UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Felipe Reis Valente

Um framework multi-agente de aplicações de rastreamento corporal aplicado à reabilitação física e neurofuncional

São João del-Rei

2022

UNIVERSIDADE FEDERAL DE SÃO JOÃO DEL-REI

Felipe Reis Valente

Um framework multi-agente de aplicações de rastreamento corporal aplicado à reabilitação física e neurofuncional

Proposta de Dissertação de Mestrado submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de São João del-Rei, como requisito necessário à obtenção do grau de Mestre em Ciência da Computação.

Orientador: Prof. Dr. Diego Roberto Colombo Dias

Coorientador: Prof. Dr. Elder José Reioli Cirilo

Universidade Federal de São João del-Rei – UFSJ

Mestrado em Ciência da Computação

São João del-Rei

2022

Felipe Reis Valente

Um framework multi-agente de aplicações de rastreamento corporal aplicado à reabilitação física e neurofuncional

Proposta de Dissertação de Mestrado submetida ao Programa de Pós-graduação em Ciência da Computação da Universidade Federal de São João del-Rei, como requisito necessário à obtenção do grau de Mestre em Ciência da Computação.

Trabalho aprovado. São João del-Rei, 30 de outubro de 2022:

Prof. Dr. Diego Roberto Colombo Dias
Orientador

Prof. Dr. Elder José Reioli Cirilo
Coorientador

Prof. Dr. Vinícius Humberto Serapilha Durelli
Titular Interno

Prof. Dr. Rafael Serapilha Durelli
Titular Exteno

São João del-Rei
2022

Ficha catalográfica elaborada pela Divisão de Biblioteca (DIBIB)
e Núcleo de Tecnologia da Informação (NTINF) da UFSJ,
com os dados fornecidos pelo(a) autor(a)

V154f

Valente, Felipe Reis.
Um framework multi-agente de aplicações de rastreamento corporal aplicado à reabilitação física e neurofuncional / Felipe Reis Valente ; orientador Diego Roberto Colombo Dias; coorientador Elder José Reioli Cirilo. -- São João del-Rei, 2022.
81 p.

Dissertação (Mestrado - Programa de Pós-Graduação em Ciência da Computação) -- Universidade Federal de São João del-Rei, 2022.

1. Rastreamento Corporal. 2. Framework. 3. Multi agente. 4. Reabilitação. I. Dias, Diego Roberto Colombo, orient. II. Cirilo, Elder José Reioli, co orient. III. Título.

Agradecimentos

Agradeço à Deus pela dádiva da vida e por ter me iluminado durante toda essa jornada colocando pessoas importantes ao meu lado que me ajudaram e, certamente, sem elas eu não teria conseguido.

Agradeço imensamente ao meu orientador Diego Dias e ao meu coorientador Elder Cirilo pela paciência e dedicação que tiveram e também pelo profissionalismo e amizade que firmamos no decorrer dessa jornada me permitindo sair engrandecido de conhecimento.

A todos os professores, colegas do Mestrado, e ao Elvis Ribeiro, ao Tiago Trotta, ao Luis Guilherme Rodrigues e ao Marcelo Guimarães que fazem parte do grupo de pesquisa *BTS* e que compartilharam conhecimento, amizade e força.

Aos meus amigos que me ajudaram ao longo do mestrado, em especial o Alexandre Almeida e Nayara Pena, pela enorme força e apoio.

À minha família que é tudo para mim e que esteve sempre comigo.

Agradeço também a Fiama Souza que me deu força e coragem, me apoiando nos momentos de dificuldades.

Assim, agradeço a todos que me ajudaram de alguma forma no desenvolvimento desta pesquisa motivando-me intelectual e emocionalmente.

Resumo

O Acidente Vascular Cerebral (AVC) é uma das doenças que mais incapacita nos dias de hoje, de modo que a investigação de métodos voltados à reabilitação de pacientes pós-AVC é algo de extrema importância. Para tanto, benefícios significativos podem ser alcançados por meio da utilização de sistemas de rastreamento corporal e ambientes virtuais para este fim. Contudo, o desenvolvimento de tais aplicações envolve um grande conjunto de requisitos, tais como, a construção de ambientes virtuais, os dispositivos de interação e o armazenamento e processamento dos dados coletados em sessões de reabilitação. Neste trabalho, é apresentada a proposta e implementação do *BTS framework* sendo do tipo multi-agente e caixa cinza, com o intuito de abstrair as dificuldades acerca do desenvolvimento de soluções que envolvam utilização de dispositivos para rastreamento corporal e ambientes virtuais na reabilitação neuromotora e neurofuncional.

Palavras-chaves: rastreamento corporal, *framework*, multi-agente e reabilitação.

Abstract

Stroke is one of the most disabling diseases, so investigating methods for rehabilitating post-stroke patients is of utmost importance. Significant benefits can be achieved through the use of body tracking systems and virtual environments for this purpose. However, the development of such applications involves a large set of requirements, such as the construction of virtual environments, interaction devices, and the storage and processing of data collected during rehabilitation sessions. In this work, a proposal and implementation of the BTS framework is presented, being of the multi-agent type and gray box, aiming to abstract way the difficulties involved in developing solutions that involve the use of body-tracking devices and virtual environments in neuromotor and neurofunctional rehabilitation.

Keywords: body tracking, framework, multi-agent and rehabilitation.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

3D	Três dimensões
ACM	*Association for Computing Machinery*
API	*Application Programming Interface*
AVC	Acidente Vascular Cerebral
BLE	*Bluetooth Low Energy*
BSN	*Biomechanics Sensor Node*
DTO	*Data Transfer Object*
EMG	Sensores de eletromiografia
IEEE	*Institute of Electrical and Electronics Engineers*
REST	*Representational State Transfer*
RGB	*Red,Green,Blue*
ReBase	*Rehabilitation Database*
SDK	*Software Development Kit*
SVM	*Support Vector Machine*
UDP	*User Datagram Protocol*
UML	*Unified Modeling Language*
USB	*Universal Serial Bus*
UTI	Unidade de Terapia Intensiva
UUID	*Universally Unique Identifier*
VGA	*Video Graphics Array*
XMML	*Extensible Markup Language*

Sumário

1 Introdução

A reabilitação consiste num processo de terapia que tem como objetivo recuperar parcialmente ou complemente as capacidades motoras de uma pessoa doente. Eventos como o Acidente Vascular Cerebral (AVC) e doenças degenerativas limitam a mobilidade, assim como o avançar da idade ou acidentes que a pessoa possa sofrer (WHITALL et al., 2000). No entanto, a reabilitação tradicional possui algumas limitações, sendo uma das principais limitações das sessões de fisioterapia com aparelhos tradicionais (ex: andarilhos e muletas) é que eles não permitem uma análise objetiva da evolução do quadro do paciente (CIFUENTES et al., 2014).

O profissional de fisioterapia exerce papel primordial durante a realização das sessões, contudo, há métricas de mobilidade e progresso do paciente que não são fáceis de serem observadas visualmente pelos profissionais, tornando assim subjetiva a avaliação do paciente (POSTOLACHE, 2015). Métricas menos subjetivas necessitam de equipamentos de reabilitação com sensores que permitam monitorar os movimentos com mais precisão, tendo um *feedback* mais detalhado para o fisioterapeuta. Novas abordagens terapêuticas podem auxiliar na reabilitação, tal como o uso de jogos destinados a este propósito, capturando os movimentos por meio de sensores ou câmeras de profundidade, auxiliando em novas tarefas motoras funcionais e/ou cognitivas com utilização dos jogos virtuais (GORDON; ROOPCHAND-MARTIN; GREGG, 2012).

É possível aliar a tecnologia com a área da saúde utilizando jogos sérios (*serious games*), que são jogos feitos para atingir um objetivo relevante, ou seja, que extrapolam a ideia de entretenimento, oferecendo possibilidades de aprendizado e treinamento (BARNES; ENCARNAÇÃO; SHAW, 2009). Outra abordagem é o emprego de ambientes de Realidade Virtual e Aumentada, que, diferente dos jogos, não possuem elementos gamificados, mas permitem a simulação de atividades do cotidiano por meio de ambientes tridimensionais (BRANDÃO et al., 2020). Neste trabalho, os ambientes virtuais têm o papel de ajudar na reabilitação por meio de elementos como avatares (representação do usuário por um personagem digital), pontuação para objetivos alcançados, elementos lúdicos que permitam uma maior interação do usuário em um ambiente com *feedback* em tempo real para o paciente e também para o fisioterapeuta.

O meio de interação é muito importante no contexto de reabilitação, sendo assim, a utilização de câmeras de profundidade têm papel central na captação dos movimentos do corpo humano. Por meio de sensores infravermelho, os dados das articulações são capturados e podem ser armazenados para posterior reprodução (KITSUNEZAKI et al., 2013; BARBOSA, 2021). Existem também os sensores inerciais para esse propósito que

capturam dados que permitem inferir a posição e orientação de objetos rastreados, tais como: acelerômetros, magnetômetros e giroscópios (RODRIGUEZ-MARTIN et al., 2013). Sendo assim, com base nos dados coletados do paciente é possível verificar se os movimentos estão sendo executados de forma apropriada (GAMA et al., 2012), auxiliando o trabalho do fisioterapeuta.

Este trabalho visa dar início ao estudo e desenvolvimento de um *framework* multiagente direcionados para a área da reabilitação motora, para criar aplicações interativas que auxiliem o fisioterapeuta no seu cotidiano por meio de utilização de câmeras de profundidade e sensores de movimento. O *framework* proporciona vantagens para os desenvolvedores, como facilitar o reúso, padrões de codificação e diminuir o tempo para desenvolver novas funcionalidades (SILVA, 2000).

A dissertação está organizada da seguinte forma: no Capítulo 2 é apresentado o referencial teórico necessário para construir o *framework*, mostrando os padrões de projetos e dispositivos de captura de movimento; no Capítulo 3 são demonstrados os trabalhos prévios realizados pelo grupo de pesquisa que iniciou as investigações na área; no Capítulo 4 são apresentados os trabalhos relacionados; no Capítulo 5 o *framework* é proposto descrevendo suas classes e métodos; no Capítulo 6 é mostrado o modo de utilização do *framework*; e o Capítulo 7 apresenta as conclusões do trabalho.

Dois artigos foram publicados ao longo do desenvolvimento da dissertação: *A multi-agent body tracking application framework applied to physical and neurofunctional rehabilitation* (VALENTE et al., 2022b) e *A framework for neuromotor and neurofunctional rehabilitation using multi-agent systems* (VALENTE et al., 2022a).

2 Referencial Teórico

2.1 *Framework*

Há muitas definições sobre o que é um *framework*, segundo Gamma et al. (1995) um *framework* é um conjunto de classes que cooperam entre si provendo assim um projeto reutilizável para um domínio específico de classes de sistema.

A utilização de um *framework* no domínio da Engenharia de Software fornece ao desenvolvedor uma estrutura modular específica para um determinado contexto, permitindo o reúso de componentes e o acoplamento de tecnologias novas e criando uma nova aplicação (BRAGA, 2002).

Analisando a forma de reúso, o *framework* pode ser classificado de três formas: caixa branca (*white box*), caixa cinza (*gray box*) e caixa preta (*black box*). O comportamento de um *framework* de aplicação normalmente é feito adicionando-se métodos às subclasses de uma ou mais de suas classes.

O *framework* caixa branca necessita de grande conhecimento para sua implementação, ou seja, precisa de uma curva de aprendizado alta e documentação robusta. Ele permite que o desenvolvedor tenha acesso ao código fonte. No *framework* caixa branca o reúso acontece por herança, assim, o desenvolvedor deve criar subclasses das classes abstratas contidas no *framework* para elaborar as aplicações, para utilizá-lo, deve-se entender os detalhes de como o *framework* funciona (EXTERKOETTER et al., 2003).

Entretanto, no *framework* caixa preta o reúso ocorre por composição, que é o desenvolvedor combinar diversas classes concretas existentes para construir a aplicação. Assim, o desenvolvedor deve entender apenas a interface para poder utilizar o *framework* (YASSIN; FAYAD, 2000).

O *framework* caixa cinza é um intermediário entre a caixa branca e a caixa preta, tendo em vista que, um *framework* baseado em herança (caixa branca) é mais fácil de desenvolver e possui uma estrutura mais flexível, sendo, contudo, mais difícil de utilizá-lo (EXTERKOETTER et al., 2003). Já o baseado em composição de objetos (caixa preta) é mais simples de estender para uma nova aplicação, entretanto é mais difícil de desenvolvê-lo. Nesse mesmo sentido Silva (2006) afirma que o *framework* caixa cinza permite a extensão tanto em termos de herança quanto em termos de composição, dependendo da necessidade da aplicação, se beneficiando, portanto, das duas técnicas.

Um passo importante para elaboração do *framework* é o levantamento de requisitos funcionais e requisitos não funcionais. Os requisitos funcionais tratam de funções

que o sistema deve fornecer de como o sistema tem que reagir de acordo com entradas específicas em determinadas situações. Já os requisitos não funcionais são restrições aos serviços ou funções oferecidos pelo sistema, como restrições de tempo, padrões, usabilidade, entre outros (PRESSMAN; MAXIM, 2016). Faz-se necessário verificar quais eventos e funcionalidades são comuns a todas as interfaces, assim, o *framework* precisa fazer essa integração de forma transparente, deixando o desenvolvedor responsável pelo desenvolvimento da aplicação, não se preocupando na forma que o domínio se comporta, banco de dados, aprendizado de máquina, entre outros.

2.2 Padrões de projetos

Padrões de projeto (*design patterns*) podem ser vistos como soluções típicas para problemas comuns em projetos de software. Segundo Christopher et al. (1977), cada padrão descreve um problema que ocorre repetidas vezes em nosso ambiente e então descreve o núcleo da solução para esse problema, de tal forma que você pode usar esta solução um milhão de vezes, sem nunca o fazer da mesma maneira duas vezes, embora ele esteja se referindo a Engenharia Civil, o mesmo vale para padrões de projetos orientado a objetos (GAMMA et al., 1995).

Os padrões de projetos foram feitos para serem reutilizados, evitando que soluções sejam desenvolvidas do principio. Segundo Gamma et al. (1995), os bons projetistas sabem que não devem resolver cada problema do zero. Ao invés disso, é interessante utilizar soluções que já funcionaram previamente em outros projetos.

A utilização dos padrões de projetos facilita a manutenção, uma vez que o projeto fica melhor documentado e seguindo um padrão conhecido; aumenta a escalabilidade do projeto; e permite a reutilização permitindo assim uma maior produtividade e melhor qualidade do código (LARSEN; AARSETH, 2006).

Três padrões foram utilizados na construção do *BTS framework*: o *Abstract Factory*, o *Repository* e o *Strategy*. O padrão *Abstract Factory* tem como objetivo fornecer uma interface para criação de famílias de objetos relacionados ou dependentes sem especificar suas classes concretas (GAMMA et al., 1995). Já o padrão *Repository* é uma forma de encapsular detalhes da infraestrutura de acesso a dados, fazendo um intermédio entre as camadas mapeadoras de dados e as camadas de domínio (FOWLER, 2009). O padrão de projeto comportamental *Strategy* define uma família de algoritmos, coloca os algoritmos em classes separadas e faz com que eles sejam intercambiáveis (FERREIRA; RESENDE; COSTA, 2012).

2.3 Sistema multi-agente

Um sistema multi-agente é definido por ser um sistema onde vários agentes atuam em conjunto sobre um ambiente para resolver um problema (WOOLDRIDGE, 2009). Os sistemas multi-agentes estão se tornando um poderoso paradigma de Engenharia de Software. São inúmeros casos de sucesso na indústria e na academia, em diversos tipos de aplicações (MUBARAK, 2008).

É interessante utilizar sistemas multi-agentes, pois há vários benefícios, por exemplo: quando necessitamos manter a autonomia das subpartes e quando as interações entre os agentes são complexas e não há facilidade em detalhar o problema como um todo (JENNINGS; WOOLDRIDGE, 2012).

Um agente de software é definido como uma entidade que funciona em um dado ambiente de forma autônoma e contínua, sendo capaz de operar no ambiente de forma inteligente sem necessitar de intervenção humana constante. Ao operar em um período consideravelmente grande, o agente pode aprender e também cooperar com outros agentes que se encontram no ambiente para atingir um determinando objetivo (SILVEIRA, 2001).

Um conjunto de agentes em um dado ambiente tem o objetivo de solucionar um problema que está além de suas capacidades individuais, logo, é necessário um orquestramento dos agentes para alcançar o objetivo (O'HARE; JENNINGS, 1996). A comunicação desse orquestramento acontece com envio de mensagens por protocolos de comunicação que operam por meio da rede, sendo importante a definição dos protocolos e a estratégia de orquestramento para que ocorra um funcionamento adequado dos métodos de comunicação (SILVEIRA, 2001).

Os sistemas multi-agentes podem ser reativos quando são compostos por uma grande quantidade de agentes mais simples. Esses não são inteligentes e funcionam na base da ação e reação, entretanto, globalmente eles são inteligentes, mas não individualmente. A inteligência está na cooperação entre eles e o ambiente, ou seja, não é preciso que o agente seja inteligente para obter um comportamento global inteligente, pode-se exemplificar com uma colônia de formigas, como um todo é bem estruturado e faz com que as formigas sejam mais perspicazes juntas (MAES, 1993).

Já os sistemas multi-agentes, quando são formados por agentes cognitivos, esses por sua vez são inteligentes, pois individualmente são dotados de noções de estados mentais, como intenção, crença, desejo, competências, propósitos, entre outros. Assim eles são mais inteligentes, podendo agir de acordo com seu conhecimento e habilidades, podendo negociar, planejar e ter atitudes mais interessantes para resolução de um problema (MAES, 1993).

2.4 Neuroreabilitação

A neuroreabilitação é uma forma de intervenção para tratar prejuízos cognitivos sem intervenção cirúrgica ou farmacológica, portanto trata-se de uma técnica não invasiva, muito válida também em pacientes com problemas de memória devido a sintomas da Doença de Alzheimer (BAHAR-FUCHS; CLARE; WOODS, 2013).

Essa forma de intervenção é possível pois o Sistema Nervoso Central não é uma estrutura rígida. Ele é uma estrutura flexível em que suas funções podem ser regeneradas. Mediante fisioterapia específica, é possível que o paciente recupere movimentos perdidos. O Sistema Nervoso Central possui uma grande adaptabilidade, mesmo em cérebros de adultos (OLIVEIRA; SALINA; ANNUNCIATO, 2001).

Dessa forma, tem-se que a neuroreabilitação é pautada em dois princípios. O primeiro princípio é que o encéfalo possui plasticidade e, consequentemente, capacidade de recuperação. O segundo é que o paciente ao longo do tempo tem condições de fazer ajustes de comportamento de acordo com as circunstâncias (TWAMLEY, 2010).

O AVC é uma das maiores causas de morbidades. Aproximadamente 80% dos pacientes que conseguem sobrevir a fase mais grave ficam com sequelas. A maioria dos pacientes conseguem recuperar a capacidade de andar, entretanto, de 30% a 66% dos pacientes não conseguem utilizar os membros superiores após sofrer um AVC (SILVA et al., 2003).

A aplicação de jogos com rastreamento corporal em centros de reabilitação para ajudar na terapia do paciente é uma crescente e também uma tendência, pois com o barateamento da tecnologia, facilita até mesmo levar os equipamento para a residência do paciente (BRANDÃO et al., 2020; RIBEIRO, 2021; BARBOSA, 2021).

Dessa forma, é possível que o paciente faça a fisioterapia em sua casa, enquanto os dados são capturados e enviados para o fisioterapeuta, para posterior análise. Além disso, facilita o uso em lugares remotos e de difícil acesso, como na área rural, necessitando apenas de internet para enviar os dados.

2.5 *Unity*

O *Unity* é um motor de jogos multiplataforma proprietário criado pela *Unity Technologies*. Pode ser utilizado no *Windows*, *Linux* e *macOS*. Sua utilização é permitida para fins pessoais e educacionais de forma gratuita (UNITY, 2021b).

Lançado no ano de 2005, o *Unity* possui uma grande portabilidade, permitindo que a aplicação seja feita apenas uma vez e seja facilmente portada para inúmeras plataformas, tais como: (*Windows*, *Linux*, *macOS*, *iOS*, *Android*, *Playstation*, *Xbox*, *Nintendo*

Switch, *tvOS*, *Microsoft HoloLens*, *Oculus Rift*, entre outros) (UNITY, 2021a). O fato do *Unity* rodar em diversos dispositivos proporciona uma fácil adequação às necessidades e limitações tecnológicas de qualquer clinica de reabilitação. A Figura 1 ilustra a interface gráfica do *Unity* onde foi desenvolvido o *framework* proposto neste trabalho.

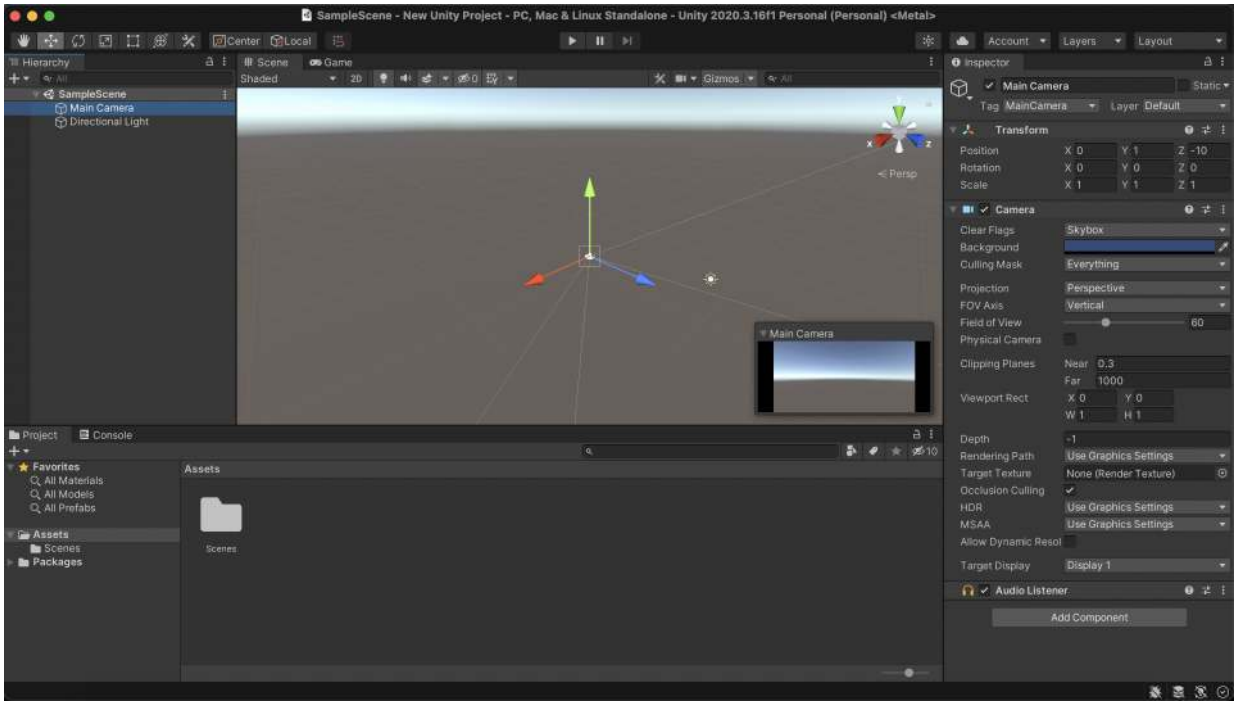


Figura 1 – Interface do *Unity* versão 2020.3.16f1

2.6 Rastreamento Corporal

O corpo humano é composto pelo sistema esquelético e pelo sistema articular. Os ossos se conectam por meio das articulações, sendo que as articulações funcionam como eixos e os ossos como alavancas, favorecendo assim a mobilidade do corpo humano (GRAAFF, 2003).

O rastreamento corporal é responsável por capturar as posições X, Y e Z de cada articulação. O número de articulações rastreadas depende do dispositivo empregado. O *Kinect*, por exemplo, realiza o rastreamento de 20 articulações a uma frequência de 30 Hz. O corpo humano possui mais articulações, entretanto, neste trabalho foram adotadas as 20 articulações rastreadas pelo *Kinect*, conforme a (Figura 2).

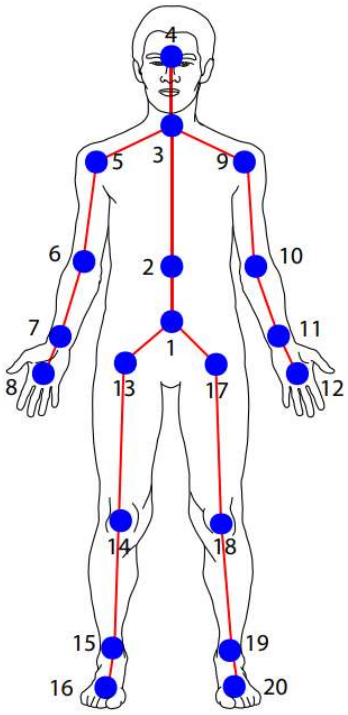


Figura 2 – Padrão de articulações utilizado (MOTIIAN et al., 2015)

Pode-se dividir o corpo humano em três planos: Coronal, Transversal e Sagital, conforme a Figura 3.

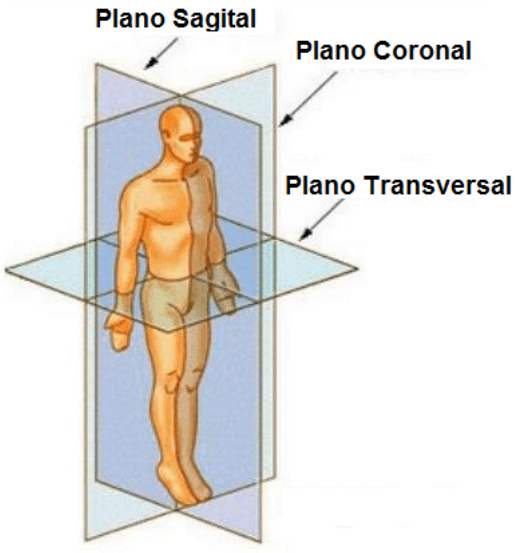


Figura 3 – Imagens do corpo humano nas secções Coronal, Transversal e Sagital (DHAWAN; HUANG; KIM, 2008)

O Plano Coronal é o plano que passa sobre a sutura coronal do crânio, dividindo o corpo em posterior e anterior; por sua vez o Plano Transversal divide o corpo em duas partes, uma mais perto do solo como inferior e mais distante como superior; e o Plano

Sagital podemos definir como o plano que passa sobre a sutura sagital do crânio, dividindo o corpo em esquerdo e direito (DHAWAN; HUANG; KIM, 2008).

Tradicionalmente, a avaliação feita pelo fisioterapeuta envolve mensurar, apalpar e verificar o paciente de forma manual, diferentemente dos modelos computacionais, que estão sendo usados como uma forma dinâmica de fazer essa avaliação.

2.6.1 Rastreamento Óptico

A Visão Computacional é uma área de pesquisa que tem como objetivo simular a visão humana. A máquina recebe como entrada uma imagem, faz a extração, caracterização e interpretação de informações do mundo real por meio da imagem captada (GAVRILA, 1999). Fazendo uso da Visão Computacional, a máquina é capaz de fazer o rastreamento corporal, pelo fato dela ser capaz de processar e interpretar as imagens capturadas das sessões de fisioterapia (FORSYTH; PONCE, 2012).

O rastreamento óptico reside no fato do usuário não precisar usar nenhum fio ou transmissor no seu corpo, utilizando câmera de vídeo, o que oferece uma boa liberdade ao usuário (MACHADO; CARDOSO, 2006). Além disso, permite também a captura de múltiplos usuários. As áreas de captura podem ser maiores que de outros sistemas de rastreamento.

Assim como vantagens podem ser atribuídas ao rastreamento óptico, várias desvantagens também lhe são. Pode-se citar como desvantagens o pré-processamento, que demanda maior poder de processamento; os problemas de oclusão, que ocasionam perda de rastreamento; assim como a iluminação ambiente, que precisa ser controlada.

2.6.2 Rastreamento Inercial

O rastreamento inercial é um tipo de rastreamento que utiliza sensores de medição inerciais, tal como acelerômetros, magnetômetros e giroscópios. Com os dados capturados por esses sensores, obtêm-se a velocidade angular e a aceleração linear, reconstruindo as informações referentes à posição e orientação dos objetos rastreados (RODRIGUEZ-MARTIN et al., 2013).

O que proporcionou a utilização desses dispositivos para rastreamento corporal foi o avanço da tecnologia, permitindo a redução do tamanho físico e preço de aquisição do dispositivo. Entretanto, a miniaturização também trouxe problemas como a diminuição da precisão. Diferente do rastreamento óptico o rastreamento inercial não sofre com problemas de oclusão.

2.7 Abordagens de Rastreamento Corporal

2.7.1 *Arduino*

O *Arduino* (ARDUINO, 2022) é um microcontrolador de código aberto, barato, que consome pouca energia elétrica. O desenvolvimento de sistemas é feito na linguagem C/C++. O microcontrolador pode ser conectado ao computador por meio do *Universal Serial Bus* (*USB*) e desta forma ser programado, podendo trabalhar tanto com sinais analógicos quanto digitais, facilitando assim o trabalho com sensores.

Um exemplo é a utilização de sensores ultrassônicos (HC-SR04) com o *Arduino*, que permite que se faça o rastreamento corporal, conforme apresentado na Figura 4.

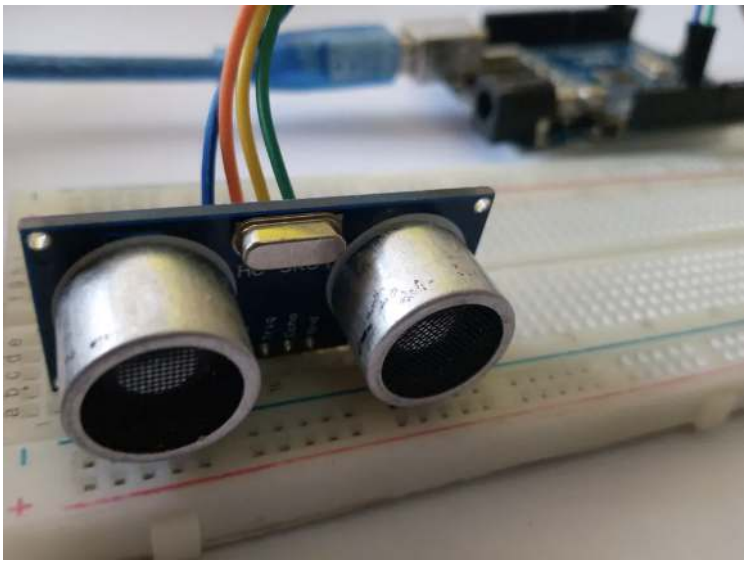


Figura 4 – *Arduino* com sensor HC-SR04 (ARDUINO PROJECT HUB, 2022)

2.7.2 *Biomechanics Sensor Node*

O dispositivo *Biomechanics Sensor Node* (*BSN*) (BRANDÃO et al., 2020) é menor que as soluções baseadas em *Arduino* e foi criado devido ao requisito de tamanho de outros dispositivos presentes no mercado. Dessa forma, foi criada uma pulseira que contorna esse problema, além de possuir bateria e uma melhor ergonomia para utilização, conforme a Figura 5.

Figura 5 – Dispositivo de interação *BSN*

A comunicação do dispositivo é feita por meio do uso de *Bluetooth Low Energy* (*BLE*). O dispositivo pode ser utilizado anexado às articulações do usuário, e, devido a sua forma ergonômica, oferece um maior conforto. O *BSN* possui diversos sensores, como giroscópio, acelerômetro e bússola, que combinados conseguem fazer o rastreamento corporal do usuário (BRANDÃO et al., 2020).

2.7.3 *Mediapipe*

O *Mediapipe* é um *framework* de código aberto desenvolvido pela *Google*, feito para ser utilizado em dispositivos móveis e *desktops*, tendo como finalidade facilitar o desenvolvimento ao se trabalhar com detecção de gestos efetuados pelo usuário, conforme a Figura 6.

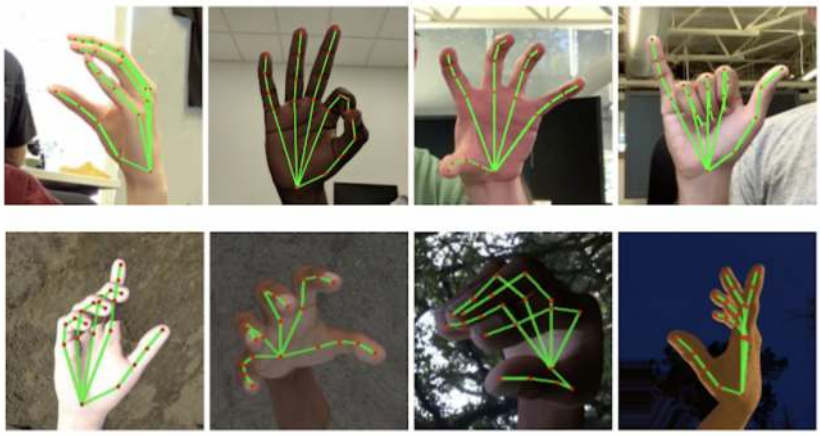


Figura 6 – Rastreamentos com *Mediapipe* (GOOGLE BLOG, 2019)

É possível também a utilização do *Mediapipe* junto a técnicas de aprendizado de máquina para detectar e rastrear objetos, faces e até mesmo fios de cabelo de uma pessoa (GOOGLE BLOG, 2019).

As técnicas de aprendizado de máquina alcançam um bom desempenho em tempo real até mesmo em dispositivos mais limitados, como dispositivos móveis. Há duas abordagens utilizadas pelo *Mediapipe*: primeiro, ele delimita uma caixa ao redor do objeto em uma imagem completa, identificando o objeto a ser trabalhado e depois de encontrado o objeto, a segunda abordagem trabalha somente na região delimitada para identificar determinados pontos-chave.

2.7.4 *Kinect*

O *Kinect* é um dispositivo de rastreamento corporal desenvolvido e mantido pela *Microsoft*, conforme a Figura 7. Ele é capaz de capturar as articulações do usuário, assim como a sua voz (CORREA et al., 2012).

Inicialmente o *Kinect* foi desenvolvido para ser utilizado em jogos eletrônicos no *console Xbox360*, mas com a popularização do dispositivo, hoje ele está sendo utilizado na área de Medicina, Robótica, Fisioterapia, Educação Física entre outras (MACKNOJIA et al., 2013; DUARTE; POSTOLACHE; SCHARCANSKI, 2014; VAGHETTI et al., 2012; KURILLO et al., 2014). O *Kinect* possui ótima documentação e um conceituado *Software Development Kit* (*SDK*) que facilita o desenvolvimento das mais diversas aplicações.

Figura 7 – Dispositivo *Kinect* (WEBB; ASHLEY, 2012)

Ele é composto por uma câmera *Red,Green,Blue* (RGB) e uma outra câmera infravermelho, ambas com resolução de 640x480 *pixels* do tipo *Video Graphics Array* (*VGA*), com taxa de 30 quadrados por segundo. Além disso, possui microfones, um acelerômetro de três eixos e um giroscópio para a inclinação do dispositivo (STOWERS; HAYES; BAINBRIDGE-SMITH, 2011).

O sensor de profundidade é feito mediante um padrão infravermelho que é projetado e a deformação é medida, assim, é possível reconstruir o espaço tridimensional com precisão na casa de um milímetro, sendo possível capturar a profundidade do usuário entre 0,8m e 3,5m.

Conforme o exposto, tem-se dois tipos de rastreamentos corporais apresentados: o rastreamento óptico e o inercial. No escopo deste projeto, pode-se agrupar as abordagens da seguinte maneira: rastreamento óptico com a utilização do *Kinect* e do *MediaPipe*, e rastreamento inercial com *Arduino* e *BSN*.

3 Projetos anteriores

Alguns trabalhos foram desenvolvidos pelo grupo de pesquisa, tal como o *ReBase* (BARBOSA, 2021) que é um sistema de aquisição e processamento de dados para reabilitação neuromotora. O Rastreamento Corporal por meio de Sensores Biomecânicos (RIBEIRO, 2021), também desenvolvido pela equipe, foi criado um *asset* para *Unity* para desenvolver aplicações utilizando como meio de interação o *BSN*. Esses trabalhos foram desenvolvidos pelo grupo de pesquisa antes da criação do *BTS framework*.

3.1 *ReBase*: sistema de aquisição, armazenamento e gerenciamento de dados de reabilitação neuromotora

O *ReBase* foi feito para construir *datasets* de movimentos independentes do dispositivo de rastreamento corporal. Com base nos *datasets* criados, é possível fazer uma análise posterior ao aplicar técnicas de aprendizado de máquina e conseguir realizar inferências, auxiliando o trabalho do fisioterapeuta.

É importante a construção de *datasets*, pois não existem opções na literatura no contexto de reabilitação neuromotora e neurofuncional, mesmo havendo necessidade de analisar os movimentos corporais dos pacientes. Os dados são armazenados em um banco de dados não-relacional, implementado utilizando o *Apache Cassandra*.

Com o objetivo de validar o *ReBase* foram desenvolvidas três aplicações para *desktop* utilizando *Unity* e a linguagem de programação C#: o *Kinect Session Recorder*, o *ReBase Session Player* e o *ReBase Session Manager*.

O *Kinect Session Recorder* é um gravador de sessões de fisioterapia e tem como finalidade popular o banco de dados, capturando os dados das articulações dos usuários por meio do *Kinect*, conforme apresentado na Figura 8. Importante ressaltar que foi utilizado o *Kinect*, mas fazendo adaptações poderiam ser usados outros dispositivos.

Figura 8 – Tela principal do *Kinect Session Recorder* durante gravação

O *ReBase Session Player* foi feito para reproduzir as sessões armazenadas no banco de dados *ReBase*, conforme apresentado na Figura 9. Devido à padronização dos dados na hora que são armazenados, essa aplicação é capaz de reproduzir os dados que foram salvos independentemente dos dispositivos que foram utilizados para gravar.

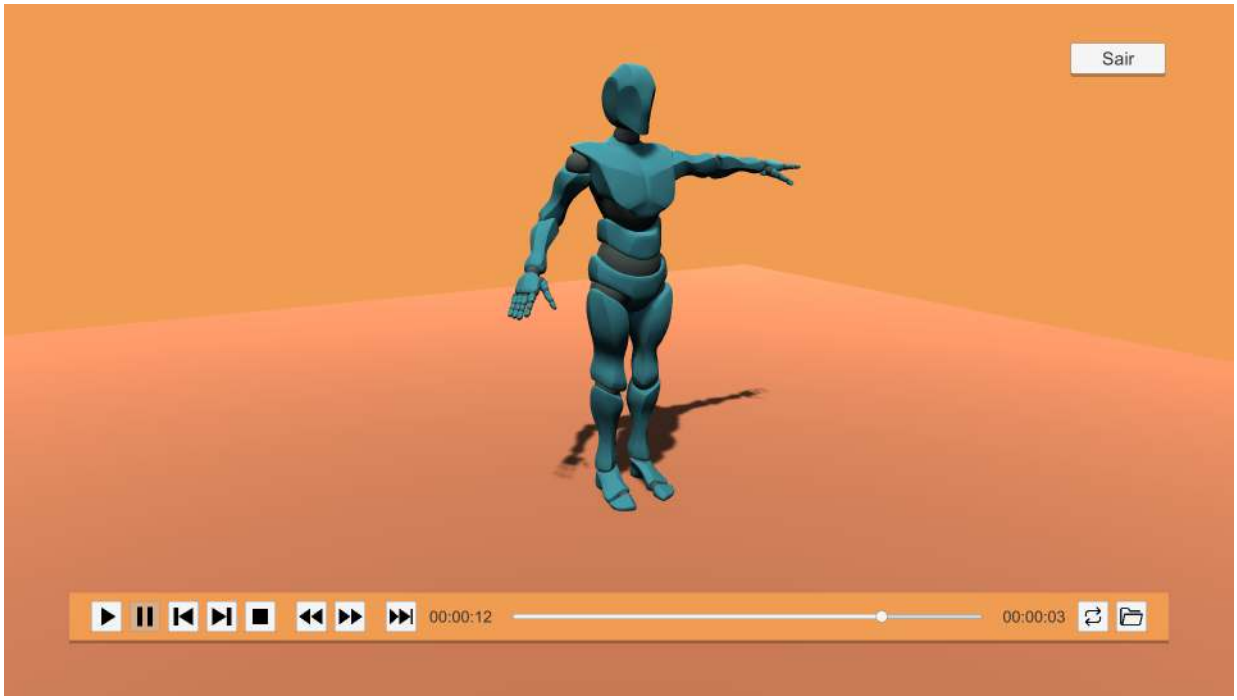


Figura 9 – Tela principal do *Kinect Session Player*

O *ReBase Session Manager* foi feito para alterar, remover e visualizar sessões

armazenadas no *ReBase*, conforme apresentado na Figura 10.

Figura 10 – Tela principal do *Kinect Session Manager*

3.2 Rastreamento Corporal por meio de Sensores Biomecânicos

Nesse trabalho foi criado um modelo de interação com o paciente baseado no uso de sensores inerciais, com o foco na utilização da interface natural do usuário para realizar interações com ambientes virtuais voltados à reabilitação neuromotora. Foi criado um *asset* para usar o dispositivo *BSN* com o motor de jogo *Unity* e a linguagem de programação C#, com as seguintes características:

- **Conexão de múltiplos *BSNs***: É necessário para capturar várias articulações do corpo humano, assim, é necessário sincronizar diversos *BSNs*;
- **Interface de fácil utilização**: Uma interface que seja fácil e intuitiva para prover o pareamento dos dispositivos *BSN*; e
- **Configuração remota**: A possibilidade de fazer as configurações de um outro dispositivo, podendo ser um dispositivo móvel ou *desktop*.

Para validar o trabalho, alguns testes foram realizados utilizando algumas aplicações desenvolvidas em *Unity*, tais como: *Immersive Brainn Puzzle* (ARAÚJO et al., 2021), Simulador de Pênalti, entre outros.

O *Immersive Brain Puzzle* foi feito para auxiliar no processo de reabilitação de pacientes que sofreram AVC, ainda na Unidade de Terapia Intensiva (UTI), sabendo que

as primeiras horas são cruciais para a reabilitação. Foi desenvolvido um jogo de quebra-cabeça virtual em que o paciente deve rotacionar partes das figuras para completar a imagem original, utilizando um *BSN* fixo no punho, conforme apresentado na Figura 11.

Figura 11 – Usuário interagindo com o *Immersive Brainn Puzzle*

O Simulador de Pênalti possui um goleiro e o cobrador de pênalti em um campo de futebol virtual, conforme apresentado na Figura 12. Foi desenvolvido para demonstrar o acelerômetro do *BSN*. No jogo, o usuário controla a perna e a tíbia do avatar com dois *BSNs*: um na parte superior da perna, um pouco acima do joelho e o outro na parte inferior, na tíbia.

Figura 12 – Usuário interagindo com o Simulador de Pênalti

3.3 Desafios enfrentados

A grande motivação para a criação do *BTS framework* foram os problemas enfrentados nos trabalhos do grupo de pesquisa. Vários problemas foram constatados ao longo do desenvolvimento dos projetos. Um dos problemas foi a falta de reúso do código, ocorrendo muito retrabalho por parte dos desenvolvedores.

Para integrar o *ReBase*, a outros projetos também não foi fácil, visto que eram projetos diferentes, em que não se pensou de forma prévia a integração. Foi feito um servidor que segue o padrão *REST* (*Representational State Transfer*) para tentar contornar esse problema, estabelecendo a comunicação do banco de dados com as aplicações.

Com a falta de flexibilidade dos projetos, foi constatada grande dificuldade em realizar pequenos ajustes, tendo que mudar o código e ficar recompilando o projeto como resultado de qualquer pequena alteração, perdendo muito tempo no desenvolvimento.

Como cada dispositivo de rastreamento corporal gera os dados em seu respectivo padrão, foi identificada a necessidade de ter uma classe responsável no projeto para fazer os devidos ajustes, conversões e, consequentemente, definir uma padronização dos dados.

Com a captura das sessões e persistência dos dados no *ReBase*, foi identificada a necessidade de aplicar aprendizado de máquina para conseguir realizar inferências nos dados coletados.

Diante do exposto acima, foi identificado a necessidade de criar um *framework*, visando aumentar a produtividade, diminuir a quantidade de código com erros, melhorar a manutenção, padronizar o projeto, reduzir o tempo de desenvolvimento, entre outras vantagens.

4 Trabalhos relacionados

Foi realizado uma revisão da literatura sendo as etapas essenciais: (i) definição das perguntas da pesquisa, (ii) conduzir a busca por estudos relevantes, (iii) triagem de artigos, e (iv) extração e mapeamento de dados.

Neste trabalho, foram buscadas respostas para as seguintes questões de pesquisa:

- Q1 Quais tipos de sistemas de software e hardware são utilizados no contexto de rastreamento corporal?
- Q2 Quais as patologias que são beneficiadas por técnicas de rastreamento corporal?
- Q3 Quais objetivos podem ser alcançados com o emprego de um *framework*?
- Q4 Qual o uso do aprendizado de máquina no domínio de técnicas de rastreamento corporal?

Para iniciar o estudo, precisamos definir a *string* de busca, assim como as bases de dados eletrônicas a serem pesquisadas. Com relação à *string*, foi utilizada uma combinação de palavras-chave: *body tracking*, *framework*, *multi-agent* e *rehabilitation*. A palavra-chave *machine learning* não foi utilizada na busca pois restringia muito os resultados, assim, verificamos artigos que utilizavam ou não aprendizado de máquina. Foi observado que utilizando a palavra-chave *multi-agent* não retornou nenhum artigo, portanto, foi utilizado apenas as palavras-chave: *body tracking*, *framework* e *rehabilitation*, conforme Tabela 1.

Palavra-chave	Estrutura
body tracking - *framework* - *rehabilitation*	*body tracking* AND *framework* AND *rehabilitation*

Tabela 1 – Estrutura de pesquisa elaborada

As base de dados eletrônica utilizadas foram: *Institute of Electrical and Electronics Engineers* (*IEEE*) (IEEE, 2021) e *Association for Computing Machinery* (*ACM*) (ACM, 2021).

A triagem visa determinar quais estudos são relevantes para responder as perguntas da pesquisa, portanto, para este fim, foram aplicados um conjunto de critérios de inclusão e exclusão para cada estudo recuperado. Dois critérios de inclusão devem ser satisfeitos para que o estudo possa ser incluído na revisão. Os critérios de inclusão são:

- *Frameworks* de rastreamento corporal.
- Aplicações voltadas a reabilitação de pacientes.

- Técnicas de análise com aprendizado de máquina.
- Uso de sistemas multi-agentes.

Os critérios de exclusão são:

- Sistemas de hardware que sejam muito complexos e caros.
- Publicações fora do escopo do trabalho.
- Relatórios técnicos, documentos que estão disponíveis na forma de resumos ou apresentações.
- Revisões sistemáticas da literatura e mapeamento de estudos.

4.0.1 Análise

A busca retornou 30 artigos na *IEEE* [1] e 43 artigos na *ACM* [2] utilizando a *string* de busca definida. Após aplicação dos critérios de exclusão, restaram 15 artigos na *IEEE* e 8 na *ACM*.

Os artigos selecionados, após os critérios estabelecidos no protocolo, foram mapeados em: Título, Autores, Base de Dados e Ano, conforme Tabela 2 e Tabela 3.

[1] https://ieeexplore.ieee.org
[2] https://dl.acm.org

Título	**Autores**	**Base de Dados**	**Ano**
A Framework to Enhance Assistive Technology Based Mobility Tracking in Individuals with Spinal Cord Injury	AMIRI, A. M.; SHOAIB, N.; HIREMATH, S. V. A	*IEEE*	2017
Accurate Estimation of Joint Motion Trajectories for Rehabilitation Using Kinect	SINHA, S.; BHOWMICK, B.; SINHA, A.; DAS, A.	*IEEE*	2017
Evaluating the Effect of Robot Feedback on Motor Skill Performance in Therapy Games	BROWN, L.; GARCÍA-VERGARA, S.; HOWARD, A. M.	*IEEE*	2015
EXTRA: Exercise Tracking and Analysis Platform for Remote-monitoring of Knee Rehabilitation	GWAK, M.; FAZELI, S.; ERSHADI, G.; SARRAFZADEH, M.; GHODSI, M.;AMINIAN, A.; SCHLECHTER, J. A. E	*IEEE*	2019
Fusing Data from Inertial Measurement Units and a 3D Camera for Body Tracking	DROBNJAKOVIC, F.; DOUANGPASEUTH, J. B.; GADEA, C.; HAIDER, M.; IONESCU, D.; IONESCU, B.; POON, L.	*IEEE*	2018
Human Body Contour Data Based Activity Recognition	MYAGMARBAYAR, N.; YUKI, Y.; IMAMOGLU, N.; GONZALEZ, J.; OTAKE, M.;YU, W.	*IEEE*	2013
Human Movement Quantification using Kinect for In-Home Physical Exercise Monitoring	GAUTHIER, S.; CRETU, A.-M.	*IEEE*	2014
Integration of Virtual Reality with an Omnidirectional Treadmill System for Multi-directional Balance Skills Intervention	METSIS, V.; SMITH, K. S.; GOBERT, D.	*IEEE*	2017
Landmark-Based Methods for Temporal Alignment of Human Motions	DIOS, P. F. de; CHUNG, P. W. H.; MENG, Q. L	*IEEE*	2014
Machine Learning Approach for Physiotherapy Assessment	DURVE, I.; GHUGE, S.; PATIL, S.; KALBANDE, D. M	*IEEE*	2019
Modeling Therapy Rehabilitation Sessions using Non-Invasive Serious Games	RAHMAN, M. A.; AHMED, M.; QAMAR, A.; HOSSAIN, D.; BASALAMAH, S.	*IEEE*	2019
Predictive Trajectory Estimation During Rehabilitative Tasks in Augmented Reality Using Inertial Sensors	HUNT, C. L.; SHARMA, A.; OSBORN, L. E.; KALIKI, R. R.; THAKOR, N. V.	*IEEE*	2018
Towards a Framework for Rehabilitation and Assessment of Upper Limb Motor Function Based on Serious Games	OÑA, E. D.; BALAGUER, C.; JARDÓN, A.	*IEEE*	2018
An Interactive 3D Health App with Multimodal Information Representation for Frozen Shoulder : Co-Creation and Evaluation with Patients	STÜTZ, T.; DOMHARDT, M.; EMSENHUBER, G.; HUBER, D.; TIEFENGRABNER,M.; MATIS, N.; GINZINGER, S.	*ACM*	2017

Tabela 2 – Lista de artigos selecionados para pesquisa

Título	Autores	Base de Dados	Ano
Lessons Learnt from Designing a Smart Clothing Telehealth System for Hospital Use	AGGARWAL, D.; HOANG, T.; PLODERER, B.; VETERE, F.; KHOT, R. A.;BRADFORD, M.	*ACM*	2020
MM-Fit: Multimodal Deep Learning for Automatic Exercise Logging across Sensing Devices	STRÖMBÄCK, D.; HUANG, S.; RADU, V.	*ACM*	2020
Multi-sensor Exercise-based Interactive Games for Fall Prevention and Rehabilitation	SANTOS, A.; GUIMARÃES, V.; MATOS, N.; CEVADA, J.; FERREIRA, C.; SOUSA,I.	*ACM*	2015
Physio@Home: Exploring Visual Guidance and Feedback Techniques for Physiotherapy Exercises	CHANG, R.; LAU, S.; SIM, K.; TOO, M. K	*ACM*	2015
Realizing a Low-latency Virtual Reality Environment for Motor Learning	WALTEMATE, T.; HÜLSMANN, F.; PFEIFFER, T.; KOPP, S.; BOTSCH, M.	*ACM*	2015
Supporting Seniors Rehabilitation through Videogame Technology: A Distributed Approach	MAGGIORINI, D.; RIPAMONTI, L. A.; ZANON, E.	*ACM*	2012
Videogame Technology to Support Seniors	MAGGIORINI, D.; RIPAMONTI, L.; SCAMBIA, A. V	*ACM*	2012

Tabela 3 – Lista de artigos selecionados para pesquisa (continuação)

4.0.2 Q1 Quais tipos de sistemas de software e hardware são utilizados no contexto de rastreamento corporal?

Amiri, Shoaib e Hiremath (2017) utilizam no contexto de rastreamento corporal para captura de movimento seis sensores vestíveis. Nesse ínterim, Sinha et al. (2017), Brown, García-Vergara e Howard (2015), Myagmarbayar et al. (2013), Gauthier e Cretu (2014), Ahmed et al. (2019), Oña, Balaguer e Jardón (2018), Maggiorini, Ripamonti e Zanon (2012), Maggiorini, Ripamonti e Scambia (2012), Chang et al. (2016), Dios, Chung e Meng (2014) utilizam *Kinect* para a captura de movimento. Todavia, Gwak et al. (2019) usam neste contexto de rastreamento corporal um sensor flexível embutido em uma joelheira. Do mesmo modo, Drobnjakovic et al. (2018) valem-se de uma câmera de três dimensões (3D) para geração de um ponto de referência, proporcionando uma maior precisão, atuando em conjunto na saída dos sensores sem fios que são colocados em todo o corpo. Outrossim, Metsis, Smith e Gobert (2017) utilizam o *Kinect* e uma esteira omnidirecional para o paciente caminhar em um ambiente usando um *headset* de Realidade Virtual, assim como sensores inerciais para rastreamento corporal, sensores de pressão para medir a reação do solo com o paciente e sensores de eletromiografia (EMG) para monitorar padrões de ativação muscular.

Ainda em relação aos tipos de sistemas de software e hardware, Durve et al. (2019) utilizam o *Kinect* para fazer o rastreamento corporal e uma pulseira desenvolvida que vibra para indicar a correção dos movimentos do paciente. Ademais, Rahman et al. (2014) contam com o *Kinect* e o *Leap Motion* para tal finalidade. Por conseguinte, Hunt et al. (2018) valem-se de uma rede de rastreamento cinemático utilizando sensores para fazer o rastreamento corporal para ajudar na fisioterapia. Aggarwal et al. (2020) empregam uma meia inteligente (*SoPhy*) com sensores para demonstrar a melhora na marcha do paciente. Do mesmo modo, Strömbäck, Huang e Radu (2020) utilizam sensores diversos para tal,

podendo trabalhar com rastreamento óptico ou inercial.

Stütz et al. (2017) utilizam um sistema com oito câmeras ópticas da *OptiTrack*, enquanto Santos et al. (2015) trabalham com dispositivos diversos, tais como: *Kinect*, *Leap Motion*, *Orbotix Sphero* e *smartphones* com sensores inerciais. Já Tang et al. (2015) utilizam diversos dispositivos ópticos para captura de movimento corporal. Por fim, Waltemate et al. (2015) empregam um protótipo que orienta as pessoas mediante exercícios de fisioterapia pré-gravados usando guias visuais em tempo real e visualizações com multi câmeras.

4.0.3 Q2 Quais as patologias que são beneficiadas por técnicas de rastreamento corporal?

Amiri, Shoaib e Hiremath (2017) beneficiam pacientes com lesão na medula espinhal por meio do emprego de técnicas de rastreamento corporal. Outrossim, Sinha et al. (2017), Oña, Balaguer e Jardón (2018) destinam-se a pacientes que sofreram AVC e outros distúrbios neurológicos. Nesse mesmo sentido, Brown, García-Vergara e Howard (2015), Myagmarbayar et al. (2013), Chang et al. (2016) endereçam-se a pacientes com distúrbio de habilidade motora. Ademais, Gwak et al. (2019) beneficiam pacientes que realizaram cirurgias reconstrutivas do joelho. Enquanto Drobnjakovic et al. (2018), Ahmed et al. (2019), Durve et al. (2019), Rahman et al. (2014), Hunt et al. (2018), Chang et al. (2016) destinam-se a pacientes que fazem fisioterapia por doenças diversas.

Drobnjakovic et al. (2018), Strömbäck, Huang e Radu (2020), Waltemate et al. (2015) dirigem-se a pacientes que fazem fisioterapia, ou também pode ser utilizado na área esportiva. Nesse ínterim, Metsis, Smith e Gobert (2017) destinam-se a pacientes com déficits de equilíbrio causados por lesão, distúrbios neurológicos ou envelhecimento. Ademais, Dios, Chung e Meng (2014) endereçam-se a idosos que precisam fazer atividades físicas. Aggarwal et al. (2020) destinam-se a pacientes com problemas em membros inferiores. Outrossim, Stütz et al. (2017) beneficiam pacientes com problema de síndrome do ombro congelado. Santos et al. (2015) destinam-se a pacientes que sofrem quedas devido a idade elevada, podendo sofrer fraturas ou danos neurológicos a longo prazo. Por fim, Maggiorini, Ripamonti e Zanon (2012), Maggiorini, Ripamonti e Scambia (2012) dirigem-se a pacientes idosos.

4.0.4 Q3 Quais objetivos podem ser alcançados com o emprego de um *framework*?

Amiri, Shoaib e Hiremath (2017) apresentam como objetivo criar um *framework* para capturar e rastrear mobilidade baseada em tecnologia assistiva em indivíduos com lesão na medula espinhal. Ademais, Sinha et al. (2017) possuem a intenção de criar um

framework para melhorar a precisão do rastreamento esquelético do *Kinect*. Nesse mesmo sentido, Brown, García-Vergara e Howard (2015) têm o propósito de criar um *framework* para acoplar *serious games* em um robô que fornece *feedback* corretivo durante a interação. Gwak et al. (2019) possuem o intuito de criar um *framework* para monitorar e melhorar a qualidade dos exercícios de fisioterapia; um aplicativo de *smartphone Android* conecta na pulseira e transmite os dados em tempo real para o armazenamento no banco de dados, apresentando a evolução do movimento.

Drobnjakovic et al. (2018) têm a finalidade de criar um *framework* utilizando o *Unity* para analisar a eficácia de utilizar uma câmera 3D para estabilizar a leitura de dados de sensores inerciais no contexto de reabilitação. Myagmarbayar et al. (2013) têm o objetivo de criar um *framework* voltado ao reconhecimento da atividade humana, por exemplo, identificando se a pessoa está sentada, andando, parada, entre outros, mediante uso do Modelo Oculto de Markov. Gauthier e Cretu (2014) apresentam como objetivo criar um *framework* para quantificação de movimentos para exercícios físicos feitos em casa. Metsis, Smith e Gobert (2017) têm como objetivo criar um *framework* que integra uma esteira omnidirecional com Realidade Virtual para fornecer uma experiência única de reabilitação.

Ahmed et al. (2019) têm o intuito criar um *framework* de redução de ruído, utilizando filtro de Kalman e um algoritmo de redução de ruído recursivo. Ambos são usados para melhorar a precisão e a consistência dos dados capturados dos pacientes. Ademais, Chang et al. (2016), Stütz et al. (2017), Maggiorini, Ripamonti e Zanon (2012), Maggiorini, Ripamonti e Scambia (2012) têm como finalidade criar um *framework* baseado em *Kinect* para reabilitação motora. Dios, Chung e Meng (2014) têm como objetivo criar um *framework* para comparação de desempenho em atividades físicas. Por fim, Durve et al. (2019) propõem um *framework* que pretende ser uma ferramenta útil e eficaz para sessões de fisioterapia à distância, permitindo uma redução de custos considerável, e acesso em áreas mais remotas, como regiões rurais.

Rahman et al. (2014) têm como propósito criar um *framework* com tecnologia web para ser usado com *serious games*. Hunt et al. (2018) propõem um *framework* que proporciona o uso de Realidade Virtual e Aumentada, sendo viável para fisioterapia devido à sua alta precisão, capacidade de modelagem não linear e baixa carga computacional. Na mesma linha, Oña, Balaguer e Jardón (2018) têm a finalidade de criar um *framework* para uso de *serious games* como ferramenta de reabilitação e avaliação do movimento do membro superior. Aggarwal et al. (2020) têm como objetivo criar um *framework* para sistema de telessaúde para roupas inteligentes. Strömbäck, Huang e Radu (2020) têm como propósito criar um *framework* para *Multimodal Deep Learning* para registro automático de exercícios em dispositivos diversos.

Santos et al. (2015) têm como objetivo criar um *framework* para *Unity*, para ajudar

na reabilitação física e reduzir o risco de quedas em idosos, melhorando o equilíbrio, a força muscular e a mobilidade. Já Waltemate et al. (2015) têm como propósito criar um *framework* para sistemas de Realidade Virtual para aprendizagem motora, com um foco especial na captura e renderização de movimento. Enquanto Chang et al. (2016) têm como intuito criar um *framework* para exercícios de fisioterapia que exploram técnicas de *feedback* explorando orientação visual, para que seja feito a fisioterapia em casa de forma correta.

4.0.5 Q4 Qual o uso de aprendizado de máquina no domínio de técnicas de rastreamento corporal?

Foi observado que Amiri, Shoaib e Hiremath (2017) fazem a classificação utilizando os algoritmos *Naïve Bayes* e Árvore de Decisão para detectar níveis de mobilidade do paciente. Outrossim, Gwak et al. (2019) usam aprendizado de máquina para fazer inferências nos dados coletados. Dios, Chung e Meng (2014) empregam aprendizado de máquina para classificar os exercícios, utilizando para tanto: C4.5, *Support Vector Machine* (*SVM*) e *Naïve Bayes*. Nesse mesmo sentido, Durve et al. (2019) fazem a utilização do aprendizado de máquina nos dados coletados para fornecer *feedback* ao usuário. Rahman et al. (2014) usam aprendizado de máquina para compreender o comportamento e os padrões de progresso dos pacientes. Strömbäck, Huang e Radu (2020) utilizam aprendizado de máquina para analisar múltiplas fontes de dados para segmentação robusta e precisa de atividades, reconhecimento de exercícios e contagem de repetições.

4.1 Considerações finais

Dispositivos ópticos e inerciais foram utilizados para fazer o rastreamento corporal, sendo o *Kinect* o mais utilizado com 60,9%. Outros dispositivos foram utilizados, tais como: *OptiTrack*, *Orbotix Sphero*, *Leap Motion*, *EMG*, sensores vestíveis para captura de movimento (*SenseWear Armbands BodyMedia Inc*), *Arduino* com sensores, entre outros.

Todos os trabalhos revisados tratam lesões motoras e neuromotoras, alguns também atuam na área esportiva, ajudando atletas a melhorar o seu desempenho. A maioria dos trabalhos estão relacionados ao público diverso, no entanto, há trabalhos que tratam de pacientes específicos, tais como: idosos, pacientes que sofreram AVC, lesão na medula espinhal e cirurgia reconstrutiva do joelho.

O objetivo da maioria dos trabalhos é criar um *framework* para reabilitação motora utilizando *Kinect*. Há trabalhos que possuem propostas diferentes, tais como: criar um *framework* para redução de ruído para melhorar a precisão de movimentos capturados, comparar desempenho em atividades físicas, utilizar Realidade Virtual e Aumentada junto ao *Kinect* e *framework* para o desenvolvimento de roupas vestíveis.

A maioria dos trabalhos (73,7%) não utiliza aprendizado de máquina. Os que utilizam, têm como finalidade: reconhecer exercícios e contagem de repetições, detectar nível de mobilidade do paciente, fazer inferência nos dados coletados e fornecer *feedback* ao usuário.

Há muitos artigos sobre rastreamento corporal, entretanto, quando realizada a busca sem utilizar a palavra-chave *framework*, a quantidade de artigos retornados aumentou consideravelmente – de 30 artigos na *IEEE* para 369; e na *ACM* de 43 para 135. Esse fato demonstra que não existem muitos *frameworks* sendo criados para trabalhar com rastreamento corporal, motivo pelo qual fica explícito a inovação e contribuição tecnológica dessa pesquisa.

5 *BTS Framework*

Foi desenvolvido, neste trabalho, o *BTS framework*, um *framework* multi-agente do tipo reativo, caixa cinza e com o intuito de abstrair as dificuldades no desenvolvimento de soluções que envolvam utilização de dispositivos para rastreamento corporal. O *framework* foi desenvolvido para o *Unity* com a linguagem de programação C# com o intuito de integrar todas as soluções já desenvolvidas pelo grupo de pesquisa.

O desenvolvedor, para utilizar o *framework* proposto, precisa importar o pacote do *framework* no *Unity*, assim, pode utilizar as classes já definidas com seus respectivos métodos, permitindo que o desenvolvedor foque exclusivamente na aplicação, não se preocupando com particularidades dos dispositivos para rastreamento corporal, banco de dados, autenticação, importação de dados, aprendizado de máquina, entre outros.

5.0.1 Sistema multi-agente

São quatro classes que compõem o ***Core*** do sistema multi-agente: ***Input***, ***DataManager***, ***Analysis*** e ***Utils***. A classe ***Core*** é a principal responsável pelo orquestramento das demais classes agentes, conforme observado na Figura 13.

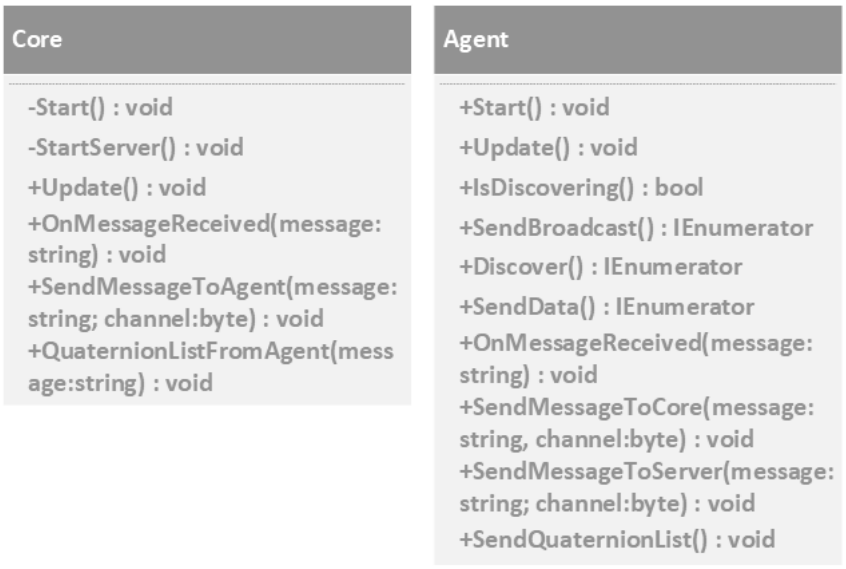


Figura 13 – Estrutura geral do sistema multi-agente

Uma das vantagens da utilização do sistema multi-agente é conseguir fazer com que dispositivos que não sejam multiplataforma possam ser empregados em aplicações distribuídas, se comunicando por meio de *sockets*. Assim, é possível que um dispositivo como o *BSN*, que funciona apenas no *Android*, envie as informações coletadas pelo sensor para uma aplicação que esteja, por exemplo, sendo executada no *Windows*. Desse modo, é

permitida uma interoperabilidade que não seria possível sem o sistema multi-agente. Viabiliza também, de posse das informações coletadas pelo ***Core***, a delegação de instruções para as demais classes orquestradas.

Para realizar a comunicação, foi empregada a *API (Application Programming Interface) Ruffles*, uma biblioteca *UDP* (*User Datagram Protocol*) totalmente gerenciada e feita para que tenha alto desempenho com latência baixa (CORÉN, 2020). A primeira etapa é definir o *UUID* (*Universally Unique Identifier*) da aplicação, que deve ser igual no cliente que são os agentes e no servidor que é o ***Core***. O *UUID* serve para a identificação da aplicação na rede. A porta 5557 foi a definida como padrão para a comunicação, conforme o código da classe orquestrada 5.1.

```
readonly string uuid =
   "b6af0cc1-4e8d-43cd-b771-321125895433";
readonly int port = 5557;
```

Listagem 5.1 – *UUID* e porta

Depois, na classe orquestrada, deve-se iniciar o *socket*, configurar os canais para comunicação e iniciar as co-rotinas, conforme o código 5.2.

```
void Start()
    {
        // Initializes the client
        socket = new RuffleSocket(new
           Ruffles.Configuration.SocketConfig()
        {
            ChallengeDifficulty = 20, // Difficulty 20 is
               fairly hard
            ChannelTypes = new[]
            {
                ChannelType.Reliable,
                ChannelType.ReliableSequenced,
                ChannelType.Unreliable,
                ChannelType.UnreliableOrdered,
            },
            AllowBroadcasts = true,
            AllowUnconnectedMessages = true,
        });
        socket.Start();

        StartCoroutine(Discover());
        StartCoroutine(SendBroadcast());
```

```
    }
```

Listagem 5.2 – Iniciando *socket* da classe orquestrada

O próximo passo é enviar na rede um *broadcast* com a porta e o *UUID* determinados, conforme o código 5.3.

```
IEnumerator SendBroadcast()
    {
        while (IsDiscovering)
        {
            ArraySegment<byte> data = new
               ArraySegment<byte>(Encoding.UTF8.
            GetBytes(uuid), 0,
               Encoding.UTF8.GetBytes(uuid).Length);
            socket.SendBroadcast(data, port);
            yield return new WaitForSeconds(2);
        }
    }
```

Listagem 5.3 – *Broadcast* na rede

Posteriormente ao *broadcast* é o *discover*, ou seja, a conexão só é feita com sucesso se for recebido o mesmo *UUID* enviado. Enquanto não fechar a conexão nas duas pontas, o método ***Recycle*** é chamado, para não ter estouro de memória, ou seja, não permite que conexões sejam realizadas de forma indeterminada. Quando concluída a conexão, ou seja, quando o *UUID* é encontrado, então define-se o ***isDiscovering*** como *false* e o ***isConnected*** como *true*, conforme o código. 5.4.

```
IEnumerator Discover()
    {
        while (IsDiscovering)
        {
            yield return new WaitForSeconds(1);
            @event = socket.Poll();
            if (@event.Type == NetworkEventType.Nothing)
            {
                @event.Recycle();
                continue;
            }

            if (@event.Type ==
               NetworkEventType.UnconnectedData)
```

```
            {
                string message =
                   Encoding.UTF8.GetString(@event.
                Data.Array, @event.Data.Offset,
                   @event.Data.Count);
                if (message.Equals(uuid))
                {
                    isDiscovering = false;
                    serverConnection =
                       socket.Connect(@event.EndPoint);
                    isConnected = true;
                    StartCoroutine(SendData());

                    yield return new WaitForSeconds(5);
                    //Connection ok
                    break;
                }
            }
        }
    }
```

Listagem 5.4 – Descoberta na rede

Para o envio de mensagem ao ***Core***, a co-rotina ***SendData*** só inicia o envio após a variável ***isConnected*** for *true*. Note que o envio é configurado com o tempo de espera em 1/30f, ou seja, 30 quadros por segundo.

Existem vários tipos de canais na *API Ruffles*: 0 - ***Reliable***, que garante a entrega, mas não a ordem; 1 - ***ReliableSequenced***, que consegue garantir a entrega e a ordem; 2 - ***Unreliable*** não garante a entrega nem a ordem e os pacotes duplicados são descartados; 3 - ***UnreliableOrdered*** não garante a entrega, mas a ordem sim, pacotes duplicados e antigos são descartados.

A mensagem é enviada na forma de *ArraySegment*, pelo método ***SendMessageToCore***, passando como parâmetro a mensagem. O canal escolhido foi o ***ReliableSequenced***, em que o envio é feito com garantia de entrega e ordem, conforme o código 5.5.

```
    public void SendMessageToCore(string message, byte
       channel = 1)
    {
        ArraySegment<byte> data =
            new ArraySegment<byte>(Encoding.UTF8.
```

```
            GetBytes(message), 0,
               Encoding.UTF8.GetBytes(message).Length);
        serverConnection?.Send(data, channel, false,
           (ulong)messagesSent);
        messagesSent++;
    }

    IEnumerator SendData()
    {
        while (isConnected)
        {
            SendQuaternionList();
            yield return new WaitForSeconds(1 / 30f);
        }
    }
```

Listagem 5.5 – Envio de mensagem

O módulo ***Core*** pode enviar mensagem para o módulo orquestrado. Essa mensagem é recebida pelo método ***OnMessageReceived***, conforme o código 5.6.

```
void OnMessageReceived(string messageCore)
    {
       // do something
    }
```

Listagem 5.6 – Recebimento de mensagem do ***Core***

Com o *UUID* e portas já definidas, deve-se iniciar também o *socket* no módulo do ***Core***, conforme o código 5.7.

```
  private void StartServer()
    {
        IsConnected = false;
        socket = new RuffleSocket(new
           Ruffles.Configuration.SocketConfig()
        {
            ChallengeDifficulty = 20, // Difficulty 20 is
               fairly hard
            ChannelTypes = new ChannelType[]
             {
                 ChannelType.ReliableSequenced,
             },
```

```
            AllowBroadcasts = true,
            AllowUnconnectedMessages = true,
            DualListenPort = port
        });
        socket.Start();
    }
    }
```

Listagem 5.7 – Iniciando o socket no módulo ***Core***

No método ***Update*** do ***Core*** são feitas diversas tratativas – se não recebeu nenhuma mensagem, o método ***Recycle*** é chamado, para não ter estouro de pilha; se recebeu o *broadcast* com o *UUID* correspondente, ele responde a classe orquestrada; se fechou a conexão nas duas pontas, atribui a variável ***IsConnected*** como *true*; e se recebeu dados da classe orquestrada, encaminha para o método ***OnMessageReceived***, conforme o código 5.8.

```
void Update()
    {
        @event = socket.Poll();
        if (@event.Type == NetworkEventType.Nothing)
           //nothing received
        {
            @event.Recycle();
            return;
        }
        else if
            (@event.Type == NetworkEventType.BroadcastData)
               //received broadcast, if the uuid matches,
               respond to client
        {
            string message =
               Encoding.UTF8.GetString(@event.Data.Array,
               @event.Data.Offset, @event.Data.Count);
            if (message.Equals(uuid))
            {
                socket.SendUnconnected(@event.Data,
                   @event.EndPoint);
            }

            Debug.Log("Broadcast: " +
```

```
                @event.EndPoint.AddressFamily.ToString() + "
                send " +
                        System.Text.Encoding.
                        UTF8.GetString(@event.Data.Array));
            // We got a broadcast. Reply to them with the
                same token they used
        }

        else if (@event.Type == NetworkEventType.Connect)
            //received connection request
        {
            clientConnection = @event.Connection;
            Debug.Log(@event.Connection.State + "to " +
                @event.EndPoint.AddressFamily.ToString());
            IsConnected = true;
        }
        else if (@event.Type == NetworkEventType.Data) //the
            client's commands go here
        {
            string message =
                Encoding.UTF8.GetString(@event.Data.Array,
                @event.Data.Offset, @event.Data.Count);
            OnMessageReceived(message);
        }

        // Recycle the event
        @event.Recycle();
    }
```

Listagem 5.8 – Tratativas do ***Core***

O módulo ***Core*** pode enviar mensagens para os agentes utilizando o método ***SendMessageToAgent***, conforme o código 5.9.

```
public void SendMessageToAgent(string message, byte channel
    = 1)
    {
        ArraySegment<byte> data =
            new ArraySegment<byte>(Encoding.UTF8.
            GetBytes(message), 0,
                Encoding.UTF8.GetBytes(message).Length);
```

```
            clientConnection?.Send(data, channel, false,
                (ulong)messagesSent);
            messagesSent++;
        }
```

Listagem 5.9 – Envio de mensagem do ***Core***

Existe uma particularidade nos agentes, além de trocar mensagens com o ***Core*** utilizando o método ***SendMessageToCore***, foi criado um *DTO* (*Data Transfer Object*) para que se possa enviar dados dos sensores, conforme a Figura 14.

Figura 14 – *DTO* para envio de dados dos sensores

O *DTO* envia o *quaternion* com o respectivo nome da rotação, conforme o código 5.10.

```
[System.Serializable]
public class RotationDTO
{
    public string name;
    public QuaternionDTO rotation;

    public RotationDTO(string name, QuaternionDTO rotation)
    {
        this.name = name;
        this.rotation = rotation;
    }
}

[System.Serializable]
public class QuaternionDTO
{
    public float x, y, z, w;

    public QuaternionDTO(float x, float y, float z, float w)
    {
```

```
        this.x = x;
        this.y = y;
        this.z = z;
        this.w = w;
    }
}
```

Listagem 5.10 – *DTO*

Dessa forma, podemos receber os dados do *BSN* mediante uso do sistema multi-agente, bastando utilizar o método ***SendQuaternionList*** para enviar e o método ***QuaternionListFromAgent*** para receber os dados. O *quaternion* já é atribuído ao *GameObject* da rotação, pois o nome é enviado junto ao *quaternion*, conforme o código 5.11.

```
public void SendQuaternionList()
    {
        string a = "Quaternions/{\"rotations\":[";
        List<RotationDTO> rt = new List<RotationDTO>();
        var rotatableObjects =
           FindObjectsOfType<RotatableObject>().Where(ro =>
           ro.bsnDevice != null);

        foreach (RotatableObject ro in rotatableObjects)
        {
            var transform1 = ro.transform;
            var rotation = transform1.rotation;
            var quaternion = new QuaternionDTO(rotation.x,
               rotation.y,
                rotation.z, rotation.w);
            rt.Add(new RotationDTO(ro.SimplifiedName,
               quaternion));
        }

        rt.ForEach(rotation => { a +=
           JsonUtility.ToJson(rotation) + ","; });

        a = a.TrimEnd(',') + "]}";
        SendMessageToCore(a, 1);
    }

    public void QuaternionListFromAgent(string message)
```

```
{
    if (message.StartsWith("Quaternions/"))
    {
        string json = message.Substring(12);
        JsonUtility.FromJsonOverwrite(json, this);

        foreach (RotationDTO rdto in rotations)
        {
            Quaternion q = new
               Quaternion(rdto.rotation.x,
               rdto.rotation.y, rdto.rotation.z,
               rdto.rotation.w);
            GameObject.Find(rdto.name).transform.rotation
               = q;
        }
    }
}
```

Listagem 5.11 – Envio e recebimento de *quaternions* pelo sistema multi-agente

5.0.2 Módulo *Input*

O módulo ***Input*** é responsável por controlar os dispositivos ópticos e inerciais, iniciar os dispositivos, parar e ter um retorno se o mesmo está disponível, conforme a Figura 15.

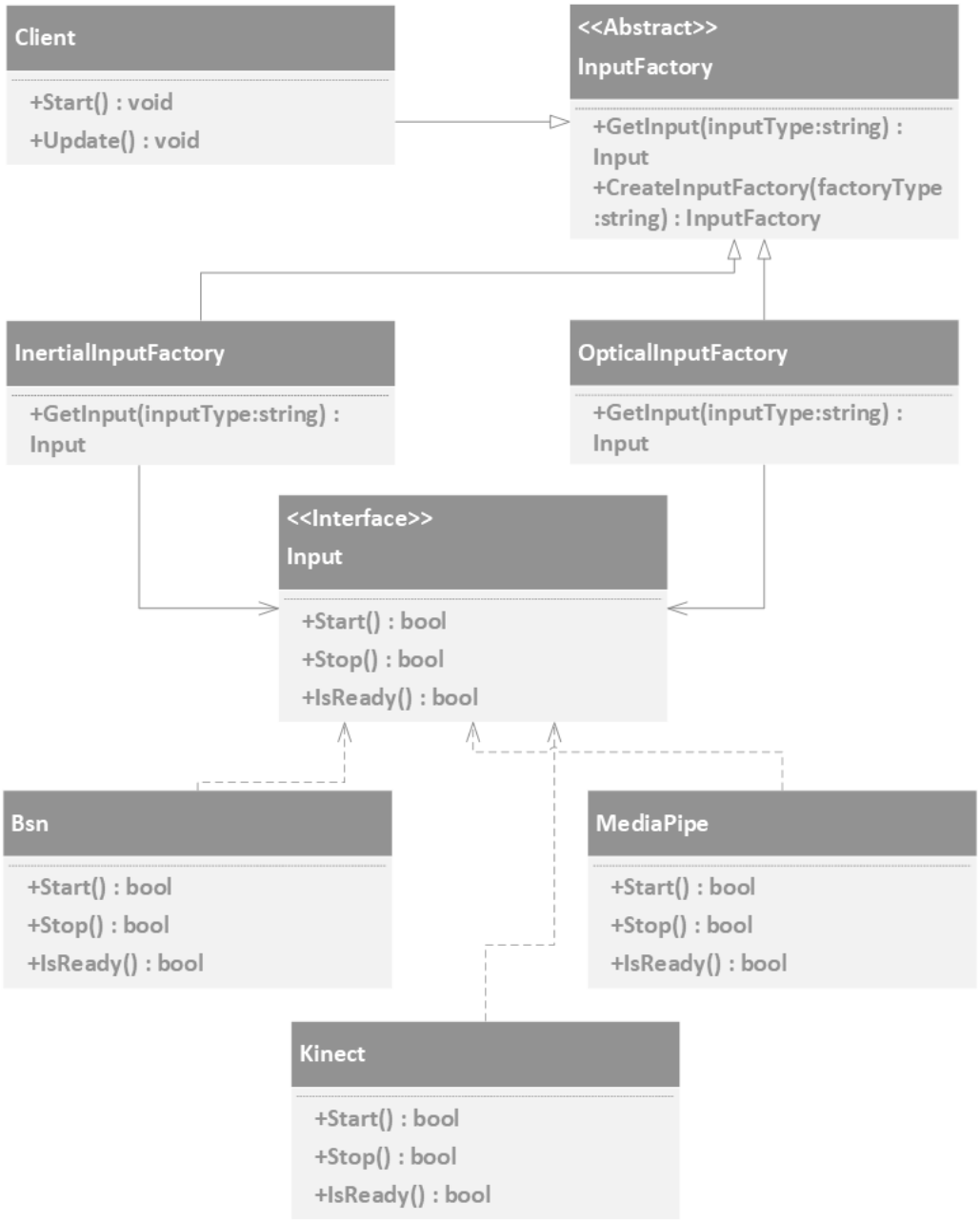


Figura 15 – *UML (Unified Modeling Language) do módulo* ***Input***

O padrão de projetos utilizado no módulo ***Input*** foi o *Abstract Factory*. O *Abstract Factory* é um padrão de projeto do tipo criacional que permite criar famílias de objetos relacionados sem ter que especificar suas classes concretas. Proporciona algumas vantagens, como saber que os produtos obtidos de uma fábrica são compatíveis entre eles, facilidade de manutenção e evita acoplamento entre produto concreto e código cliente. Como desvantagem, pode tornar o código mais complexo num primeiro momento devido a maior quantidade de classes e interfaces (GAMMA et al., 1995).

No *framework* desenvolvido, o *AbstractFactory* declara uma interface para operações que irão criar objetos *AbstractProduct*. Nesse caso, são dispositivos de entrada. Essa classe contém dois métodos – o método ***GetInput***, que é um método abstrato que será

implementado pelas classes de fábrica filhas; e o método **CreateInputFactory**, que recebe um parâmetro de entrada, ou seja, tipo de fábrica e, em seguida, cria e retorna o objeto de fábrica apropriado para o chamador, conforme o código 5.12.

```
//AbstractFactory

namespace AbstractFactoryDesignPattern
{
    public abstract class InputFactory
    {
        public abstract Input GetInput(string inputType);

        public static InputFactory CreateInputFactory(string
           factoryType)
        {
            if (factoryType.Equals("Optical"))
                return new OpticalInputFactory();
            else
                return new InertialInputFactory();
        }
    }
}
```

Listagem 5.12 – *AbstractFactory*

Conforme apresentado no Código 5.13, a classe concreta implementa a classe *AbstractFactory* e *InputFactory*. No *framework* desenvolvido, essa classe vai implementar o método ***GetInput*** da classe *InputFactory*. Existem dois tipos concretos de fábricas em nosso exemplo, ou seja, *InertialInputFactory*, para dispositivos inerciais e *OpticalInputFactory* para dispositivos ópticos.

```
//ConcreteFactory1

namespace AbstractFactoryDesignPattern
{
    public sealed class InertialInputFactory : InputFactory
    {
        public override Input GetInput(string inputType)
        {
            if (inputType.Equals("Bsn"))
            {
                return new Bsn();
```

```
            }
            else
                return null;
        }
    }
}
```

Listagem 5.13 – *InertialInputFactory*

Podemos observar no código supracitado que o método ***GetInput*** cria e retorna o objeto inercial apropriado com base no parâmetro de entrada – o *InputType* recebido. Analogamente, o mesmo vale para os dispositivos ópticos da classe concreta *OpticalInputFactory*, conforme o Código 5.14.

```
//ConcreteFactory2

namespace AbstractFactoryDesignPattern
{
    public sealed class OpticalInputFactory : InputFactory
    {
        public override Input GetInput(string inputType)
        {
            if (inputType.Equals("Mediapipe"))
            {
                return new Mediapipe();
            }
            else if (inputType.Equals("Kinect"))
            {
                return new Kinect();
            }
            else
                return null;
        }
    }
}
```

Listagem 5.14 – *OpticalInputFactory*

Conforme apresentado no Código 5.15, a interface declara os métodos do produto. No *framework*, são os métodos ***Start***, ***Stop*** e ***IsReady*** do objeto *Input*.

O método ***Start*** é responsável por iniciar o dispositivo; o ***Stop*** por parar o dispositivo iniciado; e o ***IsReady*** verifica se está pronto ou não.

```
//Abstract Product

namespace AbstractFactoryDesignPattern
{
    public interface Input
    {
        public bool Start();
        public bool Stop();
        public bool IsReady();
    }
}
```

Listagem 5.15 – *AbstractProduct*

A classe concreta do produto implementa a interface *AbstractFactory* (*InputFactory*) para criar produtos concretos. Em nosso exemplo, as classes dos produtos são *Mediapipe*, *BSN* e *Kinect*.

Podemos ver um exemplo da classe *Mediapipe* que implementa os métodos ***Start***, ***Stop*** e ***IsReady*** da interface *Input*, conforme o código 5.16 e, analogamente, para os demais dispositivos.

```
//Concrete Product

namespace AbstractFactoryDesignPattern
{
    public class Mediapipe : Input
    {
        private static bool _isMediaPipeReady;

        public bool Start()
        {
            return _isMediaPipeReady = true;
        }

        public bool Stop()
        {
            return _isMediaPipeReady = false;
        }

        public bool IsReady()
        {
```

```
            return _isMediaPipeReady;
        }
    }
}
```

Listagem 5.16 – *Mediapipe*

O Cliente usa as interfaces *AbstractFactory* e *AbstractProduct* para criar uma família de objetos relacionados, conforme mostrado no Código 5.17.

```
using UnityEngine;

namespace AbstractFactoryDesignPattern
{
    class ClientInput : MonoBehaviour
    {
        private Input input;
        private InputFactory inputFactory;

        // Start is called before the first frame update
        void Start()
        {
            // Create the OpticalInput factory object by
               passing the factory type as Optical
            inputFactory =
               InputFactory.CreateInputFactory("Optical");

            // Get Mediapipe Input object by passing the
               input type as Mediapipe
            input = inputFactory.GetInput("Mediapipe");
            input.Start();

            // Get Kinect Input object by passing the input
               type as Kinect
            input = inputFactory.GetInput("Kinect");
            input.Start();
        }

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

```
            }
        }
}
```

Listagem 5.17 – *ClientInput*

5.0.3 Módulo *DataManager*

O módulo ***DataManager*** é responsável por manipular os dados, sejam esses capturados dos dispositivos, persistidos em disco ou importados de um arquivo externo para a aplicação, ou simplesmente usando os dados capturados diretamente na aplicação, conforme a Figura 16.

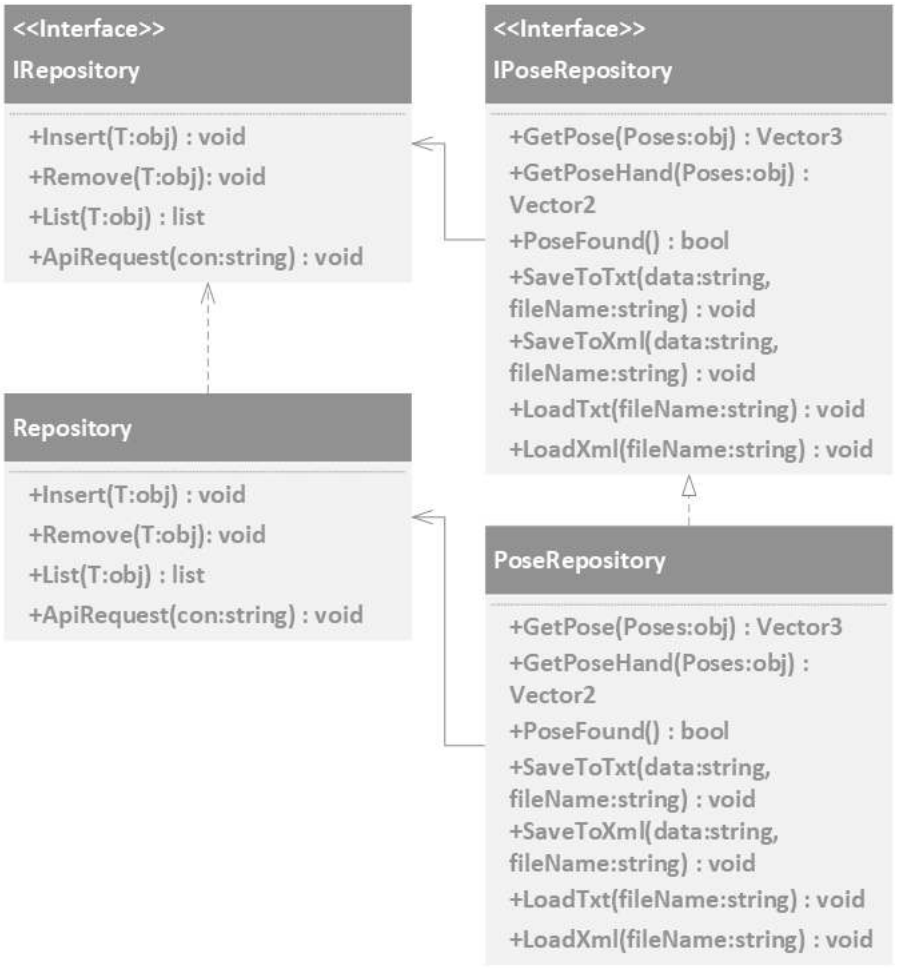


Figura 16 – *UML* do módulo ***DataManager***

O *framework* utiliza o mesmo padrão de 20 articulações do *Kinect*, entretanto, é flexível e permite utilizar outros padrões, como o padrão de rastreamento corporal e de mão do *Mediapipe*, conforme apresentado na Figura 17.

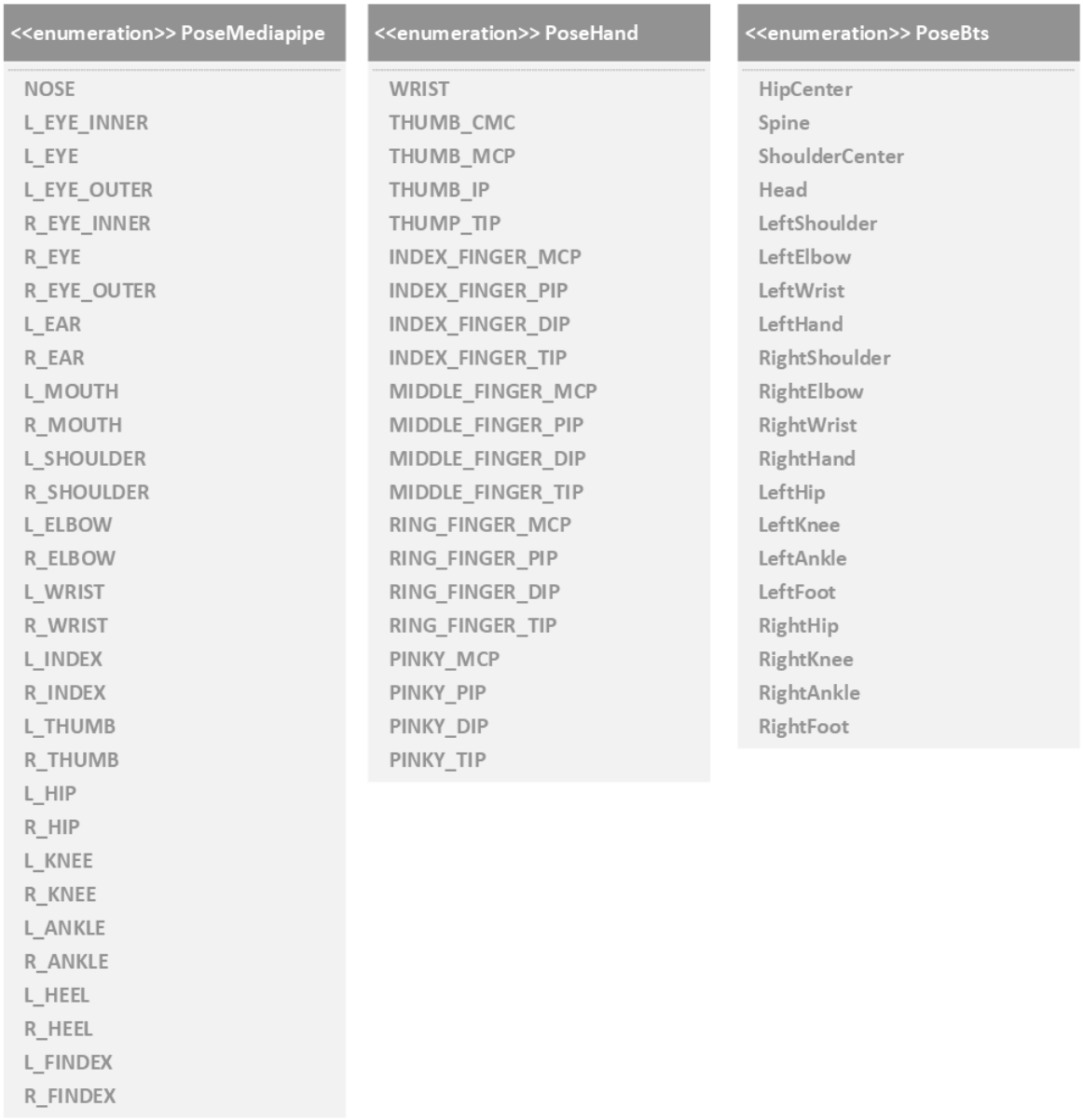
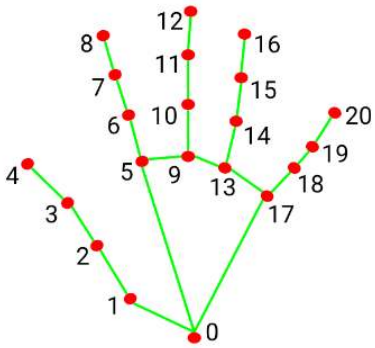


Figura 17 – *UML* de enumerações de *Poses*

O rastreamento de mão do *Mediapipe* é interessante porque captura 21 articulações, conseguindo capturar com precisão a movimentação da mão. As articulações são definidas conforme a Figura 18.



0. WRIST
1. THUMB_CMC
2. THUMB_MCP
3. THUMB_IP
4. THUMB_TIP
5. INDEX_FINGER_MCP
6. INDEX_FINGER_PIP
7. INDEX_FINGER_DIP
8. INDEX_FINGER_TIP
9. MIDDLE_FINGER_MCP
10. MIDDLE_FINGER_PIP
11. MIDDLE_FINGER_DIP
12. MIDDLE_FINGER_TIP
13. RING_FINGER_MCP
14. RING_FINGER_PIP
15. RING_FINGER_DIP
16. RING_FINGER_TIP
17. PINKY_MCP
18. PINKY_PIP
19. PINKY_DIP
20. PINKY_TIP

Figura 18 – Mapeamento da mão pelo *Mediapipe* (GOOGLE, 2022)

Essa flexibilidade é importante caso seja necessário realizar a captura de outras articulações não previstas nas 20 articulações do *framework*. É importante destacar que o mapeamento também é diferente: pode haver índices diferentes para uma mesma articulação corporal, assim, é importante o desenvolvedor escolher adequadamente os padrões, conforme mostrado no Código 5.18.

```
using UnityEngine;

public class Poses : MonoBehaviour
{
    //Mediapipe Standard
    public enum PoseMediapipe
    {
        NOSE = 0,
        L_EYE_INNER = 1,
        L_EYE = 2,
        L_EYE_OUTER = 3,
        R_EYE_INNER = 4,
        R_EYE = 5,
        R_EYE_OUTER = 6,
        L_EAR = 7,
        R_EAR = 8,
        L_MOUTH = 9,
        R_MOUTH = 10,
        L_SHOULDER = 12,
        R_SHOULDER = 11,
        L_ELBOW = 14,
        R_ELBOW = 13,
        L_WRIST = 16,
        R_WRIST = 15,
```

```
        L_PINKY = 17,
        R_PINKY = 18,
        L_INDEX = 19,
        R_INDEX = 20,
        L_THUMB = 21,
        R_THUMB = 22,
        L_HIP = 24,
        R_HIP = 23,
        L_KNEE = 26,
        R_KNEE = 25,
        L_ANKLE = 28,
        R_ANKLE = 27,
        L_HEEL = 30,
        R_HEEL = 29,
        L_FINDEX = 32,
        R_FINDEX = 31
    }

    //Mediapipe Only
    public enum PoseHand
    {
        WRIST = 0,
        THUMB_CMC = 1,
        THUMB_MCP = 2,
        THUMB_IP = 3,
        THUMP_TIP = 4,
        INDEX_FINGER_MCP = 5,
        INDEX_FINGER_PIP = 6,
        INDEX_FINGER_DIP = 7,
        INDEX_FINGER_TIP = 8,
        MIDDLE_FINGER_MCP = 9,
        MIDDLE_FINGER_PIP = 10,
        MIDDLE_FINGER_DIP = 11,
        MIDDLE_FINGER_TIP = 12,
        RING_FINGER_MCP = 13,
        RING_FINGER_PIP = 14,
        RING_FINGER_DIP = 15,
        RING_FINGER_TIP = 16,
        PINKY_MCP = 17,
```

```
        PINKY_PIP = 18,
        PINKY_DIP = 19,
        PINKY_TIP = 20
    }

    // BTS Standard = Kinect - Mediapipe different mapping
    public enum PoseBts
    {
        HipCenter = 1,
        Spine = 2,
        ShoulderCenter = 3,
        Head = 4,
        LeftShoulder = 5,
        LeftElbow = 6,
        LeftWrist = 7,
        LeftHand = 8,
        RightShoulder = 9,
        RightElbow = 10,
        RightWrist = 11,
        RightHand = 12,
        LeftHip = 13,
        LeftKnee = 14,
        LeftAnkle = 15,
        LeftFoot = 16,
        RightHip = 17,
        RightKnee = 18,
        RightAnkle = 19,
        RightFoot = 20
    }
}
```

Listagem 5.18 – Padrões disponíveis

No ***DataManager*** foi utilizado o padrão de projeto *Repository* que é usado para criar uma camada de abstração entre a camada de acesso a dados e a camada de negócios. Tem-se como vantagem da utilização desse padrão: testes facilitados, gerenciamento mais simples da aplicação com o armazenamento de dados, e permite facilmente trocar por vários *data stores* sem alterar a *API* (CARNEIRO; MAIA, 2020).

A interface *IRepository* é uma interface genérica que define alguns métodos, como ***Insert*** responsável por inserir no SGBD, ***Remove*** para deletar, ***List*** para pesquisar

na lista e o ***ApiRequest*** para trabalhar com *API*. Note que em vez da entidade *Poses*, estamos usando T em todos os lugares de forma genérica, conforme o Código 5.19.

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

public interface IRepository<T> where T : class
{
    void Insert(T obj);
    void Remove(T obj);
    List<T> List(Func<T, bool> expression = null);
    void ApiRequest(String con);
}
```

Listagem 5.19 – *IRepository*

O *Repository* é uma classe genérica que implementa a interface *IRepository*. Note que o repositório genérico não possui funções especificas para funcionar com as articulações, conforme o Código 5.20.

```
public class Repository<T> : IRepository<T> where T : class
{
    List<T> lt;

    public Repository()
    {
        lt = new List<T>();
    }

    public void Insert(T obj)
    {
        lt.Add(obj);
    }

    public void Remove(T obj)
    {
        lt.Remove(obj);
    }

    public List<T> List(Func<T, bool> expression = null)
    {
```

```
        return expression != null ?
            lt.Where(expression).ToList() : lt;
    }

    public void ApiRequest(String con)
    {
        // Waiting for the new rebase to be ready
    }
}
```

Listagem 5.20 – *Repository*

A interface *IPoseRepository* é uma interface específica que define alguns métodos próprios para trabalhar com as articulações, como o ***GetPose***, responsável por retornar uma articulação específica; ***GetHandPose*** que retorna as articulações da mão; ***PoseFound*** que verifica se algum rastreamento já foi feito; ***SaveToTxt*** que persiste em disco em um arquivo txt; ***SaveToXml*** persiste em disco em um arquivo *XML* (*Extensible Markup Language*). O ***LoadTxt*** e ***LoadXml*** carregam as articulações de um arquivo de texto e XML respectivamente , conforme o Código 5.21.

```
using System;
using UnityEngine;
using System.IO;
using System.Xml;

public interface IPoseRepository : IRepository<Poses>
{
    Vector3 GetPose(Poses.PoseBts obj);
    Vector2 GetPoseHand(Poses.PoseHand obj);

    bool PoseFound();
    void SaveToTxt(String data, String fileName);
    void SaveToXml(String data, String fileName);
    void LoadTxt(String fileName);
    void LoadXml(String fileName);
}
```

Listagem 5.21 – *IPoseRepository*

O *PoseRepository* é um repositório específico que implementa a interface *IPoseRepository*, com métodos próprios para lidar com as articulações, ao contrário do *Repository*, que é um repositório genérico, conforme o Código 5.22. O *PoseRepository* faz uso do *Ki-*

nectManager, classe desenvolvida pela Microsoft para utilizar os recursos do *Kinect*, e também utiliza o *MediapipeManager*, análogo a classe anterior porém foi criada para o *framework*, pois não havia algo semelhante para explorar os recursos do *Mediapipe*.

Como foi adotado o padrão de articulações do *framework* quando utilizado o *Mediapipe*, precisamos converter um padrão para o outro, assim, foi utilizado o método ***ConvertKinectMapToMediaPipe***. Dessa forma, fica transparante para o desenvolvedor independente do dispositivo óptico que ele esteja usando.

```
public class PoseRepository : Repository<Poses>,
   IPoseRepository
{
    // Get pose from optical device
    public Vector3 GetPose(Poses.PoseBts obj)
    {
        if (InputSelector.inputFactory.GetInput("Mediapipe")
        .IsReady())
        {
            return MediapipeManager.
            GetPoint(MediapipeManager.
            ConvertKinectMapToMediaPipe(obj));
        }
        else if
           (InputSelector.inputFactory.GetInput("Kinect")
        .IsReady())
        {
            return KinectManager.
            GetRawSkeletonJointPos(KinectManager.
            GetPlayer1ID(), (int)obj);
        }
        else
        {
            return Vector3.zero;
        }
    }

    //Mediapipe only
    public Vector2 GetPoseHand(Poses.PoseHand obj)
    {
        if (MediapipeManager.VectorHand.Count > 0)
        {
```

```
            return new Vector2(
                MediapipeManager.VectorHand[(int)obj].X,
                MediapipeManager.VectorHand[(int)obj].Y
            );
        }
        else
        {
            return Vector2.zero;
        }
    }

    // If PoseFound return true
    public bool PoseFound()
    {
        if (InputSelector.inputFactory.
        GetInput("Mediapipe").IsReady())
        {
            return MediapipeManager.HasPose();
        }
        else if (InputSelector.inputFactory.
        GetInput("Kinect").IsReady())
        {
            return KinectManager.Player1Calibrated;
        }
        else
        {
            return false;
        }
    }

    // Save to txt file
    public void SaveToTxt(String data, String fileName)
    {
        string path = Application.dataPath + "/" + fileName
           + ".txt";
        File.AppendAllText(path, data + "\n\n");
    }

    // Save to xml file
```

```
public void SaveToXml(String data, String fileName)
{
    XmlDocument xmlDocument = new XmlDocument();
    XmlElement root = xmlDocument.CreateElement("Save");
    root.SetAttribute("FileName", fileName);

    // Begin node
    XmlElement pose = xmlDocument.CreateElement("Pose");
    pose.InnerText = data;
    root.AppendChild(pose);
    // End node

    xmlDocument.AppendChild(root);

    xmlDocument.Save(Application.dataPath + "/" +
       fileName + ".xml");
}

// Load txt file
public void LoadTxt(String fileName)
{
    string path = Application.dataPath + "/" + fileName
       + ".txt";
    string textFromFile = File.ReadAllText(path);
    Debug.Log(textFromFile);
}

// Load xml file
public void LoadXml(String fileName)
{
    string path = Application.dataPath + "/" + fileName
       + ".xml";
    XmlDocument xmlDocument = new XmlDocument();
    xmlDocument.Load(path);

    XmlNodeList pose =
       xmlDocument.GetElementsByTagName("Pose");
    Debug.Log(pose[0].InnerText);
}
```

```
}
```

Listagem 5.22 – *PoseRepository*

5.0.4 Módulo *Analysis*

A classe ***Analysis*** é responsável por fazer uma análise dos dados, sendo possível utilizar modelos de aprendizado de máquina para identificar o tipo de movimento feito, pontuação do movimento comparando com a base histórica para acompanhar a evolução do paciente, entre outras possibilidades, conforme a Figura 19.

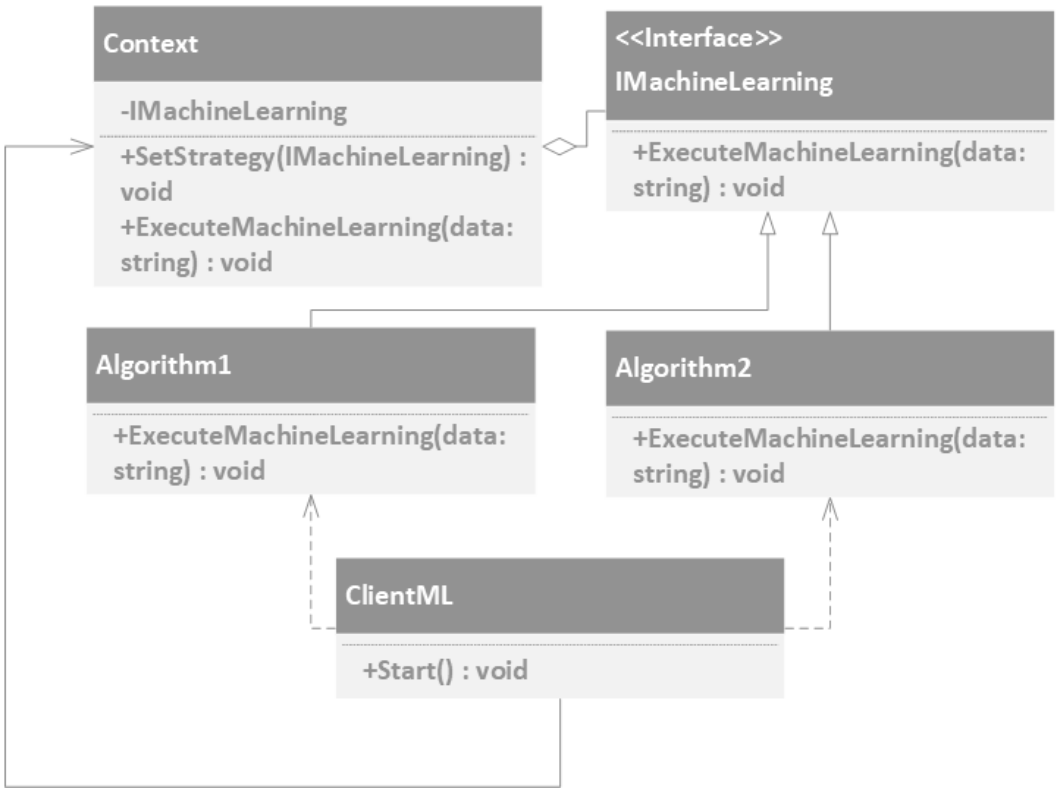


Figura 19 – *UML* do módulo ***Analysis***

Foi utilizado o padrão de projeto comportamental *Strategy*, porque várias classes relacionadas diferem apenas em seus comportamentos e é necessário variantes dos algoritmos de aprendizado de máquina no *framework* (FERREIRA; RESENDE; COSTA, 2012).

A classe *IMachineLearning* declara uma interface que é comum a todos os algoritmos de aprendizado de máquina, ela declara um método que o contexto da aplicação utiliza para executar uma estratégia, conforme o Código 5.23.

```
namespace StrategyDesignPattern
{
    public interface IMachineLearning
```

```
    {
        // Interface for methods
        public void ExecuteMachineLearning(string data);
    }
}
```

Listagem 5.23 – *IMachineLearning*

Cada classe concreta deve implementar o método ***ExecuteMachineLearning*** da interface *IMachineLearning*. Conforme o Código 5.24 tem-se a implementação do primeiro algoritmo e o Código 5.25 referente a um segundo algoritmo de aprendizado de máquina. Esses algoritmos devem ser implementados pelo desenvolvedor de acordo com a sua necessidade. Caso seja necessário ter mais algoritmos deve-se adicionar novas classes concretas que implementem a interface supracitada. O método ***ExecuteMachineLearning*** recebe uma *string* com os dados para que seja processado. Assim, a classe concreta implementa diferentes variações de um algoritmo que o contexto da aplicação necessitar.

```
using UnityEngine;

namespace StrategyDesignPattern
{
    public class Algorithm1 : IMachineLearning
    {
        // Implement Algorithm 1
        public void ExecuteMachineLearning(string data)
        {
            Debug.Log("Data: '" + data);
        }
    }
}
```

Listagem 5.24 – Algorithm 1

```
using UnityEngine;

namespace StrategyDesignPattern
{
    public class Algorithm2 : IMachineLearning
    {
        // Implement Algorithm 2
        public void ExecuteMachineLearning(string data)
        {
```

```
            Debug.Log("Data: '" + data);
        }
    }
}
```

Listagem 5.25 – Algorithm 2

A classe *Context* contém uma propriedade que é usada para conter a referência de um objeto *strategy*. Esta propriedade é ajustada pelo *ClientML* de acordo com o algoritmo de aprendizado de máquina necessário, conforme o Código 5.26.

```
namespace StrategyDesignPattern
{
    // Strategy Pattern Context
    public class Context
    {
        private IMachineLearning ml;

        public Context(IMachineLearning ml)
        {
            this.ml = ml;
        }

        public void SetStrategy(IMachineLearning ml)
        {
            this.ml = ml;
        }

        public void ExecuteMachineLearning(string data)
        {
            ml.ExecuteMachineLearning(data);
        }
    }
}
```

Listagem 5.26 – *Context*

A classe *ClientML* cria um objeto *strategy* específico e passa para o contexto. O contexto define um *setter* que permite o *ClientML* alterar a estratégia associada com o contexto durante a execução, conforme o Código 5.27.

```
using UnityEngine;
```

```
namespace StrategyDesignPattern
{
    public class ClientML : MonoBehaviour
    {
        // Start is called before the first frame update
        // Start Algorithm 1 and 2 - ML
        void Start()
        {
            Context ctx = new Context(new Algorithm1());
            ctx.ExecuteMachineLearning("data1");
            ctx.SetStrategy(new Algorithm2());
            ctx.ExecuteMachineLearning("data2");
        }
    }
}
```

Listagem 5.27 – *ClientML*

5.0.5 Módulo *Utils*

A classe ***Utils*** é uma classe de apoio para facilitar o desenvolvedor na manipulação dos dados obtidos por meio do rastreamento corporal, conforme a Figura 20.

Figura 20 – Estrutura da classe ***Utils***

O método ***Get2dAngle*** é utilizado para conseguir o ângulo quando duas dimensões são empregadas. É necessário inserir três posições - inicial, central e final, assim o método retorna o ângulo. Analogamente, tem-se o método ***Get3dAngle*** para informações em três dimensões. Já os métodos ***Compressing*** e ***Decompress*** servem para compri-

mir e descomprimir dados, facilitando a manipulação com bancos de dados. O método ***CopyTo*** é utilizado como auxiliar aos métodos ***Compressing*** e ***Decompress***.

5.0.6 Considerações finais

Os quatro módulos definidos: ***Input***; ***DataManager***; ***Analysis*** e ***Utils*** definem a estrutura do *framework*. O sincronismo do módulo ***Core***, devido ao sistema multi-agente, com os módulos ***Input*** e ***DataManager*** permitem que o *framework* trabalhe com dispositivos que possuem incompatibilidades de plataforma, por exemplo: utilizar os dados do *Kinect*, que só funciona no sistema operacional *Windows*, em um sistema operacional *Linux* ou *Android*.

6 Cenários de Uso

Dois projetos foram criados com o intuito de demonstrar o *BTS framework* no *Unity*. O primeiro exemplo explora os dispositivos ópticos; o exemplo possui um avatar se movendo com os dados capturados pelo *MediaPipe* e *Kinect*. Com um novo projeto criado, o avatar na cena e o *framework* importado, o *DataManager* é utilizado para lidar com as 20 articulações do avatar.

Foi utilizado o método ***GetPose*** para pegar as posições e movimentar o avatar. O método ***HasPose*** também foi usado, pois ele verifica se a captura foi feita, e caso seja aí devemos atribuir a articulação ao movimento capturado, caso contrário será lançado um erro pelo *Unity*.

Todas as bibliotecas necessárias para utilizar o *Kinect* e o *MediaPipe* já estão incluídas no *framework*, então toda a compilação e configuração foram desnecessárias para que os dispositivos funcionassem.

O código responsável pela utilização do módulo ***Input*** deve ser incluído no projeto. O *ClientInput* usa as interfaces *AbstractFactory* e *AbstractProduct* para criar uma família de objetos relacionados. Inicialmente foi utilizado o *MediaPipe*, conforme o Código 6.1.

```
using UnityEngine;

namespace AbstractFactoryDesignPattern
{
    class ClientInput : MonoBehaviour
    {
        private Input input;
        private InputFactory inputFactory;

        // Start is called before the first frame update
        void Start()
        {
            // Create the OpticalInput factory object by
               passing the factory type as Optical
            inputFactory =
               InputFactory.CreateInputFactory("Optical");

            // Get Mediapipe Input object by passing the
               input type as Mediapipe
```

```
                input = inputFactory.GetInput("Mediapipe");
            }
        }
}
```

Listagem 6.1 – *ClientInput*

Após o *Input* ser adicionado no projeto, faz-se necessário adicionar o código ***input.Start()*** para iniciar o dispositivo e começar a capturar as articulações, com isso, já é possível mover o avatar, conforme a Figura 21.

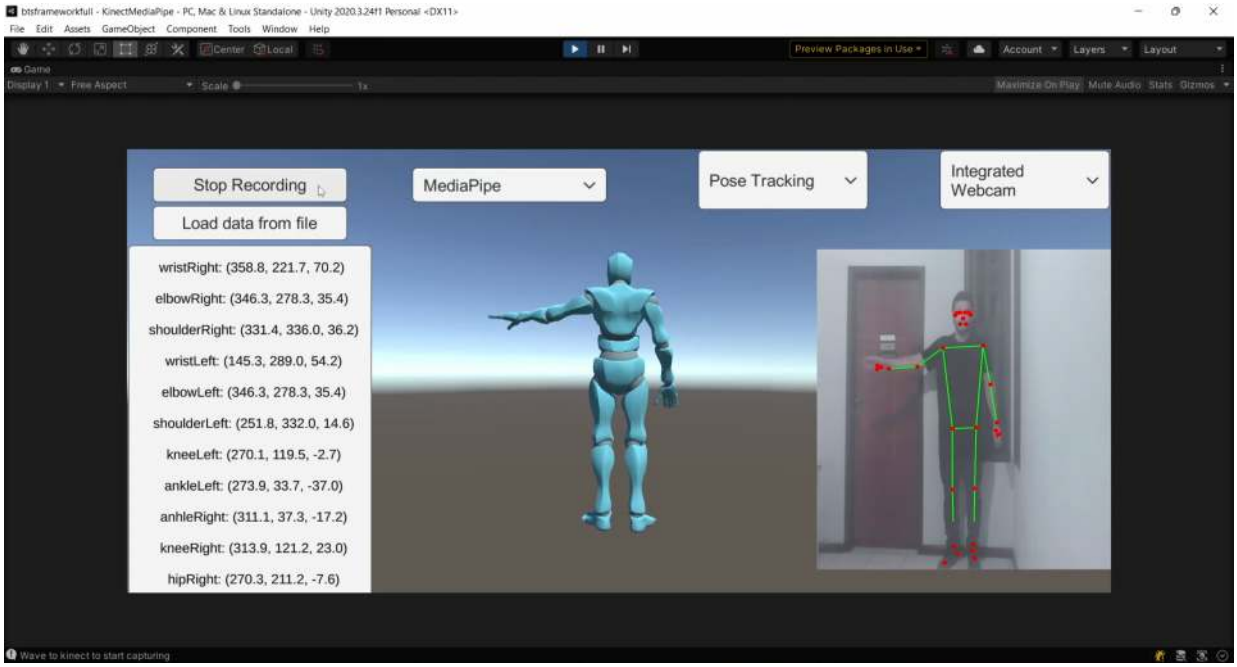


Figura 21 – Avatar se movimentando com o uso do *Mediapipe*

Para alternar para o *Kinect* é simples: basta mudar o ***GetInput("MediaPipe")*** para ***GetInput("Kinect")***. O *DataManager* funciona com qualquer dispositivo, sem a necessidade de se mapear as articulações, pois isso já é feito pelo *framework*, compatibilizando padrões de capturas diferentes. Para alternar para o *Kinect*, deve-se usar o método ***Stop*** no *Mediapipe* e iniciar o método ***Start*** no *Kinect*. Os métodos são os mesmos, porém, atuam em produtos diferentes do *AbstractFactory*. Com esses passos, agora é possível mover o avatar utilizando o *Kinect*, conforme a Figura 22.

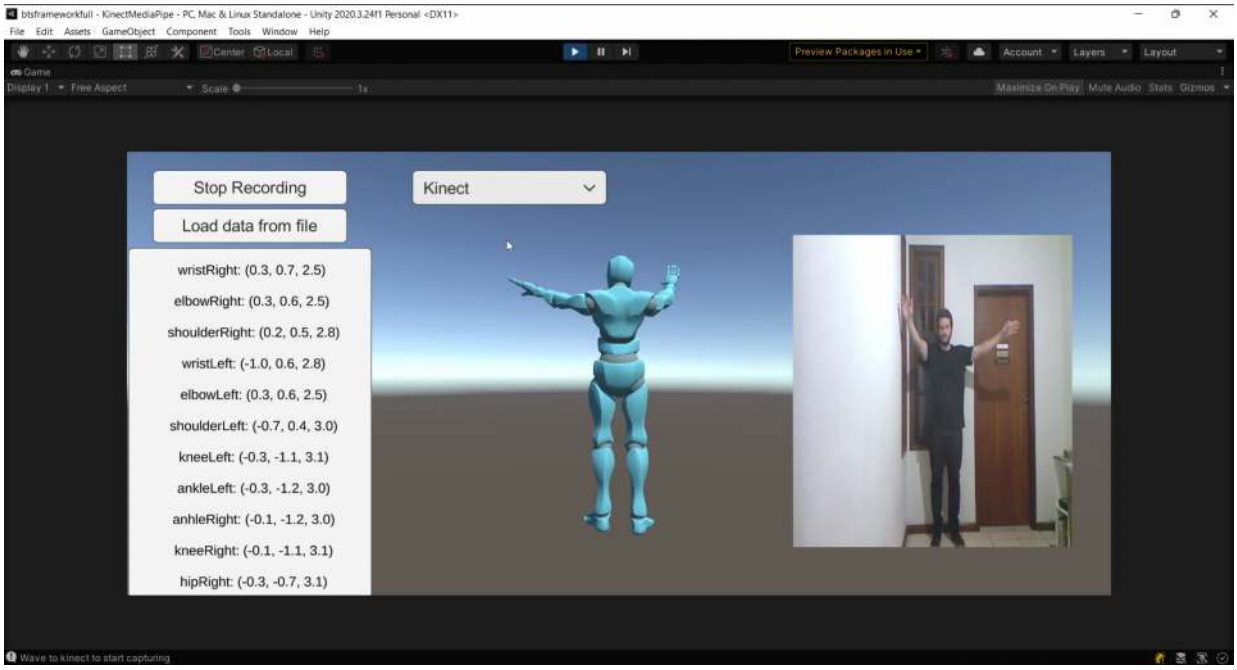


Figura 22 – Avatar se movimentando com o uso do *Kinect*

Para persistir os dados capturados em disco, foi utilizado o método ***SaveToTxt*** e, para carregar os dados salvos, o método ***LoadTxt***, conforme a Figura 23. Outras opções possíveis seriam utilizar a *API* para persistir em um banco de dados ou utilizar os métodos ***SaveToXml*** e ***LoadXml*** para manipular arquivos *XML*.

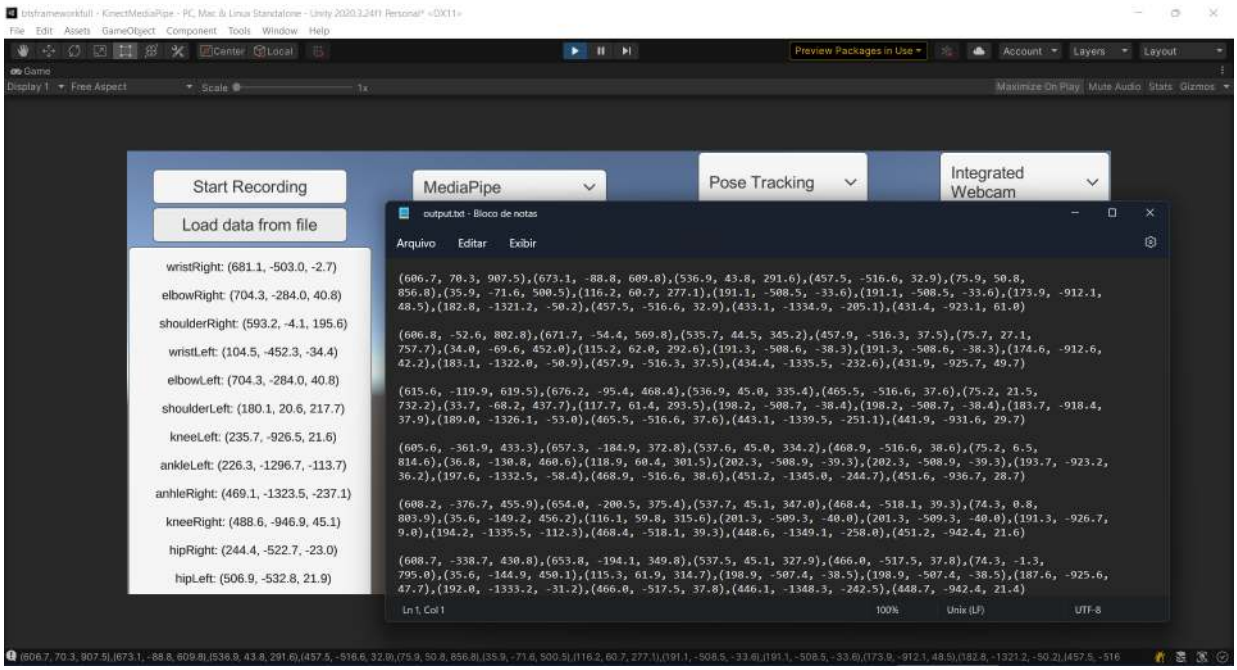


Figura 23 – Persistência e carregamento de dados usando o *framework*

O *Mediapipe* permite um rastreamento próprio da mão, rastreando 21 articulações. Utilizou-se o método ***GetPoseHand*** para capturar essas articulações, utilizando o

padrão de mapeamento do *framework*, o ***Poses.PoseHand***, conforme a Figura 24.

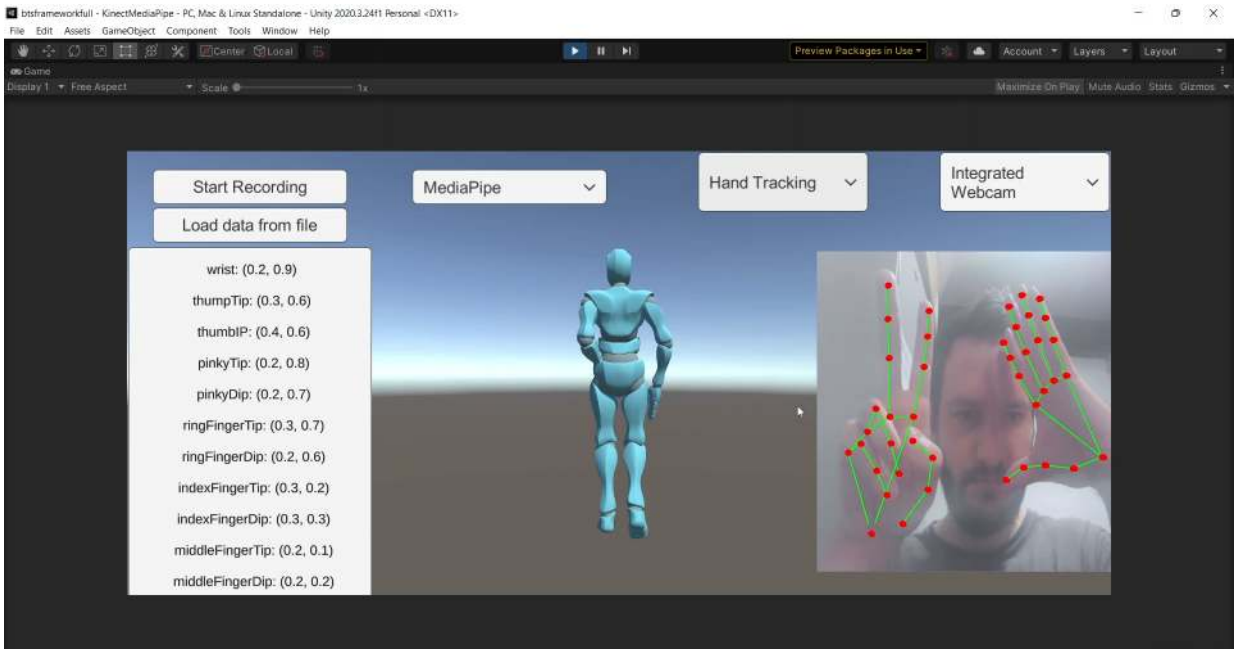


Figura 24 – Rastreamento da mão

O outro exemplo desenvolvido, demonstra a importância do sistema multi-agente. Uma aplicação *Android* foi criada com o *BSN* movimentando um cubo. O *BSN* é compatível apenas com o *Android*, portanto não é possível utilizá-lo em ambientes *desktop*. Assim, foi inserido na aplicação *Android* o **Agente**, ou seja, o ***Netclient***. Para fins de validação, foi desenvolvida uma outra aplicação para o *macOS*, com o ***Core*** do sistema multi-agente, isso significa, o ***Netserver***.

Dessa maneira, devido a comunicação entre os módulos do sistema multi-agente, é possível ter os dados do módulo *DataManager* disponíveis em outro sistema. Os *quaternion* são enviados utilizando o *DTO* criado no *framework*, girando dessa forma o cubo no *macOS*. Os dados são enviados pela rede. Portanto, é possível movimentar um cubo em uma aplicação no *macOS*, recebendo os dados do *BSN*, que por sua vez está conectada via *Bluetooth* em um *smartphone Android*, conforme a Figura 25.

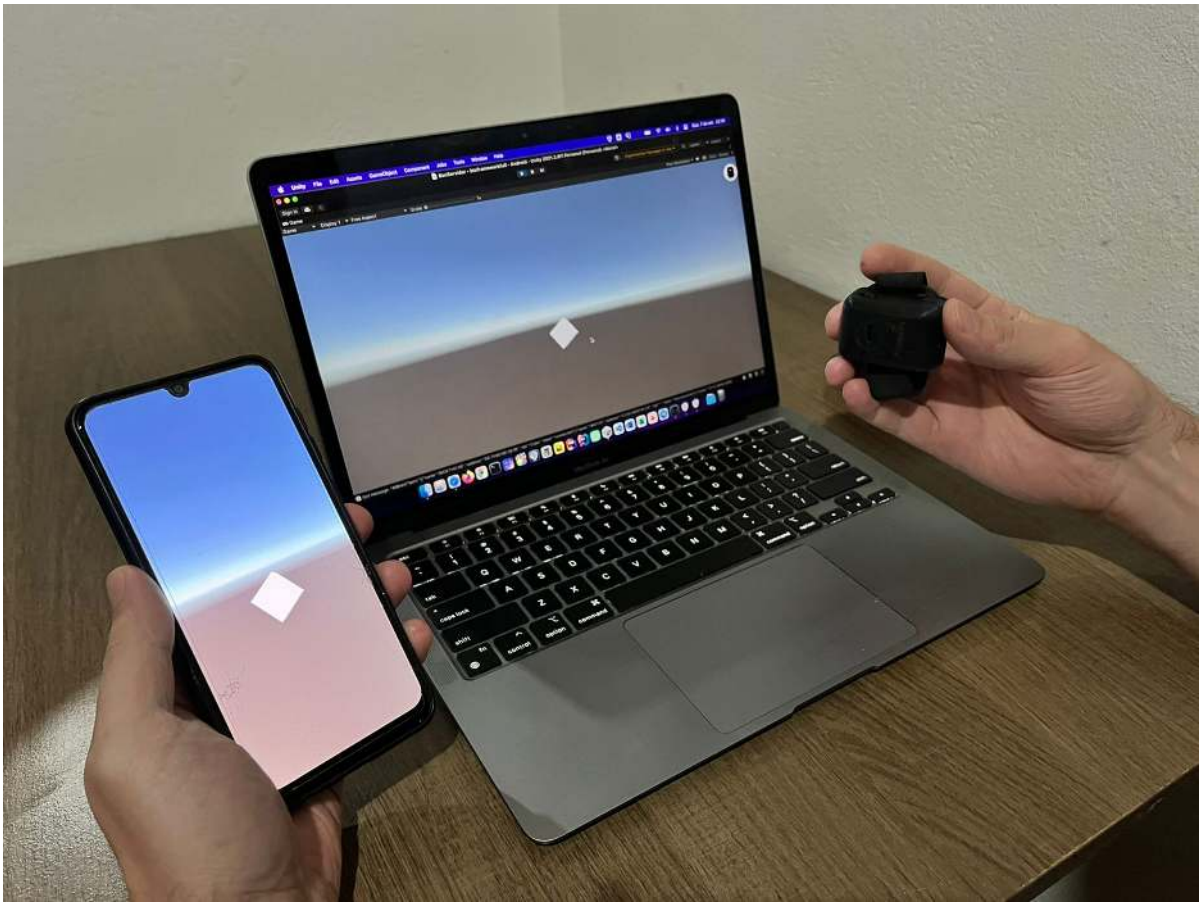


Figura 25 – Uso do sistema multi-agente

Para facilitar a utilização do *framework*, foi criado um site no endereço eletrônico https://www.btsframework.com.br, conforme a Figura 26. O site tem o intuito de auxiliar o aprendizado e emprego do *framework*. São apresentados tutoriais e exemplos para acelerar o desenvolvimento dos projetos. Dessa forma, com a documentação em um local adequado e de fácil acesso, tende a facilitar a usabilidade do *framework*, assim como manter uma paridade do código com a documentação ao longo do tempo.

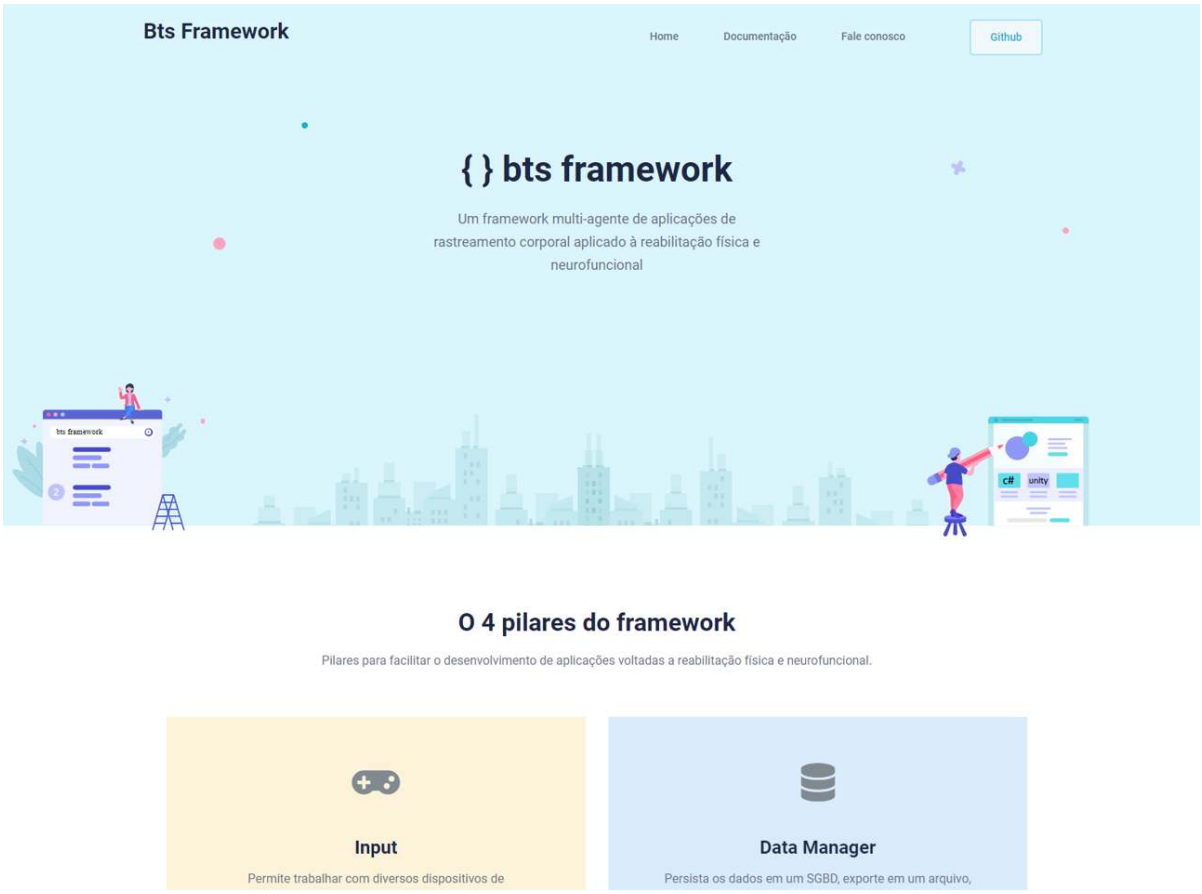


Figura 26 – Site do *framework*

7 Conclusão

Esta dissertação apresentou a criação do *BTS framework* capaz de facilitar a criação de aplicações voltadas à reabilitação física e neurofuncional. Para verificarmos e confirmarmos a aplicabilidade deste trabalho, foram desenvolvidas duas aplicações para demonstrar os módulos do *framework*.

Ao comparar as aplicações feitas com o *framework* e sem a utilização do mesmo, observou-se que é necessário muito mais tempo para conseguir o mesmo resultado, além disso, diversos outros fatores dificultam o desenvolvimento de uma aplicação de qualidade: o alto acoplamento do código sem o *framework*, a falta da construção de um projeto seguindo os conceitos da Engenharia de Software, documentação precária, dificuldade de reúso, entre outros.

Atualmente, o *framework* está disponível no *GitHub* de forma pública no endereço eletrônico https://github.com/felipereis/btsframework para que mais pessoas possam contribuir com o trabalho. Profissionais de saúde já estão utilizando alguns trabalhos do nosso grupo de pesquisa, o *feedback* está sendo positivo e eles estão engajados conosco para aperfeiçoar as ferramentas. Esperamos, que com a adoção do *framework*, mais trabalhos possam ser desenvolvidos com melhor qualidade de código, maior reúso, melhor manutenção e maior produtividade para que mais sistemas de *software* possam chegar aos pacientes, tendo assim, um impacto social mais significativo e, ao mesmo tempo, coletando mais dados que possam ajudar em tratamentos futuros.

Os dados capturados podem auxiliar a trazer informações importantes sobre como os pacientes se comportam a tratamentos específicos. Esses dados podem auxiliar a desenvolver melhores tratamentos e ajudar a evoluir os sistemas desenvolvidos. No futuro, algoritmos de aprendizado de máquina serão aplicados a esses dados.

Referências

ACM. *ACM Digital Library*. 2021. Disponível em: <https://dl.acm.org-/>https://dl.acm.org/. Citado na página 30.

AGGARWAL, D. et al. Lessons learnt from designing a smart clothing telehealth system for hospital use. In: *32nd Australian Conference on Human-Computer Interaction*. [S.l.: s.n.], 2020. p. 355–367. Citado 3 vezes nas páginas 33, 34 e 35.

AHMED, F. et al. Kalman filter-based noise reduction framework for posture estimation using depth sensor. In: IEEE. *2019 IEEE 18th International Conference on Cognitive Informatics & Cognitive Computing (ICCI* CC)*. [S.l.], 2019. p. 150–158. Citado 3 vezes nas páginas 33, 34 e 35.

AMIRI, A. M.; SHOAIB, N.; HIREMATH, S. V. A framework to enhance assistive technology based mobility tracking in individuals with spinal cord injury. In: IEEE. *2017 IEEE Global Conference on Signal and Information Processing (GlobalSIP)*. [S.l.], 2017. p. 467–471. Citado 3 vezes nas páginas 33, 34 e 36.

ARAÚJO, J. d. F. O. et al. Immersive brain puzzle: a virtual reality application aimed at the rehabilitation of post-stroke patients. In: IEEE. *2021 16th Iberian Conference on Information Systems and Technologies (CISTI)*. [S.l.], 2021. p. 1–6. Citado na página 27.

ARDUINO. *Arduino - Home*. 2022. Disponível em: <https://www.arduino-.cc>https://www.arduino.cc. Acesso em: 7 de abril de 2022 às 12:54h. Citado na página 21.

ARDUINO PROJECT HUB. *Ultrasonic Sensor HC-SR04 with Arduino Tutorial*. 2022. Disponível em: <https://create.arduino.cc-/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6>https://create.arduino.cc/projecthub/abdularbi17/ultrasonic-sensor-hc-sr04-with-arduino-tutorial-327ff6. Acesso em: 7 de abril de 2022 às 12:52h. Citado 2 vezes nas páginas 7 e 21.

BAHAR-FUCHS, A.; CLARE, L.; WOODS, B. Cognitive training and cognitive rehabilitation for mild to moderate alzheimer's disease and vascular dementia. *Cochrane database of systematic reviews*, John Wiley & Sons, Ltd, n. 6, 2013. Citado na página 17.

BARBOSA, T. T. L. *ReBase: sistema de aquisição, armazenamento e gerenciamento de dados de reabilitação neuromotora.* 2021. Monografia (Bacharel em Ciência da Computação), UFSJ (Universidade Federal de São João del-Rei), São João del-Rei, Brasil. Citado 3 vezes nas páginas 12, 17 e 25.

BARNES, T.; ENCARNAÇÃO, L. M.; SHAW, C. D. Serious games. *IEEE Computer Graphics and Applications*, IEEE, v. 29, n. 2, p. 18–19, 2009. Citado na página 12.

BRAGA, R. T. V. *Um processo para construção e instanciação de frameworks baseados em uma linguagem de padrões para um domínio específico.* Tese (Doutorado) — Universidade de São Paulo, 2002. Citado na página 14.

BRANDÃO, A. F. et al. Biomechanics sensor node for virtual reality: A wearable device applied to gait recovery for neurofunctional rehabilitation. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2020. p. 757–770. Citado 4 vezes nas páginas 12, 17, 21 e 22.

BROWN, L.; GARCÍA-VERGARA, S.; HOWARD, A. M. Evaluating the effect of robot feedback on motor skill performance in therapy games. In: IEEE. *2015 IEEE International Conference on Systems, Man, and Cybernetics*. [S.l.], 2015. p. 1060–1065. Citado 3 vezes nas páginas 33, 34 e 35.

CARNEIRO, C. H. G.; MAIA, P. H. M. Dataqi. net: a framework for specifying query criteria using the repository pattern. In: *Proceedings of the 34th Brazilian Symposium on Software Engineering*. [S.l.: s.n.], 2020. p. 520–525. Citado na página 57.

CHANG, R. et al. Kinect-based framework for motor rehabilitation. In: IEEE. *2016 International Conference on Robotics, Automation and Sciences (ICORAS)*. [S.l.], 2016. p. 1–4. Citado 4 vezes nas páginas 33, 34, 35 e 36.

CHRISTOPHER, A. et al. *FIKSDAHL! KING, I., ANGEL, S., A Pattern Language*. [S.l.]: Oxford University Press, New York, 1977. Citado na página 15.

CIFUENTES, C. A. et al. Multimodal human–robot interaction for walker-assisted gait. *IEEE Systems Journal*, IEEE, v. 10, n. 3, p. 933–943, 2014. Citado na página 12.

CORÉN, A. *Ruffles*. [S.l.]: GitHub, 2020. https://github.com/MidLevel/Ruffles. Citado na página 39.

CORREA, D. S. O. et al. Mobile robots navigation in indoor environments using kinect sensor. In: IEEE. *2012 Second Brazilian Conference on Critical Embedded Systems*. [S.l.], 2012. p. 36–41. Citado na página 23.

DHAWAN, A. P.; HUANG, B. H.; KIM, D.-s. *Principles and advanced methods in medical imaging and image analysis*. [S.l.]: World Scientific, 2008. Citado 3 vezes nas páginas 7, 19 e 20.

DIOS, P. F. de; CHUNG, P. W. H.; MENG, Q. Landmark-based methods for temporal alignment of human motions. *ieee Computational intelligenCe magazine*, IEEE, v. 9, n. 2, p. 29–37, 2014. Citado 4 vezes nas páginas 33, 34, 35 e 36.

DROBNJAKOVIC, F. et al. Fusing data from inertial measurement units and a 3d camera for body tracking. In: IEEE. *2018 IEEE International Instrumentation and Measurement Technology Conference (I2MTC)*. [S.l.], 2018. p. 1–6. Citado 3 vezes nas páginas 33, 34 e 35.

DUARTE, N.; POSTOLACHE, O.; SCHARCANSKI, J. Ksgphysio-kinect serious game for physiotherapy. In: IEEE. *2014 International Conference and Exposition on Electrical and Power Engineering (EPE)*. [S.l.], 2014. p. 606–611. Citado na página 23.

DURVE, I. et al. Machine learning approach for physiotherapy assessment. In: IEEE. *2019 International Conference on Advances in Computing, Communication and Control (ICAC3)*. [S.l.], 2019. p. 1–5. Citado 4 vezes nas páginas 33, 34, 35 e 36.

EXTERKOETTER, F. et al. Blendwork: framework orientado a objetos para desenvolvimento rápido de aplicações comerciais cliente/servidor. Florianópolis, SC, 2003. Citado na página 14.

FERREIRA, I. A.; RESENDE, A. M. P. de; COSTA, H. A. X. Análise do impacto da aplicação de padrões de projeto na manutenibilidade de um sistema orientado a objetos. *Monografia de Graduação do Departamento de Ciência da Computação da Universidade Federal de Lavras, 78p*, 2012. Citado 2 vezes nas páginas 15 e 63.

FORSYTH, D. A.; PONCE, J. *Computer vision: a modern approach*. [S.l.]: Pearson„ 2012. Citado na página 20.

FOWLER, M. *Padrões de arquitetura de aplicações corporativas*. [S.l.]: Bookman, 2009. Citado na página 15.

GAMA, A. D. et al. Poster: improving motor rehabilitation process through a natural interaction based system using kinect sensor. In: IEEE. *2012 IEEE Symposium on 3D User Interfaces (3DUI)*. [S.l.], 2012. p. 145–146. Citado na página 13.

GAMMA, E. et al. *Design patterns: elements of reusable object-oriented software*. [S.l.]: Pearson Deutschland GmbH, 1995. Citado 3 vezes nas páginas 14, 15 e 48.

GAUTHIER, S.; CRETU, A.-M. Human movement quantification using kinect for in-home physical exercise monitoring. In: IEEE. *2014 IEEE International Conference on Computational Intelligence and Virtual Environments for Measurement Systems and Applications (CIVEMSA)*. [S.l.], 2014. p. 6–11. Citado 2 vezes nas páginas 33 e 35.

GAVRILA, D. M. The visual analysis of human movement: A survey. *Computer vision and image understanding*, Elsevier, v. 73, n. 1, p. 82–98, 1999. Citado na página 20.

GOOGLE. *MediaPipe Hands*. 2022. Disponível em: <https://google.github.io/mediapipe-/solutions/hands.html>https://google.github.io/mediapipe/solutions/hands.html. Acesso em: 06 de agosto de 2022 às 23:26h. Citado 2 vezes nas páginas 7 e 55.

GOOGLE BLOG. *On-Device, Real-Time Hand Tracking with MediaPipe*. 2019. Disponível em: <https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html>https://ai.googleblog.com/2019/08/on-device-real-time-hand-tracking-with.html. Acesso em: 18 de abril de 2021 às 12:13h. Citado 2 vezes nas páginas 7 e 22.

GORDON, C.; ROOPCHAND-MARTIN, S.; GREGG, A. Potential of the nintendo wii™ as a rehabilitation tool for children with cerebral palsy in a developing country: a pilot study. *Physiotherapy*, Elsevier, v. 98, n. 3, p. 238–242, 2012. Citado na página 12.

GRAAFF, K. M. Van de. Anatomia humana. In: *Anatomia humana*. [S.l.: s.n.], 2003. p. 840–840. Citado na página 18.

GWAK, M. et al. Extra: Exercise tracking and analysis platform for remote-monitoring of knee rehabilitation. In: IEEE. *2019 IEEE 16th International Conference on Wearable and Implantable Body Sensor Networks (BSN)*. [S.l.], 2019. p. 1–4. Citado 4 vezes nas páginas 33, 34, 35 e 36.

HUNT, C. L. et al. Predictive trajectory estimation during rehabilitative tasks in augmented reality using inertial sensors. In: IEEE. *2018 IEEE Biomedical Circuits and Systems Conference (BioCAS)*. [S.l.], 2018. p. 1–4. Citado 3 vezes nas páginas 33, 34 e 35.

IEEE. *IEEE Xplore*. 2021. Disponível em: <https://ieeexplore.ieee.org/Xplore/home-.jsp>https://ieeexplore.ieee.org/Xplore/home.jsp. Citado na página 30.

JENNINGS, N. R.; WOOLDRIDGE, M. J. *Agent technology: foundations, applications, and markets*. [S.l.]: Springer Science & Business Media, 2012. Citado na página 16.

KITSUNEZAKI, N. et al. Kinect applications for the physical rehabilitation. In: IEEE. *2013 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. [S.l.], 2013. p. 294–299. Citado na página 12.

KURILLO, G. et al. Tele-mfast: Kinect-based tele-medicine tool for remote motion and function assessment. In: *MMVR*. [S.l.: s.n.], 2014. p. 215–221. Citado na página 23.

LARSEN, S.; AARSETH, E. Level design patterns. *IT University of Copenhagen*, 2006. Citado na página 15.

MACHADO, L.; CARDOSO, A. Dispositivos de entrada e saída para sistemas de realidade virtual. *Fundamentos e tecnologia de Realidade Virtual e Aumentada*, p. 39–50, 2006. Citado na página 20.

MACKNOJIA, R. et al. Calibration of a network of kinect sensors for robotic inspection over a large workspace. In: IEEE. *2013 IEEE Workshop on Robot Vision (WORV)*. [S.l.], 2013. p. 184–190. Citado na página 23.

MAES, P. Behavior-based artificial intelligence. In: *Proceedings of the Fifteenth Annual Meeting of the Cognitive Science Society*. [S.l.: s.n.], 1993. p. 74–83. Citado na página 16.

MAGGIORINI, D.; RIPAMONTI, L.; SCAMBIA, A. Videogame technology to support seniors. In: *Proceedings of the 5th International ICST Conference on Simulation Tools and Techniques*. [S.l.: s.n.], 2012. p. 270–277. Citado 3 vezes nas páginas 33, 34 e 35.

MAGGIORINI, D.; RIPAMONTI, L. A.; ZANON, E. Supporting seniors rehabilitation through videogame technology: A distributed approach. In: IEEE. *2012 Second International Workshop on Games and Software Engineering: Realizing User Engagement with Game Engineering Techniques (GAS)*. [S.l.], 2012. p. 16–22. Citado 3 vezes nas páginas 33, 34 e 35.

METSIS, V.; SMITH, K. S.; GOBERT, D. Integration of virtual reality with an omnidirectional treadmill system for multi-directional balance skills intervention. In: IEEE. *2017 International Symposium on Wearable Robotics and Rehabilitation (WeRob)*. [S.l.], 2017. p. 1–2. Citado 3 vezes nas páginas 33, 34 e 35.

MOTIIAN, S. et al. Automated extraction and validation of children's gait parameters with the kinect. *Biomedical engineering online*, Springer, v. 14, n. 1, p. 1–36, 2015. Citado 2 vezes nas páginas 7 e 19.

MUBARAK, H. Developing flexible software using agent-oriented software engineering. *IEEE software*, IEEE, v. 25, n. 5, p. 12–15, 2008. Citado na página 16.

MYAGMARBAYAR, N. et al. Human body contour data based activity recognition. In: IEEE. *2013 35th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.], 2013. p. 5634–5637. Citado 3 vezes nas páginas 33, 34 e 35.

O'HARE, G. M.; JENNINGS, N. R. *Foundations of distributed artificial intelligence*. [S.l.]: John Wiley & Sons, 1996. v. 9. Citado na página 16.

OLIVEIRA, C. E. N. de; SALINA, M. E.; ANNUNCIATO, N. F. Fatores ambientais que influenciam a plasticidade do snc. *Acta Fisiátrica*, v. 8, n. 1, p. 6–13, 2001. Citado na página 17.

OÑA, E. D.; BALAGUER, C.; JARDÓN, A. Towards a framework for rehabilitation and assessment of upper limb motor function based on serious games. In: IEEE. *2018 IEEE 6th International Conference on Serious Games and Applications for Health (SeGAH)*. [S.l.], 2018. p. 1–7. Citado 3 vezes nas páginas 33, 34 e 35.

POSTOLACHE, O. Physical rehabilitation assessment based on smart training equipment and mobile apps. In: IEEE. *2015 E-Health and Bioengineering Conference (EHB)*. [S.l.], 2015. p. 1–6. Citado na página 12.

PRESSMAN, R.; MAXIM, B. *Engenharia de Software-8ª Edição*. [S.l.]: McGraw Hill Brasil, 2016. Citado na página 15.

RAHMAN, M. A. et al. Modeling therapy rehabilitation sessions using non-invasive serious games. In: IEEE. *2014 IEEE International Symposium on Medical Measurements and Applications (MeMeA)*. [S.l.], 2014. p. 1–4. Citado 4 vezes nas páginas 33, 34, 35 e 36.

RIBEIRO, E. H. *Rastreamento Corporal por meio de Sensores Biomecânicos.* 2021. Monografia (Bacharel em Ciência da Computação), UFSJ (Universidade Federal de São João del-Rei), São João del-Rei, Brasil. Citado 2 vezes nas páginas 17 e 25.

RODRIGUEZ-MARTIN, D. et al. Svm-based posture identification with a single waist-located triaxial accelerometer. *Expert Systems with Applications*, Elsevier, v. 40, n. 18, p. 7203–7211, 2013. Citado 2 vezes nas páginas 13 e 20.

SANTOS, A. et al. Multi-sensor exercise-based interactive games for fall prevention and rehabilitation. In: IEEE. *2015 9th International Conference on Pervasive Computing Technologies for Healthcare (PervasiveHealth)*. [S.l.], 2015. p. 65–71. Citado 2 vezes nas páginas 34 e 35.

SILVA, E. Q. d. *Um framework baseado em componentes para desenvolvimento de aplicações web e um processo de instanciação associado*. Tese (Doutorado) — Universidade de São Paulo, 2006. Citado na página 14.

SILVA, R. P. e. Suporte ao desenvolvimento e uso de frameworks e componentes. Instituto de Informática da Universidade Federal do Rio Grande do Sul., 2000. Citado na página 13.

SILVA, S. R. da et al. Reabilitação funcional para membros superiores pós-acidente vascular encefálico. *Fisioterapia Brasil*, v. 4, n. 3, p. 195–199, 2003. Citado na página 17.

SILVEIRA, R. A. Modelagem orientada a agentes aplicada a ambientes inteligentes distribuídos de ensino: Jade: Java agent framework for distance learning environments. 2001. Citado na página 16.

SINHA, S. et al. Accurate estimation of joint motion trajectories for rehabilitation using kinect. In: IEEE. *2017 39th Annual International Conference of the IEEE Engineering in Medicine and Biology Society (EMBC)*. [S.l.], 2017. p. 3864–3867. Citado 2 vezes nas páginas 33 e 34.

STOWERS, J.; HAYES, M.; BAINBRIDGE-SMITH, A. Altitude control of a quadrotor helicopter using depth map from microsoft kinect sensor. In: IEEE. *2011 IEEE International Conference on Mechatronics*. [S.l.], 2011. p. 358–362. Citado na página 23.

STRÖMBÄCK, D.; HUANG, S.; RADU, V. Mm-fit: Multimodal deep learning for automatic exercise logging across sensing devices. *Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies*, ACM New York, NY, USA, v. 4, n. 4, p. 1–22, 2020. Citado 4 vezes nas páginas 33, 34, 35 e 36.

STÜTZ, T. et al. An interactive 3d health app with multimodal information representation for frozen shoulder. In: *Proceedings of the 19th International Conference on Human-Computer Interaction with Mobile Devices and Services*. [S.l.: s.n.], 2017. p. 1–11. Citado 2 vezes nas páginas 34 e 35.

TANG, R. et al. Physio@ home: Exploring visual guidance and feedback techniques for physiotherapy exercises. In: *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*. [S.l.: s.n.], 2015. p. 4123–4132. Citado na página 34.

TWAMLEY, E. W. *A review of Donald T. Stuss, Gordon Winocur, & Ian H. Robertson (Eds.).(2008). Cognitive Neurorehabilitation: Evidence and Application . New York: Cambridge University Press. Pp. 606. ISBN: 978–0–521–87133–4. 180.00*. [S.l.]: Taylor & Francis, 2010. Citado na página 17.

UNITY. *Unity Real-Time Development Platform 3D, 2D VR & AR*. 2021. Disponível em: <https://www.unity.com/>https://www.unity.com/. Acesso em: 15 de agosto de 2021 às 10:26h. Citado na página 18.

UNITY. *Unity – Student. 2021*. 2021. Disponível em: <https://store.unity.com-/academic/unity-student>https://store.unity.com/academic/unity-student. Acesso em: 15 de agosto de 2021 às 10:17h. Citado na página 17.

VAGHETTI, C. A. et al. Using exergames as social networks: testing the flow theory in the teaching of physical education. In: *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)*. [S.l.: s.n.], 2012. v. 23, n. 1. Citado na página 23.

VALENTE, F. R. et al. A framework for neuromotor and neurofunctional rehabilitation using multi-agent systems. *Symposium on Virtual and Augmented Reality*, 2022. Citado na página 13.

VALENTE, F. R. et al. A multi-agent body tracking application framework applied to physical and neurofunctional rehabilitation. In: SPRINGER. *International Conference on Computational Science and Its Applications*. [S.l.], 2022. p. 459–472. Citado na página 13.

WALTEMATE, T. et al. Realizing a low-latency virtual reality environment for motor learning. In: *Proceedings of the 21st ACM symposium on virtual reality software and technology*. [S.l.: s.n.], 2015. p. 139–147. Citado 2 vezes nas páginas 34 e 36.

WEBB, J.; ASHLEY, J. *Beginning kinect programming with the microsoft kinect SDK*. [S.l.]: Apress, 2012. Citado 2 vezes nas páginas 7 e 23.

WHITALL, J. et al. Repetitive bilateral arm training with rhythmic auditory cueing improves motor function in chronic hemiparetic stroke. *Stroke*, Am Heart Assoc, v. 31, n. 10, p. 2390–2395, 2000. Citado na página 12.

WOOLDRIDGE, M. *An introduction to multiagent systems*. [S.l.]: John wiley & sons, 2009. Citado na página 16.

YASSIN, A.; FAYAD, M. *Domain-Specif Application Frameworks: Frameworks Experience by Industry*. [S.l.]: John Wiley & Sons, 2000. Citado na página 14.