

UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA
GRADUAÇÃO EM ENGENHARIA MECÂNICA

MATHEUS DE SOUZA FREITAS

Uso de Técnicas de Inteligência Artificial para o Desenvolvimento de Jogos
Sérios Aplicados a Estrutura Robótica para Reabilitação do Tornozelo Humano

Uberlândia – MG

2024

MATHEUS DE SOUZA FREITAS

Uso de Técnicas de Inteligência Artificial para o Desenvolvimento de Jogos Sérios Aplicados
a Estrutura Robótica para Reabilitação do Tornozelo Humano

Trabalho de Conclusão de Curso apresentado à
Faculdade de Engenharia Mecânica da Universidade
Federal de Uberlândia como requisito parcial para
obtenção do título de Engenheiro Mecânico.

Orientador: Prof. Dr. Rogério Sales Gonçalves

Uberlândia – MG

2024

MATHEUS DE SOUZA FREITAS

Uso de técnicas de inteligência artificial para o desenvolvimento de jogos sérios
aplicados a estrutura robótica para reabilitação do tornozelo humano

Trabalho de Conclusão de Curso apresentado à
Faculdade de Engenharia Mecânica da Universidade
Federal de Uberlândia como requisito parcial para
obtenção do título de Engenheiro Mecânico.

Uberlândia, 25 de abril de 2024

Banca Examinadora:

Prof. Dr. Rogério Sales Gonçalves – UFU (Orientador)

Prof.^a Dr.^a Guênia Mara Vieira Ladeira - UFU

Eng.^a M.^a Gabriella Faina Garcia

AGRADECIMENTOS

Primeiramente, agradeço a Deus por todos os encontros, desencontros, oportunidades, escolhas e crescimento que ocorreram ao longo não só desse projeto, como também da minha vida.

Agradeço também aos meus pais, Ione e Samir, pela educação que me fez possuir total confiança em mim mesmo; pela base que me fez ter uma enorme flexibilidade; pelos valores que me ensinaram a como viver nesse mundo e a quando questioná-lo; pelo incentivo a usar essas características a fim de fazer o que é do meu desejo; e pelo amor que sempre compartilharam comigo não importasse o momento.

Expresso também minha enorme gratidão a minha irmã, Isabella; aos meus amigos que carregou a tempos, Thiguin, Pedro e João; àqueles que a faculdade me deu, Arthur, Doutor (Lucas), e a muitos outros colegas próximos que sempre me ajudaram ao longo dessa fase; àqueles da UFU Racing que juntos nos moldamos como melhores Engenheiros.

Meu reconhecimento aos professores e colegas de toda vida que de alguma forma ajudaram a me fortalecer no caminho que sigo. A Lucas Pinheiro e Gustavo Vasconcelos que foram de grande ajuda na conclusão desse projeto. A todo corpo docente da Faculdade de Engenharia Mecânica (FEMEC) e da Universidade Federal de Uberlândia (UFU) que de diversas formas ampliaram meu horizonte para o que significa ser um engenheiro na sociedade.

A meu orientador Rogério Sales Gonçalves, meus sinceros agradecimentos, por sempre indicar o melhor caminho a seguir, por me incentivar a pesquisar e aplicar áreas do conhecimento que antes eu desconhecia completamente, e por toda ajuda ao longo desse tempo.

Finalmente, agradeço a CNPq pela bolsa do projeto 408013/2022-2; e, também, a todas as fontes de conhecimento como autores, pesquisadores e criadores de conteúdo digital que me alimentaram durante essa jornada e foram citados nesse trabalho. A todos que me apoiaram ao longo desse projeto, muito obrigado.

“Só é lutador quem sabe lutar consigo mesmo.”

Carlos Drummond de Andrade

RESUMO

Este Trabalho de Conclusão de Curso (TCC) envolve o uso de técnicas de Inteligência Artificial (IA) na criação de jogos sérios voltados a reabilitação do tornozelo humano. Seu objetivo é fazer o uso de IAs para ajudar o paciente em tempo real na realização de sua terapia, guiando-o para um movimento do pé adequado a sua condição enquanto joga, favorecendo o sentimento de realização ao conseguir uma boa pontuação no jogo.

Primeiro estudou-se quanto a criação de jogos e jogos sérios, que fatores devem-se levar em consideração para divertir o jogador, e que características deve ter para permitir uma reabilitação mais participativa ao usuário. Observou-se também o cenário de aplicação de IA em jogos e em fisioterapias, os tipos de equipamentos existentes e feedbacks fornecidos pelos pacientes.

Desenvolveu-se então um jogo para cada um dos três movimentos principais do tornozelo humano, buscando uma aproximação, e fácil associação, do movimento a ser performedo pelo usuário e o resultado deste na tela do jogo.

Na sequência foi realizado o estudo para o estabelecimento da comunicação entre hardwares e softwares, levando o movimento realizado na bancada de testes utilizada até o personagem do jogo (no Unity), possibilitando uma transmissão de informação responsiva e rápida a partir dos protocolos de comunicação. Houve também o processo de integração, centralizando as principais etapas de processamento e análise de dados no Matlab de forma a facilitar a comunicação e futuras alterações.

Posteriormente, focou-se na seleção da técnica de inteligência artificial mais adequada, na sua aplicação ao sistema já existente e no seu treinamento, alterando configurações e buscando uma Rede Neural Artificial (RNA) capaz de jogar sozinha, isto é, tomar o controle do personagem e alcançar a melhor pontuação possível.

Finalmente, integrou-se todo o sistema, desenvolveu-se uma forma de deixar a IA controlar o motor e foi possível fazê-la jogar, de acordo com objetivo inicial do trabalho, mostrando a possibilidade de os jogos sérios desenvolvidos serem utilizados nos procedimentos de reabilitação do tornozelo. Com os resultados simulados, apresenta-se a possibilidade da realização de uma assistência por parte do jogo ao usuário, de maneira a conduzi-lo, quando necessário, a completar as ações desejadas para a movimentação dos personagens nos jogos.

Palavras-chave: reabilitação do tornozelo humano, jogos sérios, inteligência artificial, estrutura robótica, protocolo de comunicação.

ABSTRACT

This Bachelor's Thesis (TCC) involves the use of Artificial Intelligence (AI) techniques to the creation of serious games aimed at the rehabilitation of the human ankle. Its objective is to use AIs to help the patient in real time in carrying out their therapy, guiding them to a foot movement appropriate to their condition while playing, favoring the feeling of accomplishment when achieving a good score in game.

Primarily, the creation of games and serious games was studied, searching for what factors must be considered to entertain the player, and what characteristics must be present to allow a engaged rehabilitation for the user. The scenario of AI in games and physiotherapy, as well as the types of existing equipment and feedback provided by patients were also taken into account.

A game was then developed for each of the three main movements of the human ankle, seeking an approximation, and easy association, of the movement to be performed by the user and its outcome on the game screen.

Subsequently, the establishment of a communication between hardware and software was done, taking the movement carried out on the existing test bench to the game character (in Unity), enabling a responsive and fast transmission of information based on the communication protocols employed. This phase also consists of the integration process, centralizing the main data processing and analysis steps in Matlab in order to facilitate communication and future changes.

Afterwards, the most appropriate artificial intelligence technique was then selected and applied to the existing system and then trained, while changing configurations and searching for an Artificial Neural Network (ANN) capable of playing by itself, which means, taking control of the character and reaching the best possible score.

At last, the entire system was integrated, a method was developed to let the AI manipulate the electric motor and it was possible to make it play, consistent with the initial objective of the study, thus demonstrating the potential for the developed serious games to be used in ankle rehabilitation procedures. With these results, there is potential for the game in aiding the user by guiding them as needed to complete the desired actions for character movement in the games.

Keywords: human ankle rehabilitation, serious games, artificial intelligence, robotic structure, communication protocol.

LISTA DE ILUSTRAÇÕES

Figura 1 –	Machine Learning	17
Figura 2 –	Movimentos do tornozelo humano	22
Figura 3 –	Bancada de Testes.	24
Figura 4 –	Floating Board	27
Figura 5 –	Tricor	28
Figura 6 –	City Walk	30
Figura 7 –	Exemplo de Calibração no Tricor	31
Figura 8 –	Gráfico 1 – Recompensa cumulativa ao longo do tempo	44
Figura 9 –	Gráfico 2 –Entropia ao longo do tempo	44
Figura 10 –	Gráfico 3 –Perda de valor ao longo do tempo	45
Figura 11 –	Gráfico 4 –Perda de política ao longo do tempo	46

LISTA DE ABREVIATURAS E SIGLAS

CNPq	Conselho Nacional de Desenvolvimento Científico e Tecnológico
UFU	Universidade Federal de Uberlândia
CAD	<i>Computer Aided Design</i>
LAR	Laboratório de Automação e Robótica
PLA	Ácido Polilático
TCP-IP	Protocolo de Controle de Transmissão - Protocolo de Internet
UDP	Protocolo de datagramas do usuário
IDE	<i>Arduino Integrated Development Environment</i>
IA	Inteligência Artificial
ML	<i>Machine Learning</i>
PPO	<i>Proximal Policy Optimization</i>
DRL	<i>Deep Reinforcement Learning</i>
PWM	<i>Pulse Width Modulation</i>
PPR	Pulsos por Revolução
PIVIC	Programa Institucional Voluntário de Iniciação Científica
bps	<i>bits per second</i>
NaN	<i>Not a Number</i>
GAIL	<i>Generative Adversarial Imitation Learning</i>
pip	<i>Preferred Installer Program</i>
ANN	<i>Artificial Neural Network</i>
AVC	Acidente Vascular Cerebral

SUMÁRIO

1	INTRODUÇÃO	14
2	OBJETIVOS.....	16
3	REVISÃO BIBLIOGRÁFICA.....	17
3.1.	Machine Learning.....	17
3.2.	Deep Learning.....	18
3.3.	Jogos Sérios e o software Unity 3D.....	18
3.4.	Equipamento Robótico para Reabilitação.....	19
3.5.	Fundamentos para Reabilitação.....	19
3.6.	Revisão Cinesiológica do Tornozelo.....	20
4	METODOLOGIA	23
4.1.	Bancada de Teste	23
4.1.1	Motor Bosch 24 V L006WM0310	24
4.1.2	Ponte H BTS7960B	24
4.1.3	Encoder Rotativo E40S6-1000	25
4.1.4	Arduino Mega	25
4.2.	Jogos Sérios	25
4.2.1	Floating Board	26
4.2.3	Tricor	27
4.2.4	City Walk.....	29
4.2.5	Calibração	30
4.2.6	Controles.....	31
4.3.	Comunicação e Interpretação.....	32
4.3.1	Arduino - Matlab	33
4.3.2	Matlab – Unity	34
4.4.	Implementação da Inteligência Artificial	35
4.4.1	Técnicas Escolhidas	35
4.4.2	Implementação com ML-agents	36
4.4.3	Configurações de aprendizado	37
4.4.4	Treinamento	39
4.4.5	IA ajudando.....	42

5	RESULTADOS.....	43
6	DISCUSSÃO	47
7	CONCLUSÕES	48
8	REFERÊNCIAS	49
	Apêndices	51
	Apêndice A – Códigos relacionados com a comunicação dos softwares, controle, dados e IA.....	52
	Código 1: Código do IDE Arduino de leitura do encoder e envio ao Matlab	52
	Código 2: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo Tricor	53
	Código 3: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo Floating Board	58
	Código 4: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo City Walk.....	62
	Código 5: Código do Unity2D do jogo Tricor para comunicação com o Matlab	67
	Código 6: Código do Unity2D do jogo City Walk para comunicação com o Matlab.....	70
	Código 7: Código do Unity2D do jogo Floating Board para comunicação com o Matlab.....	73
	Código 8: Arquivo de configuração tricolorB_Dimt_2_11.yaml para treinador PPO.....	76
	Código 9: Código Matlab do jogo Tricor para comunicação com o Unity quando há calibração.....	78
	Código 10: Código do IDE Arduino para quando a IA é capaz de jogar sozinha	83
	Código 11: Código do Matlab do jogo Tricor para comunicação com o Unity quando a IA é capaz de jogar sozinha.....	84
	Código 12: Código do Unity do jogo Tricor para o agente a ser treinado pela IA.....	89
	Apêndice B – Códigos dos jogos.....	93
	Código 13: Player do jogo Tricor quando controlado por IA	93
	Código 14: Agente do Jogo Tricor capaz de jogar sozinho	97
	Tricor sem IA 1.....	100
	Código 15: Calibração no jogo Tricor	100
	Código 16: Player no jogo Tricor, sem IA.....	103
	Tricor sem IA 2:	106
	Código 17: Menu Principal jogo Tricor	106
	Código 18: Gerenciamento do jogo Tricor.....	108
	Código 19: Menu de Fim de Jogo do jogo Tricor	113

Código 20: Comunicação UDP do Jogo Tricor.....	114
Código 21: Funcionamento do ambiente no jogo Tricor	117
Código 22: Funcionamento das moedas no jogo Tricor	118
Código 23: Funcionamento das barreiras no jogo Tricor	119
Jogo Floating Board.....	121
Código 24: Comunicação UDP do jogo Floating Board	121
Código 25: Player do Jogo Floating Board	123
Código 26: Menu principal do jogo Floating Board	127
Código 27: Menu de Fim de Jogo do jogo Floating Board	128
Código 28: Gerenciador de Jogo do jogo Floating Board.....	129
Código 29: Controle da chuva no jogo Floating Board	133
Código 30:Funcionamento da plataforma no jogo Floating Board	135
Jogo City Walk	136
Código 31: Comunicação UDP do jogo City Walk.....	136
Código 32: Player do jogo City Walk	140
Código 33: Controlador da interface de usuário para o Menu Principal do jogo City Walk	149
Código 34: Controlador da interface de usuário do jogo City Walk.....	150
Código 35: Controlador das animações do personagem.....	152
Código 36: Efeito Paralaxe do jogo City Walk.....	153
Código 37: Funcionamento dos obstáculos do jogo City Walk	154
Código 38: Funcionamentos dos prédios do jogo City Walk.....	155
Código 39: Funcionamento das moedas do jogo City Walk.....	159
ANEXO 1 – Datasheet do encoder rotativo E40S6 - 1000	160

1 INTRODUÇÃO

A integração da tecnologia na reabilitação está transformando a abordagem de lesões musculoesqueléticas e neurológicas, ampliando as opções terapêuticas e oferecendo novas oportunidades de personalização e monitoramento. O uso de estruturas robóticas na reabilitação se destaca como uma abordagem inovadora para melhorar a função muscular de forma quantitativa, permitindo intervenções controladas, personalizadas e intensivas, melhorando a função, a mobilidade e a força muscular. Essa combinação de expertise clínica e tecnologia está redefinindo os padrões de cuidados e oferecendo novas perspectivas para a recuperação de pacientes (ZHANG; DAVIES; XIE, 2013).

Complementarmente, o uso de jogos sérios na reabilitação tem se destacado como uma abordagem inovadora e promissora para auxiliar no tratamento e na recuperação de pacientes em diversas áreas da saúde, como na reabilitação, melhorando a qualidade de vida dos indivíduos nesse processo. O desenvolvimento desses jogos tem avançado globalmente, com pesquisadores e profissionais de saúde explorando maneiras de integrar essa tecnologia nos tratamentos (SUDARMILAH et al., 2018).

A personalização dos jogos para atender às necessidades individuais de cada paciente, combinada com o uso da Inteligência Artificial para adaptar dinamicamente os desafios e fornecer feedback personalizado, representa uma abordagem promissora para promover a recuperação e o bem-estar dos pacientes na área da reabilitação (SUDARMILAH et al., 2018).

Neste projeto de conclusão de curso, foram confeccionados 3 jogos sérios correspondentes aos 3 eixos de movimentação do tornozelo humano. Tais jogos foram projetados para serem conectados a uma estrutura robótica desenvolvida para a reabilitação do tornozelo. Por meio dela o paciente deve ser capaz de jogar ao mesmo tempo em que é auxiliado por um dispositivo robótico.

Esses movimentos são calculados pelo uso de inteligência artificial de maneira a ajudar o paciente a recuperar sua amplitude de movimento. Simultaneamente, ele também poderá se engajar visualmente de diferentes formas a depender do jogo desenvolvido. Com o passar do tempo, é esperado que o paciente precise de menos e menos assistência; e o jogo deve reconhecer essa situação deixando a pessoa mais livre. Todos os dados do processo de recuperação podem ser coletados, futuramente analisados, e, eventualmente, fornecer relatórios mais detalhados ao profissional da saúde.

Aspectos teóricos e práticos a respeito do desenvolvimento dos jogos, da comunicação necessária entre *hardware* e *softwares* e das técnicas de inteligência artificial consideradas e utilizadas serão abordados neste trabalho. Em todas essas etapas pensou-se na efetividade do sistema, na praticidade dos jogos, e no divertimento dos pacientes, que podem ser pessoas em variadas condições.

Em resumo, este trabalho estuda e aplica o uso de técnicas de inteligência artificial em jogos sérios controlados por uma estrutura robótica de forma a auxiliar o paciente ao mesmo tempo em que se espera aumentar a eficácia e a personalização da reabilitação do tornozelo, facilitando também o trabalho do profissional da área, sendo, assim, capaz de obter dados mais precisos a respeito da evolução do paciente e sugerir tratamentos mais adequados.

2 OBJETIVOS

O principal objetivo do uso de terapia assistida por robôs na reabilitação é melhorar a função e a mobilidade de pacientes com lesões musculoesqueléticas ou neurológicas, proporcionando intervenções altamente controladas, personalizadas e intensivas. Essa abordagem visa otimizar a recuperação do tornozelo, promovendo a reabilitação de forma precisa e eficaz, com o auxílio de dispositivos robóticos projetados para fornecer suporte, feedback e estímulo durante as sessões terapêuticas. A terapia assistida por robôs busca maximizar os resultados clínicos, acelerar o processo de reabilitação, aumentar a independência funcional dos pacientes e melhorar sua qualidade de vida, oferecendo uma abordagem inovadora para a reabilitação do tornozelo (ZHANG; DAVIES; XIE, 2013).

Paralelamente, os jogos sérios nesse contexto têm como objetivo principal proporcionar uma abordagem inovadora e eficaz para auxiliar no processo de recuperação física, cognitiva e emocional dos pacientes. Esses jogos são projetados para estimular a participação ativa dos indivíduos em seu tratamento, promover a aprendizagem de habilidades específicas, melhorar a motivação e o engajamento na reabilitação, e, em última análise, contribuir para a melhoria da qualidade de vida dos pacientes. Integrando elementos lúdicos e interativos, buscam tornar a recuperação mais atraente, envolvente e personalizada, adaptando-se às necessidades individuais, oferecendo desafios adequados, monitorando o progresso e fornecendo feedback personalizado (SUDARMILAH et al., 2018).

Nesse projeto, objetiva-se estudar os procedimentos para viabilizar a reabilitação com estruturas robóticas, bem como das técnicas para incentivar o jogador a sentir-se mais confiante e disposto a participar mais ativamente na seção de reabilitação. Além disso, compreender o desenvolvimento de jogos, incluindo os passos para tornar um código mais estruturado, o manejo de ativos online, o desenvolvimento e a utilização de animações, e a sistematização das interações do usuário para promover conforto e engajamento.

Também faz parte dos objetivos aprender sobre os diferentes tipos de comunicação entre sistemas hardware e software, saber como manipulá-los para atender seus objetivos e como integrá-los a sistemas que utilizam processamento paralelo. Outro ponto importante é o estudo, seleção e uso de variadas técnicas de Inteligência Artificial, bem como a análise dos resultados provenientes desses modelos. Por fim, a intenção é integrar todo esse conhecimento para desenvolver jogos sérios voltados para a reabilitação do tornozelo, utilizando a inteligência artificial para fornecer auxílio.

3 REVISÃO BIBLIOGRÁFICA

Nesta seção serão apresentados os fundamentos teóricos necessários para o desenvolvimento deste projeto.

3.1. Machine Learning

Machine learning é um subcampo da inteligência artificial que se concentra no desenvolvimento de algoritmos e modelos que permitem aos computadores aprender padrões e tomar decisões com base em dados, sem serem explicitamente programados. Em essência, o objetivo do machine learning é capacitar os sistemas computacionais a aprender e melhorar automaticamente a partir da experiência, sem intervenção humana direta. Isso é alcançado através da análise de dados, identificação de padrões e construção de modelos preditivos ou descritivos para realizar tarefas específicas (HU; LAURIÈRE, 2023).

Figura 1: Machine Learning



Fonte: “El Blog de Jordi: La MÁQUINA que aprende como un HUMANO”, [s.d.].

3.2. Deep Learning

Deep learning (DL) é um conceito de aprendizado de máquina baseado em redes neurais artificiais. Houve avanços significativos em algoritmos sofisticados e técnicas de pré-processamento de dados, resultando em redes neurais cada vez mais profundas e capazes, conhecidas como *deep learning*. O termo “profunda” se refere a um substancial número de camadas na rede neural, favorecendo aplicações mais complexas, com dados grandes e de alta dimensão, superando algoritmos rasos em aplicações de processamento de texto, imagem, vídeo, fala e áudio. Embora o DL possa ter desempenho super-humano em muitos casos, problemas que exigem capacidades de IA fortes ainda não podem ser resolvidos diretamente (JANIESCH; ZSCHECH; HEINRICH, 2021).

3.3. Jogos Sérios e o software Unity 3D

Os jogos sérios são jogos digitais desenvolvidos com o propósito de transmitir conhecimento, treinar habilidades específicas, simular situações do mundo real ou promover a mudança de comportamento em áreas como educação, treinamento, saúde, pesquisa e terapia. Na reabilitação, são aplicados de várias formas para auxiliar na recuperação e melhoria da qualidade de vida de pacientes. Podem ajudar na reabilitação física, melhorando coordenação motora, força muscular e mobilidade; na reabilitação cognitiva, estimulando funções como memória, atenção, raciocínio e resolução de problemas; na reabilitação de distúrbios neurológicos, treinando habilidades como coordenação motora fina, comunicação e interação social; e na reabilitação pediátrica, proporcionando uma forma lúdica de promover a recuperação e o bem-estar emocional após procedimentos médicos (SUDARMILAH et al., 2018).

O Unity3D é uma *engine* de desenvolvimento de jogos amplamente utilizada, inclusive na criação de jogos sérios para reabilitação. Seus benefícios incluem: baixo custo de entrada, oferecendo versão gratuita com benefícios quase ilimitados; facilidade de uso, com interface amigável e curva de aprendizado acessível, permitindo que desenvolvedores novatos criem jogos bem eficientes; comunidade forte e ativa; Asset Store, onde os desenvolvedores podem encontrar e compartilhar recursos; flexibilidade e extensibilidade, sendo altamente personalizável e permitindo a integração de diferentes tecnologias e dispositivos; suporte a múltiplas plataformas; e a utilização do C# como linguagem de programação que, por ser

desenvolvida pela *Microsoft*, possui amplo material de pesquisa disponível online (REPUBLIC; 2015, [s.d.]).

3.4. Equipamento Robótico para Reabilitação

Equipamentos robóticos de reabilitação são dispositivos projetados para auxiliar na recuperação física e funcional de pacientes com lesões neurológicas, musculoesqueléticas ou outras condições que afetam a mobilidade e a função motora. Eles variam em complexidade e funcionalidade, desde dispositivos simples de suporte de peso até sistemas sofisticados com sensores e atuadores avançados (GONÇALVES; KREBS, 2017).

Esses dispositivos fornecem assistência personalizada e repetitiva durante exercícios terapêuticos, ajudando os pacientes a melhorar suas habilidades motoras e acelerar o processo de recuperação. Podem ser utilizados em uma variedade de contextos de reabilitação e são uma ferramenta complementar aos tratamentos tradicionais, proporcionando uma abordagem inovadora e eficaz para a reabilitação neuro-motora (GONÇALVES; KREBS, 2017).

Apesar de a construção desse equipamento não ser parte dos objetivos deste projeto, os jogos foram pensados de forma a atendê-lo.

3.5. Fundamentos para Reabilitação

Levam, usualmente, a necessidade da reabilitação física casos como: lesões ortopédicas (fraturas, luxações, lesões nos tecidos moles), cirurgias ortopédicas, lesões cerebrais (como AVC), condições neurológicas (como esclerose múltipla, paralisia cerebral e doença de Parkinson), lesões medulares, lesões desportivas ou amputações.

Na reabilitação do tornozelo sem o uso de equipamentos robóticos, os profissionais enfatizam a forma de terapia, preferindo o "exercício de assistência ativa" para minimizar a assistência manual. Eles prescrevem exercícios específicos adaptados às necessidades do paciente para fortalecer os músculos ao redor do tornozelo, melhorar a estabilidade articular e promover a mobilidade. Também utilizam técnicas de alongamento e mobilização articular para melhorar a flexibilidade e restaurar a amplitude de movimento. Além disso, focam no treinamento de equilíbrio e propriocepção para aprimorar a consciência corporal, estabilidade postural e prevenção de lesões. A reeducação da marcha e dos padrões de movimento também

é parte essencial para garantir uma distribuição adequada do peso corporal durante a caminhada e atividades funcionais (ZHANG; DAVIES; XIE, 2013).

A terapia assistida por robô pode trazer várias melhorias em relação aos métodos convencionais de reabilitação do tornozelo, sendo elas, mais especificadamente:

- **Personalização e Feedback Preciso:** oferecem uma abordagem mais personalizada, adaptando-se às necessidades individuais de cada paciente e fornecendo feedback preciso sobre o desempenho durante a reabilitação do tornozelo;
- **Controle de Movimento:** podem fornecer um controle preciso do movimento, permitindo a execução de exercícios terapêuticos específicos de forma consistente e controlada, o que pode ser desafiador de alcançar com métodos convencionais;
- **Intensidade e Duração da Terapia:** facilitam a realização de sessões de reabilitação mais intensivas e prolongadas, o que pode acelerar o processo de recuperação em comparação com os métodos convencionais que podem ser limitados pela fadiga do terapeuta;
- **Monitoramento Objetivo do Progresso:** possibilitam a obtenção de dados objetivos sobre o progresso do paciente, permitindo uma avaliação mais precisa e uma adaptação mais eficaz do plano de tratamento ao longo do tempo;
- **Engajamento do Paciente:** a natureza interativa e muitas vezes lúdica dos sistemas robóticos pode aumentar o engajamento e a motivação dos pacientes durante a terapia de reabilitação do tornozelo, tornando o processo mais agradável e potencialmente mais eficaz.

Essas melhorias destacam com a terapia assistida por robô, e pode complementar e aprimorar os métodos convencionais de reabilitação do tornozelo, oferecendo benefícios adicionais que podem resultar em uma recuperação mais eficaz e eficiente (ZHANG; DAVIES; XIE, 2013).

3.6. Revisão Cinesiológica do Tornozelo

A cinesiologia do tornozelo é um campo de estudo fundamental para compreender a anatomia, os movimentos e as limitações dessa articulação complexa que permite uma ampla

gama de movimentos essenciais para a locomoção, equilíbrio e estabilidade do corpo. O tornozelo é uma articulação sinovial, também conhecida como articulação em dobradiça. Esse tipo de articulação permite movimentos em plano único (SAGLIA et al., 2009).

O pé, ligado ao tornozelo, tem como funções principais:

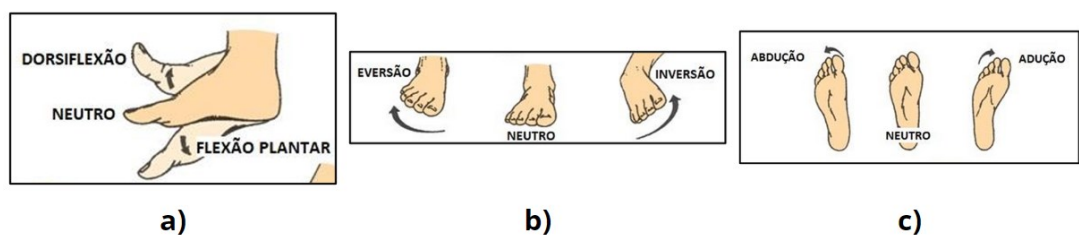
- Suportar o peso do corpo durante a postura em pé e durante atividades como caminhar, correr e pular. A distribuição adequada do peso através das estruturas do pé é crucial para a estabilidade e prevenção de lesões;
- A propulsão já que durante essa fase da marcha, o pé atua como uma alavanca, impulsionando o corpo para frente. A capacidade de flexão e extensão dos músculos e articulações do pé é fundamental para gerar impulso e movimento eficiente;
- A absorção de choque, visto que o pé possui uma estrutura complexa de arcos (longitudinal e transversal) que atua como um amortecedor natural, absorvendo o impacto gerado durante o contato com o solo. Essa capacidade de absorção de choque protege as articulações superiores, como os joelhos e quadris, contra lesões;
- O equilíbrio e a estabilidade, onde os músculos intrínsecos e extrínsecos do pé trabalham em conjunto para manter o equilíbrio do corpo em diferentes superfícies e posições. A propriocepção e a sensibilidade tátil dos receptores sensoriais do pé desempenham um papel crucial na regulação do equilíbrio.

Por esses motivos é importante o estudo dos tipos de movimento que esse membro consegue efetuar (SAGLIA et al., 2009), temos então:

- Dorsiflexão e Plantiflexão: Dorsiflexão é o movimento de levantar a parte superior do pé em direção à tíbia, enquanto plantiflexão é o movimento oposto, apontando o pé para baixo em direção ao solo. São essenciais para atividades como caminhar, correr e subir escadas.
 - Amplitude de movimento: Flexão plantar, 0 a 45°. Flexão dorsal, 0 a 20° (Valores de referência)(SILVA, 2022).
 - O eixo de rotação é horizontal e passa lateralmente de um lado para o outro com os movimentos ocorrendo principalmente no plano sagital;
 - Limitações: A dorsiflexão pode ser limitada por tensão nos músculos da panturrilha (gastrocnêmio e sóleo) ou por restrições articulares, como osteófitos ou contraturas;

- Inversão e Eversão: Inversão é o movimento de inclinar a sola do pé para dentro, em direção à linha média do corpo, enquanto a eversão é o movimento oposto, inclinando a sola do pé para fora.
 - Amplitude de movimento: Inversão, 0 a 20°. Eversão, 0 a 5° (Valores de referência) (SILVA, 2022).
 - O eixo de rotação é vertical e passa de cima para baixo com os movimentos ocorrendo no plano frontal;
 - Limitações: As lesões de ligamentos, como uma entorse de tornozelo, podem limitar a inversão e eversão, resultando em instabilidade e dor durante a marcha lateral ou atividades de equilíbrio;
- Abdução e Adução: Abdução é o movimento de afastar o pé da linha média do corpo, enquanto a adução é o movimento de trazê-lo de volta para a linha média.
 - Amplitude de movimento: Abdução, 0 a 20°. Adução, 0 a 40° (Valores de referência)(SILVA, 2022).
 - O eixo para esses movimentos seria anteroposterior;
 - Limitações: A amplitude de movimento na abdução e adução pode ser afetada por lesões articulares, como osteoartrite, ou por fraqueza muscular nos músculos intrínsecos do pé;

Figura 2: – Movimentos do tornozelo humano. (a) Dorsiflexão e flexão plantar; (b) Inversão e Eversão; (c) Abdução e Adução.



Fonte: Adaptado de <https://www.elearnuk.co.uk/uploads/courses/168.pdf>.

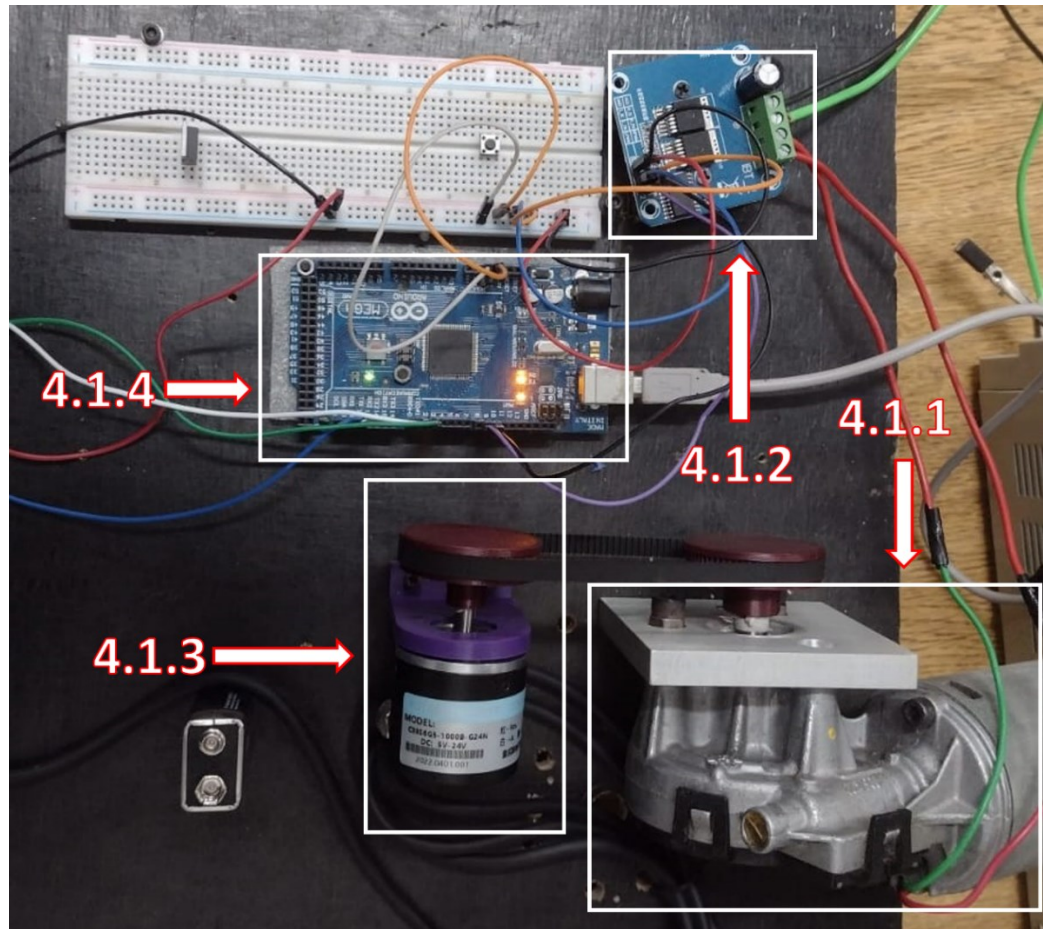
4 METODOLOGIA

Este projeto utiliza uma metodologia composta de 3 principais etapas. Primeiramente, ocorreu o desenvolvimento dos Jogos Sérios, com preocupação voltada para a diversidade nas mecânicas de interatividade. Logo em seguida, trabalhou-se na consolidação de uma comunicação efetiva entre Arduino, Matlab e Unity. Por fim, pesquisou-se, e houve a aplicação e treinamento de inteligência artificial através da técnica de PPO (Otimização de Política Próxima) presente dentro do Pacote ML-agents no Unity2D.

4.1. Bancada de Teste

A bancada de testes utilizada (Figura 3) foi desenvolvida pelo aluno Alexandre Mendes Marcelo, no projeto da modalidade PIVIC de nome “Controle de velocidade de um motor de corrente contínua”, sob orientação do professor Pedro Augusto Queiroz de Assis. Ela tem como base uma estrutura de compensado de madeira, a qual se fixou, por meio de parafusos, um motor Bosch 24 V (acoplado a uma placa de alumínio para fixação em posição adequada), um Encoder Incremental de 1000 PPR, uma ponte H, um Arduino MEGA, e uma protoboard. O motor é conectado ao encoder através de uma correia e duas rodas dentadas (produzidas por impressão 3D). Dessa forma, com o uso da ponte H, se é capaz de enviar sinais PWM de velocidade (tensão) ao motor, além disso, ocorre a leitura da posição atual do encoder (movimentado pelo motor ou pelo usuário). Essa disposição permite o teste constante dos Jogos Sérios, do sistema de comunicação e da Inteligência Artificial. O desenvolvimento deste equipamento não é objetivo deste projeto.

Figura 3: Bancada de Testes



Fonte: Autor, 2024.

A seguir são detalhados os principais componentes do protótipo utilizado neste projeto.

4.1.1 Motor Bosch 24 V L006WM0310

Este motor Bosch é de corrente contínua, próprio para uso em testes com Arduino. Tem nele acoplado uma caixa de redução que se liga a uma das engrenagens. É alimentado pelas conexões com a ponte H. Este terá como objetivo corrigir o paciente quando este se desvia demasiadamente do curso ideal de movimentação do tornozelo para o jogo. Neste projeto, no entanto, ele age apenas com controle não adaptativo, representando o mencionado curso ideal de movimentação proporcionado pela IA.

4.1.2 Ponte H BTS7960B

Com a utilização da ponte H se torna possível controlar o sentido e a velocidade de rotação do motor (pela conexão com a placa Arduino), isso é feito por meio de quatro transistores agindo como chaves eletrônicas.

4.1.3 Encoder Rotativo E40S6-1000

É um encoder incremental, ou seja, tem zero resetado toda vez que o programa de Arduino é reiniciado. Dessa forma, o valor fornecido representa o número de pulsos existentes entre a posição no qual ele se encontrava ao ser ligado e a atual (multiplicado pelo número de fases de cada pulso, quatro). Assim, é possível saber o ângulo entre essas duas posições. Isso é usado a todo momento pois é através do encoder que o paciente pode controlar o personagem dentro dos jogos, e pelo qual o profissional acompanha seu desenvolvimento. No anexo 01 tem-se o datasheet do encoder.

4.1.4 Arduino Mega

O IDE Arduino é utilizado por sua fácil programação e integração com diferentes dispositivos. Nesse caso utiliza-se a placa Arduino Mega, contudo, para a aplicação em questão, ela não é estritamente necessária, podendo ser uma com menos conectores. Com ela é possível ler dados do encoder e enviar comandos ao motor.

4.2. Jogos Sérios

Inicialmente, estudou-se a respeito do desenvolvimento de jogos no Unity (desenvolvido pela Unity Technologies). Isso foi feito, principalmente, através de uma playlist de vídeos no canal Fredi no Youtube (“Fredi - YouTube”, [s.d.]), chamada “Unity” (“Unity - YouTube”, [s.d.]). Com inspiração no jogo desenvolvido nessa sequência de vídeos, foi criado o primeiro jogo, “Floating Board”.

Uma das bases para a criação dos jogos foi maximizar estímulos positivos e minimizar as punições, isso foi feito com o objetivo de evitar o desânimo e o sentimento de impotência no paciente, de forma que ele sempre tenha vontade de testar outras opções. Por isso, em dois dos jogos não existe morte como punição, e a única forma de encerrar é pausar o jogo e clicar no

botão correspondente. Além disso, em todos eles a pontuação sobe bem mais rápido que diminui, pelo mesmo motivo.

Nos três jogos houve o uso da fonte Dimbo (“Dimbo Font | dafont.com”, [s.d.]); e em alguns utilizou-se as *skybox* de *Boxophobic* (“FREE Skybox Extended Shader | VFX Shaders | Unity Asset Store”, [s.d.]). Os *assets* da *Craftpix* são grátis e licenciados (“File Licenses - CraftPix.net”, [s.d.]).

4.2.1 Floating Board

O jogo funciona da seguinte forma, existe, acima de um lago, uma plataforma de madeira que gira no sentido horário e anti-horário, aleatoriamente entre posições pré-determinadas. O jogador se encontra como um quadrado em cima dessa plataforma. Ele pode se movimentar por meio do atrito com ela girando, também, nos sentidos horário e anti-horário, se deslocando para a direita ou para a esquerda.

Quando a plataforma gira para qualquer das posições ela é acompanhada por uma chuva que faz o jogador perder pontos caso ele esteja do lado mais próximo do lago. Se, por acaso, ele entrar no lago, ele perde mais pontos ainda. O objetivo é evitar os elementos que levam a essas penalidades, como se estivesse se “equilibrando” numa prancha. Uma imagem é a apresentada na Figura 4.

O objetivo aqui é fazer com que o paciente tenha que prestar atenção em mais de um elemento para inferir seu desempenho de acordo com a pontuação mostrada na tela. O tipo de movimentação associado a esse jogo é o de **inversão-eversão**, para simular aquilo que se está vendo na tela e tornar a associação mais fácil. Esse é um jogo de baixa dificuldade, mas que conta com uma divertida interação por parte do paciente pela forma como o bloco que age como o jogador se mexe.

Os *scripts* desse jogo são apresentados no Apêndice B, dos Códigos 24 até 30.

Figura 4: Floating Board



Fonte: Elaborado pelo autor, 2024.

4.2.3 Tricor

Tricor é o segundo jogo, ele foi desenvolvido pensando nos movimentos de **abdução-abdução**, dando a ideia de que o jogador se movimenta na direção para a qual o pé do paciente aponta.

Ele é composto de uma “estrada” que está sempre se deslocando em velocidade constante que é dividida em três pistas de cores diferentes que alternam entre si. Cada cor dá uma pontuação diferente ao jogador, por vezes positiva, por outras negativa. Dessa forma o objetivo é obter a maior pontuação possível conforme o tempo passa. Existem também obstáculos (cercas) e recompensas (moedas) ao longo do caminho que podem dar ou tirar pontos. As moedas fornecem uma dica da cor que dá mais ponto (essa não é uma mecânica necessária).

Como o jogo depende muito da identificação das cores, cada uma delas foi associada ao símbolo de uma certa moeda: €, ¥, \$. Além disso, priorizou-se o uso de cores de fácil identificação para a maioria dos casos de daltonismo: amarelo, cinza e azul. A associação é respectiva. Há um exemplo na Figura 5.

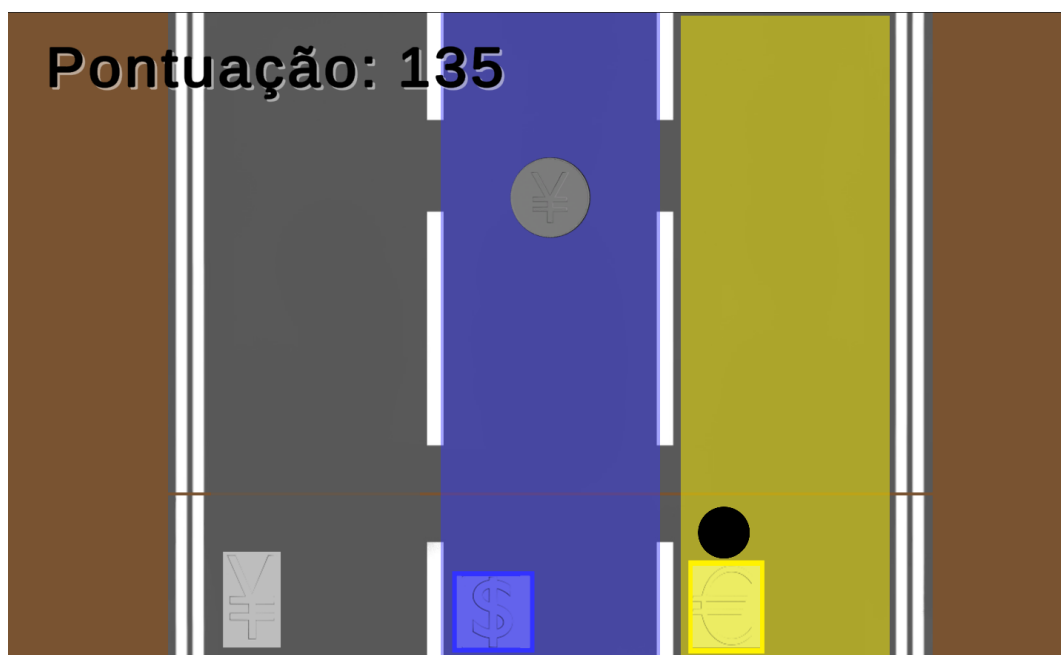
As seis possíveis combinações de cores na estrada, podem ser divididas em três grupos distintos, nos quais uma cor diferente estará disposta no centro. Essa cor corresponderá a uma outra cor que possibilita o aumento da pontuação, esteja ela à sua esquerda ou direita. Um exemplo: sempre que a cor **cinza** estiver no **meio**, o **amarelo** será a cor que dá **mais pontos**, e o azul aquela que faz o jogador perder mais pontos; usualmente o cinza fará a pontuação aumentar ou diminuir menos que as outras duas.

Esse jogo é bem desafiador, pois requer:

- memória para lembrar a correspondências corretas;
- atenção difusa para perceber o padrão atual e, ao mesmo tempo, se esquivar de obstáculos e ir na direção de recompensas;
- pensamento rápido para identificar o padrão corretamente e associar a cor que fornecerá maior pontuação;
- baixo tempo de resposta e ação correspondente.

Tomou-se, portanto, bastante cuidado para evitar que seja um jogo desmotivador, encontrando o tempo adequado para troca de cores. Futuramente, seria possível ajustar esse tempo conforme o desempenho do jogador. Os *scripts* desse jogo estão presentes no Apêndice B, dos Códigos 13 até 23.

Figura 5: Tricor



Fonte: Elaborado pelo autor, 2024.

4.2.4 City Walk

O terceiro jogo é o City Walk, um jogo de plataforma estilo *infinite runner* (corredor infinito), pensado para os movimentos de **dorsiflexão e flexão plantar**. Quando o pé sobe o jogador pula, quando o pé desce jogador se agacha. Dessa forma a associação pé-jogador fica mais fácil.

O objetivo é correr a maior distância possível, quanto mais se corre mais a velocidade aumenta e vai ficando mais difícil de controlar o personagem. Deve-se pular de um prédio para o outro (evitando o vão entre eles) e desviar de obstáculos à altura do peito que obrigam o jogador a agachar, caso não queiram perder velocidade.

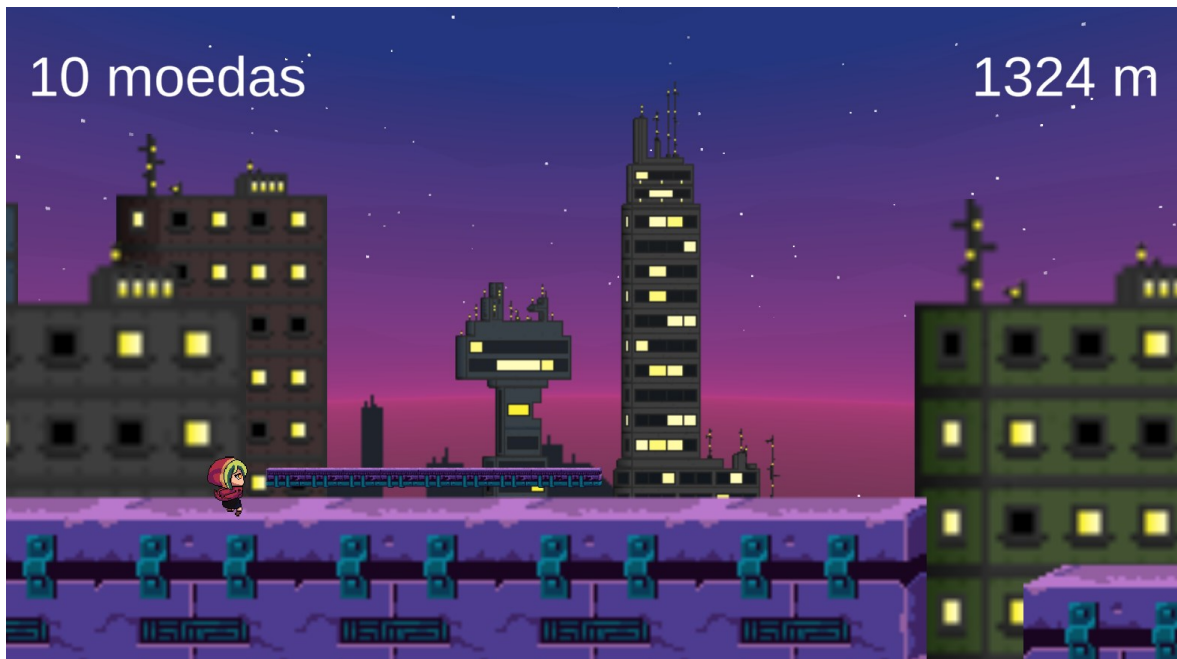
Existem também moedas ao longo do caminho, que são uma outra medição de desempenho, elas estão mais próximas das bordas dos prédios, sendo então mais difíceis de serem pegadas. A cada cinco moedas é acionado um evento que aumenta momentaneamente a velocidade do personagem.

Devido a aleatoriedade de posições e distâncias dos prédios, assim como da presença ou não de obstáculos acima deles, esse é um jogo bastante dinâmico e divertido de ser jogado, apesar dos seus controles simples. A preocupação com a dificuldade é menor visto que ele é feito para ser jogado mais vezes. Uma imagem pode ser vista na Figura 6.

A física desse jogo é baseada no jogo desenvolvido pelo canal (“superjustin5000 - YouTube”, [s.d.]) durante a série de vídeos dedicadas a um jogo que se assemelha a CANABALT (um clássico *infinite runner* desenvolvido por Adam Saltsman lançado em 2009) (“Making An Infinite Runner like Canabalt in Unity - YouTube”, [s.d.]). Ademais, foram usados *assets* de *Warped City 2* (“Warped City 2 | 2D Environments | Unity Asset Store”, [s.d.]), e da personagem *Enchantess* da *CraftPix* (“Fantasy Chibi Female Sprite Sheets Pixel Art - CraftPix.net”, [s.d.]).

Os *scripts* desse jogo podem ser visualizado no Apêndice B, dos Códigos 31 até 39.

Figura 6: City Walk



Fonte: Elaborado pelo autor, 2024.

4.2.5 Calibração

Como cada pessoa tem uma amplitude natural de movimento distinta, e cada paciente terá uma amplitude diferente com o passar dos dias de reabilitação, é necessário haver a calibração do equipamento sempre que uma nova sessão se iniciar. Para isso, no início de cada jogo há uma tela de calibração para três posições: a mais confortável (a partir da qual se medirá o zero), e os dois extremos de cada conjunto de movimento.

Em posse desses valores pode-se:

- determinar os limites a serem exigidos, pelo programa, do paciente ao jogar para que certos comandos sejam enviados;
- ter um histórico de melhora na condição do paciente, a partir do qual o profissional pode determinar se aquele tipo de tratamento é mais ou menos efetivo;
- futuramente, determinar zonas de angulação nas quais se percebe um maior controle do paciente e aquelas nas quais ele tem maior dificuldade, sendo possível, também, incentivar aos poucos uma maior amplitude de movimento.

Figura 7 – Exemplo de Calibração no Tricor



Fonte: Elaborado pelo autor, 2024.

4.2.6 Controles

Não basta definir a direção do movimento apropriada para cada jogo, deve-se pensar com cuidado em como será feito o controle do personagem pelo paciente. Primeiramente, deve ser um controle intuitivo, que com poucos segundos de jogo a pessoa possa se adaptar, e torne fácil a manipulação, evitando, a todo custo, qualquer sentimento de impotência por essa razão.

Além disso, não é interessante que o paciente tenha que realizar movimentos bruscos ou muito frequentes e relativamente rápidos para evitar maiores lesões ou fadiga dos músculos. No entanto, é importante incentivar a movimentação, e para isso, além de um jogo interessante, é importante oferecer certo nível conforto nas tarefas mais simples.

Com essa intenção, tomou-se algumas decisões no que se refere aos controles dos jogos:

- Nos três jogos existe uma zona neutra, onde os movimentos feitos pelos pacientes não afetarão em nada os personagens. Ela existe com o objetivo de fornecer aos usuários uma posição de descanso dentro do jogo, onde o jogador simplesmente não se movimenta, para que não se sintam obrigados a sempre estar jogando. Como se pudessem “tirar as mãos do controle” por alguns

segundos. Essa zona corresponde a 10% da amplitude total de movimento no Floating Board e no Tricor;

- No City Walk esse valor é maior, 90%. O motivo é, como não existem outras posições além de “Andando em pé”, “Andando agachado” e “Pulando” não faz sentido existir posições de controle intermediárias. Dessa forma, incentiva-se o paciente, repetidas vezes, a tentar se esforçar até um pouco menos que sua máxima amplitude. O objetivo é obter certo conforto para essas posições ao longo do tempo. Existem bons intervalos entre cada ação, por esse motivo tem-se prédios bem largos.
- Já nos outros dois jogos existem posições intermediárias pois o personagem efetivamente se desloca para um lado e para o outro da tela. A fim de manter a existência de uma posição de conforto, nesses jogos o paciente controla o sentido e a intensidade com que o jogador se move. A intensidade é controlada pela distância da posição atual em relação ao “zero”. Dessa forma se o usuário deseja se movimentar mais rápido, ele deve distanciar seu pé da região central.

Essas decisões evitam que movimentos bruscos sejam necessários já que o paciente pode se posicionar com antecedência conforme a situação atual no jogo sem ter que se preocupar com velocidade, estimulando assim sua capacidade cognitiva. Além disso, devido a aleatoriedade de posições e cenários os jogos ficam mais interessantes e os controles menos maçantes e fatigantes.

4.3. Comunicação e Interpretação

Toda entrada de informação fornecida pelo usuário precisa ser verificada, manipulada e enviada ao jogo. Ademais, é necessário que tudo que ocorre no jogo seja também interpretado pela inteligência artificial e transmitido a um lugar capaz de interpretar essas informações e definir qual a melhor forma de ajudar o paciente naquele instante.

Para isso desenvolveu-se uma rede de comunicação entre os principais softwares e hardwares desse projeto. Decidiu-se centralizar a maior parte da interpretação e manipulação de dados no Matlab de forma a facilitar a rastreabilidade da informação. Além disso, os

protocolos de comunicação utilizados são Comunicação Serial e UDP. Importante dizer, parte dessa rede foi desenvolvida em conjunto com Gustavo Vasconcelos (VASCONCELOS, 2023).

Simplificadamente, esse é o fluxo dos dados: o encoder recebe a informação de posição do paciente; esta é enviada pelo Arduino para o Matlab; o Matlab interpreta, filtra e manipula esses valores e os envia para o Unity; este faz com que o personagem se mova, então ele “mostra” para o Matlab os principais acontecimentos dentro do jogo e, junto a isso, envia os comandos sugeridos pela rede neural já treinada; o Matlab é, então, capaz de utilizar essas informações para julgar a performance do paciente. Futuramente, ele decidiria com base nessa análise a necessidade ou não de entrar em ação; quando o motor exercesse certa força o encoder conseguiria ler a posição para a qual o motor movimentou o paciente, e o ciclo continuaria.

4.3.1 Arduino - Matlab

Sendo mais específico, quando o Arduino é alimentado com energia, o encoder já resetou seu “zero”, e a partir de duas entradas digitais (podem também ser PWM) ele envia sua atual posição ao IDE Arduino por comunicação serial, e o mesmo a interpreta através da biblioteca Encoder.h. Para o valor enviado cada 4000 unidades, aproximadamente, representam 360 graus de rotação, visto que é um encoder rotativo de 1000 pulsos por rotação.

Essa informação é atualizada a cada 5 ms (o que pode ser visto no Apêndice A, Código 1; ou, para o caso de controle por parte da IA, no Apêndice A, Código 10), e no Matlab se lê novas informações a cada 20 a 30 ms. Definiu-se assim para que o Matlab leia sempre 5 novas posições, de forma a minimizar erros de leitura dos equipamentos e para desconsiderar movimentações muito pequenas realizadas pelo usuário, como tremedeiras). Lê-se sempre os últimos 5 valores e calcula-se a média e o desvio padrão desse conjunto, posteriormente caso o valor mais recente esteja fora dos limites estabelecidos pelo desvio padrão, ele é substituído pela média das últimas 5 posições, a qual é enviada ao jogo.

Devido ao grande volume de informações enviadas e à velocidade necessária, a porta serial usada deve ter *baud rate* de no mínimo 19200 bps (bits por segundo), que é sua velocidade de transmissão. Por isso, também, é necessário limpar o *buffer* da porta serial antes de cada leitura do Matlab, já que não é garantia que o conjunto de binários mais atual na porta serial corresponda ao valor correto. Assim, foi utilizado um caractere separador, o “!”, e após a limpeza do *buffer* se procura por esse caractere como forma de validação do número recebido.

Caso ele não seja encontrado na última posição, o número atual é desconsiderado, e lê-se o próximo valor.

O valor que passou no teste é então verificado para que não seja um NaN (do inglês, *Not a Number*, não é um número), ou seja um conjunto de binários que não representa um número. O valor também é salvo em uma matriz para ser analisado posteriormente, e adicionado ao vetor que representa as últimas cinco posições, para então ocorrer os cálculos de média e desvio padrão já mencionados. Esse procedimento pode ser observado na Seção %%Arduino no Apêndice A, Códigos 2, 3 e 4 (para os jogos Tricor, Floating Borad e City Walk, respectivamente). Há também uma adaptação desse código para a versão com tela de calibração (Apêndice A, Código 9), e para a versão em que a IA têm o controle do personagem (Apêndice A, Código 11).

4.3.2 Matlab – Unity

O código escrito no Matlab utiliza do *loop* estilo *while* para fazer as devidas conexões, ou seja, tudo ocorre em uma só *thread* (o envio e o recebimento não ocorrem ao mesmo tempo), por isso a velocidade dessa conexão é, assim como a anterior, de 20 a 30 ms. Implementou-se também um *time out* (tempo limite de espera para conexão ou reconexão) de 12 s para conexão com o Unity, medida importante para evitar a perda de dados caso ocorra um erro durante a operação do programa.

Antes de receber os dados do Arduino, se recebe um vetor de informações do Unity através de um Protocolo de Datagrama do Usuário (UDP, do inglês), tipo de comunicação voltada para a velocidade, mas com menor confiabilidade, que é ideal para o caso.

O vetor em questão contém as informações relevantes para cada jogo. Usualmente, pontuação, posição do *player*, posição de elementos na tela (como obstáculos, recompensas, plataformas, etc.), e modo de jogo. O modo de jogo é uma solução encontrada para gerenciar a adição de janelas de calibração aos jogos. Isso ajuda a informar o Matlab quando ele deve enviar a posição atual do encoder para ser gravada na etapa de calibração e quando realmente o jogo começa, a fim de interpretar melhor os dados gerados ao longo da atividade.

As informações são gravadas em um arquivo Excel a cada 120 atualizações e a cada 5 mil atualizações se adiciona mais esse tanto de linhas à matriz do Matlab, uma técnica de pré-alocação com o fim de otimizar o fluxo do programa. No final do programa também se salva

um arquivo completo com todas essas informações. Após a recepção dos dados do Arduino, utiliza-se uma outra porta de conexão UDP para enviar a posição atual ao jogo.

Quanto a parte correspondente ao Unity, é importante notar que a comunicação com o Matlab é realizada numa *thread* separada ao programa principal do jogo, a fim de possibilitar uma maior velocidade de conexão (ou seja, menor latência), ao mesmo tempo que o jogo não perde qualidade, no caso, *frames* (quadros de vídeo) por unidade de tempo. Alguns dos jogos necessitam mais dessa medida que outros, mas ela é benéfica em ambos os casos.

Para enviar os dados corretos de cada jogo ao Matlab é necessário realizar certo tratamento. Eles foram coletados dos outros *scripts* (códigos de programação) dos jogos, foram convertidos em valores numéricos para facilitar a interpretação, mas eles vêm em números reais e inteiros. Para maior unificação no tratamento, os números reais foram transformados em inteiros após serem multiplicados por 100 (de forma a levar em consideração seus algarismos após a vírgula). Então todos eles são transformados em uma *string* (sequências de caracteres alfanuméricos) com separadores (“!”) que poderão ser identificados pelo Matlab, e eventualmente restaurados a sua forma original e analisados.

O *script* referente a essa comunicação é preparado para lidar com erros e perda de conexão. Ademais, ele só é encerrado quando uma sessão de jogo é finalizada (isso deve ser adicionado pelo programador para evitar consumo excessivo de recursos), ou seja, continua funcionando em pausas ou recomeços de jogos. Ele está apresentado no Apêndice A, Código 5, para o jogo Tricor, e nos Códigos 6 e 7 para os outros dois jogos.

4.4. Implementação da Inteligência Artificial

O jogo deve ser capaz de mostrar, a todo momento, a melhor sequência de comandos a serem dados. Para isso, independentemente da aleatoriedade dos jogos, a IA deve descobrir a melhor sequência de ações, ou seja, aquela que oferece a maior pontuação.

4.4.1 Técnicas Escolhidas

Para alcançar a melhor pontuação independentemente da aleatoriedade e pela facilidade de implementação escolheu-se a técnica de Aprendizado Profundo por Reforço (DRL, do inglês), utilizando-se o algoritmo de Otimização de Política Proximal (PPO), através do ML-

agents. Além disso, optou-se por utilizar de Aprendizado por Imitação (*Imitation Learning*), juntamente com Aprendizado Curricular (*Curriculum Learning*).

Assim, processo de implementação da IA, aconteceu da seguinte forma. Primeiro, o jogo foi modificado de forma a ter sua dificuldade reduzida (isso não afeta os controles), e a intenção é que o cenário fosse ficando cada vez mais difícil, para que o *Agent* (agente, aquele que está sendo treinado) se adapte aos poucos a cada nova dificuldade. Então se insere os chamados “dados de demonstração de especialistas”, aqui isso apenas significa que um usuário humano (suscetível a erros) que sabe jogar o jogo bem, teve uma seção de jogos gravada. Ou seja, são informações que, de certa forma, contêm a estratégia do jogador.

Isso ajudará a IA a se desenvolver muito mais rápido, entretanto, são dados sub-ótimos, visto que inevitavelmente conterão erros. A intenção é então criar uma rede neural que ultrapasse o ser humano no desempenho nos jogos. Essa rede (um arquivo .onnx) poderá agir em segundo plano, quando o *player* estiver jogando, com o objetivo de ajudá-lo.

4.4.2 Implementação com ML-agents

Esse tipo de técnica (DRL) se baseia em punições e recompensas como forma de incentivar o agente a ter um determinado comportamento. Para isso, é importante delimitar bem o espaço de treinamento para que não ocorra nenhum tipo de movimento que deveria ser impossível.

Atribui-se sensores que se acredita serem relevantes ao agente, eles devem dar uma noção completa de espaço. Tudo aquilo que um jogador humano prestaria atenção é então adicionado. Dependendo do tipo de resultado esperado, pode-se ter mais ou menos sensores associados, mas tudo depende de um bom ambiente de treinamento, da capacidade computacional e do tempo disponível.

A interface com o *Inspector* do Unity possibilita que o *Agent* utilize ações discretas ou contínuas, isto é, os *outputs* (saída do programa) são valores inteiros ou qualquer valor real entre -1 e 1, respectivamente. Para ações mais simples é mais fácil e mais indicado utilizar ações discretas, isso foi feito nesse projeto como forma de tentar minimizar o espaço de ações possíveis e o tempo de treinamento. Utilizou-se 3 estados para todos os jogos, um deles para ficar parado, e os outros dois para se mover nos dois sentidos.

As ações devem ser programadas para ocorrerem com a mesma liberdade de movimentação de um jogador real, ou seja, sem limitações que não existem no jogo para evitar

uma rede neural inutilizável. O programa, aleatoriamente, envia um dos possíveis valores dependendo do tipo de ação escolhida, e conforme as recompensas que recebe vai “descobrir” a melhor solução. Um exemplo de *script* com a implementação do ML-agents pode ser observado no Apêndice A, Código 12.

Deve-se, no entanto, tomar cuidado durante as configurações do treinamento (arquivo .yaml), escolhendo as opções mais adequadas a situação em questão.

4.4.3 Configurações de aprendizado

Na tabela 1 estão presentes as principais variáveis a serem definidas, o intervalo recomendado para cada uma, e o valor utilizado nesse projeto. Importante dizer que esses valores foram definidos após dezenas de testes, mas que não necessariamente representam o ideal para a situação.

Tabela 1 – Variáveis de Configuração

Nome	Faixa Recomendada	Descrição
<i>gamma</i>	0.8~0.995	Fator de desconto: determina o quão importante são as recompensas futuras para o agente. Um valor de 0 significa que o agente só se importa com recompensas imediatas, enquanto um valor de 1 significa que ele considera todas as recompensas futuras igualmente importantes.
<i>beta</i>	1e-4~1e-2	Coefficiente de entropia: controla o quão aleatórias são as decisões do agente. À medida que o agente aprende, a entropia diminui, então diminuir o valor de beta pode ajudar a reduzir a aleatoriedade das decisões.
<i>buffer size</i>	2048~4,096e+5 (múltiplo do <i>batch size</i>)	Tamanho do <i>buffer</i> de experiências: determina quantas experiências o agente armazena antes de começar a aprender com elas.
<i>batch size</i>	contínuo: 512~5120; discreto: 32~512;	Tamanho do lote: o número de experiências que o agente usa em cada iteração do gradiente. Um tamanho maior pode levar a um treinamento mais estável e preciso, mas também pode tornar o treinamento mais lento.
<i>lambda</i>	0.9~0.95	Fator de confiança: determina o quanto o agente confia no valor atual ao calcular o valor estimado. Um valor mais alto faz com que o agente confie mais no valor atual.
<i>num_epoch</i>	3~10	O número de passagens pelo buffer de experiência durante a descida do gradiente. Mais passagens podem

Nome	Faixa Recomendada	Descrição
		levar a um aprendizado mais estável, mas também podem tornar o treinamento mais lento.
<i>learning_rate</i>	1e-5~1e-3	Taxa de aprendizado: determina o quão rápido o agente ajusta seus pesos com base no gradiente. Reduzir a taxa de aprendizado pode ajudar a tornar o aprendizado mais estável.
<i>epsilon</i>	0.1~0.3	Divergência aceitável de políticas: determina o quão diferente as novas políticas podem ser das antigas. Um valor menor pode tornar o aprendizado mais estável, mas também mais lento.
<i>normalize</i>	-	Normalização das entradas: pode ser útil para problemas mais complicados e contínuos, ajudando o agente a aprender de forma mais eficaz.
<i>num_layers</i>	1~3	Número de camadas ocultas: determina a profundidade da rede neural. Mais camadas podem ajudar a resolver problemas mais complexos.
<i>hidden_units</i>	32~512	Número de unidades ocultas: determina o número de nós em cada camada oculta. Mais unidades podem ajudar a resolver problemas mais complexos.
<i>time_horizon</i>	32~2048	Horizonte de tempo: determina quantos passos o agente coleta antes de adicionar ao <i>buffer</i> de experiência. Deve ser grande o suficiente para capturar todos os comportamentos importantes.
<i>gail:strength</i>	0.01~1.0	Força do GAIL: fator multiplicativo aplicado à recompensa bruta. Um valor baixo pode ser adequado se o agente estiver sendo treinado com recompensas extrínsecas.
<i>gail:use_vail</i>	-	Força o discriminador a aprender uma representação mais geral, o que pode tornar o aprendizado mais estável, mas também aumentar o tempo de treinamento.
<i>gail:learning_rate</i>	1e-5~1e-3	Taxa de aprendizado do GAIL: determina o quão rápido o discriminador é atualizado. Deve ser reduzido se a perda do GAIL for instável.
<i>BC:steps</i>	-	Número de passos em que o agente deve começar a se concentrar menos no <i>behavioral cloning</i> e mais na aprendizagem por reforço.

Fonte: “Training with Proximal Policy Optimization - gsrjzcx/ML-agents - GitHub”,
[s.d.].

As principais preocupações são, definir um *time_horizon* adequado, número de passos suficiente para o desenvolvimento do aprendizado; um *batch_size* e um *buffer_size* otimizados

para um treinamento mais rápido e com resultados menos instáveis; um *gamma* que corresponda à forma como se deseja que o agente aprenda (planejador ou mais reativo); e um *beta* para definir melhor o nível de aleatoriedade necessária nas decisões.

Ademais, todos esses pontos devem ser equilibrados também juntamente com as variáveis que manipulam o processo de aprendizado por imitação. São elas: *strength* e *gamma* para *gail*, *extrinsic* e *behavioral_cloning*; e *gail use_vail*. Explicando: Behavioral Cloning (BC) ou Clonagem Comportamental, no ML-agents se trata de um termo cujo *strength* (força) representa a que nível se deseja que o agente priorize mais a demonstração em questão ou as recompensas que ele obteve.

Já GAIL (Generative Adversarial Imitation Learning) é uma sigla para Aprendizado por Imitação Generativa Adversarial, seu fator *strength* também multiplica a recompensa crua, mas sua principal função é fornecer certa aleatoriedade aos dados de demonstração, enganando o discriminador que decide se esses vêm da base de dados ou se foram gerados no momento.

A intenção aqui é treinar o *Agent* para se comportar de forma semelhante ao jogador humano, e para isso é necessário aumentar o *strength* do *behavioral_cloning*, depois se deseja fazê-lo melhorar através do aumento do *strength* do *gail* e do *extrinsic rewards*. Para ajudar nisso determina-se *use_vail* como verdadeiro para forçar uma maior aleatoriedade nos dados gerados pelo GAIL. Um exemplo desse tipo de arquivo de configuração pode ser visto em Apêndice A, Código 8.

4.4.4 Treinamento

Usualmente o treinamento de um agente é realizado dentro do ambiente do Unity, ao se deixar o modo *Default* no *Inspector* após se ter criado várias cópias do modelo principal dentro do espaço 2D ou 3D da cena, podendo dessa forma fazer um treinamento mais rápido. No entanto, para se evitar uma extensa modificação do código a fim de tornar isso possível, existe um outro método. Se trata de realizar a compilação de servidor do jogo (chamado, comumente de *build server*), gerando um executável que pode, então, ser utilizado dentro do Python (onde roda o treinamento do ML-agents) para gerar vários ambientes virtuais de teste. O limite para máquina utilizada nesse projeto é de 34 ambientes.

Para iniciar o treinamento, é necessário já ter baixado e configurado um ambiente virtual. No caso presente, foi necessário baixar e instalar: ML-agents (*Release 21*), Python (versão 3.10.12), Numpy, CUDA (12.1), C++ Builder, e Pytorch (2.2.1). Além disso é

necessário atualizar o pip no ambiente virtual, esse deve ser criado, de preferência, dentro da pasta do Unity do jogo, e todos esses softwares devem ser instalados nessa pasta. Importante: foram necessárias algumas mudanças no código de *setup.py* das pastas *ml-agents* e *ml-agents-envs*, para permitir que as versões citadas pudessem trabalhar em conjunto (esse é um erro do próprio ML-agents).

Após realizados esses passos deve-se abrir o ambiente virtual em questão na pasta do jogo e digitar uma linha parecida com essa:

```
ML-agents-learn config/tricorB_1.yaml --initialize-from=Tricor_D_1 --  
env=_Builds/Pistas_1/Pistas --num-envs=34 --run-id=Tricor_D_2
```

Separando por seções:

- ML-agents-learn: é a função para iniciar um treinamento;
- config/tricorB_1.yaml: caso deseje adicionar uma configuração própria, essa é a localização do arquivo .yaml dentro da pasta do jogo;
- --initialize-from=Tricor_D_1: caso queira continuar um treinamento anterior, deve-se fornecer a ID do treinamento, que também é o nome da pasta que contém os resultados do referido treinamento;
- env=_Builds/Pistas_1/Pistas: essa é a referência do arquivo executável para quando se deseja utilizar uma *server build* para rodar o treinamento;
- --num-envs=34: diz o número de ambientes criados com o arquivo do item anterior;
- --run-id=Tricor_D_2: o comando para rodar o treinamento com um nome especificado pelo usuário, para facilitar o monitoramento.

Assim que o treinamento começa a ocorrer é importante verificar seu andamento. Para isso, existe a ferramenta do TensorBoard. Para acessá-la é necessário criar um segundo ambiente virtual dentro da pasta do jogo utilizando o seguinte comando:

```
tensorboard --logdir results
```

Então, o programa será executado na máquina em questão, para acessá-lo deve-se ir ao navegador e digitar: <http://localhost:6006/>. Nesse programa estarão disponíveis vários gráficos dentro da aba “Escalares”. Alguns deles e seus significados estão na tabela 2.

Tabela 2 – Gráficos Relevantes para Análise

Entropia (<i>Entropy</i>)	Quão aleatórias são as decisões do modelo. Devem diminuir lentamente durante um processo de treinamento bem-sucedido.
Taxa de Aprendizagem (<i>Learning Rate</i>)	Quão grande é o passo que o algoritmo de treinamento dá ao procurar pela política ideal. Deve diminuir com o tempo.
Recompensa Extrínseca (<i>Extrinsic Reward</i>)	Recompensa cumulativa média recebida do ambiente por episódio.
Estimativa de Valor (<i>Value Estimate</i>)	Estimativa do valor médio para todos os estados visitados pelo agente. Deve aumentar com o tempo.
Recompensa GAIL (<i>GAIL Reward</i>)	Recompensa cumulativa média baseado em discriminador gerada por episódio.
Estimativa de Valor GAIL (<i>GAIL Value Estimate</i>)	Estimativa de valor do agente para a recompensa GAIL
Estimativa de Política GAIL (<i>GAIL Policy Estimate</i>)	Estimativa do discriminador para estados e ações geradas pela política.
Estimativa do especialista GAIL (<i>GAIL Expert Estimate</i>)	Estimativa do discriminador para estados e ações extraídas de especialistas.
Perda de Política (<i>Policy Loss</i>)	A magnitude média da função de perda de política. Correlaciona-se com o quanto a política (processo de decisão de ações) está mudando. Deve diminuir com o tempo.
Perda de Valor (<i>Value Loss</i>)	Perda média de atualização da função de valor. Correlaciona-se com o quão bem o modelo é capaz de prever o valor de cada estado. Isso deve aumentar enquanto o agente está aprendendo e diminuir quando a recompensa se estabilizar.
Perda de Pré-treinamento (<i>Pretraining Loss</i>)	A magnitude média da perda de clonagem comportamental. Corresponde ao quão bem o modelo imita os dados de demonstração.
Perda GAIL (<i>GAIL Loss</i>)	A magnitude média da perda do discriminador GAIL. Corresponde ao quão bem o modelo imita os dados de demonstração.

Fonte: “Training Configuration File - Unity ML-Agents Toolkit”, [s.d.].

Como se pode perceber, os gráficos mais importantes são de Perda de Valor que dá uma ideia em relação a capacidade de aprendizado do modelo; de Perda de Política, o qual permite verificar se está chegando perto de um processo de decisões de ações mais uniforme; de Entropia, permitindo verificar a aleatoriedade de decisões; e Recompensa Extrínseca, para ter noção de quão bem a IA está pontuando.

Foram necessários vários testes antes de definir um ambiente de treinamento com o qual se possa contar para variar apenas as configurações. As recompensas e punições principais são:

- Punição por estar com “*encoder* virtual” (criado para que a IA possa se adaptar com esse tipo de equipamento) numa posição fora dos limites estipulados na calibração. Isso foi feito observando a posição atual e verificando se a IA está promovendo movimentação na direção contrária ao zero. $(-4,0)$;
- Pequena recompensa por voltar aos limites $(+0,4)$;
- Recompensa por se manter na pista de cor certa $(+4,0)$;
- Punição por se manter na pista de cor errada $(-4,0)$;
- Recompensa por encontrar moeda $(+20,0)$;
- Punição por colidir com obstáculo $(-20,0)$.

Após definidas essas recompensas e o máximo de passos por jogo, de acordo com o desempenho dos treinamentos, e obtidos dados de especialista melhores e mais volumosos foi possível focar na manipulação das variáveis de configuração de treinamento.

4.4.5 IA ajudando

A intenção é criar um segundo personagem “fantasma” controlado pela IA que se movimenta num plano invisível ao paciente. Esse personagem não interagiria com o jogo, apenas responderia aos estímulos do ambiente enviando sinais que representam a movimentação “ideal” naquele cenário. Esses sinais seriam convertidos nas posições ideais, então, estipulara-se um erro máximo de posição angular a que o paciente pode chegar. Caso a posição atual superasse esse erro, a IA passaria então a controlar o motor de forma a levá-lo para a posição ideal.

No entanto, não foi possível finalizar essa etapa no período estipulado, dessa forma, aplicou-se o modelo de inteligência artificial treinado a uma versão incompleta do Tricor, sem obstáculos. Nesse caso, a saída de comando gerado pela IA foi enviada através da matriz recebida pelo Matlab para o motor de forma a controlá-lo, assim o modelo se torna capaz de “jogar sozinho”.

5 RESULTADOS

Nos testes feitos sem *imitation learning*, há dificuldade de o modelo entender a presença dos limites do mapa, o que o faz tomar a decisão de escolher se movimentar para apenas um lado em todas as partidas, tornando sua pontuação altamente aleatória.

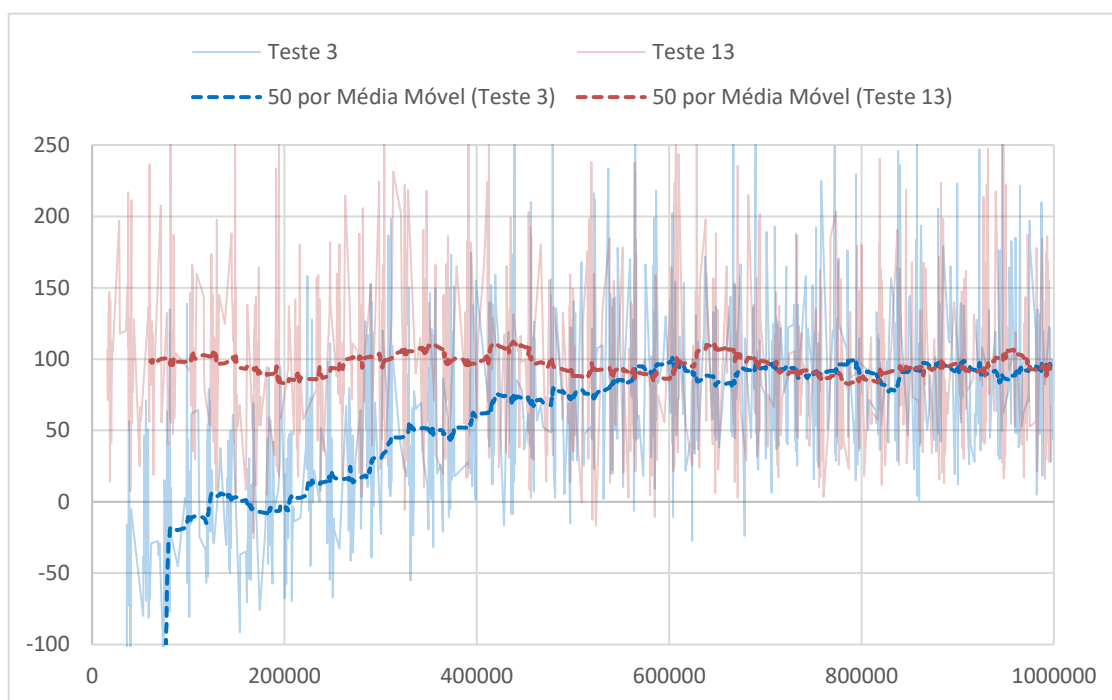
Com *imitation learning* é comum ver uma melhora enorme na pontuação nas primeiras iterações. Em vários dos treinamentos, principalmente naqueles com grande aleatoriedade, é bem difícil obter uma Entropia, Perda de Valor e Perda de Política que sempre apresentam tendência de diminuir, é muito comum haver bastante ruído para esses três gráficos. No entanto, independentemente das configurações adotadas, chega-se a um limite médio máximo de recompensas possíveis. Com a ajuda de *curriculum learning* é possível aumentar essa média aos poucos, mas isso também tem um limite.

Entretanto, há bastante variação na recompensa geral (desvio padrão maior que 40 em todas as ocasiões) devido às várias condições aleatórias que afetam o treinamento da IA. Algumas delas são: condições de calibração de acordo com o paciente e sua velocidade de deslocamento; a aleatoriedade do próprio mapa, e futuramente, relativa à presença de obstáculos.

Por fim, conseguiu-se treinar um modelo capaz de jogar Tricor (sem os obstáculos) bem, muito decisivo, com pouco atraso na decisão de ações, com bastante consistência na forma de agir, e obtendo, na maioria das vezes boas pontuações dentro do jogo. Esse modelo foi alcançado com apenas duas etapas de 1 milhão de passos cada. Inicialmente adotou-se uma configuração de forças (respectivamente: *extrinsic*, *gail*, *behavioral cloning*) 1,0; 0,5; 0,5. E posteriormente 1,0; 0,1; 0,1. Nessa segunda fase com *use_vail* ativo possibilitando maior exploração de possibilidades. Esse modelo partiria para treinamento com obstáculos.

Para os resultados desse modelo tem-se o Gráfico 1 que apresenta a recompensa geral. É importante ter em mente que o teste 13 é uma continuação do teste 3, portanto houve um aumento claro dessa pontuação inicialmente, até aproximadamente 600 mil passos, e posteriormente, verifica-se uma pontuação de aproximadamente 96 com desvio padrão de 53, destacando uma consolidação, mas que devido a grande aleatoriedade de cenários não conseguiu melhorar mais.

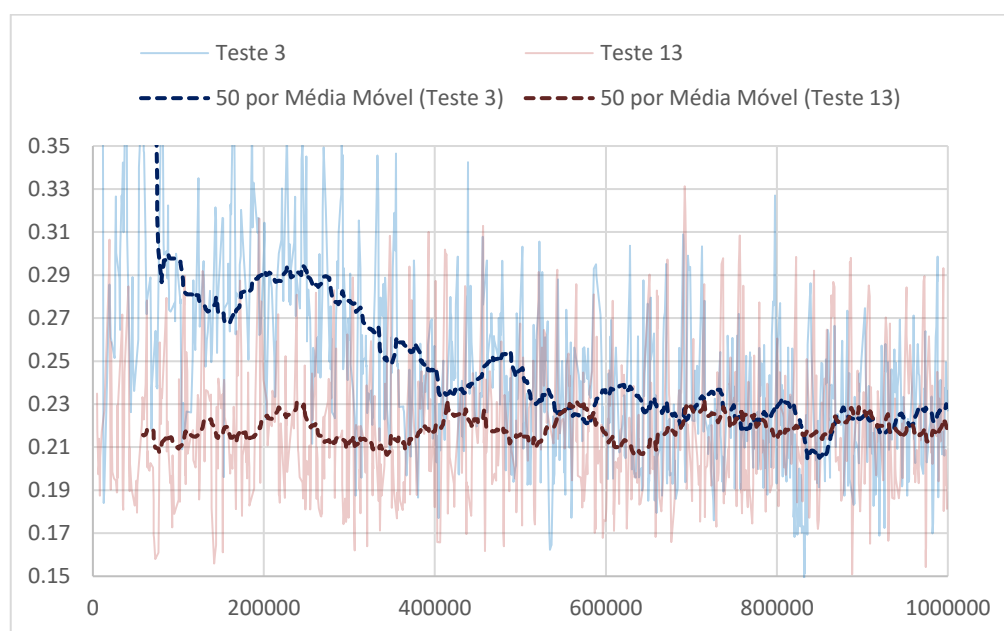
Gráfico 1 – Recompensa cumulativa ao longo do tempo



Fonte: Elaborado pelo autor, 2024.

No que se refere a entropia do modelo, percebe-se uma clara queda dos valores que depois se estabiliza, mas ainda com grande variação, destacando a grande aleatoriedade de decisões do modelo.

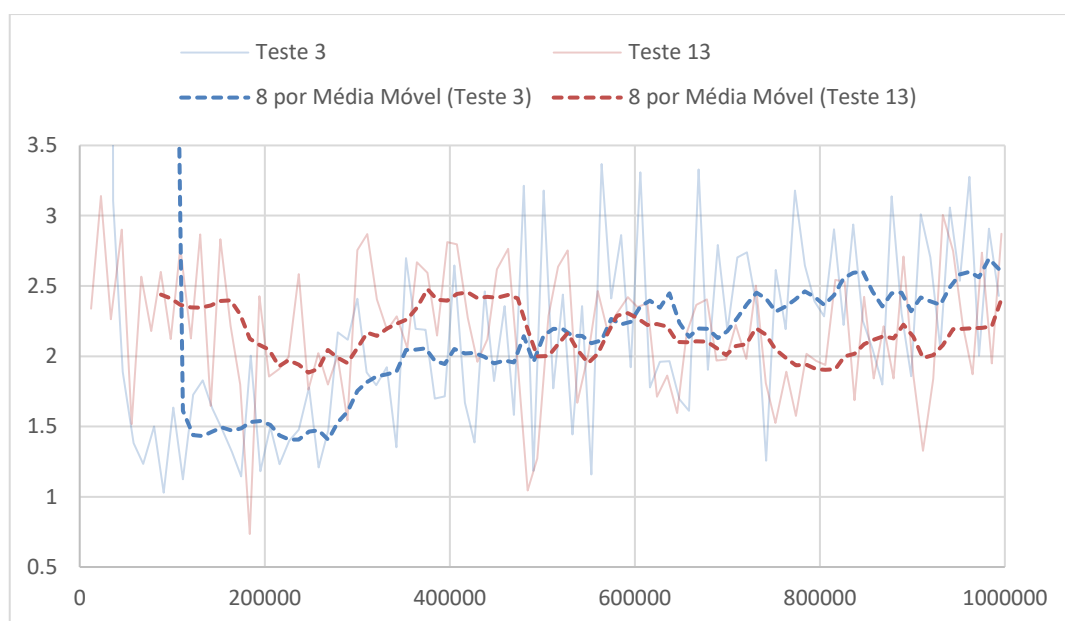
Gráfico 2 –Entropia ao longo do tempo



Fonte: Elaborado pelo autor, 2024.

Ao se tratar da perda de valor, nota-se a quantidade de ciclos de aprendizado e consolidação ao longo do tempo, no entanto, predomina um lento crescimento no teste 3 e uma certa estagnação no teste 13, demonstrando que apesar dos bons resultados práticos, ainda não se configura como o modelo ideal para a situação em questão.

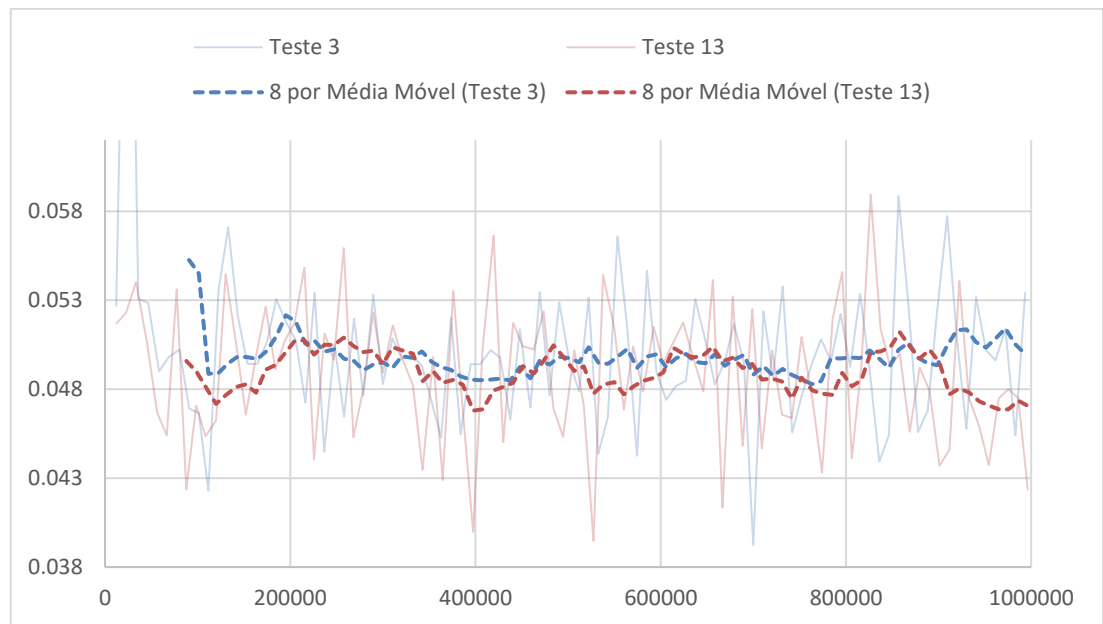
Gráfico 3 –Perda de valor ao longo do tempo



Fonte: Elaborado pelo autor, 2024.

Analisando-se a perda de política ao longo do tempo, nota-se uma queda extremamente lenta ao longo dos dois testes, que apesar de representar algo positivo, que o processo de decisão de ações está se consolidando com o tempo, devido à aleatoriedade apresentada há dúvida com relação à essa afirmação.

Gráfico 4 –Perda de política ao longo do tempo



Fonte: Elaborado pelo autor, 2024.

6 DISCUSSÃO

Neste projeto foram desenvolvidos 3 jogos diferentes, funcionais e interativos. O sistema é capaz de reter toda a informação referente a movimentação do paciente e das condições do jogo, facilitando uma posterior análise. Um dos jogos possui janela de calibração que pode ser facilmente replicada para os demais.

Com base no que foi mencionado em Resultados foi possível criar uma Inteligência Artificial capaz de jogar Tricor sozinha, apesar disso, visto os gráficos apresentados, há espaço para grande melhora no método de treinamento. Isso é destacado pelo fato de que, no tempo disponível, não houve sucesso para o treinamento desse modelo quando há a adição do sistema de barreiras do Tricor.

Por outro lado, o modelo obtido apresentou utilidade para seu propósito, jogar independentemente, o que pode ser interpretado como assistência total e poderá ser totalmente aproveitado para a criação de um sistema de assistência parcial ao paciente. Conforme dito no item 4.4.5, esse resultado poderá ser assumido como o curso ideal do *encoder* e comparado ao curso desenhado pelo paciente. Caso esses deslocamentos tenham uma distância maior que um certo erro aceitável, o motor entraria em ação.

Devido à dificuldade de usar o motor Bosch em questão em condição de livre rotação, seria interessante empregar um outro motor com essa função. Pois dessa forma, com a adição de uma leitura de torque acoplada ao sistema seria possível entender o esforço aplicado pelo paciente no momento atual e prever sua capacidade de alcançar tal posição em tal instante. Isso deixaria o sistema bem mais interativo e replicaria melhor o papel do profissional de reabilitação.

Finalmente, para o pleno funcionamento dos jogos no contexto discutido, deve-se unir as duas principais versões do jogo discutidas: o jogo completo sem intervenção de IA; e a versão que possui esse segundo jogador que age como exemplar de movimentação.

7 CONCLUSÕES

A partir dos resultados preliminares e das discussões apresentadas, provou-se a capacidade de criação de jogos sérios voltados a reabilitação do tornozelo humano por meio da assistência de inteligência artificial.

Com as futuras modificações sugeridas, o projeto poderia ser aplicado em testes na área de fisioterapia, juntamente a uma estrutura robótica correspondente. Dessa forma, este trabalho reforça a importância da pesquisa, nas áreas de estruturas robóticas e inteligência artificial, voltada para o desenvolvimento de opções mais personalizadas e efetivas não só na reabilitação, como também na área de saúde como um todo.

8 REFERÊNCIAS

- Dimbo Font | dafont.com.** Disponível em: <<https://www.dafont.com/dimbo.font>>. Acesso em: 9 abr. 2023.
- El Blog de Jordi: La MÁQUINA que aprende como un HUMANO.** Disponível em: <<https://myblogjordittrives.blogspot.com/2015/12/la-maquina-que-aprende-como-un-humano.html?spref=pi>>. Acesso em: 14 abr. 2023.
- Fantasy Chibi Female Sprite Sheets Pixel Art - CraftPix.net.** Disponível em: <<https://craftpix.net/product/fantasy-chibi-female-sprite-sheets-pixel-art>>. Acesso em: 15 out. 2023.
- File Licenses - CraftPix.net.** Disponível em: <<https://craftpix.net/file-licenses/>>. Acesso em: 20 abr. 2023.
- Fredi - YouTube.** Disponível em: <<https://www.youtube.com/@FredDev>>. Acesso em: 3 abr. 2023.
- FREE Skybox Extended Shader | VFX Shaders | Unity Asset Store.** Disponível em: <<https://assetstore.unity.com/packages/vfx/shaders/free-skybox-extended-shader-107400>>. Acesso em: 9 abr. 2023.
- GONÇALVES, R. S.; KREBS, H. I. MIT-Skywalker: Considerations on the Design of a Body Weight Support System. **Journal of NeuroEngineering and Rehabilitation**, v. 14, n. 1, 6 set. 2017.
<https://www.elearnuk.co.uk/uploads/courses/168.pdf>.
- HU, R.; LAURIÈRE, M. Recent Developments in Machine Learning Methods for Stochastic Control and Games. 17 mar. 2023.
- JANIESCH, C.; ZSCHECH, P.; HEINRICH, K. Machine learning and deep learning. **Electronic Markets**, v. 31, n. 3, p. 685–695, 1 set. 2021.
- Making An Infinite Runner like Canabalt in Unity - YouTube.** Disponível em: <<https://www.youtube.com/playlist?list=PL6ynPcXXvDY-dMil96IHEI8wvHO3MmQc1>>. Acesso em: 6 jun. 2023.
- ML-agents/docs/Training-PPO.md at master · gzejcx/ML-agents · GitHub.** Disponível em: <<https://github.com/gzejcx/ML-agents/blob/master/docs/Training-PPO.md>>. Acesso em: 21 nov. 2023.
- REPUBLIC, A. P.-C.; 2015, UNDEFINED. Architecture and implementation of the system for serious games in Unity 3D. **is.muni.cz**, [s.d.].
- SAGLIA, J. et al. A high-performance redundantly actuated parallel mechanism for ankle rehabilitation. **journals.sagepub.com**, v. 28, n. 9, p. 1216–1227, set. 2009.
- SILVA, N. Desenvolvimento de um equipamento robótico para reabilitação dos movimentos individuais do tornozelo com utilização de atuador linear. 2022.
- SUDARMILAH, E. et al. A review: Is there any benefit in serious games? **AIP Conference Proceedings**, v. 1977, 26 jun. 2018.

superjustin5000 - YouTube. Disponível em: <<https://www.youtube.com/@superjustin5000>>. Acesso em: 6 jun. 2023.

Training Configuration File - Unity ML-Agents Toolkit. Disponível em: <<https://unity-technologies.github.io/ml-agents/Training-Configuration-File/>>. Acesso em: 21 nov. 2023.

Unity - YouTube. Disponível em: <https://www.youtube.com/playlist?list=PLkhtU8XJLj-7JQm_STir4bp48DvPz8xDX>. Acesso em: 3 abr. 2023.

VASCONCELOS, G. Uso de técnicas de inteligência artificial para o desenvolvimento de jogos sérios aplicados a estrutura robótica para reabilitação do punho humano. 2023.

Warped City 2 | 2D Environments | Unity Asset Store. Disponível em: <<https://assetstore.unity.com/packages/2d/environments/warped-city-2-200208>>. Acesso em: 22 jun. 2023.

ZHANG, M.; DAVIES, T. C.; XIE, S. Effectiveness of robot-assisted therapy on ankle rehabilitation - A systematic review. **Journal of NeuroEngineering and Rehabilitation**, v. 10, n. 1, 2013.

Apêndices

Apêndice A – Códigos relacionados com a comunicação dos softwares, controle, dados e IA.

Código 1: Código do IDE Arduino de leitura do encoder e envio ao Matlab

```
#include <Encoder.h>

Encoder myEncoder(2, 3); // Substitua 2 e 3 pelos pinos conectados ao
encoder.

void setup() {
    Serial.begin(19200); // Inicializa a comunicação serial
}

void loop() {
    long encoderValue = myEncoder.read(); // Lê o valor do encoder
    String outputString = String(encoderValue) + "!";
    Serial.println(outputString); // Envia o valor pela porta serial
    //Serial.println('!'); // Adiciona o separador
    //Serial.println(); // Pula para a próxima linha

    // Aguarde um curto período de tempo para evitar sobrecarregar a porta
    serial
    delay(5);
}
```

Código 2: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo Tricor

```
clc; clear all; instrreset;

s = serial('COM3', 'BaudRate', 19200);
fopen(s); % Abre a porta serial antes do loop

% Defina os valores máximos (pMax) e mínimos (pMin) manualmente
pMax = 400; % Substitua pelo valor máximo desejado
pMin = -400; % Substitua pelo valor mínimo desejado
pula = 'pula';
agacha = 'agacha';
mantem = 'corre';
nTotal = 10; %aqui

%% Valores Encoder
% Inicializar matriz para armazenar dados do Encoder
maxRowsEnc = 5000;
rowsEncAdd = 5000;
bufferEnc = zeros(maxRowsEnc, 1); %prealocando array para mais velocidade

% Inicializa variáveis Encoder
valorAtual2 = 0;
valorAtualTrue = 0;
vAMin = 0;
vAMax = 0;
nNaN = 0;
rowCountEnc = 0; % Contador de linhas arduino
nVal = 0;
boolNVal = false;
maxNVal = 5;
buffValorAtual = zeros(maxNVal,1);

% Inicializa vetor para armazenar os últimos numLastValues valores do
% Encoder
numLastValues = 5;
lastValues = zeros(numLastValues, 1);
nFMedia = 0;

% Defina um temporizador para verificar a recepção de dados do Unity
timeoutTimer = tic;
timeoutDuration = 12; % Defina o tempo limite em segundos
playTime = true;
nTimeOut = 0;
n2TimeOut = 0;

%% UDP
% Configuração do servidor UDP no MATLAB
unityReceivePort = 35000; % Porta para receber dados do Unity
unitySendPort = 52000; % Porta para enviar dados ao Unity

udpServer = udp('127.0.0.1', unityReceivePort, 'LocalPort', unitySendPort);
set(udpServer, 'Timeout', 30);
fopen(udpServer);
```

```

% Inicializar matriz para armazenar dados
maxRows = 5000;
dataMatrix = zeros(maxRows, nTotal); %prealocando array para mais velocidade

rowCount = 0; % Contador de linhas
dataValuesInc = 0;
filename = 'dados_excel_Jogo2_calibr.xlsx'; % Nome do arquivo Excel %aqui
filenameFinal = 'dados_excel_final_Jogo2_calibr.xlsx'; % Nome do arquivo Excel
ao final da execução %aqui

% Verificar se o arquivo existe e removê-lo se existir
if exist(filename, 'file') == 2
    delete(filename);
end
if exist(filenameFinal, 'file') == 2
    delete(filenameFinal);
end

%% while
while playTime
    %%
    %% Receber dados do Unity
    if udpServer.BytesAvailable == 0
        disp('Sem dados recebidos. Verificando timeout...');
        % Verificar o tempo decorrido desde o último recebimento de dados
        elapsedTime = toc(timeoutTimer);
        if elapsedTime > timeoutDuration
            disp('Timeout atingido. Saindo do loop.');
```

playTime = false;

continue;

end

nTimeOut = nTimeOut + 1;

pause(0.5);

continue;

end

data = fread(udpServer, 1024, 'char');

% info = whos('data');

% sizeInBytes = info.bytes

dataString = char(data);

dataValues = str2double(strsplit(dataString, '!'))

%Verificar nTotal elementos em dataValues

disp(numel(dataValues))

%if numel(dataValues) == nTotal

% Incrementar o contador de linhas e verificar se excede o tamanho

prealocado

rowCount = rowCount + 1;

if rowCount > maxRows

% Se exceder o tamanho prealocado, aumente o tamanho da matriz

dataMatrix = [dataMatrix; zeros(maxRows, nTotal)]; % Adicione mais

linhas à matriz

maxRows = maxRows+5000; % Ajuste o tamanho máximo

end

if numel(dataValues) == nTotal

% Armazenar os dados na matriz prealocada

if (dataValues == zeros(1, nTotal))

dataValues = ones(1, nTotal);

```

end
dataMatrix(rowCount, :) = dataValues;

% Salvando cada conjunto de valores em uma nova linha na matriz
dataMatrix = [dataMatrix; dataValues];
else
% Se faltar valores, completar com zeros
% missingValues = nTotal - numel(dataValues);
% dataValues = [dataValues, zeros(1, missingValues)];
% dataMatrix(rowCount, :) = dataValues;
end

%aqui
pontos = dataValues(1)
playerPositX = dataValues(2)/100
scrBar = dataValues(3)
scrCoin = dataValues(4)
scrCor = dataValues(5)
scrCorB = dataValues(6)
scrCorG = dataValues(7)
scrCorR = dataValues(8)
posCor = dataValues(9)
modoJogo = dataValues(10)

%% Salvar dados em uma planilha Excel
if mod(rowCount, 120) == 0
% Salvar a cada 120 linhas
filename = 'dados_excel_Jogo2_calibr.xlsx';%aqui
writematrix(dataMatrix(1:rowCount, :), filename); % ou use writetable se
preferir trabalhar com tabelas
end

%% Verificando dados recebidos
if(modoSJ == 2 || modoSJ == 4 || modoSJ == 6)
boolModoSJ = true;
else
boolModoSJ = false;
end

if (boolModoSJ == false)
%% Tic
timeoutTimer = tic;
%% Arduino
% Leitura do Arduino

% Lê os dados disponíveis na porta serial
nVal = 0;

while s.BytesAvailable > 0
fread(s, s.BytesAvailable);
end

valorEnc = fgetl(s);

while (nVal < maxNVal)
valorEnc = fscanf(s, '%s');
if valorEnc(end) == '!'
boolNVal = false;
valorEnc = valorEnc(1:end-1);
buffValorAtual(nVal+1) = str2double(valorEnc);
valorAtual = buffValorAtual(nVal+1);

```

```

else
    boolNVal = true;
    nVal = nVal - 1;
end

%buffValorAtual = str2double(strsplit(valorEnc, '!'));

%disp(valorAtual)
if (boolNVal == false)
    % Adiciona os novos dados ao buffer
    rowCountEnc = rowCountEnc + 1;
    if rowCountEnc > maxRowsEnc
        % Se exceder o tamanho prealocado, aumente o tamanho da
matriz
        bufferEnc = [bufferEnc; zeros(rowsEncAdd, 1)]; % Adicione
mais linhas à matriz
        maxRowsEnc = maxRowsEnc + rowsEncAdd; % Ajuste o tamanho
máximo
    end

    % Armazenar os dados na matriz prealocada
    if isnan(valorAtual)
        bufferEnc(rowCountEnc) = bufferEnc(rowCountEnc-1);
        nNaN = nNaN + 1;
        %continue;
    else
        bufferEnc(rowCountEnc) = valorAtual;
    end

    % Atualiza o vetor dos últimos numLastValues valores
    lastValues(mod(rowCountEnc, numLastValues) + 1) = valorAtual;

    % Calcula a média e o desvio padrão dos últimos 5 valores
    if rowCountEnc >= numLastValues
        media = mean(lastValues);
        desvioPadrao = std(lastValues);
        vAMin = media - desvioPadrao;
        VAMax = media + desvioPadrao;
        if (valorAtual < vAMin) || (valorAtual > VAMax)
            valorAtual2 = round(media);
            nFMedia = nFMedia + 1;
        else
            valorAtual2 = valorAtual;
        end
    end

    bufferEnc(rowCountEnc) = valorAtual2;
end
nVal = nVal + 1;
end
else
    %evitamos ler posição do encoder e só mandamos a última antes de
    %mudar de modoJogo
end
% Exibe o valor atual em tempo real
disp(['Valor Atual: ' num2str(valorAtual2)]);
fprintf(udpServer, '%d', valorAtual2);
%% Processar os dados recebidos
% Lógica de movimento com base nos valores de pMax e pMin
if valorAtual2 > pMax

```

```

        disp(pula);%fwrite(tcpClient, sobe);
elseif valorAtual2 < pMin
    disp(agacha);%fwrite(tcpClient, desce);
else
    disp(mantem);%fwrite(tcpClient, mantem);
end
    %pause(0.05); % Espera 0.2 segundos (ajuste conforme necessário)

    pause(0.02);
end

filenameFinal = 'dados_excel_final_Jogo2_calibr.xlsx';%aqui
writematrix(dataMatrix(1:rowCount, :), filenameFinal);

fclose(s); % Feche a porta serial quando não for mais necessária
fclose(udpServer);
delete(udpServer);

```

Código 3: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo Floating Board

```
clc; clear all; instrreset;

s = serial('COM3', 'BaudRate', 19200);
fopen(s); % Abre a porta serial antes do loop

% Defina os valores máximos (pMax) e mínimos (pMin) manualmente
pMax = 400; % Substitua pelo valor máximo desejado
pMin = -400; % Substitua pelo valor mínimo desejado
pula = 'pula';
agacha = 'agacha';
mantem = 'corre';
nTotal = 6;

%% Valores Encoder
% Inicializar matriz para armazenar dados do Encoder
maxRowsEnc = 5000;
rowsEncAdd = 5000;
bufferEnc = zeros(maxRowsEnc, 1); %prealocando array para mais velocidade

% Inicializa variáveis Encoder
valorAtual2 = 0;
valorAtualTrue = 0;
vAMin = 0;
vAMax = 0;
nNaN = 0;
rowCountEnc = 0; % Contador de linhas arduino
nVal = 0;
boolNVal = false;
maxNVal = 5;
buffValorAtual = zeros(maxNVal,1);

% Inicializa vetor para armazenar os últimos numLastValues valores do
% Encoder
numLastValues = 5;
lastValues = zeros(numLastValues, 1);
nFMedia = 0;

% Defina um temporizador para verificar a recepção de dados do Unity
timeoutTimer = tic;
timeoutDuration = 12; % Defina o tempo limite em segundos
playTime = true;
nTimeOut = 0;
n2TimeOut = 0;

%% UDP
% Configuração do servidor UDP no MATLAB
unityReceivePort = 35000; % Porta para receber dados do Unity
unitySendPort = 52000; % Porta para enviar dados ao Unity

udpServer = udp('127.0.0.1', unityReceivePort, 'LocalPort', unitySendPort);
set(udpServer, 'Timeout', 30);
fopen(udpServer);

% Inicializar matriz para armazenar dados
maxRows = 5000;
dataMatrix = zeros(maxRows, nTotal); %prealocando array para mais velocidade

rowCount = 0; % Contador de linhas
```

```

dataValuesInc = 0;
filename = 'dados_excel_Jogo1.xlsx'; % Nome do arquivo Excel
filenameFinal = 'dados_excel_final_Jogo1.xlsx'; % Nome do arquivo Excel ao final da
execução

% Verificar se o arquivo existe e removê-lo se existir
if exist(filename, 'file') == 2
    delete(filename);
end
if exist(filenameFinal, 'file') == 2
    delete(filenameFinal);
end

%% while
while playTime
    %%
    %% Receber dados do Unity
    if udpServer.BytesAvailable == 0
        disp('Sem dados recebidos. Verificando timeout...');
        % Verificar o tempo decorrido desde o último recebimento de dados
        elapsedTime = toc(timeoutTimer);
        if elapsedTime > timeoutDuration
            disp('Timeout atingido. Saindo do loop.');
```

playTime = false;

continue;

end

nTimeOut = nTimeOut + 1;

pause(0.5);

continue;

end

data = fread(udpServer, 1024, 'char');

% info = whos('data');

% sizeInBytes = info.bytes

dataString = char(data);

dataValues = str2double(strsplit(dataString, '!'))

%Verificar nTotal elementos em dataValues

disp(numel(dataValues))

%if numel(dataValues) == nTotal

% Incrementar o contador de linhas e verificar se excede o tamanho

prealocado

rowCount = rowCount + 1;

if rowCount > maxRows

% Se exceder o tamanho prealocado, aumente o tamanho da matriz

dataMatrix = [dataMatrix; zeros(maxRows, nTotal)]; % Adicione mais

linhas à matriz

maxRows = maxRows+5000; % Ajuste o tamanho máximo

end

if numel(dataValues) == nTotal

% Armazenar os dados na matriz prealocada

if (dataValues == zeros(1, nTotal))

dataValues = ones(1, nTotal);

end

dataMatrix(rowCount, :) = dataValues;

% Salvando cada conjunto de valores em uma nova linha na matriz

dataMatrix = [dataMatrix; dataValues];

else

% Se faltar valores, completar com zeros

% missingValues = nTotal - numel(dataValues);

```

        % dataValues = [dataValues, zeros(1, missingValues)];
        % dataMatrix(rowCount, :) = dataValues;
    end
    pontos = dataValues(1)
    playerPositX = dataValues(2)/100
    playerPositY = dataValues(3)/100
    contChuva = dataValues(4)
    contLago = dataValues(5)
    posPrancha = dataValues(6)/100
    % else
    %     dataValuesInc = dataValuesInc + 1;
    %     disp('Número incorreto de elementos em dataValues');
    %     % Lógica para lidar com o erro, se necessário

    %% Salvar dados em uma planilha Excel
    if mod(rowCount, 120) == 0
        % Salvar a cada 120 linhas
        filename = 'dados_excel_Jogo1.xlsx';
        writematrix(dataMatrix(1:rowCount, :), filename); % ou use writetable se
preferir trabalhar com tabelas
    end

    %% Tic
    timeoutTimer = tic;
    %% Arduino
    % Leitura do Arduino

    % Lê os dados disponíveis na porta serial
    nVal = 0;

    while s.BytesAvailable > 0
        fread(s, s.BytesAvailable);
    end

    valorEnc = fgetl(s);

    while (nVal < maxNVal)
        valorEnc = fscanf(s, '%s');
        if valorEnc(end) == '!'
            boolNVal = false;
            valorEnc = valorEnc(1:end-1);
            buffValorAtual(nVal+1) = str2double(valorEnc);
            valorAtual = buffValorAtual(nVal+1);
        else
            boolNVal = true;
            nVal = nVal - 1;
        end

        %buffValorAtual = str2double(strsplit(valorEnc, '!'));

        %disp(valorAtual)
        if (boolNVal == false)
            % Adiciona os novos dados ao buffer
            rowCountEnc = rowCountEnc + 1;
            if rowCountEnc > maxRowsEnc
                % Se exceder o tamanho prealocado, aumente o tamanho da matriz
                bufferEnc = [bufferEnc; zeros(rowsEncAdd, 1)]; % Adicione mais
linhas à matriz
                maxRowsEnc = maxRowsEnc + rowsEncAdd; % Ajuste o tamanho máximo
            end

            % Armazenar os dados na matriz prealocada
            if isnan(valorAtual)

```

```

        bufferEnc(rowCountEnc) = bufferEnc(rowCountEnc-1);
        nNaN = nNaN + 1;
        %continue;
    else
        bufferEnc(rowCountEnc) = valorAtual;
    end

    % Atualiza o vetor dos últimos numLastValues valores
    lastValues(mod(rowCountEnc, numLastValues) + 1) = valorAtual;

    % Calcula a média e o desvio padrão dos últimos 5 valores
    if rowCountEnc >= numLastValues
        media = mean(lastValues);
        desvioPadrao = std(lastValues);
        vAMin = media - desvioPadrao;
        VAMax = media + desvioPadrao;
        if (valorAtual < vAMin) || (valorAtual > VAMax)
            valorAtual2 = round(media);
            nFMedia = nFMedia + 1;
        else
            valorAtual2 = valorAtual;
        end
    end

    bufferEnc(rowCountEnc) = valorAtual2;
end
nVal = nVal + 1;
end
% Exibe o valor atual em tempo real
disp(['Valor Atual: ' num2str(valorAtual2)]);
fprintf(udpServer, '%d', valorAtual2);
%% Processar os dados recebidos e calcular BirdControl
% Lógica de movimento com base nos valores de pMax e pMin
if valorAtual2 > pMax
    disp(pula); %fwrite(tcpClient, sobe);
elseif valorAtual2 < pMin
    disp(agacha); %fwrite(tcpClient, desce);
else
    disp(mantem); %fwrite(tcpClient, mantem);
end
%pause(0.05); % Espera 0.2 segundos (ajuste conforme necessário)

pause(0.02);
end

filenameFinal = 'dados_excel_final_Jogol.xlsx';
writematrix(dataMatrix(1:rowCount, :), filenameFinal);

fclose(s); % Feche a porta serial quando não for mais necessária
fclose(udpServer);
delete(udpServer);

% se perder conexão -> fechar (ideia)

% Configurações

```

Código 4: Código do Matlab de comunicação com o IDE Arduino e Unity2D, controle, IA, gravação e interpretação de dados referentes ao jogo City Walk

```
clc; clear all; instrreset;

s = serial('COM3', 'BaudRate', 19200);
fopen(s); % Abre a porta serial antes do loop

% Defina os valores máximos (pMax) e mínimos (pMin) manualmente
pMax = 200; % Substitua pelo valor máximo desejado
pMin = -200; % Substitua pelo valor mínimo desejado
pula = 'pula';
agacha = 'agacha';
mantem = 'corre';
%% Mudar aqui para número de informações recebidas
nTotal = 11;

%% Valores Encoder
% Inicializar matriz para armazenar dados do Encoder
maxRowsEnc = 5000;
rowsEncAdd = 5000;
bufferEnc = zeros(maxRowsEnc, 1); %prealocando array para mais velocidade

% Inicializa variáveis Encoder
valorAtual2 = 0;
valorAtualTrue = 0;
vAMin = 0; % Valor min considerando Desv.Pad.
vAMax = 0; % Valor max considerando Desv.Pad.
nNaN = 0; % Número de valores não numéricos
rowCountEnc = 0; % Contador de linhas arduino
nVal = 0; % Variável principal do loop de maxNVal
valores para encoder
boolNVal = false; % Indicador de se o valor atual é NaN
maxNVal = 5; % Número máximo de de loops para arduino
buffValorAtual = zeros(maxNVal,1); % Armazena os últimos maxNVal valores,
apenas

% Inicializa vetor para armazenar os últimos numLastValues valores do
% Encoder
numLastValues = 5; % Últimos valores considerados para cálculos
de média e Desv.Pad.
lastValues = zeros(numLastValues, 1); % Vetor para cálculo com os últimos
numLastValues valores
nFMedia = 0;

% Defina um temporizador para verificar a recepção de dados do Unity
timeoutTimer = tic; % Registra tempo atual como início
timeoutDuration = 12; % Defina o tempo limite em segundos
playTime = true; % Bool para saber se jogo está rodando
nTimeOut = 0; % Variável principal para verificar TimeOut
UDP
n2TimeOut = 0;

%% UDP
% Configuração do servidor UDP no MATLAB
unityReceivePort = 35000; % Porta para receber dados do Unity
unitySendPort = 52000; % Porta para enviar dados ao Unity

udpServer = udp('127.0.0.1', unityReceivePort, 'LocalPort', unitySendPort);
set(udpServer, 'Timeout', 30);
fopen(udpServer);
```

```

% Inicializar matriz para armazenar dados
maxRows = 5000;
dataMatrix = zeros(maxRows, nTotal); %prealocando array para mais velocidade

rowCount = 0; % Contador de linhas
dataValuesInc = 0;
filename = 'dados_excel.xlsx'; % Nome do arquivo Excel
filenameFinal = 'dados_excel_final.xlsx'; % Nome do arquivo Excel ao final da
execução

% Verificar se o arquivo existe e removê-lo se existir
if exist(filename, 'file') == 2
    delete(filename);
end
if exist(filenameFinal, 'file') == 2
    delete(filenameFinal);
end

%% while
while playTime
    %%
    %% Receber dados do Unity
    if udpServer.BytesAvailable == 0
        disp('Sem dados recebidos. Verificando timeout...');
        % Verificar o tempo decorrido desde o último recebimento de dados
        elapsedTime = toc(timeoutTimer);
        if elapsedTime > timeoutDuration
            disp('Timeout atingido. Saindo do loop.');
```

playTime = false;

continue;

end

nTimeOut = nTimeOut + 1;

pause(0.5);

continue;

end

data = fread(udpServer, 1024, 'char');

% info = whos('data');

% sizeInBytes = info.bytes

dataString = char(data);

dataValues = str2double(strsplit(dataString, '!'))

%Verificar nTotal elementos em dataValues

disp(numel(dataValues))

%if numel(dataValues) == nTotal

% Incrementar o contador de linhas e verificar se excede o tamanho prealocado

rowCount = rowCount + 1;

if rowCount > maxRows

% Se exceder o tamanho prealocado, aumente o tamanho da matriz

dataMatrix = [dataMatrix; zeros(maxRows, nTotal)]; % Adicione mais

linhas à matriz

maxRows = maxRows+5000; % Ajuste o tamanho máximo

end

if numel(dataValues) == nTotal

% Armazenar os dados na matriz prealocada

if (dataValues == zeros(1, nTotal))

dataValues = ones(1, nTotal);

end

dataMatrix(rowCount, :) = dataValues;

```

        % Salvando cada conjunto de valores em uma nova linha na matriz
        dataMatrix = [dataMatrix; dataValues];
    else
        % Se faltar valores, completar com zeros
        % missingValues = nTotal - numel(dataValues);
        % dataValues = [dataValues, zeros(1, missingValues)];
        % dataMatrix(rowCount, :) = dataValues;
    end

    distancia = dataValues(1)
    moedas = dataValues(2)
    groundAtualX = dataValues(3)/100
    groundAtualY = dataValues(4)/100
    groundNextX = dataValues(5)/100
    groundNextY = dataValues(6)/100
    velocidade = dataValues(7)/100
    eventos = dataValues(8)
    obstaculos = dataValues(9)
    pulos = dataValues(10)
    agach = dataValues(11)
% else
%     dataValuesInc = dataValuesInc + 1;
%     disp('Número incorreto de elementos em dataValues');
%     % Lógica para lidar com o erro, se necessário

%% Salvar dados em uma planilha Excel
if mod(rowCount, 120) == 0
    % Salvar a cada 120 linhas
    filename = 'dados_excel.xlsx';
    writematrix(dataMatrix(1:rowCount, :), filename); % ou use writetable se
preferir trabalhar com tabelas
end

%% Tic
timeoutTimer = tic;

%% Arduino
% Leitura do Arduino

nVal = 0; % inicializar loop

% Limpar buffer
while s.BytesAvailable > 0
    fread(s, s.BytesAvailable);
end

% Ler o primeiro valor e ignorá-lo (pois pode não ser confiável)
valorEnc = fgetl(s);

% Início do loop que pegará maxNVal valores pra evitar problema de
% tremedeira ou algo assim
while (nVal < maxNVal)
    valorEnc = fscanf(s, '%s'); % captura a string
    if valorEnc(end) == '!' % verifica se tem ! no
final
        boolNVal = false; % se sim, que dizer que
não é NaN (discutível)
        valorEnc = valorEnc(1:end-1); % ignora o '!'
        buffValorAtual(nVal+1) = str2double(valorEnc); % salva o valor como
double
        valorAtual = buffValorAtual(nVal+1); % salva em valor atual
    else
        boolNVal = true; % caso contrário = true

```

```

        nVal = nVal - 1; % nVal desconsidera o
loop atual
    end

    %buffValorAtual = str2double(strsplit(valorEnc, '!'));

    %disp(valorAtual)
    if (boolNVal == false)
        % Adiciona os novos dados ao buffer

        % Se exceder o tamanho prealocado, aumente o tamanho da matriz
        rowCountEnc = rowCountEnc + 1;
        if rowCountEnc > maxRowsEnc
            bufferEnc = [bufferEnc; zeros(rowsEncAdd, 1)]; % Adicione mais
linhas à matriz
            maxRowsEnc = maxRowsEnc + rowsEncAdd; % Ajuste o tamanho máximo
        end

        % Armazenar os dados na matriz prealocada
        if isnan(valorAtual)
            bufferEnc(rowCountEnc) = bufferEnc(rowCountEnc-1);
            nNaN = nNaN + 1;
            %continue;
        else
            bufferEnc(rowCountEnc) = valorAtual;
        end

        % Atualiza o vetor dos últimos numLastValues valores
        lastValues(mod(rowCountEnc, numLastValues) + 1) = valorAtual;

        % Calcula a média e o desvio padrão dos últimos 5 valores
        if rowCountEnc >= numLastValues
            media = mean(lastValues);
            desvioPadrao = std(lastValues);
            vAMin = media - desvioPadrao;
            VAMax = media + desvioPadrao;
            if (valorAtual < vAMin) || (valorAtual > vAMax) % se estiver fora dos
padrões
                valorAtual2 = round(media); % seta como a média dos
últimos numLastValues
                nFMedia = nFMedia + 1;
            else
                valorAtual2 = valorAtual; % caso contrário salva
em valorAtual2
            end
        end

        bufferEnc(rowCountEnc) = valorAtual2; % salva na matriz de
valores do encoder
    end
    nVal = nVal + 1;
end

% Exibe o valor atual em tempo real
disp(['Valor Atual: ' num2str(valorAtual2)]);
fprintf(udpServer, '%d', valorAtual2);

%% Processar os dados recebidos e calcular BirdControl
% Lógica de movimento com base nos valores de pMax e pMin
if valorAtual2 > pMax
    disp(pula); %fwrite(tcpClient, sobe);
elseif valorAtual2 < pMin
    disp(agacha); %fwrite(tcpClient, desce);

```

```

else
    disp(mantem);%fwrite(tcpClient, mantem);
end
%pause(0.05); % Espera 0.2 segundos (ajuste conforme necessário)

pause(0.02);
end

filenameFinal = 'dados_excel_final.xlsx';
writematrix(dataMatrix(1:rowCount, :), filenameFinal);

fclose(s); % Feche a porta serial quando não for mais necessária
fclose(udpServer);
delete(udpServer);

% se perder conexão -> fechar (ideia)

% Configurações

```

Código 5: Código do Unity2D do jogo Tricor para comunicação com o Matlab

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading;
using System.Net.Sockets;
using System.Net;
//using Codice.Client.Common;
using System.Text;
using System;
using UnityEngine.XR;

public class UDP_1 : server_abs
{
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*

    public static int ValueEncoder = 0;

    Player player;
    GameManager gameManager;

    public int playerPositX = 0;
    public int pontos = 0;
    public int scrBar = 0;
    public int scrCoin = 0;
    public int scrCor = 0;
    public int scrCorB = 0;
    public int scrCorG = 0; //Gray
    public int scrCorR = 0; //Yellow
    public int posCor = 0;
    public int gameMode = 0;
    //public int inputTec12 = 0; // -tecMot

    private Thread udpThread;
    private volatile bool _shouldStop = false;

    // Start is called before the first frame update
    void Start()
    {
        instance = this;
        player = GameObject.Find("Player").GetComponent<Player>();
    }

    void Awake()
    {
        udpClient = new UdpClient();
        matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
unitySendPort); // Endereço e porta do servidor MATLAB
        udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
Configurar a porta de recebimento do Unity
    }
}
```

```

        // Iniciar a thread para recepção assíncrona de dados UDP
        udpThread = new Thread(new ThreadStart(Comunicacao));
        udpThread.IsBackground = true;
        udpThread.Start();

    }

    // Update is called once per frame
    void Update()
    {

    }

    private void FixedUpdate()
    {
        gameMode = modoJogo;
        if (GameManager.Instance == null)
        {
            return;
        }

        pontos = (int)GameManager.Instance.pontos;
        playerPositX = (int)(player.gameObject.transform.position.x * 100);
        scrBar = errosBarreira;
        scrCoin = contadorMoedas;
        scrCor = (int)player.contadorPista;
        scrCorB = (int)player.contadorPistaB;
        scrCorG = (int)player.contadorPistaG; //gray
        scrCorR = (int)player.contadorPistaR; //yellow
        posCor = caso2;

        //inputTec12 = inputTec1; // -tecMot
    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{pontos}!{playerPositX}!{scrBar}!{scrCoin}!{scrCor}!{scrCorB}!{scrCorG}!{scrCorR}!
                {posCor}!{gameMode}";///!{inputTec12}";// -tecMot
                /**/
                //Debug.Log("Horiz.Input: " + inputTec12);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto período antes do próximo envio (evitar envio
                excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }

            // Receber dados do MATLAB
            try
            {

```

```

        IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
unityReceivePort);
        byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
        string receivedString = Encoding.ASCII.GetString(receivedData);

        // Debug para visualizar os dados recebidos no console do Unity
        Debug.Log("Dados recebidos do MATLAB: " + receivedString);

        //ValueEncoder = int.Parse(receivedString);
        // Converter para int e verificar se é NaN
        int parsedValue;
        // Tenta converter a string receivedString em um valor inteiro e
armazena o resultado na variável parsedValue.
        // O método TryParse retorna true se a conversão for bem - sucedida
e false se falhar.
        bool parseSuccess = int.TryParse(receivedString, out parsedValue);
        if (parseSuccess && !double.IsNaN(parsedValue))
        {
            ValueEncoder = parsedValue;
        }
        else
        {
            ValueEncoder = 0;
        }
    }
    catch (Exception e)
    {
        Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
    }
    Debug.Log("Thread Aqui");
}

void OnDestroy()
{
    _shouldStop = true;
}

public void Enable()
{
    gameObject.SetActive(true);
    _shouldStop = false;
    Debug.Log("Inicio");
    //stopThread = false;
}

public void Disable()
{
    _shouldStop = true;
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
    gameObject.SetActive(false);
    Debug.Log("Fim");
}
}

```

Código 6: Código do Unity2D do jogo City Walk para comunicação com o Matlab

```
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using UnityEngine;
using UnityEditor.Experimental.GraphView;
using System.Collections;
using System.Threading;
using System.Drawing;

public class UDP_1 : MonoBehaviour
{
    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*
    public float velocidade = 0.00f;
    public float groundNextX = 0.00f;
    public float groundNextY = 0.00f;
    public float groundAtualX = 0.00f;
    public float groundAtualY = 0.00f;
    public int obstaculos = 0;
    public int distancia = 0;
    public int moedas = 0;
    public int pulos = 0;
    public int agach = 0;
    public int eventos = 0;

    // Precisa por posição do obstáculo ainda
    public static int ValueEncoder = 0;

    //Threads
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    //public static bool stopThread = false;
    private Thread udpThread;
    // Volatile is used as hint to the compiler that this data
    // member will be accessed by multiple threads.
    private volatile bool _shouldStop = false;

    void Start()
    {
        {
            instance = this;
        }

        void Awake()
        {
            udpClient = new UdpClient();
            matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
            unitySendPort); // Endereço e porta do servidor MATLAB
            udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
            Configurar a porta de recebimento do Unity

            // Iniciar a thread para recepção assíncrona de dados UDP
            udpThread = new Thread(new ThreadStart(Comunicacao));
            udpThread.IsBackground = true;
            udpThread.Start();
        }
    }
}
```

```

    }

    public void OnEnable()
    {

    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.O)) // || stopThread == true
        {
            //gameObject.SetActive(false);
            //StopThreads(udpThread);
        }
    }

    private void FixedUpdate()
    {
        eventos = Player.totalEvento;
        distancia = Mathf.FloorToInt(Player.distancia);
        moedas = Player.contadorMoeda2;
        groundAtualX = (int)(Ground.posXGround * 100);
        groundAtualY = (int)(Ground.posYGround * 100);
        velocidade = (int)(Player.velocidade * 100);
        obstaculos = Player.totalHitObstacle;
        pulos = Player.totalJump;
        agach = Player.totalAgach;
        //Debug.Log(agach);
        //Debug.Log(pulos);
        // Parece q pra inteiros não precisa verificar se são nulos pq tipos de
        // valor como int não podem conter um valor null**
        //Debug.Log(stopThread);
    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{distancia}!{moedas}!{groundAtualX}!{groundAtualY}!{groundNextX}!{groundNextY}!{ve
                locidade}!{eventos}!{obstaculos}!{pulos}!{agach}";
                /**/
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto período antes do próximo envio (evitar envio
                excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }

            // Receber dados do MATLAB
            try
            {

```

```

        IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
unityReceivePort);
        byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
        string receivedString = Encoding.ASCII.GetString(receivedData);

        // Debug para visualizar os dados recebidos no console do Unity
        Debug.Log("Dados recebidos do MATLAB: " + receivedString);

        //ValueEncoder = int.Parse(receivedString);
        // Converter para int e verificar se é NaN
        int parsedValue;
        // Tenta converter a string receivedString em um valor inteiro e
armazena o resultado na variável parsedValue.
        // O método TryParse retorna true se a conversão for bem - sucedida
e false se falhar.
        bool parseSuccess = int.TryParse(receivedString, out parsedValue);
        if (parseSuccess && !double.IsNaN(parsedValue))
        {
            ValueEncoder = parsedValue;
        }
        else
        {
            ValueEncoder = 0;
        }
    }
    catch (Exception e)
    {
        Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
    }
    Debug.Log("Thread Aqui");
}

void OnDestroy()
{
    _shouldStop = true;
}

public void Enable()
{
    gameObject.SetActive(true);
    _shouldStop = false;
    Debug.Log("Inicio");
    //stopThread = false;
}

public void Disable()
{
    _shouldStop = true;
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
    gameObject.SetActive(false);
    Debug.Log("Fim");
}
}

```

Código 7: Código do Unity2D do jogo Floating Board para comunicação com o Matlab

```
using System.Collections;
using System.Collections.Generic;
using System.Net.Sockets;
using System.Net;
using UnityEngine;
using System.Threading;
using Unity.VisualScripting;
using System.Text;
using System;
using UnityEditor.PackageManager;

public class UDP_1 : MonoBehaviour
{
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*

    public static int ValueEncoder = 0;

    Player player;
    Chuva chuva;
    Platchao plataforma;

    public int playerPositX = 0;
    public int playerPositY = 0;
    public int pontos = 0;
    public int contChuva = 0;
    public int contLago = 0;
    public int posPrancha = 0;

    private Thread udpThread;
    private volatile bool _shouldStop = false;

    // Start is called before the first frame update
    void Start()
    {
        instance = this;
        player = GameObject.Find("Player").GetComponent<Player>();
        chuva = GameObject.Find("Chuva").GetComponent<Chuva>();
        plataforma = GameObject.Find("Plataforma").GetComponent<Platchao>();
    }

    void Awake()
    {
        udpClient = new UdpClient();
        matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
unitySendPort); // Endereço e porta do servidor MATLAB
        udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
Configurar a porta de recebimento do Unity

        // Iniciar a thread para recepção assíncrona de dados UDP
        udpThread = new Thread(new ThreadStart(Comunicacao));
        udpThread.IsBackground = true;
        udpThread.Start();
    }
}
```

```

    }

    // Update is called once per frame
    void Update()
    {

    }

    private void FixedUpdate()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        playerPositX = (int) (player.gameObject.transform.position.x * 100);
        playerPositY = (int) (player.gameObject.transform.position.y * 100);
        pontos = (int) GameManager.Instance.pontos;
        contChuva = chuva.contadorChuva;
        contLago = (int) player.contadorLago;
        posPrancha = (int) (plataforma.transform.rotation.eulerAngles.z * 100);
    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{pontos}!{playerPositX}!{playerPositY}!{contChuva}!{contLago}!{posPrancha}";
                /**/
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto periodo antes do próximo envio (evitar envio
                // excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }

            // Receber dados do MATLAB
            try
            {
                IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
                unityReceivePort);
                byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
                string receivedString = Encoding.ASCII.GetString(receivedData);

                // Debug para visualizar os dados recebidos no console do Unity
                Debug.Log("Dados recebidos do MATLAB: " + receivedString);

                //ValueEncoder = int.Parse(receivedString);
                // Converter para int e verificar se é NaN
                int parsedValue;
            }
        }
    }

```

```

        // Tenta converter a string receivedString em um valor inteiro e
        // armazena o resultado na variável parsedValue.
        // O método TryParse retorna true se a conversão for bem - sucedida
        // e false se falhar.
        bool parseSuccess = int.TryParse(receivedString, out parsedValue);
        if (parseSuccess && !double.IsNaN(parsedValue))
        {
            ValueEncoder = parsedValue;
        }
        else
        {
            ValueEncoder = 0;
        }
    }
    catch (Exception e)
    {
        Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
    }
}
Debug.Log("Thread Aqui");
}

void OnDestroy()
{
    _shouldStop = true;
}

public void Enable()
{
    gameObject.SetActive(true);
    _shouldStop = false;
    Debug.Log("Inicio");
    //stopThread = false;
}

public void Disable()
{
    _shouldStop = true;
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
    gameObject.SetActive(false);
    Debug.Log("Fim");
}
}

```

Código 8: Arquivo de configuração tricolorB_Dimt_2_11.yaml para treinador PPO

```
behaviors:
  TricolorB:
    trainer_type: ppo
    hyperparameters:
      batch_size: 512
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.0005
      epsilon: 0.2
      lambda: 0.99
      num_epoch: 3
      shared_critic: false
      learning_rate_schedule: linear
      beta_schedule: linear
      epsilon_schedule: linear
    checkpoint_interval: 200000
    network_settings:
      normalize: false
      hidden_units: 64
      num_layers: 2
      vis_encode_type: simple
      memory: null
      goal_conditioning_type: hyper
      use_vail: true
      deterministic: false
    reward_signals:
      extrinsic:
        gamma: 0.9
        strength: 1.0
        network_settings:
          normalize: false
          hidden_units: 64
          num_layers: 2
          vis_encode_type: simple
          memory: null
          goal_conditioning_type: hyper
          deterministic: false
          use_vail: true
      gail:
        strength: 0.4
        gamma: 0.9
        demo_path: Demos/demopistas3.demo
        learning_rate: 0.00008
        use_vail: true
    init_path: null
    keep_checkpoints: 5
```

```
even_checkpoints: false
max_steps: 2000000
time_horizon: 512
summary_freq: 20
threaded: false
self_play: null
behavioral_cloning:
  strength: 0.5
  gamma: 0.9
  demo_path: Demos/demopistas3.demo
  batch_size: 512
```

Código 9: Código Matlab do jogo Tricor para comunicação com o Unity quando há calibração

```
clc; clear all; instrreset;

s = serial('COM3', 'BaudRate', 19200);
fopen(s); % Abre a porta serial antes do loop

% Defina os valores máximos (pMax) e mínimos (pMin) manualmente
pMax = 400; % Substitua pelo valor máximo desejado
pMin = -400; % Substitua pelo valor mínimo desejado
pula = 'pula';
agacha = 'agacha';
mantem = 'corre';
nTotal = 10; %aqui

%% Valores Encoder
% Inicializar matriz para armazenar dados do Encoder
maxRowsEnc = 5000;
rowsEncAdd = 5000;
bufferEnc = zeros(maxRowsEnc, 1); %prealocando array para mais velocidade

% Inicializa variáveis Encoder
valorAtual2 = 0;
valorAtualTrue = 0;
vAMin = 0;
vAMax = 0;
nNaN = 0;
rowCountEnc = 0; % Contador de linhas arduino
nVal = 0;
boolNVal = false;
maxNVal = 5;
buffValorAtual = zeros(maxNVal, 1);

% Inicializa vetor para armazenar os últimos numLastValues valores do
% Encoder
numLastValues = 5;
lastValues = zeros(numLastValues, 1);
nFMedia = 0;

% Defina um temporizador para verificar a recepção de dados do Unity
timeoutTimer = tic;
timeoutDuration = 12; % Defina o tempo limite em segundos
playTime = true;
nTimeOut = 0;
n2TimeOut = 0;

%% UDP
% Configuração do servidor UDP no MATLAB
unityReceivePort = 35000; % Porta para receber dados do Unity
unitySendPort = 52000; % Porta para enviar dados ao Unity

udpServer = udp('127.0.0.1', unityReceivePort, 'LocalPort', unitySendPort);
set(udpServer, 'Timeout', 30);
fopen(udpServer);

% Inicializar matriz para armazenar dados
maxRows = 5000;
dataMatrix = zeros(maxRows, nTotal); %prealocando array para mais velocidade

rowCount = 0; % Contador de linhas
```

```

dataValuesInc = 0;
filename = 'dados_excel_Jogo2_calibr.xlsx'; % Nome do arquivo Excel %aqui
filenameFinal = 'dados_excel_final_Jogo2_calibr.xlsx'; % Nome do arquivo Excel ao
final da execução %aqui

% Verificar se o arquivo existe e removê-lo se existir
if exist(filename, 'file') == 2
    delete(filename);
end
if exist(filenameFinal, 'file') == 2
    delete(filenameFinal);
end

%% while
while playTime
    %%
    %% Receber dados do Unity
    if udpServer.BytesAvailable == 0
        disp('Sem dados recebidos. Verificando timeout...');
        % Verificar o tempo decorrido desde o último recebimento de dados
        elapsedTime = toc(timeoutTimer);
        if elapsedTime > timeoutDuration
            disp('Timeout atingido. Saindo do loop.');
```

playTime = false;

continue;

end

nTimeOut = nTimeOut + 1;

pause(0.5);

continue;

end

data = fread(udpServer, 1024, 'char');

% info = whos('data');

% sizeInBytes = info.bytes

dataString = char(data);

dataValues = str2double(strsplit(dataString, '!'))

%Verificar nTotal elementos em dataValues

disp(numel(dataValues))

%if numel(dataValues) == nTotal

% Incrementar o contador de linhas e verificar se excede o tamanho

prealocado

rowCount = rowCount + 1;

if rowCount > maxRows

% Se exceder o tamanho prealocado, aumente o tamanho da matriz

dataMatrix = [dataMatrix; zeros(maxRows, nTotal)]; % Adicione mais

linhas à matriz

maxRows = maxRows+5000; % Ajuste o tamanho máximo

end

if numel(dataValues) == nTotal

% Armazenar os dados na matriz prealocada

if (dataValues == zeros(1, nTotal))

dataValues = ones(1, nTotal);

end

dataMatrix(rowCount, :) = dataValues;

% Salvando cada conjunto de valores em uma nova linha na matriz

dataMatrix = [dataMatrix; dataValues];

else

% Se faltar valores, completar com zeros

% missingValues = nTotal - numel(dataValues);

```

        % dataValues = [dataValues, zeros(1, missingValues)];
        % dataMatrix(rowCount, :) = dataValues;
    end
    %aqui
    pontos = dataValues(1)
    playerPositX = dataValues(2)/100
    scrBar = dataValues(3)
    scrCoin = dataValues(4)
    scrCor = dataValues(5)
    scrCorB = dataValues(6)
    scrCorG = dataValues(7)
    scrCorR = dataValues(8)
    posCor = dataValues(9)
    modoJogo = dataValues(10)
    % else
    %     dataValuesInc = dataValuesInc + 1;
    %     disp('Número incorreto de elementos em dataValues');
    %     Lógica para lidar com o erro, se necessário

    %% Salvar dados em uma planilha Excel
    if mod(rowCount, 120) == 0
        % Salvar a cada 120 linhas
        filename = 'dados_excel_Jogo2_calibr.xlsx';%aqui
        writematrix(dataMatrix(1:rowCount, :), filename); % ou use writetable se
preferir trabalhar com tabelas
    end

    %% Verificando dados recebidos
    if(modoJogo == 2 || modoJogo == 4 || modoJogo == 6)
        boolModoJogo = true;
    else
        boolModoJogo = false;
    end

    if (boolModoJogo == false)
        %% Tic
        timeoutTimer = tic;
        %% Arduino
        % Leitura do Arduino

        % Lê os dados disponíveis na porta serial
        nVal = 0;

        while s.BytesAvailable > 0
            fread(s, s.BytesAvailable);
        end

        valorEnc = fgetl(s);

        while (nVal < maxNVal)
            valorEnc = fscanf(s, '%s');
            if valorEnc(end) == '!'
                boolNVal = false;
                valorEnc = valorEnc(1:end-1);
                buffValorAtual(nVal+1) = str2double(valorEnc);
                valorAtual = buffValorAtual(nVal+1);
            else
                boolNVal = true;
                nVal = nVal - 1;
            end

            %buffValorAtual = str2double(strsplit(valorEnc, '!'));

```

```

        %disp(valorAtual)
        if (boolNVal == false)
            % Adiciona os novos dados ao buffer
            rowCountEnc = rowCountEnc + 1;
            if rowCountEnc > maxRowsEnc
                % Se exceder o tamanho prealocado, aumente o tamanho da matriz
                bufferEnc = [bufferEnc; zeros(rowsEncAdd, 1)]; % Adicione mais
linhas à matriz
                maxRowsEnc = maxRowsEnc + rowsEncAdd; % Ajuste o tamanho máximo
            end

            % Armazenar os dados na matriz prealocada
            if isnan(valorAtual)
                bufferEnc(rowCountEnc) = bufferEnc(rowCountEnc-1);
                nNaN = nNaN + 1;
                %continue;
            else
                bufferEnc(rowCountEnc) = valorAtual;
            end

            % Atualiza o vetor dos últimos numLastValues valores
            lastValues(mod(rowCountEnc, numLastValues) + 1) = valorAtual;

            % Calcula a média e o desvio padrão dos últimos 5 valores
            if rowCountEnc >= numLastValues
                media = mean(lastValues);
                desvioPadrao = std(lastValues);
                vAMin = media - desvioPadrao;
                VAMax = media + desvioPadrao;
                if (valorAtual < vAMin) || (valorAtual > vAMax)
                    valorAtual2 = round(media);
                    nFMedia = nFMedia + 1;
                else
                    valorAtual2 = valorAtual;
                end
            end

            bufferEnc(rowCountEnc) = valorAtual2;
        end
        nVal = nVal + 1;
    end
else
    %evitamos ler posição do encoder e só mandamos a última antes de
    %mudar de modoJogo
end
% Exibe o valor atual em tempo real
disp(['Valor Atual: ' num2str(valorAtual2)]);
fprintf(udpServer, '%d', valorAtual2);
%% Processar os dados recebidos
% Lógica de movimento com base nos valores de pMax e pMin
if valorAtual2 > pMax
    disp(pula); %fwrite(tcpClient, sobe);
elseif valorAtual2 < pMin
    disp(agacha); %fwrite(tcpClient, desce);
else
    disp(mantem); %fwrite(tcpClient, mantem);
end
%pause(0.05); % Espera 0.2 segundos (ajuste conforme necessário)

pause(0.02);
end

filenameFinal = 'dados_excel_final_Jogo2_calibr.xlsx'; %aqui

```

```
writematrix(dataMatrix(1:rowCount, :), filenameFinal);

fclose(s); % Feche a porta serial quando não for mais necessária
fclose(udpServer);
delete(udpServer);

% se perder conexão -> fechar (ideia)

% Configurações
```

Código 10: Código do IDE Arduino para quando a IA é capaz de jogar sozinha

```
#include <Encoder.h>

Encoder myEncoder(2, 3); // Substitua 2 e 3 pelos pinos conectados ao encoder.
int RPWM = 10;
int LPWM = 9;
int vel1 = 0; // de 1 a 255 (min a max)
int vel = 0;
long pos = 0;
int j = 0;
int iloop = 0;
char separador = '!';

void setup() {
  Serial.begin(19200); // Inicializa a comunicação serial
  // Inicializa a Ponte H
  pinMode(RPWM, OUTPUT);
  pinMode(LPWM, OUTPUT);
  // Define a velocidade inicial do motor
  analogWrite(RPWM, 0); // Desliga o motor
  analogWrite(LPWM, 0); // Desliga o motor
}

void loop() {
  // Enviando
  if (iloop != 0){
    j = 0;
    while (j < 6) {
      //Concatenando informações
      pos = myEncoder.read(); // Lê o valor do encoder
      String outputString = String(pos) + separador + String(vel1);
      Serial.println(outputString); // Envia o valor pela porta serial
      j = j + 1;
    }
  }

  if (Serial.available() > 0) {
    vel = Serial.parseInt();

    // Define a velocidade do motor para girar
    if (vel > 0){
      analogWrite(RPWM, vel); // Velocidade escolhida
      analogWrite(LPWM, 0); // Motor desligado
    }
    else if (vel < 0){
      analogWrite(RPWM, 0); // Velocidade escolhida
      analogWrite(LPWM, abs(vel)); // Motor desligado
    }
    else{
      analogWrite(RPWM, 0); // Desliga o motor
      analogWrite(LPWM, 0); // Desliga o motor
    }
  }
  vel1 = vel;
  iloop = iloop + 1;
  // Aguarde um curto período de tempo para evitar sobrecarregar a porta serial
  delay(5);
}
```

Código 11: Código do Matlab do jogo Tricor para comunicação com o Unity quando a IA é capaz de jogar sozinha

```
clc; clear all; instrreset;

s = serial('COM3', 'BaudRate', 19200);
fopen(s); % Abre a porta serial antes do loop

% Defina os valores máximos (pMax) e mínimos (pMin) manualmente
pMax = 400; % Substitua pelo valor máximo desejado
pMin = -400; % Substitua pelo valor mínimo desejado
pula = 'pula';
agacha = 'agacha';
mantem = 'corre';
nTotal = 11; % -tecMot

%% Valores Encoder
% Inicializar matriz para armazenar dados do Encoder
maxRowsEnc = 5000;
rowsEncAdd = 5000;
bufferEnc = zeros(maxRowsEnc, 1); %prealocando array para mais velocidade

% Inicializa variáveis Encoder
valorAtual2 = 0;
valorAtualTrue = 0;
vAMin = 0;
vAMax = 0;
nNaN = 0;
rowCountEnc = 0; % Contador de linhas arduino
nVal = 0;
boolNVal = false;
maxNVal = 5;
buffValorAtual = zeros(maxNVal, 1);

% Inicializa vetor para armazenar os últimos numLastValues valores do
% Encoder
numLastValues = 5;
lastValues = zeros(numLastValues, 1);
nFMedia = 0;

% -tecMot
vetEnv = 0;
vetEnv2 = 0;
minMot = 100;
maxMot = 200;

% Defina um temporizador para verificar a recepção de dados do Unity
timeoutTimer = tic;
timeoutDuration = 12; % Defina o tempo limite em segundos
playTime = true;
nTimeOut = 0;
n2TimeOut = 0;

%% UDP
% Configuração do servidor UDP no MATLAB
unityReceivePort = 35000; % Porta para receber dados do Unity
unitySendPort = 52000; % Porta para enviar dados ao Unity

udpServer = udp('127.0.0.1', unityReceivePort, 'LocalPort', unitySendPort);
set(udpServer, 'Timeout', 30);
fopen(udpServer);
```

```

% Inicializar matriz para armazenar dados
maxRows = 5000;
dataMatrix = zeros(maxRows, nTotal); %prealocando array para mais velocidade

rowCount = 0; % Contador de linhas
dataValuesInc = 0;
filename = 'dados_excel_Jogo2_IA_tecmot.xlsx'; % Nome do arquivo Excel
filenameFinal = 'dados_excel_final_Jogo2_IA_tecmot.xlsx'; % Nome do arquivo Excel ao
final da execução

% Verificar se o arquivo existe e removê-lo se existir
if exist(filename, 'file') == 2
    delete(filename);
end
if exist(filenameFinal, 'file') == 2
    delete(filenameFinal);
end

%% while
while playTime
    %%
    %% Receber dados do Unity
    if udpServer.BytesAvailable == 0
        disp('Sem dados recebidos. Verificando timeout...');
        % Verificar o tempo decorrido desde o último recebimento de dados
        elapsedTime = toc(timeoutTimer);
        if elapsedTime > timeoutDuration
            disp('Timeout atingido. Saindo do loop.');
```

playTime = false;

continue;

end

nTimeOut = nTimeOut + 1;

pause(0.5);

continue;

end

data = fread(udpServer, 1024, 'char');

% info = whos('data');

% sizeInBytes = info.bytes

dataString = char(data);

dataValues = str2double(strsplit(dataString, '!'))

%Verificar nTotal elementos em dataValues

disp(numel(dataValues))

%if numel(dataValues) == nTotal

% Incrementar o contador de linhas e verificar se excede o tamanho

prealocado

rowCount = rowCount + 1;

if rowCount > maxRows

% Se exceder o tamanho prealocado, aumente o tamanho da matriz

dataMatrix = [dataMatrix; zeros(maxRows, nTotal)]; % Adicione mais

linhas à matriz

maxRows = maxRows+5000; % Ajuste o tamanho máximo

end

if numel(dataValues) == nTotal

% Armazenar os dados na matriz prealocada

if (dataValues == zeros(1, nTotal))

disp("zeros");%dataValues = ones(1, nTotal);

end

dataMatrix(rowCount, :) = dataValues;

```

        % Salvando cada conjunto de valores em uma nova linha na matriz
        dataMatrix = [dataMatrix; dataValues];
    else
        % Se faltar valores, completar com zeros
        % missingValues = nTotal - numel(dataValues);
        % dataValues = [dataValues, zeros(1, missingValues)];
        % dataMatrix(rowCount, :) = dataValues;
    end

    pontos = dataValues(1)
    playerPositX = dataValues(2)/100
    scrBar = dataValues(3)
    scrCoin = dataValues(4)
    scrCor = dataValues(5)
    scrCorB = dataValues(6)
    scrCorG = dataValues(7)
    scrCorR = dataValues(8)
    posCor = dataValues(9)
    modoJogo = dataValues(10)
    inputTec = dataValues(11) % -tecMot
% else
%     dataValuesInc = dataValuesInc + 1;
%     disp('Número incorreto de elementos em dataValues');
%     % Lógica para lidar com o erro, se necessário

%% Salvar dados em uma planilha Excel
if mod(rowCount, 120) == 0
    % Salvar a cada 120 linhas
    filename = 'dados_excel_Jogo2_IA_tecmot.xlsx';
    writematrix(dataMatrix(1:rowCount, :), filename); % ou use writetable se
preferir trabalhar com tabelas
end

%% Tic
timeoutTimer = tic;
%% Arduino
% Leitura do Arduino
% Cálculo da Velocidade

%% -tecMot
if (inputTec > 0)
    vetEnv = minMot + inputTec*(maxMot-minMot)/100;
elseif (inputTec < 0)
    vetEnv = -minMot + inputTec*(maxMot-minMot)/100;
else
    vetEnv = 0;
end
vetEnv2 = round(vetEnv);
disp(vetEnv2)
fprintf(s, '%d\n', vetEnv2);
%fwrite(s, vetEnv2, 'uint8');

%%
% Lê os dados disponíveis na porta serial
nVal = 0;

while s.BytesAvailable > 0
    fread(s, s.BytesAvailable);
end

valorEnc = fgetl(s);

while (nVal < maxNVal)

```

```

valorEnc = fscanf(s, '%s');
Vet_pos_vel = str2double(strsplit(valorEnc, '!'));
posEnc = Vet_pos_vel(1);
velRec = Vet_pos_vel(2); % perguntar if(velRec ~= vetEnv)
if ~isnan(posEnc)
    boolNVal = false;
    valorEnc = posEnc;
    buffValorAtual(nVal+1) = valorEnc;
    valorAtual = buffValorAtual(nVal+1);
else
    boolNVal = true;
    nVal = nVal - 1;
end

%buffValorAtual = str2double(strsplit(valorEnc, '!'));

%disp(valorAtual)
if (boolNVal == false)
    % Adiciona os novos dados ao buffer
    rowCountEnc = rowCountEnc + 1;
    if rowCountEnc > maxRowsEnc
        % Se exceder o tamanho prealocado, aumente o tamanho da matriz
        bufferEnc = [bufferEnc; zeros(rowsEncAdd, 1)]; % Adicione mais
linhas à matriz
        maxRowsEnc = maxRowsEnc + rowsEncAdd; % Ajuste o tamanho máximo
    end

    % Armazenar os dados na matriz prealocada
    if isnan(valorAtual)
        bufferEnc(rowCountEnc) = bufferEnc(rowCountEnc-1);
        nNaN = nNaN + 1;
        %continue;
    else
        bufferEnc(rowCountEnc) = valorAtual;
    end

    % Atualiza o vetor dos últimos numLastValues valores
    lastValues(mod(rowCountEnc, numLastValues) + 1) = valorAtual;

    % Calcula a média e o desvio padrão dos últimos 5 valores
    if rowCountEnc >= numLastValues
        media = mean(lastValues);
        desvioPadrao = std(lastValues);
        vAMin = media - desvioPadrao;
        VAMax = media + desvioPadrao;
        if (valorAtual < vAMin) || (valorAtual > VAMax)
            valorAtual2 = round(media);
            nFMedia = nFMedia + 1;
        else
            valorAtual2 = valorAtual;
        end
    end

    bufferEnc(rowCountEnc) = valorAtual2;
end
nVal = nVal + 1;
% if(velRec ~= vetEnv)
%     disp(['Velocidades incompatíveis: VetEnv = ' num2str(vetEnv) '.
VelRec = ' num2str(velRec)'])
% end

end
% Exibe o valor atual em tempo real
disp(['Valor Atual: ' num2str(valorAtual2)]);

```

```

fprintf(udpServer, '%d', valorAtual2);
%% Processar os dados recebidos e calcular BirdControl
% Lógica de movimento com base nos valores de pMax e pMin
if valorAtual2 > pMax
    disp(pula);%fwrite(tcpClient, sobe);
elseif valorAtual2 < pMin
    disp(agacha);%fwrite(tcpClient, desce);
else
    disp(mantem);%fwrite(tcpClient, mantem);
end
%pause(0.05); % Espera 0.2 segundos (ajuste conforme necessário)

pause(0.02);
end

filenameFinal = 'dados_excel_final_Jogo2_IA_tecmot.xlsx';
writematrix(dataMatrix(1:rowCount, :), filenameFinal);

fclose(s); % Feche a porta serial quando não for mais necessária
fclose(udpServer);
delete(udpServer);

% se perder conexão -> fechar (ideia)

% Configurações

```

Código 12: Código do Unity do jogo Tricor para o agente a ser treinado pela IA

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using System.Security.Cryptography;

public class TricorAgent : Agent
{
    private Player player;
    private server_abs server_Abs;

    private float velocidadeIA = 0;
    private float rangeEncoder = 0;
    private int capacIA = 0;
    private int encoderIA_now;
    public int pmoveX;

    private float rBar = 0;
    private float rCoi = 0;
    private float rCPB = 0;
    private float rCPG = 0;
    private float rCPR = 0;
    private float rrBar = 0;
    private float rrCoi = 0;
    private float rrCPB = 0;
    private float rrCPG = 0;
    private float rrCPR = 0;
    private float pBar = 0;
    private float pCoi = 0;
    private float pCPB = 0;
    private float pCPG = 0;
    private float pCPR = 0;

    private bool notOut = true;

    private void Awake()
    {
        player = GetComponent<Player>();
        server_Abs = GetComponent<server_abs>();
    }

    public override void OnEpisodeBegin()
    {
        Debug.Log("voltou");
        if (server_abs.inicioIA == false)
        {
            GameManager.Instance.GameOver2();
            GameManager.Instance.Enable();
        }
        if(server_abs.inicioIA == true) { server_abs.inicioIA = false; }

        rBar = server_abs.errosBarreira;
        rCoi = server_abs.contadorMoedas;
        rCPB = player.contadorPistaB;
        rCPG = player.contadorPistaG;
        rCPR = player.contadorPistaR;
    }
}
```

```

player.Reset();
server_abs.encoderIA = 0;
server_abs.encZeroCalib = Random.Range(-70, 70);
server_abs.limitLCalib = Random.Range(-600, -400);
server_abs.limitRCalib = Random.Range(400, 600);
velocidadeIA = 0.05f; // Random.Range(0.05f, 0.3f); //pra considerar a
velocidade de vários jogadores
rangeEncoder = Mathf.Abs(server_abs.limitLCalib - server_abs.limitRCalib);
//em vários ranges
capacIA = Mathf.RoundToInt(velocidadeIA * rangeEncoder); //capacidade da IA
de alterar o valor do encoder por step
//Debug.Log("LimitL: " + server_abs.limitLCalib + "| LimitR: " +
server_abs.limitRCalib);
//Debug.Log("EncZero: " + server_abs.encZeroCalib);
//Debug.Log("Velocidade: " + velocidadeIA);
//Debug.Log("Capacidade: " + capacIA);

//só pra testar
Destroy(GameObject.FindWithTag("Coin"));
Destroy(GameObject.FindWithTag("Barreira"));
}

public override void CollectObservations(VectorSensor sensor)
{
    sensor.AddObservation(player.gameObject.transform.position.x);
    sensor.AddObservation(server_abs.caso2);
    sensor.AddObservation(server_abs.xcoin);
    sensor.AddObservation(server_abs.barreiraOn);
    sensor.AddObservation(server_abs.coinOn);
}

public override void OnActionReceived(ActionBuffers actions)
{
    if (GameManager.Instance == null)
    {
        return;
    }
    float dMove = actions.DiscreteActions[0]; //antes era continuous
    int moveX = 0;
    if (dMove == 1) { moveX = -1; }
    if (dMove == 0) { moveX = 0; }
    if (dMove == 2) { moveX = 1; }

    pmoveX = moveX;
    encoderIA_now = Mathf.RoundToInt(moveX * capacIA);

    server_abs.encoderIA += encoderIA_now;
    Debug.Log(server_abs.encoderIA);
    if ((server_abs.encoderIA < server_abs.limitLCalib && moveX < 0) ||
(server_abs.encoderIA > server_abs.limitRCalib && moveX > 0))
    {
        notOut = false;
        AddReward(-4f);
    }
    else if ((server_abs.encoderIA < server_abs.limitLCalib && moveX >= 0) ||
(server_abs.encoderIA > server_abs.limitRCalib && moveX <= 0))
    {
        notOut = false;
        AddReward(+0.4f);
    }
}

```

```

        else if(server_abs.encoderIA >= server_abs.limitLCalib &&
server_abs.encoderIA <= server_abs.limitRCalib)
        {
            notOut = true;
        }
    }

    public override void Heuristic(in ActionBuffers actionsOut)
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        //ActionSegment<float> continuousActions = actionsOut.ContinuousActions;
        //continuousActions[0] = Input.GetAxis("Horizontal");
        ActionSegment<int> discreteActions = actionsOut.DiscreteActions;
        switch (Mathf.RoundToInt(Input.GetAxisRaw("Horizontal")))
        {
            case -1:
                discreteActions[0] = 1; break;
            case 0:
                discreteActions[0] = 0; break;
            case 1:
                discreteActions[0] = 2; break;
        }

        //Debug.Log(continuousActions[0]);
        //Debug.Log(StepCount);
        //Debug.Log(CompletedEpisodes);
    }

    private void FixedUpdate()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        rBar = server_abs.errosBarreira;
        rCoi = server_abs.contadorMoedas;
        rCPB = player.contadorPistaB;
        rCPG = player.contadorPistaG;
        rCPR = player.contadorPistaR;

        // Se for maior que o anterior da a recompensa pelo quanto ele é maior, se
        menor, no negativo

        pBar = rBar - rrBar; AddReward(-pBar);
        pCoi = rCoi - rrCoi; AddReward(pCoi);

        if (notOut == true)
        {
            pCPB = rCPB - rrCPB; AddReward(pCPB);

            pCPG = rCPG - rrCPG; AddReward(pCPG);

            pCPR = rCPR - rrCPR; AddReward(pCPR);
        }

        rrBar = server_abs.errosBarreira;
        rrCoi = server_abs.contadorMoedas;
        rrCPB = player.contadorPistaB;
        rrCPG = player.contadorPistaG;

```

```
        rrCPR = player.contadorPistaR;  
    }  
}
```

Apêndice B – Códigos dos jogos

Tricor com IA

Código 13: Player do jogo Tricor quando controlado por IA

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Player : server_abs
{
    [SerializeField] private float _speed = 6f;
    [SerializeField] private float _horizontalBoundaryR = 4f;
    [SerializeField] private float _horizontalBoundaryL = -4f;

    [SerializeField]
    private GameManager gameManager;

    private Rigidbody2D rb; // pra evitar de pesar o update

    public float contadorPistaG = 0;
    public float contadorPistaR = 0;
    public float contadorPistaB = 0;
    public float contadorPista = 0;
    private float horizontalInput = 0;

    // Encoder
    public float encoderAnt = 0;
    public bool encoderBoolR = false;
    public int limitR = limitRCalib; // deve ser pegado do valor UDP_1.ValueEncoder
    public float pEncR = 0;
    public bool encoderBoolL = false;
    public int limitL = limitLCalib; //aqui
    public float pEncL = 0;
    public float zonaMeioR = 50;
    public float zonaMeioL = -50;
    public float encInput = 0;
    public int encZero = encZeroCalib; //aqui
    public float encPos = 0;
    public float realInput = 0;
    public bool gameManagerInstance = false;
    public bool gameManagerInstanceAnt = false;

    void Awake() //trocamos o Start pelo Awake pq o rb não estava sendo declarado
    após o Restart
    {
        rb = GetComponent<Rigidbody2D>(); //caching -> salvando esse componente pra
        utilizar ele dentro do update
    }
    // Update is called once per frame

    private void EncoderBoolR()
    {
        //if (encPos == encoderAnt) { encoderBoolR = false; }
    }
}
```

```

        if (encPos > zonaMeioR)
        {
            encoderBoolR = true; encoderBoolL = false;
            pEncR = Mathf.Abs(encPos / (limitR - encZero));
            //Debug.Log("pEncR = " + pEncR); Debug.Log("encPos = " +
encPos); Debug.Log("limitR = " + limitR); Debug.Log("encZero = " + encZero);
            if (pEncR > 1) { pEncR = 1; }
            if (pEncR < 0) { pEncR = 0; }
            //Debug.Log("1_pEncR = " + pEncR);
        }
    }

    private void EncoderBoolL()
    {
        if (encPos < zonaMeioL)
        {
            encoderBoolR = false; encoderBoolL = true;
            pEncL = -(Mathf.Abs((float)encPos / (limitL - encZero)));
            //Debug.Log("| pEncL = " + pEncL + "| encPos = " + encPos + "| limitL =
" + limitL + " | encZero = " + encZero);
            if (pEncL < -1) { pEncL = -1; }
            if (pEncL > 0) { pEncL = 0; }
            //Debug.Log(" |1_pEncL = " + pEncL);
        }
    }

    void Update()
    {
        encPos = UDP_1.ValueEncoder - encZero;
        //encPos = encoderIA - encZero; //IA-tecMot
        if (GameManager.Instance == null)
        {
            gameManagerInstance = false;
            return; // ó código abaixo não é executado antes do jogo começar
        }
        else
        {
            gameManagerInstance = true;
        }
        if (gameManagerInstance != gameManagerInstanceAnt)
        {
            limitR = limitRCalib;
            limitL = limitLCalib;
            encZero = encZeroCalib;
            Debug.Log(encZeroCalib);
            zonaMeioL = Mathf.Floor((limitL - encZero) * 0.15f);
            zonaMeioR = Mathf.Floor((limitR - encZero) * 0.15f);
        }

        if (isPaused == false)
        {
            EncoderBoolR();
            EncoderBoolL();
            if (encPos >= zonaMeioL && encPos <= zonaMeioR) { encoderBoolR = false;
encoderBoolL = false; pEncL = 0; pEncR = 0; }

            encInput = pEncL + pEncR;
            //Debug.Log("encInput = " + encInput + " |pEncL = " + pEncL + "pEncR = "
+ pEncR);

            //horizontalInput = Input.GetAxis("Horizontal");

```

```

        //inputTecl = (int)(horizontalInput * 100); // -tecMot

        // se houver input do encoder realInput serão os valores dele, caso
        contrário o controle é devolvido as setas do teclado
        //if (encoderBoolR == false && encoderBoolL == false)
        //{
        //    realInput = horizontalInput; //Debug.Log("tecla");
        //    //realInput = 0; // -tecMot
        //}
        //else
        //{
        realInput = encInput; //Debug.Log("encoder"); // Aqui é pra IA ler
        sempre o valor certo
        //}
        //Debug.Log(horizontalInput); // + " + " + encoderBoolL + " + " +
        encoderBoolR);
        // Move o jogador na direção do input
        //Debug.Log("RealInput = " + realInput);
        rb.transform.Translate(Vector3.right * realInput * _speed *
        Time.deltaTime); // realInput é escolhido de acordo com o input recebido

        // Mantém o jogador dentro dos limites horizontais
        if (rb.transform.position.x > _horizontalBoundaryR)
        {
            rb.transform.position = new Vector3(_horizontalBoundaryR,
            rb.transform.position.y, rb.transform.position.z);
        }
        else if (rb.transform.position.x < _horizontalBoundaryL)
        {
            rb.transform.position = new Vector3(_horizontalBoundaryL,
            rb.transform.position.y, rb.transform.position.z);
        }

        contadorPista = contadorPistaB + contadorPistaG + contadorPistaR;
    }
    else return;
    encAnt = encPos;
    pEncL = 0;
    pEncR = 0;
    gameManagerInstanceAnt = gameManagerInstance;
}
private void OnEnable()
{
    transform.position = new Vector3(0, -2.97f, 0);
}
public void Reset()
{
    transform.position = new Vector3(0, -2.97f, 0);
}
private void OnTriggerStay2D(Collider2D other)
{
    // tvz compensa usar um método aqui dps
    // azul -> cinza -> amarelo
    if (other.gameObject.CompareTag("PG"))
    {
        if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
//blue meio
        {
            contadorPistaG += 4 * Time.deltaTime;
        }
        if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
        {
            contadorPistaG -= 4f * Time.deltaTime; //+0.5f

```

```

    }
    if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
    {
        contadorPistaG -= 4f * Time.deltaTime; //-1f
    }
    //Debug.Log(((int)contadorPistaG));
}
if (other.gameObject.CompareTag("PR"))
{
    if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
    {
        contadorPistaR -= 4f * Time.deltaTime;
    }
    if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
    {
        contadorPistaR -= 4f * Time.deltaTime;
    }
    if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
//grey meio
    {
        contadorPistaR += 4f * Time.deltaTime;
    }
    //Debug.Log(((int)contadorPistaR));
}
if (other.gameObject.CompareTag("PB"))
{
    if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
    {
        contadorPistaB -= 4f * Time.deltaTime; //+0.5f -2f
    }
    if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
//yellow meio
    {
        contadorPistaB += 4f * Time.deltaTime; //+5f
    }
    if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
    {
        contadorPistaB -= 4f * Time.deltaTime; //+0.5f -1f
    }
    //Debug.Log(((int)contadorPistaB));
}
}

// Chamado após as atualizações físicas
private void FixedUpdate()
{
    // Ativa o corpo rígido para garantir que as colisões sejam detectadas
    rb.WakeUp();
}
}

```

Código 14: Agente do Jogo Tricor capaz de jogar sozinho

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using Unity.MLAgents;
using Unity.MLAgents.Actuators;
using Unity.MLAgents.Sensors;
using System.Security.Cryptography;

public class TricorAgent : Agent
{
    private Player player;
    private server_abs server_Abs;

    private float velocidadeIA = 0;
    private float rangeEncoder = 0;
    private int capacIA = 0;
    private int encoderIA_now;

    private float rBar = 0;
    private float rCoi = 0;
    private float rCPB = 0;
    private float rCPG = 0;
    private float rCPR = 0;
    private float rrBar = 0;
    private float rrCoi = 0;
    private float rrCPB = 0;
    private float rrCPG = 0;
    private float rrCPR = 0;
    private float pBar = 0;
    private float pCoi = 0;
    private float pCPB = 0;
    private float pCPG = 0;
    private float pCPR = 0;

    private bool notOut = true;

    private void Awake()
    {
        player = GetComponent<Player>();
        server_Abs = GetComponent<server_abs>();
    }

    public override void OnEpisodeBegin()
    {
        Debug.Log("voltou");
        if (server_abs.inicioIA == false)
        {
            GameManager.Instance.GameOver2();
            GameManager.Instance.Enable();
        }
        if(server_abs.inicioIA == true) { server_abs.inicioIA = false; }

        rBar = server_abs.errosBarreira;
        rCoi = server_abs.contadorMoedas;
        rCPB = player.contadorPistaB;
        rCPG = player.contadorPistaG;
        rCPR = player.contadorPistaR;

        player.Reset();
        server_abs.encoderIA = 0;
    }
}
```

```

        server_abs.encZeroCalib = Random.Range(-70, 70);
        server_abs.limitLCalib = Random.Range(-600, -400);
        server_abs.limitRCalib = Random.Range(400, 600);
        velocidadeIA = 0.05f; // Random.Range(0.05f, 0.3f); //pra considerar a
velocidade de vários jogadores
        rangeEncoder = Mathf.Abs(server_abs.limitLCalib - server_abs.limitRCalib);
//em vários ranges
        capacIA = Mathf.RoundToInt(velocidadeIA * rangeEncoder); //capacidade da IA
de alterar o valor do encoder por step
        //Debug.Log("LimitL: " + server_abs.limitLCalib + "| LimitR: " +
server_abs.limitRCalib);
        //Debug.Log("EncZero: " + server_abs.encZeroCalib);
        //Debug.Log("Velocidade: " + velocidadeIA);
        //Debug.Log("Capacidade: " + capacIA);

        //só pra testar
        Destroy(GameObject.FindWithTag("Coin"));
Destroy(GameObject.FindWithTag("Barreira"));
    }

    public override void CollectObservations(VectorSensor sensor)
    {
        sensor.AddObservation(player.gameObject.transform.position.x);
        sensor.AddObservation(server_abs.caso2);
        sensor.AddObservation(server_abs.xcoin);
        sensor.AddObservation(server_abs.barreiraOn);
        sensor.AddObservation(server_abs.coinOn);
    }

    public override void OnActionReceived(ActionBuffers actions)
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        float dMove = actions.DiscreteActions[0]; //antes era continuous
        int moveX = 0;
        if (dMove == 1) { moveX = -1; }
        if (dMove == 0) { moveX = 0; }
        if (dMove == 2) { moveX = 1; }

        encoderIA_now = Mathf.RoundToInt(moveX * capacIA);
        server_abs.inputTecl = moveX * 100;
        server_abs.encoderIA += encoderIA_now;
        Debug.Log(server_abs.encoderIA);
        Debug.Log("inputTecl: " + server_abs.inputTecl);
        if ((server_abs.encoderIA < server_abs.limitLCalib && moveX < 0) ||
(server_abs.encoderIA > server_abs.limitRCalib && moveX > 0))
        {
            notOut = false;
            AddReward(-4f);
        }
        else if ((server_abs.encoderIA < server_abs.limitLCalib && moveX >= 0) ||
(server_abs.encoderIA > server_abs.limitRCalib && moveX <= 0))
        {
            notOut = false;
            AddReward(+0.4f);
        }
        else if(server_abs.encoderIA >= server_abs.limitLCalib &&
server_abs.encoderIA <= server_abs.limitRCalib)
        {

```

```

        notOut = true;
    }
}

public override void Heuristic(in ActionBuffers actionsOut)
{
    if (GameManager.Instance == null)
    {
        return;
    }
    //ActionSegment<float> continuousActions = actionsOut.ContinuousActions;
    //continuousActions[0] = Input.GetAxis("Horizontal");
    ActionSegment<int> discreteActions = actionsOut.DiscreteActions;
    switch (Mathf.RoundToInt(Input.GetAxisRaw("Horizontal")))
    {
        case -1:
            discreteActions[0] = 1; break;
        case 0:
            discreteActions[0] = 0; break;
        case 1:
            discreteActions[0] = 2; break;
    }

    //Debug.Log(continuousActions[0]);
    //Debug.Log(StepCount);
    //Debug.Log(CompletedEpisodes);
}

private void FixedUpdate()
{
    if (GameManager.Instance == null)
    {
        return;
    }
    rBar = server_abs.errosBarreira;
    rCoi = server_abs.contadorMoedas;
    rCPB = player.contadorPistaB;
    rCPG = player.contadorPistaG;
    rCPR = player.contadorPistaR;

    pBar = rBar - rrBar; AddReward(pBar);
    pCoi = rCoi - rrCoi; AddReward(pCoi);

    if (notOut == true)
    {
        pCPB = rCPB - rrCPB; AddReward(pCPB);

        pCPG = rCPG - rrCPG; AddReward(pCPG);

        pCPR = rCPR - rrCPR; AddReward(pCPR);
    }

    rrBar = server_abs.errosBarreira;
    rrCoi = server_abs.contadorMoedas;
    rrCPB = player.contadorPistaB;
    rrCPG = player.contadorPistaG;
    rrCPR = player.contadorPistaR;
}
}

```

Tricor sem IA 1

Código 15: Calibração no jogo Tricor

```
using PlasticPipe.PlasticProtocol.Messages;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UIElements;

public class Calibr : server_abs
{
    [SerializeField]
    private int simEncoder = 0;

    [SerializeField]
    private TMPro.TextMeshProUGUI AngText0;
    [SerializeField]
    private TMPro.TextMeshProUGUI AngTextSH;
    [SerializeField]
    private TMPro.TextMeshProUGUI AngTextSAH; //Teve sua classe ajustada para
    combinar com a UI

    [SerializeField]
    private GameObject instrucao;
    [SerializeField]
    private GameObject calibr0;
    [SerializeField]
    private GameObject calibrSH;
    [SerializeField]
    private GameObject calibrSAH;
    [SerializeField]
    private GameObject mainMenu;
    [SerializeField]
    private GameObject scoreTexto;

    [SerializeField]
    private GameObject rotac0;
    [SerializeField]
    private GameObject rotacSH;
    [SerializeField]
    private GameObject rotacSAH;

    private Coroutine DelayCasos0;
    private bool delayTime = false;

    private float velocidadeRotacao = 1000f;
    private float anguloDesejado;

    private float diferencaAngular = 0f;
    private int direcaoRot = 0;
    float quantidadeRotacao = 0f;

    // Start is called before the first frame update
    void Start()
    {
        //instrucao = GameObject.Find("Instructions");
        //calibr0 = GameObject.Find("Calibr0");
        //calibrSH = GameObject.Find("CalibrSH");
        //calibrSAH = GameObject.Find("CalibrSAH");
        //mainMenu = GameObject.Find("MainMenu");

        //rotac0 = GameObject.Find("rotac0");
    }
}
```

```

        //rotacSH = GameObject.Find("rotacSH");
        //rotacSAH = GameObject.Find("rotacSAH");
    }

    // Update is called once per frame
    void Update()
    {
        //posTempEncoder = UDP_1.ValueEncoder;
        //posTempEncoder = simEncoder;
        switch (modoJogo) {
            case 1:
                MoverCalibr(rotac0, AngText0);
                break;
            case 3:
                MoverCalibr(rotacSH, AngTextSH);
                break;
            case 5:
                MoverCalibr(rotacSAH, AngTextSAH);
                break;
        }
    }

    public void Confirmar()
    {
        Debug.Log(modoJogo);
        switch (modoJogo)
        {
            case 0://atual é zero, vamo passar pro 1
                scoreTexto.SetActive(false);
                calibr0.SetActive(true);
                instrucao.SetActive(false);
                modoJogo = 1;
                break;
            case 1: //atual é 1, vai passar pro 3
                calibrSH.SetActive(true);
                calibr0.SetActive(false);
                modoJogo = 2;
                StartCoroutine(DelayCasos());
                break;
            case 3:
                calibrSAH.SetActive(true);
                calibrSH.SetActive(false);
                //GetComponent<CanvasGroup>().LeanAlpha(0,
0.2f).setOnComplete(OnComplete);
                modoJogo = 4;
                StartCoroutine(DelayCasos());
                break;
            case 5:
                calibrSAH.SetActive(false);
                mainMenu.SetActive(true);
                modoJogo = 6;
                StartCoroutine(DelayCasos());
                scoreTexto.SetActive(true);

                //GetComponent<CanvasGroup>().LeanAlpha(0,
0.2f).setOnComplete(OnComplete);
                break;
            case 7:
                //GetComponent<CanvasGroup>().LeanAlpha(0,
0.2f).setOnComplete(OnComplete);
                break;
        }
    }

```

```

    }

    private void OnComplete()
    {
        //gameManager.Enable();
        Destroy(gameObject);
    }

    private void MoverCalibr(GameObject objCasos, TMPPro.TextMeshProUGUI textoAngulo)
    {
        //anguloDesejado = (simEncoder - encZeroCalib) * 360 / 4000;
        anguloDesejado = (UDP_1.ValueEncoder - encZeroCalib) * 360 / 4000;

        // Calcula a diferença angular entre o ângulo desejado e o ângulo atual
        diferencaAngular = Mathf.DeltaAngle(objCasos.transform.eulerAngles.z,
        anguloDesejado);

        // Determina a direção da rotação (horário ou anti-horário)
        direcaoRot = diferencaAngular > 0 ? 1 : -1;

        // Calcula a quantidade de rotação necessária para atingir o ângulo desejado
        quantidadeRotacao = Mathf.Min(velocidadeRotacao * Time.deltaTime,
        Mathf.Abs(diferencaAngular));

        // Rotaciona o objeto em torno do eixo Z na direção e quantidade desejadas
        objCasos.transform.Rotate(Vector3.forward, quantidadeRotacao * direcaoRot);

        textoAngulo.text = ((int)anguloDesejado).ToString();
    }

    private IEnumerator DelayCasos()
    {
        switch(modoSolo)
        {
            case 2:
                yield return new WaitForSeconds(0.5f);
                encZeroCalib = UDP_1.ValueEncoder;
                Debug.Log(encZeroCalib);
                yield return new WaitForSeconds(0.5f);
                modoSolo = 3;
                break;
            case 4:
                yield return new WaitForSeconds(0.5f);
                limitRCalib = UDP_1.ValueEncoder;
                Debug.Log(limitRCalib);
                yield return new WaitForSeconds(0.5f);
                modoSolo = 5;
                break;
            case 6:
                yield return new WaitForSeconds(0.5f);
                limitLCalib = UDP_1.ValueEncoder;
                Debug.Log(limitLCalib);
                yield return new WaitForSeconds(0.5f);
                modoSolo = 7;
                break;
        }
    }
}

```

Código 16: Player no jogo Tricor, sem IA

```
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using UnityEngine;

public class Player : server_abs
{
    [SerializeField] private float _speed = 6f;
    [SerializeField] private float _horizontalBoundaryR = 4f;
    [SerializeField] private float _horizontalBoundaryL = -4f;

    [SerializeField]
    private GameManager gameManager;

    private Rigidbody2D rb; // pra evitar de pesar o update

    public float contadorPistaG = 0;
    public float contadorPistaR = 0;
    public float contadorPistaB = 0;
    public float contadorPista = 0;
    private float horizontalInput = 0;

    // Encoder
    public float encoderAnt = 0;
    public bool encoderBoolR = false;
    public int limitR = limitRCalib; // deve ser pegado do valor UDP_1.ValueEncoder
    public float pEncR = 0;
    public bool encoderBoolL = false;
    public int limitL = limitLCalib; //aqui
    public float pEncL = 0;
    public float zonaMeioR = 50;
    public float zonaMeioL = -50;
    public float encInput = 0;
    public int encZero = encZeroCalib; //aqui
    public float encPos = 0;
    public float realInput = 0;
    public bool gameManagerInstance = false;
    public bool gameManagerInstanceAnt = false;

    void Awake() //trocamos o Start pelo Awake pq o rb não estava sendo declarado
    após o Restart
    {
        rb = GetComponent<Rigidbody2D>(); //caching -> salvando esse componente pra
        utilizar ele dentro do update
    }
    // Update is called once per frame

    private void EncoderBoolR()
    {
        //if (encPos == encoderAnt) { encoderBoolR = false; }
        if (encPos > zonaMeioR)
        {
            encoderBoolR = true; encoderBoolL = false;
            pEncR = Mathf.Abs(encPos / (limitR - encZero));
        }
    }
}
```

```

        //Debug.Log("pEncR = " + pEncR); Debug.Log("encPos = " +
encPos); Debug.Log("limitR = " + limitR); Debug.Log("encZero = " + encZero);
        if (pEncR > 1) { pEncR = 1; }
        if (pEncR < 0) { pEncR = 0; }
        //Debug.Log("l_pEncR = " + pEncR);
    }
}

private void EncoderBoolL()
{
    if (encPos < zonaMeioL)
    {
        encoderBoolR = false; encoderBoolL = true;
        pEncL = -(Mathf.Abs((float)encPos / (limitL - encZero)));
        //Debug.Log("| pEncL = " + pEncL + "| encPos = " + encPos + "| limitL =
" + limitL + " | encZero = " + encZero);
        if (pEncL < -1) { pEncL = -1; }
        if (pEncL > 0) { pEncL = 0; }
        //Debug.Log(" |l_pEncL = " + pEncL);
    }
}

void Update()
{
    encPos = UDP_1.ValueEncoder - encZero;
    if (GameManager.Instance == null)
    {
        gameManagerInstance = false;
        return; // ó código abaixo não é executado antes do jogo começar
    }
    else
    {
        gameManagerInstance = true;
    }
    if (gameManagerInstance != gameManagerInstanceAnt)
    {
        limitR = limitRCalib;
        limitL = limitLCalib;
        encZero = encZeroCalib;
        Debug.Log(encZeroCalib);
        zonaMeioL = Mathf.Floor((limitL - encZero) * 0.15f);
        zonaMeioR = Mathf.Floor((limitR - encZero) * 0.15f);
    }

    if (isPaused == false)
    {
        EncoderBoolR();
        EncoderBoolL();
        if (encPos >= zonaMeioL && encPos <= zonaMeioR) { encoderBoolR = false;
encoderBoolL = false; pEncL = 0; pEncR = 0; }

        encInput = pEncL + pEncR;
        //Debug.Log("encInput = " + encInput + " |pEncL = " + pEncL + "pEncR = "
+ pEncR);

        horizontalInput = Input.GetAxis("Horizontal");
        //inputTecl = (int)(horizontalInput * 100); // -tecMot

        // se houver input do encoder realInput serão os valores dele, caso
contrário o controle é devolvido as setas do teclado
        if (encoderBoolR == false && encoderBoolL == false)

```

```

        {
            realInput = horizontalInput; //Debug.Log("tecla");
            //realInput = 0; // -tecMot
        }
        else
        {
            realInput = encInput; //Debug.Log("encoder");
        }
        //Debug.Log(horizontalInput); // + " + " + encoderBoolL + " + " +
encoderBoolR);
        // Move o jogador na direção do input
        //Debug.Log("RealInput = " + realInput);
        rb.transform.Translate(Vector3.right * realInput * _speed *
Time.deltaTime); // realInput é escolhido de acordo com o input recebido

        // Mantém o jogador dentro dos limites horizontais
        if (rb.transform.position.x > _horizontalBoundaryR)
        {
            rb.transform.position = new Vector3(_horizontalBoundaryR,
rb.transform.position.y, rb.transform.position.z);
        }
        else if (rb.transform.position.x < _horizontalBoundaryL)
        {
            rb.transform.position = new Vector3(_horizontalBoundaryL,
rb.transform.position.y, rb.transform.position.z);
        }

        contadorPista = contadorPistaB + contadorPistaG + contadorPistaR;
    }
    else return;
    encoderAnt = encPos;
    pEncL = 0;
    pEncR = 0;
    gameManagerInstanceAnt = gameManagerInstance;
}
private void OnEnable()
{
    transform.position = new Vector3(0, -2.97f, 0);
}
private void OnTriggerStay2D(Collider2D other)
{
    // tvz compensa usar um método aqui dps
    if (other.gameObject.CompareTag("PG"))
    {
        if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
        {
            contadorPistaG += 5 * Time.deltaTime;
        }
        if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
        {
            contadorPistaG += 0.5f * Time.deltaTime;
        }
        if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
        {
            contadorPistaG -= 1f * Time.deltaTime;
        }
        //Debug.Log(((int)contadorPistaG));
    }
    if (other.gameObject.CompareTag("PR"))
    {
        if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
        {
            contadorPistaR -= 2 * Time.deltaTime;
        }
    }
}

```

```

    }
    if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
    {
        contadorPistaR -= 1f * Time.deltaTime;
    }
    if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
    {
        contadorPistaR += 5f * Time.deltaTime;
    }
    //Debug.Log(((int)contadorPistaR));
}
if (other.gameObject.CompareTag("PB"))
{
    if (GameManager.Instance.caso == 1 || GameManager.Instance.caso == 6)
    {
        contadorPistaB += 0.5f * Time.deltaTime;
    }
    if (GameManager.Instance.caso == 3 || GameManager.Instance.caso == 5)
    {
        contadorPistaB += 5f * Time.deltaTime;
    }
    if (GameManager.Instance.caso == 2 || GameManager.Instance.caso == 4)
    {
        contadorPistaB += 0.5f * Time.deltaTime;
    }
    //Debug.Log(((int)contadorPistaB));
}
}

// Chamado após as atualizações físicas
private void FixedUpdate()
{
    // Ativa o corpo rígido para garantir que as colisões sejam detectadas
    rb.WakeUp();
}
}
}

```

Tricor sem IA 2:

Os códigos adiante sofrem pouca ou nenhuma alteração ao serem adaptados para trabalhar com IA.

Código 17: Menu Principal jogo Tricor

```

using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MainMenu : server_abs
{
    public GameManager gameManager;

    private void Start()
    {
        GetComponentInChildren<TMPPro.TextMeshProUGUI>().gameObject.LeanScale(new
        Vector3(1.2f, 1.2f), 0.5f)
        .setLoopPingPong(); //fazer entrar em loop
    }
    public void Play() //pro botão de play
    {
        gameManager.inicio = true;
    }
}

```

```
        modoJogo = 8;
        GetComponent<CanvasGroup>().LeanAlpha(0, 0.2f).setOnComplete(OnComplete);
    }

    private void OnComplete()
    {
        gameManager.Enable();
        Destroy(gameObject);
    }
}
```

Código 18: Gerenciamento do jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class GameManager : server_abs
{
    [SerializeField]
    private Player player;
    private Coin coin;
    private Barreira barreira;
    private static GameManager instance;
    public static GameManager Instance => instance; //virou uma propriedade, só dá
    pra usar essa Instance pra acessar a instance do GameManager, nenhum outro cdg pode
    alterar a referência instance só o GameManager

    public bool gameOver;
    public bool inicio/* = true*/;
    public bool pistas = false;

    [SerializeField]
    private GameObject _pistaR;
    [SerializeField]
    private GameObject _pistaB;
    [SerializeField]
    private GameObject _pistaG;
    [SerializeField]
    private GameObject playerObj;

    private GameObject instPistaR;
    private GameObject instPistaB;
    private GameObject instPistaG;

    private float Dir = 3.6f;
    private float Cnt = 0f;
    private float Esq = -3.6f;
    public float pontos = 0;

    [SerializeField]
    private float tmpTrocaInicio = 6f;
    [SerializeField]
    private float tmpTrocaFim = 7f;
    [SerializeField]
    private float mapspeed2;

    private Coroutine pistasCoroutine;
    private Coroutine startCoroutine;

    public int caso;
    public float targetTimeRate = 0.5f;

    [SerializeField]
    private TMPro.TextMeshProUGUI scoreText; //Teve sua classe ajustada para
    combinar com a UI
    [SerializeField]
    private Image backgroundMenu;

    //Coin Generator
    [SerializeField]
```

```

private bool _spawncoin = true;
[SerializeField]
private GameObject _usd_blue, _eur_red, _jpy_green, /* _jpy, _uah, _gbp, _eur, */
_item;
[SerializeField]
private GameObject _barreira;
[SerializeField]
private GameObject gameOverMenu;

// Teste para diferente fps
// void Awake()
// {
// #if UNITY_EDITOR
//     QualitySettings.vSyncCount = 0; // VSync must be disabled
//     Application.targetFrameRate = 0;
// #endif
// }
// Start is called before the first frame update
void Start()
{
    instance = this;
    Application.targetFrameRate = 60;
    inicio = true;
    Debug.Log(inicio);
    startCoroutine = StartCoroutine(StartCorroutine());
}

// Update is called once per frame
void Update()
{
    if (Input.GetKeyUp(KeyCode.Escape))
    {
        //Time.timeScale = 0; //segundos por segundos reais -> dá pra acessar em
        //Edit/Project Settings/Time/ Aí vc tem que a freq. de calc de física por time Scale
        //recomendado)
        if (Time.timeScale == 0)
        {
            UnPause();
        }
        if (Time.timeScale == 1)
        {
            Pause();
        }
    }
    if (pistas == true)
    {
        SpawnItem();
    }
    Score();
    scoreText.text = ((int)pontos).ToString();
    //Debug.Log(pontos);
}

private void OnEnable2() //sempre que um objeto é ativado essa função entra
em ação // Acaba que o Start Roda Depois do OnEnable,
prestar atenção nisso
{
    gameOver = false;
    UnPause();
    Debug.Log("Onenable");
    playerObj.SetActive(true);
    //playerObj.transform.position = new Vector3(0, -2.97f, 0);
}

```

```

        Destroy(GameObject.FindWithTag("Coin"));
    Destroy(GameObject.FindWithTag("Barreira"));
    pistas = false;
    scoreText.text = "0";
    contadorMoedas = 0;
    errosBarreira = 0;
    player.contadorPistaB = 0;
    player.contadorPistaG = 0;
    player.contadorPistaR = 0;
    //Debug.Log(contadorMoedas); Debug.Log(errosBarreira);
    Debug.Log(player.contadorPista);
    Debug.Log(inicio);
    if (inicio == false)
    {
        startCoroutine = StartCoroutine(StartCorroutine());
        Debug.Log("recomeçou");
    }
}

void SpawnItem()
{
    if (GameObject.FindWithTag("Coin") == null &&
    GameObject.FindWithTag("Barreira") == null)
    {
        float _rnd = Random.value;
        if (_rnd < 1.0f && _rnd >= 0.55f) { _item = _barreira; } //0.45
        else if (_rnd < 0.55f)
        {
            if (caso2 == 1 || caso2 == 6) { _item = _jpy_green; } //trocar
green pra gray
            else if (caso2 == 2 || caso2 == 4) { _item = _eur_red; } //trocar
red pra yellow
            else if (caso2 == 3 || caso2 == 5) { _item = _usd_blue; }
        }
        float _rnd2 = Random.value;

        if (_rnd2 <= 0.333)
        {
            Instantiate(_item, Vector3.right * Esq + Vector3.up * 8f,
Quaternion.identity);
            xcoin = 1;
        }
        else if (_rnd2 > 0.333 && _rnd2 < 0.666)
        {
            Instantiate(_item, Vector3.right * Cnt + Vector3.up * 8f,
Quaternion.identity);
            xcoin = 2;
        }
        else if (_rnd2 >= 0.666)
        {
            Instantiate(_item, Vector3.right * Dir + Vector3.up * 8f,
Quaternion.identity);
            xcoin = 3;
        }
    }
}

private void Score()
{
    //contadorLagoM = player.contadorLago - contL;
    //contadorChuvaM = Chuva.contadorChuva - contC;
    //tempo = Time.time - tempo2;
    //pontos = tempo * 2 - contadorLagoM * 1.5f - contadorChuvaM / 10;
}

```

```

        pontos = player.contadorPista + contadorMoedas - errosBarreira;
        //if (pontos < 0) { tempo2 = Time.time; contL = player.contadorLago; contC =
        Chuva.contadorChuva; }
        //Debug.Log(((int)pontos));
    }

    private void UnPause() // só pra criar um método em vez de ter que chamar toda
hr
    {
        targetTimeScale = 1f;
        isPaused = false;
        LeanTween.value(Time.timeScale, targetTimeScale, targetTimeRate) //
        Vamos utilizar o LeanTween para fazer o Pause parar o mundo nesse intervalo de tempo
        .setOnUpdate(SetTimeScale) // Precisamos então que toda que que o
        "Update" rodar isso aqui esse método seja chamado
        .setIgnoreTimeScale(true); // Mas como o próprio LeanTween utiliza o
        timeScale em sua animações, utilizá-los juntos poderia
        // causar problema de demorar muito mais
        para pausar
        //StartCoroutine(ScaleTime(0, 1, 0.5f)); //
        Essa corrotina foi substituída pelo LeanTween
        backgroundMenu.gameObject.SetActive(false);
        //mapspeed = mapspeed2;
    }

    private void Pause()
    {
        targetTimeScale = 0;
        isPaused = true;
        //mapspeed2 = mapspeed;
        LeanTween.value(Time.timeScale, targetTimeScale,
targetTimeRate).setOnUpdate(SetTimeScale).setIgnoreTimeScale(true);
        //StartCoroutine(ScaleTime(1, 0, 0.5f));
        backgroundMenu.gameObject.SetActive(true);
    }

    private void SetTimeScale(float value) // esse método é utilizado para que o
    LeanTween atualize o valor do parâmetro value
        // da forma citada no Update, de forma a
    evitar o uso da corrotina e do Lerp (interpolação)
    {
        Time.timeScale = value;
        Time.fixedDeltaTime = 0.02f * value;
    }

    private IEnumerator StartCorroutine() {
        yield return new WaitForSeconds(4f);
        pistas = true;
        yield return new WaitForSeconds(5f);
        instPistaR = Instantiate(_pistaR, Vector3.right * Dir, transform.rotation);
        instPistaB = Instantiate(_pistaB, Vector3.right * Cnt, transform.rotation);
        instPistaG = Instantiate(_pistaG, Vector3.right * Esq, transform.rotation);
        //Debug.Log("oi");
        yield return new WaitForSeconds(2f);
        pistasCoroutine = StartCoroutine(PistasMov());
    }

    private IEnumerator PistasMov() {
        while (true)
        {
            // Gerar um número aleatório entre 1 e 6 para determinar o caso
            caso = Random.Range(1, 7);
            caso2 = caso;

            // Realizar a troca de posição com base no caso gerado

```

```

        switch (caso)
        {
            case 1:
                TrocarPosicao(instPistaR, instPistaB, instPistaG);
                break;
            case 2:
                TrocarPosicao(instPistaR, instPistaG, instPistaB);
                break;
            case 3:
                TrocarPosicao(instPistaB, instPistaR, instPistaG);
                break;
            case 4:
                TrocarPosicao(instPistaB, instPistaG, instPistaR);
                break;
            case 5:
                TrocarPosicao(instPistaG, instPistaR, instPistaB);
                break;
            case 6:
                TrocarPosicao(instPistaG, instPistaB, instPistaR);
                break;
        }
        // Aguardar um segundo antes de realizar a próxima troca
        var timeToWait = Random.Range(tmpTrocaInicio, tmpTrocaFim);
        //         //Debug.Log(timeToWait);
        yield return new WaitForSeconds(timeToWait); //Unity vai executar
        esse método num loop, aí precisamos retornar alguma coisa, no caso nulo

        yield return PistasMov(); // pra chamar o método de novo, recursivo
    }
}

private void TrocarPosicao(GameObject pistaEsq, GameObject pistaCnt, GameObject
pistaDir)
{
    pistaEsq.transform.position = new Vector3(Esq, 0, 0);
    pistaCnt.transform.position = new Vector3(Cnt, 0, 0);
    pistaDir.transform.position = new Vector3(Dir, 0, 0);
    //Debug.Log("trocou");
}

private void ApagarPistas()
{
    Destroy(instPistaB); Destroy(instPistaG); Destroy(instPistaR);
    Debug.Log("apagou");
}

public void GameOver()
{
    StopCoroutine(pistasCoroutine);
    ApagarPistas();
    gameOver = true;
    inicio = false;
    Debug.Log(inicio);
    gameOverMenu.SetActive(true);
    playerObj.SetActive(false);
}

public void Enable()
{
    gameObject.SetActive(true); //pra poder ativar o GameManager quando o jogo
começar
    Debug.Log(inicio);
    OnEnable2();
    Debug.Log("enable");
}
}

```

Código 19: Menu de Fim de Jogo do jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

public class GameOverMenu : server_abs
{
    [SerializeField]
    private GameObject PauseMenu;
    public void Restart()
    {
        gameObject.SetActive(false);
        Debug.Log("voltou");
        GameManager.Instance.Enable();
    }
    public void Finalizar()
    {
        GameManager.Instance.GameOver();
        PauseMenu.SetActive(false);
    }
    public void Sair()
    {
#if UNITY_EDITOR
        EditorApplication.isPlaying = false; //Faz parte do UnityEditor -> dá pra
        apertar Ctrl + . e colocar lá em cima
#else
        Application.Quit();
#endif
    }
}
```

Código 20: Comunicação UDP do Jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Threading;
using System.Net.Sockets;
using System.Net;
using Codice.Client.Common;
using System.Text;
using System;
using UnityEngine.XR;

public class UDP_1 : server_abs
{
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*

    public static int ValueEncoder = 0;

    Player player;
    GameManager gameManager;

    public int playerPositX = 0;
    public int pontos = 0;
    public int scrBar = 0;
    public int scrCoin = 0;
    public int scrCor = 0;
    public int scrCorB = 0;
    public int scrCorG = 0; //Gray
    public int scrCorR = 0; //Yellow
    public int posCor = 0;
    public int gameMode = 0;
    //public int inputTec12 = 0; // -tecMot

    private Thread udpThread;
    private volatile bool _shouldStop = false;

    // Start is called before the first frame update
    void Start()
    {
        instance = this;
        player = GameObject.Find("Player").GetComponent<Player>();
    }

    void Awake()
    {
        udpClient = new UdpClient();
        matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
unitySendPort); // Endereço e porta do servidor MATLAB
        udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
Configurar a porta de recebimento do Unity

        // Iniciar a thread para recepção assíncrona de dados UDP
        udpThread = new Thread(new ThreadStart(Comunicacao));
```

```

        udpThread.IsBackground = true;
        udpThread.Start();

    }

    // Update is called once per frame
    void Update()
    {

    }

    private void FixedUpdate()
    {
        gameMode = modoJogo;
        if (GameManager.Instance == null)
        {
            return;
        }

        pontos = (int)GameManager.Instance.pontos;
        playerPositX = (int)(player.gameObject.transform.position.x * 100);
        scrBar = errosBarreira;
        scrCoin = contadorMoedas;
        scrCor = (int)player.contadorPista;
        scrCorB = (int)player.contadorPistaB;
        scrCorG = (int)player.contadorPistaG; //gray
        scrCorR = (int)player.contadorPistaR; //yellow
        posCor = caso2;

        //inputTec12 = inputTec1; // -tecMot
    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{pontos}!{playerPositX}!{scrBar}!{scrCoin}!{scrCor}!{scrCorB}!{scrCorG}!{scrCorR}!
                {posCor}!{gameMode}"; //!{inputTec12}"; // -tecMot
                /**/
                //Debug.Log("Horiz.Input: " + inputTec12);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto periodo antes do próximo envio (evitar envio
                excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }

            // Receber dados do MATLAB
            try
            {
                IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
                unityReceivePort);

```

```

byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
string receivedString = Encoding.ASCII.GetString(receivedData);

// Debug para visualizar os dados recebidos no console do Unity
Debug.Log("Dados recebidos do MATLAB: " + receivedString);

//ValueEncoder = int.Parse(receivedString);
// Converter para int e verificar se é NaN
int parsedValue;
// Tenta converter a string receivedString em um valor inteiro e
armazena o resultado na variável parsedValue.
// O método TryParse retorna true se a conversão for bem - sucedida
e false se falhar.
bool parseSuccess = int.TryParse(receivedString, out parsedValue);
if (parseSuccess && !double.IsNaN(parsedValue))
{
    ValueEncoder = parsedValue;
}
else
{
    ValueEncoder = 0;
}
}
catch (Exception e)
{
    Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
}
}
Debug.Log("Thread Aqui");
}

void OnDestroy()
{
    _shouldStop = true;
}

public void Enable()
{
    gameObject.SetActive(true);
    _shouldStop = false;
    Debug.Log("Inicio");
    //stopThread = false;
}

public void Disable()
{
    _shouldStop = true;
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
    gameObject.SetActive(false);
    Debug.Log("Fim");
}
}
}

```

Código 21: Funcionamento do ambiente no jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Map : server_abs
{
    [SerializeField]
    private GameObject _rua;
    [SerializeField]
    private bool _nexttile = true;
    [SerializeField]
    private GameManager gameManager;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        if (isPaused == false)
        {
            transform.Translate(Vector3.down * mapspeed);
            if (transform.position.y < 0 && _nexttile)
            {
                Instantiate(_rua, Vector3.up * 10f + Vector3.right * 0f,
transform.rotation);
                _nexttile = false;
            }
            else if (transform.position.y < -10f)
            {
                Destroy(gameObject);
            }
        }
        else if (isPaused == true)
        {
            Time.timeScale = Mathf.Lerp(Time.timeScale, targetTimeScale,
gameManager.targetTimeRate * Time.deltaTime); // meio inútil no momento
        }
    }
}
```

Código 22: Funcionamento das moedas no jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.PackageManager;
using UnityEngine;

public class Coin : server_abs
{
    [SerializeField]
    private int pontosMoeda = 10;
    [SerializeField]
    private int pontosErroMoeda = 5;

    void Awake()
    {
        new_coin = 1;
    }
    // Update is called once per frame
    void Update()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        if (isPaused == false)
        {
            transform.Translate(Vector3.down * mapspeed);
            BoundControl();
        }
        else return;
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.name == "Player")
        {
            contadorMoedas+= 1 * pontosMoeda;
            //Debug.Log(contadorMoedas);
            Destroy(gameObject);
        }
    }
    void BoundControl()
    {
        if (transform.position.y < -7)
        {
            errosMoedas += 1 * pontosErroMoeda;
            //Debug.Log(errosMoedas);
            Destroy(gameObject);
        }
    }
}
```

Código 23: Funcionamento das barreiras no jogo Tricor

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Barreira : server_abs
{
    [SerializeField]
    private int pontosBarreira = 5;
    [SerializeField]
    private int pontosErroBarreira = 15;

    [SerializeField]
    private GameObject _cercaQuebrada;

    private GameObject cercaQuebrada = null;

    private bool quebrou = false;

    // Start is called before the first frame update
    void Start()
    {

    }

    // Update is called once per frame
    void Update()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        //Debug.Log(quebrou);
        if (isPaused == false)
        {
            if ((quebrou == true) && (cercaQuebrada != null))
            {
                //Debug.Log("andou");
                cercaQuebrada.transform.Translate(Vector3.down * mapspeed);
                BoundControl();
            }
            else if (quebrou == false)
            {
                transform.Translate(Vector3.down * mapspeed);
                BoundControl();
            }
        }
        else return;
    }

    void OnTriggerEnter2D(Collider2D col)
    {
        if (col.name == "Player")
        {
            errosBarreira += 1 * pontosErroBarreira;
            //Debug.Log(errosBarreira);
            //Debug.Log(quebrou);
            ReplaceBarrier();
            //Destroy(gameObject);
        }
    }
}
```

```

    }
    void BoundControl()
    {
        if (transform.position.y < -7)
        {
            contadorBarreira += 1 * pontosBarreira;
            Destroy(gameObject);
        }
    }
    void BoundControl2()
    {
        if (cercaQuebrada.transform.position.y < -7)
        {
            Destroy(cercaQuebrada);
            Debug.Log("destruiu");
            quebrou = false;
            //cercaQuebrada = null;
            Destroy(gameObject);
        }
    }
    void ReplaceBarrier()
    {
        quebrou = true;
        Vector3 spawnPosition = transform.position;
        Quaternion spawnRotation = transform.rotation;
        transform.position = new Vector3(0, -6.9f, 0);
        //gameObject.SetActive(false);
        //Destroy(gameObject);

        cercaQuebrada = Instantiate(_cercaQuebrada, spawnPosition, spawnRotation);
        Debug.Log("spawnou");
        Debug.Log(quebrou);
    }
}

```

Jogo Floating Board

Código 24: Comunicação UDP do jogo Floating Board

```
using System.Collections;
using System.Collections.Generic;
using System.Net.Sockets;
using System.Net;
using UnityEngine;
using System.Threading;
using Unity.VisualScripting;
using System.Text;
using System;
using UnityEditor.PackageManager;

public class UDP_1 : MonoBehaviour
{
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*

    public static int ValueEncoder = 0;

    Player player;
    Chuva chuva;
    Platchao plataforma;

    public int playerPositX = 0;
    public int playerPositY = 0;
    public int pontos = 0;
    public int contChuva = 0;
    public int contLago = 0;
    public int posPrancha = 0;

    private Thread udpThread;
    private volatile bool _shouldStop = false;

    // Start is called before the first frame update
    void Start()
    {
        instance = this;
        player = GameObject.Find("Player").GetComponent<Player>();
        chuva = GameObject.Find("Chuva").GetComponent<Chuva>();
        plataforma = GameObject.Find("Plataforma").GetComponent<Platchao>();
    }

    void Awake()
    {
        udpClient = new UdpClient();
        matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
unitySendPort); // Endereço e porta do servidor MATLAB
        udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
Configurar a porta de recebimento do Unity
    }
}
```

```

        // Iniciar a thread para recepção assíncrona de dados UDP
        udpThread = new Thread(new ThreadStart(Comunicacao));
        udpThread.IsBackground = true;
        udpThread.Start();

    }

    // Update is called once per frame
    void Update()
    {

    }

    private void FixedUpdate()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        playerPositX = (int) (player.gameObject.transform.position.x * 100);
        playerPositY = (int) (player.gameObject.transform.position.y * 100);
        pontos = (int) GameManager.Instance.pontos;
        contChuva = chuva.contadorChuva;
        contLago = (int) player.contadorLago;
        posPrancha = (int) (plataforma.transform.rotation.eulerAngles.z * 100);

    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{pontos}!{playerPositX}!{playerPositY}!{contChuva}!{contLago}!{posPrancha}";
                /**/
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto período antes do próximo envio (evitar envio
                // excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }

            // Receber dados do MATLAB
            try
            {
                IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
                unityReceivePort);
                byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
                string receivedString = Encoding.ASCII.GetString(receivedData);

                // Debug para visualizar os dados recebidos no console do Unity
                Debug.Log("Dados recebidos do MATLAB: " + receivedString);
            }
            catch { }
        }
    }

```

```

        //ValueEncoder = int.Parse(receivedString);
        // Converter para int e verificar se é NaN
        int parsedValue;
        // Tenta converter a string receivedString em um valor inteiro e
armazena o resultado na variável parsedValue.
        // O método TryParse retorna true se a conversão for bem - sucedida
e false se falhar.
        bool parseSuccess = int.TryParse(receivedString, out parsedValue);
        if (parseSuccess && !double.IsNaN(parsedValue))
        {
            ValueEncoder = parsedValue;
        }
        else
        {
            ValueEncoder = 0;
        }
    }
    catch (Exception e)
    {
        Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
    }
    Debug.Log("Thread Aqui");
}

void OnDestroy()
{
    _shouldStop = true;
}

public void Enable()
{
    gameObject.SetActive(true);
    _shouldStop = false;
    Debug.Log("Inicio");
    //stopThread = false;
}

public void Disable()
{
    _shouldStop = true;
    if (udpClient != null)
    {
        udpClient.Close();
        udpClient = null;
    }
    gameObject.SetActive(false);
    Debug.Log("Fim");
}
}

```

Código 25: Player do Jogo Floating Board

```

using System;
using System.Collections;
using System.Collections.Generic;
using System.Linq;
using System.Runtime.CompilerServices;

```

```

using Unity.VisualScripting;
using UnityEngine;
//using Cinemachine;

public class Player : MonoBehaviour
{
    public float forceMultiplier = 400f; //coloca lá no campo do Script no inspector
do player; assim não precisa compilar o cdg denovo pra mudar a força
    public float maximumVelocity = 10f;
    public float torqueMultiplier = 125f;
    public float maximumAngularVelocity = 100f;
    public float contadorLago = 0;

    // Encoder
    public float encoderAnt = 0;
    public bool encoderBoolR = false;
    public int limitR = 400; // deve ser pegado do valor UDP_1.ValueEncoder
    public float pEncR = 0;
    public bool encoderBoolL = false;
    public int limitL = -400;
    public float pEncL = 0;
    public float zonaMeio = 50;
    public float encInput = 0;
    public int encZero = 0;
    public float encPos = 0;
    public float mediaEnc = 0;
    public float realInput = 0;

    //teste anterior
    private const int MediaMovelTamanho = 4; // dependente do fps para possibilitar
uma responsividade maior do jogo
    private Queue<float> valoresAnteriores = new Queue<float>();

    //public ParticleSystem deathParticles;

    public Rigidbody2D rb; // pra evitar de pesar o update
    //private CinemachineImpulseSource cinemachineImpulseSource;

    void Awake() //trocamos o Start pelo Awake pq o rb não estava sendo declarado
após o Restart no onEnable()
    {
        rb = GetComponent<Rigidbody2D>(); //caching -> salvando esse componente pra
utilizar ele dentro do update
        transform.rotation = Quaternion.identity; //Euler(0,0,45);
        transform.position = new Vector2(0, -2.25f);
        rb.velocity = Vector2.zero;
        //cinemachineImpulseSource = GetComponent<CinemachineImpulseSource>();
    }
    // começa a mexer pra direita | > 50 : enquanto movimenta pra d. cubo gira pra
d., se contrário tbm| > 300: só gira pra d. (se parar perde força)|
    // >500: se parar cubo continua girando pra d.| limite é 600
    //apertar botão ao poucos: enquanto tiver fora do zero da sendo apertado pra um
lado ou pro outro -> preocupação: tremedeira, não apresentar variação
    //no jogo se a variação for muito pequena e não se mantiver por muito tempo
    //
    private void SuavizarEncoder()
    {
        valoresAnteriores.Enqueue(encPos); //Adiciona o valor atual do encoder
(encPos) à fila

        if (valoresAnteriores.Count > MediaMovelTamanho)

```

```

        valoresAnteriores.Dequeue(); //remove o valor mais antigo da fila (o
valor no início da fila)

        mediaEnc = valoresAnteriores.Average();

    }

    private void EncoderBoolR()
    {
        //if (encPos == encoderAnt) { encoderBoolR = false; }
        if(encPos > zonaMeio)
        {
            encoderBoolR = true; encoderBoolL = false;
            pEncR = Mathf.Abs(encPos/(limitR - encZero));
            //Debug.Log(pEncR);
            if (pEncR > 1) { pEncR = 1; }
            if (pEncR < 0) { pEncR = 0; }
        }
    }

    private void EncoderBoolL()
    {
        if (encPos < -zonaMeio)
        {
            encoderBoolR = false; encoderBoolL = true;
            pEncL = -(Mathf.Abs(encPos / (limitL - encZero)));
            if (pEncL < -1) { pEncL = -1; }
            if (pEncL > 0) { pEncL = 0; }
        }
    }

    // Update is called once per frame
    void Update()
    {
        encPos = UDP_1.ValueEncoder - encZero;
        if (GameManager.Instance == null) //instance foi setado como this lá no
Gamemanager Start -> ou seja, antes de executar o GameManager o player não se move
        {
            return; //ou seja não será executado o código abaixo disso
        }

        //SuavizarEncoder();
        EncoderBoolR();
        EncoderBoolL();
        if(encPos >= -zonaMeio && encPos <=zonaMeio) { encoderBoolR = false;
encoderBoolL = false; pEncL = 0; pEncR = 0; }
        encInput = pEncL + pEncR;

        //Debug.Log("Passando Player");
        var HorizontalInput = Input.GetAxis("Horizontal");
        //float torqueDirection = Mathf.Sign(HorizontalInput);
        //rb.AddTorque(-torqueDirection * torqueMultiplier * Time.deltaTime);
        //Debug.Log(HorizontalInput); // + " " + encoderBoolL + " " +
encoderBoolR);
        //Debug.Log(pEncL + " e " + pEncR);

        if (encoderBoolR == false && encoderBoolL == false)
        {
            realInput = HorizontalInput; //Debug.Log("tecla");
        }
        else
        {

```

```

        realInput = encInput; //Debug.Log("encoder");
    }
    //Debug.Log(contador);
    Debug.Log(realInput);
    Debug.Log(encPos);
    if (rb.velocity.magnitude <= maximumVelocity &&
Mathf.Abs(rb.angularVelocity) <= maximumAngularVelocity) //não é recomendado o
método GetComponent aqui pq ele é um pouco pesado
    {
        rb.AddForce(new Vector2(realInput * forceMultiplier * Time.deltaTime,
0)); // Se vc deixar só isso (s/ time.deltaTime), a movimentação dependerá do seu
fps
        rb.AddTorque(-realInput * torqueMultiplier * Time.deltaTime);
    }
    encoderAnt = encPos;
    pEncL = 0;
    pEncR = 0;
}

private void OnTriggerStay2D(Collider2D other)
{
    if (other.gameObject.CompareTag("Lago"))
    {
        contadorLago += 1*Time.deltaTime;
        //Debug.Log((int)contadorLago);
    }
}

public void Desenable()
{
    gameObject.SetActive(false);
}

public void Enable()
{
    gameObject.SetActive(true);
}

private void OnEnable()
{
    contadorLago = 0;
    transform.rotation = Quaternion.identity;//Euler(0,0,45);
    transform.position = new Vector2(0, -2.25f);
    rb.velocity = Vector2.zero;
}
}

```

Código 26: Menu principal do jogo Floating Board

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class MainMenu : MonoBehaviour
{
    public GameManager gameManager;

    public void Awake()
    {
#if UNITY_EDITOR
        QualitySettings.vSyncCount = 0; // VSync must be disabled
        Application.targetFrameRate = 30;
#endif
    }

    public void Play()
    {
        Debug.Log("Botao apertado");
        gameManager.Enable();
        Destroy(gameObject);
    }

    void Update()
    {
        //Debug.Log(Time.time);
    }
}
```

Código 27: Menu de Fim de Jogo do jogo Floating Board

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor;
using UnityEngine;

public class GameOverMenu : MonoBehaviour
{
    public void Restart()
    {
        GameManager.Instance.Enable();
        gameObject.SetActive(false);
    }

    public void Quit()
    {
#if UNITY_EDITOR
        EditorApplication.isPlaying = false;
#else
        Application.Quit(); //para qd tiver fora do unity
#endif
    }
}
```

Código 28: Gerenciador de Jogo do jogo Floating Board

```
using System.Collections;
using System.Collections.Generic;
using UnityEditor.Experimental.GraphView;
using UnityEngine;
using UnityEngine.Timeline;
using UnityEngine.UI;

public class GameManager : MonoBehaviour
{
    //public float maxRotation = 30f;
    //public float minRotation = -30f;
    private Coroutine rotatePlataformaCoroutine;
    private float currentRotation;

    public float pontos = 0;
    private float contadorLagoM;
    private float contadorChuvaM;
    private float contL = 0;
    private float contC = 0;
    private float tempo;
    private float tempo2 = 0;
    private float tempoStart = 0;
    public float posPlataforma = 0.00f;

    public TMPro.TextMeshProUGUI scoreText;
    public Image backgroundMenu;
    public bool gameOver = true;
    public static bool firstTime = true;
    public GameObject gameOverMenu;

    [SerializeField] private Player player;
    [SerializeField] private Chuva chuva;
    [SerializeField] private Platchao Plataforma;

    private static GameManager instance = null;
    private float pontoFinal;

    public static GameManager Instance => instance;

    // Start is called before the first frame update
    void Start()
    {
        Debug.Log("Começou");
        instance = this;
        chuva = GameObject.Find("Chuva").GetComponent<Chuva>();
    }

    private void Awake()
    {
        #if UNITY_EDITOR
            QualitySettings.vSyncCount = 0; // VSync must be disabled
            Application.targetFrameRate = 60;
        #endif
    }
}
```

```

// Update is called once per frame
void Update()
{
    //Debug.Log("Com 2");
    if (Input.GetKeyDown(KeyCode.Escape))
    {
        if(Time.timeScale == 0)
        {
            StartCoroutine(ScaleTime(0, 1, 0.5f));
            backgroundMenu.gameObject.SetActive(false);
        }
        if(Time.timeScale == 1)
        {
            StartCoroutine(ScaleTime(1, 0, 0.5f));
            backgroundMenu.gameObject.SetActive(true);
        }
    }
    if (gameOver) { return; }
    Score();
    scoreText.text = ((int)pontos).ToString();
    IsPlayerOn();
    //Debug.Log(scoreText);
}

private void IsPlayerOn()
{
    if (player.transform.position.y <= -12)
    {
        //Destroy(player.gameObject);
        GameOver();
    }
}

private void Score()
{
    //Debug.Log("1: " + Time.time);
    //Debug.Log("Com 3");
    contadorLagoM = player.contadorLago - contL;
    contadorChuvaM = chuva.contadorChuva - contC;
    // Considerar tempo qd se clica em Jogar
    if (firstTime) { tempoStart = Time.time; firstTime = false; }
    tempo = Time.time - tempo2 - tempoStart;
    pontos = tempo * 2 - contadorLagoM*1.5f - contadorChuvaM/10;
    if (pontos < 0) { tempo2 = Time.time; contL = player.contadorLago; contC =
chuva.contadorChuva; }
    //Debug.Log(tempo);
    //Debug.Log(((int)pontos));
}

//private IEnumerator rotatePlataforma() // isso é um método, é só clicar no
carinha ali em cima e fz alguma coisa
// // permite a gente escrever qualquer coisa
que a gente quiser entre os yield returns
//{
//    float[] angulos = {0, -20, -30, 20, 30 };
//    float targetRotation = angulos[Random.Range(0, angulos.Length)];
//    Debug.Log(targetRotation);
//    //float targetRotationRadians = targetRotation * Mathf.Deg2Rad;
//    float sinal = Mathf.Sign(targetRotation - transform.rotation.z);
//    float sinal = Mathf.Sign(Mathf.DeltaAngle(transform.eulerAngles.z,
targetRotation));
//    float rotationStep = sinal * 0.9f *60f;

```

```

        // while (Quaternion.Angle(transform.rotation, Quaternion.Euler(0f, 0f,
targetRotation)) > 0.01f)
        // {
        //     Debug.Log(Quaternion.Angle(transform.rotation, Quaternion.Euler(0f,
0f, targetRotation)));
        //     rotationStep = sinal * 0.9f * 60f;
        //     Debug.Log(sinal);
        //     transform.Rotate(Vector3.forward, rotationStep*Time.deltaTime);
        //     yield return null;
        // }
        // transform.rotation = Quaternion.Euler(0f, 0f, targetRotation);

        // var timeToWait = Random.Range(2f, 3f);
        // Debug.Log(timeToWait);
        // yield return new WaitForSeconds(timeToWait); //Unity vai executar esse
método num loop, aí precisamos retornar alguma coisa, no caso nulo

        // yield return rotatePlataforma(); // pra chamar o método de novo, recursivo
        //}
        private IEnumerator rotatePlataforma() // isso é um método, é só clicar no
carinha ali em cima e fz alguma coisa
        // permite a gente escrever qualquer
coisa que a gente quiser entre os yield returns
        {
            //Debug.Log("Rotate Plat");
            float[] angulos = { 0, -20, -30, 20, 30 };
            float targetRotation = angulos[Random.Range(0, angulos.Length)];
            //Debug.Log(targetRotation);

            Quaternion targetQuaternion = Quaternion.Euler(0f, 0f, targetRotation);

            while (Quaternion.Angle(Plataforma.transform.rotation, targetQuaternion) >
0.01f)
            {
                Quaternion rotation =
Quaternion.RotateTowards(Plataforma.transform.rotation, targetQuaternion, 0.1f * 60f
* Time.deltaTime);
                Plataforma.transform.rotation = rotation;

                yield return null;
            }

            Plataforma.transform.rotation = targetQuaternion;

            var timeToWait = Random.Range(5f, 7f);
            //Debug.Log(timeToWait);
            yield return new WaitForSeconds(timeToWait); //Unity vai executar esse
método num loop, aí precisamos retornar alguma coisa, no caso nulo

            yield return rotatePlataforma(); // pra chamar o método de novo,
recursivo
        }

IEnumerator ScaleTime(float start, float end, float duration)
{
    //Debug.Log("Scale Time");
    float lastTime = Time.realtimeSinceStartup;
    float timer = 0.0f;
    while(timer < duration)
    {
        Time.timeScale = Mathf.Lerp(start, end, timer / duration);
        Time.fixedDeltaTime = 0.02f * Time.timeScale;
        timer += (Time.realtimeSinceStartup - lastTime);
    }
}

```

```

        lastTime = Time.realtimeSinceStartup;
        yield return null;
    }
    Time.timeScale = end;
    Time.fixedDeltaTime = 0.02f * Time.timeScale;
}

public void GameOver()
{
    gameOver = true;
    gameOverMenu.SetActive(true);
    StopCoroutine(rotatePlataformaCoroutine);
    chuva.Desenable();
    player.Desenable();
    pontoFinal = pontos;
    gameObject.SetActive(false);
}

public void Enable()
{
    gameObject.SetActive(true);
}

private void OnEnable()
{
    gameOver = false;
    Plataforma.transform.rotation = Quaternion.Euler(0f, 0f, 0f); // plataforma
na posição inicial
    rotatePlataformaCoroutine = StartCoroutine(rotatePlataforma());
    //setando ponto pra zero
    chuva.Enable();
    player.Enable();
    firstTime = true;
    scoreText.text = "0";
    contL = 0;
    contC = 0;
    pontos = 0;
}
}

```

Código 29: Controle da chuva no jogo Floating Board

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Chuva : MonoBehaviour
{
    [SerializeField] private GameObject plat;
    public float angulo_z;
    public int posicao;
    public int contadorChuva = 0;
    ParticleSystem ps;
    //List<ParticleSystem.Particle> outside = new List<ParticleSystem.Particle>();
    List<ParticleCollisionEvent> colisao = new List<ParticleCollisionEvent>();

    // Start is called before the first frame update
    void Start()
    {
        Quaternion rotacao = plat.transform.rotation;
        angulo_z = rotacao.z;
        //ps = GetComponent<ParticleSystem>();
        //ps.startColor = new Color32(0, 25, 255, 200);
    }

    // Update is called once per frame
    void Update()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        //Debug.Log("Passando Chuva");
        PosChuva();
    }

    private void OnParticleCollision(GameObject other)
    {
        if (other.CompareTag("Player")) // vê se a chuva tá colidindo com o
player
            contadorChuva++;
        //Debug.Log(contadorChuva);
        //int numCollisions = 0; // é pra tentar fazer a chuva sair
da plataforma, ignorando a colisão com as laterais
        //numCollisions = ps.GetCollisionEvents(other, colisao);
        //for (int i = 0; i < numCollisions; i++)
        //{
            //GameObject collidedObject = colisao[i].colliderComponent.gameObject;
            //if (!collidedObject.CompareTag("Platchao"))
            //{
                return;
            //}
        //}
    }

    private void PosChuva() {
        Quaternion rotacao = plat.transform.rotation;
        angulo_z = rotacao.z;
    }
}
```

```

        if (angulo_z < 0)
            posicao = 1;
        if (angulo_z > 0)
            posicao = -1;
        if (angulo_z == 0f)
            posicao = 0; // Random.Range(-1, 1);
        transform.position = new Vector3(5 * posicao, 9, 0);
    }

    public void Enable()
    {
        gameObject.SetActive(true);
    }

    private void OnEnable()
    {
        contadorChuva = 0;
    }

    public void Desenable()
    {
        gameObject.SetActive (false);
    }
}

```

Código 30: Funcionamento da plataforma no jogo Floating Board

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Platchao : MonoBehaviour
{
    private Rigidbody2D plat_chao;
    //[SerializeField] private GameObject plat;
    // Start is called before the first frame update
    void Start()
    {
        transform.rotation = gameObject.transform.rotation;
    }

    // Update is called once per frame
    void Update()
    {
        if (GameManager.Instance == null)
        {
            return;
        }
        transform.rotation = gameObject.transform.rotation;
    }
}
```

Jogo City Walk

Código 31: Comunicação UDP do jogo City Walk

```
using System;
using System.Text;
using System.Net;
using System.Net.Sockets;
using UnityEngine;
using UnityEditor.Experimental.GraphView;
using System.Collections;
using System.Threading;
using System.Drawing;

public class UDP_1 : MonoBehaviour
{
    private UdpClient udpClient;
    private IPEndPoint matlabEndPoint;
    private int unityReceivePort = 35000; // Porta para receber dados do MATLAB
    private int unitySendPort = 52000;    // Porta para enviar dados ao Unity*
    public float velocidade = 0.00f;
    public float groundNextX = 0.00f;
    public float groundNextY = 0.00f;
    public float groundAtualX = 0.00f;
    public float groundAtualY = 0.00f;
    public int obstaculos = 0;
    public int distancia = 0;
    public int moedas = 0;
    public int pulos = 0;
    public int agach = 0;
    public int eventos = 0;

    // Precisa por posição do obstáculo ainda
    public static int ValueEncoder = 0;

    //Threads
    private static UDP_1 instance;
    public static UDP_1 Instance => instance;

    //public static bool stopThread = false;
    private Thread udpThread;
    // Volatile is used as hint to the compiler that this data
    // member will be accessed by multiple threads.
    private volatile bool _shouldStop = false;

    void Start()
    {
        instance = this;
    }

    void Awake()
    {
        udpClient = new UdpClient();
        matlabEndPoint = new IPEndPoint(IPAddress.Parse("127.0.0.1"),
unitySendPort); // Endereço e porta do servidor MATLAB
        udpClient.Client.Bind(new IPEndPoint(IPAddress.Any, unityReceivePort)); //
Configurar a porta de recebimento do Unity

        // Iniciar a thread para recepção assíncrona de dados UDP
    }
}
```

```

        udpThread = new Thread(new ThreadStart(Comunicacao));
        udpThread.IsBackground = true;
        udpThread.Start();

    }

    public void OnEnable()
    {

    }

    void Update()
    {
        if (Input.GetKeyDown(KeyCode.O)) // || stopThread == true
        {
            //gameObject.SetActive(false);
            //StopThreads(udpThread);
        }
    }

    private void FixedUpdate()
    {
        eventos = Player.totalEvento;
        distancia = Mathf.FloorToInt(Player.distancia);
        moedas = Player.contadorMoeda2;
        groundAtualX = (int)(Ground.posXGround * 100);
        groundAtualY = (int)(Ground.posYGround * 100);
        velocidade = (int)(Player.velocidade * 100);
        obstaculos = Player.totalHitObstacle;
        pulos = Player.totalJump;
        agach = Player.totalAgach;
        //Debug.Log(agach);
        //Debug.Log(pulos);
        // Parece q pra inteiros não precisa verificar se são nulos pq tipos de
        // valor como int não podem conter um valor null**
        //Debug.Log(stopThread);
    }

    void Comunicacao()
    {
        while (!_shouldStop)
        {
            // Enviar dados para o MATLAB
            try
            {
                string dataToSend =
                $"{distancia}!{moedas}!{groundAtualX}!{groundAtualY}!{groundNextX}!{groundNextY}!{ve
                locidade}!{eventos}!{obstaculos}!{pulos}!{agach}";
                /**/
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                byte[] dataBytes = Encoding.ASCII.GetBytes(dataToSend);
                //Debug.Log("Tamanho de dataToSend: " + dataBytes.Length);
                udpClient.Send(dataBytes, dataBytes.Length, matlabEndPoint);

                // Aguardar um curto periodo antes do próximo envio (evitar envio
                // excessivo)
                Thread.Sleep(50);
            }
            catch (Exception e)
            {
                Debug.LogError("Erro no envio de dados UDP: " + e.ToString());
            }
        }
    }

```

```

        // Receber dados do MATLAB
        try
        {
            IPEndPoint receivedEndPoint = new IPEndPoint(IPAddress.Any,
unityReceivePort);
            byte[] receivedData = udpClient.Receive(ref receivedEndPoint);
            string receivedString = Encoding.ASCII.GetString(receivedData);

            // Debug para visualizar os dados recebidos no console do Unity
            Debug.Log("Dados recebidos do MATLAB: " + receivedString);

            //ValueEncoder = int.Parse(receivedString);
            // Converter para int e verificar se é NaN
            int parsedValue;
            // Tenta converter a string receivedString em um valor inteiro e
            // armazena o resultado na variável parsedValue.
            // O método TryParse retorna true se a conversão for bem - sucedida
            // e false se falhar.
            bool parseSuccess = int.TryParse(receivedString, out parsedValue);
            if (parseSuccess && !double.IsNaN(parsedValue))
            {
                ValueEncoder = parsedValue;
            }
            else
            {
                ValueEncoder = 0;
            }
        }
        catch (Exception e)
        {
            Debug.LogError("Erro na recepção de dados UDP: " + e.ToString());
        }
        Debug.Log("Thread Aqui");
    }

    void OnDestroy()
    {
        _shouldStop = true;
    }

    public void Enable()
    {
        gameObject.SetActive(true);
        _shouldStop = false;
        Debug.Log("Inicio");
        //stopThread = false;
    }

    public void Disable()
    {
        _shouldStop = true;
        if (udpClient != null)
        {
            udpClient.Close();
            udpClient = null;
        }
        gameObject.SetActive(false);
        Debug.Log("Fim");
    }
}

```


Código 32: Player do jogo City Walk

```
using System;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UIElements;

public class Player : MonoBehaviour
{
    public float gravity;
    public Vector2 velocity;
    public static float velocidade;
    public float acceleration = 10;
    public float maxAcceleration = 5;
    public float distance = 0;
    public float jumpVelocity = 20;
    public float maxVelocity = 500;
    public float groundHeight = 10;
    public bool isGrounded = false;
    float offSetYUp = 1.3f; //1.3f;
    float offSetYDown = 0.66f; //
    float offSetDiff = 0.64f; // -> diferença entre os dois

    public bool isHoldingJump = false;
    public float maxHoldJumpTime = 0.0f;
    public float maxMaxHoldJumpTime = 0.4f;
    public float holdJumpTimer = 0.0f;

    public int contadorMoeda = 0;
    public static int contadorMoeda2 = 0;
    public bool didHitObstacle = false;
    public static int totalHitObstacle = 0;
    public int numPunish = 0;
    public bool evento = false;
    public Vector2 velocity2;
    public int numEvento = 0;
    public static int totalEvento = 0;
    public static float distancia = 0;

    public static int totalJump = 0;
    public static int totalAgach = 0;

    public float jumpGroundGap = 2f;
    public bool isDead = false;
    //public bool _llive = true;

    public LayerMask groundLayerMask; //pra poder adicionar as duas
    camadas que foram criadas aqui e colocá-las como parâmetros no Raycast
    public LayerMask obstacleLayerMask;

    //public PlayerAnimationController playerAnim;

    public Animator playerAnimator;
    UDP_1 Udp_1;

    private bool _moveu = true;
    private bool Moveu
    {
        get { return _moveu; }
        set { _moveu = value; playerAnimator.SetBool("moveu", value); }
    }
}
```

```

    }

    private bool _correu = false;
    public bool Correu
    {
        get { return _correu; }
        set { _correu = value; playerAnimator.SetBool("correu", value); }
    }

    private bool _morreu = false;
    public bool Morreu
    {
        get { return _morreu; }
        set { _morreu = value; playerAnimator.SetBool("morreu", value); }
    }

    private bool _agachou = false;
    public bool Agachou
    {
        get { return _agachou; }
        set { _agachou = value; playerAnimator.SetBool("agachou", value); }
    }

    public bool agachouAux = true;
    public bool agachouShift = false;
    public bool agachouEncoder = false;

    public bool pulouEspaco = false;
    public bool pulouEncoder = false;
    public int encoderAnt = 0;
    public bool encoderBoolUpP = false;
    public bool encoderBoolDwP = false;
    public int limitP = 120;
    public int limitC = -120;
    void Awake()
    {
#ifdef UNITY_EDITOR
        QualitySettings.vSyncCount = 0; // VSync must be disabled
        Application.targetFrameRate = 30;
#endif
    }

    // Start is called before the first frame update
    void Start()
    {
        Moveu = true;
        Debug.Log("Moveu");
    }

    private void EncoderBoolUpP()
    {
        if (UDP_1.ValueEncoder == encoderAnt) { encoderBoolUpP = false; }
        if (encoderAnt <= limitP && UDP_1.ValueEncoder > encoderAnt &&
UDP_1.ValueEncoder > limitP) { encoderBoolUpP = true; }
        if (encoderAnt <= limitP && UDP_1.ValueEncoder > encoderAnt &&
UDP_1.ValueEncoder <= limitP) { encoderBoolUpP = false; }
        if (encoderAnt <= limitP && UDP_1.ValueEncoder < encoderAnt)
{ encoderBoolUpP = false; }
        if (encoderAnt > limitP && UDP_1.ValueEncoder > encoderAnt) { encoderBoolUpP
= false; }
        if (encoderAnt > limitP && UDP_1.ValueEncoder < encoderAnt) { encoderBoolUpP
= false; }
    }
    private void EncoderBoolDwP()

```

```

    {
        if (UDP_1.ValueEncoder == encoderAnt) { encoderBoolDwP = false; }
        if (encoderAnt > limitP && UDP_1.ValueEncoder < encoderAnt &&
UDP_1.ValueEncoder <= limitP) { encoderBoolDwP = true; }
        if (encoderAnt > limitP && UDP_1.ValueEncoder < encoderAnt &&
UDP_1.ValueEncoder > limitP) { encoderBoolDwP = false; }
        if (encoderAnt > limitP && UDP_1.ValueEncoder > encoderAnt) { encoderBoolDwP
= false; }
        if (encoderAnt <= limitP && UDP_1.ValueEncoder > encoderAnt)
{ encoderBoolDwP = false; }
        if (encoderAnt <= limitP && UDP_1.ValueEncoder < encoderAnt)
{ encoderBoolDwP = false; }
    }

    // Update is called once per frame
    void Update()
    {
        if (!isDead)
        {
            Vector2 pos = transform.position;
            float groundDistance = Mathf.Abs(pos.y - groundHeight); //distancia até
o chão considerando tamanho (no caso de um cubo) - transform.localScale.y/2

            EncoderBoolUpP();
            EncoderBoolDwP();

            if (isGrounded || groundDistance <= jumpGroundGap) //pro player poder
pular sem ter que esperar chegar totalmente no chão
            {
                if (Input.GetKeyDown(KeyCode.Space) || (encoderBoolUpP))
                {
                    if (Input.GetKeyDown(KeyCode.Space) && !encoderBoolUpP)
{ pulouEspaco = true; pulouEncoder = false; }
                    if (!Input.GetKeyDown(KeyCode.Space) && encoderBoolUpP)
{ pulouEspaco = false; pulouEncoder = true; }
                    if (Input.GetKeyDown(KeyCode.Space) && encoderBoolUpP)
{ pulouEspaco = true; pulouEncoder = true; }
                    isGrounded = false;
                    velocity.y = jumpVelocity;
                    isHoldingJump = true;
                    holdJumpTimer = 0.0f;
                }
            }
            if ((Input.GetKeyUp(KeyCode.Space) && pulouEspaco == true) ||
(encoderBoolDwP && pulouEncoder == true))
            {
                isHoldingJump = false;
                totalJump += 1;
            }
            OnLow();
            encoderAnt = UDP_1.ValueEncoder;
        }
    }

    private void FixedUpdate()
    {
        OnDead();
        //Time.fixedDeltaTime;
        if (isDead)
        {
            //velocity.x = 0;
            return;
        }
    }

```

```

Vector2 pos = transform.position;

if (pos.y < -20)
{
    velocity.x = 0;
    isDead = true;
}

if(!isGrounded) {
    if (isHoldingJump)
    {
        holdJumpTimer += Time.fixedDeltaTime;
        if(holdJumpTimer >= maxHoldJumpTime)
        {
            isHoldingJump = false;
        }
    }
    pos.y += velocity.y * Time.fixedDeltaTime;
    if(!isHoldingJump)
    {
        velocity.y += gravity * Time.fixedDeltaTime;
    }

    Vector2 rayOrigin = new Vector2(pos.x + 0.7f, pos.y - OffSetY()); //pro
    player poder ter um pouco de espaço pra erro tbm 0.2f * transform.localScale.x
    Vector2 rayDirection = Vector2.up; //pq a velocidade é negativa
    float rayDistance = velocity.y * Time.fixedDeltaTime;
    RaycastHit2D hit2D = Physics2D.Raycast(rayOrigin, rayDirection,
    rayDistance, groundLayerMask);
    if (hit2D.collider != null ) {
        Ground ground = hit2D.collider.GetComponent<Ground>(); // pra
        impedir que reconheça outras coisas que não o chão
        if (ground != null)
        {
            if (pos.y >= ground.groundHeight)
            {
                groundHeight = ground.groundHeight + OffSetY();
                if (Agachou == true) { agachouAux = false; } //para saber se
                estava no ar e acabou de cair com o botão de agachado pressionado
                else if (Agachou == false) { agachouAux = true; }
                //Debug.Log(OffSetY());
                pos.y = groundHeight;// + transform.localScale.y/2;
                //condição de ficar no chão
                velocity.y = 0;
                isGrounded = true;
            }
            //Debug.Log("No chão");
        }
    }
    Debug.DrawRay(rayOrigin, rayDirection * rayDistance, Color.red);

    Vector2 wallOrigin = new Vector2(pos.x, pos.y - OffSetY());
    Vector2 wallDir = Vector2.right;
    RaycastHit2D wallHit = Physics2D.Raycast(wallOrigin, wallDir, velocity.x
    * Time.fixedDeltaTime, groundLayerMask);
    if(wallHit.collider != null)
    {
        Ground ground = wallHit.collider.GetComponent<Ground>();
        if (ground != null)
        {
            if(pos.y < ground.groundHeight) // pra ter ctz que o lugar que a
            gente bateu não é a parte de cima da plat

```

```

        {
            velocity.x = 0; // caiu por bater no lado da
parede -> BUG
            velocity.y = 0;
            acceleration = 0f; // Dá pra resolver assim,
colocando os dois (ou senão impedindo o código de pensar que isGrounded
            isDead = true;
        }
    }
}

; }

distance += velocity.x * Time.fixedDeltaTime;
distancia = distance;
EventoAc();

if (isGrounded)
{
    if (agachouAux == false) //Offset 0.66
    {
        if (Agachou == true) { pos.y = groundHeight; } //agachou? mantem
        else if (Agachou == false) { pos.y = groundHeight + offSetDiff; }
//levantou? sobe o player
    }
    if (agachouAux == true) //Offset 1.3
    {
        if (Agachou == false) { pos.y = groundHeight; }
// tá levantado ainda? mantem
        else if (Agachou == true) { pos.y = groundHeight - offSetDiff; }
// agachou? abaixa o player
    }
    float velocityRatio;
    if (numEvento >= 5) // Se tiver voltando do evento, pra próxima ser
gerada adequadamente
    {
        velocityRatio = velocity2.x / maxVelocity;
    }
    else { velocityRatio = velocity.x / maxVelocity; }

    maxHoldJumpTime = maxMaxHoldJumpTime * velocityRatio; //pra poder
aumentar capacidade de pulo com o tempo
    //Debug.Log(maxHoldJumpTime);
    velocity.x += acceleration * Time.fixedDeltaTime;
    velocidade = velocity.x;
    //Debug.Log(velocity.x);
    if (velocity.x >= maxVelocity) {
        velocity.x = maxVelocity;
    }

    Vector2 rayOrigin = new Vector2(pos.x - 0.7f, pos.y - OffSetY()); //pro
player poder ter um pouco de espaço pra errar na hora de pular 0.2f *
transform.localScale.x
    Vector2 rayDirection = Vector2.up; //pq a velocidade é negativa
    float rayDistance = velocity.y * Time.fixedDeltaTime;
    RaycastHit2D hit2D = Physics2D.Raycast(rayOrigin, rayDirection,
rayDistance);
    if (hit2D.collider == null)
    {
        isGrounded = false; //pulou
    }
    Debug.DrawRay(rayOrigin, rayDirection * rayDistance, Color.yellow);
}

```

```

        // esse aqui copieei do cima pq ele não atualizava a posição Agachado qd
tava grounded
        //if (hit2D.collider != null)
        //{
        //    Ground ground = hit2D.collider.GetComponent<Ground>(); // pra
impedir que reconheça outras coisas que não o chão
        //    if (ground != null)
        //    {
        //        groundHeight = ground.groundHeight + OffSetY();
        //        Debug.Log(OffSetY());
        //        pos.y = groundHeight;// + transform.localScale.y/2; //condição
de ficar no chão
        //    }
        //}
        // Até aqui
    }

    //Obstáculos

    Vector2 obstOrigin = new Vector2(pos.x, pos.y);
    Vector2 obstDirection = Vector2.right;
    float obstDistance = velocity.x * Time.fixedDeltaTime;
    RaycastHit2D obstHitx = Physics2D.Raycast(obstOrigin, obstDirection,
obstDistance, obstacleLayerMask);
    if(obstHitx.collider != null)
    {
        Obstacle obstacle = obstHitx.collider.GetComponent<Obstacle>(); //tenta
achar um objeto do tipo Obstáculo
        if (obstacle != null) //Se
tiver encontrado roda a função abaixo
        {
            hitObstacle(obstacle);
        }

        Coin moeda = obstHitx.collider.GetComponent<Coin>();
        if (moeda != null)
        {
            hitMoeda(moeda);
        }
    }

    Debug.DrawRay(obstOrigin, obstDirection * obstDistance, Color.green);

    Vector2 obstDirectionY = Vector2.up;
    float obstDistanceY = velocity.y * Time.fixedDeltaTime;
    RaycastHit2D obstHity = Physics2D.Raycast(obstOrigin, Vector2.up, velocity.y
* Time.fixedDeltaTime, obstacleLayerMask);
    if (obstHity.collider != null)
    {
        Obstacle obstacle = obstHity.collider.GetComponent<Obstacle>();
        if (obstacle != null)
        {
            hitObstacle(obstacle);
        }
        Coin moeda = obstHity.collider.GetComponent<Coin>();
        if (moeda != null)
        {
            hitMoeda(moeda);
        }
    }

    Debug.DrawRay(obstOrigin, obstDirectionY * obstDistanceY, Color.blue);

```

```

//Moedas

    transform.position = pos;    // pega a posição no início do FixedUpdate e
atualiza no final                // é bom usar essa função para usar
Time.fixedDeltaTime              //

    OnRun(velocity.x);

}

public void EventoAc()
{
    //Debug.Log(numEvento);
    //caminho esperado -> a,e (bis) -> entrou no evento -> b,d,c (c só uma) ->
a,e
    if(numEvento == 0) //Sempre vai gravar a velocidade qd o evento não estiver
acontecendo, ela só é modificada no ground
    {
        velocity2 = velocity;
        //Debug.Log("a");
    }
    if (contadorMoeda % 5 == 0 && contadorMoeda != 0 && numEvento <= 5) //se
resto da divisão do contador por 5 é zero
    {
        evento = true;
        acceleration = 3f;
        //Debug.Log("b");
    }
    else if (contadorMoeda % 5 == 1 && contadorMoeda != 1 && numEvento >= 5)
//numEvento tbm pq acho que tava gerando moeda dps de entrar no evento aí ficava
sempre zerando de volta e dava bug
    { //isso é pra sempre que onseguir pegar mais uma moeda o numEvento ir pra
zero (mas na real ele sempre acontece só por 5 plataformas)
        numEvento = 0;
        totalEvento += 1;
        //Debug.Log("c");
    }
    else if(numEvento > 5)
    { // pra se não conseguir pegar moeda ao sair do evento isso não ser um
problema e volta pra aceleração original
        acceleration = 0.4f;
        //Debug.Log("d");
    }
    else
    {
        evento = false;
        acceleration = 0.4f;
        //Debug.Log("e");
    }
}

private void hitMoeda(Coin moeda) //realiza ações que não podem ser acessadas
externamente
{
    Destroy(moeda.gameObject);
    contadorMoeda += 1;
    contadorMoeda2 = contadorMoeda;
}

```

```

void hitObstacle(Obstacle obstacle)
{
    Destroy(obstacle.gameObject);
    velocity.x *= 0.7f;
    didHitObstacle = true;
    totalHitObstacle += 1;
}

public float OffSetY()
{
    if(Agachou == false)
    { return offSetYUp;}
    else
    { return offSetYDown;}
}

public void OnRun(float vel)
{
    if(vel >= 12f)
    {
        Correu = true;
    } else if(vel < 12f)
    {
        Correu = false;
    }
}

public void OnLow()
{
    if (Input.GetKeyDown(KeyCode.LeftShift) || UDP_1.ValueEncoder < limitC)
    {
        if(Input.GetKeyDown(KeyCode.LeftShift) && UDP_1.ValueEncoder >= limitC)
        { agachouShift = true; agachouEncoder = false; }
        if (!Input.GetKeyDown(KeyCode.LeftShift) && UDP_1.ValueEncoder < limitC)
        { agachouShift = false; agachouEncoder = true; }
        if (Input.GetKeyDown(KeyCode.LeftShift) && UDP_1.ValueEncoder < limitC)
        { agachouShift = true; agachouEncoder = true; }
        //Debug.Log(Agachou);
        Agachou = true;
        //Debug.Log(Agachou);
        //Debug.Log("Entrou Agachou");
    }
    if ((Input.GetKeyUp(KeyCode.LeftShift)&& agachouShift==true) || (Agachou ==
true && UDP_1.ValueEncoder> limitC && agachouEncoder == true))
    {
        //Debug.Log(UDP_1.ValueEncoder);
        //Debug.Log(Input.GetKeyUp(KeyCode.LeftShift));
        //Debug.Log(Agachou);
        Agachou = false;
        totalAgach += 1;
        //Debug.Log(totalAgach);
        //Debug.Log(Agachou);
        //Debug.Log("Saiu Agachou");
    }
}

public void OnDead()
{
    if (isDead)
    {
        Morreu = true;
    }
}

```

```

        Moveu = false;
        Correu = false;
        //UDP_2.stopThread = true;
        UDP_1.Instance.Disable(); // Pode dar problema dps por ele ser chamado
sempre
        //_llive = true;
        //Debug.Log("stopThread Player");

    }else if (!isDead)
    {
        Morreu=false;
        Moveu = true;
        //if (_llive)
        //{
        //    UDP_1.Instance.Enable();
        //    _llive = false;
        //}
        //UDP_1.Instance.Enable(); // Pode dar problema dps por ele ser chamado
sempre
        //UDP_2.stopThread=false;
    }
}
}

```

Código 33: Controlador da interface de usuário para o Menu Principal do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class UIControllerMenu : MonoBehaviour
{
    public void jogar()
    {
        UnityEngine.SceneManagement.SceneManager.LoadScene("SampleScene");
        //UDP_2.stopThread = false;
        UDP_1.Instance.Enable();
    }
}
```

Código 34: Controlador da interface de usuário do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using TMPro;
using UnityEngine.SceneManagement;

public class UIController : MonoBehaviour
{
    Player player;
    private TMPro.TextMeshProUGUI distanceText;
    private TMPro.TextMeshProUGUI coinsText;
    //TMP_Text distanceText;

    GameObject results;
    TextMeshProUGUI distancia;
    TextMeshProUGUI moedas;

    private void Awake()
    {
        player = GameObject.Find("Player").GetComponent<Player>();
        distanceText =
GameObject.Find("DistanceText").GetComponent<TextMeshProUGUI>();
        coinsText = GameObject.Find("CoinsText").GetComponent<TextMeshProUGUI>();

        distancia = GameObject.Find("Distancia").GetComponent <TextMeshProUGUI>();
        moedas = GameObject.Find("Moedas").GetComponent<TextMeshProUGUI>();
        results = GameObject.Find("Results");
        results.SetActive(false);
    }

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void Update()
    {
        int distance = Mathf.FloorToInt(player.distance);
        int coins = player.contadorMoeda;
        distanceText.text = distance + " m";
        coinsText.text = coins + " moedas";

        if (player.isDead)
        {
            results.SetActive(true);
            distancia.text = distance + " m";
            moedas.text = coins + "";
        }
    }

    public void Sair()
    {
        SceneManager.LoadScene("Menu");
    }
}
```

```
        UDP_1.Instance.Disable();  
        //UDP_2.stopThread = true;  
    }  
    public void Denovo()  
    {  
        SceneManager.LoadScene("SampleScene");  
        UDP_1.Instance.Enable();  
        //UDP_2.stopThread = false;  
    }  
}
```

Código 35: Controlador das animações do personagem

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PlayerAnimationController : MonoBehaviour
{
    public Animator playerAnimator;

    public void PlayAnimation(string animationName)
    {
        playerAnimator.Play(animationName);
    }
}
```

Código 36: Efeito Paralaxe do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Parallax : MonoBehaviour
{
    [SerializeField]
    public float depth = 1;

    Player player;

    private void Awake()
    {
        player = GameObject.Find("Player").GetComponent<Player>();
    }

    // Start is called before the first frame update
    void Start()
    {
    }

    // Update is called once per frame
    void FixedUpdate()
    {
        float realVelocity = player.velocity.x / depth;
        Vector2 pos = transform.position;

        pos.x -= realVelocity * Time.fixedDeltaTime;

        if (pos.x <= -30)
        {
            pos.x = 85;
        }

        transform.position = pos;
    }
}
```

Código 37: Funcionamento dos obstáculos do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Obstacle : MonoBehaviour
{
    Player player;

    void Start()
    {
        player = GameObject.Find("Player").GetComponent<Player>();
    }

    // Update is called once per frame
    void Update()
    {
    }

    private void FixedUpdate()
    {
        Vector2 pos = transform.position;

        pos.x -= player.velocity.x * Time.fixedDeltaTime;
        if(pos.x < -100)
        {
            Destroy(gameObject);
        }
        transform.position = pos;
    }
}
```

Código 38: Funcionamentos dos prédios do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using Unity.VisualScripting;
using UnityEngine;

public class Ground : MonoBehaviour
{
    Player player;

    public float groundHeight;
    public float groundRight;
    public float screenRight;

    public static float posXGround;
    public static float posYGround;

    BoxCollider2D colliderG;
    Camera camera1;

    bool didGenerateGround = false;

    private float Ytol = 0.6f;
    private float Xtol = 0.6f;

    public Obstacle boxTemplate;
    public Coin coin;

    private void Awake()
    {
        player = GameObject.Find("Player").GetComponent<Player>();

        colliderG = GetComponent<BoxCollider2D>();
        groundHeight = transform.position.y + (colliderG.size.y / 2);
        camera1 = GameObject.Find("MainCamera").GetComponent<Camera>();
        screenRight = camera1.transform.position.x * 2;
    }

    // Start is called before the first frame update
    void Start()
    {
        //numPunish = 0f;
    }

    // Update is called once per frame
    void Update()
    {
    }

    private void FixedUpdate()
    {
        Vector2 pos = transform.position;
        posXGround = pos.x;
        posYGround = pos.y;
        pos.x -= player.velocity.x * Time.fixedDeltaTime; // adicionado pra mover o
        chão dps de ter tirado o parallax nele pra teste

        groundRight = transform.position.x + (colliderG.size.x / 2);
    }
}
```

```

    if (groundRight < 0)
    {
        Destroy(gameObject);
        return;
    }

    if (!didGenerateGround)
    {
        if (groundRight < screenRight)
        {
            didGenerateGround = true;
            generateGround();
        }
    }
    transform.position = pos;
}

void generateGround()
{
    GameObject go = Instantiate(gameObject);
    BoxCollider2D goColliderG = go.GetComponent<BoxCollider2D>();
    Vector2 pos;
    float h1 = player.jumpVelocity * player.maxHoldJumpTime; // altura máxima
    que o player pode chegar segurando o botão
    //Debug.Log(player.jumpVelocity);
    //Debug.Log(player.maxHoldJumpTime);
    float t = player.jumpVelocity / -player.gravity; //  $V = V_0 + at \rightarrow t = (V - V_0)/a \rightarrow V = V_f = 0$ 
    float h2 = player.jumpVelocity * t + (0.5f * (player.gravity * (t * t))); //
    a altura que falta pro player subir após ele perder propulsão
    float maxJumpHeight = h1 + h2;
    //Debug.Log(maxJumpHeight);
    float maxY = maxJumpHeight * Ytol; //pro player poder errar um pouco a
    altura de pulo
    maxY += groundHeight;
    //Debug.Log(maxY);
    float minY = 1;
    float actualY = Random.Range(minY, maxY);
    //Debug.Log(actualY);
    pos.y = actualY - goColliderG.size.y / 2;
    //Debug.Log(pos.y);
    if (pos.y > 2.7f)
    {
        pos.y = 2.7f;
    }

    float t1 = t + player.maxHoldJumpTime;
    //Debug.Log(player.maxHoldJumpTime);
    float t2 = Mathf.Sqrt((2.0f * (maxY - actualY)) / -player.gravity);
    float totalTime = t1 + t2;
    //Debug.Log(totalTime);
    float maxX;
    if (player.numEvento >= 5) // Se tá voltando do evento tem que considerar a
    velocidade que ele vai ter no fim da plataforma pq senão não consegue pular
    {
        //Debug.Log(player.velocity2.x);
        maxX = totalTime * player.velocity2.x;
    }
    else { maxX = totalTime * player.velocity.x; }
    //Debug.Log(player.velocity.x);
    maxX *= Xtol;
    //Debug.Log(maxX);
}

```

```

maxX += groundRight;
float minX = screenRight + 0.5f; // Ideia de melhoria: fazer previsão da
velocidade do player quando ele chegar mais perto da borda
float actualX = Random.Range(minX, maxX);
//Debug.Log(actualX - screenRight);

pos.x = actualX + goColliderG.size.x/2;

go.transform.position = pos;

// Obstáculos
Ground goGround = go.GetComponent<Ground>();
goGround.groundHeight = go.transform.position.y + (goColliderG.size.y / 2);
float halfWidth = goColliderG.size.x / 2 - 1;
int obstacleNum;
if (player.evento == true)
{
    obstacleNum = 0;
}
else
{
    obstacleNum = Random.Range(0, 3);
}
for (int i =0; i<obstacleNum; i++)
{
    GameObject box = Instantiate(boxTemplate.gameObject);
    float boxYsize = box.transform.localScale.y;
    float boxXsize = (16.29f)/box.transform.localScale.x *
box.transform.localScale.x;
    Debug.Log(boxXsize);
    float y = goGround.groundHeight + 0.5f * boxYsize;
    float left = go.transform.position.x - halfWidth + boxXsize/2 + 5.5f;
    float right = go.transform.position.x + halfWidth - boxXsize / 2 - 6f;
    // tvz ajustar pq fica difícil pegar a prox. moeda se obst. for mt tarde
    float x = Random.Range(left, right);
    Vector2 boxPos = new Vector2(x, y);
    box.transform.position = boxPos;

}
//Debug.Log(player.numPunish);
//Debug.Log(player.didHitObstacle);
// Moedas
if (player.didHitObstacle == true) //Se bateu no obstáculo vai começar a
contar pra ver se já pode voltar ao normal
{
    if (player.numPunish >= 5)
    {
        player.didHitObstacle = false;
        player.numPunish = 0;
        GenMoeda();
        //Debug.Log(player.didHitObstacle);
    }
    else if(player.numPunish < 5)
    {
        player.numPunish++;
        //Debug.Log(player.numPunish);
    }
}
else if (player.didHitObstacle == false && player.evento == true)
{
    if (player.numEvento >= 5)
    {
        player.evento = false;
    }
}

```

```

        player.numEvento++; // pra considerar a possibilidade de não
consequir pegar uma moeda ao sair do evento
        player.velocity = player.velocity2; //velocidade voltando ao que
tava antes

        Debug.Log(player.velocity.x);
        GenMoeda();
    }
    else if (player.numEvento < 5)
    {
        player.numEvento++;
        Debug.Log("bug");
    }
}
else if (player.didHitObstacle == false && player.evento == false)
{
    GenMoeda();
}
void GenMoeda()
{
    GameObject moeda = Instantiate(coin.gameObject);
    float coinYsize = moeda.transform.localScale.y;
    float coinXsize = moeda.transform.localScale.x;
    float yCoin = goGround.groundHeight + 0.5f * coinYsize;
    float xCoin = go.transform.position.x - halfWidth + coinXsize + 2f;
    Vector2 coinPos = new Vector2(xCoin, yCoin);
    moeda.transform.position = coinPos;
}
}
}

```

Código 39: Funcionamento das moedas do jogo City Walk

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Coin : MonoBehaviour
{
    Player player;

    // Start is called before the first frame update
    void Start()
    {
        player = GameObject.Find("Player").GetComponent<Player>();
    }

    // Update is called once per frame
    void Update()
    {
    }

    private void FixedUpdate()
    {
        Vector2 pos = transform.position;

        pos.x -= player.velocity.x * Time.fixedDeltaTime;
        if (pos.x < -100)
        {
            Destroy(gameObject);
        }
        transform.position = pos;
    }
}
```

ANEXO 1 – Datasheet do encoder rotativo E40S6 - 1000

E40 Series

Diameter ø40mm Shaft type/Hollow type/Built-in type Incremental Rotary Encoder

■ Features

- 12-24VDC power supply of line driver output(Line-up)
- Easy installation at narrow space
- Small moment of inertia
- Power supply : 5VDC, 12-24VDC ±5%
- Various output types

⚠ Please read "Caution for your safety" in operation manual before using.



■ Ordering information

E40	H	8	5000	3	N	24	
-----	---	---	------	---	---	----	--

Series	Shaft type	Hollow type	Pulse/1Revolution	Output phase	Output	Power supply	Cable
S: Shaft type H: Hollow type HB: Hollow built-in type	External 6: ø6mm 8: ø8mm	Inner 6: ø6mm 8: ø8mm 10: ø10mm 12: ø12mm	Series	2: A, B 3: A, B, Z 4: A, $\bar{A}$, B, $\bar{B}$ 6: A, $\bar{A}$, B, $\bar{B}$, Z, $\bar{Z}$	T: Totem pole output N: NPN open collector output V: Voltage output L: Line driver output	5 : 5VDC ±5% 24: 12-24VDC ±5%	No mark: Cable type C: Connector cable type(※)

※Standard: E40S6-PULSE-3-N-24, E40H8-PULSE-3-N-24 ※Standard: A, B, Z
E40HB8-PULSE-3-N-24

※Cable length : 250mm

■ Specifications

Item	Diameter ø40mm shaft/hollow shaft/hollow built-in type of incremental rotary encoder	
Resolution(P/R) ^{※1}	•1, •2, •5, 10, •12, 15, 20, 23, 25, 30, 35, 40, 45, 50, 60, 75, 100, 120, 150, 192, 200, 240, 250, 256, 300, 360, 400, 500, 512, 600, 800, 1000, 1024, 1200, 1500, 1800, 2000, 2048, 2500, 3000, 3600, 5000 (Not indicated resolution is customizable.)	
Electrical specification	Output phase	A, B, Z phase(Line driver A, $\bar{A}$, B, $\bar{B}$, Z, $\bar{Z}$ phase)
	Phase difference of output	Phase difference between A and B : $\frac{T}{4} \pm \frac{T}{8}$ (T=1cycle of A phase)
	Control output	Totem pole output
		• Low - Load current:Max. 30mA, Residual voltage : Max. 0.4VDC • High - Load current:Max. 10mA, Output voltage(Power voltage 5VDC): Min. (Power voltage-2.0)VDC, Output voltage(Power voltage 12-24VDC):Min. (Power voltage-3.0)VDC
		NPN open collector output
		Load current : Max. 30mA, Residual voltage : Max. 0.4VDC
	Response time (Rise/Fall)	Voltage output
		Load current : Max. 10mA, Residual voltage : Max. 0.4VDC
		Line driver output
		• Low - Load current : Max. 20mA, Residual voltage : Max. 0.5VDC • High - Load current : -20mA, Output voltage(Power voltage 5VDC): Min. 2.5VDC, Output voltage(Power voltage 12-24VDC): Min. (Power voltage-3.0)VDC
	Max. Response frequency	Totem pole output
		NPN open collector output
		Max. 1μs
		Voltage output
Mechanical specification	Connection	Line driver output
		Max. 0.5μs
		Max. 300kHz
		Power supply
	Current consumption	• 5VDC ±5%(Ripple P-P : Max. 5%) • 12-24VDC ±5%(Ripple P-P : Max. 5%)
		Max. 80mA(disconnection of the load), Line driver output : Max. 50mA(disconnection of the load)
		Insulation resistance
		Min. 100MΩ(at 500VDC megger between all terminals and case)
	Dielectric strength	750VAC 50/60Hz for 1 minute(Between all terminals and case)
		Connection
		Cable type, 250mm connector cable type
		Starting torque
	Moment of inertia	Shaft type : Max. 40gf·cm(0.004N·m), Hole type : Max. 50gf·cm(0.005N·m)
		Max. 40g·cm ² (4×10 ⁻⁶ kg·m ²)
		Shaft loading
		Radial : Max. 2kgf, Thrust : Max. 1kgf
Environmental specification	Max. allowable revolution ^{※2}	5000rpm
		Vibration
		1.5mm amplitude or 300m/s ² at frequency of 10 to 55Hz(for 1 min.) in each of X, Y, Z directions for 2 hours
		Shock
	Environment	Approx. Max. 50G
		Ambient temperature
		-10 to 70°C, storage : -25 to 85°C
		Ambient humidity
	Protection	35 to 85%RH, storage : 35 to 90%RH
		IP50(IEC standard) ※Option type is available for IP64 (IEC standard).
		Cable
		ø5, 5-wire, Length : 2m, Shield cable(Line driver output : ø5, 8-wire) (AWG24, Core diameter: 0.08, Number of cores: 40, Insulator out diameter: ø1)
Accessory	Approval	• Shaft type : ø6mm coupling standard, ø8mm coupling(Sold separately) • Hole type : Bracket
		CE (Except line driver output)
		Unit weight
	Unit weight	Approx. 160g

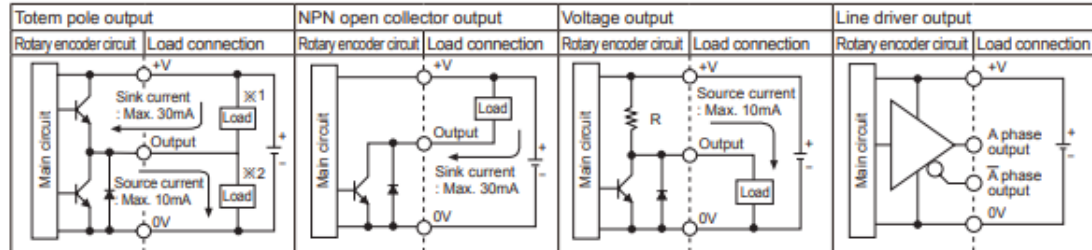
※1: "P" pulse is only for A, B phase(Line Driver output is for A, $\bar{A}$, B, $\bar{B}$ phase)

※2: Make sure that, Max response revolution should be lower than or equal to max. allowable revolution when selecting the resolution.

(Max. response revolution(rpm)) = $\frac{\text{Max. response frequency}}{\text{Resolution}} \times 60 \text{ sec}$ ※Environment resistance is rated at no freezing or condensation.

Incremental ø40mm Shaft/Hollow Shaft/Built-in type

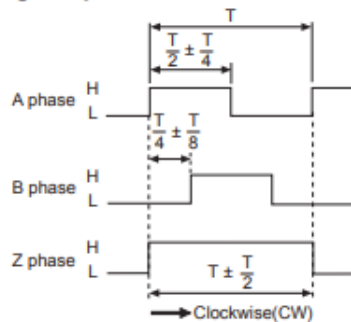
Control output diagram



- Totem pole output type can be used for NPN open collector output type(※1) or Voltage output type(※2).
- All output circuits of A, B, Z phase are the same. (Line driver output is A, $\bar{A}$, B, $\bar{B}$, Z, $\bar{Z}$)

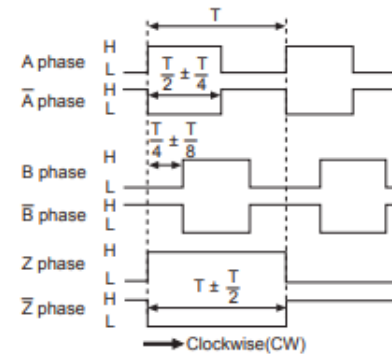
Output waveform

- Totem pole output / NPN open collector output / Voltage output



- ※Z phase output is option.
- ※CW : Right turn as from the shaft.

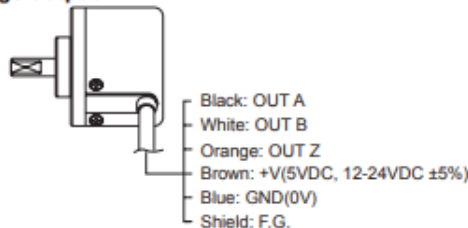
- Line driver output



Connections

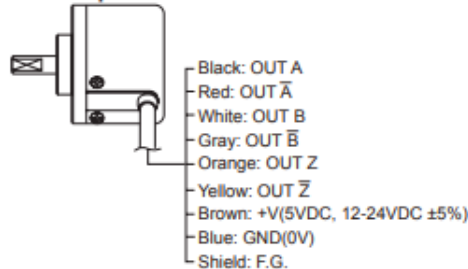
Cable type

- Totem pole output / NPN open collector output / Voltage output



- ※Unused wires must be insulated.
- ※The metal case and shield wire of encoder should be grounded(F.G.).

- Line driver output



Connector cable type

- Totem pole output / NPN open collector output / Voltage output
- Line driver output



Totem pole output/ NPN open collector output/ Voltage output			Line driver output		
Pin No	Function	Cable color	Pin No	Function	Cable color
①	OUT A	Black	①	OUT A	Black
②	OUT B	White	②	OUT $\bar{A}$	Red
③	OUT Z	Orange	③	+V	Brown
④	+V	Brown	④	GND	Blue
⑤	GND	Blue	⑤	OUT B	White
⑥	F.G.	Shield	⑥	OUT $\bar{B}$	Gray
			⑦	OUT Z	Orange
			⑧	OUT $\bar{Z}$	Yellow
			⑨	F.G.	Shield

- ※F.G.(Field Ground): It should be grounded separately.

(A)	Photo electric sensor
(B)	Fiber optic sensor
(C)	Door/Area sensor
(D)	Proximity sensor
(E)	Pressure sensor
(F)	Rotary encoder
(G)	Connector/Socket
(H)	Temp. controller
(I)	SRP/Power controller
(J)	Counter
(K)	Timer
(L)	Panel meter
(M)	Tachol/Speed/Pulse meter
(N)	Display unit
(O)	Sensor controller
(P)	Switching mode power supply
(Q)	Stepper motor& Driver&Controller
(R)	Graphic/Logic panel
(S)	Field network device
(T)	Software
(U)	Other

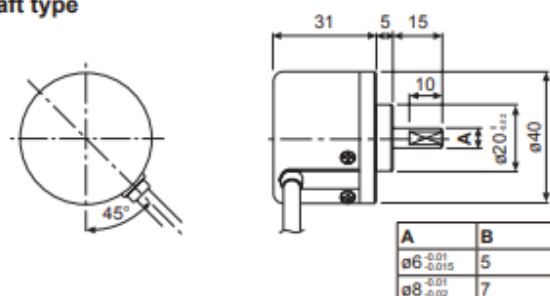
E40 Series

■ Dimensions

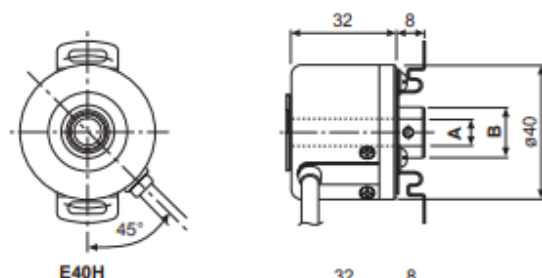
(unit: mm)

◎ Cable type

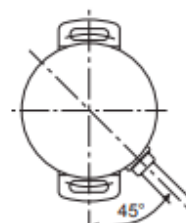
● Shaft type



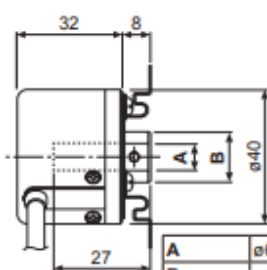
● Hollow shaft / Hollow shaft built-in type



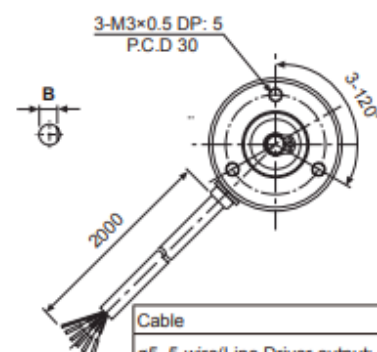
E40H



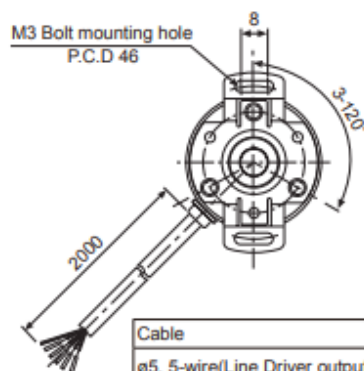
E40HB



A	ø6	ø8	ø10	ø12
B	ø15	ø17		
Tolerance	$+0.015$ S			

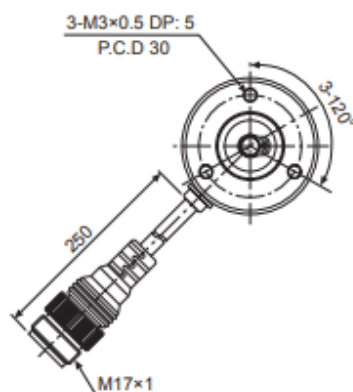


Cable
ø5, 5-wire(Line Driver output: 8-wire),
Length:2000, Shield cable



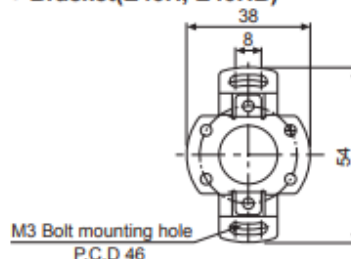
Cable
ø5, 5-wire(Line Driver output: 8-wire),
Length: 2000, Shield cable

◎ Connector cable type



※Connector cable is customizable and refer to the G-10 for specifications.

● Bracket(E40H, E40HB)

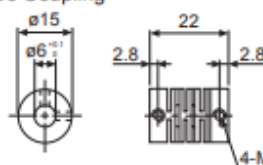


M3 Bolt mounting hole
P.C.D 46

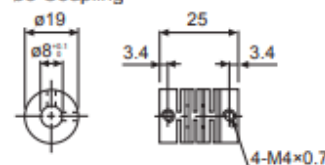


● Coupling(E40S)

● ø6 Coupling



● ø8 Coupling



- Parallel misalignment : Max. 0.25mm
- Angular misalignment: Max. 5°
- End-play: Max. 0.2mm

- ※For parallel misalignment, angular misalignment, end-play terms, refer to the F-78 page.
- ※For flexible coupling(ERB Series) information, refer to the F-71 page.