UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

PROGRAMA DE PÓS-GRADUAÇÃO EM COMPUTAÇÃO

MESTRADO ACADÊMICO EM COMPUTAÇÃO

LUIS ILDERLANDIO DA SILVA OLIVEIRA

GENERALIZAÇÃO DO CONTROLE DE MOVIMENTO EM TEMPO REAL COM DRL USANDO RECOMPENSAS CONDICIONAIS E RESTRIÇÕES DE SIMETRIA

QUIXADÁ

2024

LUIS ILDERLANDIO DA SILVA OLIVEIRA

GENERALIZAÇÃO DO CONTROLE DE MOVIMENTO EM TEMPO REAL COM DRL USANDO RECOMPENSAS CONDICIONAIS E RESTRIÇÕES DE SIMETRIA

Dissertação apresentada ao Curso de Mestrado Acadêmico em Computação do Programa de Pós-Graduação em Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Computação. Área de Concentração: Sistemas de Computação

Orientador: Prof. Dr. Rubens Fernandes Nunes

QUIXADÁ

2024

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

O48g Oliveira, Luis Ilderlandio da Silva.
Generalização do controle de movimento em tempo real com drl usando recompensas condicionais e restrições de simetria / Luis Ilderlandio da Silva Oliveira. – 2024.
107 f. : il.

Dissertação (mestrado) – Universidade Federal do Ceará, Campus de Quixadá, Programa de Pós-Graduação em Computação, Quixadá, 2024.
Orientação: Prof. Dr. Rubens Fernandes Nunes.

1. Animação de personagens baseada em física. 2. Controle de movimento. 3. Aprendizado por reforço profundo. 4. Controle em tempo real. I. Título.

CDD 005

LUIS ILDERLANDIO DA SILVA OLIVEIRA

GENERALIZAÇÃO DO CONTROLE DE MOVIMENTO EM TEMPO REAL COM DRL USANDO RECOMPENSAS CONDICIONAIS E RESTRIÇÕES DE SIMETRIA

Dissertação apresentada ao Curso de Mestrado Acadêmico em Computação do Programa de Pós-Graduação em Computação do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do título de mestre em Computação. Área de Concentração: Sistemas de Computação

Aprovada em: 26 de Setembro de 2024

BANCA EXAMINADORA

Prof. Dr. Rubens Fernandes Nunes (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Regis Pires Magalhães
Universidade Federal do Ceará (UFC)

Prof. Dr. Joaquim Bento Cavalcante Neto
Universidade Federal do Ceará (UFC)

Prof. Dr. Creto Augusto Vidal
Universidade Federal do Ceará (UFC)

AGRADECIMENTOS

Ao meu orientador, Dr. Rubens Nunes, por sua orientação cuidadosa, incentivos genuínos, paciência e dedicação. Agradeço por compartilhar seu vasto conhecimento comigo, me guiando com maestria ao longo de toda a jornada de pesquisa, compreendendo minhas limitações, especialmente em relação ao tempo. Você fez mais do que seu papel como orientador, sendo um verdadeiro pilar de apoio, sempre acreditando no meu trabalho, mesmo quando eu duvidava de mim mesmo. Direcionou nossas ideias com habilidade, permitindo que finalizássemos este trabalho com sucesso. Sou eternamente grato por tudo que fez por mim, e tenho a certeza de que nunca conseguirei retribuir tudo que você fez. Você não foi apenas um mentor, mas sempre será um mestre e amigo.

A minha esposa, Sâmya Maria, pelo amor, parceria, compreensão e incentivo contínuo. Agradeço por estar ao meu lado durante todos os desafios, especialmente quando nossa filha nasceu no meio do mestrado. Sua paciência e apoio foram fundamentais, compreendendo que eu nem sempre podia estar presente devido às exigências dos estudos. Você foi minha força, me incentivando e mantendo nossa família unida. Sou eternamente grato por sua dedicação e por acreditar em mim. Sem você, esta jornada teria sido muito mais difícil, te amo.

À minha filha, Laura Martins de Oliveira, nascida durante o período de pesquisa, você é a razão pela qual nossas vidas fazem sentido. Sua chegada foi um incentivo a mais para eu não desistir do mestrado diante de tantas dificuldades. Todo este esforço é para lhe proporcionar oportunidades que seus pais não tiveram. Você trouxe ainda mais significado e propósito a esta jornada. Que este trabalho seja um passo a mais para construir um futuro brilhante para você.

Quero expressar minha imensa gratidão à minha sogra, Vanusa Landin, por cuidar tão bem de nossa filha Laura. Sua paciência, carinho e dedicação fizeram toda a diferença enquanto eu me dedicava aos estudos. Obrigado por estar sempre presente, cuidando da Laura com tanto amor e por ajudar também a mim e à Sâmya nos momentos mais desafiadores. Seu apoio foi essencial para manter nossa família em equilíbrio, e saber que Laura estava em boas mãos me trouxe tranquilidade. Agradeço de coração por tudo que faz por nós, com tanto cuidado e amor.

Agradeço profundamente à minha tia Antônia Ferreira de Oliveira (Tia Toinha) por ter nos incentivado, a mim e à minha mãe, a sair da roça e prestar o exame vestibular após o término do ensino médio. Sua iniciativa foi a pedra fundamental que deu início a toda essa jornada e transformou a perspectiva da minha vida. Sem o seu estímulo e visão, talvez eu nunca

tivesse dado os primeiros passos rumo à educação superior. Sou eternamente grato por seu papel crucial nesta mudança.

Agradeço profundamente à minha mãe, Felícia da Silva Oliveira, por sempre acreditar em mim e por todo o apoio financeiro durante o primeiro ano da faculdade. Sua confiança e sacrifício foram essenciais para que eu pudesse seguir em frente e alcançar meus objetivos. Sou eternamente grato por seu amor incondicional e por estar ao meu lado em cada passo dessa jornada.

Agradeço imensamente ao Torquato Gonçalves (Totó) por ter me permitido trabalhar nos finais de semana e feriados durante o primeiro ano da faculdade, possibilitando que eu custeasse as despesas de transporte e estadia nesse período. Sua generosidade não se limitou a isso; as ajudas financeiras que, vez ou outra, me oferecia foram essenciais para que eu pudesse enfrentar os desafios iniciais dessa jornada acadêmica. Sou profundamente grato pelo seu apoio, que foi crucial para que eu pudesse continuar meus estudos e seguir em frente.

Agradeço de coração à minha tia, Francisca Oliveira, e à minha irmã, Maria Sanderlúcia, por me acolherem durante as transições do sítio para a cidade onde eu estudava. Sua hospitalidade e apoio foram fundamentais nesse período de mudança, e sou muito grato por terem me oferecido um lar durante essa fase da minha vida.

Agradeço ao amigo, colega de trabalho e de mestrado, professor João Victor Rocha de Araújo, pela parceria durante as disciplinas do programa, pelos incentivos diários, pelos momentos de descontração que tornaram a jornada mais leve e por ter me encorajado a me inscrever no mestrado. Serei eternamente grato.

Aos professores e ao programa de Mestrado em Ciência da Computação da Universidade Federal do Ceará - Campus Quixadá, agradeço por me proporcionar não apenas o conhecimento racional, mas também a manifestação do caráter e afetividade da educação no processo de formação profissional. Sou grato por toda a dedicação e por terem me ensinado e me feito aprender.

Agradeço profundamente à Escola Estadual de Educação Profissional Amélia Figueiredo de Lavor, na pessoa de seus gestores João Paulo, Estelino, Dulce Maria e Eriglécia, por toda a compreensão e apoio durante o período em que cursei o mestrado. Sou grato pela flexibilidade na organização dos horários e pela liberação nos dias em que precisei me dedicar exclusivamente aos estudos. Esse suporte foi essencial para que eu pudesse conciliar minhas responsabilidades e alcançar meus objetivos acadêmicos.

Agradeço à Universidade Federal do Ceará - UFC, campus de Quixadá, por proporcionar as condições necessárias para que eu, um ex-agricultor, pudesse realizar o sonho de defender um mestrado. Este é um exemplo claro de como uma instituição de ensino pode transformar vidas, oferecendo as oportunidades que tantas pessoas precisam para mudar sua realidade. A educação é, sem dúvida, a ferramenta mais poderosa para abrir portas e construir um futuro melhor, e a UFC tem sido essencial nesse processo. Sou profundamente grato por fazer parte dessa comunidade acadêmica e por todo o apoio recebido ao longo desta jornada.

Aos meus pais, irmãos, sobrinhos e tias, que, mesmo morando em outras cidades e com minha ausência devido aos estudos para o mestrado, compreenderam que estou plantando sementes agora para colher frutos no futuro.

Para estas e todas as outras pessoas que contribuíram na minha vida, um muito obrigado.

"Ninguém que alcança o sucesso o faz sem a ajuda de outras pessoas. Os sábios e confiantes reconhecem esta ajuda com gratidão."

(Alfred North Whitehead)

RESUMO

Produzir movimentos naturais baseados em física para personagens articulados é um problema desafiador. Deep Reinforcement Learning (DRL) tem sido explorado como uma solução, mas seu uso isolado ainda resulta em movimentos não naturais e requer ajustes significativos. Fornecer novas ferramentas de controle apropriadas é necessário para guiar melhor o processo de aprendizagem e permitir que o animador tenha mais controle e confiabilidade sobre a animação resultante. Este trabalho aborda esse desafio propondo ferramentas para facilitar o processo de treinamento e fornecer novas formas de controle. As ferramentas propostas consistem na generalização do controle em tempo real, recompensas condicionais, restrições de simetria e uma interface de usuário adaptada às generalizações propostas. A generalização do controle em tempo real permite que qualquer parâmetro seja ajustado dinamicamente, aumentando a flexibilidade. As recompensas condicionais introduzem tolerâncias para que as condições possam ser satisfeitas simultaneamente, simplificando a concorrência entre recompensas e evitando a necessidade de ajustes precisos dos pesos. As restrições de simetria reduzem o espaço de ações, evitando movimentos descoordenados, e oferecem uma camada adicional de controle. Uma interface de usuário demonstra essas generalizações, permitindo que os parâmetros de treinamento e animação sejam especificados mais facilmente sem necessidade de programar. As ferramentas de controle propostas se mostraram úteis e promissoras, de acordo com resultados obtidos que foram compatíveis com as escolhas realizadas pela interface e que foram capazes de se adaptar ao ambiente.

Palavras-chave: animação de personagens baseada em física; controle de movimento; aprendizado por reforço profundo; controle em tempo real.

ABSTRACT

Producing natural, physics-based movements for articulated characters is a challenging problem. Deep Reinforcement Learning (DRL) has been explored as a solution, but its isolated use still results in unnatural movements and requires significant adjustments. Providing appropriate new control tools is necessary to better guide the learning process and allow the animator to have more control and reliability over the resulting animation. This work addresses this challenge by proposing tools to facilitate the training process and provide new forms of control. The proposed tools consist of the generalization of real-time control, conditional rewards, symmetry constraints, and a user interface adapted to the proposed generalizations. The generalization of real-time control allows any parameter to be adjusted dynamically, increasing flexibility. Conditional rewards introduce tolerances so that conditions can be met simultaneously, simplifying the competition between rewards and avoiding the need for precise weight adjustments. Symmetry constraints reduce the action space, preventing uncoordinated movements, and offer an additional layer of control. A user interface demonstrates these generalizations, allowing training and animation parameters to be specified more easily without the need for programming. The proposed control tools have proven to be useful and promising, according to results obtained that were compatible with the choices made by the interface and that were able to adapt to the environment.

Keywords: physics-Based character animation; motion control; deep reinforcement learning; real-time control.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

DRL Deep Reinforcement Learning

MDP Markov Decision Process

NPC Non-Player Character

PD Proporcional-Derivativo

PPO Proximal Policy Optimization

SAC Soft Actor-Critic

SVR Symposium on Virtual and Augmented Reality

SUMÁRIO

1 INTRODUÇÃO

Este capítulo está dividido em cinco seções. A primeira seção apresenta o problema abordado e a motivação para tratá-lo. A segunda seção define os objetivos gerais e específicos. Em seguida, na terceira seção, são apresentadas as questões de pesquisa que guiam a investigação. Na quarta seção, são apresentados o escopo e as contribuições propostas do trabalho. A última seção mostra como os capítulos restantes estão organizados.

1.1 Problema e motivação

A representação gráfica de personagens capazes de simular pessoas, animais ou seres fictícios é uma área bastante atrativa da computação gráfica, com uma ampla gama de aplicações. Sua aplicação abrange desde usos tradicionais, como na indústria de jogos, cinema e publicidade, até avanços mais recentes em realidades imersivas, como a realidade virtual e o metaverso (Agrawal; Panne, 2016; Nunes, 2012). Contudo, a animação de personagens articulados baseados em física continua sendo um grande desafio. Métodos eficazes para simular a diversidade e a complexidade dos comportamentos observados no mundo real ainda são escassos (Peng *et al.*, 2018).

A animação baseada em física de personagens articulados envolve a solução de equações diferenciais complexas, considerando o contato físico do personagem com o ambiente, mas requer principalmente a solução do problema de controle da atuação ativa do personagem, por meio de seus músculos. Além de realizar o movimento desejado de forma natural, é necessário tratar a manutenção de seu equilíbrio em conjunto. Trabalhar com métodos de solução adequados para esse problema de controle, que tem sido alvo de pesquisa há bastante tempo (Geijtenbeek; Pronost, 2012), é altamente desafiador mesmo para animadores experientes. Portanto, tornar esse processo de animação mais automático e geral é vital para aplicações em jogos e realidade virtual com personagens animados. Mas isso implica em lidar com o controle autônomo dos atuadores do personagem, que devem aplicar torques internos apropriados para realizar movimentos compatíveis com os objetivos da animação e com o ambiente utilizado.

O Aprendizado por Reforço Profundo, ou *Deep Reinforcement Learning* (DRL) surgiu como uma abordagem promissora para tratar o problema de controlar estruturas articuladas baseadas em física, especialmente para gerar movimentos de locomoção (Coros *et al.*, 2009). Uma rede neural é usada para lidar com a relação entre as entradas de controle, que consistem

nas informações sensoriais do personagem na medida em que interagem com o ambiente, e as informações de saída a serem utilizadas pelos atuadores. Essa rede neural treinada de maneira automática é essencial para tratar essa relação não linear e não intuitiva entre os parâmetros de controle e a animação resultante. No entanto, utilizar apenas o método DRL para animar personagens humanoides pode resultar em movimentos não naturais, como oscilações descoordenadas de alta frequência nos braços, que são injustificáveis como tentativas de alcançar o equilíbrio (Juliani *et al.*, 2020). Por concorrer com o próprio funcionamento básico do controlador, o ajuste da qualidade visual do movimento, mesmo usando DRL, ainda é um desafio (Kwiatkowski *et al.*, 2022).

Movimentos de referência podem ser utilizados para melhorar a qualidade visual e servir como uma ferramenta de controle para guiar o resultado esperado do DRL (Peng *et al.*, 2018), mas alcançar movimentos naturais de uma maneira que seja fácil de generalizar ainda é um problema em aberto. Como movimentos capturados são exemplos de animações obtidas para situações específicas, isso os torna difíceis de se adaptar a novas situações e, de certa forma, limita o resultado do treinamento ao reduzir o benefício do DRL no possível surgimento de novos movimentos mais espontâneos. Portanto, o uso da captura de movimentos (mocap) para novos personagens e novas situações frequentemente exige novas sessões de captura.

Sem movimentos de referência, a inclusão de termos de recompensa cuidadosamente escolhidos pode também melhorar os resultados e balancear melhor o equilíbrio entre a liberdade de espontaneidade dos movimentos experimentados durante o treinamento e um controle necessário para guiar melhor os movimentos premiados para os objetivos da animação (Yu *et al.*, 2018). Mas evitar os artefatos visuais e a não naturalidade ainda se mantém como um desafio para o animador, na hora de definir as recompensas. Além disso, especificar as recompensas no formato de uma função matemática pode ser considerado difícil por muitos usuários e pode desencorajá-los a utilizar o DRL.

1.2 Objetivos

1.2.1 Objetivo geral

Desenvolver e avaliar uma abordagem de animação de personagens fisicamente simulados utilizando aprendizado por reforço profundo, integrada com técnicas de controle em tempo real, recompensas condicionais, restrições de simetria e uma interface de usuário intuitiva,

visando melhorar a confiabilidade, estabilidade e flexibilidade dos movimentos gerados.

1.2.2 Objetivos específicos

1. Explorar a definição e implementação de recompensas condicionais: Propor um método para a construção das recompensas, facilitando a integração de múltiplos objetivos sem a necessidade de ajustes rigorosos dos pesos, utilizando tolerâncias e uma estrutura padronizada para a formulação matemática.
2. Desenvolver e testar a generalização do controle em tempo real: Implementar uma arquitetura que permita ao animador controlar diversas informações em tempo real, proporcionando maior flexibilidade e adaptabilidade ao controle dos movimentos do personagem.
3. Investigar o impacto das restrições de simetria na animação: Avaliar como a aplicação de restrições de simetria e anti-simetria pode melhorar a qualidade dos movimentos, prevenindo descoordenações e permitindo um controle adicional sobre os resultados da animação.
4. Projetar e implementar uma interface amigável do usuário: Desenvolver uma interface que simplifique o processo de configuração e controle das animações, permitindo que usuários sem conhecimentos técnicos avancem na definição de parâmetros de treinamento e controle, sem necessidade de programação ou edição de scripts.
5. Avaliar o uso de restrições artificiais para experimentos controlados: Explorar como as restrições artificiais podem ser integradas ao processo de treinamento, facilitando a realização de experimentos controlados e contribuindo para a obtenção de resultados mais consistentes e previsíveis.

1.3 Questões de pesquisa

QP1 Qual a dificuldade na definição de recompensas na animação de personagens fisicamente simulados utilizando DRL?

Essa questão visa investigar como recompensas podem ser implementadas de forma eficaz para equilibrar a espontaneidade dos movimentos com os objetivos específicos de animação.

QP2 Como a generalização do controle em tempo real pode influenciar a flexibilidade e adaptabilidade dos movimentos de personagens animados?

Essa questão visa analisar a eficácia do controle em tempo real, permitindo ao animador ajustar os movimentos durante a execução sem necessidade de um novo treinamento.

QP3 Quais são os benefícios e limitações do uso de restrições de simetria na geração de movimentos naturais para personagens fisicamente simulados?

Essa questão visa explorar como as restrições de simetria e anti-simetria podem prevenir movimentos descoordenados e fornecer um controle adicional sobre a animação.

QP4 Como a interface do usuário pode facilitar o processo de configuração e controle das animações sem a necessidade de programação ou edição de scripts?

Essa questão visa avaliar a eficácia da interface proposta em simplificar o processo de configuração do treinamento e controle das animações para usuários sem conhecimentos técnicos aprofundados.

QP5 De que forma a utilização de restrições artificiais pode contribuir para a realização de experimentos mais controlados e resultados mais consistentes?

Essa questão visa investigar o impacto das restrições artificiais na facilidade de realização de experimentos e na obtenção de resultados bem comportados.

1.4 Escopo e contribuições

Buscando enfatizar a potencial emergência de novos movimentos, característica do DRL, e devido às dificuldades de lidar com mocap, este trabalho está situado dentro do escopo em que DRL é utilizado sem captura de movimentos. Mantendo a capacidade de tomada de decisão do agente, deseja-se fornecer um maior grau de controle ao animador sobre a animação resultante e tratar aspectos relacionados a possíveis melhorias na estabilidade e realismo, por exemplo, amenizando algumas oscilações indesejadas.

Fornecer novas ferramentas de controle apropriadas é necessário para melhor e mais facilmente orientar o processo de aprendizagem e permitir que o animador tenha mais controle e confiabilidade sobre a animação resultante. Uma das ferramentas que este trabalho propõe atualizar envolve considerar informações desejadas, como a velocidade desejada do personagem (Sousa *et al.*, 2021), como uma entrada extra para a rede neural. Durante o treinamento, essas informações desejadas são exploradas com diferentes valores, por exemplo, utilizando um sorteio no início de cada episódio, para que a rede, uma vez treinada, esteja preparada para permitir o controle dessas informações. O animador então é capaz de alterar essa entrada da rede em tempo real para guiar a animação conforme desejado. Este trabalho propõe uma generalização dessa

forma de controle, permitindo a escolha de qualquer informação a ser controlada em tempo real através de uma interface para facilitar a execução de diferentes experimentos.

Foram realizados diversos testes para demonstrar a generalização do controle em tempo real e a influência da simetria e das restrições artificiais configuradas pela interface. Foram obtidos diversos movimentos, como diferentes estilos de corrida, movimento de agachar e levantar, controle de posição 2D e 3D em tempo real e interação com ambiente.

Os resultados de todos os experimentos realizados neste trabalho pode ser encontrado no link disponível no Capítulo 5.

Abordando os desafios mencionados, são propostas as seguintes contribuições:

1. **Generalização do Controle em Tempo Real**: A escolha de qualquer informação a ser controlada em tempo real é facilitada, proporcionando maior flexibilidade e adaptabilidade no controle dos movimentos do personagem.
2. **Recompensas Condicionais**: Uma nova abordagem para a definição de recompensas introduz as chamadas *recompensas condicionais*. Através do uso de tolerâncias, várias recompensas condicionais podem ser satisfeitas em conjunto, facilitando o problema de competição e evitando ajustes rigorosos de pesos. As recompensas condicionais propostas têm uma estrutura padronizada, permitindo a construção automática das fórmulas matemáticas necessárias para representá-las.
3. **Restrições de Simetria**: Restrições de simetria relativas aos membros do personagem são abordadas pela redução do espaço de ação correspondente às saídas da rede. Além de prevenir movimentos descoordenados ou indesejados dos membros, diferentes relações de simetria (ou anti-simetria) são permitidas, configurando uma forma adicional de controle sobre o resultado final da animação.
4. **Interface do Usuário**: Uma interface que demonstra na prática as generalizações propostas, tanto das informações controladas em tempo real quanto das recompensas e simetrias, é desenvolvida e apresentada. O usuário pode especificar o treinamento desejado sem a necessidade de programar ou editar *scripts*, apenas através da interface.
5. **Restrições Artificiais**: A interface também permite o uso prático de restrições artificiais como uma forma de facilitar a realização de certos experimentos, tornando-os mais fáceis de controlar e, consequentemente, obtendo resultados mais bem comportados.

1.5 Aprovação de artigo

Durante o desenvolvimento deste trabalho ocorreu a submissão e aprovação do artigo *Generalization of Real-Time Motion Control with DRL Using Conditional Rewards and Symmetry Constraints*, no *Symposium on Virtual and Augmented Reality (SVR)'24*. O referido artigo apresenta, de forma mais resumida, as principais contribuições deste trabalho. Nesta dissertação, as ideias apresentadas no artigo são exploradas com mais detalhes.

1.6 Estrutura da dissertação

O restante da dissertação está organizado nos seguintes capítulos. O Capítulo 2 apresenta o referencial teórico, abordando brevemente os principais conceitos que fundamentam a pesquisa. Em seguida, no Capítulo 3, são apresentados os principais trabalhos que serviram de base e inspiração para o desenvolvimento dessa pesquisa. O Capítulo 4 aborda a estrutura de controle utilizada neste trabalho para detalhar cada uma das contribuições apresentadas, bem como o método proposto para a obtenção das animações. Em seguida, no Capítulo 5, são explorados diversos experimentos, a fim de demonstrar como as ferramentas propostas podem ser utilizadas para obter uma diversidade de simulações. Finalmente, a conclusão e os trabalhos futuros são discutidos no Capítulo 6.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, serão apresentados os principais fundamentos teóricos que permitem uma melhor compreensão sobre a simulação baseada em física de personagens virtuais articulados de corpos rígidos. A primeira seção aborda a animação de personagens, detalhando os conceitos de personagens articulados. Em seguida, técnicas de animação relacionadas ao contexto do trabalho são apresentadas, incluindo captura de movimentos, simulação dinâmica e a combinação de métodos, destacando a diferença entre sistemas passivos e atuados. O problema de controle também é explorado, com ênfase no controlador proporcional-derivativo (PD). Além disso, é feita uma breve explanação sobre aprendizado de máquina, abordando classificação, redes neurais, aprendizado por reforço e aprendizado por reforço profundo. Finalmente, são discutidas a plataforma *Unity* e a ferramenta *ML-Agents*, que são fundamentais para o desenvolvimento e a implementação das técnicas discutidas.

2.1 Animação de personagens

A animação de personagens virtuais é uma subárea da computação gráfica que recebe grandes incentivos de diversos segmentos, principalmente devido às aplicações no campo cinematográfico, na indústria de jogos e na realidade virtual. O fascínio pela animação gráfica é justificado pelo fato de estar diretamente relacionada à manifestação de vida em um ambiente virtual (Nunes, 2012; Nunes, 2006; Cavalcante, 2018). O realismo é importante na produção desses efeitos visuais, especialmente quando os personagens animados interagem com o ambiente virtual que ocupam. Em tais cenários, os movimentos de um personagem virtual devem corresponder visualmente ao comportamento no ambiente real, ou a discrepância será óbvia para o espectador. Por exemplo, um personagem que está pulando e sendo levado ao solo pela gravidade deve corresponder visualmente a um objeto que está sendo lançado na mesma cena sob a mesma força gravitacional (Shapiro; Lee, 2010). O personagem nem sempre precisa representar um humano ou um animal existente; pode ser um ser em extinção ou uma criatura exótica fictícia qualquer (Silva, 2014). Algumas categorias de personagens podem ser observadas na Figura 1.

Figura 1 – Alguns exemplos de personagens arbitrários.

Fonte: Adaptado de Nunes (2012)

2.1.1 Personagens articulados

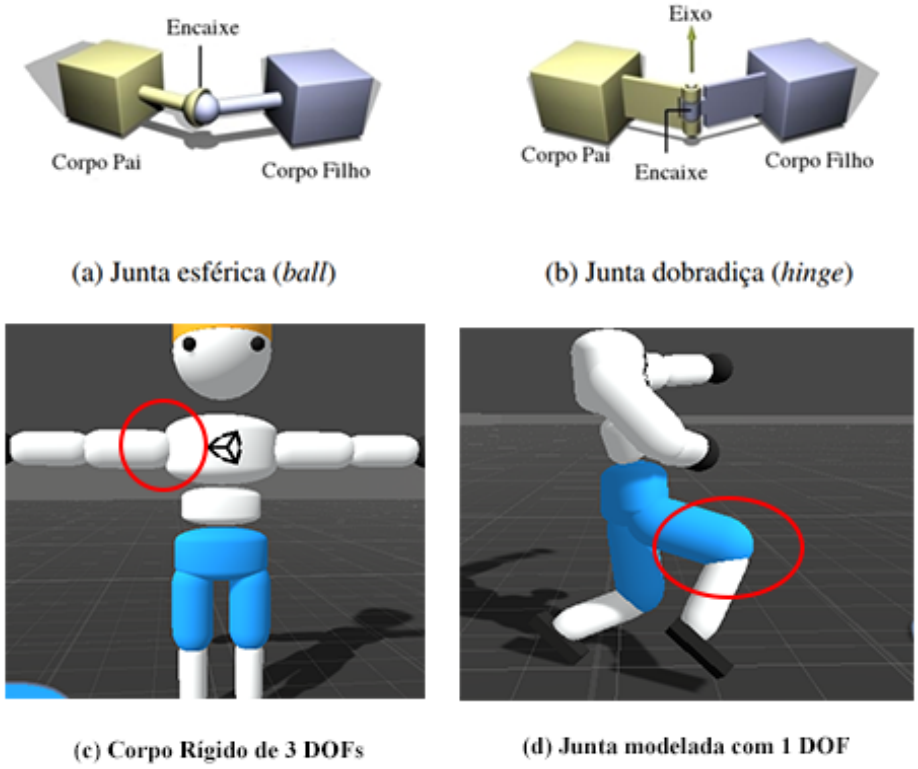
Dentre as diversas categorias de representação de personagens, a representação utilizada neste trabalho consiste em uma estrutura de corpos rígidos que se conectam através de articulações (ou juntas). Por isso, esses personagens são classificados como articulados. As juntas são responsáveis principalmente por determinar restrições de movimento entre essas partes, reduzindo seus graus de liberdade (DOF *(Degrees Of Freedom)*) (Cavalcante, 2018). Na Figura 2 estão ilustradas duas categorias de juntas mais utilizadas na modelagem de personagens humanoides.

A junta esférica, mostrada na Figura 2(a), possui três graus de liberdade e pode, por exemplo, ser usada para representar a articulação do ombro de um personagem, mostrado na Figura 2(c). Já a junta do tipo dobradiça, Figura 2(b), está limitada a um grau de liberdade e pode, por exemplo, ser usada para representar o joelho, Figura 2(d). Geralmente, são especificados ainda limites angulares inferiores e superiores para cada DOF de modo a torná-los mais parecidos com os limites naturais da articulação (Sousa *et al.*, 2021; Nunes, 2012).

2.2 Técnicas de animação relacionadas

Nesta seção, serão brevemente exploradas algumas técnicas de animação de personagens. Embora existam várias abordagens, como captura de movimentos, quadros-chave *(key-frames)*, simulação dinâmica e cinemática inversa, duas delas são consideradas mais relevantes para serem discutidas neste trabalho: captura de movimentos e simulação dinâmica.

Figura 2 – Tipos comuns de juntas de personagens articulados.



Fonte: Adaptado de Silva (2014), Nunes (2012)

2.2.1 Captura de movimentos

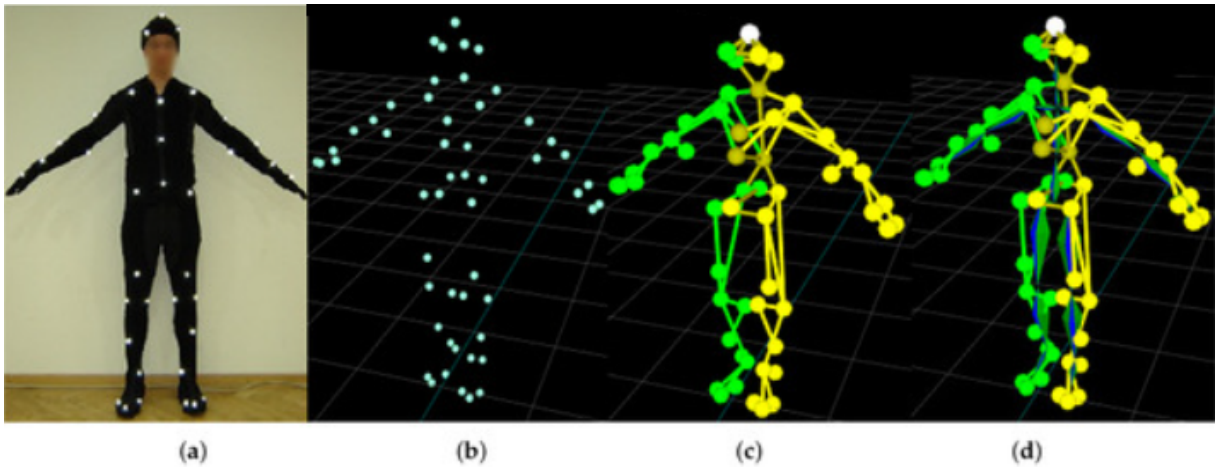
A captura de movimentos é classificada como um método cinemático, pois está diretamente ligada às posições e informações que definem os estados do personagem em determinado instante da animação. Essas poses são definidas pela manipulação direta das coordenadas de localização do personagem no ambiente, bem como dos ângulos e velocidades das juntas (Nunes, 2012). Essas informações de definição dos estados não são influenciadas por forças e torques, internos ou externos, que poderiam agir sobre o personagem e, portanto, não possuem relação com as leis de Newton. Embora as velocidades possam ser consideradas, devido à necessidade de definir o tempo de transição entre estados para determinar o ritmo da animação, elas não envolvem influências de forças ou torques.

Atualmente, a animação de personagens baseada em clipes de movimento é uma técnica muito comum na indústria de jogos e cinema (Chentanez *et al.*, 2018). Os dados dos movimentos são obtidos de atores reais, que podem ser pessoas ou outros seres executando ações (Laszlo, 1997; Araujo, 2015). Araujo (2015) aborda diversas técnicas para obter dados de movimentos capturados. Com essas informações, o animador pode transformar clipes captados em animação gráfica. A Figura 3 ilustra uma sequência abstrata das etapas para captura de

movimentos usando sistemas ópticos. Este trabalho não se aprofunda nas técnicas de mocap, mas mais informações sobre esses processos podem ser encontradas em Skurowski e Pawlyta (2021), Araujo (2015).

Uma das principais vantagens do uso de mocap é a capacidade de obter animações realísticas, muito próximas do observado no mundo real. No entanto, existem várias limitações, como a dificuldade de manter a naturalidade quando são necessárias modificações no movimento capturado devido a influências externas, à adição de outros elementos no ambiente ou a alterações do próprio ambiente, como demonstrado em Araujo (2015), Skurowski e Pawlyta (2021). Outro requisito crítico do mocap é a necessidade de usar ferramentas e hardware adequados para obter os clipes desejados, que nem sempre são acessíveis. Além disso, há limitações sobre quais tipos de seres (vivos ou fictícios) são viáveis ou possíveis de serem utilizados para capturar seus movimentos, destacando a dificuldade de trabalhar com animais e a impossibilidade de tratar espécies extintas ou totalmente fictícias. Mesmo com humanos, há cenas que podem causar risco aos atores e são difíceis de serem adaptadas. Outra desvantagem é o fato de que o personagem não possui nenhum nível de autonomia durante a animação, dependendo basicamente de reproduzir o movimento gravado.

Figura 3 – Etapas do mocap: (a) Ator com sensores; (b) Marcadores; (c) Malha do corpo; (d) Esqueleto combinado.



Fonte: Skurowski e Pawlyta (2021)

2.2.2 Simulação dinâmica

O principal critério para classificar um método utilizado na animação de personagens virtuais é identificar se a técnica considera as causas que resultam nos movimentos, ou seja, se o

método leva em conta as forças atuantes e as massas dos corpos para gerar os resultados. Como afirmado na seção 2.2.1, os métodos cinemáticos não se preocupam com os motivos que levam ao movimento e não consideram a influência dessas causas para definir as poses dos personagens em cada instante da animação (Cavalcante, 2018). Nos métodos dinâmicos, conforme Nunes (2012), as poses dos personagens são obtidas com base nas equações de movimento do sistema, através das forças e torques aplicados, em vez de serem definidas diretamente por informações de posição e velocidade dos graus de liberdade, como nos métodos cinemáticos. A simulação física é, portanto, classificada como um método dinâmico, pois sua principal característica é a obtenção das poses dos corpos a serem animados com base nos princípios fundamentais da física. Em outras palavras, a simulação física procura obedecer às leis da física clássica (Nunes, 2012; Silva, 2014).

Animações de personagens e ambientes que respeitam as leis da física procuram imitar o mundo real o mais fielmente possível, representando situações que as pessoas estão acostumadas a perceber. Dessa maneira, o uso da física em animação resulta em implementações mais complexas, exigindo maior esforço computacional e conhecimentos em áreas como dinâmica e integração numérica. A animação desses sistemas virtuais também envolve áreas como biomecânica, controle de movimento e otimização. Além disso, a animação geralmente depende de um número maior de parâmetros, às vezes não intuitivos, que necessitam de ajustes trabalhosos ou otimizações demoradas para que o resultado desejado seja obtido (Nunes, 2012). As dificuldades aumentam quando se deseja animar personagens articulados fisicamente simulados, muitas vezes resultando em simulações que podem parecer pouco naturais (até mesmo cômicas), devido aos movimentos incomuns que o personagem realiza para manter o equilíbrio, resultando em simulações não realísticas (Bergamin *et al.*, 2019).

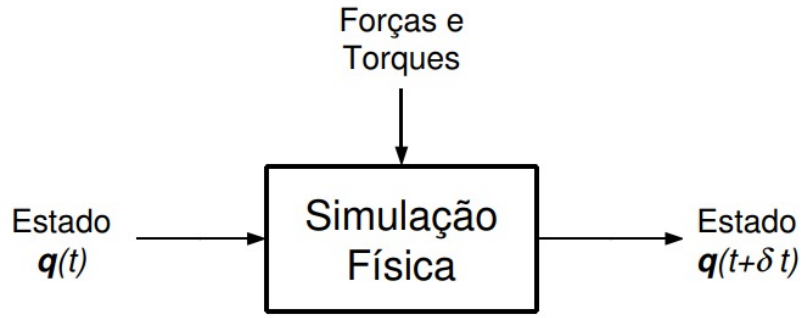
Devido às dificuldades citadas, o uso de simulação física se difundiu mais rapidamente na indústria para animar fenômenos passivos, como movimentos de fluidos, roupas e objetos rígidos não articulados ou passivos, sendo mais fáceis de serem representados graficamente usando as abstrações das leis da física. Durante muito tempo, o alto custo computacional imposto pelo uso dos simuladores físicos foi uma limitação crítica. No entanto, avanços na tecnologia de processadores e nas placas de vídeo possibilitaram que as animações baseadas em física sejam executadas em tempo real (Silva, 2014).

Como afirma Nunes (2012), a maior motivação para o uso da física em animação por computador é produzir movimentos fisicamente corretos de maneira automática. Quando a

simulação física é usada, a animação é implicitamente produzida apenas através da aplicação de forças e torques ao sistema, o que implica que o movimento resultante é automaticamente gerado e fisicamente correto conforme o modelo simplificado de física seguido pelo simulador específico utilizado. Embora exista a desvantagem de perder o controle direto sobre as poses do personagem, a vantagem de obter movimentos fisicamente corretos de maneira automática é um excelente atrativo para o uso da física (Cavalcante, 2018). É importante citar que tanto a cinemática quanto a dinâmica podem ser classificadas em direta e inversa. Para mais detalhes sobre essas abordagens, a seguinte referência pode ser consultada (Nogueira, 2007).

A Figura 4 mostra um esquema ilustrando a utilização de simulação física de um sistema que usa dinâmica direta para gerar animação, que pode ser descrito da seguinte forma: dado um modelo no seu estado atual $\boldsymbol{q}(t)$, usando dinâmica direta, forças e torques são aplicados a ele e um novo estado $\boldsymbol{q}(t+\delta t)$ é obtido através da solução de suas equações diferenciais de movimento (Nunes, 2012; Silva, 2014).

Figura 4 – Animação gerada com uso de simulação física.



Fonte: Nunes (2012)

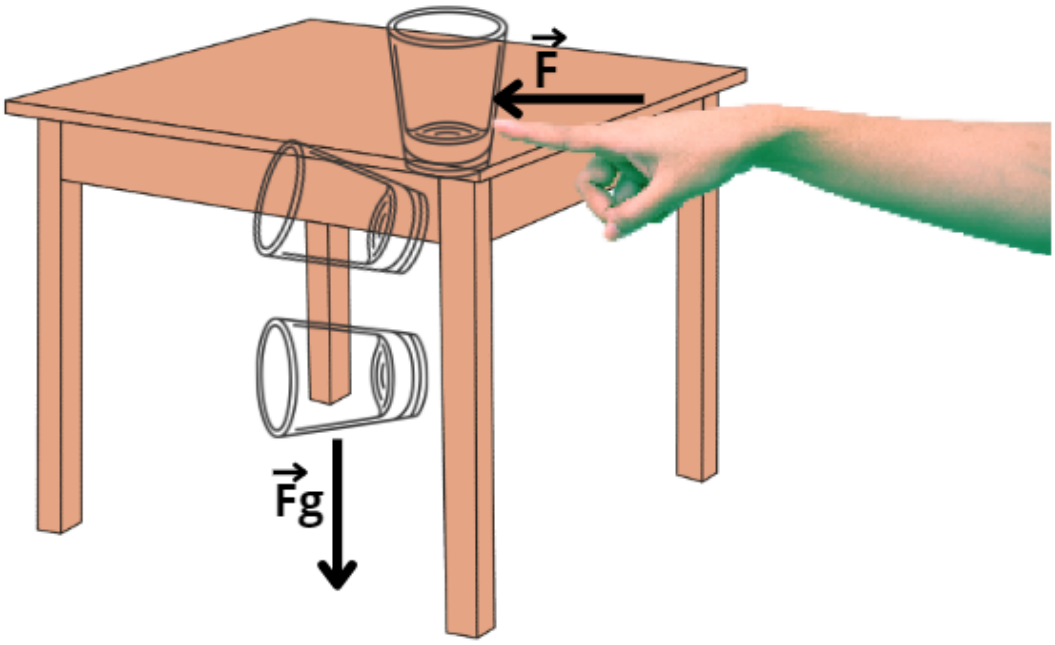
A simulação física usada neste trabalho é implementada pela ferramenta utilizada para modelagem e simulação do personagem, descrita nas seções 2.5 e 5.1. Portanto, já é um mecanismo existente que atualiza o estado do sistema adequadamente, mas que precisa ser considerado pelo animador.

2.2.2.1 Sistemas passivos versus atuados

De acordo com Nunes (2012), os sistemas fisicamente simulados são classificados em sistemas passivos e sistemas atuados. Sistemas passivos não geram torques e forças internamente, dependendo totalmente de forças e torques externos para a simulação resultante. Um exemplo

de simulação de movimento passivo seria a cena ilustrada na Figura 5, onde um copo cai de uma mesa após ser empurrado. O empurrão gera uma força através do contato, que modifica a aceleração do copo, consequentemente alterando sua velocidade a ponto de deslocá-lo para fora dos limites da mesa. A força gravitacional então age sobre o copo, resultando em uma animação plausível.

Figura 5 – Movimento passivo de um objeto.



Fonte: Própio autor (2024)

Antes de citar os sistemas atuados, é bom entender o que são atuadores em animação. Eles podem ser definidos como dispositivos integrantes do sistema que conseguem produzir movimento de acordo com os estímulos recebidos de outros dispositivos. Portanto, os sistemas atuados conseguem gerar forças e torques que se distribuem internamente em suas estruturas, conseguindo produzir movimentos por sua própria anatomia, desde que ela permita, e de forma voluntária. Os atuadores funcionam como uma simplificação dos músculos dos seres, aplicando torques nas juntas das estruturas articuladas. Nunes (2012) lembra que sistemas apenas com molas e amortecedores, desde que as propriedades dessas molas e amortecedores, tais como posição de relaxamento e rigidez, não sejam alteradas de acordo com comandos voluntários, também são considerados estruturas passivas.

Sintetizando, os sistemas dinâmicos atuados se diferenciam dos passivos principalmente por possuírem mecanismos de geração de torques internos e por conseguirem executar tarefas voluntariamente, por meio do controle de seus movimentos. Esse controle consiste na

especificação adequada das forças e torques gerados pelo conjunto de seus atuadores e, consequentemente, o sistema não depende exclusivamente de forças externas (Nunes, 2012; Silva, 2014; Cavalcante, 2018; Raibert; Hodgins, 1991).

No contexto de animação baseada em física de personagens virtuais representados por estruturas articuladas atuadas, os atuadores se localizam nas juntas e funcionam como músculos, convertendo energia armazenada em torques que agem na estrutura mecânica do sistema (Nunes, 2012; Silva, 2014; Cavalcante, 2018). A Seção 2.3 dará detalhes sobre um controlador para simulação dinâmica.

2.2.3 Combinação de métodos

Na tentativa de reduzir as desvantagens das técnicas de animação dinâmica ou cinemática e buscando potencializar as vantagens de cada uma dessas abordagens, várias pesquisas têm procurado criar soluções que usem ambas em partes do processo. A simulação física (dinâmica) pode, por exemplo, ser combinada com mocap (cinemática), usando as poses da captura de movimento como poses desejadas para os controladores baseados em física (observar Seção 2.3.1). Esses controladores, por sua vez, vão calcular os torques necessários para atingir a pose orientada pelo mocap (Wrotek *et al.*, 2006).

Aplicações da animação de personagens, tais como videogames e robótica, exigem a criação de novos movimentos em tempo real, combinando movimentos das fontes dinâmicas e cinemáticas. Por exemplo, os ataques de um boxeador virtual são principalmente impulsionados pela captura de movimento. No entanto, ele deve reagir ou cair segundo a simulação dinâmica quando é atingido. Contudo, criar um movimento em tempo real combinando movimentos simultâneos de dois tipos diferentes de fontes é um problema complexo. A combinação de simulação física com captura de movimento é também classificada como um método dinâmico, por considerar as forças que atuam no sistema (Wrotek *et al.*, 2006).

A ideia de combinar métodos não é nova. Wrotek *et al.* (2006), inspirados por Zordan e Hodgins (2002), usaram movimentos capturados para servir como poses desejadas a serem alcançadas por um controlador dinâmico para um personagem articulado. Atualmente, diversos trabalhos (Peng *et al.*, 2018; Chentanez *et al.*, 2018; Aberman *et al.*, 2019) combinam métodos cinemáticos e dinâmicos, onde os movimentos são obtidos a partir da captura de movimentos e as restrições físicas são acrescentadas para tornar os personagens animados capazes de interagir automaticamente com o ambiente.

O problema de controle, característico em simulações físicas de sistemas atuados, é muitas vezes representado como um problema de otimização. Recentemente, os avanços no campo da inteligência artificial têm contribuído significativamente, principalmente com o uso de algoritmos de aprendizado por reforço profundo, que têm apresentado resultados visualmente interessantes usando métodos dinâmicos para: realização de movimentos acrobáticos (Peng *et al.*, 2018); controle de movimentos em tempo real (Bergamin *et al.*, 2019); jogos de basquete (Liu; Hodgins, 2018); ações emergentes (Heess *et al.*, 2017) e diversas outras categorias de movimentos.

Nas pesquisas que usam abordagem dinâmica combinada com algum método cinemático, não é mais possível garantir todas as vantagens da simulação física, normalmente algum potencial da simulação física é perdido.

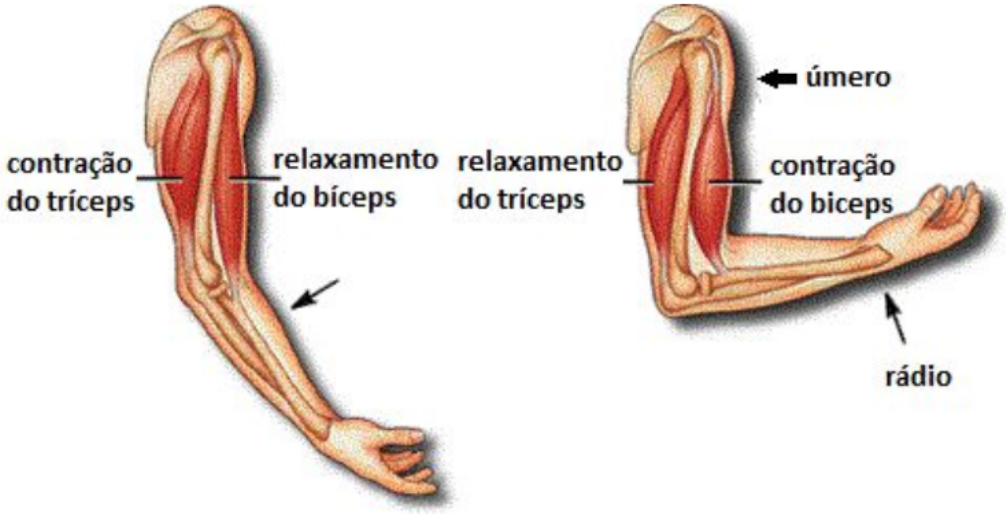
2.3 Controle da animação

Um desafio importante no controle de personagens é manter uma alta qualidade de movimentos naturais e conseguir reproduzir uma grande variedade de movimentos e estilos (Chentanez *et al.*, 2018). O fato de os personagens serem simulados fisicamente não garante que a animação seja realista. Em uma simulação que respeita as leis da física, um personagem articulado inicialmente em pé, ou seja, em uma postura ereta, não consegue manter sua postura se as forças necessárias não forem aplicadas internamente na sua estrutura. A falta de um controle adequado geralmente faz com que o personagem simplesmente caia no chão. Além de permitir que o personagem execute uma sequência de ações desejadas, um controlador ideal também deve ser responsável por manter o equilíbrio do personagem de maneira estável e ter como critério um gasto mínimo de energia (Cavalcante, 2018; Sousa, 2022).

A animação de sistemas dinâmicos atuados requer, portanto, a solução de um problema de controle. Este problema consiste em determinar as forças e os torques adequados a serem produzidos pelos atuadores do sistema, de modo a realizar o movimento desejado. Os controladores são as entidades responsáveis por fornecer esses comandos aos atuadores, funcionando como o cérebro do personagem. Em suma, o controlador de movimento corresponde à parte do cérebro responsável por enviar os impulsos nervosos que transportam a informação de como contrair os músculos de maneira adequada para aplicar as forças corretas na estrutura esquelética e realizar o movimento desejado (Nunes, 2006; Geijtenbeek *et al.*, 2011; Geijtenbeek; Pronost, 2012).

Tendo as articulações humanas como referência, os atuadores funcionam como pares de músculos. A Figura 6 ilustra o funcionamento conjunto do bíceps e do tríceps, responsáveis por realizar o movimento do antebraço. Os músculos bíceps e tríceps estão ligados ao osso rádio do antebraço e ao osso chamado úmero, localizado no braço. Quando o bíceps se contrai, puxando o osso rádio, e o tríceps relaxa, o antebraço se eleva. Ao relaxar o bíceps e contrair o tríceps, o antebraço desce. A função do cérebro, que neste caso é o controlador, é enviar sinais aos músculos indicando se devem contrair ou relaxar. Por questão de simplicidade, o efeito desses músculos agindo aos pares é representado através de um atuador angular (Cavalcante, 2018).

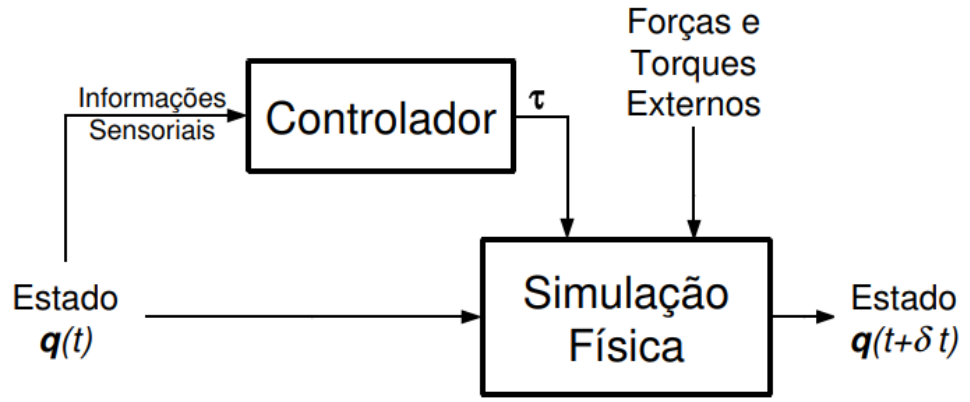
Figura 6 – Visão Anatômica de bíceps e tríceps relaxando e contraindo.



Fonte: Cavalcante (2018)

A abstração dos eventos relatados na Figura 6 é implementada computacionalmente através da ideia de um controlador realimentado, como mostrado na Figura 7. Esta figura ilustra a inserção do componente controlador agindo no laço da simulação física. A cada instante da simulação, o controlador é responsável por enviar os torques internos, τ, a serem produzidos pelos atuadores e considerados na simulação. Controladores de laço fechado recebem esta definição quando podem usar informações sensoriais para influenciar na ação do controlador (Nunes, 2012). Essas informações sensoriais são chamadas de *feedback* e consistem de qualquer informação relacionada ao estado atual do sistema, incluindo tanto a estrutura articulada quanto o ambiente. O *feedback* é geralmente usado para corrigir o movimento de acordo com mudanças imprevistas no sistema, proporcionando um controle mais estável do movimento (Silva *et al.*, 2017; Nunes, 2012).

Figura 7 – Sistema de um controlador de laço fechado.



Fonte: Nunes *et al.* (2012)

2.3.1 Controlador proporcional-derivativo

Segundo Nunes (2012), os controladores locais de baixo nível mais usados em animação de personagens virtuais são os chamados *Proportional-Derivative controllers*, ou controladores PD. O objetivo desses controladores de laço fechado é corrigir o erro entre o estado atual e o estado desejado da junta atuada, tentando fazer com que o estado desejado seja alcançado. A nomenclatura PD é usada porque a equação definida para o cálculo do torque possui um termo proporcional ao erro e outro termo que depende da derivada do erro. O cálculo do torque é dado pela Equação 2.1:

$$\tau = \kappa_p \cdot (\theta_d - \theta) + \kappa_d \cdot (\dot{\theta}_d - \dot{\theta}), \tag{2.1}$$

onde τ é o torque a ser aplicado na junta, θ representa a orientação atual da junta, θ_d a orientação desejada da junta, $\dot{\theta}$ a velocidade atual da junta, e $\dot{\theta}_d$ a velocidade desejada da junta. κ_p e κ_d são chamados ganhos do controlador, duas constantes definidas pelo animador representando as constantes da mola e do amortecedor, respectivamente. Os valores dessas constantes estão geralmente relacionados à modelagem física do personagem simulado. Para mais detalhes de como esses valores influenciam o controle, as seguintes referências podem ser consultadas (Allen *et al.*, 2007; Nunes, 2006).

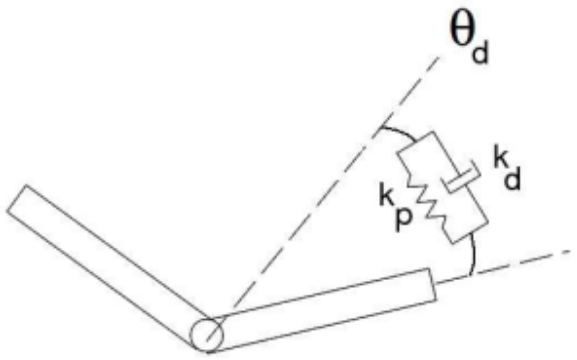
Quando os controladores PD são usados localmente no nível mais baixo do controlador, a animação do personagem pode ser definida atribuindo valores aos parâmetros θ_d e $\dot{\theta}_d$ para cada junta, em cada instante da simulação. Note que, ao especificar uma orientação desejada fixa e ao manter a velocidade desejada igual a zero ($\dot{\theta}_d = 0$), a ação do controlador PD corresponde à ação passiva de uma mola angular com amortecimento localizada na junta. Nesse caso, θ_d define

a orientação da junta em que a mola se encontra relaxada, κ_p é a rigidez da mola e κ_d define as propriedades do amortecimento.

A Figura 8 ilustra o comportamento de uma mola com amortecimento posicionada em uma junta. Uma mola angular funciona apenas levando a junta a assumir sua orientação de relaxamento, e isso não pode ser confundido com a ação de um atuador, como o funcionamento de um motor, por exemplo (Nunes, 2012). Entretanto, um motor ou atuador (influência ativa) pode ser simulado através de um ângulo de relaxamento da mola que pode ser continuamente atualizado pelo controlador. Esse ângulo de relaxamento sendo atualizado representa o θ_d.

O trabalho de Panne *et al.* (1994) usa controladores PD para gerar os torques necessários no seu *Pose Control Graphs*, que é uma máquina de estados que tem uma pose desejada particular do personagem associada a cada um desses estados. Cada uma dessas poses, uma por vez, passa a ser a pose desejada a ser alcançada ao longo do tempo. Zordan e Hodgins (2002) usam controladores PD para seguir dados de captura de movimento. Em trabalhos mais recentes, Yang *et al.* (2020) e Peng *et al.* (2020) também usam controladores PD com DRL para tarefas de locomoção.

Figura 8 – Mola com amortecimento agindo em uma junta.



Fonte: Silva (2014), Nunes (2012)

2.4 Aprendizado de máquina

Nesta seção, serão discutidos alguns conceitos sobre o campo da inteligência artificial definido como aprendizado de máquina, mais especificamente sobre a categoria de aprendizado por reforço profundo, abordagem usada neste trabalho.

2.4.1 Categorias de aprendizado

O Aprendizado de Máquina (*Machine Learning — ML*) é uma ramificação da inteligência artificial, que pode ser definida como o campo de estudo que dá aos computadores a capacidade de aprender sem serem explicitamente programados, reduzindo a necessidade de um grande conjunto de dados e o tempo computacional (Youssef *et al.*, 2019). O ML fornece a um sistema a capacidade de aprender com os dados automaticamente e não de uma maneira planejada. Funciona alimentando o sistema com informações e observações que podem ser usadas para encontrar padrões e fazer previsões sobre resultados futuros (Learning, 2017). A máquina aprende por sessões de treinamentos (Lanham, 2018). Um problema de aprendizado de máquina geralmente consiste na aplicação de um algoritmo que constrói um modelo matemático para um conjunto de dados, minimizando uma função de perda (Ma, 2019) e pode ser classificado em três categorias: aprendizado supervisionado, não supervisionado e aprendizagem por reforço (Raschka; Mirjalili, 2017), existindo ainda uma extensão do último, definida como aprendizagem por reforço profundo. Lanham (2018) descreve a seguir estes métodos como treinamentos de aprendizagem de máquina:

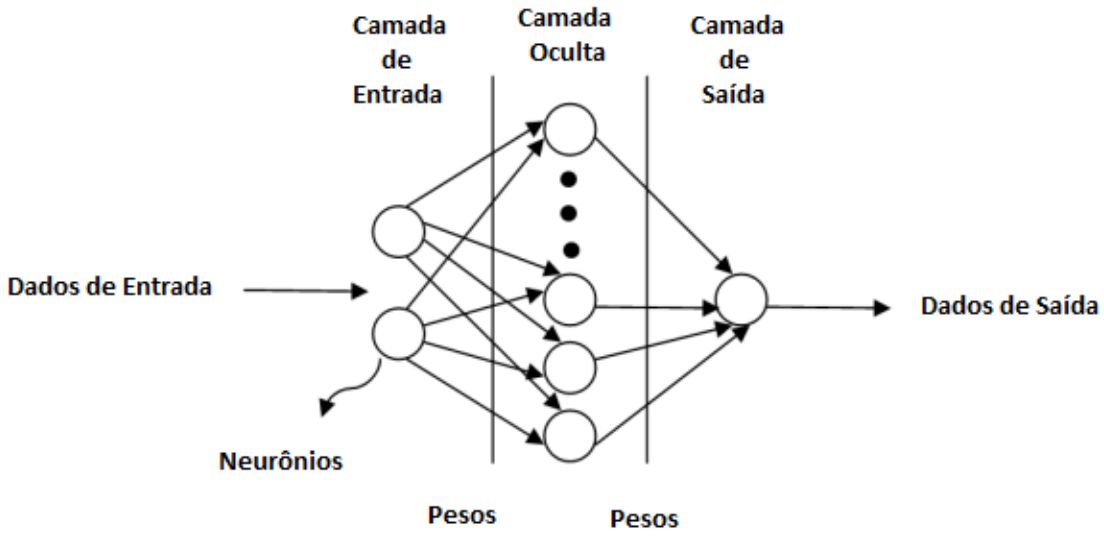
- *Supervised Training* (Treinamento Supervisionado): possui um conjunto de dados organizado com entradas e suas saídas correspondentes, requisitando dados de entrada e de saída rotulados, usando um conjunto de informações de treinamento para construir um modelo. Os algoritmos constroem este modelo matemático relacionando as entradas às saídas correspondentes do conjunto de dados. Este é um método comum geralmente usado para duas categorias de problemas: fazer predições, problemas de regressão em que o objetivo é relacionar a entrada a uma saída de valor contínuo; ou fazer classificações, quando o objetivo é encontrar a classe para uma nova entrada.
- *Unsupervised Training* (Treinamento Não Supervisionado): É um método que examina os dados de forma autônoma e desenvolve uma classificação. O conjunto de dados possui as entradas, mas não possui as saídas correspondentes, usando os algoritmos para identificar os padrões no conjunto de dados de treinamento e categorizar os objetos de entrada com base nos padrões que o próprio sistema identifica. Os algoritmos analisam a estrutura subjacente dos conjuntos de dados, extraindo informações úteis ou recursos deles. A tarefa mais comum em aprendizado não supervisionado é a clusterização, onde o conjunto de dados é dividido em grupos com características semelhantes.
- *Reinforcement Learning* (Aprendizagem por Reforço): nesta técnica não há um conjunto

de dados com pares de entradas e saídas. Em vez disso, existe um agente com a habilidade de perceber o estado do ambiente e emitir uma resposta conforme os estímulos. Os agentes tornam-se inteligentes por meio da modelagem dos dados extraídos dos sensores, recompensando os acertos e punindo os erros, aumentando sua acurácia na tarefa de atingir o objetivo determinado (Silva *et al.*, 2020). Neste trabalho, foi utilizado aprendizado por reforço para tratar o problema de locomoção em personagens fisicamente simulados.

2.4.2 Redes neurais

Os métodos de aprendizado de máquina que mais se destacam se baseiam em redes neurais artificiais, inspiradas na estrutura e em aspectos funcionais das redes neurais biológicas (Cavalcante, 2018). Segundo Poole e Mackworth (2010), as redes neurais artificiais podem ser entendidas como modelos computacionais inspirados na abstração de um neurônio real, cuja finalidade é simular o processo de aprendizado e resolução de problemas complexos. Com essas redes é possível reconhecer padrões, relacionar, agrupar e classificar dados e informações, visando aprimorar continuamente sua busca por melhores resultados. Sua arquitetura é baseada no sistema neurológico humano, que possui "receptores" dispostos em uma ou várias camadas. Para cada receptor é atribuído um peso recalculado na medida em que a rede aprende. As redes neurais são sistemas de neurônios interconectados e a Figura 9 ilustra de forma geral esse conceito.

Figura 9 – Abstração de uma rede neural simples.



Fonte: Okwuashi *et al.* (2012)

Na primeira camada, a rede recebe valores de entrada, chamados recursos. Em seguida, as camadas intermediárias escondidas realizam o processamento necessário para gerar os valores de resposta, obtidos na última camada, a camada de saída. O número de nós e a estrutura da rede é normalmente escolhida previamente, fazendo parte dos chamados hiperparâmetros. Os valores nas arestas são chamados de pesos, responsáveis por definir a influência que os nós correspondentes terão no resultado da rede. Ou seja, os pesos correspondem ao grau de ativação da informação que será transmitida entre os nós que cada aresta conecta e cada novo conjunto de pesos corresponde a um novo comportamento da rede. Como uma rede neural possui muitos pesos, eles precisam ser ajustados automaticamente com o intuito de continuamente melhorar os resultados parciais seguindo algum critério. Os significados dos valores de entrada e saída da rede neural variam conforme o problema atacado (Cavalcante, 2018).

A rede neural precisa passar por um processo de treinamento onde várias entradas e suas saídas desejadas são mostradas. Para cada instância de treinamento, a rede neural altera seus pesos internos, num esforço para obter o conjunto de saída desejado para um determinado conjunto de entrada. Esse processo continua até que um critério de parada seja atendido. Após essa fase, essa rede memoriza seu aprendizado salvando os pesos encontrados, sendo capaz de predizer a saída correspondente a uma nova entrada (Sousa, 2022).

No contexto deste trabalho, as entradas da rede consistem de informações sensoriais obtidas das observações e interações com o ambiente de simulação do personagem, enquanto as saídas da rede consistem de informações que serão traduzidas para uma pose desejada, influenciando na atuação do personagem.

2.4.3 Aprendizado por reforço

Aprendizado por reforço tem chamado muita atenção atualmente com seus resultados em várias aplicações: videogames, como jogos de *Atari* (Mnih *et al.*, 2013) e *Starcraft* (Vinyals *et al.*, 2019); jogos de tabuleiro, como Xadrez (Lai, 2015) e GO (Silver *et al.*, 2017); animação de personagens articulados (Peng *et al.*, 2018; Yu *et al.*, 2018); e robótica (Yang *et al.*, 2020; Liang *et al.*, 2018).

Correspondente à parte do aprendizado de máquina que estuda os agentes de treinamento, fazendo uma série de soluções decisivas, o RL permite que a rede seja treinada sem um conjunto de dados (*dataset*) ou um modelo prévio. A aprendizagem é obtida em um processo de treinamento ganhando experiências por ações passadas (Sutton; Barto, 1998; Urmanov *et al.*,

2019). O treinamento é modelado para recompensar ações que aumentam a probabilidade de sucesso do objetivo determinado, ou seja, o ganho de experiência se dá através de uma recompensa após as ações, que pode ser positiva, se a ação realizada implicar em uma aproximação do objetivo, ou negativa, caso contrário (Sutton; Barto, 1998; Lanham, 2018).

Os conceitos de agentes, ambientes, estados, ações e recompensas são cruciais para entender o método de aprendizagem por reforço, os quais são descritos a seguir (Reis, 2003; Ottoni *et al.*, 2012; Sutton; Barto, 1998):

Agente: Um agente deve ter capacidade de percepção do ambiente situado e a partir delas executar ações. Em animação gráfica, o agente é equivalente ao personagem a ser animado;

Ação: Pode ser entendida como as habilidades do agente, as ações que ele pode executar;

Ambiente: O espaço onde o agente realiza suas ações;

Estado: É uma representação do ambiente no instante em que o agente se encontra. Este instante relaciona o agente a objetos que possam estar ao seu redor, tais como ferramentas, obstáculos, inimigos ou prêmios. O acesso às informações do estado atual é fundamental na escolha da próxima ação a ser executa, o que irá provocar um novo estado;

Recompensa: É o retorno pelo qual medimos o sucesso ou fracasso das ações tomadas por um agente. O agente não precisa ter conhecimento das ações que deve tomar, mas deve descobrir quais ações o levam a maximizar a quantidade de recompensas recebidas;

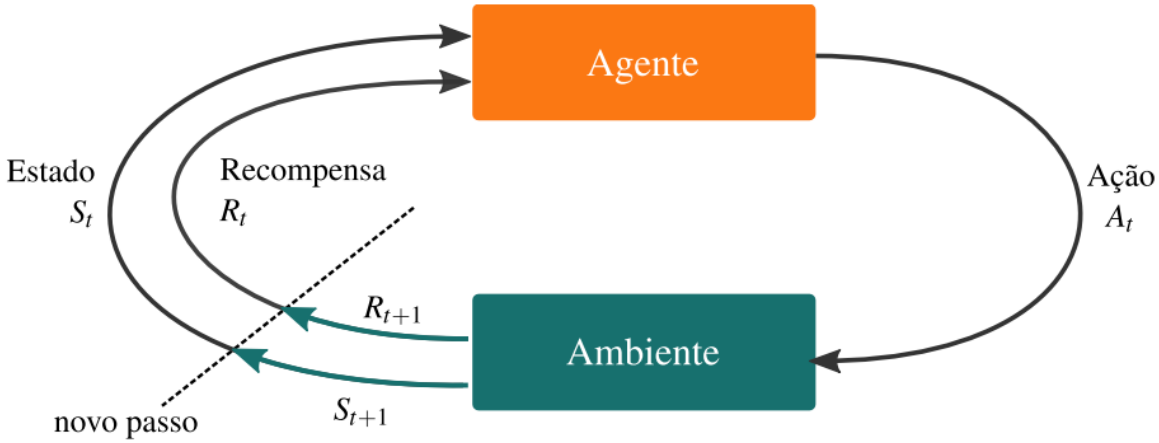
Política: É uma estratégia que o agente emprega para determinar a próxima ação com base no estado atual. Dependendo de sua atuação, o agente recebe uma recompensa positiva ou é penalizado, sempre buscando um conjunto de ações que o leve a percorrer o caminho ótimo. A uma estratégia ideal, que sempre escolhe as melhores ações possíveis, dá-se o nome de política ótima.

A Figura 10 mostra o ciclo de aprendizagem por reforço, onde o agente interage com o ambiente e recebe uma recompensa (Rt), positiva ou negativa, orientando suas ações e definindo seus objetivos conforme o estado (St) e ações (At) em cada fase do treinamento.

Avaliar o quanto um programa deve explorar novos estados e ações (*exploration*) ou estados e ações já conhecidos por explorações anteriores (*exploitation*) é um desafio em aprendizado por reforço. Uma maneira para descrever problemas de aprendizado por reforço é utilizar o formalismo matemático de Processo de Decisão de Markov.

Processo de Decisão de Markov (Markov Decision Process (MDP)) é um modelo

Figura 10 – Ciclo do aprendizado por reforço.



Fonte: Silva *et al.* (2020)

matemático que descreve problemas de tomada de decisão sequencial que podem sofrer influência de fatores aleatórios. Eles possuem a propriedade de Markov que diz que o estado atual depende apenas de um número finito de estados anteriores, e o mais simples é o MDP de primeira ordem, em que o estado atual depende apenas do estado anterior e não do seu histórico (Sutton; Barto, 2018). Segundo Arulkumaran *et al.* (2017), MDP é definido como uma 5-tupla $\langle S, A, T, R, \gamma \rangle$, onde:

- S é o conjunto dos possíveis estados do ambiente;
- A é o conjunto das ações que podem ser executadas em um estado;
- T é a função de transição $T : S \times A \times S \Rightarrow [0,1]$, que representa a probabilidade de alcançar o estado $s' \in S$ se a ação $a \in A$ for executada estando em $s \in S$, $T(s,a,s')$;
- R é a função de recompensa $R : S \times A \Rightarrow \mathbb{R}$, que mede a qualidade de cada estado alcançado pela execução da respectiva ação $a \in A$ no estado $s \in S$, $R(s,a)$;
- $\gamma \in [0,1]$ é o fator de desconto, que representa a preferência do agente pelas recompensas atuais em comparação com as recompensas futuras.

Para cada instante t no MDP, o agente encontra-se em um estado s_t do ambiente. Ele executa uma ação a_t, dentre as ações disponíveis para aquele estado. A função de transição $T(s_t, a_t, s'_{t+1})$ define a probabilidade de atingir o estado s' no próximo instante $t+1$, para essa ação executada em s. Depois da mudança de estado o agente recebe uma recompensa $R(s,a)$ e o fator de desconto indica quanto as recompensas futuras são importantes em comparação com as atuais.

Para solucionar um MDP, é necessário encontrar uma política (π) que representa

um mapeamento de uma ação para cada estado do ambiente. As recompensas obtidas em cada estado são usadas para calcular a utilidade esperada (U_h) do histórico das ações executadas, dessa forma podendo medir a qualidade da política:

$$U_h = R(s_0) + \gamma R(s_1) + \gamma^2 R(s_2) + ... + \gamma^n R(s_n). \tag{2.2}$$

A utilidade esperada de um estado s usando uma política π pode ser definida como:

$$U^{\pi}(s) = E\left[\sum_{t=0}^{\infty} \gamma^t R(s_t)\right]. \tag{2.3}$$

Para escolher a melhor ação a ser executada em um estado, busca-se a utilidade máxima esperada para aquele estado, ou seja, o agente deve escolher a ação que maximize sua recompensa esperada. Segundo Russell e Norvig (2002), a utilidade de um estado é a recompensa imediata correspondente a esse estado mais a utilidade descontada esperada do próximo estado, assumindo que o agente escolha a ação ótima. Esse cálculo é feito utilizando a equação de Bellman:

$$U(s) = R(s) + \gamma \max_a \sum_{s'} T(s,a,s')U(s'). \tag{2.4}$$

Com o valor das utilidades dos estados se constrói uma política ótima (π^*), obtendo as ações que maximizem a utilidade esperada do estado subsequente:

$$\pi^*(s) = \operatorname*{argmax}_a \sum_{s'} T(s,a,s')U(s'). \tag{2.5}$$

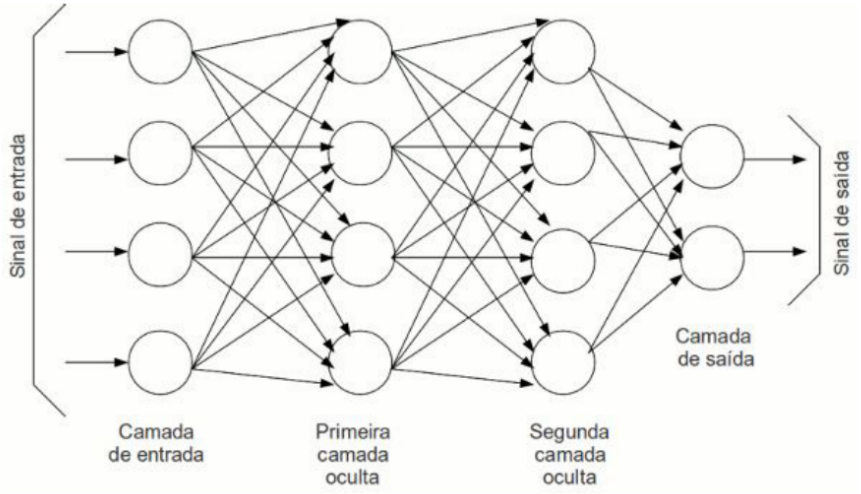
Para construir as políticas ótimas, duas abordagens podem ser usadas: iteração de valor e iteração de política. Na iteração de valor, calcula-se a utilidade de cada estado usando a equação de Bellman (2.4) e, de posse dos valores de cada estado, a melhor ação é escolhida (2.5). Na iteração de política, usa-se a equação de Bellman (2.4) sem o operador max, usando como referência a ação obtida pela política atual. A partir de uma política inicial aleatória, a cada iteração, a política vai sendo aperfeiçoada de maneira gulosa, buscando modificar as ações recomendadas para cada estado (França, Ícaro Goulart Faria Motta, 2019).

2.4.4 Aprendizado por reforço profundo

A principal diferença entre aprendizagem de máquina por reforço e aprendizagem de máquina por reforço profundo é que a segunda possui uma quantidade maior de camadas de processamento, como ilustra a Figura 11. Na primeira camada, a rede recebe valores de entrada

e em seguida as camadas ocultas intermediárias realizam o processamento necessário da rede neural para gerar os valores de resposta, possibilitando a aprendizagem em diferentes níveis de abstração. Cada camada da rede neural baseia-se em sua camada anterior, gerando relações entre os dados observados e detectando padrões (Cavalcante, 2018).

Figura 11 – Representação de uma rede neural de múltiplas camadas.



Fonte: Silva (2014), Nunes (2012)

A vantagem de usar aprendizagem de máquina por reforço profundo para animar personagens é o fato de não requisitar dados de treinamento prévios e poder aprender explorando ambientes desconhecidos com o fornecimento de um mecanismo de recompensa bem definido, descrevendo o propósito do agente. Com a execução de várias tentativas (episódios) para explorar o ambiente e receber recompensas pelas ações selecionadas, o agente pode construir um conhecimento derivado da experiência real, em vez de dados predefinidos. Isso pode ser particularmente útil em ambientes interativos, como jogos de computador, quando o agente pode encontrar inúmeras situações diferentes que não podem ser analisadas e rotuladas por especialistas (Albuainain; Gatzoulis, 2020).

Este trabalho usa o treinamento de uma rede neural profunda para animar personagens fisicamente simulados, combinado com técnicas apresentadas em outras pesquisas.

2.5 Unity

Para implementar os ambientes dos experimentos deste trabalho, foi utilizada a *Unity* 3D, uma ferramenta multiplataforma de desenvolvimento 3D em tempo real. Ela recebeu adoção generalizada nos setores de jogos, AEC (Arquitetura, Engenharia, Construção), automotivo e

cinematográfico, e é usada por uma grande comunidade de desenvolvedores para fazer uma variedade de simulações interativas, desde pequenos jogos móveis e baseados em navegador, até jogos de alta qualidade como jogos de console e experiências AR (Realidade Aumentada) / VR (Realidade Virtual) (Juliani *et al.*, 2020).

A *Unity* suporta diversos *plug-ins* para acrescentar diferentes funções, tais como: realidade virtual, realidade aumentada, inteligência artificial, além de possuir uma loja onde diversos *assets* (recursos) podem ser adquiridos. Existe a versão *Personal*, gratuita, tornando a *Unity* mais acessível a desenvolvedores de jogos e pesquisadores. Os experimentos realizados nesta pesquisa foram realizados usando a versão 2020.3.48f1 *Personal* do *Unity* 3D.

2.5.1 ML-Agents

Com o aumento das pesquisas usando DRL, destaca-se o surgimento de ferramentas modernas que oferecem ambientes de simulação e treinamento de agentes, como a usada nos experimentos deste trabalho, o *Unity Machine Learning Agents Toolkit (ML-Agents)*, um kit de aprendizado de máquina da *Unity Engine* (Urmanov *et al.*, 2019).

O *ML-Agents* é um *plug-in* de código aberto da *Unity* que oferece a oportunidade de combinar o mecanismo da *Unity* com o aprendizado por reforço. Os agentes podem ser treinados por diferentes métodos de aprendizado de máquina: aprendizado por reforço, aprendizado por imitação e outros métodos de aprendizado de máquina com a ajuda de APIs baseadas em *PyTorch* (Juliani *et al.*, 2020). Este kit de ferramentas permite realizar treinamentos com base em algoritmos de última geração através das bibliotecas TensorFlow, a biblioteca padrão para as primeiras versões do *ML-Agents*, e *PyTorch*, a biblioteca padrão para as versões atuais (usada nos experimentos desta pesquisa).

O *ML-Agents* utiliza dois principais algoritmos de aprendizado por reforço profundo: *Proximal Policy Optimization (PPO)* e *Soft Actor-Critic (SAC)* (Juliani *et al.*, 2020; Urmanov *et al.*, 2019). Neste trabalho, os treinamentos dos agentes foram realizados com o algoritmo PPO. Atualmente, o *ML-Agents* está disponível na versão 22, que apresenta melhorias significativas em relação a versões anteriores, incluindo correções de erros, ajustes de desempenho e a adição de novos recursos. Entre as novidades estão dois métodos: *Self-Play*, que permite o treinamento de agentes em cenários competitivos, favorecendo a maximização das recompensas em jogos adversariais complexos, e o *Generative Adversarial Imitation Learning (GAIL)*, uma técnica voltada para a imitação de comportamentos humanos a partir de demonstrações, ideal quando se

deseja que os agentes aprendam a executar tarefas de maneira similar a humanos.

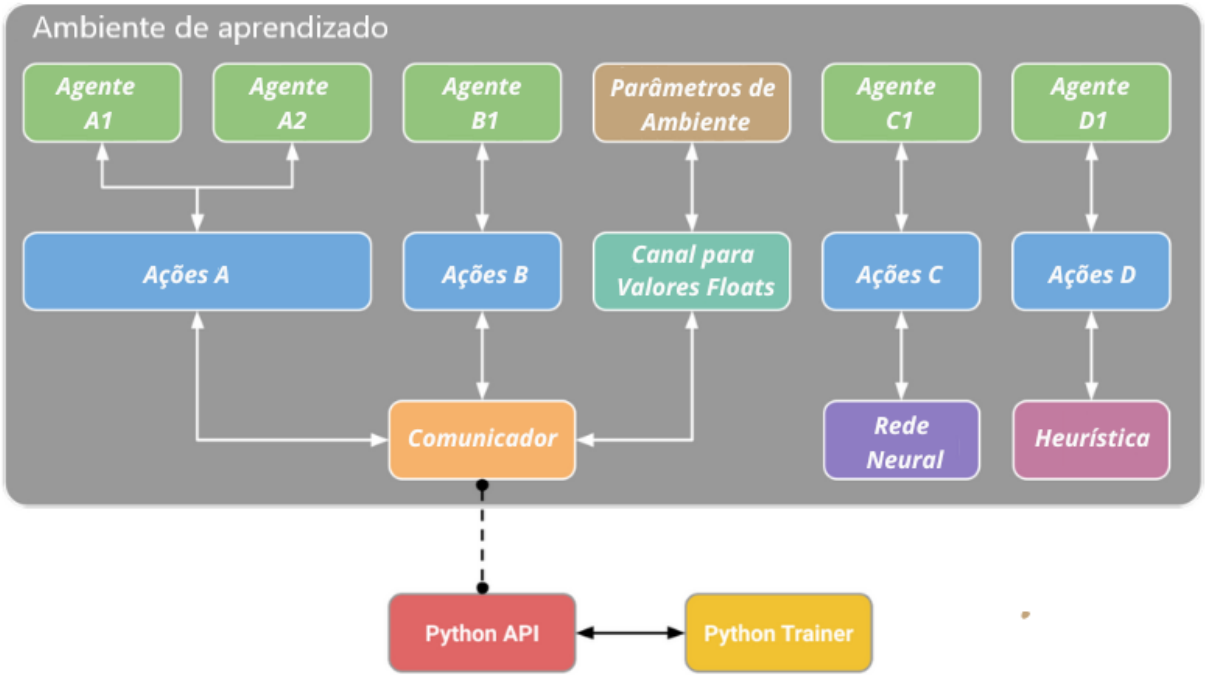
Segundo Juliani *et al.* (2020) e a própria documentação da Unity (2020), o ML-Agents é composto de sete elementos de alto nível, que podem ser visualizados na Figura 12:

- Ambiente de aprendizado: composto por uma cena na *Unity*, contém os agentes e todos os elementos necessários para ele fazer suas observações, agir e aprender. Possui três componentes básicos:
  - Agente: um *gameobject* na cena da *Unity* com capacidade de fazer observações do ambiente, realizar ações e receber recompensas. Cada agente está vinculado a um comportamento.
  - Comportamento: responsável por receber as observações e recompensas feitas pelo agente do ambiente para tomar as decisões e realizar as ações. Pode ser de três tipos: aprendizagem, que não possui uma política vinculada e está pronto para ser treinado; heurística, que contém um *script* que diz ao agente o que fazer; e inferência, que usa um modelo já treinado.
  - Parâmetros do Ambiente e Canal Lateral: permitem a comunicação de propriedades específicas entre o ambiente e os agentes, ajudando na customização e controle dos experimentos.
- *API Python*: é a interface em *Python* para interagir e manipular um ambiente de aprendizado (não faz parte da *Unity*). Possibilita utilizar outros algoritmos de aprendizado por reforço, além dos implementados pelo *ML-Agents*.
- Comunicador: faz parte do ambiente de aprendizado e permite que os componentes da *Unity* se comuniquem com a *API Python*.
- *Python Trainer*: contém os algoritmos de aprendizado de máquina utilizados para treinar os agentes.

Os experimentos apresentados neste trabalho foram realizados usando a versão 14 do *ML-Agents*[1].

[1] https://github.com/Unity-Technologies/ml-agents/blob/release_14_docs/docs/Readme.md

Figura 12 – Um ambiente de aprendizado criado usando o Editor Unity, contendo agentes e um conjunto de componentes responsáveis pela coordenação global da simulação do ambiente. Os agentes são responsáveis por coletar observações e executar ações. O ambiente inclui parâmetros do ambiente e propriedades de canais laterais de ponto flutuante, que ajudam na customização e controle dos experimentos.



Fonte: Unity (2020), Juliani *et al.* (2020)

3 TRABALHOS RELACIONADOS

A animação de personagens fisicamente simulados ainda não recebeu ampla adoção na indústria do entretenimento porque os métodos de controle nesse contexto ainda não apresentam o mesmo grau de estabilidade, realismo e controle quando comparados com métodos que editam diretamente as poses, sem influências de forças (Silva *et al.*, 2017), ou que simplesmente reproduzem as poses gravadas (mocap). Nos últimos anos, trabalhos usando aprendizagem por reforço profundo têm conseguido avanços surpreendentes. A principal vantagem do uso do DRL, quando usado sem combinação com outros métodos, é a capacidade de os personagens se adaptarem automaticamente ao ambiente de aprendizado de maneira espontânea. Entretanto, problemas com realismo, controle e estabilidade ainda precisam continuar sendo trabalhados.

Para superar o problema da falta de realismo do DRL, alguns autores exploram o método combinando com captura de movimento. Os dados de mocap têm servido para guiar o processo de aprendizado e tornar os resultados mais naturais, gerando avanços significativos (Peng *et al.*, 2018; Liu; Hodgins, 2018; Park *et al.*, 2019; Chentanez *et al.*, 2018). Porém, a combinação do DRL com mocap pode reduzir a capacidade de generalização de ações para novos personagens e situações inesperadas, além de apresentar outras limitações como citadas na Seção 2.2.1.

Visando aproveitar a característica mais atraente do uso de DRL, sendo a expectativa de que os personagens possam aprender comportamentos por conta própria, emergindo com o mínimo de informação necessária para definir o resultado desejado, este trabalho busca explorar estratégias que possam ser combinadas com o DRL, sem precisar usar as amarras da captura de movimento.

No restante deste capítulo, são apresentados breves resumos dos trabalhos que mais contribuíram para esta obra, começando com aqueles que utilizam *Deep Reinforcement Learning (DRL)* com e sem captura de movimentos. Em seguida, são citados os estudos que abordaram a simetria como forma de controle da animação. Depois, são mencionados alguns trabalhos que recorreram à intervenção artificial, usando torques externos para o controle da estabilidade dos personagens. O artigo de Nunes *et al.* (2012), que inspirou a adoção das recompensas condicionais, também é apresentado. Finalmente, discutimos alguns trabalhos que, como este, usaram a mesma plataforma de desenvolvimento e aplicaram DRL através do ML-Agents.

3.1 Aprendizado por reforço profundo (DRL)

Recentes avanços no campo do controle de movimento e animação de personagens usando DRL levaram a contribuições significativas. Kwiatkowski *et al.* (2022) realizaram uma revisão aprofundada das técnicas de RL aplicadas à animação de personagens, fornecendo uma visão detalhada dos avanços, desafios e direções futuras.

Um trabalho notável destacou o potencial do DRL em imitar movimentos capturados, redirecionando a atenção da comunidade para técnicas de DRL. Ao alcançar resultados comparáveis em qualidade às sequências originais de captura de movimento (mocap), Peng *et al.* (2018) expandiram o conceito inicial de integração de mocap com simulação física introduzido por Zordan e Hodgins (2002). Peng *et al.* (2018) conseguiram simular com sucesso personagens de diferentes estruturas, como humano, robô, dinossauro e dragão. Esses personagens simulados aprendem diversas categorias de movimentos de locomoção e acrobacias utilizando uma função de recompensa que os premia por seguir poses de clipes curtos de movimentos capturados e por atingir objetivos secundários especificados pelo usuário. A política aprendida permite ao personagem interagir com o ambiente e se recuperar de perturbações. Eles também exploraram um método para integrar diferentes tipos de movimentos em um único personagem. Seu trabalho inspirou o uso de mocap e vídeo no contexto de DRL aplicado à animação de personagens, como observado em estudos subsequentes de Bergamin *et al.* (2019), Peng *et al.* (2021), Yang e Yin (2021), Gamage *et al.* (2021), Peng *et al.* (2018), Xie *et al.* (2021). Dentre esses, é importante destacar o trabalho de Bergamin *et al.* (2019), que, assim como este trabalho, apresenta o controle em tempo real como uma de suas contribuições principais. Eles criaram o DReCon, um sistema que utiliza aprendizado por reforço (RL) para controlar personagens humanos fisicamente simulados. O sistema aprende a partir de dados de captura de movimento não estruturados, com foco na responsividade ao input do usuário, na qualidade da animação e no baixo custo de execução em tempo real. O treinamento envolve a criação de políticas que podem responder de forma responsiva e natural às entradas do usuário.

Outros trabalhos também basearam-se nos fundamentos lançados por Peng, utilizando DRL para refinar a imitação de dados de mocap. Chentanez *et al.* (2018) introduziram métodos para melhorar a robustez dessas imitações, enquanto Won *et al.* (2022) exploraram o uso de modelos generativos para aumentar o realismo físico das animações de personagens. Ren *et al.* (2023) introduziram o DiffMimic, uma técnica eficiente para imitar movimentos usando física diferenciável, representando um avanço significativo no tempo de treinamento e no controle de

movimentos complexos.

Ainda em relação à captura de movimentos combinada com o DRL, vale citar o trabalho de Liu e Hodgins (2018), que utilizou dados de captura de movimentos de jogadores de basquete, usando DRL para criar controladores robustos que conseguem animações de dribles diversos. Eles também adotaram um mecanismo que separa o controlador do sistema em componentes de controle de locomoção e controle de braço, onde cada componente é treinado separadamente. Park *et al.* (2019) também utilizou clipes de movimentos, todavia, uma base de dados de movimentos não organizados e pouco rotulados, onde os personagens podem interagir entre si e com o ambiente, utilizando um gerador de movimentos e redes neurais recorrentes para o controlador. O gerador de movimentos é responsável por guiar a simulação usando uma sequência de quadros de movimentos futuros a serem rastreados pelo DRL. A pesquisa propõe um controle interativo onde o usuário pode modular os objetivos de controle, permitindo que o personagem realize várias habilidades motoras dinâmicas e interaja fisicamente com o ambiente, apresentando um nível de interatividade e controle multi-objetivo não comum em outros trabalhos que utilizam captura de movimento e DRL.

3.2 DRL sem mocap

No entanto, confiar em dados de mocap para guiar o aprendizado pode introduzir vieses ou limitações inerentes aos movimentos capturados, que abordagens de DRL sem mocap podem superar aprendendo diretamente do ambiente. Embora a integração de mocap continue valiosa em certos contextos, como capturar nuances ou gestos humanos, adotar DRL sem mocap apresenta um caminho promissor para avançar técnicas que alcançam animação de personagens não restrita por dados gravados, levando a resultados de animação mais versáteis e criativos (Yu *et al.*, 2018).

Li *et al.* (2023) utilizaram *replay* de experiência retrospectiva para desenvolver habilidades de locomoção diversas e fisicamente realistas. Sua abordagem, denominada Curricular *Hindsight*, ajusta dinamicamente as tarefas de treinamento com base no desempenho anterior do agente. Isso permite a geração de uma variedade mais ampla de movimentos, demonstrando a adaptabilidade e robustez do aprendizado por reforço profundo (DRL) em diferentes contextos. Eles testaram sua técnica em vários tipos de robôs com pernas, mostrando que a abordagem é capaz de melhorar a eficiência do aprendizado e a capacidade de adaptação a terrenos e condições variadas. Yin *et al.* (2021) empregaram uma busca *bayesiana* por diversidade e um *Autoencoder*

Variacional de Pose (P-VAE) que exploram eficientemente estados de decolagem (momento em que o personagem impulsiona o chão e perde contato) e desenvolvem políticas de controle para descobrir estratégias de salto atlético diversas e realistas sem depender de dados de mocap. P-VAE melhora a qualidade visual ao restringir o espaço de ação para um subespaço de poses naturais. Da mesma forma, as restrições de simetria propostas neste trabalho também operam no espaço de ação.

Sem usar captura de movimentos, Heess *et al.* (2017) treinam personagens para andar e correr em solos irregulares e dinâmicos, com dificuldades gradualmente elevadas, utilizando uma função de recompensa simples. Eles naturalmente obtêm movimentos de correr, pular, agachar e virar, na medida em que essas habilidades vão sendo exigidas pelo próprio ambiente, sem ajustes explícitos na função de recompensa. O artigo demonstra que recompensas simples, quando combinadas com ambientes variados, podem resultar em comportamentos complexos. Yu *et al.* (2018) exploram uma abordagem sem o uso de captura de movimento (mocap), utilizando aprendizado por reforço profundo (DRL), para criar controladores de locomoção que são simétricos e consomem pouca energia. Ao invés de modificar a função de recompensa, como é comum em muitos trabalhos, eles introduzem um termo na função de perda para encorajar a simetria na locomoção, resultando em movimentos mais naturais e eficientes energeticamente. Além disso, eles definem termos específicos de recompensa para controlar a velocidade da locomoção. O controle da velocidade, combinado com a premiação pelo baixo uso de energia, faz com que a locomoção mais adequada para a respectiva velocidade surja naturalmente. No entanto, para atingir uma nova velocidade desejada, é necessário realizar um novo treinamento.

Ainda no contexto de DRL, Ranganath *et al.* (2019) propõem uma abordagem inovadora para o aprendizado de habilidades motoras em personagens altamente articulados, utilizando uma base reduzida de movimentos para definir uma representação simplificada, mas significativa, para a saída da rede. Esses movimentos, conhecidos como coativações, correspondem a oscilações coordenadas dos *Degrees of Freedom* (Graus de Liberdade) e são extraídos automaticamente a partir de uma base de dados de movimentos semelhantes aos desejados. Eles mostram que a dimensionalidade de muitas tarefas motoras é menor do que os graus de liberdade completos dos personagens e utilizam DRL para treinar controladores que ativam coordenadamente múltiplas juntas, suportando uma variedade de funções de recompensa para gerar locomoções naturais sem a necessidade de exemplos de captura de movimento. Essa abordagem se alinha à nossa estratégia de focar em recompensas específicas para ações desejadas.

Mais especificamente relacionado a este trabalho, embora já existam trabalhos que usem DRL e que apresentem controle em tempo real de algumas informações específicas, como velocidade desejada (Sousa *et al.*, 2021; Tang *et al.*, 2020; Peng *et al.*, 2021; Won *et al.*, 2022; Li *et al.*, 2023), este trabalho se diferencia ao oferecer uma interface geral e amigável ao usuário, que permite escolher quais informações devem ser consideradas no processo de aprendizado para ajustes de controle em tempo real, sem a necessidade de programação. Continuando a abordagem de Sousa *et al.* (2021), este trabalho propõe a generalização dessa forma de controle em tempo real, alinhando-se à necessidade de ferramentas de animação mais flexíveis e adaptáveis.

O design da função de recompensa é também um aspecto crucial do DRL (Sousa *et al.*, 2021). Enquanto Lillicrap *et al.* (2015) focam na eficácia do algoritmo DDPG em espaços de ação contínuos e sua aplicação em tarefas de física simulada, Haarnoja *et al.* (2018) avançam na compreensão e desenvolvimento de funções de recompensa no DRL através do SAC, influenciando pesquisas subsequentes em animação de personagens. Esses estudos forneceram percepções sobre como diferentes estruturas de recompensa podem impactar o processo de aprendizado e o comportamento final dos personagens animados.

O trabalho de Lillicrap *et al.* (2015) é conhecido por introduzir o algoritmo DDPG *(Deep Deterministic Policy Gradient)*, uma extensão do algoritmo de gradiente de política determinística (DPG) para espaços de ação contínuos, utilizando aprendizado profundo. Esse trabalho trouxe avanços importantes para o campo do DRL, especialmente no controle contínuo, que é um desafio significativo em tarefas de física simulada. Lillicrap *et al.* (2015) introduziram um algoritmo de ator-crítico, sem modelo, baseado em gradiente de política determinística, que pode operar em espaços de ação contínuos, resolvendo de forma robusta mais de 20 tarefas simuladas de física, incluindo problemas clássicos como o balanceamento de pêndulo invertido, manipulação de objetos e locomoção de robôs com múltiplas articulações. Esse trabalho demonstra a eficácia de métodos de aprendizado profundo para o controle contínuo em ambientes simulados, o que é diretamente relevante para a precisão e suavidade das ações dos personagens em animações baseadas em física.

Por outro lado, Haarnoja *et al.* (2018) apresentam o algoritmo *Soft Actor-Critic (SAC)*, que incorpora a maximização da entropia na política de tomada de decisão, influenciando diretamente a estrutura da função de recompensa. A entropia é usada para incentivar a exploração e permitir que a política capture múltiplos modos de comportamento, em vez de se fixar em uma única estratégia.

Em relação ao uso de algoritmos de otimização de políticas, o algoritmo *Proximal Policy Optimization (PPO)* tem se destacado por seu desempenho e simplicidade. Schulman *et al.* (2017) introduziram o PPO como uma nova família de métodos de gradiente de política para aprendizado por reforço, que alternam entre a amostragem de dados através da interação com o ambiente e a otimização de uma função objetivo utilizando ascensão de gradiente estocástico. O PPO combina a eficiência de amostragem do TRPO *(Trust Region Policy Optimization)* com uma implementação mais simples, sendo mais geral e com melhor complexidade de amostra. Experimentos demonstraram que o PPO supera outros métodos de gradiente de política online em tarefas de controle contínuo e jogos Atari, equilibrando favoravelmente entre complexidade de amostra, simplicidade e tempo de execução (Schulman *et al.*, 2017). Este trabalho se alinha com o uso de algoritmos avançados de DRL, onde o PPO é empregado para otimizar as políticas de agentes através da biblioteca *ML-Agents* da *Unity*.

Escontrela *et al.* (2022) propõem substituir funções de recompensa complexas por precursores de movimento adversários *(Adversarial Motion Priors - AMP)*, que são aprendidos a partir de um conjunto de dados de captura de movimento. Eles demonstram que esses precursores podem ser combinados com recompensas de tarefas específicas para treinar políticas que realizam tarefas utilizando estratégias de movimento naturalísticas. Esse método não só simplifica o processo de aprendizado, evitando a necessidade de um ajuste trabalhoso de recompensas manuais, mas também facilita a transferência de políticas treinadas para robôs reais. Por exemplo, eles mostraram que um precursor de estilo eficaz pode ser aprendido a partir de apenas alguns segundos de dados de captura de movimento de um Pastor Alemão, resultando em estratégias de locomoção energeticamente eficientes com transições de marcha naturais.

3.3 Restrições de simetria

Aspectos de simetria têm sido explorados em conjunto com eficiência energética. Yu *et al.* (2018) apresentaram um método para explorar a geração de locomoções simétricas e de baixa energia usando aprendizado por currículo, enquanto Geijtenbeek *et al.* (2013) desenvolveram algoritmos para otimizar a eficiência energética e a simetria na caminhada bípede. Geijtenbeek *et al.* (2013) apresentaram um método de controle baseado em músculos para bípedes simulados, onde tanto o roteamento dos músculos quanto os parâmetros de controle são otimizados. Isso garante que os padrões de torque gerados incorporem restrições biomecânicas, levando a uma locomoção mais natural e eficiente em termos de energia. Além disso,

os controladores sintetizados conseguem encontrar diferentes padrões de marcha, lidar com terrenos irregulares e direcionar-se para alvos específicos, tudo isso sem a necessidade de dados pré-existentes de captura de movimento.

3.4 Equilíbrio artificial

Antes do sucesso do DRL, muitas pesquisas para animação de personagens fisicamente simulados exploraram algoritmos de otimização para desenvolver seus controladores de movimentos. Alguns autores combinaram otimizações com ajustes artificiais, como torques externos e modelagem não real do personagem ou parte dele. Usar esses artifícios para criar controladores requer bastante cuidado quanto a naturalidade da animação. Conforme os personagens simulados se tornam mais sobrenaturais, eles podem se tornar arbitrariamente robustos, mas deve-se ter um cuidado especial para evitar que a animação dos personagens se torne muito artificial visualmente (Silva *et al.*, 2017).

O uso de forças artificiais para aprimorar o controle continua sendo uma abordagem atraente, especialmente quando aplicada de maneira sutil, de modo que o espectador não perceba a artificialidade no resultado. A seguir, são apresentadas algumas obras que utilizaram esses recursos e que inspiraram a criação das Restrições Artificiais deste trabalho, as quais serão combinadas com o controle por DRL.

3.4.1 Hand Of God - HOG

Uma estratégia de ajuste artificial importante é a utilizada por Panne e Lamouret (1995). Eles utilizaram torques externos no tronco do personagem para permitir o aprendizado progressivo de ações de controle para locomoção equilibrada. O mecanismo usado é uma abordagem de *Curriculum Learning* (aprendizagem por currículo), batizada *Hand Of God — HOG* (mão de Deus). Baseado na experiência cotidiana, a ideia é recorrer a uma ajuda externa durante um período para simplificar o problema de aprendizagem. Uma analogia seria um pai ajudando uma criança a aprender a andar ou andar de *skate*, oferecendo uma mão amiga. Assim como essa mão serve para estabilizar o movimento que está sendo aprendido no mundo real, a "mão de Deus" é introduzida para ajudar o personagem a se equilibrar no início de sua corrida ou caminhada. Na prática, a técnica é implementada por meio da aplicação de um torque externo em uma parte específica do personagem para impor uma postura ereta equilibrada durante a

simulação. Este torque corresponde à mão de Deus (HOG), uma ajuda externa (ver Figura 13) realizada no início do processo de aprendizado. Uma vez aprendido o movimento com a ajuda externa, essa ajuda é removida parcial ou totalmente, de forma gradativa.

Figura 13 – A "mão de Deus" opera aplicando um torque τ externo para garantir que o vetor **b**, fixado ao tronco, nunca se desvie significativamente a partir de sua direção ideal dada por **u**.

Fonte: Panne e Lamouret (1995)

O preceito é estruturado em três etapas descritas a seguir:

Etapa 01: No início da simulação, é aplicada a "mão de Deus" para facilitar o equilíbrio enquanto as ações básicas de um movimento desejado são aprendidas. Assim, o problema de equilíbrio é temporariamente (e artificialmente) resolvido, simplificando o processo de encontrar as ações de controle apropriadas para produzir um movimento desejado.

Etapa 02: A segunda etapa busca reduzir a dependência de orientação externa, produzindo um movimento mais equilibrado, mesmo que, em alguns movimentos mais difíceis não seja possível se livrar totalmente da ajuda.

Etapa 03: Uma vez sintetizada uma estratégia básica de controle para caminhada, o suporte externo pode ser subsequentemente reduzido ou eliminado (se possível) por uma otimização adicional, para produzir um movimento livre e equilibrado.

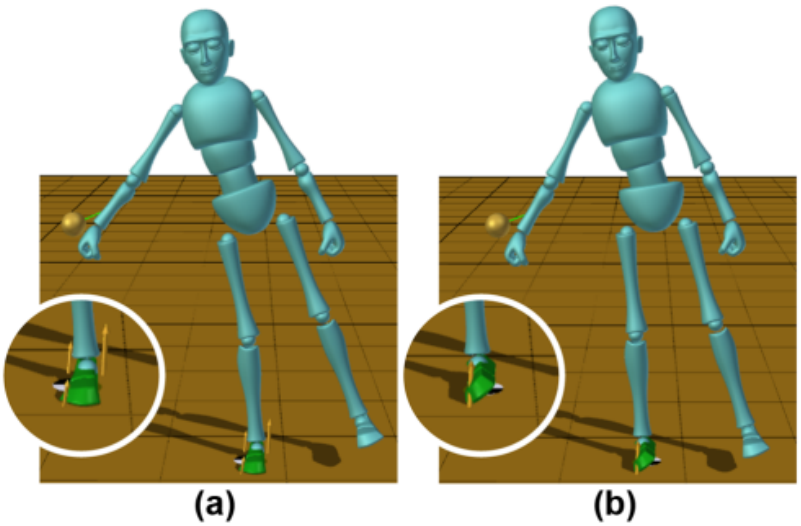
3.4.2 Tunable robustness

Na mesma linha de uso de controle artificial, Silva *et al.* (2017) desenvolveram um controlador, que emprega atuadores virtuais, capaz de seguir movimentos capturados. Eles combinam esse controlador com uma abordagem que chamaram de *topple-free foot — TFF* (pé

sem queda). A artificialidade foi usada através de um modelo simplificado de pé, desacoplando a geometria do pé da interação física do pé com o solo, que permite aplicar a técnica *topple-free foot — TFF*, como ilustra a Figura 14. O TFF é responsável por adicionar artificialmente um torque compensatório ao pé de apoio para fornecer robustez artificial para a estabilidade do personagem, quando desejado. O torque é aplicado somente enquanto o pé toca o chão e a ideia é que ele seja discreto e customizável para equilibrar robustez no equilíbrio e realismo. O TFF dá ao personagem capacidade de equilíbrio arbitrariamente robusta a um custo de possivelmente sacrificar a naturalidade. O modelo de pé foi projetado para ser facilmente ajustável para que um animador possa ter acesso intuitivo a um espectro de soluções, desde uma solução em que nenhuma artificialidade é usada, até uma solução usando um equilíbrio artificial sobre-humano. O usuário pode definir todas as constantes necessárias.

A eficiência do controlador é demonstrada através de uma diversidade de movimentos tais como: caminhar para frente e para trás, agachar e levantar, dar socos e chutes, e jogar capoeira adaptando-se a perturbações externas. A combinação proposta no artigo sacrifica um grau de realismo físico de uma forma ajustável, em troca de um controle rápido e robusto.

Figura 14 – **(a)** A estratégia de contato simples aumenta a estabilidade entre o pé e o solo, mesmo quando a representação do pé (por exemplo, geometria, suavidade) é inadequada. **(b)** Sem a estratégia proposta, o torque interno aplicado ao pé não é compensado corretamente (sem uso de otimizações complexas).



Fonte: Silva *et al.* (2017)

3.4.3 Dynamo

Ainda em relação ao uso de restrições artificiais, Wrotek *et al.* (2006) combinaram captura de movimentos com simulação física, utilizando uma abordagem que aplica forças e torques artificiais usando coordenadas globais para melhorar a estabilidade do personagem. Nesse contexto, os torques externos são aplicados diretamente a todos os corpos do personagem, ajudando a manter o equilíbrio durante a animação em jogos interativos.

Movimentos capturados são usados para obter as orientações desejadas de controladores PD definidos usando o sistema de coordenadas global como referência. Os torques são calculados usando o sistema de coordenadas global como referência de correção, o que implica em aplicar um torque externo a cada corpo, em contraste a um torque interno aplicado em cada junta, como normalmente é realizado o controle. Essa correção é feita de acordo com algum movimento capturado escolhido.

O controle de movimento dessa forma gera torques motores que minimizam os erros das orientações globais de cada corpo em relação às poses vindas do movimento capturado. Assim, torques externos são aplicados em todas as partes para permitir que os personagens se equilibrem em ambientes dinâmicos. O controlador foi combinado com um mecanismo de mola raiz que pode ser ajustado para permitir que os personagens evitem o controle “super-humano” e caiam em situações desejadas.

Eles alegam que os torques motores definidos no espaço global são mais estáveis e as constantes de rigidez de mola exigem menos ajuste manual do que quando definidos no espaço hierárquico das juntas, permitindo que o personagem mantenha mais facilmente poses robustas e interações dinâmicas. Como resultado, são demonstrados controle de movimento em várias situações, incluindo diferentes formas de locomoção e de interação com objetos e com outros personagens. No entanto, a artificialidade da animação resultante deste trabalho é facilmente percebida devido à grande influência dos torques externos sobre os corpos.

Neste trabalho, assim como em Wrotek *et al.* (2006), usamos o recurso de aplicação do torques externos para reduzir o espaço de busca nos treinamentos com DRL e simplificar o problema de controle da simulação física, mas podemos através da interface escolher em qual corpo vamos aplicar artificialidade, dependendo do tipo de movimento que se deseja obter.

3.4.4 Relação entre equilíbrio e artificialidade

A tabela 1 mostra uma expectativa de estabilidade, relacionada ao equilíbrio, que diferentes graus de artificialidade (influência externa) trariam para a simulação física, consequentemente facilitando o processo de treinamento. Um ponto importante de comparação das 3 formas de equilíbrio artificial mencionadas seria perceber o quanto o próprio DRL conseguiria disfarçar a percepção visual dessa artificialidade nos movimentos aprendidos em cada caso. Outra informação importante seria saber o quão útil é um equilíbrio mais estável, mesmo que artificial, no contexto de DRL. Por exemplo, essa maior estabilidade pode ajudar na definição das recompensas, por facilitar o próprio processo de aprendizado? Ela pode tornar os resultados menos sensíveis a pequenos ajustes nas recompensas usadas? Outra informação interessante seria identificar qual grau de estabilidade seria suficiente para simular um determinado tipo de movimento.

Tabela 1 – Expectativa de estabilidade para os diferentes graus de artificialidade.

Trabalhos	**Artificialidade**	**Estabilidade**
Silva *et al.* (2017)	- Artificial	- Estável
Panne e Lamouret (1995)	+/- Artificial	+/- Estável
Wrotek *et al.* (2006)	+ Artificial	+ Estável

Fonte: Elaboração própria (2024).

3.5 Easy constraints e recompensas condicionais

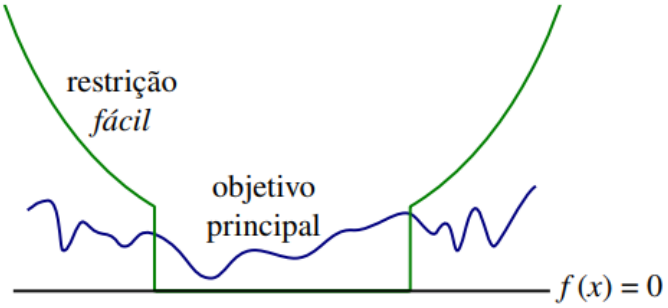
Uma abordagem diferente das anteriores, mas que também serviu de fundamento para este trabalho, é o conceito de *easy constraints* (restrições fáceis), criado por Nunes *et al.* (2012). De maneira geral, nessa pesquisa eles levantam a hipótese de que os sistemas biológicos exploram vibrações naturais na produção de movimentos dinâmicos e que esses modos de vibração são uma base útil para a construção de simulações de locomoção. A partir dessa ideia eles criaram um método que explora as vibrações modais naturais de um personagem fisicamente simulado para fornecer uma paleta de coordenações básicas que os animadores podem usar para montar o movimento desejado. Com o mecanismo, o usuário pode guiar o resultado através da seleção ou inibição de modos de vibração específicos.

Mais relacionado a este trabalho, Nunes *et al.* (2012) também introduziram as chamadas *restrições fáceis*. As restrições fáceis consistem em termos de penalidade adicionados

à função objetivo. Intuitivamente, restrições fáceis representam rampas íngremes de penalidade colocadas sobre as regiões mal comportadas para melhorar o comportamento da função objetivo, ajudando o otimizador a evitá-las e a encontrar as regiões mais comportadas durante a otimização.

Definir uma restrição fácil corresponde a determinar regiões do espaço de busca, supostamente mal comportadas, a serem evitadas, em que a função objetivo apresenta uma maior quantidade de ótimos locais. No caso da função objetivo principal (azul) da Figura 15, pode-se perceber que as regiões à esquerda da primeira linha verde e as à direita da segunda linha verde tendem a ser mais mal comportadas.

Figura 15 – Restrição fácil (verde), ilustrada didaticamente em uma única dimensão, representada através de uma função quadrática condicional bilateral. Restrições fáceis são adicionadas ao objetivo principal (azul) para ajudar o otimizador a encontrar o mínimo global.



Fonte: Nunes (2012)

Uma versão simplificada das restrições fáceis pode ser representada como:

$$Q_u(d,\varepsilon) = \begin{cases} (\varepsilon - d)^2 + h, & \text{se } d < \varepsilon \\ 0, & \text{caso contrário,} \end{cases} \tag{3.1}$$

onde h corresponde à altura de cada restrição fácil, d corresponde à grandeza que se pretende restringir e ε é a tolerância escolhida.

Neste trabalho, a ideia de *easy constraints* serviu de inspiração para a criação das chamadas recompensas condicionais no contexto de DRL, como descritas na Seção 4.5. A semelhança consiste em usar expressões para as recompensas de modo a permitir que elas sejam eventualmente satisfeitas, evitando a concorrência entre os diferentes objetivos (termos). Na Equação 3.1, essa característica é explorada através da descontinuidade propositalmente modelada ao usar uma função condicional.

3.6 Plataforma de desenvolvimento

Quanto à plataforma de desenvolvimento, Juliani *et al.* (2020) destacam a flexibilidade da plataforma Unity para criar agentes inteligentes, possibilitando a aplicação de algoritmos de aprendizado por reforço profundo em simulações utilizando o motor de jogos *Unity* 3D, facilitando assim a integração de DRL em cenários complexos e interativos. Juliani *et al.* (2020) fornece uma visão geral dos ambientes de simulação para aprendizado de máquina, apresentando o *ML-Agents da Unity* como uma plataforma visual, flexível, interativa e facilmente configurável. O artigo também apresenta alguns ambientes de exemplo, como *walker*, *crawler*, *reacher* e *worm*, que podem servir de referência e ponto de partida para pesquisas na área. Este trabalho também utiliza a Unity e o kit de ferramentas *ML-Agents*.

Outros trabalhos também utilizaram o *ML-Agents* para animação, como Booth e Booth (2019), que criou um conjunto de *benchmarks* de código aberto para tarefas de controle contínuo envolvendo locomoção de personagens, permitindo a reprodução de pesquisas avançadas de controle contínuo, como as de Peng *et al.* (2018), Heess *et al.* (2017), entre outras. No mesmo trabalho, são apresentadas estratégias para reduzir o tempo de treinamento dos agentes.

No trabalho de Booth e Ivanov (2020), o *ML-Agents* foi usado para criar um controle realista de personagens utilizando animações padrão do Unity e interação com entradas de usuário. Enquanto Krupnik *et al.* (2020) utilizaram o *ML-Agents* para o treinamento de agentes que possam interagir de forma cooperativa ou competitiva.

3.7 Resumo dos trabalhos relacionados

A tabela a seguir relaciona alguns artigos que, dentro do contexto proposto neste trabalho, possuem aspectos importantes, seja através do uso de aprendizado por reforço profundo, controle em tempo real, restrições artificiais, recompensas condicionais, restrições de simetria ou captura de movimentos para animação de personagens articulados fisicamente simulados. A forma como cada trabalho implementa essas técnicas pode variar, mas eles serviram de inspiração para o desenvolvimento dos nossos métodos. Para mais detalhes sobre a implementação dessas técnicas em cada trabalho, recomenda-se a leitura cuidadosa do respectivo texto neste capítulo. Os números sobrescritos em cada referência do quadro permitem que o leitor seja direcionado diretamente à parte dos trabalhos relacionados que discute o método do artigo. Caso ainda seja necessário um entendimento mais profundo, é interessante a leitura integral dos artigos

mencionados.

Tabela 2 – Comparação com os principais trabalhos relacionados.

Trabalhos	DRL	CTR	RC	RS	RA	Mocap
Juliani *et al.* (2020)[3.6]	Sim	—-	—-	—-	—-	—-
Sousa *et al.* (2021)[3.2]	Sim	Sim	—-	—-	—-	—-
Heess *et al.* (2017)[3.2]	Sim	—-	—-	—-	—-	—-
Yu *et al.* (2018)[3.3]	Sim	—-	—-	Sim	—-	—-
Bergamin *et al.* (2019)[3.1]	Sim	Sim	—-	—-	—-	Sim
Geijtenbeek *et al.* (2013)[3.3]	—-	Sim	—-	Sim	—-	—-
Panne e Lamouret (1995)[3.4.1]	—-	—-	—-	—-	Sim	—-
Silva *et al.* (2017)[3.4.2]	—-	Sim	—-	—-	Sim	Sim
Wrotek *et al.* (2006)[3.4.3]	—-	—-	—-	—-	Sim	Sim
Nunes *et al.* (2012)[3.5]	—-	—-	Sim	—-	—-	—-
Este trabalho	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**—-**

Fonte: Elaboração própria (2024).

Legenda:

- **DRL:** Deep Reinforcement Learning (Aprendizado por reforço profundo)
- **CTR:** Controle em Tempo Real
- **RC:** Recompensas Condicionais
- **RS:** Restrições de Simetria
- **RA:** Restrições Artificiais
- **Mocap:** Motion Capture (Captura de Movimento)

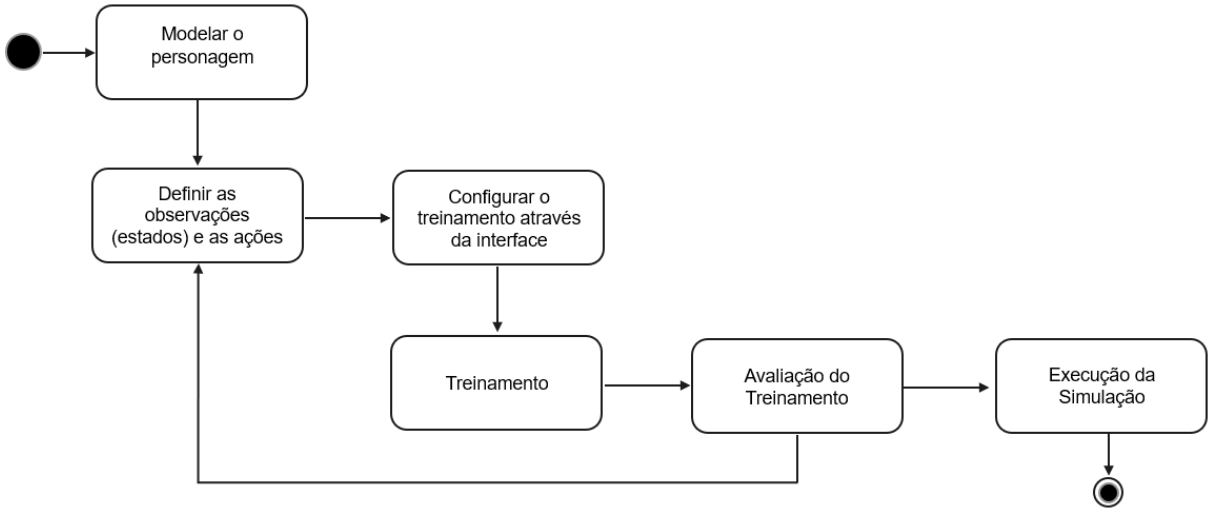
4 ESTRUTURA DE CONTROLE

Neste capítulo, são apresentadas as etapas necessárias para realizar os experimentos desta pesquisa, começando com uma visão geral do processo de animação e da estrutura de controle por DRL de um personagem articulado. Em seguida, detalham-se as especificações do personagem utilizado e os componentes que correspondem às principais contribuições deste trabalho: Generalização do Controle em Tempo Real, Recompensas Condicionais, Restrições de Simetria, Interface de Usuário e Restrições Artificiais, com foco na generalização da estrutura de controle para facilitar tanto o treinamento quanto o controle em tempo real dos personagens.

4.1 Etapas para a realização dos experimentos

Seguindo os fundamentos apresentados na subseção 2.5.1, o diagrama de atividades da Figura 16, apresenta os passos para a criação de um ambiente de treinamento de aprendizado por reforço profundo para personagens articulados fisicamente simulados, utilizando a *Unity* e o *ML-Agents*.

Figura 16 – Fluxo de atividades para o treinamento do agente usando a Unity e o ML-Agents



Fonte: Elaborado pelo autor (2024).

4.1.1 Modelar o personagem

Os personagens são modelados utilizando corpos rígidos conectados por juntas. São utilizadas formas geométricas simples (cápsulas, cubos e esferas) para facilitar o tratamento de colisões. As juntas utilizadas nos exemplos apresentados neste trabalho são de dois tipos: juntas

dobradiças, que possuem um grau de liberdade, e juntas esféricas, que possuem três graus de liberdade. Cada junta apresenta limites angulares inferior e superior para simular a amplitude do movimento real. Os controladores PD já estão disponíveis nas juntas, não sendo necessária sua implementação. São adicionados às partes que formam o personagem os componentes físicos que permitem que os corpos rígidos sejam influenciados pela simulação. Para que os personagens sejam capazes de serem utilizados como agentes de aprendizado por reforço, se faz necessário adicionar um *script* que herda da classe *Agent*.

4.1.2 Definir as observações (estados) e as ações

As observações (estados) são as informações necessárias para que o agente consiga realizar suas tarefas. Essas informações correspondem às entradas da rede neural. São utilizados dados de posição, orientação, velocidade, velocidade angular e toque (valor discreto informando se o corpo está tocando o chão) para cada parte do personagem. Outras informações podem ser utilizadas caso se deseje controlá-las em tempo real, tais como velocidade e direção desejadas do personagem. As ações são o resultado do uso da política, que mapeia um estado em uma ação adequada. No início do treinamento, essas ações são aleatórias e na medida em que o aprendizado vai ocorrendo, elas vão ficando melhores. As ações geradas são os ângulos desejados de cada grau de liberdade que são utilizados pelos controladores PD para gerar os torques. Outras informações de saída podem ser usadas, tais como o limite de força máxima que pode ser aplicada a cada junta pelo controlador PD.

4.1.3 Configurar o treinamento através da interface

Em tarefas de aprendizado por reforço, a recompensa avalia o quão boa ou ruim foi uma ação executada, guiando assim o processo de aprendizado. Tradicionalmente, a criação da função de recompensa envolve a formulação manual de uma expressão matemática e depois codificada via programação para alcançar um resultado desejado. Contudo, a interface desenvolvida automatiza este processo, permitindo que a função de recompensa seja definida automaticamente com base nas informações fornecidas pelo usuário.

Após a escolha do movimento que o personagem deve realizar, a interface deve ser preenchida com os termos que possibilita tal objetivo. Com base nas recompensas condicionais propostas, a função de recompensa é gerada automaticamente, alimentada pelos termos definidos na própria interface, como descrito nas Seções 4.4, 4.5, e 4.7. Essa automação reduz a necessidade

de tentativa e erro na formulação da recompensa, simplificando o processo e garantindo maior consistência nos resultados.

4.1.4 Treinamento

A definição do algoritmo que será utilizado para o treinamento e a configuração dos hiper-parâmetros do algoritmo e da rede neural são feitas em um arquivo separado, que é acessado no início do treinamento. As configurações dos parâmetros da simulação física são feitas na própria Unity, no componente Physics, sendo possível definir, por exemplo, o tamanho do passo de tempo usado na simulação física ou a aceleração da gravidade. Para possibilitar um treinamento mais rápido, vários agentes são treinados em paralelo.

As escolhas das configurações e dos hiper-parâmetros dependem dos agentes a serem treinados, da complexidade do treinamento, e do algoritmo utilizado *(Proximal Policy Optimization (PPO)* ou *Soft Actor-Critic (SAC)*). Nos experimentos deste trabalho a maioria deles foram realizados usando as configurações padrões disponíveis pela biblioteca *ML-Agents*.

Ao iniciar o treinamento, um processo é criado fora do ambiente da *Unity* através do *Python trainers*, um pacote que contém todos os algoritmos DRL utilizados pelo ML-Agents para o treinamento dos agentes. Os algoritmos de aprendizado por reforço são implementados em Python, utilizando inicialmente a biblioteca *TensorFlow* nas primeiras versões do ML-Agents, e nas versões mais atuais, utilizando a biblioteca *PyTorch*. O pacote expõe um único utilitário de linha de comando, chamado mlagents-learn, que oferece suporte a todos os métodos e opções de treinamento. Este utilitário permite definir parâmetros de treinamento e a localização do arquivo de configurações, onde estão as definições dos hiperparâmetros a serem usados durante o treinamento. As sessões de treinamento podem ser feitas tanto utilizando um arquivo executável do ambiente quanto diretamente no ambiente da *Unity*.

O processo de treinamento para a geração de animações de personagens fisicamente simulados é demorado, podendo levar cerca de 9 horas, como observado neste trabalho. Em outros experimentos, como em (Peng *et al.*, 2018), o treinamento pode durar dias. No entanto, pesquisas mais recentes, como Ren *et al.* (2023), conseguiram reduzir significativamente o tempo de treinamento.

4.1.5 Avaliação do treinamento

Além dos aspectos visuais, que podem ser observados acompanhando a evolução do treinamento através da *Unity* ou do executável da simulação, algumas informações mais objetivas também podem ser usadas para a avaliação do treinamento. A cada determinada quantidade de passos do treinamento, são geradas estatísticas que permitem acompanhar a evolução do aprendizado. Exemplos de informações úteis utilizadas são a recompensa acumulada, que deve crescer na medida em que o treinamento progride, e a entropia, que tende a diminuir durante o processo. Essas estatísticas são acompanhadas no utilitário do *Tensorflow* chamado *Tensorboard*.

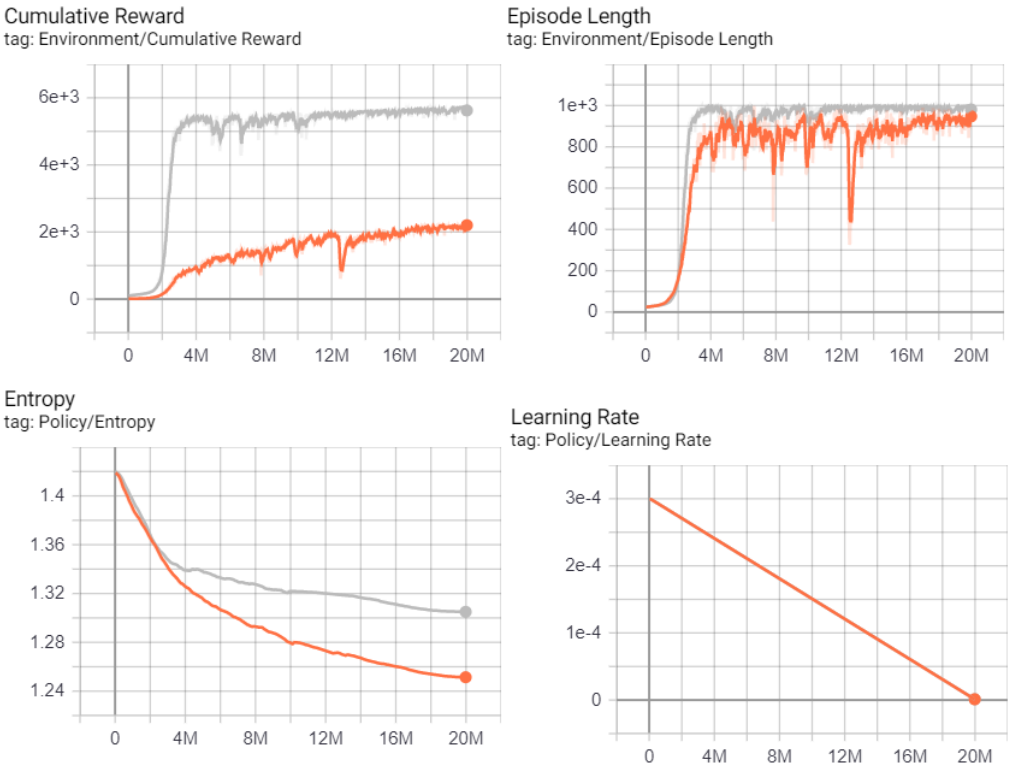
A Figura 17 ilustra a visualização, fornecida pelo *Tensorboard*, de quatro parâmetros monitorados durante o treinamento do agente: recompensa acumulada, tamanho dos episódios, entropia e taxa de aprendizado. Esses gráficos são atualizados em tempo real, permitindo um acompanhamento detalhado da evolução do aprendizado e facilitando ajustes no treinamento para melhorar o desempenho do modelo.

- Recompensa Acumulada (*Cumulative Reward*): Esse gráfico mostra a soma total das recompensas recebidas pelo agente ao longo do tempo. Ele reflete a eficiência do agente em atingir os objetivos definidos no ambiente de treinamento. Um aumento consistente da recompensa acumulada indica que o agente está aprendendo a maximizar a performance. No gráfico, observa-se uma estabilização da curva, o que sugere que o agente atingiu um ponto de convergência.
- Comprimento dos Episódios (*Episode Length*): Esse parâmetro indica a duração média de cada episódio ao longo do treinamento. Em muitos casos, episódios mais longos podem sugerir que o agente está sobrevivendo por mais tempo ou atingindo um comportamento mais eficiente. A estabilização do comprimento dos episódios demonstra que o agente adquiriu um comportamento consistente e repetitivo.
- Entropia (*Entropy*): A entropia mede a aleatoriedade das ações do agente. Valores mais altos indicam que o agente ainda está explorando diferentes ações e aprendendo sobre o ambiente, enquanto valores mais baixos sugerem que o agente se tornou mais seguro das ações que deve executar. Uma redução gradual na entropia ao longo do tempo é esperada e desejável, pois mostra que o agente está convergindo para um comportamento otimizado e previsível
- Taxa de Aprendizado (*Learning Rate*): Esse gráfico mostra a evolução da taxa de aprendizado do agente durante o treinamento. Em alguns métodos, a taxa de aprendizado é

ajustada dinamicamente para permitir um aprendizado mais estável. Normalmente, a taxa de aprendizado é reduzida conforme o treinamento avança para evitar grandes oscilações e garantir a convergência do modelo.

Essas métricas permitem identificar rapidamente se o agente está progredindo conforme o esperado ou se é necessário realizar ajustes nos hiperparâmetros ou até mesmo encerrar o treinamento antecipadamente para revisar o processo.

Figura 17 – Gráficos do *Tensorboard* correspondentes a dois treinamentos distintos. Um deles representado na cor cinza e um outro na cor laranja.



Fonte: Elaborado pelo autor (2024).

4.1.6 Execução da simulação

Após o término do treinamento, é gerado um arquivo contendo a política aprendida. Esse arquivo gerado pode ser executado tanto no ambiente de simulação na *Unity* quanto no executável da cena, sendo anexado aos agentes para que possam executar em modo inferência. Nesse modo, o agente utiliza a política aprendida para executar suas ações dentro da simulação. Mesmo com os dados do treinamento sendo acompanhados no gráfico do *Tensorboard* e evoluindo de acordo com o esperado, o resultado da animação gerada pode não ser a desejada, sendo necessário recomeçar os passos. Uma aplicação útil para as redes que são treinadas com

sucesso é usá-las na animação de Non-Player Character (NPC) em jogos ou em ambientes de simulação em geral.

4.2 Visão geral do controle com DRL

No processo de animação usando DRL, o controlador do personagem é representado por uma rede neural, que é projetada para mapear a relação entre as informações sensoriais do personagem e seus atuadores. Inicializada com pesos de conexão aleatórios, a rede neural passa por um processo de treinamento que envolve iterativamente os seguintes passos resumidos:

1. As entradas sensoriais são alimentadas na rede.
2. A rede emite sinais de controle para os atuadores do personagem.
3. O ambiente de simulação atualiza o estado do personagem com base nas saídas.
4. As recompensas são calculadas com base no desempenho do personagem e nos resultados desejados.
5. Os pesos da rede são atualizados usando um algoritmo de DRL (por exemplo, Proximal Policy Optimization, PPO).

As recompensas são definidas pelo animador e representadas por uma função que deve resultar em um único valor e guiar a rede neural durante o aprendizado para produzir o movimento desejado. De acordo com essa função, o espaço de estados e ações é explorado através da execução exaustiva de várias tentativas de simulação, chamadas de episódios, e para cada estado, é avaliado se uma determinada ação contribui para aumentar o valor da função de valor único.

A informação de recompensa local de cada estado é calculada e combinada com as recompensas dos estados vizinhos, de modo que caminhos compostos por sequências de estados e ações que tenham a maior acumulação de recompensas possam ser identificados. O cálculo dessa acumulação é determinado pela soma das recompensas, que é definida pela equação:

$$R_t = \sum_{i=0}^{T} \gamma^i r_{t+i}, \tag{4.1}$$

onde T é o número total de amostras em um episódio, $\gamma \in (0,1]$ é o fator de desconto, e t representa o tempo. Assim, a rede neural, que representa a política π no contexto de DRL, tem seus pesos atualizados com base em R_t. Durante o processo de treinamento, a política π é continuamente atualizada para mapear cada estado para uma ação que maximize o valor de

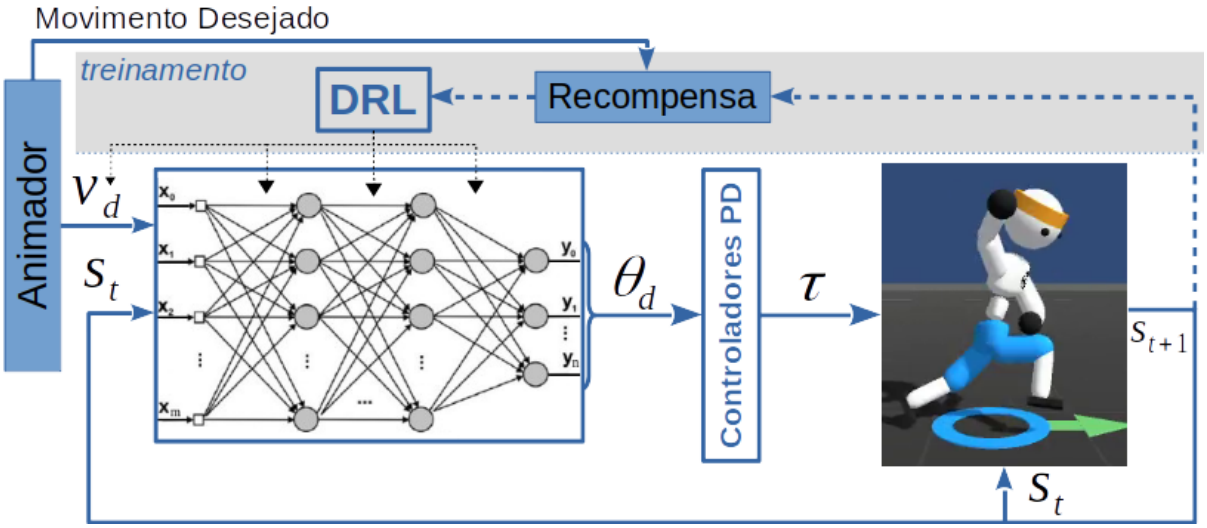
R_t. O objetivo é alcançar a política ótima, $\pi^* = \arg\max_a \mathbb{E}[R|\pi]$, onde as ações geram a maior recompensa acumulada.

Explorar todo o espaço de estado-ação não é possível; portanto, métodos de DRL tentam identificar regiões do espaço de busca onde os melhores caminhos podem ser encontrados. A atualização dos pesos das conexões da rede neural corresponde à "memorização" desses caminhos que acumulam os valores de recompensa mais altos para os estados visitados através das melhores ações escolhidas. Uma vez treinada, a rede pode ser usada em tempo real para decidir rapidamente qual a melhor ação local a tomar a partir de cada estado atual, embora cada ação já tenha sido analisada em um contexto global, dentro de vários possíveis caminhos (episódios) explorados durante o treinamento.

4.2.1 Visão geral da estrutura de controle de um personagem articulado

A Figura 18 apresenta a visão geral do funcionamento do controlador, que é representado por uma rede neural treinada usando DRL e que conduz a simulação física do personagem para gerar a animação resultante.

Figura 18 – Visão geral da estrutura de controle e do treinamento da rede usando DRL.



Fonte: Sousa *et al.* (2021)

A simulação física consiste em, a partir de um estado do sistema s_t, definido em um determinado instante t, integrar as equações diferenciais de movimento da estrutura modelada para obter um novo estado s_{t+1} correspondente ao próximo instante de tempo $t + \delta t$ considerado. Entretanto, apenas a simulação física não é suficiente para gerar animação natural para sistemas ativos. Se nenhuma atuação interna for exercida por parte do personagem, ele se comporta como

um sistema passivo, que obedece as regras físicas definidas mas não se movimenta naturalmente como um humano real. No caso, o personagem agiria como uma boneca de pano (*ragdoll*) e apenas cairia sob o efeito da gravidade. O problema de controle consiste em definir os torques τ a serem aplicados em cada instante da simulação física pelos atuadores localizados nas articulações do personagem. Esses torques aplicados são automaticamente adicionados às equações de movimento e influenciam no cálculo de cada novo estado s_{t+1}.

No contexto deste trabalho, uma rede neural é responsável por gerar os ângulos desejados para os Controladores Proporcional-Derivativo (PD), que geram os torques necessários para manter o movimento do personagem e funcionam como uma camada de controle de mais baixo nível, responsável por calcular os torques capazes de atingir as orientações desejadas fornecidas pela rede.

Para funcionar de maneira adequada, a rede precisa ser treinada e uma função de recompensa definida pelo animador avalia a qualidade das ações executadas de acordo com um tipo de movimento desejado escolhido. Os movimentos não são especificados em detalhes. Movimentos adequados, de acordo com as características do movimento definido através da recompensa, emergem naturalmente usando DRL. O método de aprendizado da rede basicamente explora o espaço de estados e ações do sistema, através da execução exaustiva de várias tentativas de simulação, chamados de episódios, e avalia para cada estado se uma determinada ação contribui para aumentar o valor da função de recompensa. Essas informações locais de recompensa em cada estado vão sendo computadas e combinadas com as recompensas dos estados vizinhos de modo que os caminhos compostos por sequências de estados e ações ([estado] $\rightarrow$ [ação tomada] $\rightarrow$ [estado alcançado devido à respectiva ação tomada] $\rightarrow$ [nova ação] $\rightarrow \ldots$) que possuem o maior acúmulo de recompensas possam ser identificados.

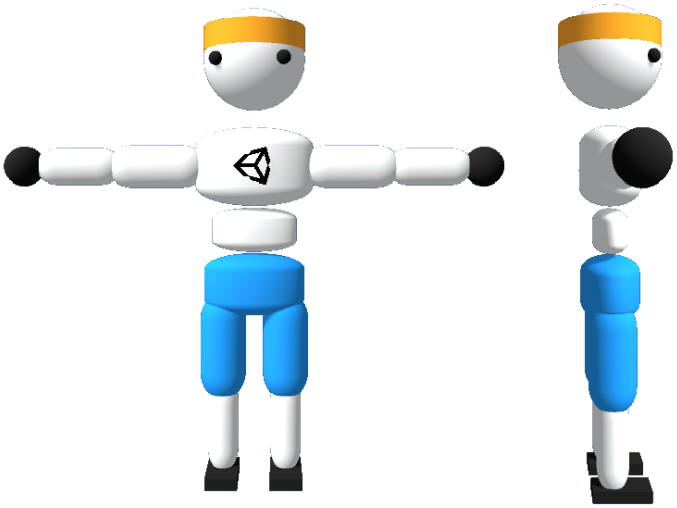
Após o processo de aprendizagem, durante a simulação em tempo real, a rede treinada é responsável por decidir qual melhor ação local θ_d (saída da rede) tomar a partir do estado atual do sistema s_t (entrada da rede), embora cada ação já tenha sido analisada num contexto global, dentro de vários possíveis caminhos explorados (episódios) durante o treinamento.

4.3 Personagem humanoide articulado

Os experimentos foram realizados com um personagem (agente) humanoide 3D articulado baseado em física, composto de 16 corpos rígidos e 28 graus de liberdade. Esses DoFs correspondem às articulações das seguintes partes do corpo: quadril, tórax, coluna, cabeça, coxas,

canelas, pés, braços e antebraços, como pode ser observado na Figura 19. O quadril corresponde ao corpo raiz e os outros corpos são conectados através de 13 juntas do tipo *Configurable Joint* da *Unity*. Os pulsos são fixos.

Figura 19 – Personagem humanoide articulado.



Fonte: Unity (2020)

Esse modelo corresponde ao mesmo utilizado no ambiente exemplo disponibilizado com o ML-Agents chamado *Walker*.

A rede neural usada nos experimentos possui 3 camadas ocultas, contendo 512 nós cada uma. Na configuração inicial do Humano 3D a saída da rede corresponde a 39 valores, mas para cada experimento, dependendo do objetivo este valor pode precisar ser alterado para mais ou menos, os valores da saída são referentes aos ângulos desejados dos 28 graus de liberdade das articulações e aos limites máximos de torque que os controladores PD podem aplicar em cada uma das 13 articulações.

A quantidade de entradas da rede neural para o personagem em questão é inicialmente configurada com 243 valores referentes a posição, orientação, velocidade linear, velocidade angular e contato com o chão (informação discreta) de cada corpo rígido. Essa quantidade de entradas também pode precisar ser ajustada para cada experimento, se por exemplo, for controlar em tempo real apenas uma informação de valor escalar, é importante adicionar pelo menos 2 novas entradas, para que a rede possa receber o valor alvo e valor atual da informação, mas se for controlar uma posição 3D, pode ser importante adicionar uma nova entrada para o valor de cada coordenada, tanto para o valor alvo como para o valor atual. A solução proposta adiciona automaticamente a quantidade de entrada adequada de acordo com a configuração do

experimento na interface.

4.4 Generalização do controle em tempo real

A generalização do controle em tempo real envolve permitir que o animador escolha qualquer informação do personagem para ajustá-la em tempo real durante a execução da rede já treinada. A generalização também implica que mais de uma informação pode ser controlada em tempo real.

4.4.1 Controle em tempo real de uma informação específica

Primeiramente, para que uma informação específica seja controlada no contexto de DRL, uma recompensa corrigindo o erro entre seus valores atual, capturado da simulação física, e desejado deve ser adicionada ao processo de aprendizado. Entretanto, ao usar um valor desejado fixo durante todo o treinamento, o animador não tem mais controle sobre a informação escolhida depois da rede treinada, pois sempre o mesmo valor vai ser buscado. Por exemplo, ao realizar o treinamento tentando atingir sempre uma velocidade de 5 m/s, a rede sempre buscará essa mesma velocidade depois de treinada. No caso, o animador não poderia mais escolher uma velocidade desejada diferente.Assim, a rede deve ser treinada explorando diferentes valores desejados, para que ela esteja preparada para permitir que o animador controle a informação mesmo depois do treinamento.

Além disso, a rede precisa armazenar esses diferentes valores desejados explorados, e essa "memorização" é realizada através das conexões adicionais associadas a uma entrada extra que deve ser acrescentada à rede. Durante o treinamento, essa entrada extra é então alimentada com os diferentes valores, gerados automaticamente dentro de um intervalo de possíveis valores desejados. Depois de escolhidos os limites do intervalo, uma opção, por exemplo, é sortear um novo valor desejado apenas no início de cada episódio. Nesse caso, ele é mantido fixo até que um novo episódio inicie. Outra alternativa é atualizar o valor desejado continuamente, percorrendo todo o intervalo no decorrer do episódio.

Assim, uma vez que a rede esteja treinada, durante a execução da simulação em tempo real, o animador pode controlar a informação escolhida simplesmente modificando a entrada correspondente da rede. A interface disponibiliza um *slider* para que o animador escolha cada novo valor que ele queira que a informação atinja, e esse valor é diretamente transferido

para a entrada da rede.

Além do valor desejado, é recomendado acrescentar também o valor atual da informação como entrada da rede. Portanto, cada informação sendo controlada em tempo real exige a adição de uma nova recompensa corrigindo o erro entre seus valores atual e desejado e a adição de duas entradas extras na rede. Além disso, um intervalo de possíveis valores desejados e um procedimento para explorar diferentes valores desejados dentro desse intervalo, durante o treinamento, também são necessários.

4.4.2 Generalização do controle em tempo real de mais de uma informação

Entendido como é realizado o controle de uma informação específica, deseja-se estender o controle para qualquer informação que o usuário precise. Um dos desafios da generalização é obter os valores atuais dessas informações, necessários para o cálculo e correção dos erros em relação aos valores desejados. Para isso, a generalização do controle em tempo real proposta exige:

- um *label* para cada informação possível de ser controlada; e
- um método, que recebe um dos possíveis *labels* como entrada e retorna o valor atual da respectiva informação.

O método então utiliza uma estrutura condicional switch para tratar os diferentes cases de acordo com cada *label* registrado em um *enum*. Por exemplo, considerando que a altura da mão esquerda seja uma das possíveis informações a ser escolhida, um *label* correspondente (e.g, LeftHandHeight) deve ser registrado no *enum*, o método deve ter um *case* para acessar e retornar a coordenada y da posição do objeto correspondente à mão esquerda do personagem. Um protótipo da implementação do método pode ser observado na Figura 20

Caso o animador não queira usar as informações já disponíveis, as únicas partes do código que precisam ser modificadas são o *enum*, adicionando um *label* extra associado à nova informação, e o *switch*, adicionando um *case* extra para mostrar como a nova informação, ao ser escolhida, deve ser acessada na simulação física. Fora isso, toda a especificação do treinamento é feita diretamente na interface. Note que este método apenas se preocupa em acessar o valor atual da informação (e.g., altura da mão esquerda). A influência desse valor no cálculo da recompensa (Subseção 4.5.1) é definida automaticamente e pode ser especificada diretamente através da interface, sem intervenções adicionais no código.

Para que mais de uma informação possa ser controlada em tempo real, a interface

Figura 20 – Enum InfoLabel e método GetInfoValue.

```
// Enum para registrar rótulos das possíveis informações a serem controladas
public enum InfoLabel
{
    LeftHandHeight,
    // Adicione mais rótulos, conforme necessário
}
// Método para retornar o valor atual do rótulo da informação respectiva
public float GetInfoValue(InfoLabel label)
{
    switch (label)
    {
        case InfoLabel.LeftHandHeight:
            // Acesse e retorne a coordenada y da posição da mão esquerda
            return GameObject.Find("LeftHand").transform.position.y;
        // Adicione mais casos conforme necessário
        default:
            return 0f;
    }
}
```

Fonte: Elaborado pelo autor (2024).

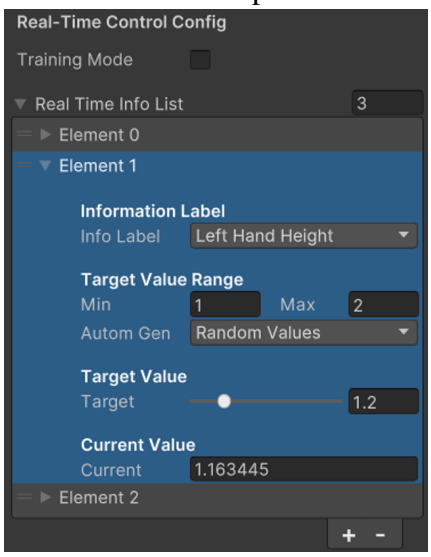
permite que o usuário especifique as informações selecionadas no formato de uma lista, podendo adicionar, editar ou excluir itens. Como ilustrado na Figura 21, para o treinamento, o animador deve especificar para cada item:

- o label correspondente à informação selecionada;
- os limites mínimo e máximo do intervalo de possíveis valores desejados; e
- a forma de geração automática, durante o treinamento, dos diferentes valores desejados dentro desse intervalo.

Dois métodos de geração automática estão disponíveis: sorteio no início de cada episódio (*RandomValues*) ou passando continuamente por todo o intervalo durante o episódio (*GoThroughRange*). Após treinar a rede, o animador pode escolher o valor desejado através de um controle deslizante (*slider*) e verificar a funcionalidade de controle através da atualização e exibição do valor atual.

Percorrendo a lista de informações *RealTimeInfoList*, a partir do *label* de cada item, a respectiva recompensa de correção do erro e as duas entradas da rede já podem ser definidas automaticamente. As entradas da rede são definidas de maneira mais fácil, simplesmente adicionando os valores desejado e atual de cada item ao vetor de sensores que alimenta a rede neural. A rede é que fica responsável por definir como essas novas entradas devem influenciar suas saídas, e consequentemente o personagem. Já as recompensas podem ser definidas através de diferentes fórmulas, e não usam esses dois valores diretamente. Portanto, para que as correções dos erros possam ser definidas automaticamente a partir da interface, uma padronização e generalização na definição das recompensas também é necessária (Subseção 4.5.2).

Figura 21 – Interface usada para controle em tempo real.



Fonte: Elaborado pelo autor (2024).

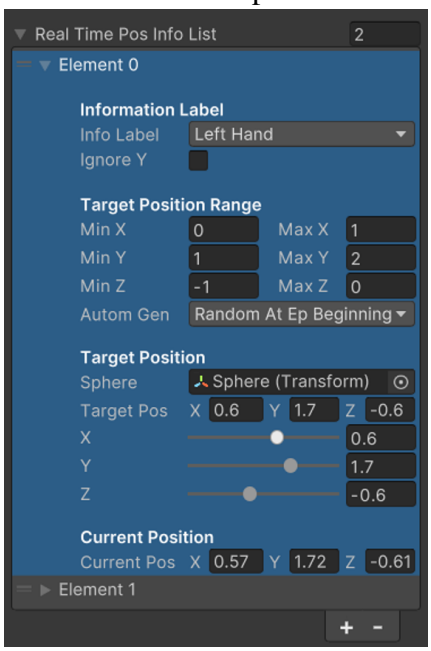
4.4.3 Controle em tempo real de uma posição

A generalização do controle em tempo real descrita acima está completa e já permite o controle de uma posição. Bastaria incluir três itens na *RealTimeInfoList* (Figura 21), um para cada coordenada. Entretanto, agrupar informações de coordenadas de uma posição pode facilitar alguns experimentos. Portanto, a interface proposta também oferece esse controle agrupado por meio de uma lista de posições a serem controladas em tempo real, *RealTimePosInfoList*, Como mostrado na Figura 22, cada item nessa lista possui:

- o *label* correspondente à informação selecionada;
- os limites mínimo e máximo para cada coordenada, representando o intervalo de possíveis posições desejadas; e
- a forma de geração automática, durante o treinamento, das diferentes posições desejadas dentro desse intervalo.

Além da opção de sorteio apenas no início do episódio (*RandomAtEpBeginning*), sorteios também podem ser feitos durante o episódio em instantes uniformemente distribuídos (*RandomDuringEp*). Nesse caso, devem ser escolhidos intervalos menores, que se tornam relativos. Ou seja, o sorteio resulta em um deslocamento a ser adicionado à última posição desejada. Após treinada a rede, o animador pode atualizar a posição desejada através do movimento do mouse na tela. Uma esfera é usada para representá-la visualmente na cena e facilitar conferir o funcionamento do controle. A interface ainda disponibiliza três *sliders* para atualizar cada coordenada individualmente, e mostra a posição atual, também para comparação.

Figura 22 – Interface usada para controle em tempo real.



Fonte: Elaborado pelo autor (2024).

Uma outra possibilidade é controlar apenas uma projeção horizontal da posição, ignorando a coordenada y. Nesse caso, a projeção desejada é visualmente representada apenas por um círculo desenhado em um plano horizontal, e pode ser controlada através do clique do mouse nesse plano.. Assim, a interface fornece três tipos de controle em tempo real: posições 3D, posições horizontais ou qualquer informação geral isolada.

Para calcular a posição atual, um novo método chamado *GetPosInfo* foi usado para obter os valores atuais de qualquer posição. Embora o mesmo *enum InfoLabel* possa ser usado, já que o método *GetInfoValue* retorna apenas um *float*, o *GetPosInfo* foi necessário para retornar posições. Portanto, o novo método é bastante semelhante e consiste basicamente em um local diferente para organizar os *cases*, separando a informação escalar da informação agrupada em posições. Ou seja, os rótulos correspondentes às posições devem ter seus *cases* implementados no *GetPosInfo*. Da mesma forma que na *RealTimeInfoList*, a *RealTimePosInfoList* também é iterada automaticamente para adicionar as posições desejadas e atuais de cada item como entradas da rede.

4.5 Recompensas condicionais

As recompensas definidas pelo animador guiam a rede neural durante o aprendizado para produzir o movimento desejado. Quanto melhor o desempenho do personagem, maior deve ser o valor da recompensa. O Aprendizado por Reforço Profundo (DRL) requer uma função

de valor único, tipicamente definida como uma soma ponderada de termos onde cada termo representa uma dessas recompensas. No entanto, cada termo ainda pode ser definido por meio de diferentes fórmulas.

Para automatizar e generalizar a construção dessas fórmulas matemáticas, este trabalho propõe uma padronização na definição das recompensas, permitindo que sejam definidas através de uma interface. Um aspecto da padronização é que cada recompensa deve estar associada a uma informação específica, visando maximizar ou minimizar essa informação até que uma tolerância definida seja alcançada.

Por exemplo, se o animador quiser maximizar a coordenada x da velocidade do centro de massa (CoM), uma tolerância máxima tol_{max} deve ser definida. Definir $tol_{max} = 5$ significa que a recompensa é satisfeita quando $CoM_{vel_x} \geq 5$. Inspirado pela ideia de que essa condição seja atendida, a padronização proposta introduz as chamadas *recompensas condicionais*. Outro aspecto fundamental é garantir que cada recompensa seja normalizada dentro de um intervalo comum [0,1], permitindo uma comparação coerente entre diferentes recompensas.

Semelhante à informação controlada em tempo real, múltiplas recompensas podem ser configuradas de acordo com o padrão adotado. A interface permite que o usuário adicione, edite ou remova recompensas condicionais de uma lista durante a especificação do treinamento. Conforme ilustrado na Figura 23, para cada recompensa condicional, o animador deve especificar:

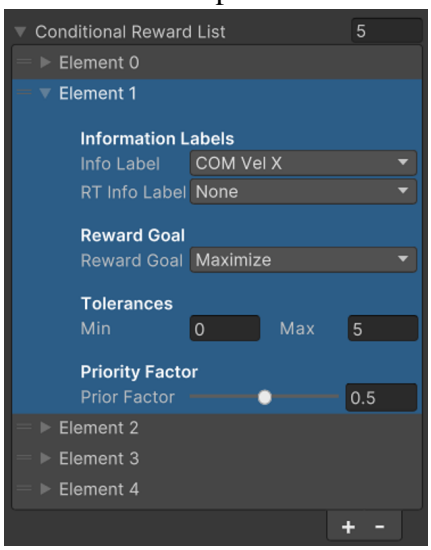
- O rótulo correspondente à informação analisada.
- O objetivo da recompensa, seja maximizar ou minimizar.
- Os limites de tolerância, tanto mínimo quanto máximo.
- Um fator de prioridade.

Iterando pela ConditionalRewardList, esses parâmetros são usados para calcular o termo correspondente a cada recompensa na lista. Os termos são simplesmente somados para obter o resultado da função de valor único usada pelo DRL para avaliar o estado atual do personagem. A cada passo da simulação, a função de valor único é recalculada.

4.5.1 Cálculo da função de valor único

Para cada termo, o valor atual da informação, $info$, é obtido usando o método *GetInfoValue* (Figura 20), com o rótulo correspondente como entrada. Para maximização, quanto mais próximo $info$ estiver de tol_{max}, melhor. Quando tol_{max} é alcançado, a recompensa atinge seu valor mais alto, e exceder esse valor mantém a recompensa em seu valor mais alto, indicando

Figura 23 – Interface usada para definir recompensas condicionais.



Fonte: Elaborado pelo autor (2024).

que a recompensa condicional está satisfeita e a prioridade pode mudar para outras recompensas que precisam de melhoria. Note que as tolerâncias escolhidas devem ser possíveis de alcançar para que as múltiplas recompensas condicionais possam ser satisfeitas juntas, reduzindo o problema de competição e evitando ajustes trabalhosos de seus pesos.

Para normalizar, um tol_{min} também é necessário. Juntamente com tol_{max}, essa faixa delimitada permite mapear $info$ para a faixa normalizada [0,1]. Ao alcançar tol_{min}, a recompensa atinge seu valor mais baixo, 0. Abaixo desse valor, a recompensa permanece em seu valor mais baixo. Defina o delta de tolerância como $\Delta_{tol} = tol_{max} - tol_{min}$. Como $info = tol_{min}$ é mapeado para 0, um novo delta, Δ_{info}, é definido como:

$$\Delta_{info} = clamp(info - tol_{min}, 0, \Delta_{tol}), \tag{4.2}$$

onde a função clamp significa

$$clamp(x, \min, \max) = \begin{cases} \min, & \text{se } x < \min \\ \max, & \text{se } x > \max \\ x, & \text{caso contrário.} \end{cases} \tag{4.3}$$

Note que quando $info = tol_{max}$ ou maior, $\Delta_{info} = \Delta_{tol}$, uma vez que o clamp limita Δ_{info} para não exceder Δ_{tol}. Portanto, dividir Δ_{info} por Δ_{tol} resulta em uma recompensa normalizada entre 0 e 1.

$$r_{norm} = \frac{\Delta_{info}}{\Delta_{tol}} \tag{4.4}$$

Para minimização, funciona de forma inversa, mas a formulação segue uma ideia semelhante. Agora, quanto mais próximo $info$ estiver de tol_{min}, melhor. Como $info = tol_{max}$ é mapeado para 0, Δ_{info} agora é:

$$\Delta_{info} = clamp(tol_{max} - info, 0, \Delta_{tol}). \tag{4.5}$$

Note que a ordem da subtração foi alterada para que os cortes do clamp ainda funcionem. Assim, quando $info = tol_{min}$ ou menor, $\Delta_{info} = \Delta_{tol}$.

A Equação 4.4 e os últimos passos se aplicam de forma semelhante tanto para maximização quanto para minimização.

A recompensa normalizada, r_{norm}, é elevada ao quadrado para melhorar o comportamento da função de otimização para cálculos de derivadas mais úteis. Finalmente, a recompensa normalizada ao quadrado é ponderada pelo fator de prioridade e adicionada à função de valor único:

$$AddReward(p \cdot r_{norm}^2). \tag{4.6}$$

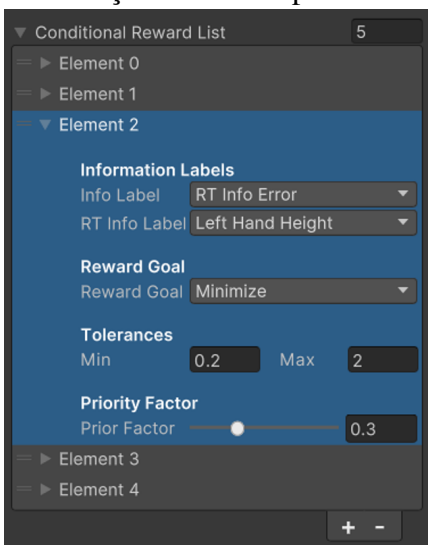
A soma de todos os fatores de prioridade não deve exceder um. À medida que algumas recompensas condicionais são eventualmente satisfeitas, a prioridade automaticamente se desloca entre as recompensas, evitando ajustes difíceis dos fatores.

4.5.2 Recompensas de correção de erro geradas automaticamente

Entendida a padronização das recompensas condicionais, uma recompensa nesse formato deve ser criada para cada informação selecionada para ser controlada em tempo real. Isso é feito automaticamente uma vez que o campo *InfoLabel* de qualquer item adicionado à *RealTimeInfoList* é escolhido (Figura 21). A Figura 24 ilustra uma dessas recompensas criadas e adicionadas automaticamente à *ConditionalRewardList*. Note que ao adicionar essas recompensas à *ConditionalRewardList*, elas já são mostradas na interface como novos itens e elas já entram no cálculo da função de valor único juntamente com outras. Caso o item que gera a recompensa for removido da *RealTimeInfoList*, a recompensa também é automaticamente removida.

Ao ser gerada, o campo *InfoLabel* recebe um *label* geral já registrado chamado *RTInfoError*, para indicar que a recompensa está associada à correção da informação de algum *RTInfoItem* da *RealTimeInfoList*, e um segundo campo *RTInfoLabel* é que recebe o mesmo *label*

Figura 24 – Interface usada para definição das recompensas condicionais.



Fonte: Elaborado pelo autor (2024).

escolhido (e.g., *LeftHandHeight*) do *RTInfoItem* correspondente. Note que *RTInfoLabel* só é usado para recompensas geradas automaticamente e simplesmente ignorado para recompensas definidas diretamente na *ConditionalRewardList*. Como esse tipo de recompensa deve corrigir um erro, o objetivo da recompensa deve ser minimizar e tol_{min} deve ter um valor próximo a 0. tol_{max} indica o tamanho do intervalo que será usado na normalização da recompensa. $tol_{min} = 0.2$, $tol_{max} = 2$ e $p = 0.3$ são definidos automaticamente como mostrado e o animador pode ajustá-los caso necessário.

No cálculo da função de valor único (Subseção 4.5.1), essas recompensas geradas automaticamente não precisam que um *case* diferente seja implementado para cada uma delas. O método *GetInfoValue*, que precisa retornar o erro a ser minimizado, já possui um case implementado para o *label* geral *RTInfoError*, que apenas precisa encontrar na *RealTimeInfoList* o item correspondente, $RTInfoItem$, cujo *InfoLabel* é igual ao *RTInfoLabel* da recompensa. Esse *RTInfoLabel* é transmitido através de uma segunda entrada no método *GetInfoValue* (*rtLabel* mostrado na Figura 20), e uma simples busca encontra o $RTInfoItem$ e acessa seus valores atuais e desejado. A informação retornada no *case* Math.Abs$(RTInfoItem.target - RTInfoItem.current)$ é o erro a ser corrigido.

De mesma forma, assim que o campo *InfoLabel* de qualquer de algum item adicionado à *RealTimePosInfoList* é escolhido (Figura 22), uma recompensa semelhante também é criada e adicionada automaticamente à *ConditionalRewardList*. A diferença é que o campo *InfoLabel* dessa recompensa recebe um outro *label* geral, também já registrado, chamado *RTPosInfoError*. Note também que, no cálculo da função de valor único, o campo *RTInfoLabel* da

recompensa precisa ser buscado na *RealTimePosInfoList* dessa vez. Um *label* diferente do *RTInfoError* é necessário porque, no método *GetInfoValue*, o case correspondente ao controle de uma posição obtém cada operando da diferença $RTPosInfoItem.target - RTPosInfoItem.current$ no formato de um vetor 3D.

Depois de verificar se a coordenada y desse vetor diferença deve ser mantida ou zerada (*IgnoreY*), sua magnitude, representando o erro a ser corrigido, é retornada. Note que zerar a coordenada y do vetor diferença, marcando *IgnoreY*, significa controlar apenas a componente horizontal da posição.

4.6 Restrições de simetria

As restrições de simetria reduzem diretamente o espaço de ações da rede neural com o intuito de evitar movimentos descoordenados indesejados dos membros do personagem e fornecer uma camada adicional de controle.

Por exemplo, em uma corrida tradicional, enquanto uma perna se movimenta para trás do tronco para que o pé tocando o chão possa impulsionar o personagem para frente, a outra perna se movimenta para frente para já preparar a próxima passada. Nesse caso, os movimentos das pernas ocorrem de forma sincronizada e anti-simétrica. Os movimentos dos braços apresentam essa anti-simetria lateral semelhante para compensar o movimento das pernas e ajudar no equilíbrio. A anti-simetria lateral significa que os movimentos dos membros do lado esquerdo e do lado direito do corpo são espelhados inversamente. Já em uma corrida canguru, deseja-se que os membros se movimentem de maneira simétrica.

Assim, em vez de deixar o DRL buscar soluções em um espaço de busca bem mais amplo, visitando estados do personagem em que seus membros não estariam sincronizados da maneira desejada, e que dependeriam de cuidadosas escolhas de recompensas para penalizar e desencorajar o acesso a esses estados assimétricos, o espaço de ações é diretamente reduzido para impedir que essas regiões indesejadas sejam exploradas e evitar que os movimentos com membros descoordenados apareçam nos resultados.

Junto com essa vantagem da própria redução apropriada do espaço de busca, este trabalho propõe uma forma adicional de controle que consiste em permitir que o animador faça essas escolhas de maneira independente para cada parte dos membros, influenciando suas respectivas articulações aos pares. A Figura 25 ilustra como, através da simples marcação de verificações de simetria *S* ou anti-simetria *A* na interface, o aprendizado pode ser orientado

para diferentes tipos de locomoção, resultantes apenas a partir da combinação das restrições de simetria disponíveis.

Figura 25 – Interface usada para selecionar restrições de simetria

Fonte: Elaborado pelo autor (2024).

Para implementar essa redução da saída da rede, um lado do personagem deve ser escolhido como referência, o esquerdo por exemplo. Uma vez que o usuário escolha que os ombros (*Arms*) devem se movimentar de maneira simétrica por exemplo, as três saídas da rede responsáveis por definir o torque a ser aplicado no ombro direito são simplesmente excluídas e os mesmos três valores de saída do ombro esquerdo passam a ser também aplicados no direito, sendo necessário ajustar apenas os sinais dos valores para definir se o espelhamento deve ser simétrico ou anti-simétrico. Ao desmarcar os dois *checks*, *S* e *A*, correspondentes aos ombros (*Arms*) na interface, as três saídas do ombro direito voltam a existir e os dois ombros voltam a ser controlados de maneira independente. Isso deve ser escolhido como parte da especificação do treinamento, antes do processo de aprendizado.

4.7 Interface do usuário

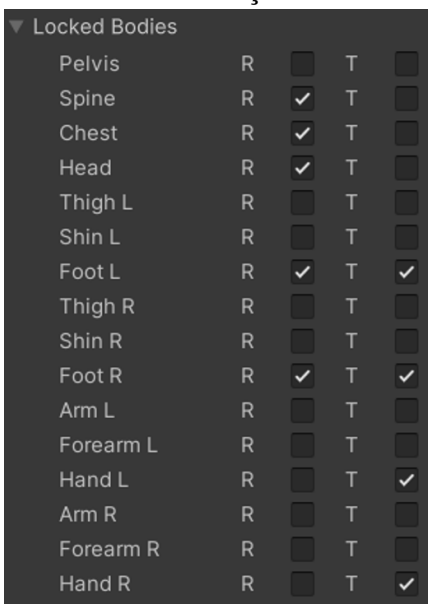
Uma interface do usuário foi desenvolvida para demonstrar as generalizações propostas, facilitando a especificação do treinamento, sem a necessidade de edição de *scripts*, e o controle em tempo real das informações escolhidas. A interface permite que os usuários ajustem recompensas, simetrias e informações controladas em tempo real de maneira intuitiva e organizada. A descrição do design e das principais funcionalidades da interface foi apresentada em conjunto com as contribuições, destacando como ela integra-se ao sistema de controle. A Seção 5 exemplifica casos de uso que mostram sua utilidade e praticidade.

4.7.1 Restrições Artificiais

Além das generalizações de controle, recompensas e simetria, já explicadas nas subseções anteriores, a interface de usuário também permite a aplicação de restrições artificiais para facilitar a realização de determinados experimentos. Essas restrições ajudam a tornar os experimentos mais estáveis, no intuito de obter resultados mais consistentes e bem comportados.

A Figura 26 ilustra como os vários corpos do personagem podem ter seus movimentos de rotação R ou de translação T travados ou liberados, ao marcar ou desmarcar, respectivamente, os *checks* correspondentes.

Figura 26 – Interface usada para selecionar restrições artificiais.



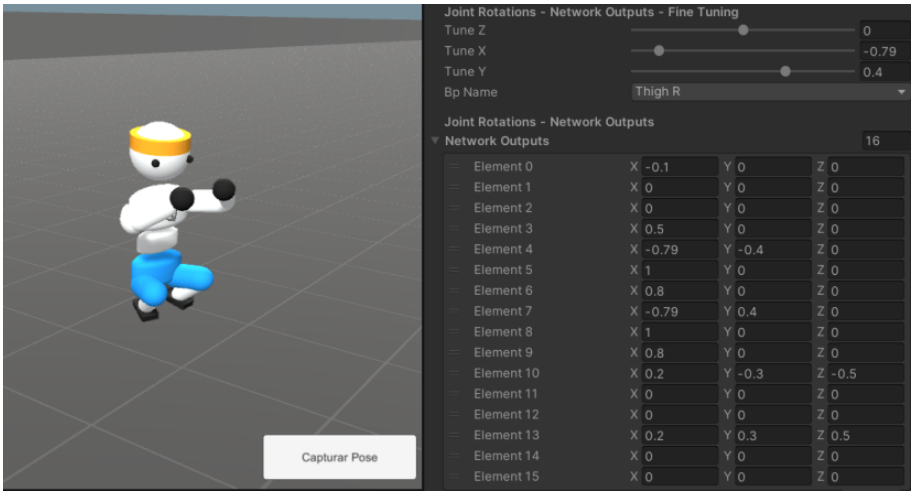
Fonte: Elaborado pelo autor (2024).

As restrições são impostas ajustando propriedades específicas da própria estrutura da *Unity* (Juliani *et al.*, 2020). No caso, *ConfigurableJoints* extras são adicionadas a cada corpo do personagem. Diferente dos *ConfigurableJoints* existentes, responsáveis por conectar os corpos do personagem, essas novas juntas não conectam o corpo a outro. Deixar o campo *ConnectedBody* vazio significa que as restrições impostas pela *join* são relativas ao mundo, correspondendo ao uso de torques ou forças externas aplicadas aos corpos para mantê-los travados. Isso traz mais estabilidade, mas também artificialidade ao equilíbrio, com a chance de ser visualmente perceptível se não usado com moderação.

Em alguns casos, as restrições podem fazer parte do contexto do próprio experimento. Em outros, as restrições podem ser úteis para o animador acessar uma versão prévia facilitada do experimento. Esse ambiente controlado pode gerar compreensões e intuições importantes para

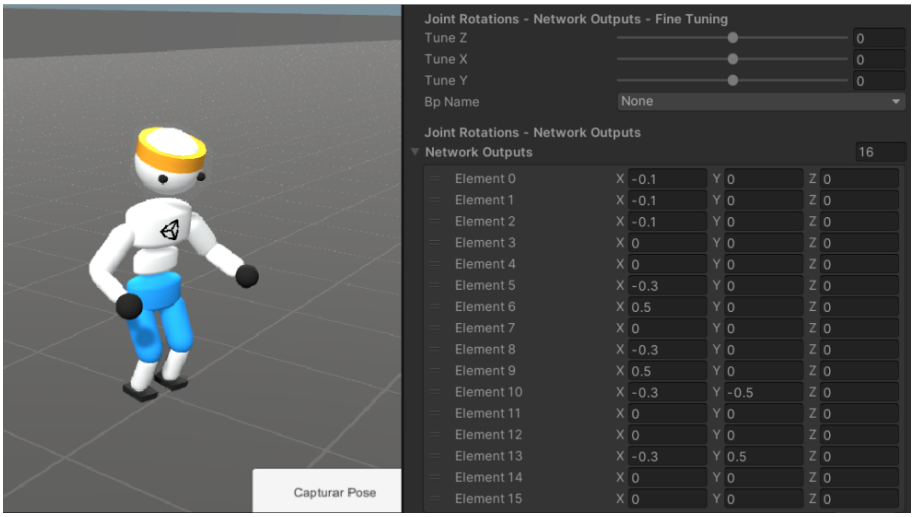
ajudar a lidar posteriormente com a versão original mais difícil, sem o uso das restrições.

Figura 27 – Configurando e salvando pose inicial do episódio.



Fonte: Elaborado pelo autor (2024).

Figura 28 – Pose usada nos experimentos de corrida com equilíbrio artificial.



Fonte: Elaborado pelo autor (2024).

A interface também permite que poses sejam visualmente editadas e salvas, como ilustra a Figura 27, para serem usadas como poses iniciais, carregadas no início do episódio de acordo com o experimento a ser realizado. A pose inicial é útil principalmente para que as restrições artificiais, que são definidas independentemente para cada corpo do personagem, possam ser simplificadas para opções de habilitar ou desabilitar, já que a própria pose inicial traz as informações de posição e orientação dos corpos, a serem usadas como referência para as restrições.

A pose salva fica disponível em uma estrutura de dados no projeto, necessitando de um *check* na interface ser marcado para ser carregada ou não, já que nem todo experimento se

beneficia de uma pose inicial específica. Neste trabalho foram usadas duas poses específicas, uma pose para ajudar no experimento de agachar, a mesma pose da Figura 27, e uma outra pose para os experimentos relacionados a corridas que precisavam de fixar corpos para melhorar o resultado, observadas na Figura 28.

5 RESULTADOS

Este capítulo descreve os experimentos realizados para avaliar a utilidade e a praticidade das ferramentas de controle propostas e da interface integrada, implementadas utilizando *Unity 3D* e *ML-Agents*. Para uma análise mais detalhada das animações resultantes, favor consultar o vídeo suplementar, disponível neste link (Vídeos[1]). As seções a seguir fornecem detalhes sobre a plataforma de desenvolvimento e os experimentos específicos realizados.

5.1 Plataforma de desenvolvimento

Os experimentos dessa pesquisa foram feitos no Unity 3D, que é uma plataforma de desenvolvimento abrangente para criar jogos e animações. Inclui recursos como um motor de física integrado (*PhysX*), detecção de colisão, gráficos de alta fidelidade com bom desempenho computacional, simulação distribuída e uma linguagem flexível de controle de animação (Juliani *et al.*, 2020). Neste estudo, o *Unity* foi usado para modelar estruturas de corpos rígidos com vários tipos de juntas. Além disso, fornece funcionalidades úteis para simulações físicas, como controle *PD* e imposição de limites angulares. O *Unity* emprega uma abordagem de programação baseada em *scripts* usando a linguagem *C#* para implementar ferramentas e interfaces de controle. Mais detalhes sobre plataforma são obtidos na Seção 2.5.

Neste trabalho também é utilizada a biblioteca ML-Agents, uma ferramenta integrada ao *Unity*, oferece um conjunto de ferramentas de aprendizado de máquina de código aberto projetadas para treinar agentes em ambientes realistas e complexos, utilizando algoritmos de Aprendizado por Reforço Profundo (*DRL*) como *Proximal Policy Optimization (PPO)* e *Soft Actor-Critic (SAC)*, para mais detalhes sobre a biblioteca do *Ml-Agents*, consultar a Subseção 2.5.1. Os experimentos deste estudo utilizaram o personagem fornecido pelo *ML-Agents* para o experimento *Walker*, que foi detalhado na Seção 4.3. O ambiente de desenvolvimento incluiu *Unity 3D* versão *2020.3.48f1* no sistema operacional *Windows* 10 e *ML-Agents* versão 14. O treinamento foi realizado em um computador pessoal equipado com um processador *Intel Core i5-10400F 2.9 GHz, 16 GB de RAM* e uma *GPU NVIDIA GTX 1660 Super*. As sessões de treinamento duraram aproximadamente 9 horas e envolveram 30 milhões de passos (observações e ações realizadas) usando o algoritmo *PPO*.

[1] Vídeos Complementares: https://ilderlucas.github.io/

5.2 Controle de informações gerais em tempo real

Após modelar modelar o personagem e configurá-lo dentro do contexto do *ML-Agents*, o animador pode acessar facilmente as ferramentas de controle e a interface propostas incorporando um *script* de configuração de controle em tempo real ao *GameObject* do personagem. O *script* inclui e torna acessível no *Inspector* as três listas de controle: *RealTimeInfoList*, *RealTimePosInfoList* e *ConditionalRewardList* (Figuras 21, 22, 23 e 24), assim como dois grupos de restrições (Figuras 25 e 26).

Inicialmente, essas listas estão vazias, o que significa que nenhuma recompensa será considerada durante o treinamento, e o aprendizado não ocorrerá. Sem as ferramentas propostas, um animador precisaria abrir o código do *script ML-Agents* e editar manualmente o método responsável pelo cálculo das recompensas. Para cada recompensa a ser considerada durante o treinamento, o animador construiria uma fórmula para representá-la e adicionaria seu resultado à função de valor único usando o método *AddReward*.

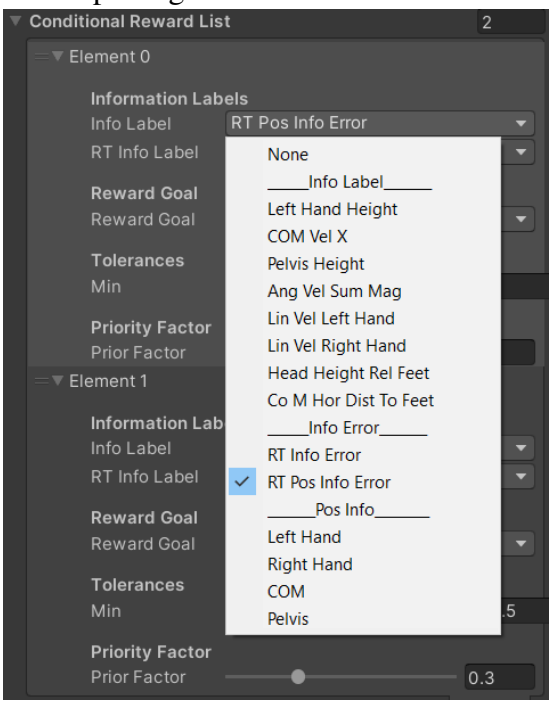
Já usando as ferramentas propostas, o animador apenas precisa incluir elementos às listas e especificar as propriedades para cada item. Durante o treinamento, a *ConditionalRewardList* é automaticamente iterada e, para cada item, a função de valor único é atualizada de acordo com a Equação 4.6. A primeira propriedade a ser especificada para uma recompensa é o *label* de sua informação associada. Para que o animador não tenha necessariamente que editar código (Figura 20), vários *InfoLabels* relevantes já estão pré-registrados e disponibilizados pela ferramenta, como ilustra a Figura 29.

5.2.1 Corrida original de referência

Usando apenas dois dos *InfoLabels* registrados, um dos experimentos (*Walker*) de (Juliani *et al.*, 2020) foi facilmente adaptado para se ajustar ao formato de interface proposto. No caso, o primeiro item foi adicionado à *ConditionalRewardList* para maximizar *CoMVelX*, com tolerâncias $tol_{max} = 5$ e $tol_{min} = 0$ e $p = 0.4$ como fator de prioridade. Para evitar que o personagem caia e incentivar uma postura mais ereta, um segundo item tenta maximizar *HeadHeightRelFeet*, calculado como a diferença entre a altura da cabeça e a altura média dos pés. Para este segundo item, $tol_{max} = 5$, $tol_{min} = 0$ e $p = 0.1$ foram usados. A configuração pode ser conferida da Figura 30.

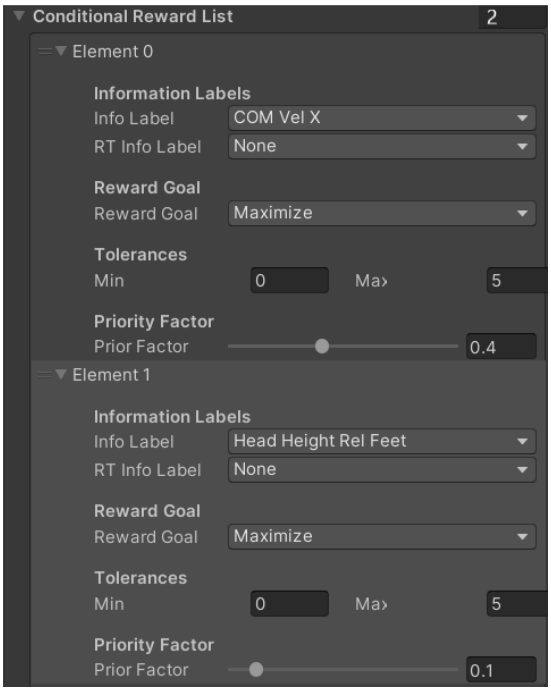
Para garantir que todas as recompensas condicionais sejam atendidas, o que pode ser

Figura 29 – Lista de *infoLabels* pré-registrado



Fonte: Elaborado pelo autor (2024).

Figura 30 – Configuração da interface para corrida original de referência



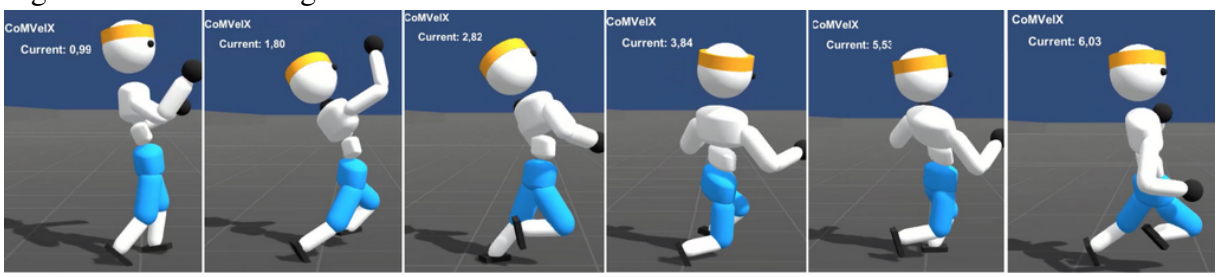
Fonte: Elaborado pelo autor (2024).

o objetivo do animador, todas as tolerâncias escolhidas devem ser possíveis de ser eventualmente atingidas. Entretanto, o animador pode também escolher uma das recompensas para ser a responsável por guiar o aprendizado quando todas as outras forem satisfeitas, podendo ser melhorada o máximo possível, sem concorrência com as outras recompensas. Dessa forma, pode-se usar uma tolerância mais difícil, desde que se use também um fator de prioridade mais

baixo, para que sua prioridade não se sobressaia antes que as outras recompensas possam ser satisfeitas.

Essa ideia foi usada neste experimento. Usando primeiramente $tol_{max} = 3.5$, que foi considerada fácil de atingir, e $p = 0.4$ para a *HeadHeightRelFeet*,o personagem conseguiu satisfazer as duas recompensas. Sua velocidade estava de acordo com o desejado, mas a altura da sua cabeça estava abaixo do que se previa. Então essa foi a recompensa escolhida para ser forçada ao limite, atualizando sua tol_{max} para 5 e seu p para 0.1. Isso resultou em um melhor resultado, semelhante ao *Walker* original, recortes desse resultado são ilustrados na Figura 31. O uso das recompensas condicionais permite que, em vez dos seus pesos, as tolerâncias, que são informações mais intuitivas, sejam ajustadas.

Figura 31 – Corrida original de referência



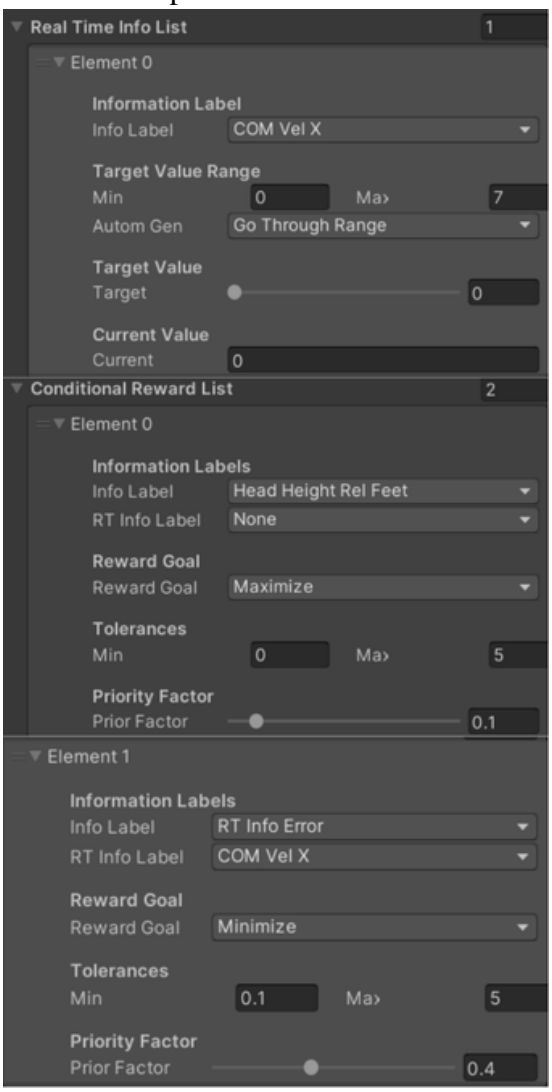
Fonte: Elaborado pelo autor (2024).

5.2.2 Corrida com controle de velocidade em tempo real

A partir do experimento anterior, duas pequenas e rápidas alterações, feitas usando a própria interface, foram suficientes para passar a permitir o controle em tempo real da *CoMVelX*. A primeira foi adicionar um item com o mesmo *InfoLabel*, *CoMVelX*, à *RealTimeInfoList*, gerando automaticamente uma nova recompensa na *ConditionalRewardList*, como ilutrado na Figura 24, mas com o campo *RTInfoLabel* correspondente definido para *CoMVelX*. O intervalo de possíveis valores desejados foi definido como [0, 7], e o método de geração automática foi *GoThroughRange*. Para a recompensa de erro, os valores automáticos foram ajustados para $tol_{min} = 0.1$, $tol_{max} = 5$ e $p = 0.4$ para melhorar a precisão do controle e estender a faixa usada na normalização.

A segunda foi remover o item que já estava maximizando *CoMVelX* na *Conditional-RewardList*, o item já existia da configuração do experimento anterior, e precisa ser excluído para evitar conflito com a recompensa gerada automaticamente. A configuração final da interface para o experimento pode ser ser conferida na Figura 32.

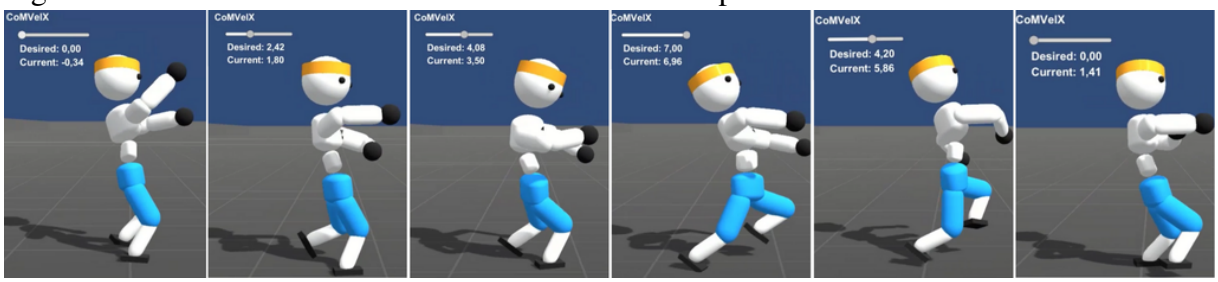
Figura 32 – Configuração da interface para corrida com controle de velocidade em tempo real



Fonte: Elaborado pelo autor (2024).

O controle de *CoMVelX* funcionou bem, com um pequeno atraso, mas com boa precisão. Entretanto, como no experimento anterior, a corrida exibiu movimentos descoordenados, especialmente nos braços. Recortes da simulação do resultado em questão são ilustrados na Figura 33.

Figura 33 – Corrida com controle de velocidade em tempo real



Fonte: Elaborado pelo autor (2024).

5.2.3 Corrida anti-simétrica com controle de velocidade em tempo real

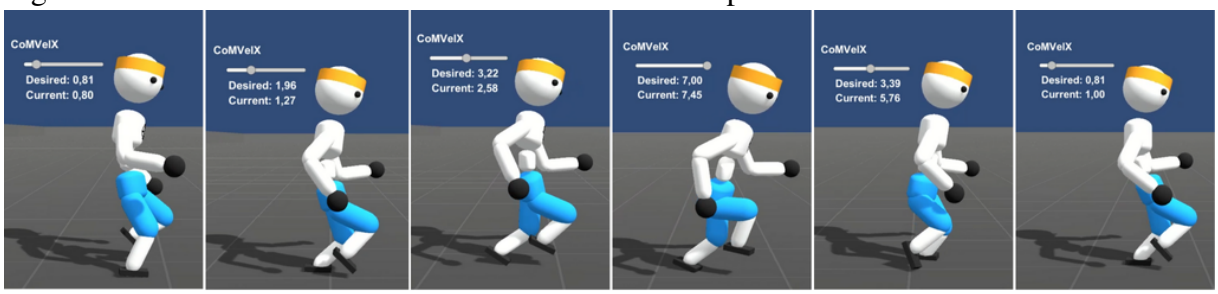
O uso das restrições de simetria propostas, aplicadas facilmente por meio de simples marcações na interface, melhorou significativamente os resultados anteriores. As restrições mostradas na Figura 25, foram marcadas como anti-simétricas *A* para todos os membros, como mostra a Figura 34.

Figura 34 – Configuração da interface adicionando restrições de simetria

Fonte: Elaborado pelo autor (2024).

Consequentemente, a corrida resultante, ilustrada na Figura 35, foi mais estável e coordenada, mantendo boa precisão no controle em tempo real.

Figura 35 – Corrida com controle de velocidade em tempo real anti-simétrica



Fonte: Elaborado pelo autor (2024).

5.2.4 Agachamento com controle de altura em tempo real

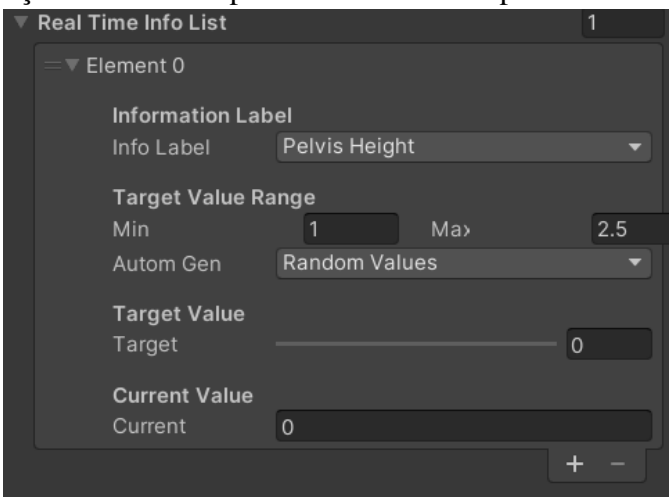
Para demonstrar a generalização do controle em tempo real, foi também realizado um experimento de agachamento, que apresenta uma proposta diferente de movimentação, através do controle da altura da pélvis. Para realizar o agachamento, um item com o *InfoLabel PelvisHeight* foi adicionado à *RealTimeInfoList*, Figura 36. A faixa utilizada foi [1.0, 2.5], e o método de geração automática foi *RandomValues*. A recompensa gerada automaticamente não foi alterada.

Devido a grandes oscilações no resultado obtido, uma recompensa adicional foi usada para minimizar as velocidades angulares do personagem, *AngVelSumMag*. Esta informação é obtida somando as velocidades angulares de todos os corpos e retornando a magnitude do vetor

soma. Foram usados $tol_{min} = 0$, $tol_{max} = 50$ e $p = 0.1$. Restrições de simetria foram marcadas como simétricas S para todos os membros. Restrições artificiais também foram aplicadas: restrições de rotação R foram marcadas para a cabeça e o peito, e restrições de translação T para os pés e mãos. O uso da recompensa adicional reduziu as oscilações indesejadas.

Como restringir excessivamente o personagem pode comprometer sua mobilidade, um novo teste sem as restrições artificiais para as mãos e o peito foi realizado. No entanto, uma recompensa adicional foi necessária para evitar que a artificialidade do equilíbrio se tornasse perceptível, as configurações da interface finais podem ser observadas nas Figuras 36 e 37. Esta recompensa visava minimizar a distância horizontal entre o *CoM* do personagem e o ponto médio entre seus pés, *CoMHorDistToFeet*. Portanto, isso permitiu que os braços se ajustassem automaticamente para compensar o movimento da pelvis. Durante a descida, os braços se moviam para frente enquanto a pelvis se movia para trás, mantendo o *CoM* sobre a área de suporte. Isso resultou em um agachamento mais estável e com melhor controle de altura em tempo real, como ilustrado na Figura 38.

Figura 36 – Configuração da interface para controle em tempo real da altura da pelvis

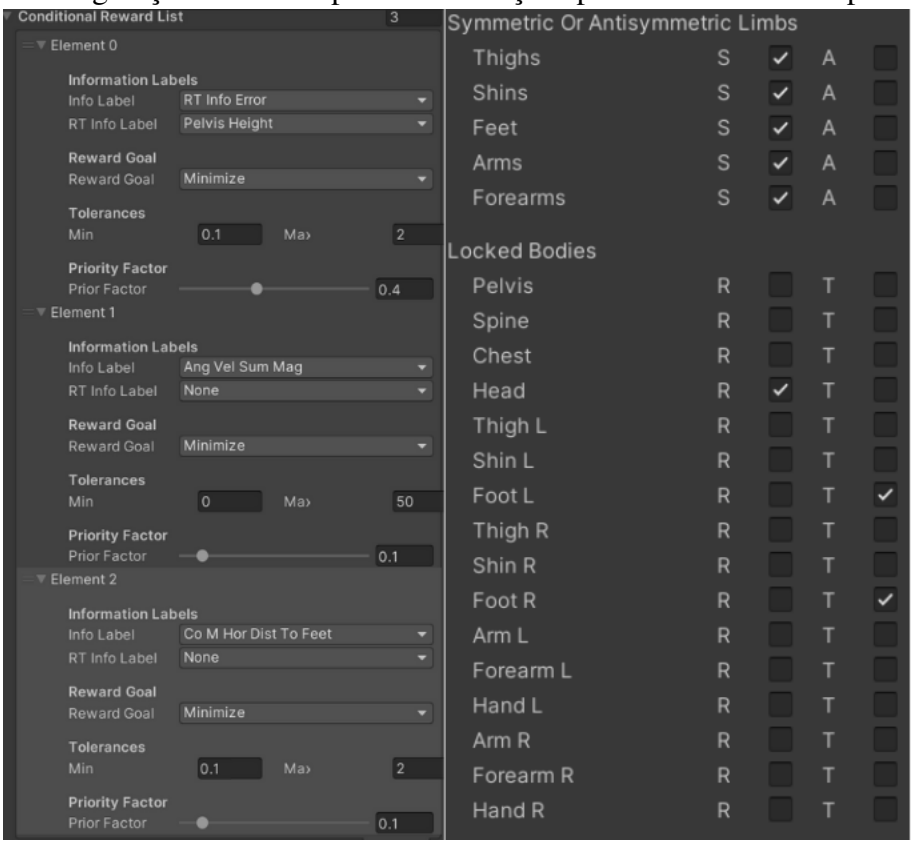


Fonte: Elaborado pelo autor (2024).

5.3 Controle de posição em tempo real

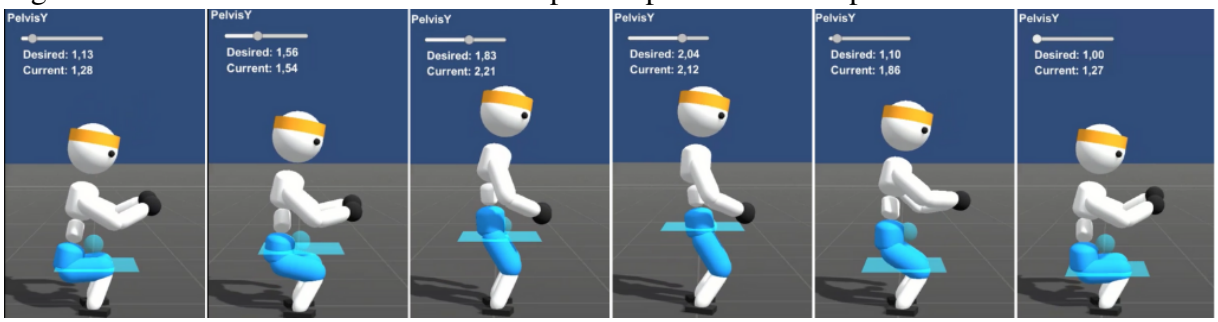
Para testar a forma agrupada de controle em tempo real disponível na interface, foram realizados dois experimentos, um para controlar as posições das mãos e outro para controlar a posição horizontal da pelvis, esses testes serão descritos a seguir.

Figura 37 – Configuração das recompensas e restrições para controle em tempo real



Fonte: Elaborado pelo autor (2024).

Figura 38 – Resultado do controle em tempo real para a altura da pelvis



Fonte: Elaborado pelo autor (2024).

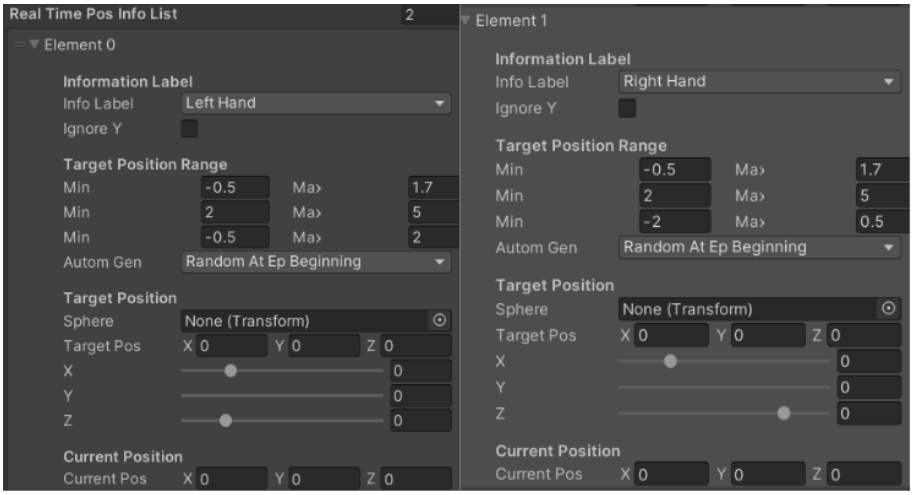
5.3.1 Controle de posição das mãos em tempo real

Imaginando uma solução alternativa parcial para cinemática inversa. Como essa solução é desenvolvida dentro do contexto de simulação física, uma vantagem é que os atuadores finais já são automaticamente influenciados por possíveis colisões, desviando da trajetória desejada e retornando naturalmente a segui-la.

A especificação do treinamento foi realizada incluindo dois elementos na *RealTimePosInfoList*. Os itens tiveram seus campos *InfoLabel* preenchidos com *LeftHand* e *RightHand*. Os intervalos de valores desejados para as coordenadas x, y e z de *LeftHand* foram [-0.5, 1.7],

[2.0, 5.0], e [-0.5, 2.0], respectivamente. Para *RightHand*, os intervalos para x e y foram os mesmos, mas para z, [-2.0, 0.5] foi usado, estendendo-se para o lado oposto do personagem. *RandomAtEpBeginning* foi escolhido para a geração automática de posições desejadas, como ilustrado na Figura 39. Recompensas geradas automaticamente não foram alteradas. Para evitar grandes oscilações nas mãos, duas recompensas condicionais adicionais para reduzir as velocidades lineares das mãos, foram definidas para minimizar suas respectivas velocidades, usando $tol_{min} = 0.0$, $tol_{max} = 10$, e $p = 0.3$ para ambas, essas configurações podem ser observadas na Figura 40 . Restrições de simetria não foram marcadas para nenhum membro do personagem. As restrições artificiais foram aplicadas à rotação (R) e à translação (T) de todos os corpos, exceto nos ombros, antebraços e mãos, como ilustrado na Figura 41.

Figura 39 – Configuração da *RealTimePosInfoList* controle de posição das mãos em tempo real.



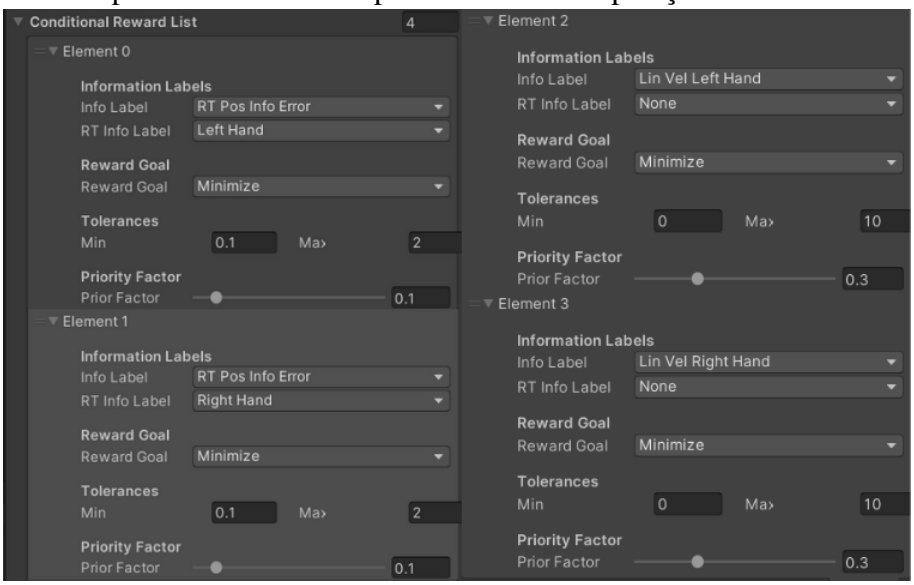
Fonte: Elaborado pelo autor (2024).

Apesar das restrições artificiais utilizadas, o resultado demonstrou a capacidade de controlar simultaneamente seis informações em tempo real, embora a precisão possa ser melhorada em alguns instantes da simulação, como ilustra a Figura 42. Investigações adicionais são necessárias para medir a capacidade de controle em relação à quantidade de informações permitidas.

5.3.2 Controle em tempo real de uma posição horizontal

Um outro experimento foi realizado com o intuito de definir um caminho a ser seguido pelo personagem. A ideia era controlar a posição horizontal da pelvis e permitir que o animador escolhesse a localização desejada em tempo real. Neste caso, a altura da pelvis não importa, apenas sua projeção no chão. Para a especificação do treinamento, um item com

Figura 40 – Recompensas condicionais para o controle de posição das mãos em tempo real.

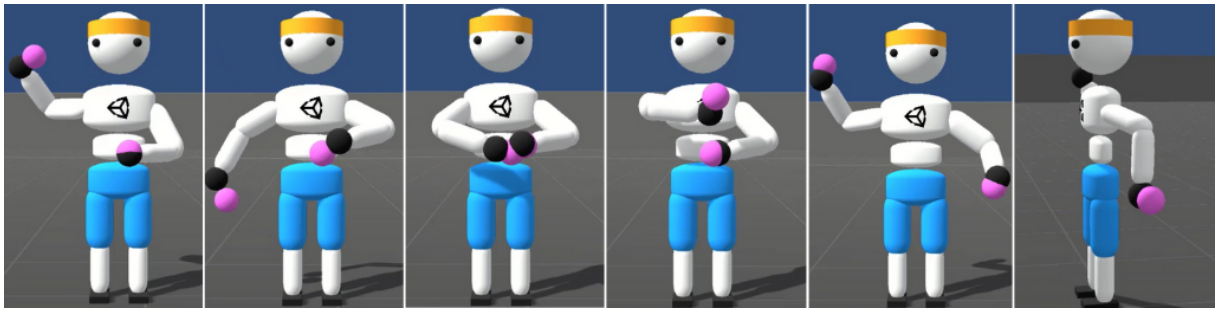


Fonte: Elaborado pelo autor (2024).

Figura 41 – Restrições artificiais para controle de posição das mãos em tempo real.

Fonte: Elaborado pelo autor (2024).

Figura 42 – Controle de posição das mãos em tempo real.



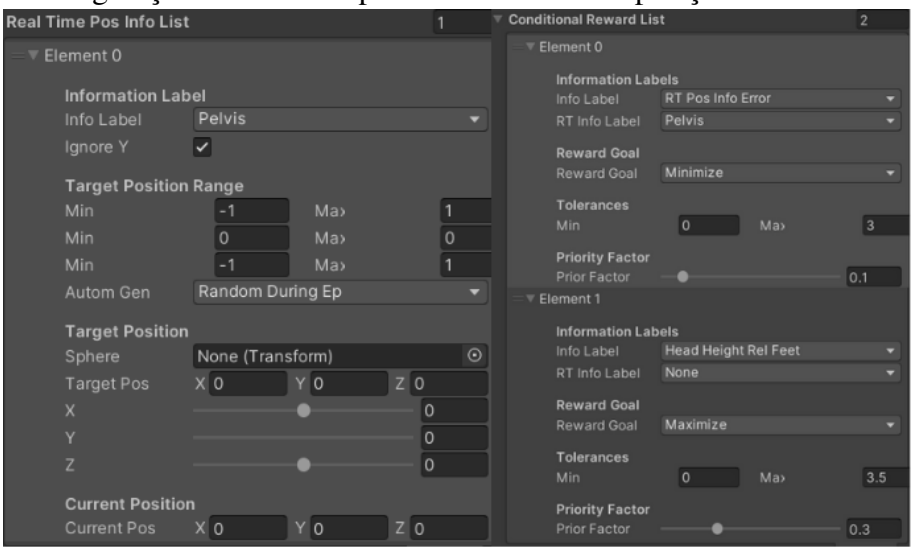
Fonte: Elaborado pelo autor (2024).

Pelvis como *InfoLabel* foi incluído na *RealTimePosInfoList*, e *IgnoreY* foi marcado para controlar apenas sua projeção no chão. *RandomDuringEp* foi escolhido para a geração automática de

posições desejadas, mudando o significado dos intervalos de sorteio, que se tornaram relativos à última posição desejada. Assim, o intervalo [-1, 1] foi usado para as coordenadas x e z da *Pelvis*, enquanto a coordenada y foi ignorada.

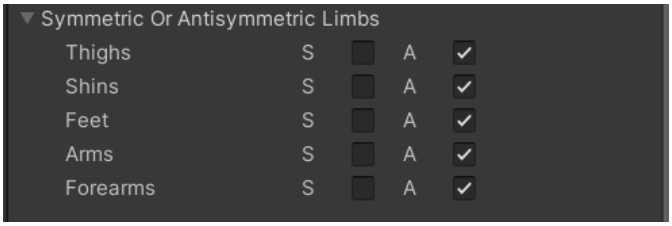
Para a recompensa gerada automaticamente, os valores automáticos foram ajustados para $tol_{min} = 0$, $tol_{max} = 3$, e $p = 0.1$. A mesma recompensa para maximizar *HeadHeightRelFeet* também foi adicionada à *ConditionalRewardList*, mas com $tol_{max} = 3.5$, $tol_{min} = 0$, e $p = 0.3$, visando eventualmente transferir a prioridade para corrigir a projeção da pelvis, observar Figura 43. Restrições de simetria foram marcadas como anti-simétricas (A) para todos os membros, e restrições artificiais não foram utilizadas (Figura 44). Embora com tempo de resposta nem sempre ideal, mas com uma precisão considerável, o personagem treinado conseguiu acompanhar as posições escolhidas em tempo real (Figura 45).

Figura 43 – Configuração da interface para controle de uma posição horizontal em tempo real



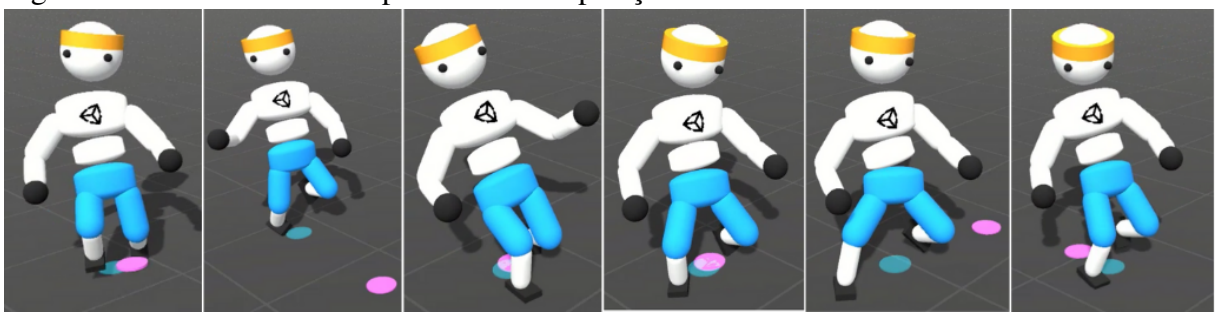
Fonte: Elaborado pelo autor (2024).

Figura 44 – Configuração das restrições de simetria para controle de posição horizontal



Fonte: Elaborado pelo autor (2024).

Figura 45 – Controle em tempo real de uma posição horizontal



Fonte: Elaborado pelo autor (2024).

5.4 Controle por restrições de simetria

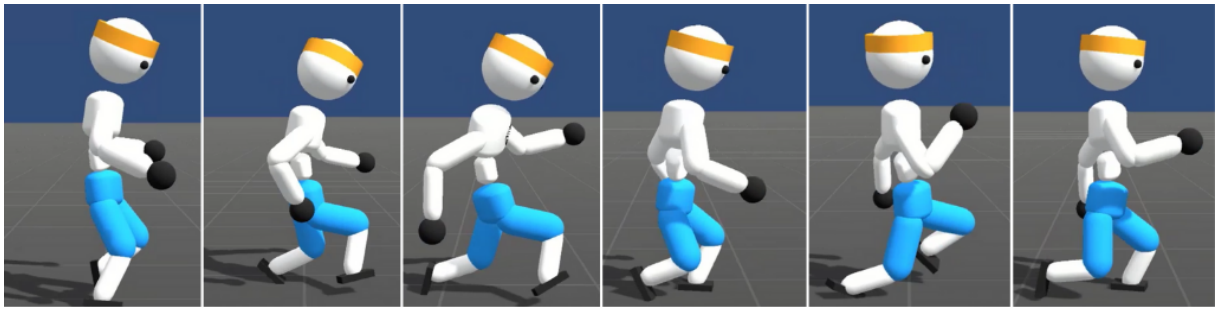
Além de promover movimentos mais coordenados através da sincronização dos membros, as restrições de simetria também influenciam o estilo de locomoção. Para testar sua influência isolada, o mesmo experimento de referência descrito na Subseção 5.2.1 foi repetido apenas usando diferentes combinações das restrições de simetria disponíveis. Através da simples marcação dos *checks* de simetria na interface, como descrito na Seção 4.6, o aprendizado pode ser direcionado para corrida bípede tradicional, saltos estilo canguru ou galope. As configurações da interface e recortes dos resultados são demonstrados nas subseções a seguir.

5.4.1 Corrida bípede tradicional

No caso, para obter a corrida bípede, as restrições de simetria foram marcadas como anti-simétricas *A* para todos os membros, mesma configuração da Figura 44.

Foi observado que a anti-simetria dos joelhos é essencial para que as pernas alternem para frente e para trás na corrida bípede, como ilustra o resultado do experimento na Figura 46. Isso ocorre porque a perna de trás, quando deixa de ser a de suporte, precisa se mover para frente dobrada para que o pé no ar não colida com o chão durante a passada, enquanto a perna da frente permanece mais estendida para melhor segurar o pé de suporte no chão.

Figura 46 – Corrida bípede tradicional com anti-simetria

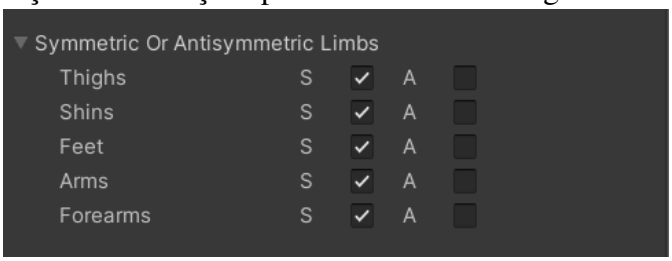


Fonte: Elaborado pelo autor (2024).

5.4.2 Corrida estilo canguru

Para a corrida estilo canguru, as restrições foram marcadas como simétricas *S* para todos os membros (Figura 47).

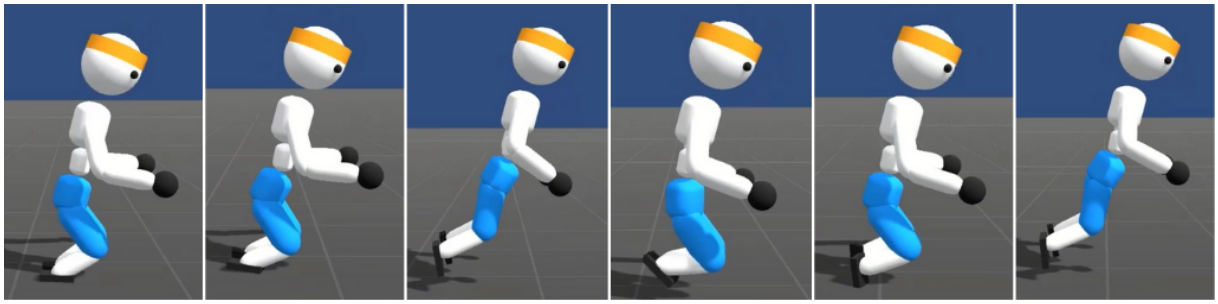
Figura 47 – Configuração das restrições para corrida estilo canguru



Fonte: Elaborado pelo autor (2024).

As restrições aplicadas resultaram no comportamento semelhante para os membros restritos, caracterizando uma locomoção estilo canguru, como pode ser observado na Figura 48.

Figura 48 – Corrida estilo canguru com simetria



Fonte: Elaborado pelo autor (2024).

5.4.3 Corrida estilo galope

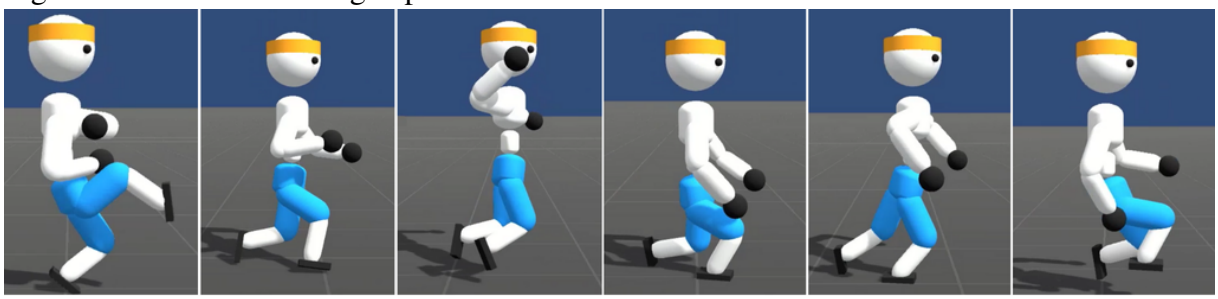
Para galope, as marcações foram usadas exatamente como mostrado na Figura 49.

Figura 49 – Configuração da restrições para corrida estilo galope

Fonte: Elaborado pelo autor (2024).

Logo, com as coxas *A* e os joelhos *S*, uma das pernas tende a ficar sempre atrás e a outra sempre à frente, caracterizando o galope, veja na Figura 50.

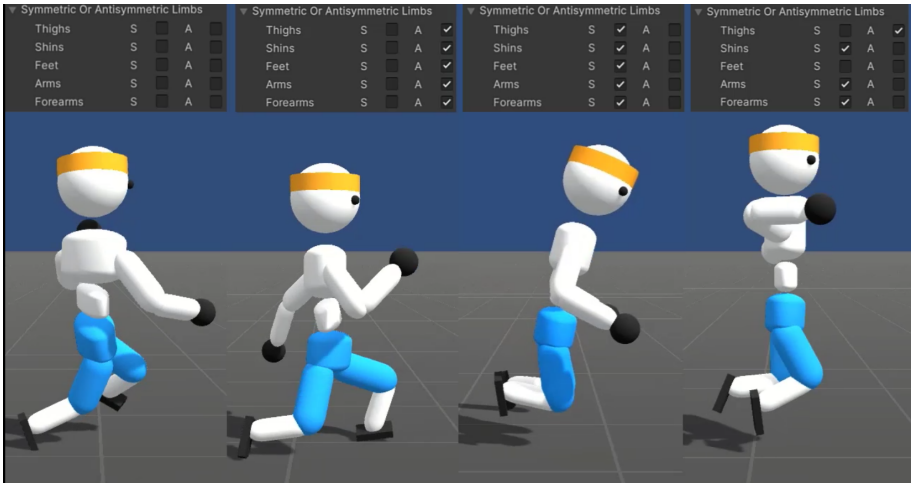
Figura 50 – Corrida estilo galope com simetria



Fonte: Elaborado pelo autor (2024).

A Figura 51 demonstra bem a diversidade de configurações que a ferramenta de restrições de simetria possibilita e seus respectivos estilos de movimento, destacando a melhoria significativa na corrida que pode ser percebida entre os dois primeiros personagens da figura, além dos diferentes estilos de locomoção obtidos para configurações específicas das restrições de simetria.

Figura 51 – Quadro resumo da diversidade de movimentos com simetria



Fonte: Elaborado pelo autor (2024).

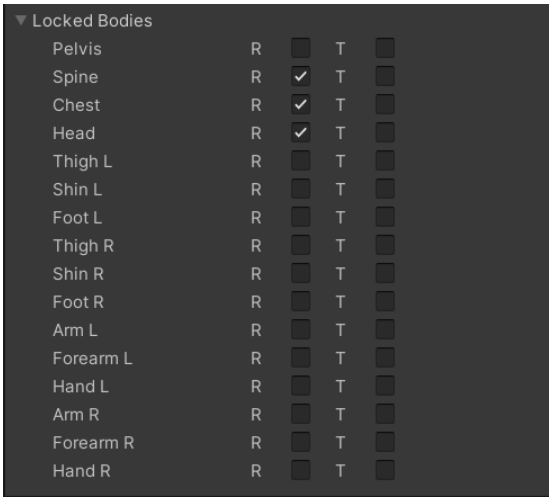
5.5 Controle por restrições artificiais

Para demonstrar os benefícios do uso do equilíbrio artificial também foram feitos experimentos dos estilos de corridas descritos nas seções anteriores com restrições de rotações do tronco superior do personagem, resultando em simulações mais estáveis e contribuindo com os aspectos do controle em tempo real.

5.5.1 Corrida bípede tradicional com controle de velocidade em tempo real

Aproveitando todas as configurações que usamos para obter a corrida bípede anti-simétrica com controle de velocidade em tempo real, descrito na Subseção 5.2.3, a interface foi usada para aplicar restrições artificiais e melhorar o comportamento do personagem. Então as rotações da espinha, peito e cabeça foram fixadas, como ilustra a Figura 52.

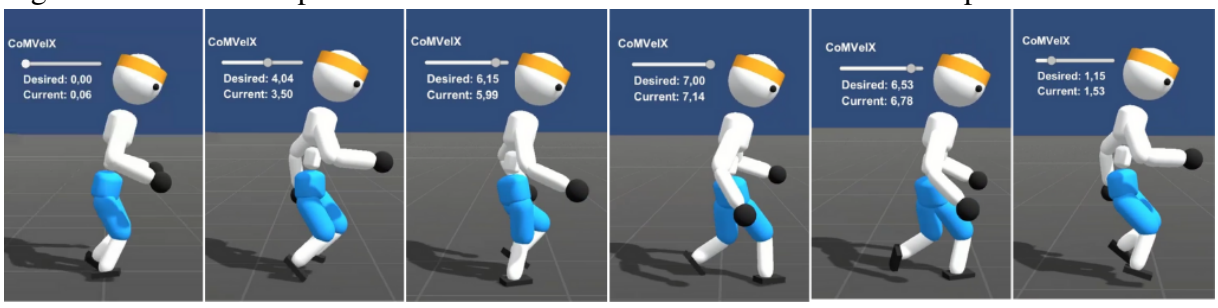
Figura 52 – Configuração da restrições artificias para corrida bípede tradicional com controle de velocidade em tempo real



Fonte: Elaborado pelo autor (2024).

Foi possível obter um resultado bem mais estável, com uma postura melhor. Foi percebido que os efeitos do uso do equilíbrio artificial também contribuem para um tempo de resposta e precisão melhores no controle em tempo real da velocidade, como ilustra a Figura 53.

Figura 53 – Corrida bípede tradicional com controle de velocidade em tempo real



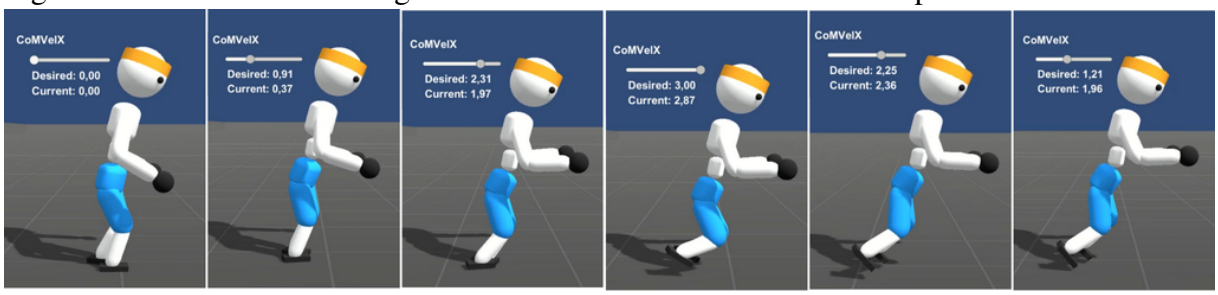
Fonte: Elaborado pelo autor (2024).

5.5.2 Corrida estilo canguru com controle de velocidade em tempo real

A corrida estilo canguru, obtida na Subseção 5.4.2, foi repetida com controle de velocidade em tempo real e as mesmas restrições artificiais mostradas na Figura 52. O experimento

em questão também usa as mesmas configurações para o controle de velocidade em tempo real mostradas na Figura 32, com exceção do intervalo de possíveis valores desejados, que é de [0, 3], em vez de [0, 7]. O resultado é ilustrado na Figura 54.

Figura 54 – Corrida estilo canguru com controle de velocidade em tempo real

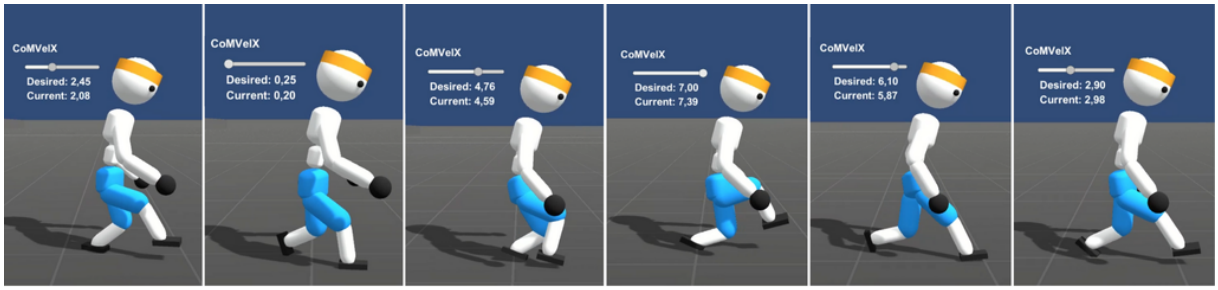


Fonte: Elaborado pelo autor (2024).

5.5.3 Corrida estilo galope com controle de velocidade em tempo real

O mesmo procedimento foi realizado para a corrida galope, na tentativa de melhorar o resultado existente, usando também as mesmas restrições artificiais da Figura 52. O resultado é ilustrado na Figura 55.

Figura 55 – Corrida estilo galope com controle de velocidade em tempo real



Fonte: Elaborado pelo autor (2024).

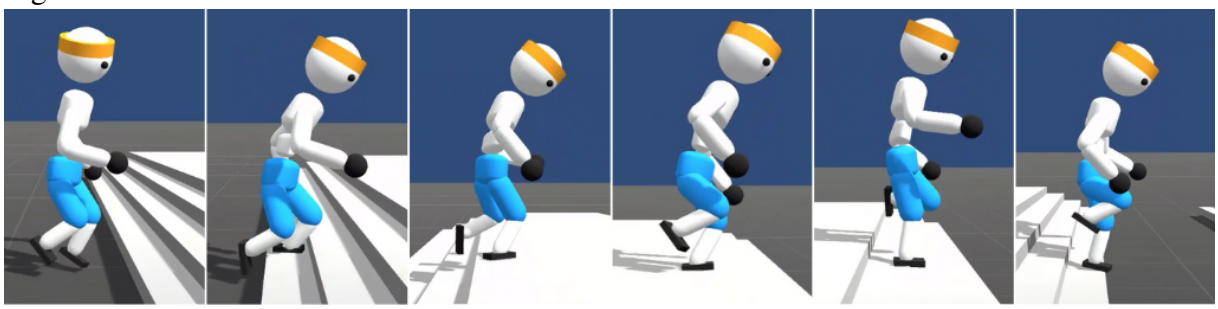
5.6 Adaptabilidade ao ambiente

Para demonstrar a capacidade do personagem treinado de se adaptar a diferentes ambientes usando DRL, dois cenários diferentes foram implementados para o treinamento.

5.6.1 Cenário com escadas

O mesmo experimento de referência descrito na Subseção 5.2.1 foi repetido em um cenário com escadas (Figura 56). As únicas mudanças na interface foram definir $tol_{max} = 2$ para

Figura 56 – Cenário com escadas



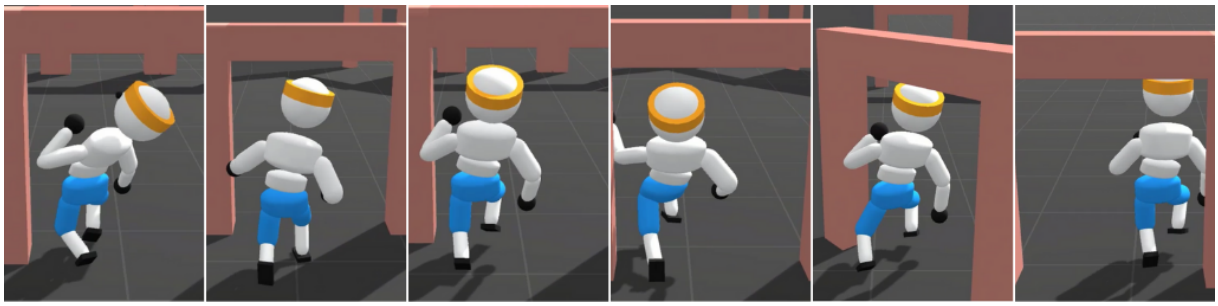
Fonte: Elaborado pelo autor (2024).

a recompensa condicional CoMVelX e marcar as restrições de simetria como antissimétricas *A* para todos os membros.

5.6.2 Cenário com portas e paredes

Da mesma forma, o mesmo experimento foi também repetido em um cenário com portas e paredes, como ilustra a Figura 57.

Figura 57 – Cenário com portas



Fonte: Elaborado pelo autor (2024).

O personagem foi capaz de navegar por ambos os cenários, embora com precisão limitada. Vale destacar a capacidade do personagem de acertar alguns degraus com precisão e de se abaixar levemente para passar por portas, mesmo sem a rede ter conhecimento explícito desses novos cenários. Trabalhos futuros investigarão melhorias nesse aspecto, tornando a rede neural ciente do cenário.

6 CONCLUSÕES E TRABALHOS FUTUROS

O controle ativo dos movimentos de um personagem articulado baseado em física é um desafio complexo, pois exige que os movimentos gerados sejam não apenas estáveis e naturais, mas também que ofereçam ao animador um grau satisfatório de controle sobre o resultado final da animação. A extensão dos campos de aprendizagem de máquina permitiu o florescimento de inúmeros campos de aplicação, não sendo diferente para a área de computação gráfica. Com o avanço dos algoritmos de aprendizagem por reforço profundo, pesquisas recentes de controle ativo de personagens articulados conseguiram resultados importantes.

Este trabalho apresenta contribuições significativas para os três componentes essenciais do controle por DRL: entrada, aprendizado e saída da rede neural. A generalização do controle em tempo real influencia a entrada da rede e, combinada com a padronização proposta através de recompensas condicionais, afeta a especificação da função de recompensa da rede. Isso é crucial para atualizar adequadamente os pesos da rede durante o aprendizado. As restrições de simetria impactam a saída da rede, influenciando diretamente o tamanho e a forma do espaço de ação.

Uma interface integrada demonstra essas generalizações propostas, facilitando a especificação do treinamento e o controle em tempo real das informações selecionadas. Embora o potencial dessas ferramentas ainda não tenha sido totalmente explorado, os resultados apresentados demonstram sua utilidade e praticidade, mostrando promessas e abrindo caminho para experimentos futuros. Espera-se que essas contribuições incentivem um uso mais amplo do DRL, não apenas por animadores experientes.

Uma das propostas da interface é permitir que os animadores explorem recompensas usando vários tipos de informações. No entanto, definir as informações mais relevantes para cada experimento continua sendo crucial para alcançar resultados de alta qualidade visual. Trabalhos futuros devem se concentrar na adaptação de recompensas e estratégias de trabalhos relacionados dentro do contexto da interface proposta.

As ferramentas de controle foram projetadas para serem facilmente adaptáveis a qualquer estrutura de corpos rígidos articulados, tornando uma extensão natural testar a interface proposta com novos personagens. Além disso, a interface deve incorporar formas mais flexíveis de restrições artificiais, explorando o Curriculum Learning para remover gradualmente essa artificialidade.

Embora as restrições de simetria propostas sincronizem efetivamente os membros

em cada instante isolado, o objetivo é aprimorar esse controle para sincronizar diferentes fases da locomoção, como diferentes passadas durante a corrida. Por exemplo, isso incentivaria o pé esquerdo em uma passada a se estender tanto para frente quanto o pé direito fez na passada anterior.

REFERÊNCIAS

ABERMAN, K.; WU, R.; LISCHINSKI, D.; CHEN, B.; COHEN-OR, D. Learning character-agnostic motion for motion retargeting in 2d. **arXiv preprint arXiv:1905.01680**, 2019.

AGRAWAL, S.; PANNE, M. van de. Task-based locomotion. **ACM Transactions on Graphics (TOG)**, ACM New York, NY, USA, v. 35, n. 4, p. 1–11, 2016.

ALBUAINAIN, A. R.; GATZOULIS, C. Reinforcement learning for physics-based competitive games. In: IEEE. **2020 International Conference on Innovation and Intelligence for Informatics, Computing and Technologies (3ICT)**. [*S. l.*], 2020. p. 1–6.

ALLEN, B.; CHU, D.; SHAPIRO, A.; FALOUTSOS, P. On the beat!: timing and tension for dynamic characters. In: SAN DIEGO, CA, USA. **Symposium on Computer Animation**. [*S. l.*], 2007. p. 239–247.

ARAUJO, P. d. A. **Analisando Técnicas de Captura de Movimento**. Monografia de Conclusão de Curso – Universidade Federal Fluminense, 2015. Disponível em: https://app.uff.br/riuff/handle/1/5701. Acesso em: 03 set 2022.

ARULKUMARAN, K.; DEISENROTH, M. P.; BRUNDAGE, M.; BHARATH, A. A. A brief survey of deep reinforcement learning. **arXiv preprint arXiv:1708.05866**, 2017. Acesso em: 13 set 2023.

BERGAMIN, K.; CLAVET, S.; HOLDEN, D.; FORBES, J. R. Drecon: data-driven responsive control of physics-based characters. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 38, n. 6, 2019. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/3355089.3356536. Acesso em: 03 set 2022.

BOOTH, J.; BOOTH, J. Marathon environments: Multi-agent continuous control benchmarks in a modern video game engine. **arXiv preprint arXiv:1902.09097**, 2019.

BOOTH, J.; IVANOV, V. Realistic physics based character controller. **arXiv preprint arXiv:2006.07508**, 2020.

CAVALCANTE, A. S. N. **Avaliando simulações físicas de um personagem controlado por rede neural na Unity**. Monografia de Conclusão de Curso – Universidade Federal do Ceará, 2018. Disponível em: https://repositorio.ufc.br/handle/riufc/39480. Acesso em: 27 ago 2024.

CHENTANEZ, N.; MüLLER, M.; MACKLIN, M.; MAKOVIYCHUK, V.; JESCHKE, S. Physics-based motion capture imitation with deep reinforcement learning. In: **Proceedings of the 11th ACM SIGGRAPH Conference on Motion, Interaction and Games**. [*S. l.*]: ACM, 2018. (MIG '18). ISBN 9781450360159.

COROS, S.; BEAUDOIN, P.; PANNE, M. Van de. Robust task-based control policies for physics-based characters. In: **ACM SIGGRAPH Asia 2009 papers**. Yokohama, Japan: ACM, 2009. p. 1–9.

ESCONTRELA, A.; PENG, X. B.; YU, W.; ZHANG, T.; ISCEN, A.; GOLDBERG, K.; ABBEEL, P. Adversarial motion priors make good substitutes for complex reward functions. ieee. In: **International Conference on Intelligent Robots and Systems (IROS)**. [*S. l.*: *s. n.*], 2022. v. 2. Acesso em: 13 ago 2024.

França, Ícaro Goulart Faria Motta. **Aprendizagem por reforço e por imitação com redes neurais aplicada ao jogo bomberman**. Dissertação (Mestrado) – Universidade Federal Fluminense, 2019. Disponível em: https://1library.org/document/ydxnm76z-universidade-fluminense-franca-aprendizagem-reforco-imitacao-aplicada-bomberman.html. Acesso em: 27 ago 2023.

GAMAGE, V.; ENNIS, C.; ROSS, R. **Data-Driven Reinforcement Learning for Virtual Character Animation Control**. 2021.

GEIJTENBEEK, T.; PANNE, M. van de; STAPPEN, A. F. van der. Flexible muscle-based locomotion for bipedal creatures. **ACM Transactions on Graphics (TOG)**, ACM, New York, NY, USA, v. 32, n. 6, p. 1–11, 2013. ISSN 0730-0301. Acesso em: 13 ago 2024.

GEIJTENBEEK, T.; PRONOST, N. Interactive character animation using simulated physics: A state-of-the-art review. In: WILEY ONLINE LIBRARY. **Computer graphics forum**. [*S. l.*], 2012. v. 31, n. 8, p. 2492–2515.

GEIJTENBEEK, T.; PRONOST, N.; EGGES, A.; OVERMARS, M. H. Interactive character animation using simulated physics. **Eurographics (State of the Art Reports)**, p. 127–149, 2011.

HAARNOJA, T.; ZHOU, A.; ABBEEL, P.; LEVINE, S. Soft actor-critic: Off-policy maximum entropy deep reinforcement learning with a stochastic actor. In: **Proceedings of the 35th International Conference on Machine Learning**. [*S. l.*]: PMLR, 2018. (Proceedings of Machine Learning Research, v. 80), p. 1861–1870. Acesso em: 13 ago 2024.

HEESS, N.; TB, D.; SRIRAM, S.; LEMMON, J.; MEREL, J.; WAYNE, G.; TASSA, Y.; EREZ, T.; WANG, Z.; ESLAMI, S. *et al.* Emergence of locomotion behaviours in rich environments. **arXiv preprint arXiv:1707.02286**, 2017.

JULIANI, A.; BERGES, V.-P.; TENG, E.; COHEN, A.; HARPER, J.; ELION, C.; GOY, C.; GAO, Y.; HENRY, H.; MATTAR, M.; LANGE, D. **Unity: A General Platform for Intelligent Agents**. 2020.

KRUPNIK, O.; MORDATCH, I.; TAMAR, A. Multi-agent reinforcement learning with multi-step generative models. In: PMLR. **Conference on Robot Learning**. [*S. l.*], 2020. p. 776–790.

KWIATKOWSKI, A.; ALVARADO, E.; KALOGEITON, V.; LIU, C. K.; PETTRÉ, J.; PANNE, M. van de; CANI, M.-P. A survey on reinforcement learning methods in character animation. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [*S. l.*], 2022. v. 41, n. 2, p. 613–639.

LAI, M. Giraffe: Using deep reinforcement learning to play chess. **arXiv preprint arXiv:1509.01549**, 2015. Acesso em: 13 set 2023.

LANHAM, M. **Learn Unity ML-Agents–Fundamentals of Unity Machine Learning: Incorporate new powerful ML algorithms such as Deep Reinforcement Learning for games**. [*S. l.*]: Packt Publishing Ltd, 2018.

LASZLO, J. **Controlling bipedal locomotion for computer animation.** [*S. l.*]: University of Toronto, 1997.

LEARNING, A. G. H.-O. M. **with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems**. [*S. l.*]: O'Reilly Media, 2017.

LI, S.; PANG, Y.; BAI, P.; LIU, Z.; LI, J.; HU, S.; WANG, L.; WANG, G. **Learning Agility and Adaptive Legged Locomotion via Curricular Hindsight Reinforcement Learning**. 2023.

LIANG, J.; MAKOVIYCHUK, V.; HANDA, A.; CHENTANEZ, N.; MACKLIN, M.; FOX, D. Gpu-accelerated robotic simulation for distributed reinforcement learning. **arXiv preprint arXiv:1810.05762**, 2018. Acesso em: 13 set 2023.

LILLICRAP, T. P.; HUNT, J. J.; PRITZEL, A.; HEESS, N.; EREZ, T.; TASSA, Y.; SILVER, D.; WIERSTRA, D. Continuous control with deep reinforcement learning. **arXiv**, 2015. Acesso em: 13 ago 2024.

LIU, L.; HODGINS, J. Learning basketball dribbling skills using trajectory optimization and deep reinforcement learning. **ACM Transactions on Graphics (TOG)**, ACM New York, NY, USA, v. 37, n. 4, p. 1–14, 2018.

MA, X. **Extending a Game Engine with Machine Learning and Artificial Intelligence**. Dissertação (Master's thesis) – Aalto University, 2019. Disponível em: https://aaltodoc.aalto.fi/server/api/core/bitstreams/76579135-c21b-464d-9207-bb4bb5473af2/content. Acesso em: 27 ago 2024.

MNIH, V.; KAVUKCUOGLU, K.; SILVER, D.; GRAVES, A.; ANTONOGLOU, I.; WIERSTRA, D.; RIEDMILLER, M. Playing atari with deep reinforcement learning. **arXiv preprint arXiv:1312.5602**, 2013. Acesso em: 13 ago 2024.

NOGUEIRA, Y. L. B. **Um modelo de sistema nervoso para o problema do controle de animação por dinâmica direta**. Tese de Doutorado – Universidade Federal do Ceará, 2007. Disponível em: https://repositorio.ufc.br/handle/riufc/18656. Acesso em: 27 ago 2023.

NUNES, R. F. **Uma representação flexível de controladores para animação fisicamente realista de personagens virtuais**. Dissertação (Dissertação (Mestrado em Ciência da Computação)) – Universidade Federal do Ceará, Fortaleza, CE, 2006. Disponível em: http://repositorio.ufc.br/handle/riufc/18560. Acesso em: 13 ago 2024.

NUNES, R. F. **Usando vibrações naturais na descrição e no controle de locomoções fisicamente simuladas de personagens articulados arbitrários**. Tese (Tese de Doutorado) – Universidade Federal do Ceará, 2012. Disponível em: https://repositorio.ufc.br/handle/riufc/18684. Acesso em: 13 ago 2024.

NUNES, R. F.; CAVALCANTE-NETO, J. B.; VIDAL, C. A.; KRY, P. G.; ZORDAN, V. B. Using natural vibrations to guide control for locomotion. In: **Proceedings of the ACM SIGGRAPH Symposium on Interactive 3D Graphics and Games**. [*S. l.*: *s. n.*], 2012. p. 87–94. Acesso em: 21 ago 2023.

OKWUASHI, O.; ISONG, M.; EYO, E.; EYOH, A.; NWANEKEZIE, O.; OLAYINKA, D. N.; UDOUDO, D. O.; OFEM, B. Gis cellular automata using artificial neural network for land use change simulation of lagos, nigeria. **Journal of Geography and Geology**, Canadian Center of Science and Education, v. 4, n. 2, p. 94, 2012.

OTTONI, A. L. C.; LAMPERTI, R. D.; NEPOMUCENO, E. G.; OLIVEIRA, M.; OLIVEIRA, F. Modelagem e simulaçao de um sistema de aprendizado de reforço para robôs. **VIII Encontro Mineiro de Engenharia de Produçao, ISSN 1983**, v. 629, 2012.

PANNE, M. v. d.; LAMOURET, A. Guided optimization for balanced locomotion. In: **Computer Animation and Simulation'95**. [*S. l.*]: Springer, 1995. p. 165–177.

PANNE, M. Van de; KIM, R.; FIUME, E. Virtual wind-up toys for animation. In: CITESEER. **Graphics Interface**. [*S. l.*], 1994. p. 208–208. Acesso em: 13 ago 2024.

PARK, S.; RYU, H.; LEE, S.; LEE, S.; LEE, J. Learning predict-and-simulate policies from unorganized human motion data. **ACM Transactions on Graphics (TOG)**, ACM New York, NY, USA, v. 38, n. 6, p. 1–11, 2019.

PENG, X. B.; ABBEEL, P.; LEVINE, S.; PANNE, M. Van de. Deepmimic: Example-guided deep reinforcement learning of physics-based character skills. **ACM Transactions On Graphics (TOG)**, ACM New York, NY, USA, v. 37, n. 4, p. 1–14, 2018.

PENG, X. B.; COUMANS, E.; ZHANG, T.; LEE, T.-W.; TAN, J.; LEVINE, S. Learning agile robotic locomotion skills by imitating animals. **arXiv preprint arXiv:2004.00784**, 2020.

PENG, X. B.; KANAZAWA, A.; MALIK, J.; ABBEEL, P.; LEVINE, S. Sfv: Reinforcement learning of physical skills from videos. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 37, n. 6, nov. 2018.

PENG, X. B.; MA, Z.; ABBEEL, P.; LEVINE, S.; KANAZAWA, A. Amp: adversarial motion priors for stylized physics-based character control. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 40, n. 4, 2021. ISSN 0730-0301.

POOLE, D. L.; MACKWORTH, A. K. **Artificial Intelligence: foundations of computational agents**. [*S. l.*]: Cambridge University Press, 2010.

RAIBERT, M. H.; HODGINS, J. K. Animation of dynamic legged locomotion. In: **Proceedings of the 18th annual conference on Computer graphics and interactive techniques**. [*S. l.*: *s. n.*], 1991. p. 349–358.

RANGANATH, A.; XU, P.; KARAMOUZAS, I.; ZORDAN, V. Low dimensional motor skill learning using coactivation. In: **Proceedings of the 12th ACM SIGGRAPH Conference on Motion, Interaction and Games**. New York, NY, USA: Association for Computing Machinery, 2019. (MIG '19). ISBN 9781450369947. Disponível em: https://doi.org/10.1145/3359566.3360071. Acesso em: 01 ago 2024.

RASCHKA, S.; MIRJALILI, V. **Machine Learning mit Python und Scikit-Learn und TensorFlow: Das umfassende Praxis-Handbuch für Data Science, Predictive Analytics und Deep Learning**. [*S. l.*]: MITP-Verlags GmbH & Co. KG, 2017.

REIS, L. P. **Coordenação em Sistemas Multi-Agente**. Tese (Doutorado) – PhD thesis, Faculdade de Engenharia da Universidade do Porto, 2003.

REN, J.; YU, C.; CHEN, S.; MA, X.; PAN, L.; LIU, Z. **DiffMimic: Efficient Motion Mimicking with Differentiable Physics**. 2023.

RUSSELL, S.; NORVIG, P. **Artificial intelligence: a modern approach**. Nova Jersey, EUA: Prentice Hall, 2002. Acesso em: 13 set 2023.

SCHULMAN, J.; WOLSKI, F.; DHARIWAL, P.; RADFORD, A.; KLIMOV, O. **Proximal Policy Optimization Algorithms**. 2017. Disponível em: https://arxiv.org/abs/1707.06347. Acesso em: 13 ago 2024.

SHAPIRO, A.; LEE, S.-H. Practical character physics for animators. **IEEE computer graphics and applications**, IEEE, v. 31, n. 4, p. 45–55, 2010.

SILVA, D. B. d. **Um modelo de contato simplificado para tratamento de equilíbrio**. Dissertação (Dissertação de Mestrado) – Universidade Federal do Ceará, 2014. Disponível em: https://repositorio.ufc.br/handle/riufc/12072. Acesso em: 13 ago 2024.

SILVA, D. B. da; NUNES, R. F.; VIDAL, C. A.; CAVALCANTE-NETO, J. B.; KRY, P. G.; ZORDAN, V. B. Tunable robustness: An artificial contact strategy with virtual actuator control for balance. In: WILEY ONLINE LIBRARY. **Computer Graphics Forum**. [*S. l.*], 2017. v. 36, n. 8, p. 499–510.

SILVA, V. A.; MORAIS, B. Y. D.; OLIVEIRA, S. M. F. de; SILVA, D. B. da. Aprendizagem profunda em unity com ml-agents. **Sociedade Brasileira de Computação**, 2020.

SILVER, D.; SCHRITTWIESER, J.; SIMONYAN, K.; ANTONOGLOU, I.; HUANG, A.; GUEZ, A.; HUBERT, T.; BAKER, L.; LAI, M.; BOLTON, A. *et al.* Mastering the game of go without human knowledge. **nature**, Nature Publishing Group, v. 550, n. 7676, p. 354–359, 2017. Acesso em: 13 set 2023.

SKUROWSKI, P.; PAWLYTA, M. Gap reconstruction in optical motion capture sequences using neural networks. **Sensors**, v. 21, n. 18, 2021. ISSN 1424-8220. Disponível em: https://www.mdpi.com/1424-8220/21/18/6115. Acesso em: 13 ago 2024.

SOUSA, A. S. d. **Controle de movimento de personagens fisicamente simulados usando funções de recompensa**. Dissertação (Mestrado) – Universidade Federal do Ceará (UFC), 2022. Disponível em: http://www.repositorio.ufc.br/handle/riufc/70419. Acesso em: 13 ago 2024.

SOUSA, A. S. de; NUNES, R. F.; VIDAL, C. A.; CAVALCANTE-NETO, J. B.; SILVA, D. B. da. Physics-based motion control through drl's reward functions. In: **Symposium on Virtual and Augmented Reality**. [*S. l.*: *s. n.*], 2021. p. 127–136.

SUTTON, R. S.; BARTO, A. G. Reinforcement learning: an introduction mit press. **Cambridge, MA**, v. 22447, 1998.

SUTTON, R. S.; BARTO, A. G. **Reinforcement learning: An introduction**. Second. Cambridge, EUA: MIT press, 2018. Acesso em: 13 set 2023.

TANG, Y.; TAN, J.; HARADA, T. Learning agile locomotion via adversarial training. In: **2020 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)**. [*S. l.*: *s. n.*], 2020. p. 6098–6105. Acesso em: 13 ago 2024.

UNITY. **Unity ML-Agents Toolkit Documentation**. 2020. Disponível em: https://github.com/Unity-Technologies/ml-agents/blob/release_14_docs/docs/Readme.md. Acesso em: 13 Abr 2024.

URMANOV, M.; ALIMANOVA, M.; NURKEY, A. Training unity machine learning agents using reinforcement learning method. In: IEEE. **2019 15th International Conference on Electronics, Computer and Computation (ICECCO)**. [*S. l.*], 2019. p. 1–4.

VINYALS, O.; BABUSCHKIN, I.; CZARNECKI, W. M.; MATHIEU, M.; DUDZIK, A.; CHUNG, J.; CHOI, D. H.; POWELL, R.; EWALDS, T.; GEORGIEV, P. *et al.* Grandmaster level in starcraft ii using multi-agent reinforcement learning. **Nature**, Nature Publishing Group, v. 575, n. 7782, p. 350–354, 2019. Acesso em: 13 set 2023.

WON, J.; GOPINATH, D.; HODGINS, J. Physics-based character controllers using conditional vaes. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 41, n. 4, 2022. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/3528223.3530067. Acesso em: 13 ago 2024.

WROTEK, P.; JENKINS, O. C.; MCGUIRE, M. Dynamo: dynamic, data-driven character control with adjustable balance. In: **Proceedings of the 2006 ACM SIGGRAPH symposium on Videogames**. [*S. l.*: *s. n.*], 2006. p. 61–70.

XIE, K.; WANG, T.; IQBAL, U.; GUO, Y.; FIDLER, S.; SHKURTI, F. Physics-based human motion estimation and synthesis from videos. In: **Proceedings of the IEEE/CVF International Conference on Computer Vision**. [*S. l.*: *s. n.*], 2021. p. 11532–11541.

YANG, C.; YUAN, K.; HENG, S.; KOMURA, T.; LI, Z. Learning natural locomotion behaviors for humanoid robots using human bias. **IEEE Robotics and Automation Letters**, IEEE, v. 5, n. 2, p. 2610–2617, 2020.

YANG, Z.; YIN, Z. Efficient hyperparameter optimization for physics-based character animation. **Proc. ACM Comput. Graph. Interact. Tech.**, Association for Computing Machinery, New York, NY, USA, v. 4, n. 1, apr 2021. Disponível em: https://doi.org/10.1145/3451254.

YIN, Z.; YANG, Z.; PANNE, M. V. D.; YIN, K. Discovering diverse athletic jumping strategies. **ACM Trans. Graph.**, ACM, New York, NY, USA, v. 40, n. 4, 2021. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/3450626.3459817. Acesso em: 13 ago 2024.

YOUSSEF, A. E.; MISSIRY, S. E.; EL-GAAFARY, I. N.; ELMOSALAMI, J. S.; AWAD, K. M.; YASSER, K. Building your kingdom imitation learning for a custom gameplay using unity ml-agents. In: IEEE. **2019 IEEE 10th Annual Information Technology, Electronics and Mobile Communication Conference (IEMCON)**. [*S. l.*], 2019. p. 0509–0514.

YU, W.; TURK, G.; LIU, C. K. Learning symmetric and low-energy locomotion. **ACM Trans. Graph.**, Association for Computing Machinery, New York, NY, USA, v. 37, n. 4, jul 2018. ISSN 0730-0301. Disponível em: https://doi.org/10.1145/3197517.3201397. Acesso em: 23 ago 2023.

ZORDAN, V. B.; HODGINS, J. K. Motion capture-driven simulations that hit and react. In: **Proceedings of the 2002 ACM SIGGRAPH/Eurographics Symposium on Computer animation**. [*S. l.*: *s. n.*], 2002. p. 89–96.