explica que diferente do aprendizado anterior, este não tem dados já separados para iniciar o treinamento e não possui intervenção humana. A própria máquina terá de aprender a determinar como os dados estão organizados.

Aprendizado com Reforço: Outro autor, Coppin (2010), fala que aprendizado com reforço é um sistema cuja forma de aprendizado é através de reforço, ou seja, o sistema receberá um reforço positivo toda vez que ele realizar uma tarefa corretamente e um reforço negativo quando realizar uma tarefa incorretamente. Exemplo quando um robô chegar a um ponto sem bater em obstáculos receberá um reforço positivo, porém se bater em algum receberá um reforço negativo.

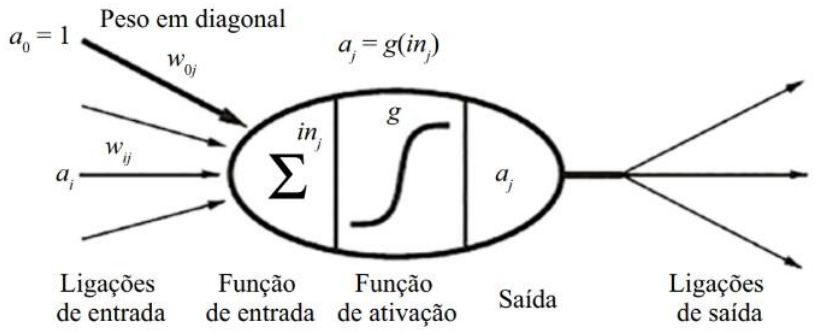
2.4.2 Redes neurais artificiais

As redes neurais artificiais, segundo SILVA, Fabrício et. al. (2019), começou como um subgrupo da inteligência artificial baseado em ciência ou modelagem cognitiva, que é como são representados os modelos computacionais de Inteligência artificial, visando processos de funcionamento da mente humana. Então partindo dos estudos de modelagem cognitiva a partir da mente humana, iniciou e foi se

desenvolvendo o estudo das redes neurais artificias, sendo o campo de inteligência artificial que simula atividades cerebrais dos neurônios utilizando modelos matemáticos dando origem as redes neurais.

As redes neurais se constituem de nós conectados por ligações direcionadas, uma ligação da entrada para uma função de ativação e gerar uma saída, cada ligação contendo um peso associado a ele, determinando assim o sinal da conexão entre os nós da rede neural.

Figura 3 - Neurônio artificial

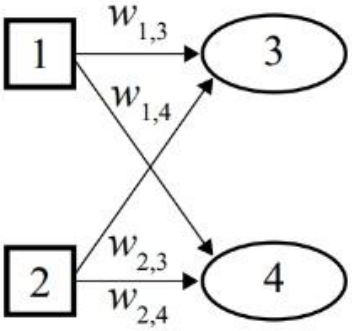


Fonte: SILVA, Fabrício et. al., **Inteligência artificial,** 2019. p.113.

Autor SILVA, Fabrício et. al. (2019), diz que as redes neurais artificiais possuem duas estruturas principais que são redes neurais de camada única e redes neurais de multicamada.

Redes neurais de camada única possuem todas as entradas conectadas em somente uma saída.

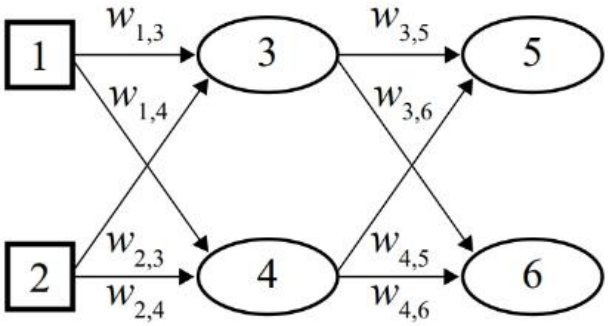
Figura 4 – Rede de camada única com duas entradas



Fonte: SILVA, Fabrício et. al., **Inteligência artificial,** 2019. p.115.

Nas redes neurais de multicamadas elas são uma rede composta de duas ou mais camadas de neurônios ligadas entre si por conexões com pesos diferentes.

Figura 5 – Rede neural artificial de multicamada

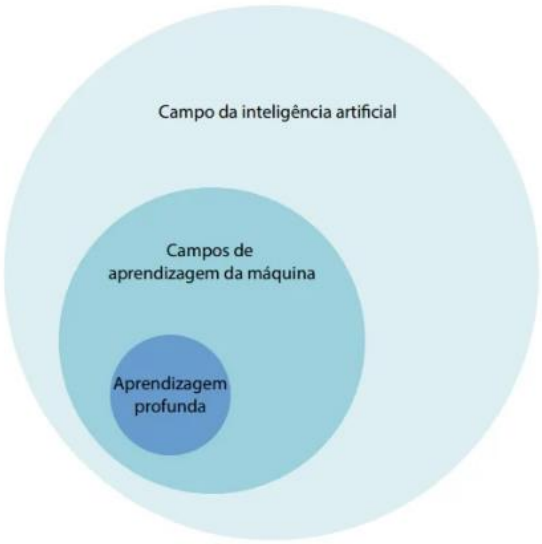


Fonte: SILVA, Fabrício et. al., **Inteligência artificial,** 2019. p.116.

2.4.3 Deep Learning

A definição de Deep Learning, ou Aprendizagem Profunda, para SILVA, Fabrício et. al. (2019) é o campo da codificação algorítmica que faz uso das técnicas de redes neurais artificiais, que imitam o cérebro. É uma técnica usada em machine learning muito precisa para a aprendizagem de máquina. Os algoritmos da rede neural são complexos construídos a partir de um empilhamento de diversas camadas de neurônios, sendo alimentados por grandes quantidades de dados usando-os para que ocorra o aprendizado de padrões para serem usados em projetos que envolvam, por exemplo, reconhecimento de voz, análise de comportamento e principalmente processamento de imagens.

Figura 6 – Campos da inteligência artificial



Fonte: SILVA, Fabrício et. al., **Inteligência artificial,** 2019. p.18.

3 MATERIAIS E MÉTODOS

Este capítulo apresenta as ferramentas utilizadas para elaborar o experimento de simulação robótica em ambiente digital e apresentar as metodologias utilizadas na pesquisa.

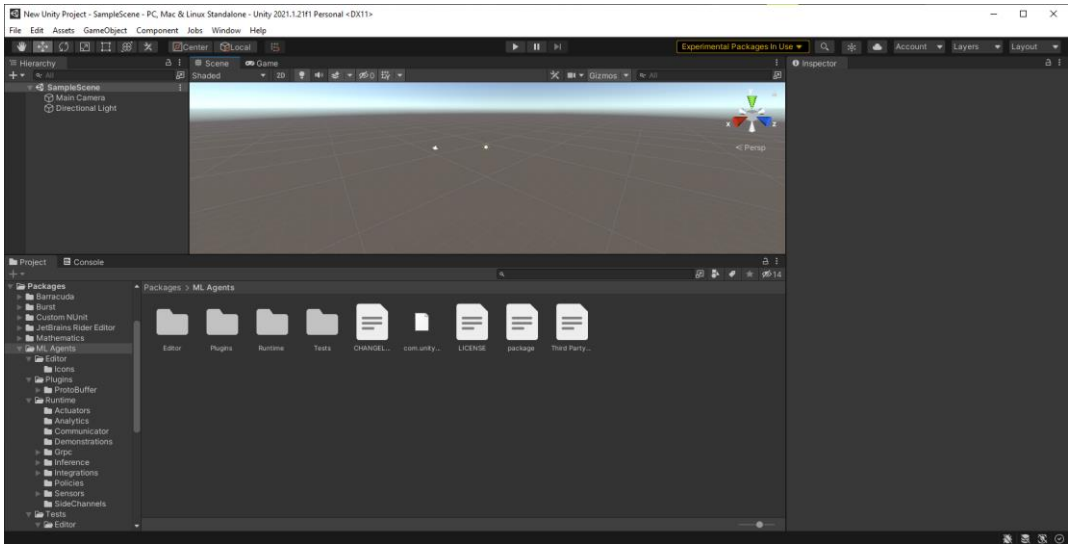
3.1 MATERIAIS

3.1.1 Unity

Unity é uma *game engine,* que pertence à *Unity Technologies*, que permite a criação de jogos em 2D e 3D. *Unity* também possui visualização imersiva em 3D para produtos automotivos, transporte e fabricação podendo utilizar Realidade Virtual para uma maior imersão. Possui funcionalidades para criar cinema, animação e cinemática utilizando da renderização em tempo real para criar objetos durante transmissões ao vivo.

Unity (Unity Technologies, 2021) utiliza a linguagem de programação C# e é orientado a objeto. Pode produzir jogos para umas diversas plataformas, por exemplo, para dispositivos móveis, navegadores web, computadores e *consoles*. Suporta plataformas que incluem Linux, Windows, Android e IOS.

Figura 7 – Interface Unity



Fonte: Próprio autor.

3.1.2 Unity ML-Agents Toolkit

É um projeto de código aberto que permite que jogos e simulações sirvam como ambientes para o treinamento de agentes inteligentes através de implementações com base no PyTorch de algoritmos que permitem o treinamento facilitado de agentes inteligentes para jogos 2D e 3D. Possibilita usar a API Python de forma simples para treinar Agentes usando aprendizado por reforço, aprendizado por imitação, neuroevolução ou quaisquer outros métodos. Esses agentes treinados podem ser usados para vários fins. O ML-Agents Toolkit é mutuamente benéfico para desenvolvedores de jogos e pesquisadores de IA, pois fornece uma plataforma central onde os avanços em IA podem ser avaliados nos ambientes ricos da Unity *(Unity Technologies, 2021)*.

O ML-Agents (*Unity Technologies, 2021*) utiliza uma técnica de aprendizado por reforço chamada *Proximal Policy Optimization* (PPO). O PPO usa uma rede neural para aproximar a função ideal que mapeia as observações de um agente para melhorar a ação que esse agente pode realizar em um determinado estado. Esse algoritmo do ML-Agents PPO é implementado no *TenshorFlow(TENSORFLOW 2021)* e executado em um processo em Python separado, comunicando-se com o Unity através de uma execução de soquete.

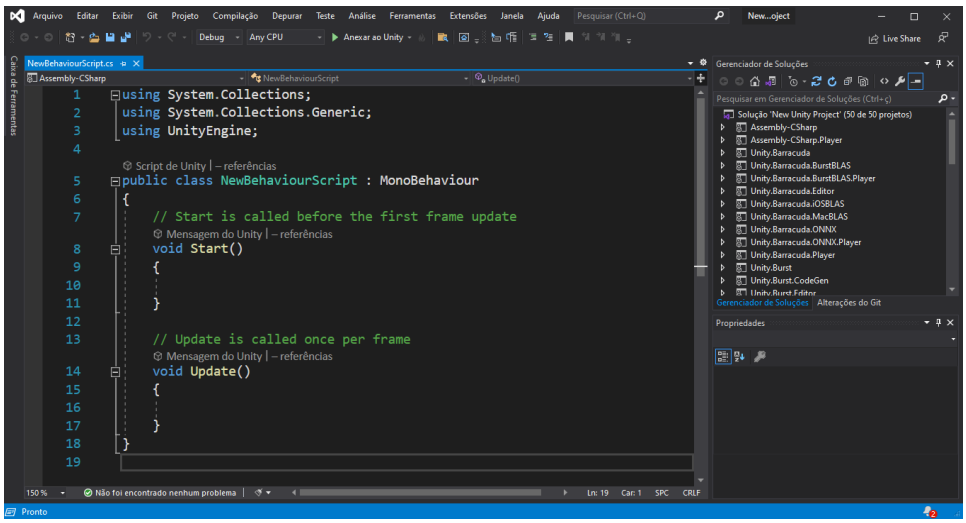
O Unity ML-Agents Toolkit (*Unity Technologies, 2021*), neste trabalho será utilizado para realizar o treinamento da inteligência artificial do robô utilizando da técnica de aprendizado por reforço.

3.1.3 Microsoft Visual Studio

É um ambiente de desenvolvimento integrado da Microsoft, para várias linguagens de programação e especialmente o C# que é a linguagem de programação do Unity.

> O Visual Studio oferece uma experiência superior de depuração para o mecanismo de jogos do Unity. Identifique problemas rapidamente ao depurar seus jogos do Unity no Visual Studio — Defina pontos de interrupção e avalie variáveis e expressões complexas. (MICROSOFT, 2019)

Figura 8 – Interface Microsoft Visual Studio



Fonte: Próprio autor.

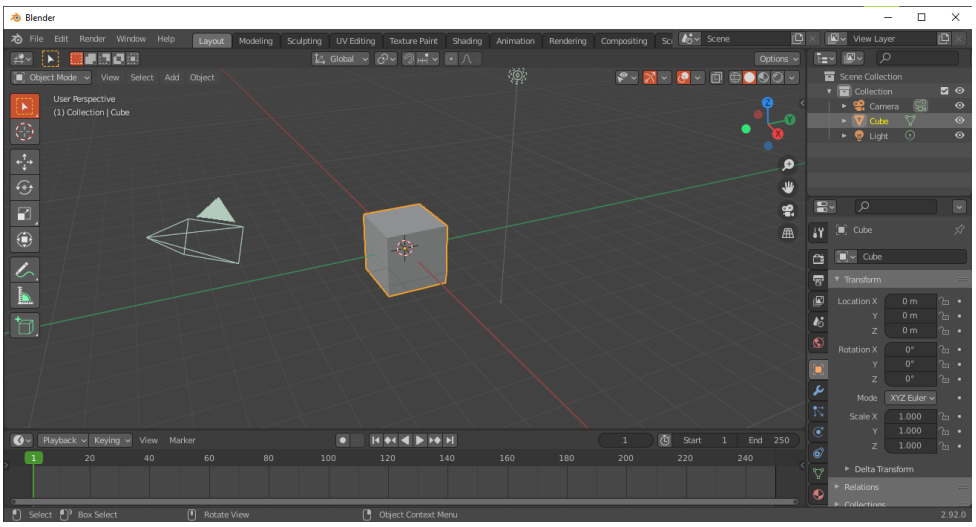
3.1.4 Blender

O *Blender (BLENDER FOUNDATION, 2021)* será utilizado para modelar o robô e o ambiente. Trata-se de um software gráfico de computador gratuito e de código aberto usado para criar filmes animados, efeitos visuais, arte, modelos 3D, gráficos em movimento, realidade virtual e jogos de computador. Os recursos do *Blender* incluem modelagem 3D, texturização, edição de gráficos, simulação de fluido e fumaça, simulação de partícula, escultura, animação, renderização, gráficos em movimento entre outros.

O *software* já é utilizado pela *National Aeronautics and Space Administration* NASA em muitos de seus modelos 3D disponíveis publicamente, como por exemplo na nave espacial *Cassini*[1].

[1] Nave espacial *CASSINI*. Disponível em: <https://nasa3d.arc.nasa.gov/detail/jpl-vtad-cassini> Acesso em: 23 de Abril de 2021.

Figura 9 – Interface Blender

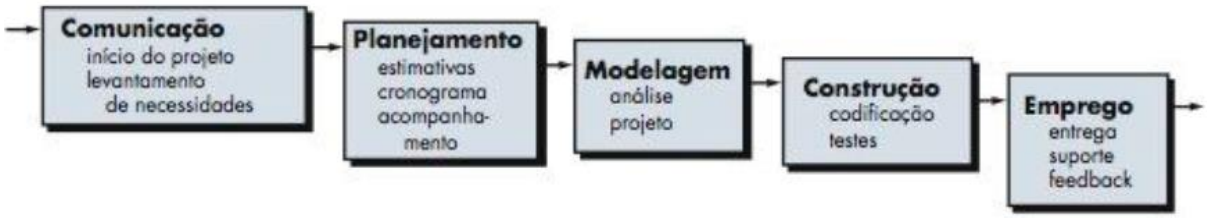


Fonte: Próprio autor.

3.2 MÉTODOS

Como não haveria mudança dos requisitos, o projeto foi realizado utilizando a metodologia em cascata segundo colocado por Pressman (2011), em que são realizadas fases consecutivas de desenvolvimento do software conforme ilustrado na figura abaixo:

Figura 10 - Metodologia de desenvolvimento em Cascata



Fonte: PRESSMAN., 2011, p.59.

O presente trabalho apresenta o início do projeto, explicitando os detalhes do experimento físico realizado por Pontuschka (2016) e as especificações que o ambiente virtual e o modelo virtual dos robôs deveriam possuir.

Este trabalho não trata propriamente do desenvolvimento de um software comercial, mas uma atividade de pesquisa em Machine learning, portanto não utilizaremos a forma tradicional de projeto de software comercial que faz uso de

documentação UML extensiva. Optamos por utilizar os princípios do desenvolvimento de projetos, em especial elementos do PMBOK 6ª edição, ou Guia de boas práticas em gerenciamento de projetos.

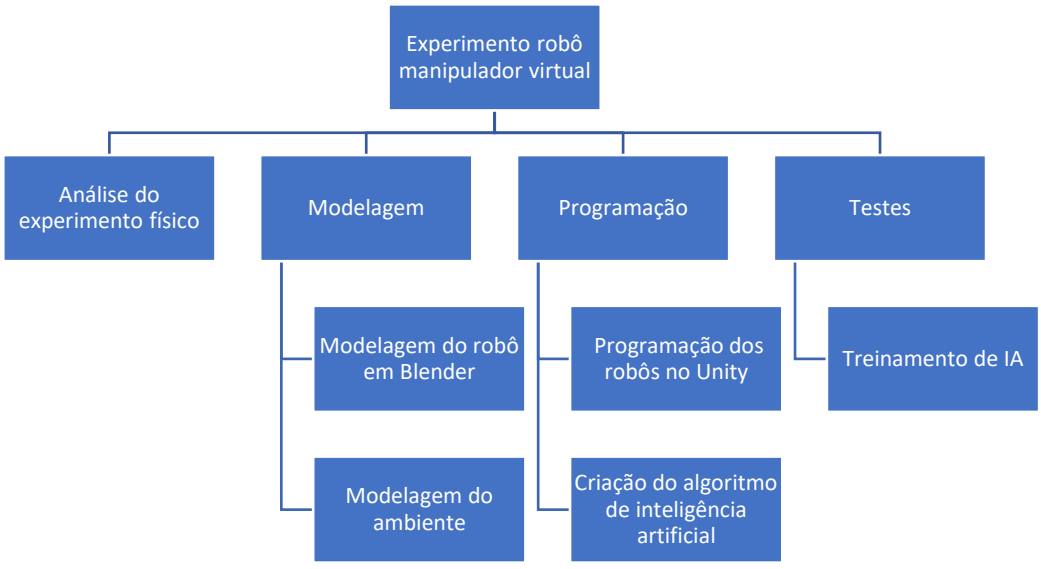
Um dos primeiros momentos da fase inicial de um projeto é utilizar a ferramenta EAP ou estrutura analítica do projeto), que se baseia em dividir o projeto de forma hierárquica e em partes menores, a fim de facilitar o desenvolvimento das entregas (figura 11).

Inicialmente fizemos o estudo do experimento inicial físico realizado por Pontuschka et. al. (2016). Em seguida foi feito o modelo 3D do robô e do ambiente no *software Blender*, logo em seguida foi montado e configurado o ambiente de testes dentro da *game engine* juntamente com o robô.

O robô foi treinado para chegar até ponto indicado. O processo de aprendizagem foi realizado com com *Machine Learning* utilizando a técnica de aprendizagem por reforço. O robô terá a habilidade de poder se movimentar livremente pelo ambiente até que possa concluir seu objetivo, tocar no outro robô, da maneira mais econômica possível.

Para que chegue da maneira mais eficiente possível foi implementado um limitador que faz com que tenha pouco recurso para se movimentar igual ao modelo real que possui um recurso de oxigênio limitado para sua movimentação.

Figura 11- EAP - Estrutura Analítica do Projeto

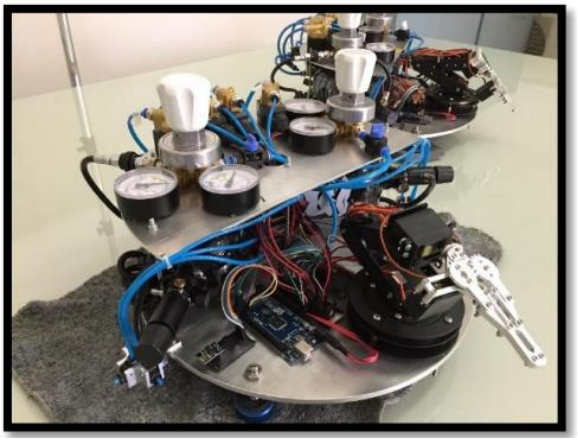


Fonte: Próprio autor.

3.3 ANÁLISE DO EXPERIMENTO FÍSICO - REQUISITOS

Segundo Pontuschka et. al. (2016), o manipulador robô (figura 12) a ser simulado possui um braço robótico de 6 graus de liberdade que é fixado em uma base deslizante sobre uma mesa de vidro balanceada. Sua base possui sapatas que permitem que fique suspensa em uma camada de ar comprimido. Isto permite que a base se movimente e rotacione nos eixos x e y em um ambiente de atrito extremamente baixo.

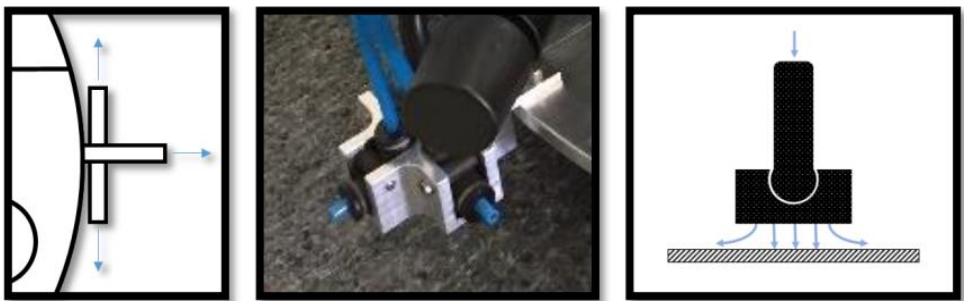
Figura 12 - Manipulador robô



Fonte: PONTUSCHKA et al., 2016, p. 2.

Para que seja possível a movimentação e rotação do robô, existem dois atuadores a ar comprimido, cada um com três saídas de ar, que são usadas para controlar sua movimentação além das sapatas de sustentação (figura 13).

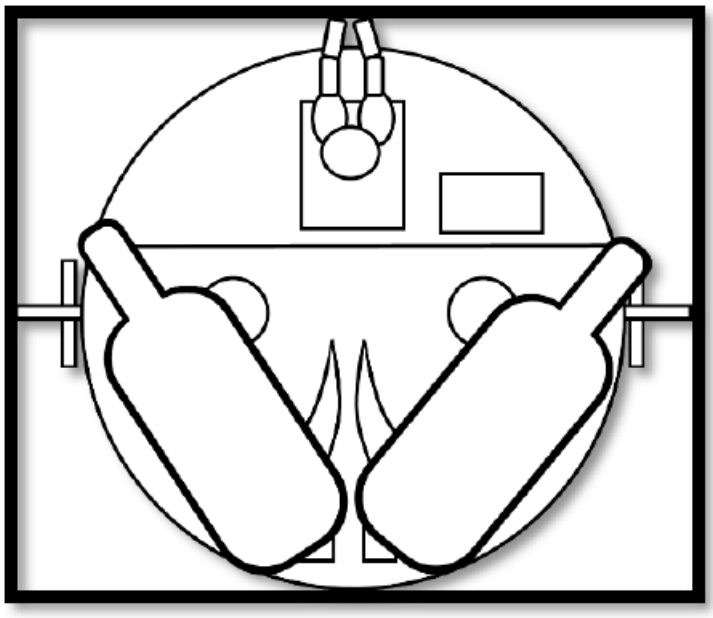
Figura 13 - Bicos de jatos de ar para movimentação do robô e sapatas de sustentação



Fonte: PONTUSCHKA et al., 2016, p.3.

O robô físico possui dois cilindros de ar comprimido, um com a responsabilidade de manter o robô suspenso por suas sapatas e outro com a função de fornecer ar para os atuadores (figura 14). O robô recebe comandos de tempo e direção para o disparo dos jatos dos atuadores.

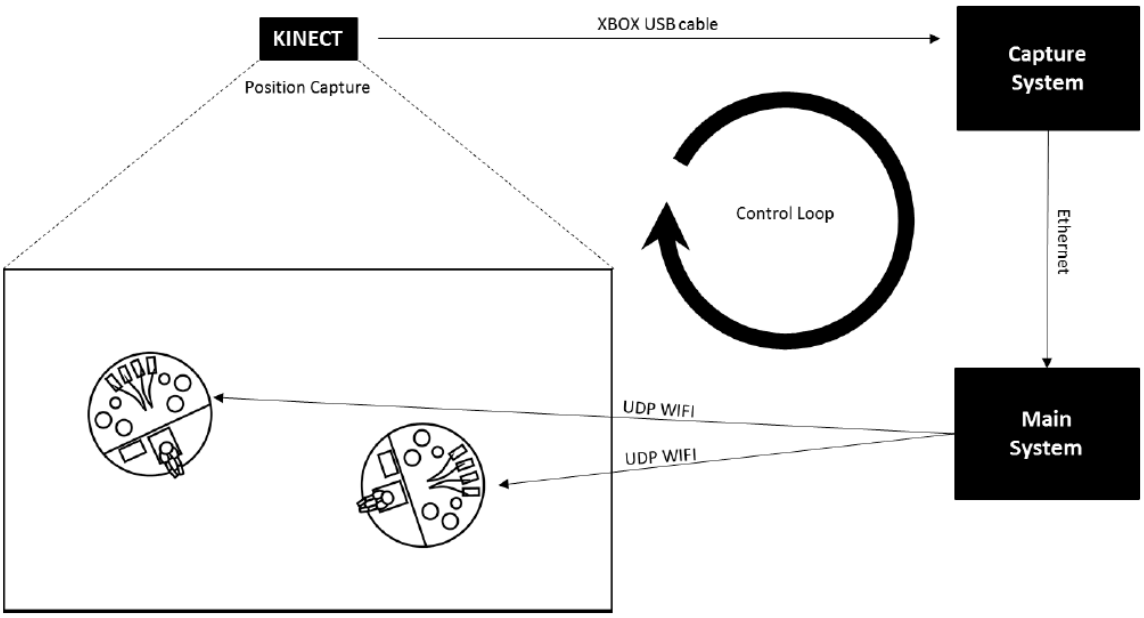
Figura 14 - Localização dos cilindros de ar comprimido no robô



Fonte: PONTUSCHKA et al., 2016, p.3.

No experimento de Pontuschka et. al. (2016) não existe nenhum sensor na estrutura dos robôs. Mas existe um sistema de captura e processamento de imagens baseado no *Kinect* da *Microsoft*, normalmente utilizado em videogames *Xbox*. A captura das imagens do *Kinect* é enviada para um computador que registra a posição, direção e velocidade de cada um dos robôs a partir de marcações coloridas no corpo dos robôs que permitem identificar a direção e posição em que cada um se encontra em um determinado momento. As informações obtidas são enviadas para um segundo computador utilizado para testar a lógica que fará os cálculos dos comandos necessários para que os robôs se aproximem e façam o acoplamento. Os comandos calculados são enviados via protocolo UDP via *WIFI* para os robôs que, por sua vez, executam os comandos (figura 15).

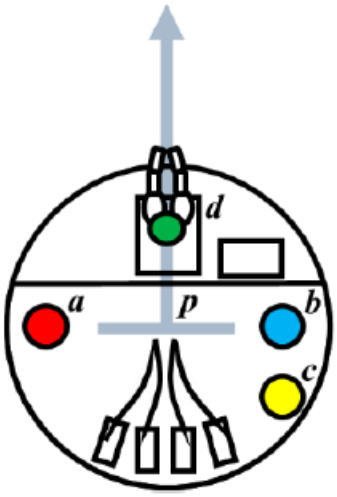
Figura 15 - Esquema do controle dos robôs no experimento físico



Fonte: PONTUSCHKA et al., 2016, p.4.

O sistema de captura utiliza as marcações coloridas no corpo de cada manipulador robô para poder verificar a posição, a velocidade a direção de cada robô e o posicionamento de seus braços robóticos.

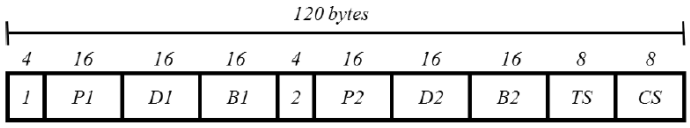
Figura 16 - Marcações coloridas para identificação da posição e posicionamento do braço do robô.



Fonte: PONTUSCHKA et al., 2016, p.4.

O sistema de captura funciona da seguinte maneira: o *Kinect* identifica as marcações coloridas posicionadas na mesa, nos robôs e nos braços robóticos. Para calcular a posição e direção da base dos robôs em relação à mesa, a posição ***p*** do robô é o ponto médio entre ***a*** e ***b.*** Para determinar a direção do robô utiliza o vetor $\vec{u}$ = (c, b). Para identificar a posição do braço robótico identifica a posição d. Os dados são enviados em pacotes de dados de posicionamento para o sistema de controle via conexão ethernet.

Figura 17 - Pacote de dados gerados pelo sistema da captura



parte	*descrição*
1	Indicação de leitura para o robô 1 (int)
2	Posição do robô 1 (double, double)
3	Direção do robô 1 (double, double)
4	Posição do braço do robô 1 (double, double)
5	Indicação de leitura para o robô 2 (int)
6	Posição do robô 2 (double, double)
7	Direção do robô 2 (double, double)
8	Posição do braço do robô 2 (double, double)
9	Time stamp
10	Checksum

Fonte: PONTUSCHKA et al., 2016, p.5.

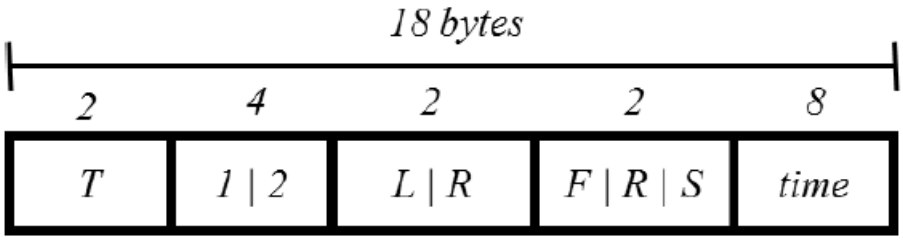
Com estes dados é possível implementar algoritmos que possam executar o acoplamento dos dois robôs, uma vez que é possível saber qual a velocidade, a posição e distância relativas de ambos.

O Sistema de comando realiza as seguintes funções:

1) calcula a velocidade e direção do movimento do robô
2) calcula a distância entre os robôs
3) calcula o alinhamento das direções dos braços dos robôs
4) Aciona os jatos de ar dos atuadores de ar comprimido

Os comandos para o acionamento dos jatos de ar são enviados por meio de pacotes de dados de 18 *bytes*.

Figura 18 - Pacote de dados para acionamento de jato de ar

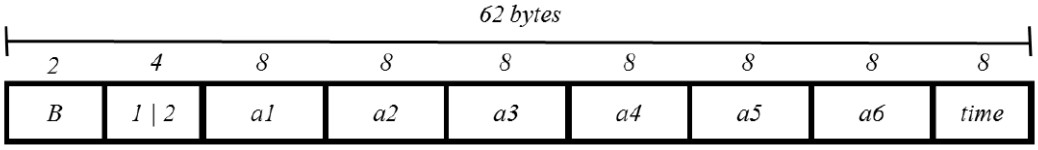


Fonte: PONTUSCHKA et al., 2016, p.5.

Estes pacotes identificam T para o comando de acionamento do jato de ar, 1 ou 2 para identificação do robô, L ou R para identificar qual dos conjuntos de atuadores devem ser usados da esquerda ou direita do robô, F para frente, R para trás e S para lateral. Informa também o tempo de acionamento.

O movimento dos braços robóticos é realizado informando o ângulo de cada um dos 6 graus de liberdade do braço além do tempo desejado para o movimento e é enviado por meio do pacote de dados na figura 8 enviados por protocolo UDP via *WIFI*.

Figura 19 - Pacote de dados para a movimentação do braço robótico do robô



Fonte: PONTUSCHKA et al., 2016, p.6.

Com todos estes comandos de atuação é possível posicionar e girar os robôs de modo a conseguir acoplá-los.

O robô possui uma placa Arduino Mega, que está conectada a um módulo de servo motores. O processador Arduino também controla as válvulas de acionamento eletrônico para os jatos de ar dos atuadores (figura 9). Os comandos, assim que recebidos e identificados, são executados.

4 DESENVOLVIMENTO

Este capítulo apresenta o procedimento de instalação e configuração dos *softwares* utilizados, utilização do *software* de modelagem, exportação e importação dentro das *game engine,* além dos códigos que constituem o ambiente de treinamento do agente por meio da inteligência artificial para que realize algumas ações básicas semelhantes ao robô físico existente.

4.1 Instalação do Blender

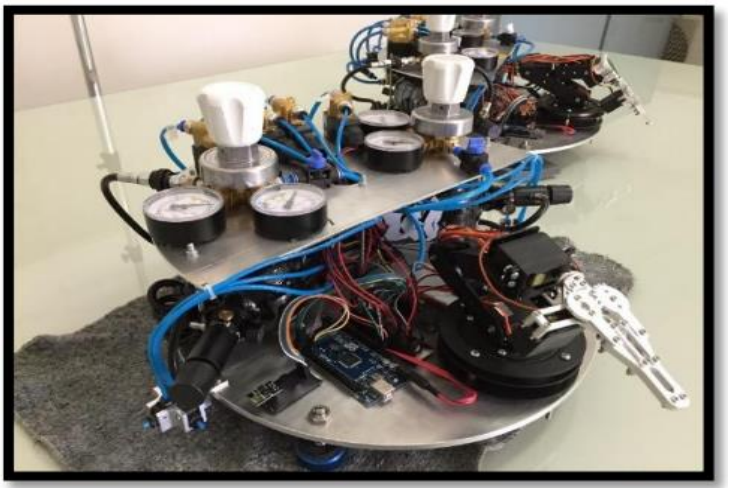
Para realizar a instalação do *Blender (BLENDER FOUNDATION 2021)* basta seguir as seguintes etapas:

- Acessar o site https://www.blender.org/download/ e realizar o *download*.
- Após o *download* concluído, realizar a instalação através da execução do arquivo baixado.

4.2 Modelagem do robô no Blender

Antes de começar a simulação na game engine, é necessário modelar o robô no *Blender (BLENDER FOUNDATION 2021)* com base nas imagens do modelo real.

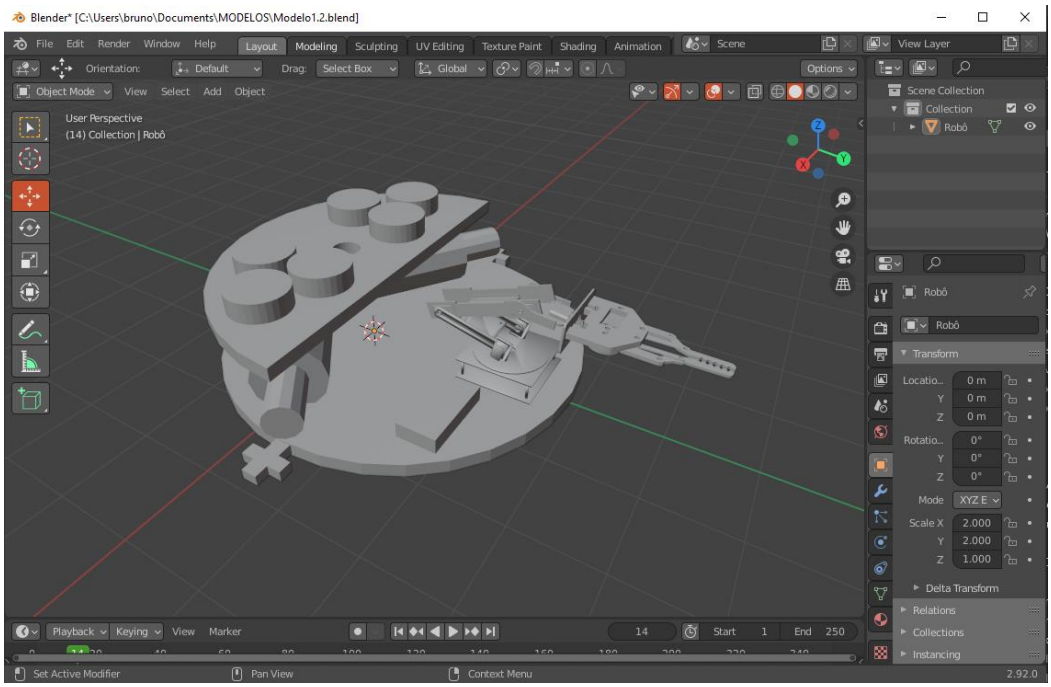
Figura 20 – Modelo real do manipulador robô



Fonte: PONTUSCHKA et al., 2016, p. 2.

Com base essa foto e em outras já vistas anteriormente no trabalho, cheguei ao seguinte modelo:

Figura 21 – Manipulador robô modelado no Blender

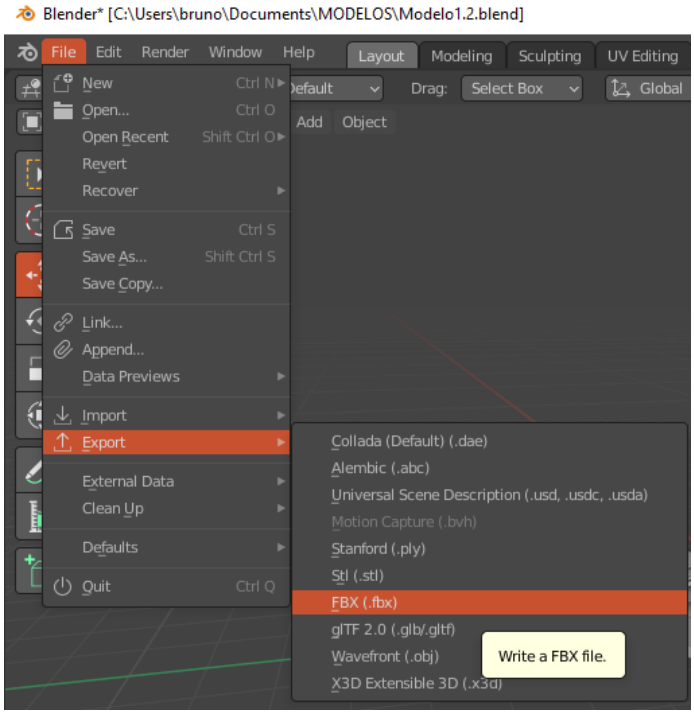


Fonte: Próprio autor.

O modelo do braço articulado, foi feito por outros membros do grupo Simulação Robótica.

Após a modelagem pronta é necessário exporta-lo o modelo, como na imagem a seguir e salvar em uma pasta.

Figura 22 – Exportando modelo



Fonte: Próprio autor.

4.2 IMPLEMENTANDO MACHINE LEARNING COM O UNITY

Para utilizar aprendizado por reforço para treinamento de agentes inteligentes, pode-se utilizar o *ML-Agents Toolkit (UNITY TECHNOLOGIES 2021)*, que funciona na *game engine Unity*, e que é como descrito anteriormente, um projeto de código aberto que permite que jogos possam ser ambientes de treinamento para agentes inteligentes.

4.3 INSTALANDO UNITY E PACOTES AUXILIARES

4.3.1 Instalando Unity

Para realizar a instalação do *Unity (UNITY TECHNOLOGIES 2021)* basta seguir as seguintes etapas:

- Acessar o site https://unity3d.com/get-unity/download e realizar o *download* de uma versão acima da 2019.4.

- Após o *download* concluído, realizar a instalação através da execução do arquivo baixado.

4.3.2 Instalando Python

Para realizar a instalação do *Python (PYTHON SOFTWARE FOUNDATION 2021)* basta seguir as seguintes etapas:

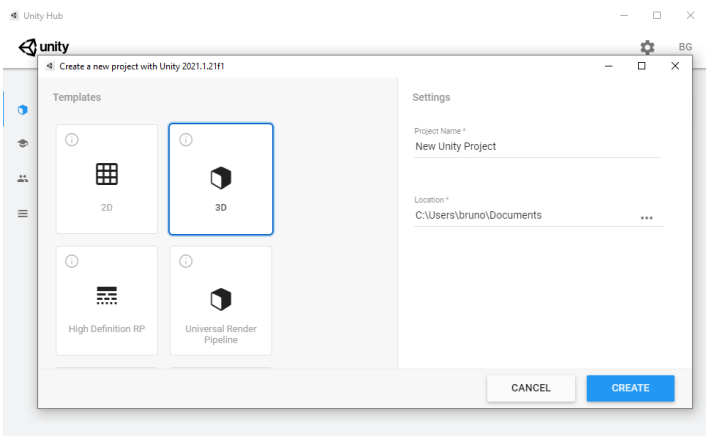
- Acessar o site https://www.python.org/downloads/ e realizar o *download* de uma versão acima da 3.6.
- Após o *download* concluído, realizar a instalação através da execução do arquivo baixado.

4.3.3 Instalando Virtual Enviroment

O *Virtual Enviroment* é um ambiente cuja função é manter nossos projetos separados uns dos outros para que seja possível ter vários projetos usando versões diferentes de pacotes *Python (PYTHON SOFTWARE FOUNDATION 2021)* sem que um atrapalhe o outro.

Antes de instalar o *Virtual Enviroment é* necessário criar um projeto no *Unity (UNITY TECHNOLOGIES 2021)* primeiro e definir a pasta, nome e escolher o *template 3D*.

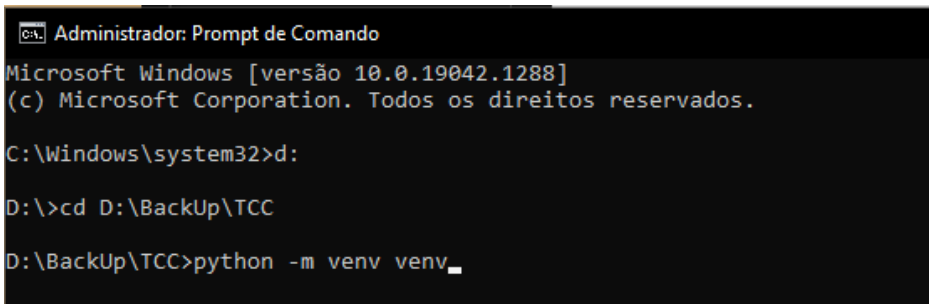
Figura 23 – Criando projeto Unity



Fonte: Próprio autor.

Com o endereço da pasta abra o CMD e acesse ela. Digite *python -m venv venv* e execute. Esse comando irá criar o *virtual environment* dentro da pasta chamada *venv*.

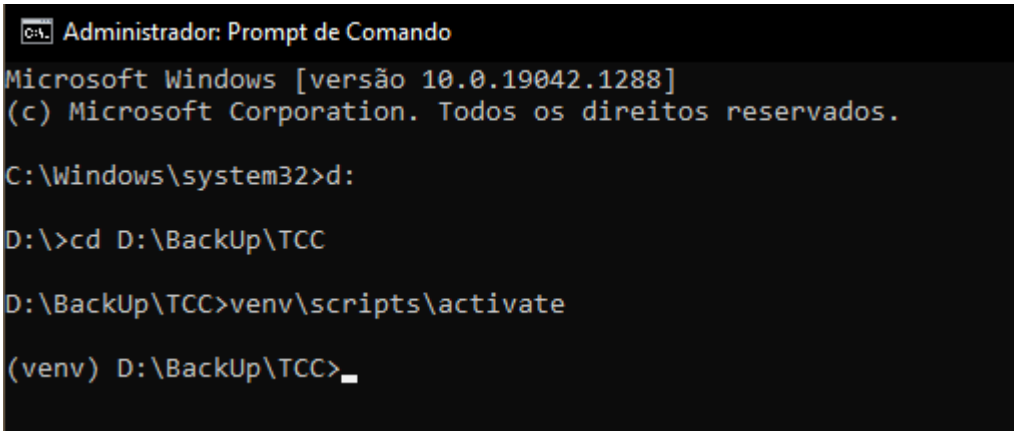
Figura 24 – Comando de execução venv



Fonte: Próprio autor.

O comando irá criar uma pasta com vários scripts *python* e um deles, o *Activate* é o que irá ativar o *Virtual Environment*. Então para ativá-lo basta digitar *venv\scripts\activate* e quando ativar, qualquer alteração que for feita nesse projeto, não irá impactar em outros.

Figura 25 – Inicializando o Virtual Environment



Fonte: Próprio autor.

Ainda dentro do *Virtual Environment*, instalamos o *Pytorch (PYTORCH 2021)*. Ele será o responsável por processar a linguagem da rede neural que irá realizar o treinamento do nosso robô. Para instalar é o seguinte comando:

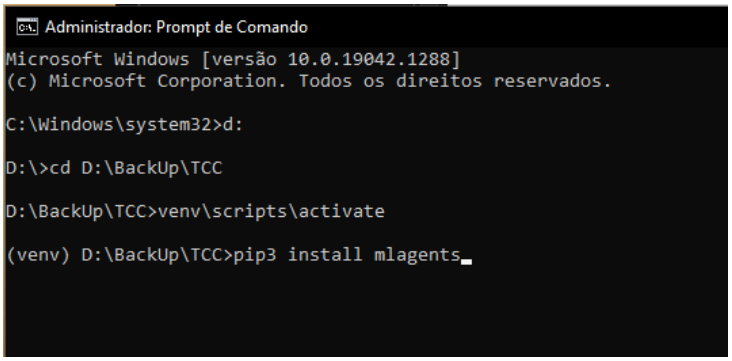
pip3 install torch~=1.7.1 -f https://download.pytorch.org/whl/torch_stable.html.

Figura 26 – Instalando pytorch

Fonte: Próprio autor.

Depois instalamos o *ml-agents*: *pip install mlagens.*

Figura 27 – Instalando ml-agents



Fonte: Próprio autor.

4.3.4 Download ML-Agents Toolkit do repositório

Para realizar a instalação do *ML-Agents Toolkit (UNITY TECHNOLOGIES 2021)* basta seguir as seguintes etapas:

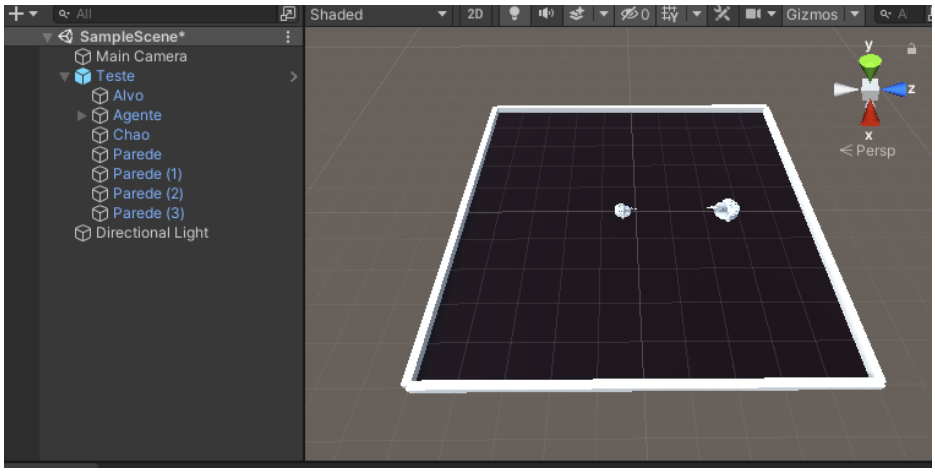
- Acessar o site https://github.com/Unity-Technologies/ml-agents e clicar em *Code* depois *Download zip*.

4.4 CONFIGURANDO AMBIENTE E PROGRAMANDO NO UNITY

4.4.1 Configurando o ambiente UNITY

O cenário é constituído de um ator, que é como a *Unity* (*UNITY TECHNOLOGIES 2021*), criamos o chão clicando no menu lateral *Hierarchy* com o botão direito, selecionando *3D object* e selecionado *plane* com dimensões de escala x:10, y:1, z:10. Para as paredes repete o processo selecionando *cube,* com escala, x:100, y:10, z:2. O ambiente montado ficou como mostra a figura a seguir.

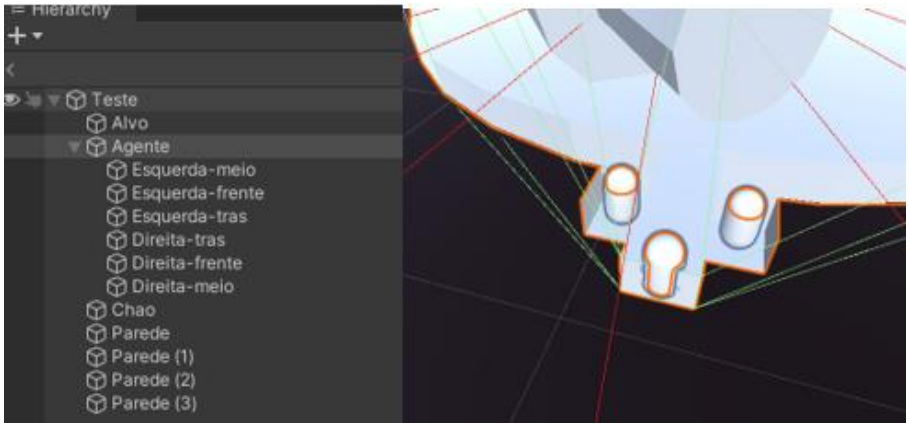
Figura 28 – Ambiente de desenvolvimento no Unity3D



Fonte: Próprio autor

Como passo seguinte, para simular o robô físico real, é necessário que sejam simuladas as saídas de ar comprimido. Para isso criamos seis cilindros, e posicionamos cada um sendo três de cada lado como na imagem a seguir e renomeamos cada cilindro de acordo com a posição.

Figura 29 - Propulsores



Fonte: Próprio autor.

Próximo passo é instalar o *ml-agents* no nosso projeto. Por meio da opção do menu *window* e *package manager*, irá abrir uma janela. Clicar no símbolo “+” e selecionar add *package from disk* e, depois, selecionar o diretório e o arquivo que fizemos download do *github* e instalamos deste modo, está pronto para usar.

4.4.2 Criação de scripts

Criamos três *scripts* c#, um chamado Agente, para escrevermos os códigos que caracterizam o treinamento e dita quais ações o robô deverá aprender, condições para recompensa e para punição.

O segundo *script* é o Alvo, para identificação e quando o agente tocar nele, o agente será premiado com uma pontuação positiva e o terceiro é o Parede, para podermos identificar qualquer parede ou obstáculo durante a colisão e também aplicar uma punição ao agente.

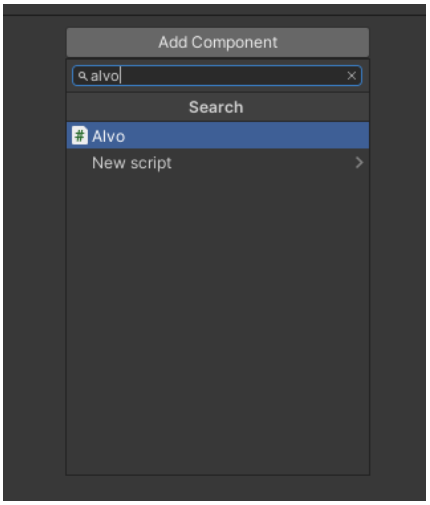
Figura 30 – Scripts criados

Fonte: Próprio autor

Os *scripts* Alvo e Parede não terão códigos nele, eles serão anexados com os respectivos objetos 3D no cenário e serão através deles que iremos identificar se ouve ou não a colisão apara a aplicação das recompensas.

Para anexarmos cada *script* ao seu respectivo objeto3D, o objeto 3D deve ser selecionado no cenário e na aba inspetor, *add component* deve ser selecionado, escrevendo o nome do *script*.

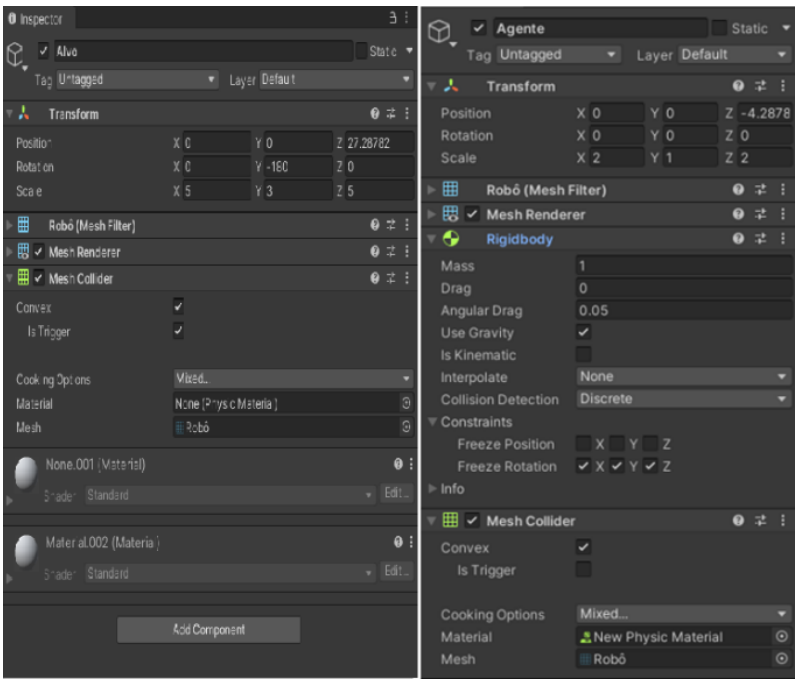
Figura 31 – Anexando script



Fonte: Próprio autor

Da mesma maneira que adicionamos o componente anterior, no Alvo devemos adicionar um *mesh collider*, e no Agente adicionamos *Rigidbody* e *Mesh Collider* e ficará como na imagem a seguir.

Figura 32 - Anexando script alvo e agente



Fonte: Próprio autor

Usando o mesmo método, pegamos os cilindros que simulam o propulsor e adicionamos a eles o componente *Fixed Joint*, para que não se separem do robô durante a movimentação.

4.4.3 Escrevendo os scripts

Abrindo o script Agente, adicionamos alguns pacotes para uso do ml-agents que são:

```
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Sensors;
using Unity.MLAgents.Actuators;
```

Depois removemos os métodos padrões e modificamos de *Monobehavior* para *Agent*, em seguida no código inserimos cada um dos propulsores do robô, uma posição inicial do agente, uma variável para a velocidade do agente e um campo para anexarmos o alvo.

```
public class Agente : Agent
{
    public Rigidbody direitaMeio;
    public Rigidbody direitaFrent;
    public Rigidbody direitaTras;
```

```
    public Rigidbody esquerdaMeio;
    public Rigidbody esquerdaFrent;
    public Rigidbody esquerdaTras;

    private Vector3 posicaoInicial;

    float velocidade;
    [SerializeField] private Transform alvo;
```

Em seguida criamos o método *Initialize* para inicializar os componentes anteriores.

```
    public override void Initialize()
    {

        direitaMeio = transform.GetComponent<Rigidbody>();
        direitaFrent = transform.GetComponent<Rigidbody>();
        direitaTras = transform.GetComponent<Rigidbody>();
        esquerdaMeio = transform.GetComponent<Rigidbody>();
        esquerdaFrent = transform.GetComponent<Rigidbody>();
        esquerdaTras = transform.GetComponent<Rigidbody>();
        }
```

Criamos o *OnEpisodeBegin*, o treinamento de aprendizagem por reforço funciona por episódios, cada vez que um episódio iniciar, ele será chamado.

```
    public override void OnEpisodeBegin()
    {
        velocidade = 1000f;
        transform.localPosition = Vector3.zero;
        alvo.transform.localPosition = new Vector3(0,0,20f);
     }
```

Adicionamos *CollectedObservations*, que envia informações parciais que descrevem o ambiente em que o agente está, nesse caso, a posição no ambiente, que é dado através do *Vector3,* que são as posições nos eixos x, y e z.

```
   public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(transform.localPosition);

            }
```

É necessário criarmos para cada uma dessas ações uma classe para acionar cada um dos propulsores, indicar qual direção ele deve ser acionado e velocidade.

```
public void moverEsquerda()
    {
             direitaMeio.AddForce(Vector3.left * (velocidade* 2) * Time.deltaTime);

    }

    public void moverDireita()
```

```
    {

        esquerdaMeio.AddForce(Vector3.right * (velocidade*2) * Time.deltaTime);
    }
    public void moverFrente()
    {

        direitaTras.AddForce(Vector3.forward * velocidade* Time.deltaTime);
        esquerdaTras.AddForce(Vector3.forward * velocidade* Time.deltaTime);

    }
    public void moverTras()
    {
        esquerdaFrent.AddForce(Vector3.back * velocidade* Time.deltaTime);
        direitaFrent.AddForce(Vector3.back * velocidade* Time.deltaTime);


            }
```

O próximo é o *OnActionReceived*, onde serão criadas as ações que o robô poderá realizar. No nosso caso, o robô poderá se impulsionar para frente, para trás, para a direita e para a esquerda usando as classes criadas anteriormente. Cada ação é passada para ela na forma de um vetor (*array)*.

```
public override void OnActionReceived(ActionBuffers actions)
    {
        var frente = actions.DiscreteActions[0];
        var paraTras = actions.DiscreteActions[1];
        var direita = actions.DiscreteActions[2];
        var esquerda = actions.DiscreteActions[3]

        if (frente > 0)
        {
            moverFrente();

        }
        if (paraTras > 0)
        {
            moverTras();
        }
        if (direita > 0)
        {

            moverDireita();
        }
        if (esquerda > 0)
        {
            moverEsquerda();
        }
        if (transform.localPosition.y < -1)
        {
            EndEpisode();
        }
            }
```

Por último *OnTriggerEnter*, onde verificamos a colisão entre robô e o objetivo ou as paredes, e através da função *AddReward*, atribuímos uma recompensa positiva para caso ele encoste no alvo, concluindo o objetivo com êxito, ou uma recompensa negativa quando encostar em uma parede. Quando uma das duas opções ocorrerem para que a recompensa seja adicionada e influencie na inteligência artificial chamamos o método *EndEpisode,* para que o episódio termine e seja adicionada a recompensa. Também é feita uma verificação se o robô saiu do cenário de treino. Neste caso o episódio é encerrado.

```
private void OnTriggerEnter(Collider other)
    {


        if (other.TryGetComponent<Alvo>(out Alvo alvo))
        {
            AddReward(+1f);
            EndEpisode();


        }

        if (other.TryGetComponent<Parede>(out Parede parede))
        {
            AddReward(-1f);
            EndEpisode();


        }

            }
```

4.4.4 Configuração dos componentes do robô agente

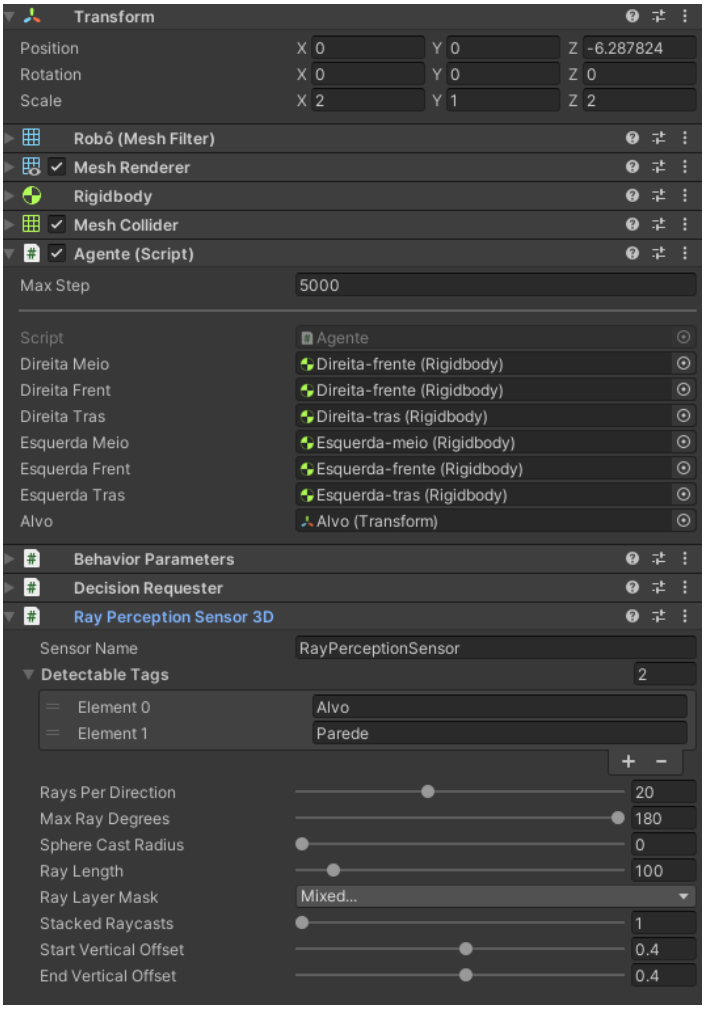
Para que o robô se movimente apropriadamente, no componente Agente, devemos configurar o campo *Max Step*, que definirá o quantos “passos” o robô estará limitado durante cada episódio de treinamento, isso também serve para simular coisas como a limitação do ar comprimido que o robô real teria. Nesse caso, inicialmente colocamos mil para testes iniciais.

No agente, adicionamos os componentes *Behavior Parameters*, que nos possibilita definir o comportamento do nosso agente e será onde anexaremos o arquivo da inteligência artificial quando ela estiver treinada.

Adicionar o componente *Dicision Request*, que solicita decisões que o agente irá tomar e possibilitar a ele realizar ações entre elas.

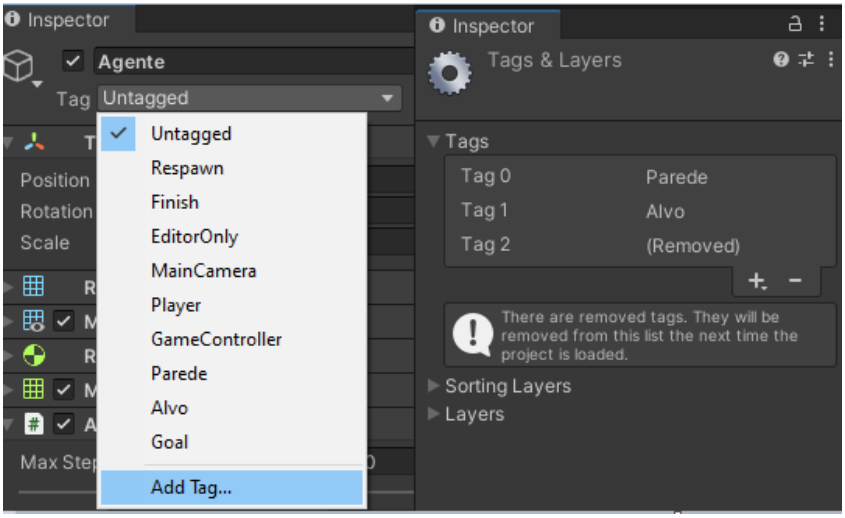
Para auxiliar no treinamento da inteligência artificial, e simular sensores reais, adicionamos o componente *Ray Perception Sensor 3D*, as configurações dos componentes são mostradas na figura 33. Para que o *Ray perception Sensor 3D* funcione devemos utilizar o sistema de *tag* do *Unity (UNITY TECHNOLOGIES 2021)*, deveremos criar uma para o alvo e uma para as paredes e atribuir a cada um, como mostra a figura 34.

Figura 33 - Configurações



Fonte: Próprio autor.

Figura 34 – Adicionando tags



Fonte: Próprio autor.

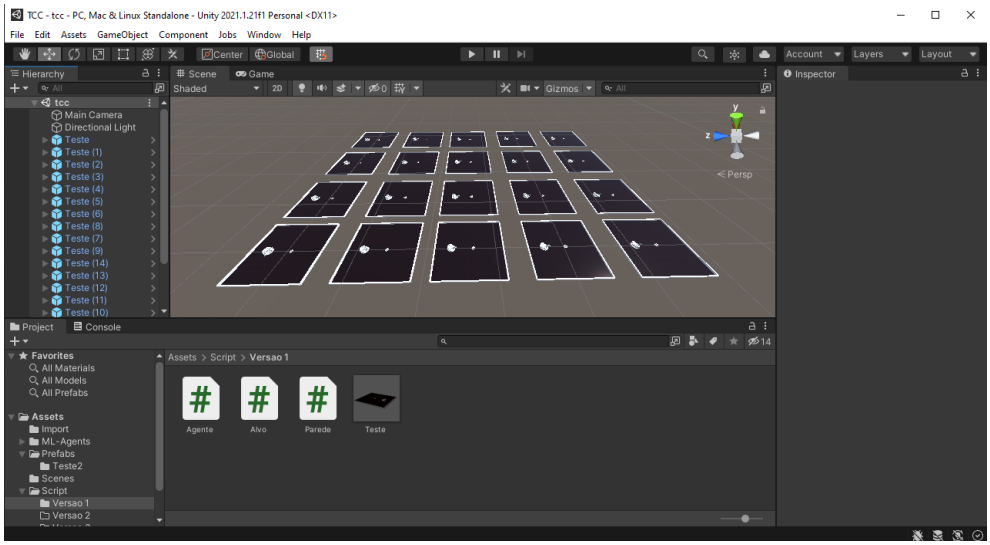
4.5 TREINAMENTO DA INTELIGENCIA ARTIFICIAL

4.5.1 Primeiro treinamento

O objetivo deste trabalho é simular o robô do experimento realizado por Pontuschka et. al (2016) no ITA/INPE e realizar pequenas ações, neste caso apenas que eles se encostem. Como objetivo desse primeiro treinamento nós determinamos que seria inicialmente fazer o robô aprender a chegar no alvo colocando-os bem perto para facilitar, apenas 20 unidades de distância e bem a frente dele.

Para realizar o treinamento deve-se digitar comandos no CMD dentro do *Virtual Enviroment*, e escrever a seguinte linha de comando: *mlagents-learn --run-id=teste01*. O *run-id* é o nome da pasta que será gerado o cérebro quando a inteligência artificial for treinada. A configuração de treinamento utilizada foi a padrão. Para que o treinamento ocorresse de maneira mais rápida, colocamos no cenário mais de um robô para treinar, fizemos vinte cópias para realizar o treinamento simultaneamente.

Figura 35 – Realização do primeiro teste



Fonte: Próprio autor.

Durante o treinamento, além da forma visual de avaliar o desempenho do treinamento vendo os robôs treinarem, a cada cinquenta mil passos são escritos no comand prompt (CMD), os dados do tempo que levou para completar os passos, o *Mean Reward* que é a média acumulativa das recompensas que os robôs conseguiram durante os episódios de treinamento, e o *Std. Reward* que é o desvio padrão ou margem de erro. Em nosso caso, o ideal seria o *Mean Reward* o mais próximo possível de 1.

Figura 36 – Dados iniciais do teste

```
INFO] Agente. Step: 50000. Time Elapsed: 101.555 s. Mean Reward: 0.999. Std of Reward: 0.032. Training.
INFO] Agente. Step: 100000. Time Elapsed: 190.420 s. Mean Reward: 0.992. Std of Reward: 0.125. Training.
INFO] Agente. Step: 150000. Time Elapsed: 279.740 s. Mean Reward: 0.993. Std of Reward: 0.117. Training.
INFO] Agente. Step: 200000. Time Elapsed: 356.428 s. Mean Reward: 1.000. Std of Reward: 0.020. Training.
INFO] Agente. Step: 250000. Time Elapsed: 437.686 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
INFO] Agente. Step: 300000. Time Elapsed: 569.768 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
INFO] Agente. Step: 350000. Time Elapsed: 722.083 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
INFO] Agente. Step: 400000. Time Elapsed: 883.753 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
INFO] Agente. Step: 450000. Time Elapsed: 1039.366 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
INFO] Agente. Step: 500000. Time Elapsed: 1191.521 s. Mean Reward: 1.000. Std of Reward: 0.000. Training.
```

Fonte: Próprio autor.

4.5.2 Avaliação gráfica do primeiro treinamento

O primeiro treinamento foi concluído e conseguimos objetivo de chegar ao alvo com êxito, então demos uma avaliada utilizando o *TensorBoard (TENSORFLOW 2021)*.

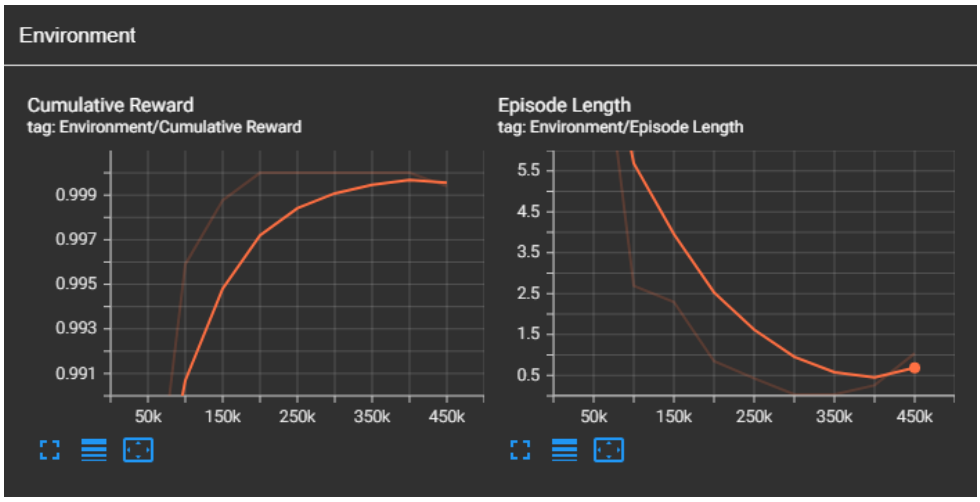
O ML-Agents Toolkit *(UNITY TECHNOLOGIES 2021)* salva estatísticas durante o treinamento e nos possibilita utilizar o utilitário do *TensorFlow (TENSORFLOW 2021)* chamado *TensorBoard (TENSORFLOW 2021)*.

Para utilizar basta digitar o seguinte comando: *tensorboard –logdir results*, e abrir o *localhost http:///localhost:6006*. Utilizando-o para a visualização de dados temos os gráficos da imagem a seguir.

O *Cumulative Reward* é a média de recompensa que os robôs conseguiram durante o treinamento e quanto mais perto e mais constante de um, melhor.

Episode Length é a duração média de cada episódio e quando menor, melhor ela é.

Figura 37 – Gráfico do Environment



Fonte: Próprio autor.

Police Loss é a média perdida durante as tomadas de decisões e conforme o treinamento for acontecendo ele deve diminuir para que o treinamento seja efetivo.

Value Loss é a média dos fatores de atualização, ela mostra o quão bem a inteligência artificial acerta em prever o valor das ações que são tomadas. Quanto mais estável a recompensa mais ela diminui.

Figura 38 – Gráficos de Policy e Value loss

Fonte: Próprio autor.

4.5.3 Melhorando o modelo de inteligência artificial

O pacote *ML-Agents (UNITY TECHNOLOGIES 2021)* também possibilita o aprimoramento do modelo inteligência artificial a partir do arquivo que foi gerado anteriormente. Basta digitar a linha de comando *mlagents-learn --initialize-from=Teste01 --run-id=Teste02.*

4.5.4 Segundo treinamento

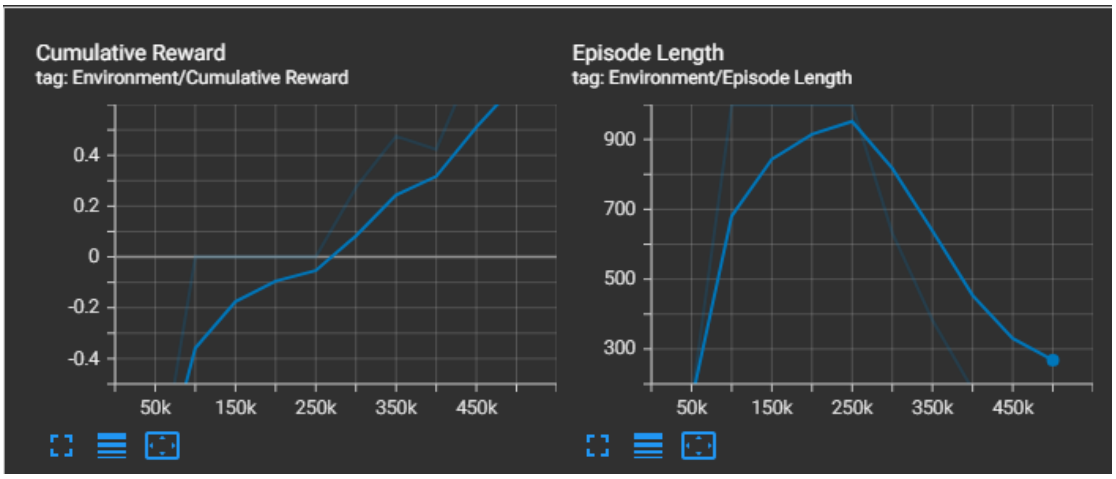
No segundo treinamento colocamos o alvo mais longe, colocamos ele no eixo x -30 e z 30. Fizemos uma alteração no código do método *OnEpisodeBegin*, para modificar a localização do alvo no início do episódio.

```
alvo.transform.localPosition = new Vector3(15f,0,15f);
```

Como o alvo ficou mais longe, decidimos mudar o *max steps* de um mil para cinco mil passos e para podermos ter um treinamento com duração maior. Foi necessário um milhão de passos ou dois treinamentos com a configuração padrão

para que o robô conseguisse com eficiência encontrar o alvo como mostra a comparação dos gráficos do *enviroment* a seguir.

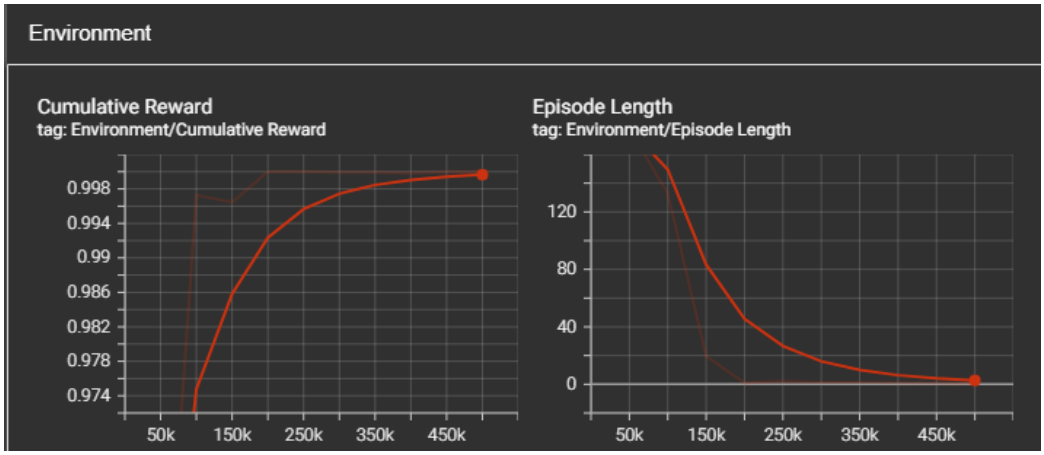
Figura 39 – Gráficos do Teste02



Fonte: Próprio autor.

Observando os gráficos percebemos no Teste02 uma média de acúmulo de recompensa negativo no início e depois uma evolução positiva de quase 0.8 no final e comprovando essa evolução o tempo de duração dos episódios do segundo gráfico mostram que o robô encontrou dificuldades em sair do primeiro treinamento que era apenas ir para frente e com isso o tempo de treino aumentou, porém, quando encontrou o alvo novamente o tempo de cada treinamento foi caindo consideravelmente.

Figura 40 – Gráfico do Teste03



Fonte: Próprio autor.

Como mostrado na imagem o Teste03 teve uma evolução muito mais rápida aumentando a média de recompensa positiva e uma redução no tempo do treino.

4.5.5 Terceiro treinamento

Como terceiro treinamento decidimos fazer o alvo sempre aparecer em uma posição aleatória dentro do ambiente toda vez que um novo episódio de treinamento começasse. Para isso editamos no *script* Agente.cs e alterando o método *OnBeginEpisode* como segue.

```
public override void OnEpisodeBegin()
    {
        velocidade = 1000f;
        transform.localPosition = Vector3.zero;
        //alvo.transform.localPosition = new Vector3(15f,0,15f);
        alvo.transform.localPosition = new Vector3(Random.Range(-40f, +40f), 0,
Random.Range(-40f, +40f));
            }
```

Para concluir esse treinamento foram necessários três milhões e quinhentos mil passos que são sete treinamentos com a configuração padrão para que o objetivo fosse concluído com maior eficácia.

Figura 41 – Teste04 em azul e Teste010 em vermelho

Fonte: Próprio autor

Como visto no gráfico anterior o treino04, em azul, o gráfico apresenta uma queda nas recompensas inicialmente, porém, ao decorrer a inteligência artificial vai aprimorando e a média aumenta junto com a duração dos episódios que diminui durante o treinamento.

No treino010, marcado em vermelho no gráfico, a inteligência artificial já está bem treinada e consegue manter uma média de recompensa perto do máximo de 1.0, e o tempo de duração dos episódios também mantém uma média baixa constante.

4.5.6 Quarto treinamento

No quarto treinamento colocamos que, no início de cada episódio, o alvo e o robô iniciariam em pontos aleatórios, modificando novamente o método *OnBeginEpisode*, como a seguir:

```
public override void OnEpisodeBegin()
    {
        velocidade = 1000f;
        //transform.localPosition = new Vector3(0f, 0, -40f);
        //alvo.transform.localPosition = new Vector3(0f,0,40f);

        transform.localPosition = new Vector3(Random.Range(-40f, +40f), 0,
Random.Range(-40f, +40f));
        alvo.transform.localPosition = new Vector3(Random.Range(-40f, +40f), 0,
Random.Range(-40f, +40f));
            }
```

Figura 42 – Teste011 em azul e Teste013 em verde

Fonte: Próprio autor

Foram necessários somente três treinamentos para que ele conseguisse atingir uma média de recompensa positiva alta e um tempo de treinamento relativamente baixo, considerando que entre um episódio e outro o alvo e o robô poderiam nascer muito perto ou muito longe um do outro.

5 CONSIDERAÇÕES FINAIS

Neste trabalho, nós descrevemos o estudo de Pontuschka et. al. (2016) realizado pelo ITA/IMPE com equipamentos robóticos para uso no espaço, fizemos uma modelagem em 3D desse robô e o inserimos em um ambiente virtual na *game engine Unity3D*. Dentro do ambiente virtual realizamos o estudo com inteligência artificial junto com aprendizado de máquina, utilizando o método de aprendizado por reforço e para realizar esse experimento de inteligência artificial utilizamos o pacote *Unity ML-Agents Toolkit* para os testes simulando o robô com objetivo de um robô encostar no outro. Durante esse treinamento da inteligência artificial, aplicamos pequenos desafios para aumentar gradualmente a dificuldade e melhorar a rede neural do robô.

A utilização do pacote *Unity ML-Agents Toolkit* foi de grande ajuda no experimento, pois oferece todo um conjunto de ferramentas para a realização de treinamento da inteligência artificial sem ter a necessidade de criar totalmente do zero um sistema próprio. Isto nos possibilitou focar onde realmente importa que é realizar testes de locomoção com o uso de inteligência artificial de modo que um robô se desloque com o uso dos propulsores e que toque o outro robô.

O resultado obtido foi satisfatório pois concluímos com êxito o que foi planejado inicialmente. Entretanto, o tema desse experimento pode ser ampliado em pesquisas futuras podendo adicionar novos desafios e outras funções necessárias para que o robô funcione um dia no espaço. Um exemplo é treinar para que não somente os robôs se toquem, mas que fiquem exatamente um de frente para o outro, o que possibilite o acoplamento dos braços robóticos.

REFERÊNCIAS

ALVES, André Felipe da Costa et al. **INTELIGÊNCIA ARTIFICIAL: Conceitos, Aplicações e Linguagens**. Conexão Eletrônica, Três Lagoas, v. 14, n. 1, p. 1733-1741, jan. 2017.

BLENDER FOUNDATION **Blender.** Blender Foundation. Disponível em: <https://www.blender.org>. Acesso em: 22, abril de 2021.

BREVE, Fabricio Aparecido. **Aprendizado de máquina em redes complexas**. 2010. Tese (Doutorado em Ciências de Computação e Matemática Computacional) - Instituto de Ciências Matemáticas e de Computação, Universidade de São Paulo, São Carlos, 2010. doi:10.11606/T.55.2010.tde-21092010-104722.

BROSTOM, Nick. **ARE YOU LIVING IN A COMPUTER SIMULATION?**. Publicado em Philosophical Quarterly (2003) Vol. 53, No. 211, pg.243-255.

COPPIN, Ben. **Inteligência Artificial**. Rio de Janeiro: Livros Técnicos e Científicos Editora Ltda, 2010.

EPIC GAMES **Unreal Engine 4**. Epic Games. Disponível em: <https://www.unrealengine.com>. Acesso em: 22, abril de 2021.

FOWLER, Martin. **UML Essencial**: um breve guia para linguagem-padrão de modelagem de objetos. 3. ed. Porto Alegre: Bookman, 2005.

GREGORY, Jason. **Game Engine Architecture**. 2. ed. New York: CRC Press, 2015.

SILVA, Fabrício. M. D.; LENZ, Maikon. L.; FREITAS, Pedro.H. C.; SANTOS, Sidney. C. Bispo. D. **Inteligência artificial**. Porto Alegre: SAGAH, 2019. 9788595029392.

Microsoft **Microsoft Visual Studio**. Disponível em: < https://visualstudio.microsoft.com/pt-br/vs/unity-tools/>. Acesso em: 10, setembro de 2021.

MONARD, Maria Carolina; BARANAUSKAS, José Augusto. Conceitos sobre Aprendizado de Máquina: capítulo 04. In: REZENDE, Solange Oliveira (org.). **Sistemas Inteligentes Para Engenharia**: fundamentos e aplicações. Barueri: Manole, 2005. Cap. 4. p. 39-56.

NORVIG, Peter; RUSSELL, Stuart. **Inteligência Artificial**. Rio de Janeiro: Elsevier Editora, 2013.

PONTUSCHKA, M. N.; FONSECA, I. M.; LIMA, G. L. **Communication System for an Experiment in Space Robotics**. Anais do XVIII *CBDO* (*2016*) 28 de novembro a 2 de dezembro de 2016. Águas de Lindóia, SP.

PRESSMAN, Roger S. **Engenharia de Software:** uma abordagem profissional – 7. ed. – Porto Alegre: AMGH, 2011

PYTHON SOFTWARE FOUNDATION. **Python**. Disponível em: <https://www.python.org/>. Acesso em: 10, setembro de 2021.

PYTORCH **PyTorch**. Disponível em: < https://pytorch.org>. Acesso em: 10, setembro de 2021.

SILVA, F. M.; LENZ, M. L.; FREITAS, P. H. C.; BISPO, S. C. **Inteligência Artificial**. Porto Alegre: Sagah, 2019. 232 p.

TENSORFLOW **TensorBoard**. Disponível em: <https://www.tensorflow.org/tensorboard>. Acesso em: 10, setembro de 2021.

UNITY TECHNOLOGIES **Unity ML-Agents Toolkit**. Disponível em: <https://github.com/Unity-Technologies/ml-agents>. Acesso em: 10, setembro de 2021.

UNITY TECHNOLOGIES **Unity**. Unity. Disponível em: <https://unity.com/pt>. Acesso em: 10, setembro de 2021.