Universidade de Brasília
Faculdade de Tecnologia

Estudo e Desenvolvimento de um Simulador de Realidade Virtual Aplicado a Cirurgia Robótica

Gabriel Ângelo Alves de Araújo

PROJETO FINAL DE CURSO

ENGENHARIA DE CONTROLE E AUTOMAÇÃO

Brasília

2023

Universidade de Brasília
Faculdade de Tecnologia

Estudo e Desenvolvimento de um Simulador de Realidade Virtual Aplicado a Cirurgia Robótica

Gabriel Ângelo Alves de Araújo

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Orientador: Prof. Walter de Britto Vidal Filho

Brasília
2023

Alves de Araújo, Gabriel Ângelo.
A474e Estudo e Desenvolvimento de um Simulador de Realidade Virtual Aplicado a Cirurgia Robótica / Gabriel Ângelo Alves de Araújo; orientador Walter de Britto Vidal Filho. -- Brasília, 2023.
81 p.

Projeto Final de Curso (Engenharia de Controle e Automação) -- Universidade de Brasília, 2023.

1. Cirurgia Robótica. 2. Simulador. 3. Realidade Virtual. 4. Da Vinci. I. Vidal Filho, Walter de Britto, orient. II. Título

Universidade de Brasília
Faculdade de Tecnologia

Estudo e Desenvolvimento de um Simulador de Realidade Virtual Aplicado a Cirurgia Robótica

Gabriel Ângelo Alves de Araújo

Projeto Final de Curso submetido como requisito parcial para obtenção do grau de Engenheiro de Controle e Automação

Trabalho aprovado. Brasília, 11 de Dezembro de 2023:

Prof. Walter de Britto Vidal Filho-ENM/UnB
Orientador

Prof. Eugênio Libório Feitosa Fortaleza
Examinador interno

Prof. Fernando Merege
Examinador externo

Brasília
2023

Este trabalho é dedicado às crianças adultas que,
quando pequenas, sonharam em se tornar cientistas.

Agradecimentos

Agradeço primeiramente à minha família por todo amor, apoio e incentivo, especialmente às minhas irmãs Amanda Beatriz e Natália Rebeca, por estarem sempre ao meu lado. Agradeço a todos os amigos e colegas de curso que ajudaram a tornar essa jornada mais agradável. Agradeço também ao orientador, Prof. Britto, por todo auxílio prestado no desenvolvimento deste projeto.

Resumo

O presente trabalho aborda o desenvolvimento de um protótipo de simulador em realidade virtual para um robô de cirurgia robótica, utilizando a Unreal Engine como plataforma principal de desenvolvimento. Inspirado em um simulador já validado, o FlexVR, o objetivo primordial deste projeto é aprimorar as técnicas empregadas no desenvolvimento do simulador, colocando à prova a eficácia do software escolhido e os princípios fundamentais utilizados. A comparação com o simulador FlexVR serve como referência para avaliar as melhorias implementadas, identificando pontos fortes e áreas passíveis de refinamento. Além disso, o trabalho destaca os desafios enfrentados durante o desenvolvimento do protótipo, explorando questões técnicas, de usabilidade e de eficácia na simulação. Na análise de resultados, será apresentada uma comparação qualitativa entre o simulador em desenvolvimento e o simulador já validado (FlexVR). Por fim, a conclusão do trabalho destaca não apenas as conquistas alcançadas no protótipo, mas também delineia caminhos potenciais para a evolução contínua do simulador. Os resultados obtidos neste trabalho visam contribuir para que, em um futuro próximo, esse protótipo possa evoluir para um simulador completo, proporcionando avanços significativos na área de simulação em cirurgia robótica.

Palavras-chave: Cirurgia Robótica. Simulador. Realidade Virtual. Da Vinci.

Abstract

This Undergraduate Work II addresses the development of a virtual reality simulator prototype for a robotic surgery robot, using the Unreal Engine as the main development platform. Inspired by an already validated simulator, FlexVR, the primary objective of this project is to improve the techniques used in the development of the simulator, testing the effectiveness of the chosen software and the fundamental principles used. The comparison with the FlexVR simulator serves as a reference to evaluate the improvements implemented, identifying strengths and areas subject to refinement. Furthermore, the work highlights the challenges faced during the development of the prototype, exploring technical, usability and effectiveness issues in the simulation. In the analysis of results, a qualitative comparison will be presented between the simulator under development and the simulator already validated (FlexVR). Finally, the conclusion of Graduation Work II highlights not only the achievements achieved in the prototype, but also outlines potential paths for the continued evolution of the simulator. The results obtained in this work aim to contribute so that, in the near future, this prototype can evolve into a complete simulator, providing significant advances in the area of simulation in robotic surgery.

Keywords: Robotic Surgery. simulator. Virtual reality. Da Vinci.

Lista de ilustrações

Lista de tabelas

Sumário

1 Introdução

A evolução da cirurgia robótica tem testemunhado avanços notáveis, impulsionados pela incessante inovação tecnológica na área da saúde. A introdução de robôs cirúrgicos promete procedimentos mais precisos e menos invasivos, redefinindo os padrões de cuidados médicos. No entanto, o desenvolvimento de habilidades essenciais para operar esses sistemas requer treinamento especializado, estimulando a necessidade de simuladores inovadores que capacitem profissionais de saúde.

A simulação torna-se uma peça-chave no contexto da cirurgia robótica, proporcionando um ambiente seguro e controlado para a prática de habilidades específicas, aprimorando a destreza do cirurgião e minimizando riscos associados à curva de aprendizado. Contudo, desenvolver um simulador eficaz demanda uma abordagem criteriosa, envolvendo diversos elementos-chave para garantir realismo e utilidade na prática cirúrgica virtual.

Diante desse cenário, este Projeto Final de Curso se propõe a explorar e apresentar os desafios e avanços associados ao desenvolvimento de um simulador de um robô de cirurgia robótica, construído em uma engine de desenvolvimento de jogos e simuladores chamada Unreal Engine, que será melhor abordada do decorrer do projeto. No desenvolvimento deste simulador foi inspirado em um simulador já existente e validado, conhecido como FlexVr. Essa abordagem de construção sobre uma estrutura previamente validada não apenas acelera o processo de desenvolvimento, mas também assegura uma base sólida para a autenticidade e eficácia do simulador resultante.

2 Revisão Bibliográfica

Essa revisão terá como objetivo a exposição de trabalhos e pesquisas semelhantes ao projeto desenvolvido neste artigo, com o intuito de situar o estudo em questão em relação aos demais estudos já realizados. Além disso, será analisada criticamente o uso do Simulador de Realidade Virtual aplicado à cirurgia robótica, identificando suas vantagens, limitações e potencialidades como ferramenta de treinamento para cirurgiões que realizam procedimentos robóticos.

2.1 Realidade Virtual

A Realidade Virtual (RV) é uma tecnologia que permite a criação de ambientes virtuais imersivos, em que o usuário pode interagir e explorar. Com essa tecnologia, é possível simular situações complexas, como cirurgias, voos espaciais, mergulhos e experiências de viagem, proporcionando um ambiente seguro e controlado para o usuário. Além disso, a RV pode ser usada para criar simulações de treinamento em diversas áreas, como medicina, aviação e militares.

A RV usa dispositivos como óculos específicos, fones de ouvido, luvas e sensores de movimento para criar uma experiência imersiva e realista. A tecnologia tem sido usada em diversos campos, como entretenimento, educação, treinamento, design e saúde.

2.1.1 Realidade virtual x realidade aumentada

Embora possam parecer semelhantes, a Realidade Virtual (RV) e a Realidade Aumentada (RA) são tecnologias diferentes com funcionalidades distintas. A Realidade Virtual é uma tecnologia que cria um ambiente totalmente simulado em que o usuário pode imergir. Ela utiliza dispositivos como óculos de RV, fones de ouvido, luvas e sensores de movimento para criar uma experiência imersiva, onde o usuário é capaz de interagir com objetos e ambientes virtuais em tempo real. O ambiente simulado é projetado para substituir a realidade, dando ao usuário uma sensação de presença em um mundo virtual.

Já a Realidade Aumentada é uma tecnologia que adiciona elementos virtuais a um ambiente real. Ela utiliza dispositivos como smartphones, tablets e óculos de RA para sobrepor informações digitais, como imagens, gráficos, vídeos e sons, a um ambiente real. A tecnologia é usada para melhorar a percepção do usuário do mundo real e fornecer informações adicionais ou contextualizar elementos que estão ao seu redor.

resumidamente, a diferença fundamental entre a Realidade Virtual e a Realidade Aumentada é que a RV cria um ambiente totalmente virtual, enquanto a RA adiciona

elementos virtuais a um ambiente real.

2.1.2 Realidade virtual na medicina

A realidade virtual vem se tornando cada vez mais um aliado importante da medicina, seja para capacitação de seus profissionais, criação de ferramentas para facilitar diagnósticos e até mesmo forma de tratamento de doenças ligadas a questões emocionais. Como exemplo, tem-se a sua utilização para treinar estudantes de medicina e cirurgiões em procedimentos médicos e cirúrgicos complexos. Os simuladores de Realidade Virtual oferecem uma experiência imersiva que ajuda a aprimorar as habilidades dos profissionais de saúde antes de eles aplicarem os procedimentos em pacientes reais.

Além disso, a RV é utilizada para tratar pacientes com problemas de saúde mental, como transtorno de estresse pós-traumático (TEPT), ansiedade e depressão. Os pacientes podem ser expostos a ambientes virtuais que os ajudam a lidar com seus medos e ansiedades, o que pode melhorar sua saúde mental. Outra aplicação clara da realidade virtual na medicina é a criação de simuladores para Cirurgia robótica, pois ela é capaz de gerar ambientes de treinamento para profissionais que tem o objetivo de aprender a manipular robôs de cirurgia robótica.

Esses são apenas alguns exemplos de como a Realidade Virtual tem sido utilizada na medicina. A tecnologia tem um grande potencial para transformar a forma como os profissionais de saúde diagnosticam, tratam e treinam para procedimentos médicos e cirúrgicos.

2.1.3 Geradores de imagens estereoscópicas

Em um sistema de realidade virtual, diferentes dispositivos são usados para gerar imagens estereoscópicas, criando uma experiência visual imersiva. Mas antes de explicar quais são esses dispositivos é importante, primeiro, entender o que é uma imagem estereoscópica. A estereoscopia refere-se à percepção de imagens tridimensionais, resultante da visualização de um objeto a partir de dois pontos de vista ligeiramente diferentes. No ser humano isso acontece porque cada olho enxerga o objeto de um ângulo ligeiramente diferente, como demonstra o exemplo da Figura 2.1. Com isso, o cérebro ao processar as informações recebidas, consegue fazer com que tenhamos a noção de profundidade (RIBAS, G. C.; RIBAS, E. C.; RODRIGUES JUNIOR, 2006).

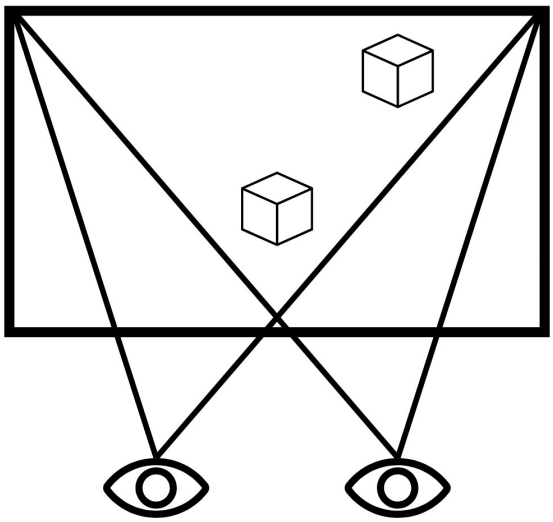

Figura 2.1 – Esquema da visão binocular

Os sistemas que utilizam gráficos tridimensionais buscam reproduzir imagens estereoscópicas para proporcionar uma experiência visual mais imersiva. A tentativa de uma reprodução realista dessas imagens precisa ser feita de forma precisa e sincronizada para cada olho, a fim de criar a sensação de profundidade.

Para essa finalidade, esses sistemas recorrem a equipamentos como os óculos 3D. De maneira bem resumida, esses óculos fornecem uma imagem diferente para cada olho, o que cria a ilusão de profundidade quando o cérebro as processa em conjunto, permitindo que o usuário explore ambientes virtuais com uma perspectiva tridimensional. As tecnologias envolvida nesses óculos serão explanadas com mais detalhes na próxima seção(Óculos 3D passivo x Óculos 3D ativo). Além dos óculos 3D, outros dispositivos, como telas 3D, projetores estereoscópicos e holografia, também podem ser usados para proporcionar a visualização de imagens estereoscópicas em um contexto de realidade virtual. No entanto, os óculos 3D são os mais comuns e populares para essa finalidade.

2.1.4 Óculos 3D passivo x Óculos 3D ativo

Os óculos passivos e ativos diferem na forma como exibem imagens estereoscópicas para criar a ilusão de profundidade e 3D. Os óculos passivos utilizam lentes coloridas para criar a ilusão 3D. Esses óculos possuem lentes de cores diferentes, geralmente uma vermelha e outra ciano (ou azul). O conteúdo 3D é projetado de maneira especial, com duas versões da imagem: uma para o olho esquerdo e outra para o olho direito. Cada lente do óculos permite que apenas uma versão da imagem seja vista pelo olho correspondente. O cérebro combina essas duas imagens, levando à percepção de profundidade e tridimensionalidade.

O ponto negativo dos óculos passivos é que a resolução da imagem gerada cai

pela metade, pois para utilizar esses óculos o gerados de imagens tem que dividir a mesma imagem duas partes, uma para cada um dos filtros do óculos. Em contrapartida, os óculos passivos são bem mais baratos que os ativos, pois utilizam uma tecnologia relativamente simples.

Já os óculos ativos são comumente usados em cinemas 3D e requerem uma tela especial polarizada. A tela exibe duas versões da imagem, polarizadas de forma diferente, uma para cada olho. Os óculos possuem lentes polarizadas correspondentes às polarizações das imagens. Cada lente do óculos permite que apenas a imagem polarizada correta seja vista pelo olho correspondente. O cérebro combina essas duas imagens polarizadas, criando a sensação de profundidade e tridimensionalidade.

O ponto negativo dos óculos ativos é o seu valor de mercado, que por se tratar de uma tecnologia mais complexa que o passivo, acaba tendo um custo maior. Além dos próprios óculos serem mais complexos, o gerador da imagem também tem que ser capaz de gerar imagens a uma taxa de exibição, pelo menos, duas vezes maior que o convencional, sendo mais um agravante para o seu custo. Em contrapartida, os óculos ativos oferecem melhor resolução e menor interferência na qualidade da imagem ao mover a cabeça.

2.2 Simuladores de realidade virtual aplicados a cirurgia robótica

Os simuladores de realidade virtual aplicados à cirurgia robótica é uma tecnologia que permite aos cirurgiões praticar e aprimorar suas habilidades em cirurgia robótica por meio de um ambiente virtual que simula situações e procedimentos médicos em 3D. Esse ambiente é criado por meio de uma combinação de tecnologias, como gráficos 3D, interação háptica e feedback em tempo real, que permitem aos cirurgiões interagir com objetos virtuais como se estivessem realizando um procedimento em um robô de cirurgia real. O objetivo do simulador é proporcionar um ambiente seguro e controlado para o treinamento de cirurgiões, ajudando a reduzir o tempo de aprendizagem, melhorar a precisão e eficácia, e reduzir custos em relação ao treinamento em pacientes reais.

2.2.1 Exemplos de Simuladores de Realidade Virtual Aplicados à Cirurgia Robótica Presentes no Mercado

1. **da Vinci® Skills Simulator™:**

    - Este simulador é associado ao sistema robótico da Vinci® e visa aprimorar as habilidades cirúrgicas do operador. Proporciona uma interface realista e cenários específicos para treinamento.(DA VINCI SURGERY COMMUNITY, 2023)

2. **Mimic dV-Trainer:**

    - Instalado pela primeira vez como protótipo no Departamento de Urologia da Universidade de Indiana em 2007, o dV-Trainer é o simulador original de cirurgia robótica. Continua a ser o único simulador robótico autónomo que foi validado de forma independente em estudos publicados. Além disso, o Simulador de Habilidades Cirúrgicas Intuitivas® para o Sistema Cirúrgico da Vinci Si é baseado na tecnologia de simulação proprietária da Mimic, MSim™.(MIMIC TECHNOLOGIES, 2023)

3. **ROSS (RobotiX Mentor):**

    - O RobotiX Mentor oferece aos cirurgiões de todos os níveis de especialização a oportunidade de praticar efetivamente as habilidades motoras e cognitivas necessárias para realizar a cirurgia robótica. Possui um poderoso sistema de gerenciamento de aprendizagem, MentorLearn, facilita a personalização de cursos, a avaliação de acordo com suas necessidades e o monitoramento do progresso dos usuários.(SURGICAL SCIENCE, 2023b)

4. **FlexVR**

    - O FlexVR ® é uma plataforma de treinamento flexível e portátil projetada para ensinar habilidades básicas e avançadas para cirurgia robótica a um público mais amplo, desde estudantes de medicina até cirurgiões experientes. O FlexVR permite que os alunos aprendam rapidamente os fundamentos da cirurgia robótica, como manipulação de instrumentos com punho, controle de câmera, embreagem, controle de energia, condução de agulha e sutura.(SURGICAL SCIENCE, 2023a)

2.2.2 Objetivos buscados com a utilização de simuladores de cirurgia robótica

A utilização de simuladores de cirurgia robótica tem como objetivo buscar a criação de treinamento mais seguro e controlado, Onde os cirurgiões em treinamento pratiquem e aprimorem suas habilidades em um ambiente similar ao real, sem colocar pacientes em risco. Além disso, com a possibilidade de realizar treinamentos em um ambiente simulado, o tempo necessário para que um cirurgião em treinamento adquira as habilidades necessárias é reduzido. Isso é importante para acelerar a formação de novos profissionais e aumentar a eficiência dos programas de treinamento (BELOTTO, 2022).

Outro ponto importane é que a prática em um simulador de cirurgia robótica pode ajudar os cirurgiões a melhorar a precisão de seus movimentos e a eficácia dos procedimentos. Isso pode resultar em melhores resultados para os pacientes e em um menor tempo de recuperação (BELOTTO, 2022).

Em resumo, a utilização de simuladores de cirurgia robótica traz benefícios tanto para os cirurgiões quanto para os pacientes, proporcionando um ambiente de treinamento mais seguro, controlado e eficiente.

2.2.3 Ambientes de desenvolvimento de Simuladores de realidade virtual

Como qualquer outro tipo de aplicação, um simulador de realidade virtual pode ser desenvolvido do zero através da utilização de programação pura, ou seja, desenvolvendo toda a engine do simulador do zero. Por mais que esse método seja eficiente, alcançar resultados significativamente realistas demandam um carga e complexidade de trabalho extremamente elevada, sendo essa uma barreira de entrada para desenvolvedores com poucos recursos.

Diante deste cenário, surgem as games engines. Também conhecidas como motores de jogos e simuladores, é um software que fornece um conjunto de ferramentas, bibliotecas e recursos para desenvolver e criar jogos digitais. Ela oferece funcionalidades essenciais, como renderização gráfica, física, animação, áudio, inteligência artificial, gerenciamento de recursos e lógica de jogo. Uma engine permite aos desenvolvedores criar, prototipar e implementar jogos e simuladores de forma mais eficiente, fornecendo uma estrutura e conjunto de funcionalidades prontas para uso, permitindo que se concentrem na criação da experiência de jogo em si, em vez de construir tudo do zero. A Unity e a Unreal Engine são as principais engines do mercado.

2.3 Estudos desenvolvidos na área

Por conta das suas vantagens, citadas em 1.3.2, os simuladores de cirurgia robótica estão se tornando cada vez mais relevantes no meio da medicina moderna. Por consequência disso, diversos trabalhos vêm sendo desenvolvidos com o intuito de estudar e aprimorar essa ferramenta. Nesta seção serão explanados alguns estudos sobre simuladores em realidade virtual aplicados à cirurgia robótica. Importante frisar que por se tratar de um assunto relativamente recente, ainda existe uma quantidade pequena de estudos na área, ainda mais se tratando a nível nacional. Portanto, também serão abordados artigos que se tratem de tecnologias similares aos simuladores de cirurgia robótica, mas que não atuam diretamente no ramo da medicina.

2.3.1 Estudos desenvolvidos na Universidade de Brasília(UnB)

Nenhum trabalho especificamente sobre “Simuladores de realidade virtual aplicado à cirurgia robótica” foi realizado na UnB, contudo, algumas linhas de pesquisa se aproximam significativamente com o tema abordado neste trabalho.

Como em (ALMEIDA, 2019), que teve como finalidade o desenvolvimento de um jogo através de uma Engine de games chamada Unity, para testar o nível de imersão de um jogador através da utilização de realidade virtual. Essa pesquisa se assemelha em muitos pontos ao projeto dessa revisão bibliográfica, pois, por mais que o conceito de simulador e de jogo sejam distintos, em muitas características eles se assemelham, como a entrada de sinais para definir os movimentos dos personagens ou objetos virtuais. Outro ponto são as plataformas de desenvolvimentos, que na maioria dos casos são as mesmas. Por fim, a utilização da tecnologia de realidade virtual, que não possui distinção quando aplicada para jogos ou simuladores. Em resumo, o estudo do trabalho de tema “Imersão em Realidade Virtual através de um jogo RPG” do autor Luciano Henrique tem muito a contribuir com o desenvolvimento do simulador de realidade virtual aplicado à cirurgia robótica.

Outro exemplo é (BASTOS, 2021), teve como foco desenvolver um ambiente em realidade virtual para meditação e descanso. Para atingir este objetivo, foi desenvolvido um ambiente que simula de maneira razoavelmente satisfatória um "oásis", tentando reproduzir ao máximo estilos reais, através de estilos sensoriais visuais e auditivos.

Estudos que abordam ferramentas de capturas de movimentos e entrada de dados para manipulação dos personagens pertencentes aos ambientes em realidade virtual também possuem semelhanças com estudo de simuladores de realidade virtual aplicado a cirurgia robótica, como (TORRES, 2020) que desenvolve uma interface de simulação em Python bem simples, para capturar um movimentos de um Joystick ou (CARVALHO, 2014) que utilizou um Kinect Box para captura de movimentos de um robô para um ambiente em realidade virtual. Esses estudos se assemelham com o desenvolvimento de um simulador de realidade virtual aplicado a cirurgia robótica, pois nele será necessário o desenvolvimento de sensores que capturam movimentos em três dimensões.

2.3.2 Estudos desenvolvidos no Brasil

Assim como na secção anterior, existem estudos a nível nacional que possuem pontos de semelhança com o alvo de estudo deste artigo.

Um estudo que se aproxima em vário pontos é (MACHADO, 2003), este trabalho aborda o uso da realidade virtual aplicada à simulação de procedimentos invasivos em oncologia pediátrica. Para tanto, é apresentado um estudo de caso em coleta de medula óssea para transplante, para o qual um simulador foi desenvolvido. Esse é um trabalho que aborda de maneira detalhada a história e os conceitos da realidade virtual, além de aplicar estes conceitos para o uso medicinal, que é o foco desse artigo. Contudo, por se tratar de um simulador desenvolvido em 2003, existe uma distancia natural entre as tecnologias envolvidas. Por se tratar de um sistema limitado pela tecnologia da época, acaba se distanciando do resultado desejado pelo trabalho dessa revisao bibliográfica. Ainda assim, o entendimento do que foi feito nas primeiras aplicações de realidade virtual na medicina brasileira é de

grande valia para o desenvolvimento desse projeto. Outro exemplo é (LIMA, 2015) que tem como propósito discriminar o processo de desenvolvimento de um simulador educacional para o curso de medicina, criando ambientes que simulem o ambiente de trabalho real de um médico, com o intuito de ensinar na prática os desafios encontrados pelos médicos no dia a dia. O desenvolvimento do Health Simulator tem semelhanças significativas com o trabalho abordado neste artigo, semelhanças essas como: o fato de ser um simulador voltado para a área da saúde, além do seu desenvolvimento principal ter sido através de uma engine chamada Unity, que é bastante similar com a Unreal Engine, que foi utilizada para desenvolver o simulador de realidade virtual aplicado a cirurgia robótica.

Neste sentido, (PINTO, 2018) também é um exemplo de estudo que tem características muito semelhantes às apresentadas em Lima(2015), com a diferença que esse simulador não é aplicado para a área da saúde, mas sim para a produção agrícola(Figura 2.2). Outra diferença é que neste estudo eles ressaltam a utilização de um óculos de realidade virtual, do modelo Google VR integrado com a Unity. Uma semelhança considerável com o projeto do desenvolvimento do simulador de realidade virtual aplicado à cirurgia robótica é o fato do código do simulador ter sido realizado na linguagem C ++, que é a linguagem aceita pela Unreal Engine.

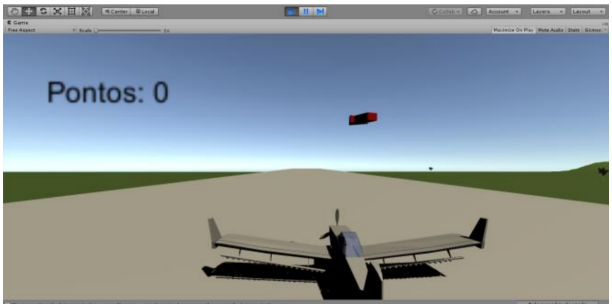

Figura 2.2 – Imagem do simulador de voo agrícola de simulação.(PINTO, 2018)

2.3.3 Estudos desenvolvidos fora do Brasil

Assim como na secção anterior, existem estudos a nível internacional que possuem pontos de semelhança com o alvo de estudo deste artigo.

Como em (LEFOR et al., 2022), cujo objetivo foi desenvolver um software de simulação de cirurgia robótica. Contudo, o artigo em questão não entra em detalhes sobre o desenvolvimento do software, além de não especificar quais os tipos de óculos de realidade virtual estão utilizando. O resultado gráfico do simulador não atingiu níveis satisfatórios. Ao analisar (LEFOR et al., 2022), conclui-se que o maior desafio foi a manipulação gráfica de tecidos moles. Outro exemplo é (MORI; IKEDA; TAKESHITA, 2022) que aborda o desenvolvimento de um simulador de realidade virtual que promete ser mais fiel com a realidade em relação às tensões causadas pelos cortes de tecidos moles. Porém, também não entra em detalhes em relação ao desenvolvimento do software.

3 Fundamentação teórica

3.1 Escolha da engine

Como discriminado na seção 2.2.3 da introdução bibliográfica (“Ambientes de desenvolvimento de realidade virtual”), os desenvolvedores de jogos e simuladores acabam optando por utilizar uma “game engine”, ao invés de criar a sua própria engine do zero.

Por mais que exitam várias games engines no mercado, a Unreal Engine se destaca pela qualidade dos gráficos que oferece. Ela possui um sistema de renderização avançado e um conjunto de ferramentas que permitem criar gráficos de alta qualidade e realistas, ferramentas estas que serão apresentadas com mais detalhes nas seções seguintes.

Além de todas as vantagens já citadas, o principal motivo para a escolha da Unreal Engine para este projeto em particular foi o fato de o autor já ter afinidade com a ferramenta. A familiaridade com a engine pode acelerar o processo de desenvolvimento, permitindo que o autor aproveite ao máximo seus conhecimentos prévios e aproveite as vantagens oferecidas pela Unreal Engine.

3.2 Introdução a Unreal Engine(UE)

A Unreal Engine é uma das principais engines de desenvolvimento de jogos e simulações disponíveis atualmente. Ela foi projetada para fornecer uma solução abrangente para o desenvolvimento de jogos e simulações interativas. Seu objetivo principal é capacitar os desenvolvedores a criar experiências visuais e interativas de alta qualidade em uma ampla variedade de plataformas. Ela oferece um conjunto de ferramentas poderosas e recursos avançados, como renderização de alta fidelidade, física realista, simulação de partículas e suporte a realidade virtual (VR) e realidade aumentada (AR).

Desenvolvida pela Epic Games, a Unreal Engine tem uma história rica e uma ampla base de usuários em todo o mundo. Criada por Tim Sweeney em 1998, com o objetivo de oferecer uma engine de jogos avançada e flexível, a primeira versão da Unreal Engine, chamada de Unreal Engine 1, foi utilizada no jogo "Unreal"lançado em 1998.

3.3 Características e capacidades principais da UE5

A versão escolhida para o desenvolvimento do simulador de cirurgia robótica foi a Unreal Engine 5.0, que é a versão mais recente desta engine. A escolha dessa versão foi baseada em sua capacidade de desenvolver jogos e simuladores extremamente realistas e

eficientes, mesmo com uma equipe de desenvolvedores reduzida. Em alguns casos, um único desenvolvedor pode criar um simulador eficiente e realista em um tempo significativamente menor em comparação com versões anteriores.

Além disso, a Unreal Engine 5.0 possui plugins que facilitam a integração do simulador com equipamentos de realidade virtual, como óculos 3D e joysticks. Isso permite uma integração mais fácil e suave, proporcionando uma experiência imersiva para os usuários do simulador.

3.4 Conhecendo a ferramenta

Nesta seção, será explanado um pouco mais sobre a Unreal Engine, apresentando suas principais ferramentas e a disposição do seu ambiente de desenvolvimento. Vale ressaltar que o idioma padrão utilizado neste projeto é o inglês (EN-US), por isso, a apresentação da ferramenta será em sua maioria utilizando termos em inglês. Todas as informações apresentadas aqui foram retiradas da documentação oficial da Unreal Engine, disponível no site oficial da Epic Games.

3.4.1 Criação e definição de parâmetros iniciais do projeto

Ao abrir o programa, o desenvolvedor se depara com a tela mostrada na figura 3.3, onde ele deve definir os parâmetros iniciais para o seu projeto. A Unreal Engine possui modos pré-definidos para diferentes tipos de projetos, adaptados às necessidades específicas de cada um. Os modos disponíveis são "Games"(jogos), "Films/Video & Live Event"(filmes/vídeos e eventos ao vivo), "Architecture"(arquitetura) e "Automotive product design & manufacture"(projeto e fabricação de produtos automotivos). Dentro de cada modo, é possível encontrar subcategorias com parâmetros mais específicos para o projeto.

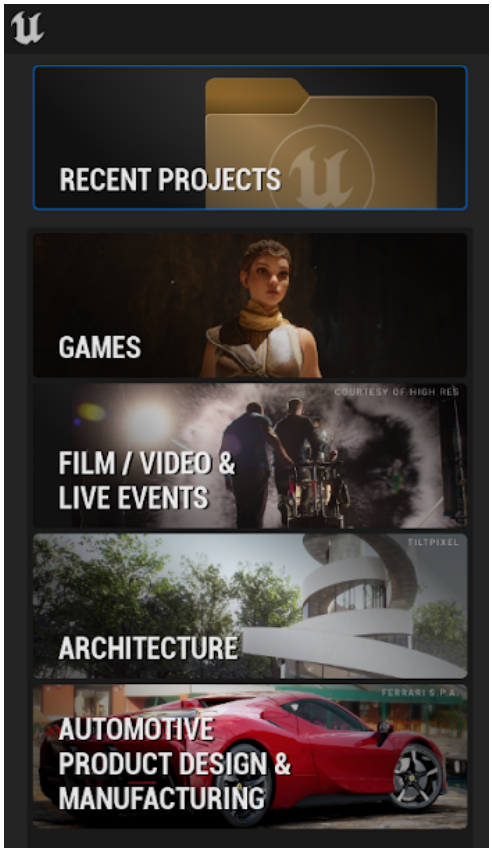

Figura 3.3 – Definição do modo de projeto na UE5

3.4.2 Interface do editor da Unreal Engine 5

Após definir os parâmetros iniciais do projeto (seção 3.4.1) e clicar em "create", o "Level Editor"é aberto, possibilitando a visualização da janela da Figura 3.4.

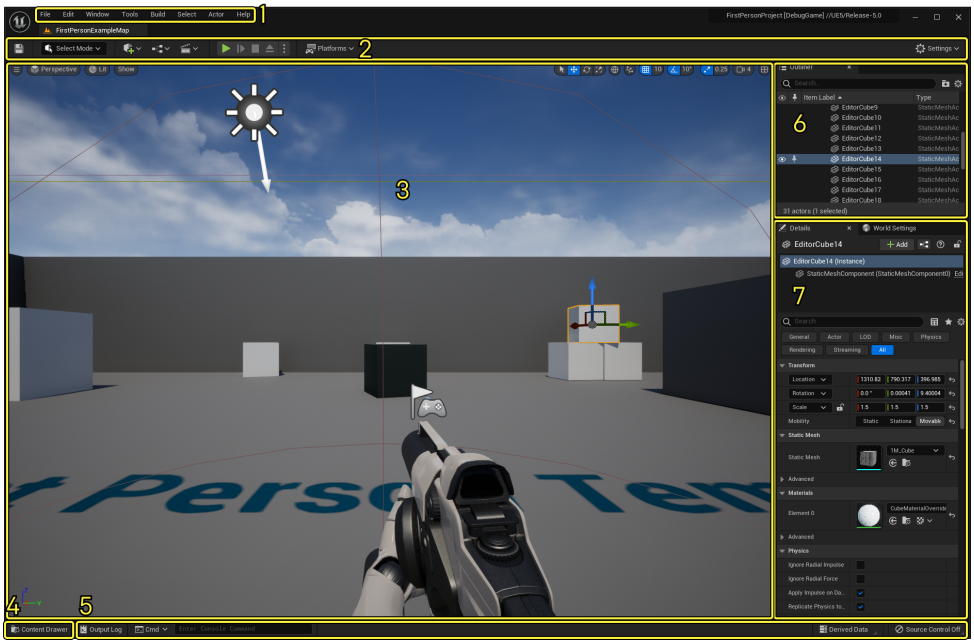

Figura 3.4 – Interface padrão do Unreal Editor no Unreal Engine 5

É possível observar que, ao escolher o modo pretendido, a engine já prepara o projeto com pré-definições baseadas nos parâmetros escolhidos. No exemplo da Figura 3.4 foi usado o modo "Game//First Person"por Isso inclui a câmera em primeira pessoa, a arma segurada pelo personagem e recursos de deslocamento do personagem e rotação da câmera comumente utilizados em jogos em primeira pessoa.

3.5 Programação na Unreal Engine

A Unreal Engine fornece duas maneiras do desenvolvedor criar suas aplicações, através do uso de Blueprints e a linguagem C++. É importante ressaltar que a programação na Unreal, principalmente em C++, é bastante extensa e possui milhares de opções diferentes. Explicar detalhadamente o seu funcionamento neste trabalho seria inviável.

A programação via Blueprint é um sistema baseado em nós que representa blocos de lógica de programação. Esses blocos podem ser conectados uns aos outros para criar fluxos de trabalho mais complexos. Com o exemplo mostrado na figura abaixo.

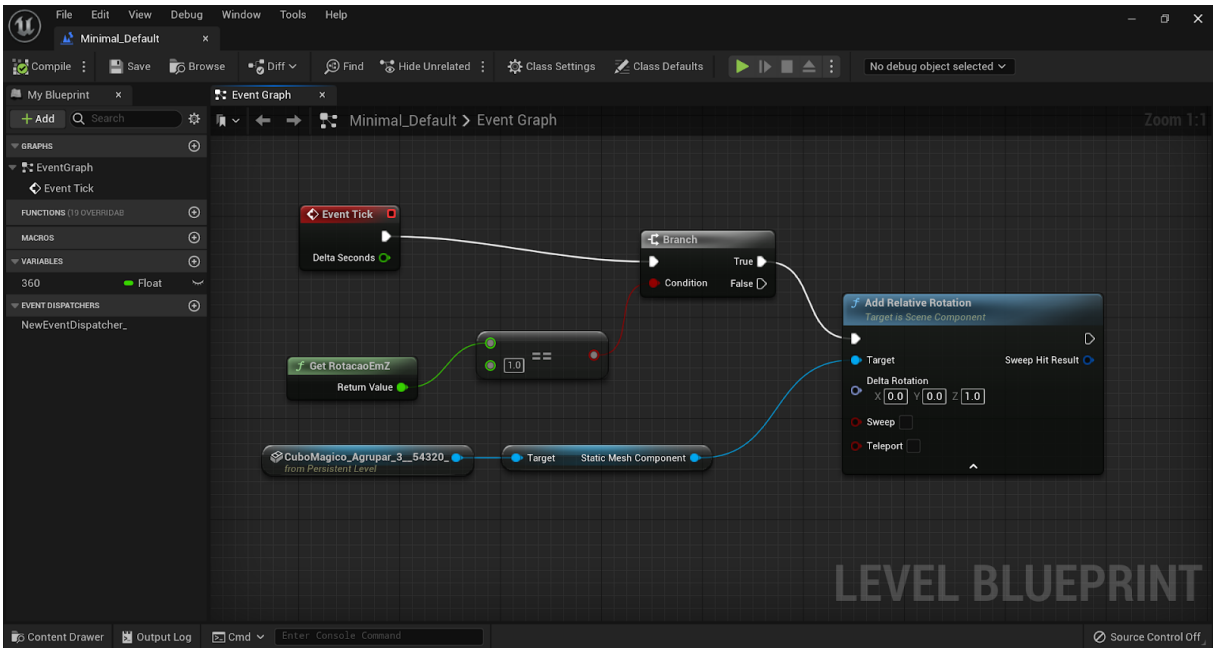

Figura 3.5 – Programação por Blueprint do exemplo

Na figura 3.5, é possível observar que cada ação dentro do projeto é iniciada a partir de um "Event Dispatcher". Esse evento pode ser o próprio início do jogo, a ação de pressionar uma tecla do teclado ou um evento personalizado criado pelo desenvolvedor. A partir desse evento inicial, os eventos seguintes são conectados por meio dessas "linhas"que podem ser vistas na figura 3.5. No exemplo da figura, temos o "Event Tick"como evento de partida.

3.6 Geração de imagem estereoscópica na Unreal Engine

Na renderização estereoscópica, a Unreal permite gerar as imagens para os olhos esquerdo e direito e codificar as imagens de saída usando os tipos formatos padrões para imagens estereoscópicas. Nesse caso, o desenvolvedor é responsável por configurar a placa gráfica, driver de exibição ou hardware para interpretar as imagens estéreo produzidas e encaminhá-las adequadamente. A Unreal oferece três formatos padrões para geração de imagem estereoscópica, sendo eles o Frame sequential, Side by side e Top-bottom.

- **Frame sequential** Este modo produz pares estéreo sequenciais para cada quadro do loop principal do motor. Por exemplo, primeiro renderiza o olho esquerdo para o quadro 1, depois o olho direito para o quadro 1, depois o olho esquerdo para o quadro 2, depois o olho direito para o quadro 2, depois o olho esquerdo para o quadro 3 e assim por diante. Na maioria dos casos, esta opção requer uma GPU com suporte especializado para renderização estereoscópica.
- **Side by side** Neste modo, a imagem produzida para cada quadro do loop principal do motor é dividida em duas. A metade esquerda da imagem contém a visão da posição do olho esquerdo e a metade direita da imagem contém a visão da posição do olho direito. Este modo tem duas vantagens. Primeiro, pode produzir taxas de quadros mais altas porque o tempo de renderização é menor para cada imagem. Em segundo lugar, você pode usá-lo com qualquer GPU. Por outro lado, a desvantagem é que as imagens são de qualidade inferior.
- **Top-bottom** Este modo é quase igual à opção **Side by side** descrita acima. A única diferença é que a imagem de cada quadro é dividida ao meio horizontalmente em vez de verticalmente. A metade superior da imagem mostra a visão do olho esquerdo e a metade inferior da imagem mostra a visão do olho direito.

3.7 Modelagem 3D

A modelagem 3D desempenha um papel fundamental na criação de um projeto de realidade virtual de alta qualidade. É por meio da modelagem 3D que os objetos, ambientes e personagens virtuais são criados, proporcionando uma experiência imersiva e realista para os usuários. Embora a Unreal Engine ofereça algumas ferramentas básicas de modelagem 3D, é importante destacar que não é um software focado exclusivamente nessa área. Embora seja possível criar modelos básicos diretamente dentro da UE, a complexidade e detalhamento exigidos para criar objetos, personagens e ambientes realistas geralmente requerem o uso de softwares especializados em modelagem 3D, que nesse trabalho será o 3ds Max, versão 2023.

3.7.1 Escolha do software de modelagem 3D

Existem muitos programas de modelagem 3D, como o Blender, Maya, ZBrush, Cinema 4D, 3ds Max, entre outros. Cada um deles possui suas próprias vantagens e recursos específicos. A escolha do programa de modelagem depende das necessidades do projeto e da preferência do artista. No entanto, o 3ds Max se destaca devido à sua funcionalidade abrangente, integração com outros softwares, versão gratuita para estudantes, além de ser desenvolvido pela Autodesk, uma empresa especializada no desenvolvimento de softwares 3D, como o AutoCAD e o Fusion 360.

O 3ds Max é um software de modelagem 3D amplamente utilizado na indústria de jogos, filmes, arquitetura e realidade virtual. Ele oferece uma variedade de recursos e ferramentas que o tornam uma escolha popular para projetos de realidade virtual pois oferece recursos abrangentes que permitem a criação de modelos complexos e detalhados. Além disso, suporta várias técnicas de modelagem, como modelagem poligonal, modelagem baseada em subdivisão, modelagem paramétrica e modelagem de spline. Isso proporciona flexibilidade ao projetista para criar diversos tipos de objetos e ambientes virtuais.

Além das características já abordadas, o principal motivo para a escolha do 3ds Max para este projeto em particular foi o fato de o autor já ter afinidade com a ferramenta. A familiaridade com o software de modelagem pode acelerar o processo de desenvolvimento, permitindo que o autor aproveite ao máximo seus conhecimentos prévios e aproveite as vantagens oferecidas pela ferramenta.

3.8 Cirurgia robótica

A implementação da cirurgia robótica tem sido um avanço significativo na área médica, trazendo benefícios e impactos positivos para pacientes, cirurgiões e sistemas de saúde. Essa tecnologia inovadora envolve o uso de robôs cirúrgicos controlados por cirurgiões para realizar procedimentos com precisão e habilidades aprimoradas.

Uma das principais vantagens da cirurgia robótica é a sua precisão e estabilidade. Os robôs cirúrgicos são capazes de realizar movimentos delicados e precisos que superam as habilidades humanas, permitindo procedimentos mais seguros e eficazes. Além disso, os sistemas robóticos fornecem uma visão tridimensional ampliada e de alta definição, melhorando a precisão do diagnóstico e aumentando a capacidade de identificar e tratar condições médicas com mais precisão.

A cirurgia robótica também oferece benefícios significativos aos pacientes. As incisões são menores e mais precisas, resultando em menos trauma cirúrgico, menor perda de sangue, menor risco de infecção e recuperação mais rápida. Os pacientes geralmente experimentam menos dor pós-operatória e têm menor tempo de internação hospitalar, permitindo que

retornem às suas atividades normais mais rapidamente.

Para os cirurgiões, a cirurgia robótica proporciona uma melhoria na destreza e no controle durante os procedimentos. Os robôs cirúrgicos têm instrumentos articulados que podem se mover com mais liberdade e alcance do que as mãos humanas, permitindo manobras complexas em áreas de difícil acesso. Além disso, os sistemas robóticos geralmente têm uma curva de aprendizado mais curta em comparação com técnicas cirúrgicas convencionais, o que significa que os cirurgiões podem adquirir habilidades robóticas mais rapidamente.

A implementação da cirurgia robótica abre novas possibilidades para a medicina moderna, melhorando a qualidade dos procedimentos cirúrgicos e proporcionando uma recuperação mais rápida e confortável para os pacientes. A contínua evolução dessa tecnologia promete avanços ainda mais impressionantes na área médica, beneficiando tanto os profissionais de saúde quanto aqueles que recebem cuidados médicos.

3.9 Robô Da Vinci

Atualmente, o sistema da Vinci, desenvolvido pela Intuitive Surgical, localizada na Califórnia, EUA, é o robô cirúrgico mais proeminente globalmente. Esse sistema é composto por três principais componentes: um console do cirurgião, um carrinho de instrumentos e uma plataforma de visão.

O console do cirurgião oferece ao profissional uma visão cirúrgica imersiva por meio de imagens 3D de alta definição. Os movimentos delicados dos dedos e punho do cirurgião são captados por sensores e convertidos em movimentos precisos e suaves, evitando vibrações indesejadas. Essa tecnologia avançada elimina o efeito de fulcro e as forças de cisalhamento comuns em instrumentos endoscópicos de eixo longo.

Um destaque fundamental do sistema é a presença de uma articulação de punho no microinstrumento, permitindo uma ação rotacional no campo cirúrgico. Essa característica aprimorada resulta em maior destreza em espaços reduzidos e possibilita uma habilidade excepcional na realização de suturas precisas.

Em suma, o sistema da Vinci tem revolucionado a cirurgia, proporcionando aos cirurgiões ferramentas avançadas para aprimorar suas habilidades e melhorar os resultados dos procedimentos. A tecnologia sofisticada tem se mostrado altamente benéfica para pacientes, uma vez que reduz o trauma cirúrgico, melhora a recuperação e impulsiona o avanço da medicina moderna.

Estes são os principais componentes do robô Da Vinci:

- **Console do Cirurgião** - O console do cirurgião é a estação onde o cirurgião se posiciona para realizar a cirurgia, como pode ser visto na Figura 3.6. Ele é equipado com

controles, alavancas e pedais que permitem ao cirurgião operar os braços robóticos do sistema. O cirurgião visualiza o campo cirúrgico em uma tela 3D de alta definição, fornecendo uma visão detalhada da área de trabalho..

- **Braços Robóticos** - O Sistema da Vinci possui vários braços robóticos que são controlados pelo cirurgião a partir do console. Esses braços são equipados com instrumentos cirúrgicos especializados, como pinças e tesouras, que podem ser movidos com precisão milimétrica dentro do paciente..
- **Câmera** - O robô da Vinci é equipado com uma câmera 3D de alta definição, montada em um dos braços robóticos. Essa câmera proporciona uma visão estereoscópica ao cirurgião, permitindo uma percepção tridimensional do campo cirúrgico..
- **Endowrist Articulado** - Os instrumentos cirúrgicos montados nos braços robóticos possuem o Endowrist Articulado, um sistema que reproduz os movimentos naturais da mão humana. Essa capacidade de articulação é crucial para permitir que o cirurgião realize movimentos precisos e delicados durante a cirurgia.

Figura 3.6 – Console do cirurgião.(MORRELL et al., 2021)

Em conjunto, esses componentes trabalham de forma coordenada para fornecer ao cirurgião uma precisão e controle excepcionais durante procedimentos cirúrgicos minimamente invasivos, reduzindo o trauma aos pacientes, o tempo de recuperação e o risco de complicações.

Por mais que não faça parte do robô, um elemento fundamental no procedimento de cirurgia robótica é o trocarte. O trocarte é um instrumento cirurgico que cria uma via de acesso para os instrumentos cirúrgicos e a câmera no interior do abdômen ou outra cavidade do corpo durante a cirurgia laparoscópica. Ele é inserido através de pequenas incisões na parede abdominal, por onde os instrumentos cirúrgicos são introduzidos, como pode ser visto na Figura 3.7.

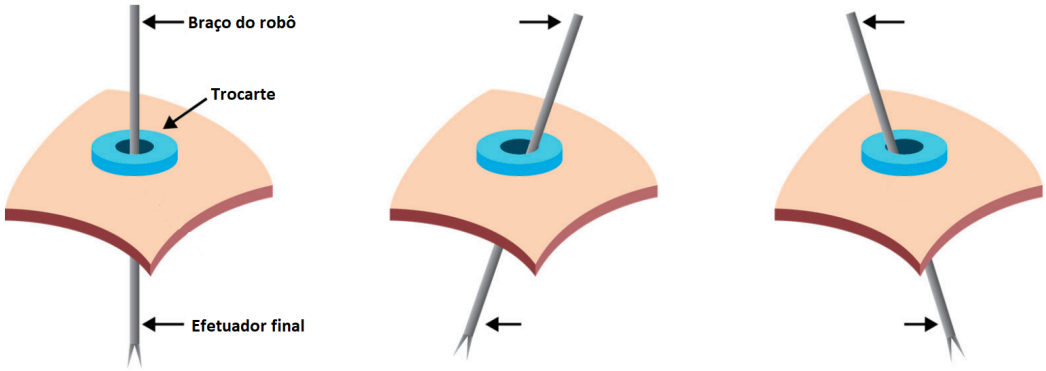

Figura 3.7 – Trocarte. (LONGMORE; NAIK; GARGIULO, 2020)

O trocarte é usado como ponto de apoio dos instrumentos do robô de cirurgia robótica que precisam ser introduzidos no corpo do paciente, como o braço robótico e a câmera. Os instrumentos giram em torno do ponto de apoio de modo que os movimentos do efetuador final sejam invertidos para os movimentos do braço robótico. Os movimentos do braço robótico externo são traduzido através do ponto de apoio do trocarte automaticamente pelo sistema do robô para corresponder ao movimento do cirurgião. Sabendo disto, é fácil entender que o braço robótico realiza apenas movimentos de rotação, avanço e recuo. O que limita esses movimentos, além do tipo de cirurgia, são as dimensões do trocarte e do próximo instrumento que esta sendo introduzido no corpo do paciente.

Importante frizar que a maior parte das informações desta seção foram retiradas do artigo (LONGMORE; NAIK; GARGIULO, 2020).

4 Desenvolvimento

4.1 Definição do projeto

Esse projeto consiste no desenvolvimento de um protótipo de um simulador de realidade virtual para cirurgia robótica. Um simulador de cirurgia robótica possui exercícios com o objetivo de treinar as habilidades motoras do usuário para o manuseio de um robô cirúrgico real. Sabendo disto, o protótipo possui um exercício teste no qual o usuário tem a tarefa de pegar um objeto cúbico e colocá-lo no pote devido (aquele que possui a mesma cor do objeto). Este exercício tem como objetivo treinar o usuário para que ele possa se adaptar ao controle das garras (EndoWrist) de um robô de cirurgia real.

Por se tratar de um protótipo ainda, o objetivo principal desse projeto é o aprimoramento das técnicas utilizadas para o desenvolvimento desse simulador, colocando à prova a utilização do software escolhido e os princípios empregados nesse desenvolvimento.

Para alcançar os objetivos citados acima, o protótipo deve atender os seguintes requisitos:

- Simular o movimento de duas garras(EndoWrist) do robô através da entrada de dados dos movimentos das mãos do usuário.
- Ser capaz de gerar imagens estereoscópicas onde o usuário tenha uma experiencia com noção de profundidade.
- Simular o movimento da câmera(endoscopes).
- Criar um sistema mínimo de colisões, onde a garra seja capaz de segurar e soltar o objeto cúbico em resposta às entradas realizadas pelo usuário.
- Limitar os movimentos das garras de acordo com variáveis que atendam aos requisitos previamente definidos, como a restrição do espaço do trocarte (será melhor explicado nas seções a seguir).

4.2 Escolha do exercício

No exercício proposto, inicialmente, o usuário irá se deparar com a visão de duas garras, três potes de cores vermelho, verde e azul e um objeto cúbico também de cor azul, como na Figura 4.8. Importante frisar que a cena da Figura 4.8 já se trata de um frame capturado do simulador desenvolvido.

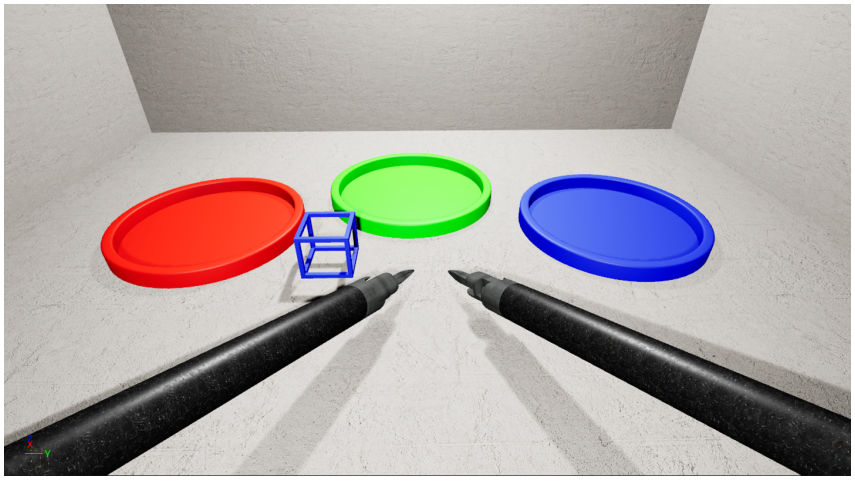

Figura 4.8 – Visão do exercício do simulador desenvolvido

Como citado na seção anterior, a tarefa consiste em o usuário tentar pegar o objeto cúbico azul e colocá-lo no pote da mesma cor do objeto. A escolha desse desafio foi inspirada no software de simulação de cirurgia robótica já validado no mercado, o FlexVR.

Por se tratar de um simulador já validado e em circulação no mercado, tê-lo como base para o desenvolvimento do protótipo deste artigo mostra-se um bom ponto de partida para a validação deste projeto. Isso se junta ao fato de ser um simulador ao qual se possui possui livre acesso, acesso esse que se mostrou essencial no desenvolvimento desse projeto.

O tarefa escolhido trata-se do exercício mais básico do simulador FlexVR, abordando a utilização dos movimentos básicos da garra, o que se mostrou o melhor desafio para o desenvolvimento do protótipo. Na figura abaixo, pode-se observar um frame do exercício do FlexVR, no qual a tarefa deste projeto (Figura 4.8) foi baseado.

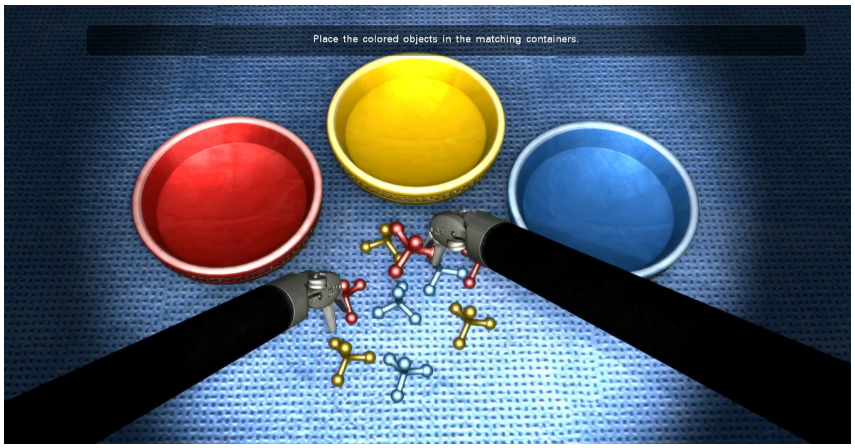

Figura 4.9 – Visão do exercício do simulador FlexVR

4.3 Obtenção das medidas das pinças

Para que o simulador alcance a máxima precisão e realismo na simulação dos movimentos cirúrgicos, foi necessário realizar as medições das dimensões de uma garra (EndoWrist) real do robô da Vinci.

Figura 4.10 – EndoWrist real 1

Figura 4.11 – EndoWrist real 2

Para melhorar o entendimento, o corpo da garra foi dividido em 8 partes. Essas partes foram nomeadas com letras de A a G e podem ser visualizadas na figura 4.12.

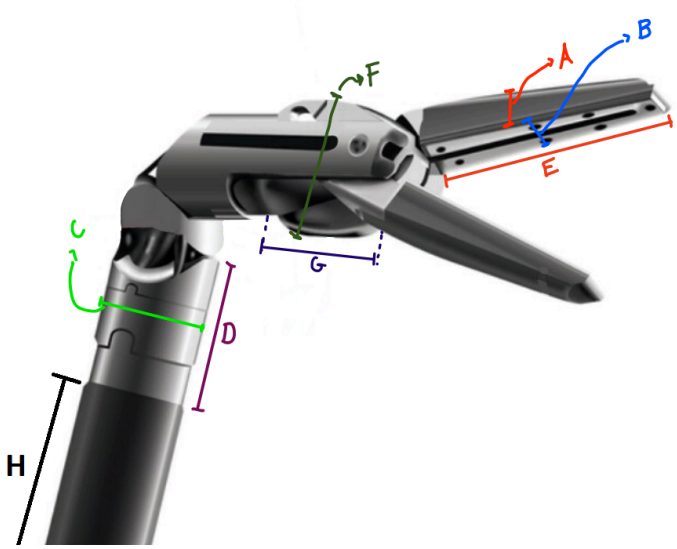

Figura 4.12 – Partes da garra do robô. (ESPECIFICADOS, Ano não especificado)

Parte	Medida(mm)
A	1.38
B	1.97
C	8.48
D	12
E	6.21
F	6.01
G	8.00
H	424

Tabela 4.1 – Medidas da garra obtida em laboratório

Os dados de medidos são fundamentais para a construção dos blocos 3D para a simulação, que será abordado na próxima seção, pois foi criado um modelo 3D que tem como base as medidas da Tabela 3.1, porém trata-se de um modelo genérico com o propósito de atender às especificações do projeto.

4.4 Modelagem 3D

Todos os objetos que compõem o cenário do protótipo foram modelados no software 3ds Max, que já foi apresentado na introdução teórica. Esses objetos incluem as EndoWrists (Figuras 4.13 e 4.14), o cenário (paredes, teto e piso), os três potes coloridos (Figura 4.16) e o objeto cúbico (Figura 4.15).

É importante frisar também que, por se tratar de um protótipo, foram realizadas simplificações na modelagem 3D das garras. O foco foi obedecer às dimensões e aos eixos de rotação da garra, ficando em segundo plano a similaridade visual para esta etapa do desenvolvimento do projeto.

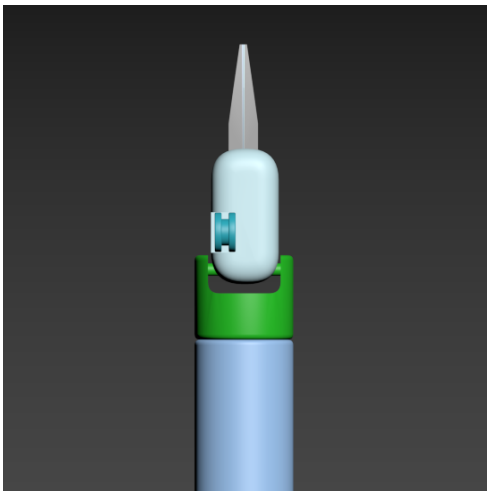

Figura 4.13 – 3D EndoWrist no 3dsMAX

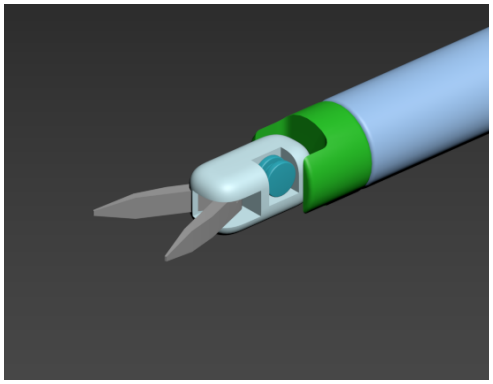

Figura 4.14 – 3D EndoWrist no 3dsMAX 2

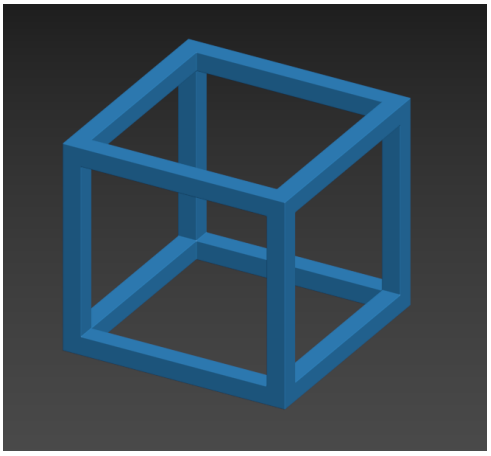

Figura 4.15 – 3D cubo no 3dsMAX

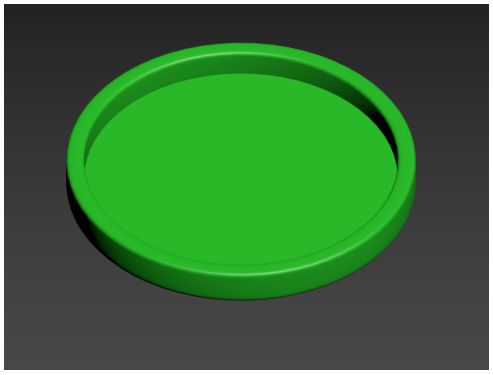

Figura 4.16 – 3D pote no 3dsMAX

Para facilitar o entendimento, foram atribuídos nomes a cada parte das garras. Esses nomes foram escolhidos de forma subjetiva pelo desenvolvedor do projeto, de maneira que lhe pareceu mais conveniente para o entendimento do código. Esses nomes podem ser observados na figura abaixo.

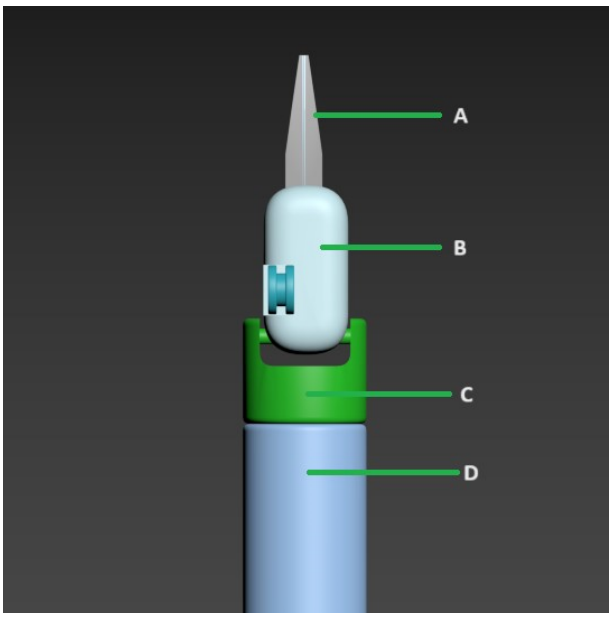

Figura 4.17 – Nomes das partes da garra utilizadas no protótipo

- **A** - Pinças.
- **B** - Apoio Pinças.
- **C** - Ponta da Garra.
- **D** - Corpo da Garra.

4.5 Entrada de dados

O propósito do robô de cirurgia robótica é ser uma extensão do cirurgião, ou seja, ele não busca substituir o profissional de medicina por uma máquina, mas sim fornecer a ele mais ferramentas para a execução da cirurgia de forma mais precisa e menos invasiva. Para alcançar tal objetivo, é necessário que o robô seja capaz de reproduzir com fidelidade os movimentos realizados pelas mãos do cirurgião. Para isso, é utilizada uma interface háptica que fica no console do cirurgião, a qual transforma os movimentos da mão do cirurgião em movimentos do robô, como visto na Figura 3.6.

Em posse desse conhecimento, um simulador em realidade virtual de um robô de cirurgia robótica, além de gerar uma imagem estereoscópica, também deve ser capaz de fazer a leitura dos movimentos das mãos do usuário e converter esses dados em movimentos das garras dentro do simulador. O projeto deste tem como objetivo final criar o seu próprio touch controller, de maneira que todo o desenvolvimento desse controlador seja focado na sua utilização para o simulador, levando em consideração o seu formato, tamanho, encaixe ideal para os dedos e capacidade de ler movimentos de deslocamento e rotação com uma precisão ideal para este fim. Contudo, especificamente para o desenvolvimento deste Trabalho de Graduação 2, visto que o controller citado acima ainda não foi desenvolvido, foi escolhido como ferramenta para captura dos dados de movimento das mãos do usuário o Oculus Rift, mais especificamente os Touch Controllers do Oculus Rift.

O Oculus Rift é um dispositivo de realidade virtual (VR) desenvolvido pela Meta (anteriormente conhecida como Facebook). Ele foi projetado para proporcionar uma experiência imersiva de realidade virtual, permitindo que os usuários mergulhem em ambientes virtuais tridimensionais.(OCULUS…, 2013)

Figura 4.18 – Oculus Rift (AMAZON…, 2023)

Figura 4.19 – Sensor do Oculus Rift(AMAZON…, 2023)

O Oculus Rift é composto por três componentes principais, que serão descrimidados a seguir:

- **Headset (HMD - Head-Mounted Display)** O headset(parte superior da figura 4.18) é a peça principal do Oculus Rift e é usado para exibir imagens estereoscópicas em

telas LED incorporadas.Contém lentes para criar a ilusão de profundidade e tridimensionalidade, além de possuir sensores que capturam o movimento da cabeça do usuário.

- **Sensores de Rastreamento(torres):**O Rift original vem com sensores externos que ajudam a rastrear a posição e o movimento do headset e dos controles(Figura 4.19). As versões mais atuais dos óculus de realidade virtual da Meta, como o Oculus Rift S e o Oculus Quest utilizam sensores incorporados para rastreamento, eliminando a necessidade de sensores externos.
- **Controladores (Touch Controllers):** São dispositivos de entrada sem fio que permitem que os usuários interajam com o ambiente virtual(Parte inferior da figura 4.18). Oferecem rastreamento preciso de posição e movimento das mãos, proporcionando uma experiência mais envolvente.

A escolha específica desse óculos para esse trabalho está ligada ao fato de os equipamentos da Meta oferecerem diversos plugins gratuitos de integração desses equipamentos com a Unreal Engine, que foi o software utilizado para desenvolver o simulador de realidade virtual aplicado à cirurgia robótica. Além disso, ele oferece o que é preciso para o desenvolvimento desse protótipo, que é um dispositivo capaz de gerar a imagem estereoscópica (Head-Mounted Display) e controladores capazes de capturar movimentos realizados pelas mãos do usuário e transformá-los em entrada de dados para o simulador (Sensores de Rastreamento e Touch Controllers).

Entretanto, qualquer outro óculos da Meta seria capaz de atender às necessidades desse projeto, como o Oculus Rift S e o Oculus Quest. Por conta disso, o principal motivo para a escolha específica do Oculus Rift está ligado ao fato de ser o modelo disponível no laboratório.

A utilização do headset Oculus Rift representa uma diferença no método de geração de imagem estereoscópica em comparação com o simulador FlexVR, que utiliza apenas óculos 3D para essa finalidade. A escolha do headset Oculus Rift baseia-se principalmente em dois fatores. O primeiro deles é a facilidade de integração do Oculus Rift com a Unreal Engine, já que possui um modo padrão de geração de imagem estereoscópica. O segundo fator é a integração com os Touch Controllers. Como esses controladores já estavam sendo utilizados no projeto, a escolha do headset facilitou a integração entre os componentes.

No entanto, a opção por um headset de realidade virtual, em vez de óculos 3D simples, foi viável apenas por se tratar de um único protótipo e já se possuir o equipamento, uma vez que o custo de aquisição dos óculos 3D é significativamente menor do que o do Oculus de realidade virtual. Isso tornaria o projeto inviável se fosse necessário desenvolver mais protótipos. Nas seções de análise de resultados e conclusão, é mencionada a possibilidade de utilizar apenas óculos 3D na continuação do projeto.

Os dados gerados pelo Oculus Rift interagem com o programa desenvolvido na Unreal Engine através de duas funcionalidades oferecidas pela própria engine, o Enhanced Input e o Motion Controller. Para entender como se deu o desenvolvimento desse simulador, é importante, primeiro, compreender resumidamente como funcionam esses dois mecanismos, pois eles serão citados várias vezes durante a explicação da criação do projeto.

4.5.1 Enhanced Input

Antes de explicar sobre o Enhanced Input, é importante frisar que nesta seção será abordado um breve resumo dessa funcionalidade, pois ela possui "infinitas"aplicações e variações na sua utilização que pouco acrescentam para o desenvolvimento desse projeto. Contudo, informações detalhadas desse mecanismo podem ser encontradas no site oficial da Unreal Engine((EPIC GAMES, 2023a)).

Resumidamente, o Enhanced Input é uma ferramenta aprimorado de entrada de dados que permite ao desenvolvedor criar "eventos"que iniciam uma cadeia de funções desejadas no programa. O Enhanced Input permite que esses eventos sejam acionados por entradas de equipamentos externos, como um mouse, controles de jogos ou um Touch Controller de um óculos de realidade virtual. Por exemplo, pode-se criar uma função "DeslocaAtor", que, quando chamada, movimenta um ator na cena para frente, e configurar um Enhanced Input que seja acionado toda vez que o usuário pressionar a tecla "w"do teclado. Ou seja, quando o usuário pressionar a tecla "w", o Enhanced Input configurado iniciará uma cadeia de eventos que, nesse exemplo, irá chamar a função "DeslocaAtor".

Na seção "Fluxo do programa", será possível entender melhor como essa ferramenta funciona na prática.

4.5.2 Motion Controller

Resumidamente, o Motion Controller na Unreal Engine refere-se a um sistema que permite o rastreamento e a interação com dispositivos de entrada específicos, como controladores de realidade virtual (VR) ou outros dispositivos de entrada de movimento. Ele possui suporte para os equipamentos da Meta, sendo possível traduzir os movimentos realizados pelos Touch Controllers do Oculus Rift em informações de deslocamento, velocidade linear, velocidade angular e aceleração que podem ser utilizados para movimentar um ator ou objeto no ambiente virtual.

Assim como na seção do Enhanced Input, foi apresentado apenas um resumo dessa funcionalidade, pois ela possui "infinitas"aplicações e variações na sua utilização que pouco acrescentam para o desenvolvimento desse projeto. Contudo, também é possível obter informações detalhadas desse mecanismo no site oficial da Unreal Engine ((EPIC GAMES, 2023b)).

O entendimento dessa ferramenta, ainda que de maneira resumida, é essencial para compreender as explicações na seção "Fluxo do programa".

4.6 Fluxo do programa

Na Figura 4.20, pode ser visto um diagrama de blocos que representa o organograma de ações realizadas pelo simulador, onde o bloco verde representa o início do programa, os blocos em vermelho os eventos de disparo e os blocos em azul as funções chamadas a partir de cada evento. O entendimento desse fluxo ficará mais claro no decorrer desta seção. Importante ressaltar que a Figura 4.20 representa o fluxo de funções aplicadas para apenas uma garra; contudo, o que ocorre com a outra garra é apenas um espelho desse fluxo, seguindo as mesmas funções.

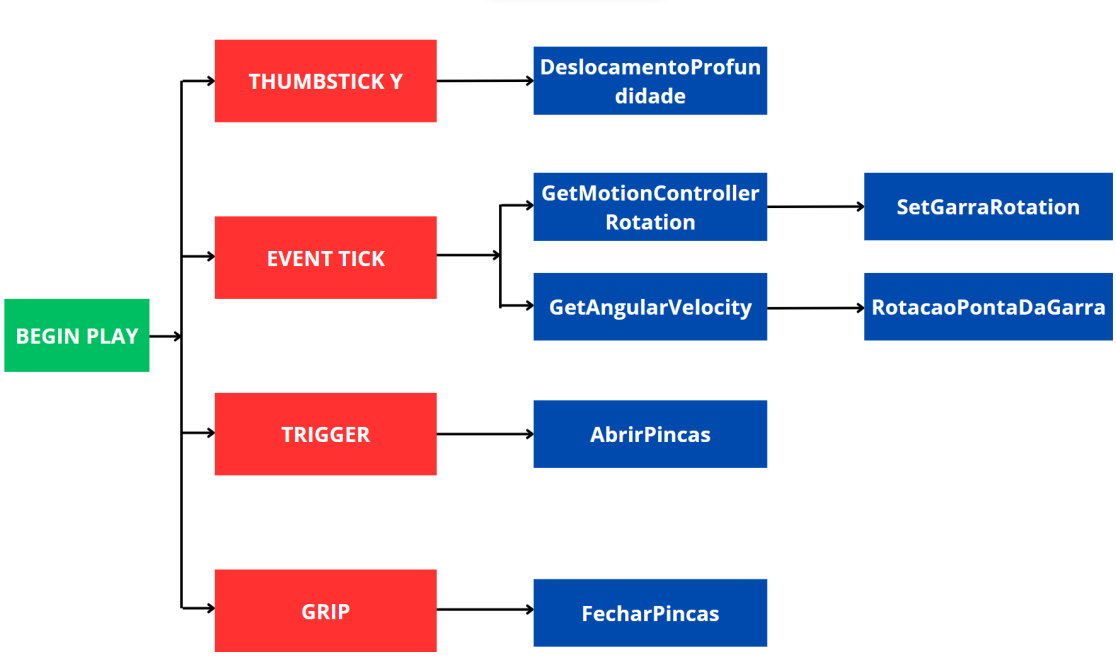

Figura 4.20 – Organograma de eventos do simulador desenvolvido

Todas as ações realizadas na Unreal Engine precisam ser disparadas por um evento(representados pelos blocos em vermelho na Figura 4.20), como por exemplo o "Event Begin", que dispara uma ação assim que o jogo começar, ou o "Press Keyboard K", que dispara uma ação assim que o jogador pressionar a tecla "K". Nesse simulador, foram utilizados vários tipos de eventos diferentes, sendo o mais utilizado deles o "Event Tick", que é o responsável por iniciar o fluxo da movimentação principal das garras do simulador.

Um evento, quando iniciado, emite uma informação para o sistema indicando que todas as ações ligadas a ele devem ser executadas. O "Event Tick"funciona como um relógio, que a cada período de tempo definido emite essa informação de inicialização ao sistema. Esse tempo é definido de acordo com as configurações de FPS (frames por segundo

do sistema). Se o software está configurado para 60 fps, isso significa que 60 informações de inicialização serão enviadas ao sistema em 1 segundo.

A Unreal Engine funciona como um sistema em cadeia, onde o fim de cada evento inicia o evento seguinte. Como pode ser visto na Figura 4.20, um dos fluxos desse projeto tem como evento de partida o "Event Tick"e a partir dele as demais funções são chamadas dentro do programa. Para tornar o entendimento mais claro, antes de explicar o organograma de funções completo do simulador em si, será detalhado abaixo a função de cada bloco de uma das ramificações do fluxo que é iniciado pelo "Event Tick".

- **GetMotionControllerRotation** - Função responsável por pegar as coordenadas de rotação do Motion Controller.
- **SetGarraRotation** - Função que define as novas coordenadas de rotação da Garra baseada nas informações colhidas pelo "GetMotionControllerRotation".

Portanto, a cada 0,016 segundos, o "Event Tick"é iniciado, chamando a função GetMotionControllerRotation que salva em uma variável do tipo float as coordenadas de rotação do Motion Controller. Em seguida, a função SetGarraRotation é chamada, recebendo como parâmetro a variável, definindo as novas coordenadas de rotação da Garra.

A explicação de uma das ramificações do fluxo de funções iniciadas pelo "Event Tick"foi apenas um exemplo para demonstrar o funcionamento do simulador. Outros eventos de disparo observados na figura 4.20 foram utilizados no desenvolvimento do protótipo e serão explanados no decorrer da próxima seção.

4.7 Movimentos da garra

Como visto na seção 3.9, em um robô de cirurgia robótica real, a garra realiza movimentos de rotação em torno de um eixo que fica apoiado em um instrumento de cirurgia chamado trocarte. O trocarte, além de servir como apoio para o robô, funciona como um "acesso"para as garras do equipamento ao corpo do paciente.

Por questões de segurança do paciente, as garras do robô não podem realizar movimentos que excedam esse limite estabelecido pelo trocarte. Como pode ser visto na Figura 3.7, caso a garra realize um movimento de forma a ultrapassar o espaço do trocarte, isso pode resultar em ferimentos graves no paciente.

Buscando simular o movimento real do robô, as garras do simulador realizam apenas dois tipos de movimento no que diz respeito ao seu deslocamento no espaço: um movimento aqui chamado "movimento de rotação"e o deslocamento de avanço no sentido do seu eixo. Estes movimentos serão melhores explicados nas seções a seguir.

4.7.1 Movimentos de rotação da garra

Para entender o movimento de rotação, primeiro, é preciso entender a posição da garra no espaço tridimensional, usando a Figura 4.21 como exemplo, o vetor "r"representa o vetor da garra do simulador e a origem do sistema representa o ponto onde está localizado o trocarte(Figura 3.7). O movimento de rotação representa a variação dos angulos do vetor "r"com os eixos x,y e z.

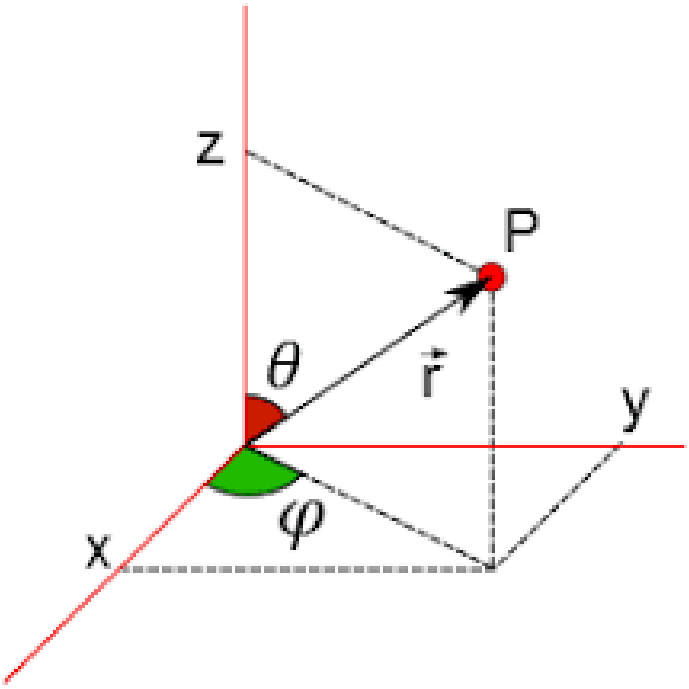

Figura 4.21 – Exemplo de espaço tridimensional. (IMAGEM..., Ano não especificado)

Os movimentos de rotação são a principal maneira de levar a garra do simulador aos pontos no espaço desejado sem infringir a limitação do trocarte. Esses movimentos Para realizar esses movimentos de rotação, o simulador utiliza os dados de rotação das mãos do usuário que foram lidas pelo Motion Controller (seção 4.5.2). Com os Touch Controllers em mãos, à medida que o usuário do simulador realiza movimentos de rotação com as mãos, os sensores do Oculus Rift capturam esses dados e salvam no Motion Controller.

Como visto na explicação do fluxo de eventos disparado pelo "Event Tick", o programa obtém esses dados de rotação e informa à garra quais serão os novos valores de seus ângulos. Na figura abaixo pode ser observado um pseudocódigo que representa de maneira simplificada o princípio de funcionamento do código para o movimento de rotação da garra.

```
1
2   /*A cada 0,016 segundos o event tick é acionado, aqui representado pelo valor 1*/
3   if (EventTick ==  1):
4
5       /*O Motion Controller é acionado  retornando os valores dos angulos das mãos do usuário*/
6       AngulosDeRotacao = MotionController(Angulos)
7       RotacaoX =AngulosDeRotacao[0]
8       RotacaoY =AngulosDeRotacao[1]
9       RotacaoZ =AngulosDeRotacao[2]
10
11      /*Afunção MovimentoDeRotacao é chamada passando
12      os parâmetros obtidos através do Motion Controller*/
13      MovimentoDeRotacao(RotacaoX,RotacaoY,RotacaoZ)
14
15  /*Definição da função MovimentoDeRotação*/
16  void MovimentoDeRotacao(RotacaoX,RotacaoY,RotacaoZ){
17
18      /*Defini os angulos recebidos como parâmtros como
19      os novos valores dos angulos da garra*/
20      AnguloGarraX = RotacaoX
21      AnguloGarraY = RotacaoY
22      AnguloGarraZ = RotacaoZ
23
24      return 0
25  }
```

Figura 4.22 – Pseudocódigo do movimento de rotação da garra

4.7.2 Movimentos de avanço

Utilizando a figura 4.21 como exemplo, colocando a garra na posição do vetor "r", o movimento de avanço representaria o deslocamento no sentido que o este vetor está apontando. Na teoria, a maneira mais simples e intuitiva de resolver a questão do avanço da garra seria a utilização das informações de deslocamento do motion controller, já que da mesma maneira que ele informa seus dados de rotação, também seria possível usar os dados de deslocamento no espaço. Contudo, notou-se uma dificuldade para implementar esse movimento utilizando esses dados do Motion Controller.

A primeira explicação para essa dificuldade é o fato de os touch controllers estarem sendo utilizados como uma "improvisação"para os sensores de movimento, já que o ideal seria a utilização de uma interface desenvolvida exclusivamente para esse projeto. Outro ponto é o fato de todo o simulador ter sido desenvolvido antes de testar esses controllers, pois no início do projeto ainda havia a possibilidade do desenvolvimento do próprio controlador. Dessa forma, ao longo do projeto, percebeu-se que as referências do plano cartesiano utilizadas pelo simulador desenvolvido eram diferentes das utilizadas pelo Motion Controller, o que exigiria uma mudança em todo o sistema desenvolvido.

Com o intuito de resolver essa questão e ainda assim proporcionar uma experiência real para o usuário, foram utilizados os eixos Y do thumbstick do touch do Oculus Rift (Como pode ser vistoa na Figura 4.22) como entrada de sinais para o deslocamento de avanço.

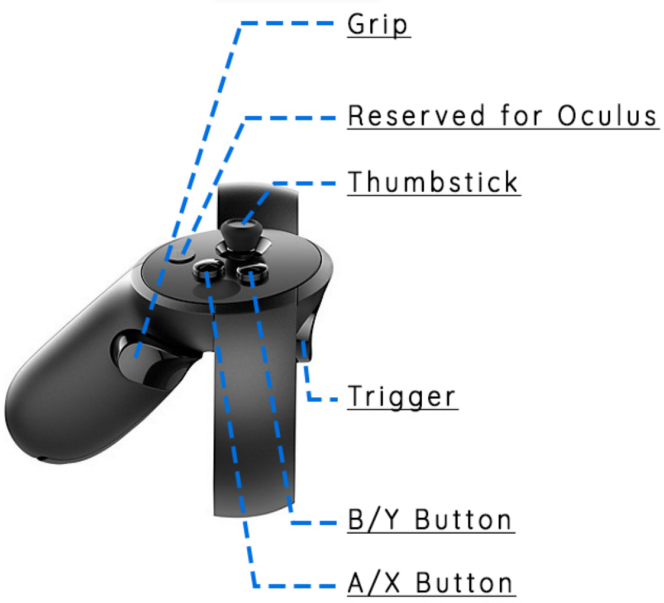

Figura 4.23 – Botões do Touch Controller (OVATION..., 2023).

Quando o usuário desloca o thumbstick na direção do seu eixo em Y, o evento denominado "ThumbstickY"é disparado, chamando a função "DeslocamentoProfundidade"que recebe como parâmetro um float relativo à sensibilidade do movimento empregado pelo usuário. Esse float pode variar de 1 a -1. Por exemplo, se o usuário desloca o eixo do thumbstick até a metade do seu movimento completo no eixo Y positivo, o float irá retornar o valor de 0.5, assim como se o movimento for em Y negativo, o float irá retornar -0.5. Após ser chamada, a função "DeslocamentoProfundidade"realiza o deslocamento da garra em direção ao seu eixo para onde a garra está apontada. Na figura abaixo pode ser observado um pseudocódigo que representa de maneira simplificada o princípio de funcionamento do código para o movimento de rotação da garra.

```
1
2   /*Quando o usuário movimenta o Thumbstick em Y, o programa reconhece um valor
3   diferente de zero, que aciona a cadeia de eventos seguinte*/
4   if (ThumbstickY != 0):
5
6       /*Ao deslocar o Thumbsticks o sistema recebe um valor que vai de -1 a 1,
7       representado o movimento feito pelo usuário no Thumbstick em Y*/
8
9       /*Esse valor é salvo em DeslocamentoThumbstickY*/
10      DeslocamentoThumbstickY = ThumbstickY
11
12      /*A função DeslocamentoProfundidade é chamada*/
13      DeslocamentoProfundidade(DeslocamentoThumbstickY)
14
15  /*Definição da função DeslocamentoProfundidade*/
16  void DeslocamentoProfundidade(DeslocamentoThumbstickY){
17
18      /*getFowardVector é uma funcionalidade da Unreal que retorna o valor
19      do vetor para onde o objeto(Garra) está apontado*/
20      FowartVectorAtual = getFowardVector(Garras)
21
22      /*Se o parâmetro recebido for menor que zero realiza um movimento de retorno*/
23
24      /*Se o parâmetro recebido for maior que zero realiza um movimento de avanço*/
25      FowartVectorNovo = FowartVectorAtual + DeslocamentoThumbstickY
26
27      /*Função da Unreal que define o novo FowardVector*/
28      setFowardVector(FowartVectorNovo)
29
30      return 0
31
32  }
```

Figura 4.24 – Pseudocódigo do movimento de avanço

Esses são os dois movimentos de deslocamento no espaço que a garra realiza. Contudo, existem mais três movimentos essenciais para que a garra possa executar suas funções dentro do simulador. São esses movimentos o de abrir e fechar as pinças da garra, a rotação da ponta da garra e a rotação do apoio das pinças. Todos esses movimentos serão melhor explicados nas seções a seguir.

4.7.3 Abrir e fechar pinças

No exemplo proposto neste trabalho, o treinamento requer que o usuário seja capaz de segurar um objeto com formato cúbico e colocá-lo em um pote com a mesma cor do objeto (o exemplo foi explicado melhor na seção 4.1). Para isso, é necessário que a pinça da garra seja capaz de abrir e fechar para segurar e soltar o objeto quando desejado. Para tal função, foram utilizados os gatilhos "trigger"e "grip"do touch controller (que podem ser vistos na Figura 4.22). Quando o gatilho "grip"é pressionado, o evento chamado "Grip"é iniciado. Assim como no movimento de avanço da garra, o "Grip", além de ser chamado, armazena em uma variável do tipo float a "sensibilidade"do movimento de fechar a garra. Após o início do evento "Grip", ele chama a função "FecharPincas", que é responsável por realizar o movimento de fechamento das pinças da garra.

Por consequência, quando o usuário pressiona o gatilho "trigger", o evento "TRIG-

GER"é iniciado, armazenando em uma variável do tipo float uma constante de sensibilidade. Essa constante é utilizada como entrada da função "AbrirPincas", que é responsável pelo movimento de abertura das pinças da garra.

4.7.4 Rotação Ponta da Garra

A rotação da ponta da garra consiste no movimento de rotação da ponta da garra em torno do próprio eixo, o mesmo eixo em que a garra faz o deslocamento de avanço. Para realizar essa rotação, foram utilizadas as informações de velocidade de rotação angular do Motion Controller. O Motion Controller fornece informações essenciais sobre os movimentos realizados pelo usuário através do touch controller. Uma dessas informações é a velocidade angular na qual os movimentos de rotação são realizados.

Para esse movimento, também foi utilizado como evento de disparo o Event Tick, que a cada 0,016 segundos verifica qual a velocidade de rotação angular do Motion Controller em torno do próprio eixo. Ele salva essa informação em uma variável do tipo float e a envia para a função chamada "RotacaoPontaDaGarra". Esse float com a informação da velocidade de rotação serve como uma constante de sensibilidade que informa quanto a ponta da garra deve rodar. Logo, quando o Motion Controller não detecta movimento de rotação em torno do próprio eixo, ele envia para a variável o valor 0, fazendo com que a ponta da garra não rotacione.

As funções "AbrirPincas", "FecharPincas", "DeslocamentoProfundidade"e "RotacaoPontaDaGarra"foram criadas em c++ e o seus códigos podem ser visualizadas na seção "Apêndice". Diferente das funções ligadas ao "Event TicK", que são funções disponibilizadas pela própria Unreal(Com exeção da "RotacaoPontaDaGarra") .

5 Validação do protótipo

Um processo de validação ideal para um simulador, seja ele de qual natureza for, passa pela comparação desse simulador com o processo simulado, ou seja, a melhor maneira de validar o simulador de realidade virtual aplicado à cirurgia robótica seria a comparação do mesmo com um robô de cirurgia robótica real, como o Da Vinci.

Contudo, embora os robôs de cirurgia robótica tenham conquistado espaço no mercado médico, ainda há escassez desses sistemas avançados em muitas instituições médicas. Em uma última atualização de julho de 2023, segundo a própria empresa que desenvolve o robô Da Vinci, existem apenas 103 exemplares desse robô no território brasileiro (BASE DE SÃO JOSÉ DO RIO PRETO, 2021). A aquisição e implementação de robôs cirúrgicos envolvem custos substanciais, incluindo o investimento inicial no equipamento e as despesas contínuas associadas à manutenção e treinamento.

Diante deste cenário, o acesso a esses robôs para pesquisas científicas é limitado, fazendo com que outras alternativas de validação sejam buscadas para obter dados satisfatórios sobre a eficiência do simulador desenvolvido.

Para a validação do protótipo desenvolvido neste projeto, pela falta de acesso ao robô de cirurgia real, foi escolhida uma validação qualitativa. Essa abordagem de pesquisa foca na compreensão aprofundada e subjetiva de fenômenos, frequentemente utilizando métodos como observação, entrevistas e análise de conteúdo. Em vez de depender de medições numéricas, busca insights, significados e padrões em dados não quantificáveis para obter uma compreensão holística de um fenômeno ou processo.

Para a validação qualitativa, um grupo de pessoas foi selecionado para praticar um exercício no simulador da FlexVR (o mesmo da Figura 4.9). Após a experiência com esse simulador já consolidado no mercado, eles foram submetidos a responder o seguinte questionário com quatro perguntas:

- **Pergunta 1** - De 0 a 10, qual nota você daria para a noção de profundidade que o simulador proporciona?
- **Pergunta 2** - De 0 a 10, qual nota você daria para a experiência visual do teste do simulador?
- **Pergunta 3** - De 0 a 10, Qual nota você daria para a resposta do simulador aos movimentos realizados pela sua mão?
- **Pergunta 4** - De 0 a 10, qual nota você daria para a curva de aprendizagem no manuseio da garra do simulador?

Em seguida, o mesmo grupo de pessoas foi submetido a um exercício similar ao da prática no simulador FlexVR, mas desta vez no simulador desenvolvido neste projeto (o mesmo exercício da Figura 4.8). Após a realização da segunda prática, eles foram submetidos a responder o mesmo questionário anterior.

	R1	R2	R3	R4	MPP
Pessoa 1	8	9	8	8	8,25
Pessoa 2	8	6	9	9	8
Pessoa 3	8	8	7	8	9,75
Pessoa 4	10	9	10	10	9,75
MPR	8,5	8	8,5	8,75	

Tabela 5.2 – Respostas do questionário do FlexVR

	R1	R2	R3	R4	MPP
Pessoa 1	3	8	4	3	4,5
Pessoa 2	2	2	6	3	3,25
Pessoa 3	5	5	6	9	6,25
Pessoa 4	4	6	4	3	4,25
MPR	3,5	5,25	5	4,5	

Tabela 5.3 – Respostas do questionário do simulador desenvoldido

Importante frisar que os participantes foram escolhidos de maneira aleatória, com o requisito de nunca terem tido contato com nenhum dos dois simuladores antes. Além disso, antes de iniciarem os testes, foi explicado para cada participante o que eram os simuladores, os seus propósitos e uma breve introdução sobre o projeto em desenvolvimento. Ficou claro também a importância da veracidade na escolha das respostas ao formulário.

As respostas dos formulários foram colocadas nas tabelas 5.23 e 5.24, onde MPP significa Média Por Pessoa e MPR significa Média Por Resposta. Além das tabelas, a seguir é possível observar gráficos que trazem um componente visual para a avaliação dos resultados da validação qualitativa. A análise dos dados obtidos nesta seção será melhor explanada na seção "Análise de Resultados".

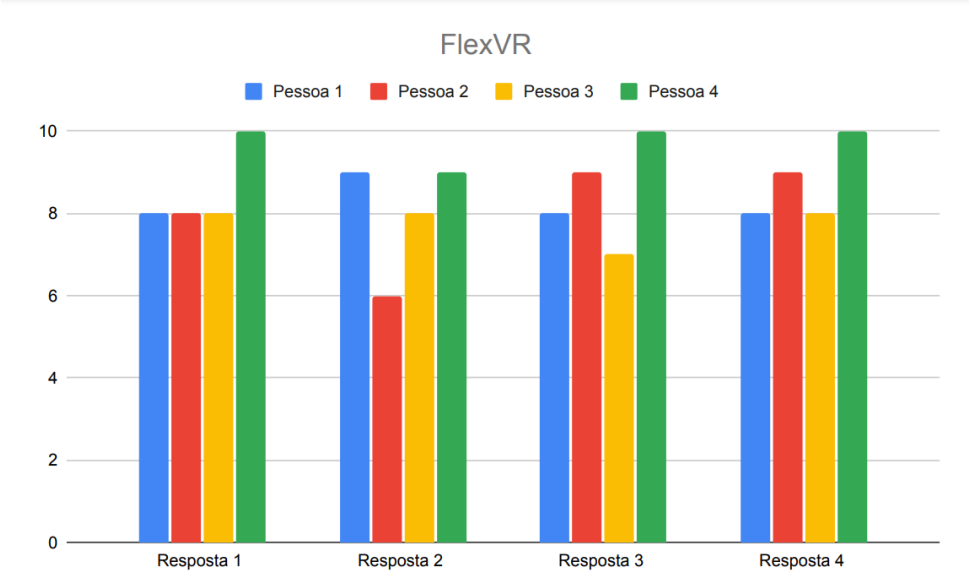

Figura 5.25 – Respostas do questionário sobre a simulação no FlexVR

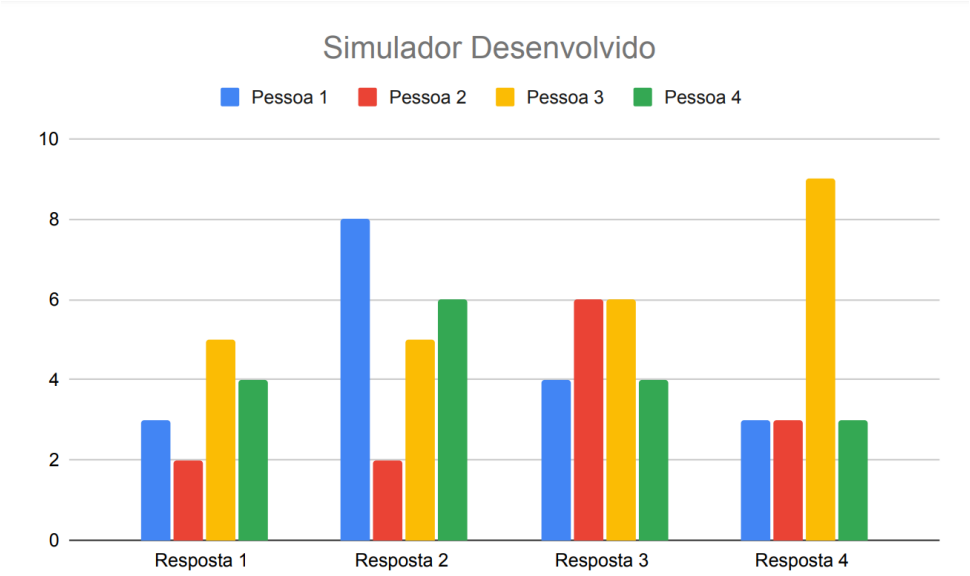

Figura 5.26 – Respostas do questionário sobre o simulação desenvolvido

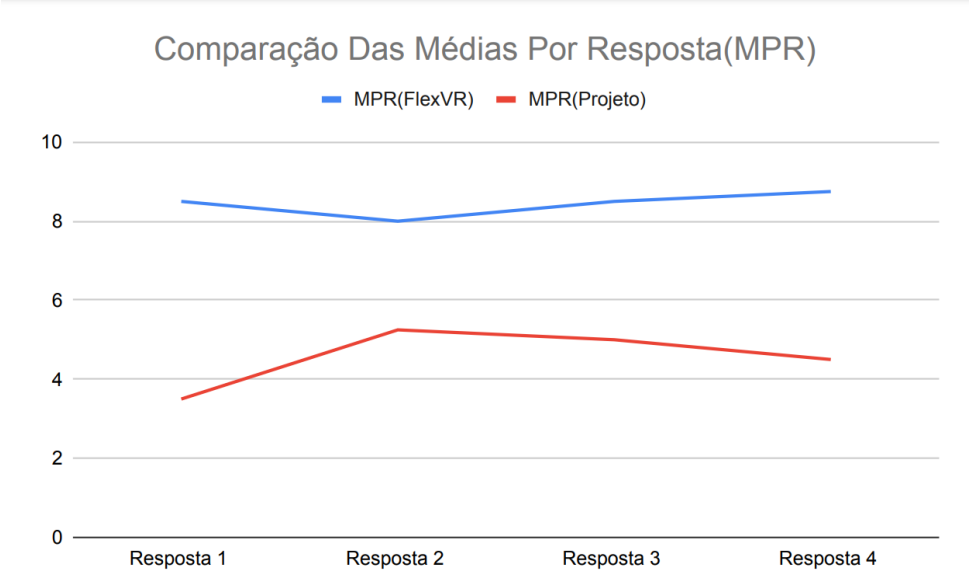

Figura 5.27 – Comparação das médias por respostas

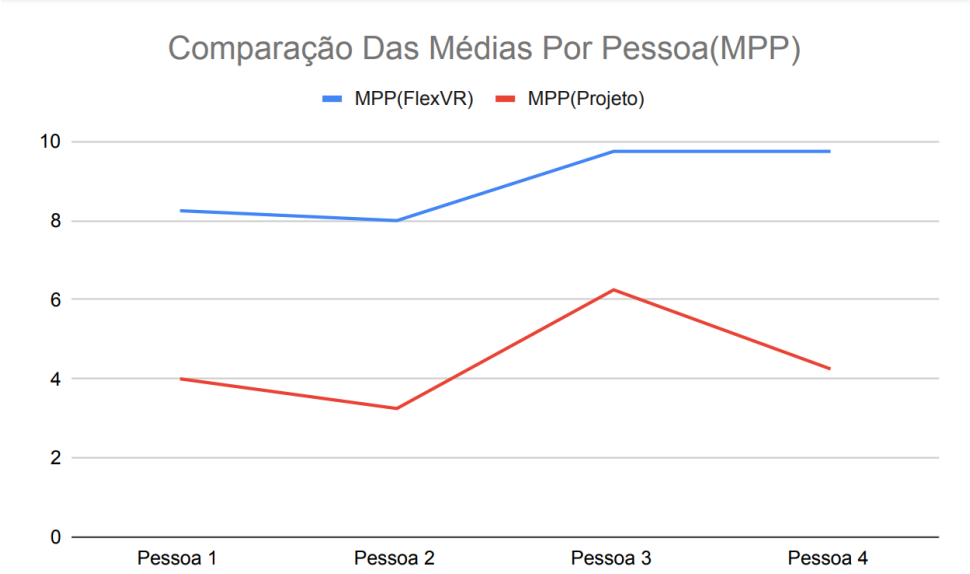

Figura 5.28 – Comparação das médias por pessoa.

6 Análise dos resultados

Esta seção será focada na análise dos resultados da validação qualitativa exposta na seção "Validação". Antes de se analisar os resultados obtidos, é necessário, primeiro, entender que se trata de uma comparação de um simulador já validado no mercado, com um grande investimento em pesquisa e validação, e um protótipo de um simulador desenvolvido por um único autor, dentro do período de aproximadamente 6 meses, sem nenhum investimento financeiro significativo. Ainda assim, essa comparação se mostra de grande valia para a continuidade deste projeto, pois traz à luz os pontos que o trabalho está caminhando na direção correta e os pontos que precisam de mais atenção no desenvolvimento.

Ao analisar os gráficos das Figuras 5.23 e 5.24, é possível observar uma grande variação nas respostas de diferentes pessoas para as mesmas perguntas. Isso é uma consequência inevitável de validações qualitativas, devido à singularidade da experiência de cada pessoa ao utilizar o simulador. Por exemplo, a pessoa 4 apresentou maior facilidade em aprender a utilizar o simulador desenvolvido, enquanto as demais pessoas tiveram maior dificuldade. Além disso, algumas perguntas envolvem questões subjetivas, como a pergunta sobre a experiência visual, que leva em consideração aspectos particulares de cada um sobre o que constitui uma boa experiência visual. Enquanto alguns focam na parte mais artística do cenário, outros valorizam a suavidade dos movimentos.

No simulador desenvolvido, as variações entre as respostas para a mesma pergunta foram mais acentuadas, como pode ser visto na Figura 5.24. Isso se deve a alguns fatores, como a utilização de um Headset na cabeça, o que pode ser uma experiência mais desconfortável para pessoas que possuem problemas de visão e não podem usar o Headset com óculos. Outro ponto é o fato de o simulador FlexVR possuir uma estrutura que torna mais intuitivos os limites de movimentação da mão do usuário, além do melhor refinamento do próprio simulador, no que diz respeito ao cenário e à experiência estereoscópica.

Um fenômeno interessante observado no gráfico 5.25 é que a pergunta para a qual o FlexVR obteve a menor média foi a mesma para a qual o simulador desenvolvido obteve sua maior média. Ao passo que o contrário também foi observado: a pergunta para a qual o FlexVR obteve a maior média foi a mesma para a qual o simulador desenvolvido obteve sua menor média. Isso mostra que os dois simuladores tiveram destaques individuais em áreas diferentes. Enquanto o FlexVR destacou-se na curva de aprendizagem e na noção de profundidade, o simulador desenvolvido destacou-se na experiência visual dos usuários, tendo como seu ponto fraco a noção de profundidade.

A Figura 5.25, ao mostrar a comparação da média por pessoa, mais uma vez, traz à luz a questão da subjetividade de cada participante da validação. Como pode ser

observado, cada pessoa que participou do teste manteve o mesmo padrão de resposta em ambos os simuladores, no sentido de que a pessoa que obteve a menor média entre as respostas utilizando o simulador FlexVR também obteve a menor nota entre as respostas para o simulador desenvolvido, e o contrário também foi observado.

Diante deste cenário, para melhorar o entendimento dos resultados obtidos, separou-se a análise de cada uma das quatro perguntas do questionário em seções diferentes, como pode-se observar a seguir.

6.1 Noção de profundidade

A primeira pergunta do questionário foi "De 0 a 10, qual nota você daria para a noção de profundidade que o simulador proporciona?". Essa pergunta tinha como objetivo avaliar a geração de imagem estereoscópica dos simuladores, sendo essencial para uma experiência realística dentro do simulador de cirurgia robótica, pois ela é responsável pela noção de profundidade.

Cirurgia robótica frequentemente envolve a manipulação de instrumentos através de pequenos orifícios ou incisões. Uma boa noção de profundidade permite que os cirurgiões robóticos avaliem com precisão a distância entre os instrumentos e os tecidos-alvo, facilitando movimentos precisos. Além disso, com uma noção adequada de profundidade, os cirurgiões podem evitar danos a tecidos circundantes que não fazem parte do procedimento cirúrgico. Isso é particularmente crucial em cirurgias delicadas, onde a precisão é essencial. Soma-se a isso o fato de que a anatomia do corpo humano é tridimensional, e uma boa noção de profundidade permite aos cirurgiões visualizar e compreender as estruturas anatômicas em três dimensões. Isso é vital para procedimentos complexos, onde a orientação espacial é essencial. Por fim, a noção de profundidade contribui para a percepção visual tridimensional, permitindo aos cirurgiões julgar a distância e a posição relativa de diferentes estruturas anatômicas. Isso é particularmente relevante em procedimentos que exigem alta precisão.

O simulador FlexVR recebeu uma média de 8,5 nos votos, demonstrando que ele entrega ótimos resultados no quesito de geração de imagem estereoscópica. Em contrapartida, o protótipo desenvolvido neste projeto recebeu uma média de 3,5 nos votos, demonstrando que a geração de imagem estereoscópica é um ponto que merece mais atenção no desenvolvimento dos próximos passos do projeto. O simulador desenvolvido utiliza a geração de imagem estereoscópica do próprio Headset do Oculus Rift, ao passo que o simulador FlexVR utiliza um óculos 3D ativo. Portanto, mesmo sendo mais simples, obteve melhores resultados. Um caminho a ser testado nos próximos passos deste trabalho é a utilização de óculos 3D ativos para a experiência estereoscópica dos usuários.

6.2 Experiência Visual

A segunda pergunta do questionário foi "De 0 a 10, qual nota você daria para a experiência visual do teste do simulador?". Essa pergunta tinha como intenção analisar a maneira como os elementos visuais são apresentados e percebidos pelo usuário durante a interação com o ambiente virtual. O conceito de experiência visual é amplo e abrange vários aspectos relacionados à representação gráfica, design de arte, animações e qualidade visual geral do simulador. A qualidade gráfica de um simulador, incluindo texturas, modelos 3D, efeitos visuais e iluminação, desempenha um papel crucial na experiência visual. Gráficos realistas, efeitos visuais impressionantes e uma apresentação visualmente atraente contribuem para uma experiência envolvente. Além disso, animações suaves e realistas melhoram a experiência visual, contribuindo para a imersão do usuário. Movimentos fluidos dos objetos em cena e ambientes aumentam a sensação de presença no mundo virtual.

O simulador FlexVR recebeu uma média de 8 nos votos, demonstrando que ele entrega uma ótima experiência visual para os seus usuários. Em contrapartida, o protótipo desenvolvido neste projeto recebeu uma média de 5,25 nos votos, demonstrando que, por mais que precise de refinamento nesta área, o projeto está no caminho certo em relação à experiência visual. Por se tratar de uma nota média, pode-se observar esse resultado sob dois aspectos. O primeiro, é o aspecto positivo, onde se atribui essa nota à utilização acertiva da Unreal Engine, pois trata-se de uma engine que se destaca no mercado pela sua experiência visual. O segundo aspecto é o negativo, onde atribui-se a ausência de uma nota maior na validação qualitativa ao menor investimento de tempo no quesito visual do projeto em comparação com a sua usabilidade.

Pensando na continuidade do projeto, pode-se analisar a possibilidade de montar uma equipe focada na parte visual do projeto, envolvendo profissionais como designers e artistas gráficos. Além disso, aprofundar o conhecimento nas animações visuais dentro do simulador, fluidez nos movimentos, efeitos de transição e posições da câmera e dos objetos em cena.

6.3 Respostas aos Movimentos

A terceira pergunta do questionário foi "De 0 a 10, Qual nota você daria para a resposta do simulador aos movimentos realizados pela sua mão?". Essa pergunta tinha como intenção analisar a parte mais complexa do desenvolvimento do projeto, a leitura do movimento das mãos através dos Touch Controllers e a tradução da leitura desses dados em movimento das garras dentro do simulador. A leitura precisa do movimento das mãos permite que o simulador interprete os movimentos do cirurgião de forma precisa. Isso é essencial para a manipulação precisa dos instrumentos cirúrgicos, replicando a destreza manual necessária em procedimentos reais. Além disso, uma leitura precisa e em tempo

real do movimento das mãos contribui para uma experiência do usuário mais realista. Isso aumenta a sensação de imersão e proporciona um ambiente de treinamento virtual mais próximo das condições reais de uma sala de cirurgia. Soma-se a isso o fato de que cirurgias robóticas exigem uma coordenação precisa entre os movimentos das mãos e a visualização da tela. A leitura do movimento das mãos no simulador contribui para o treinamento eficaz dessa coordenação mão-olho, fundamental para cirurgias precisas.

O simulador FlexVR recebeu uma média de 8,5 nos votos, demonstrando que ele entrega uma ótima resposta aos movimentos das mãos do usuário. Em contrapartida, o protótipo desenvolvido neste projeto recebeu uma média de 5 nos votos. Parte desta diferença está ligada ao fato de o simulador FlexVR possuir o seu próprio controller para a captura dos movimentos das mãos do usuário, criado especialmente para as demandas de um simulador de cirurgia robótica, incluindo o seu formato e principalmente a sua integração com o simulador. Em contrapartida, o simulador desenvolvido neste projeto utilizou o Touch Controller do Oculus Rift, que não foi desenvolvido especificamente para esse fim, tornando mais complexas as soluções para se chegar a um resultado de resposta aos movimentos satisfatório. Para os próximos passos do projeto, com o intuito de melhorar o desempenho do simulador em relação às respostas ao movimento, pretende-se desenvolver um controller específico para o simulador.

6.4 Curva de Aprendizagem

A quarta pergunta do questionário foi "De 0 a 10, qual nota você daria para a curva de aprendizagem no manuseio da garra do simulador?". Essa pergunta tinha como objetivo analisar o tempo que o usuário demorava para se habituar com os comandos do simulador. A curva de aprendizagem em um simulador de cirurgia robótica é um fator crucial que influencia diretamente a eficácia do treinamento, por conseguinte, a competência do cirurgião em realizar procedimentos robóticos. A utilização de um simulador de cirurgia robótica muitas vezes envolve a familiarização com uma interface específica, que pode incluir consoles de controle e monitores. A curva de aprendizagem ajuda os cirurgiões a se adaptarem a essa interface, garantindo uma operação eficaz durante procedimentos reais. Além disso, a curva de aprendizagem também serve como uma ferramenta de avaliação do desempenho do cirurgião em treinamento. A monitorização contínua permite identificar áreas que precisam de aprimoramento e personalizar o treinamento para atender às necessidades individuais.

O simulador FlexVR recebeu uma média de 8,75 nos votos, demonstrando que, por mais que se trate de um software complexo em relação ao seu desenvolvimento, novos usuários demoram pouco tempo para se habituar com os comandos do simulador. Em contrapartida, o protótipo desenvolvido neste projeto recebeu uma média de 4,5 nos votos, demonstrando que, por mais que precise de refinamento nesta área, o projeto está no caminho

certo em relação ao quão intuitivo é para os usuários se habituarem com os comandos do simulador.(MARON, 2019)(TECHTUDO, 2017)

6.5 Video do Simulador sendo Testado

No link a seguir, é possível visualizar o simulador desenvolvido neste trabalho sendo testado. Embora a imagem da tela do simulador não esteja clara, é possível analisar a resposta aos movimentos realizados pelas mãos do usuário, além de obter uma noção geral de como funciona a utilização do simulador.

(TESTE SIMULADOR DE CIRURGIA ROBÓTICA, Link)

7 Conclusões

Em síntese, este trabalho buscou desenvolver um protótipo de um simulador de cirurgia robótica com características parecidas com simuladores no mercado. Durante o processo de pesquisa, implementação e validação do simulador, diversas considerações e desafios foram enfrentados, fornecendo insights valiosos para os próximos passos do desenvolvimento do projeto.

A análise dos resultados obtidos na fase de validação qualitativa revelou pontos de destaque e áreas que requerem atenção adicional no desenvolvimento futuro do simulador. A comparação com o simulador consolidado no mercado permitiu identificar tanto pontos de convergência quanto características distintivas, destacando a singularidade do simulador desenvolvido neste projeto.

A noção de profundidade, fundamental para procedimentos cirúrgicos precisos, mostrou-se como um ponto sensível, demandando investimento de maior atenção e levando à possibilidade da substituição do Oculus de Realidade Virtual por um óculos 3D ativo para a experiência estereoscópica do usuário. A experiência visual, embora tenha obtido resultados promissores, apontou para a necessidade de uma abordagem mais refinada, indicando a possibilidade da criação de uma equipe de especialistas focada na parte visual do simulador, visando aprimorar a qualidade gráfica e a imersão do usuário.

As respostas aos movimentos, especialmente a leitura precisa das mãos, diante da sua complexidade, obtiveram um resultado satisfatório, tratando-se de um protótipo. Ainda assim, trouxeram à tona a importância de desenvolver controladores específicos para o simulador para uma integração mais eficaz e realista. A curva de aprendizagem, um aspecto vital na formação de cirurgiões, indicou uma aceitação razoável, mas ressaltou a relevância contínua de tornar a interface do simulador mais intuitiva e fácil de aprender.

Olhando para o futuro, recomenda-se a continuidade do projeto com foco na otimização da noção de profundidade, no refinamento da experiência visual e na pesquisa de soluções para a leitura de movimentos. A colaboração interdisciplinar com profissionais de design, engenharia de software e medicina pode enriquecer ainda mais o desenvolvimento do simulador.

Este trabalho não apenas contribui para o avanço na área de simulação médica, mas também destaca a importância contínua de inovações tecnológicas no treinamento de cirurgiões. O simulador desenvolvido representa uma etapa significativa nesse processo, fornecendo uma plataforma sólida para futuras melhorias e contribuindo para a formação de profissionais capacitados na área de cirurgia robótica.

Referências

ALMEIDA, L. H. N. d. Imersão em realidade virtual através de um jogo RPG, Trabalho de conclusão de curso,FGA-unb, Brasília, 2019. Brasília, 2019. Citado na p. 19.

AMAZON Product Image. 2023. Disponível em: <https://m.media-amazon.com/images/I/61ueGFutGgL._AC_UF1000,1000_QL80_.jpg>. Citado na p. 35.

AMAZON Product Image 2. 2023. Disponível em: <https://m.media-amazon.com/images/I/51CQ+1JsMSL.jpg>. Citado na p. 35.

BASE DE SÃO JOSÉ DO RIO PRETO, H. de. **Hospital de Base de São José do Rio Preto adquire da Vinci Xi, o robô cirúrgico mais avançado do mundo**. Acesso em: DD/MM/AAAA. 2021. Disponível em: <https://www.hospitaldebase.com.br/noticia/hospital-de-base-de-sao-jose-do-rio-preto-adquire-da-vinci-xi-o-robo-cirurgico-mais-avancado-do-mundo#:~:text=Segundo%20o%20executivo%20da%20Strattner,m%C3%A9dicas%E2%80%9D%2C%20pontua%20Jos%C3%A9%20Ricardo.>. Citado na p. 45.

BASTOS, V. T. **Oásis: refúgio digital em realidade virtual**. 2021. Trabalho de Conclusão de Curso – Universidade de Brasília, Brasília. Citado na p. 19.

BELOTTO, M. e. a. INFLUÊNCIA DA EXPERIÊNCIA DA CIRURGIA LAPAROSCÓPICA NA DESTREZA DA CIRURGIA ROBÓTICA. **ABCD. Arquivos Brasileiros de Cirurgia Digestiva (São Paulo)**, v. 34, 2022. Citado na p. 17.

CARVALHO, M. P. d. **Controle de movimentação de humanóide em tempo real por teleoperação**. 2014. Trabalho de Graduação em Engenharia de Controle e Automação – Universidade de Brasília, Brasília. Citado na p. 19.

DA VINCI SURGERY COMMUNITY. **da Vinci Surgery Skills Simulator**. 2023. Disponível em: <https://www.davincisurgerycommunity.com/Systems_I_A/Skills_Simulator>. Citado na p. 16.

EPIC GAMES. **Enhanced Input in Unreal Engine**. 2023a. Disponível em: <https://docs.unrealengine.com/5.0/en-US/enhanced-input-in-unreal-engine/#:~:text=Getting%20Started,Plugin%2C%20then%20restart%20the%20editor.>. Citado na p. 37.

EPIC GAMES. **Unreal Engine VR Motion Controller How-Tos**. 2023b. Disponível em: <https://docs.unrealengine.com/4.27/en-US/SharingAndReleasing/XRDevelopment/VR/VRHowTos/MotionController/>. Citado na p. 37.

ESPECIFICADOS, A. não. **Range of Motion of the da Vinci EndoWrist End Effector**. Ano não especificado. Disponível em: <https://www.researchgate.net/figure/The-range-of-motion-of-the-da-Vinci-EndoWrist-end-effector-The-EndoWrist-has-motions_fig4_341714471>. Citado na p. 32.

IMAGEM do Google. Ano não especificado. Disponível em: <https://encrypted-tbn0.gstatic.com/images?q=tbn:ANd9GcTk1cvHgg4zaeJkelbFGkLo3VKEojULkRwPpMjO87Y_-AtIDPXJO5DMQwJe2OE51YXTZvw&usqp=CAU>. Citado na p. 40.

LEFOR, A.; HEREDIA PÉREZ, S.; SHIMIZU, A.; LIN, H.; WITOWSKI, J.; MITSUISHI, M. Development and Validation of a Virtual Reality Simulator for Robot-Assisted Minimally Invasive Liver Surgery Training. **J Clin Med**, v. 11, n. 14, p. 4145, 2022. DOI: 10.3390/jcm11144145. Citado na p. 20.

LIMA, A. e. a. Projeto para desenvolvimento do simulador Health Simulator. In: ANAIS do Computer on the Beach. 2015. P. 279–288. Citado na p. 20.

LONGMORE, S.; NAIK, G.; GARGIULO, G. Laparoscopic Robotic Surgery: Current Perspective and Future Directions. **Robotics**, v. 9, n. 2, p. 42, 2020. DOI: 10.3390/robotics9020042. Citado na p. 29.

MACHADO, L. d. S. **A realidade virtual no modelamento e simulação de procedimentos invasivos em oncologia pediátrica: um estudo de caso no transplante de medula óssea**. 2003. Tese de Doutorado – Universidade de São Paulo. Citado na p. 19.

MARON, P. **Câncer de Próstata e a Cirurgia Robótica**. Acesso em: [Inserir a data em que você acessou o artigo]. 2019. Disponível em: <https://drpaulomaron.com.br/cancer-prostata-cirurgia-robotica/>. Citado na p. 53.

MIMIC TECHNOLOGIES. **Mimic dV-Trainer**. 2023. Disponível em: <https://www.medicalexpo.com/pt/prod/mimic-technologies/product-112216-739694.html>. Citado na p. 17.

MORI, T.; IKEDA, K.; TAKESHITA, N. e. a. Validation of a novel virtual reality simulation system with the focus on training for surgical dissection during laparoscopic sigmoid colectomy. **BMC Surg**, v. 22, p. 12, 2022. DOI: 10.1186/s12893-021-01441-7. Citado na p. 20.

MORRELL, A. L. G.; MORRELL-JUNIOR, A. C.; MORRELL, A. G.; MENDES, J. M. F.; TUSTUMI, F.; MORRELL, A. Evolução e história da cirurgia robótica: da ilusão à realidade. **Colégio Brasileiro de Cirurgiões**, Brasil, 2021. DOI: 10.1590/0100-6991e-20202798. Disponível em: <https://doi.org/10.1590/0100-6991e-20202798>. Citado na p. 28.

OCULUS Rift: entenda como funciona e como ele pode revolucionar os games. 2013. Disponível em: <https://www.techtudo.com.br/noticias/2013/07/oculus-rift-entenda-como-funciona-e-como-ele-pode-revolucionar-os-games.ghtml>. Citado na p. 35.

OVATION VR Motion Controllers Documentation. 2023. Disponível em: <https://docs.ovationvr.com/ovation/legacy/pc-vr/ovation-vr/motion-controllers>. Citado na p. 42.

PINTO, L. T. G. e. a. DESENVOLVIMENTO DE SIMULADOR DE VOO AGRÍCOLA DE PULVERIZAÇÃO USANDO REALIDADE VIRTUAL. In: VII JORNACITEC-Jornada Científica e Tecnológica. 2018. Citado na p. 20.

RIBAS, G. C.; RIBAS, E. C.; RODRIGUES JUNIOR, A. J. O cérebro, a visão tridimensional, e as técnicas de obtenção de imagens estereoscópicas. **Revista de Medicina**, v. 85, n. 3, p. 78–90, 2006. Acesso em: 6 jun. 2023. DOI: 10.11606/issn.1679-9836.v85i3p78-90. Disponível em: <https://www.revistas.usp.br/revistadc/article/view/59218>. Citado na p. 14.

SURGICAL SCIENCE. **FlexVR**. 2023a. Disponível em: <https://surgicalscience.com/simulators/flexvr/?gclid=Cj0KCQiA67CrBhC1ARIsACKAa8QF9jBvVq0kKlBjKgG5Qj3xqknLx__YWsC5VnEbNuW0U_fiUe2SdnoaAhpYEALw_wcB>. Citado na p. 17.

SURGICAL SCIENCE. **RobotiX Mentor**. 2023b. Disponível em: <https://surgicalscience.com/simulators/robotix-mentor/>. Citado na p. 17.

TECHTUDO. **Descubra se vale investir em equipamentos para realidade virtual**. Acesso em: [Inserir a data em que você acessou o artigo]. 2017. Disponível em: <https://www.techtudo.com.br/noticias/2017/10/descubra-se-vale-investir-em-equipamentos-para-realidade-virtual.ghtml>. Citado na p. 53.

TESTE SIMULADOR DE CIRURGIA ROBÓTICA, L. **0:00 / 0:50 - Trabalho de Conclusão de Curso: Simulador de Cirurgia Robótica**. Link. Disponível em: <https://youtu.be/WqOiFd8v3kE?si=aftfNWkIYlTB41f1>. Citado na p. 53.

TORRES, R. A. **Instrumentação e controle de braço robótico para deficientes**. 2020. Trabalho de Conclusão de Curso – Universidade de Brasília, Brasília. Citado na p. 19.

Appendices

8 Apêndice

```
1
2  // Fill out your copyright notice in the Description page of
       Project Settings.
3
4
5  #include "AtorGarra.h"
6  #include "Components/StaticMeshComponent.h"
7  #include "Camera/CameraComponent.h"
8  #include "Components/InputComponent.h"
9  #include "GameFramework/Actor.h"
10 #include "Engine/EngineTypes.h"
11 #include "Engine/EngineBaseTypes.h"
12 #include <cmath>
13 #include "Components/BoxComponent.h"
14
15
16
17 // Sets default values
18 AAtorGarra::AAtorGarra()
19 {
20   // Set this pawn to call Tick() every frame.  You can turn this
       off to improve performance if you don't need it.
21   PrimaryActorTick.bCanEverTick = true;
22
23   OverlapContato1 = 0;
24   OverlapContato2 = 0;
25   OverlapContato1_2 = 0;
26   OverlapContato2_2 = 0;
27   ContatorMotion = 0;
28   PosicaoMC_R_Anterior = FVector(0.f, 0.f, 0.f);
29   PosicaoMC_L_Anterior = FVector(0.f, 0.f, 0.f);
30   DistanciaProfundidadeAntiga = 0;
31
32   Referencia =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia"));
33   RootComponent = Referencia;
34
35   ReferenciaGarras =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ReferenciaGarras"));
       //Referencia que substitui a camera
36   ReferenciaGarras->SetupAttachment(RootComponent);
37   ReferenciaGarras->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
38   ReferenciaGarras->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
       0.f, 0.f)));
39
```

```
40
41
42
43
44  //CAMERA
       ------------------------------------------------------------------------
45  ReferenciaCamera =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ReferenciaCamera"));
       //Referencia para deslocamento da camera
46  ReferenciaCamera->AttachToComponent(ReferenciaGarras,
       FAttachmentTransformRules::KeepRelativeTransform);
47  ReferenciaCamera->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
48  ReferenciaCamera->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
       0.f, 0.f)));
49
50  Camera =
       CreateDefaultSubobject<UCameraComponent>(TEXT("CameraFixa"));
       //Criando a Camera
51  Camera->AttachToComponent(ReferenciaCamera,
       FAttachmentTransformRules::KeepRelativeTransform);
52  Camera->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
53  Camera->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
       -35.f, 0.f)));
54  //------------------------------------------------------------------------
55
56  GarraDireita =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
       Garra Direita"));
57  GarraDireita->AttachToComponent(ReferenciaGarras,
       FAttachmentTransformRules::KeepRelativeTransform);
58  GarraDireita->SetRelativeLocation(FVector(-15.49347f,
       56.909849f, -25.715203f));
59  GarraDireita->SetRelativeRotation(FRotator::MakeFromEuler(FVector(139.4713
       -71.108736f, 181.229533f)));
60
61  ReferenciaGarraDireita =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
       Garra Deslocamento Lateral"));
62  ReferenciaGarraDireita->AttachToComponent(GarraDireita,
       FAttachmentTransformRules::KeepRelativeTransform);
63  ReferenciaGarraDireita->SetRelativeLocation(FVector(0.f, 0.f,
       0.f));
64  ReferenciaGarraDireita->SetRelativeRotation(FRotator::MakeFromEuler(FVecto
       0.f, 0.f)));
65
66  StaticMeshGarra =
       CreateDefaultSubobject<UStaticMeshComponent>(TEXT("CorpoGarra"));
67  StaticMeshGarra->AttachToComponent(ReferenciaGarraDireita,
       FAttachmentTransformRules::KeepRelativeTransform);
68  StaticMeshGarra->SetVisibility(true);
69  StaticMeshGarra->SetSimulatePhysics(false);
```

```
70  StaticMeshGarra->SetRelativeLocation(FVector(0.f, 0.f,
        -1760.0f));
71  StaticMeshGarra->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
        0.f, 0.f)));
72
73
74  PontaGarra =
        CreateDefaultSubobject<UStaticMeshComponent>(TEXT("PontaGarra"));
75  PontaGarra->AttachToComponent(StaticMeshGarra,
        FAttachmentTransformRules::KeepRelativeTransform);
76  PontaGarra->SetVisibility(true);
77  PontaGarra->SetSimulatePhysics(false);
78  PontaGarra->SetRelativeLocation(FVector(0.f, 0.f, 42.445f));
79  PontaGarra->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
        0.f, 0.f)));
80
81  ReferenciaRotacaoApoioPinca =
        CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
        Rotacao Apoio Pinca"));
82  ReferenciaRotacaoApoioPinca->AttachToComponent(PontaGarra,
        FAttachmentTransformRules::KeepRelativeTransform);
83  ReferenciaRotacaoApoioPinca->SetRelativeLocation(FVector(0.f,
        0.0f, 0.95f));
84  ReferenciaRotacaoApoioPinca->SetRelativeRotation(FRotator::MakeFromEuler(F
        0.f, 0.f)));
85
86  ApoioPinca =
        CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ApoioPinca"));
87  ApoioPinca->AttachToComponent(ReferenciaRotacaoApoioPinca,
        FAttachmentTransformRules::KeepRelativeTransform);
88  ApoioPinca->SetVisibility(true);
89  ApoioPinca->SetSimulatePhysics(false);
90  ApoioPinca->SetRelativeLocation(FVector(0.f, 0.f, 0.0f));
91  ApoioPinca->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.0f,
        0.f, 0.f)));
92
93  ReferenciaRotacaoPinca =
        CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
        Rotacao das Pincas"));
94  ReferenciaRotacaoPinca->AttachToComponent(ApoioPinca,
        FAttachmentTransformRules::KeepRelativeTransform);
95  ReferenciaRotacaoPinca->SetRelativeLocation(FVector(0.0f, 0.f,
        0.417f));
96  ReferenciaRotacaoPinca->SetRelativeRotation(FRotator::MakeFromEuler(FVecto
        0.f, 0.f)));
97
98  RolagemPinca =
        CreateDefaultSubobject<UStaticMeshComponent>(TEXT("RolagemPinca"));
99  RolagemPinca->AttachToComponent(ReferenciaRotacaoPinca,
        FAttachmentTransformRules::KeepRelativeTransform);
100 RolagemPinca->SetVisibility(true);
101 RolagemPinca->SetSimulatePhysics(false);
```

```
102   RolagemPinca->SetRelativeLocation(FVector(0.f, 0.02f, 0.f));
103   RolagemPinca->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
104
105   ReferenciaRotacaoPinca1 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao Pinca 1"));
106   ReferenciaRotacaoPinca1->AttachToComponent(RolagemPinca,
          FAttachmentTransformRules::KeepRelativeTransform);
107   ReferenciaRotacaoPinca1->SetRelativeLocation(FVector(0.f, 0.f,
          0.f));
108   ReferenciaRotacaoPinca1->SetRelativeRotation(FRotator::MakeFromEuler(FVect
          0.f, 0.f)));
109
110   ReferenciaRotacaoPinca2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao Pinca 2"));
111   ReferenciaRotacaoPinca2->AttachToComponent(RolagemPinca,
          FAttachmentTransformRules::KeepRelativeTransform);
112   ReferenciaRotacaoPinca2->SetRelativeLocation(FVector(0.f, 0.f,
          0.f));
113   ReferenciaRotacaoPinca2->SetRelativeRotation(FRotator::MakeFromEuler(FVect
          0.f, 0.f)));
114
115   Pinca01 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Pinca1"));
116   Pinca01->AttachToComponent(ReferenciaRotacaoPinca1,
          FAttachmentTransformRules::KeepRelativeTransform);
117   Pinca01->SetVisibility(true);
118   Pinca01->SetSimulatePhysics(false);
119   Pinca01->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
120   Pinca01->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
121
122   Pinca02 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Pinca2"));
123   Pinca02->AttachToComponent(ReferenciaRotacaoPinca2,
          FAttachmentTransformRules::KeepRelativeTransform);
124   Pinca02->SetVisibility(true);
125   Pinca02->SetSimulatePhysics(false);
126   Pinca02->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
127   Pinca02->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
128
129   ContatoPinca1 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Contato
          Pinca 1"));
130   ContatoPinca1->AttachToComponent(Pinca01,
          FAttachmentTransformRules::KeepRelativeTransform);
131   ContatoPinca1->SetVisibility(true);
132   ContatoPinca1->SetSimulatePhysics(false);
133   ContatoPinca1->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
```

```
134   ContatoPinca1->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
135   ContatoPinca1->SetCollisionProfileName("Trigger");
136   ContatoPinca1->OnComponentBeginOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoContatoPinca1);
137   ContatoPinca1->OnComponentEndOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoSeparacaoPinca1);
138
139
140
141   ContatoPinca2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Contato
          Pinca 2"));
142   ContatoPinca2->AttachToComponent(Pinca02,
          FAttachmentTransformRules::KeepRelativeTransform);
143   ContatoPinca2->SetVisibility(true);
144   ContatoPinca2->SetSimulatePhysics(false);
145   ContatoPinca2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
146   ContatoPinca2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
147   ContatoPinca2->SetCollisionProfileName("Trigger");
148   ContatoPinca2->OnComponentBeginOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoContatoPinca2);
149   ContatoPinca2->OnComponentEndOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoSeparacaoPinca2);
150
151
152   Argola =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Argola"));
153   Argola->AttachToComponent(Referencia,
          FAttachmentTransformRules::KeepRelativeTransform);
154
155
156
157   //GARRA2//-------------------------------------------------------------
158
159   GarraEsquerda =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Garra Esquerda"));
160   GarraEsquerda->AttachToComponent(ReferenciaGarras,
          FAttachmentTransformRules::KeepRelativeTransform);
161   GarraEsquerda->SetRelativeLocation(FVector(-15.49347f,
          56.909849f, -25.715203f));
162   GarraEsquerda->SetRelativeRotation(FRotator::MakeFromEuler(FVector(150.f,
          -75.f, 540.f)));
163
164   ReferenciaGarraEsquerda =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Garra Deslocamento Lateral - 2"));
165   ReferenciaGarraEsquerda->AttachToComponent(GarraEsquerda,
          FAttachmentTransformRules::KeepRelativeTransform);
```

```
166   ReferenciaGarraEsquerda->SetRelativeLocation(FVector(-67.821901f,
          -95.170846f, 4.193073f));
167   ReferenciaGarraEsquerda->SetRelativeRotation(FRotator::MakeFromEuler(FVect
          -85.f, 180.f)));
168
169   StaticMeshGarra_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("CorpoGarra
          - 2"));
170   StaticMeshGarra_2->AttachToComponent(ReferenciaGarraEsquerda,
          FAttachmentTransformRules::KeepRelativeTransform);
171
172
173   PontaGarra_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("PontaGarra
          - 2"));
174   PontaGarra_2->AttachToComponent(StaticMeshGarra_2,
          FAttachmentTransformRules::KeepRelativeTransform);
175   PontaGarra_2->SetVisibility(true);
176   PontaGarra_2->SetSimulatePhysics(false);
177   PontaGarra_2->SetRelativeLocation(FVector(0.f, 0.f, 42.445f));
178   PontaGarra_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
179
180
181   ReferenciaRotacaoApoioPinca_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao Apoio Pinca - 2"));
182   ReferenciaRotacaoApoioPinca_2->AttachToComponent(PontaGarra_2,
          FAttachmentTransformRules::KeepRelativeTransform);
183   ReferenciaRotacaoApoioPinca_2->SetRelativeLocation(FVector(0.f,
          0.0f, 0.95f));
184   ReferenciaRotacaoApoioPinca_2->SetRelativeRotation(FRotator::MakeFromEuler
          0.f, 0.f)));
185
186   ApoioPinca_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("ApoioPinca
          - 2"));
187   ApoioPinca_2->AttachToComponent(ReferenciaRotacaoApoioPinca_2,
          FAttachmentTransformRules::KeepRelativeTransform);
188   ApoioPinca_2->SetVisibility(true);
189   ApoioPinca_2->SetSimulatePhysics(false);
190   ApoioPinca_2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
191   ApoioPinca_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.0f,
          0.f, 0.f)));
192
193   ReferenciaRotacaoPinca_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao das Pincas - 2"));
194   ReferenciaRotacaoPinca_2->AttachToComponent(ApoioPinca_2,
          FAttachmentTransformRules::KeepRelativeTransform);
195   ReferenciaRotacaoPinca_2->SetRelativeLocation(FVector(0.0f, 0.f,
          0.417f));
```

```
196   ReferenciaRotacaoPinca_2->SetRelativeRotation(FRotator::MakeFromEuler(FVec
          0.f, 0.f)));
197
198   RolagemPinca_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("RolagemPinca
          - 2"));
199   RolagemPinca_2->AttachToComponent(ReferenciaRotacaoPinca_2,
          FAttachmentTransformRules::KeepRelativeTransform);
200   RolagemPinca_2->SetVisibility(true);
201   RolagemPinca_2->SetSimulatePhysics(false);
202   RolagemPinca_2->SetRelativeLocation(FVector(0.f, 0.02f, 0.f));
203   RolagemPinca_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
204
205   ReferenciaRotacaoPinca1_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao Pinca 1 - 2"));
206   ReferenciaRotacaoPinca1_2->AttachToComponent(RolagemPinca_2,
          FAttachmentTransformRules::KeepRelativeTransform);
207   ReferenciaRotacaoPinca1_2->SetRelativeLocation(FVector(0.f, 0.f,
          0.f));
208   ReferenciaRotacaoPinca1_2->SetRelativeRotation(FRotator::MakeFromEuler(FVe
          0.f, 0.f)));
209
210   ReferenciaRotacaoPinca2_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Referencia
          Rotacao Pinca 2 - 2"));
211   ReferenciaRotacaoPinca2_2->AttachToComponent(RolagemPinca_2,
          FAttachmentTransformRules::KeepRelativeTransform);
212   ReferenciaRotacaoPinca2_2->SetRelativeLocation(FVector(0.f, 0.f,
          0.f));
213   ReferenciaRotacaoPinca2_2->SetRelativeRotation(FRotator::MakeFromEuler(FVe
          0.f, 0.f)));
214
215   Pinca01_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Pinca1 -
          2"));
216   Pinca01_2->AttachToComponent(ReferenciaRotacaoPinca1_2,
          FAttachmentTransformRules::KeepRelativeTransform);
217   Pinca01_2->SetVisibility(true);
218   Pinca01_2->SetSimulatePhysics(false);
219   Pinca01_2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
220   Pinca01_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
221
222   Pinca02_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Pinca2
          -2"));
223   Pinca02_2->AttachToComponent(ReferenciaRotacaoPinca2_2,
          FAttachmentTransformRules::KeepRelativeTransform);
224   Pinca02_2->SetVisibility(true);
225   Pinca02_2->SetSimulatePhysics(false);
```

```
226   Pinca02_2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
227   Pinca02_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
228
229   ContatoPinca1_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Contato
          Pinca 1_2"));
230   ContatoPinca1_2->AttachToComponent(Pinca01_2,
          FAttachmentTransformRules::KeepRelativeTransform);
231   ContatoPinca1_2->SetVisibility(true);
232   ContatoPinca1_2->SetSimulatePhysics(false);
233   ContatoPinca1_2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
234   ContatoPinca1_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
235   ContatoPinca1_2->SetCollisionProfileName("Trigger");
236   ContatoPinca1_2->OnComponentBeginOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoContatoPinca1);
237   ContatoPinca1_2->OnComponentEndOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoSeparacaoPinca1);
238
239
240
241   ContatoPinca2_2 =
          CreateDefaultSubobject<UStaticMeshComponent>(TEXT("Contato
          Pinca 2_2"));
242   ContatoPinca2_2->AttachToComponent(Pinca02_2,
          FAttachmentTransformRules::KeepRelativeTransform);
243   ContatoPinca2_2->SetVisibility(true);
244   ContatoPinca2_2->SetSimulatePhysics(false);
245   ContatoPinca2_2->SetRelativeLocation(FVector(0.f, 0.f, 0.f));
246   ContatoPinca2_2->SetRelativeRotation(FRotator::MakeFromEuler(FVector(0.f,
          0.f, 0.f)));
247   ContatoPinca2_2->SetCollisionProfileName("Trigger");
248   ContatoPinca2_2->OnComponentBeginOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoContatoPinca2);
249   ContatoPinca2_2->OnComponentEndOverlap.AddDynamic(this,
          &AAtorGarra::ColisaoSeparacaoPinca2);
250
251 }
252
253 // Called when the game starts or when spawned
254 void AAtorGarra::BeginPlay()
255 {
256   Super::BeginPlay();
257
258   FVector PosicaoInicialAtor = FVector(-410.0f, 40.f, 90.f);
259   SetActorLocation(FVector(PosicaoInicialAtor));
260   SetActorRotation(FRotator::MakeFromEuler(FVector(0.f, 0.f,
          0.f)));
261
262   FVector PosicaoInicialArgola = FVector(-334.f, -5.f, 15.f);
```

```
263   FRotator AnguloInicialArgola =
          (FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f)));
264
265   Argola->SetWorldLocation(PosicaoInicialArgola);
266   Argola->SetWorldRotation(AnguloInicialArgola);
267 }
268
269 // Called every frame
270 void AAtorGarra::Tick(float DeltaTime)
271 {
272   Super::Tick(DeltaTime);
273
274
275
276 }
277
278 //Momimento eixo da garra
       ----------------------------------------------------------------------------
279
280 void AAtorGarra::SetupPlayerInputComponent(UInputComponent*
       PlayerInputComponent)
281 {
282
283 }
284
285 void AAtorGarra::DeslocamentoProfundidade(float EixoProfundidade)
286 {
287   if (EixoProfundidade != 0) {
288     FVector PosicaoAtualGarra =
            StaticMeshGarra->GetRelativeLocation();
289     FVector DeslocamentoX = FVector(0.f, 0.f, EixoProfundidade *
            SensibilidadeProfundidade);
290     FVector PosicaoFinalGarra = DeslocamentoX + PosicaoAtualGarra;
291
292     StaticMeshGarra->SetRelativeLocation(PosicaoFinalGarra);
293
294   };
295 }
296
297
298 void AAtorGarra::RotacaoXGarra(float AnguloX)
299 {
300
301   if (AnguloX != 0) {
302
303     FRotator AnguloAtualGarra =
            ReferenciaGarraDireita->GetRelativeRotation();
304     FRotator RotacaoHorizontal =
            FRotator::MakeFromEuler(FVector(AnguloX *
            SensibilidadeMouse, 0.f, 0.f));
305     FRotator RotacionadoHorizontalmente = AnguloAtualGarra +
            RotacaoHorizontal;
```

```
306     ReferenciaGarraDireita->SetRelativeRotation(RotacionadoHorizontalmente);
307
308
309
310   };
311
312 }
313
314 void AAtorGarra::RotacaoYGarra(float AnguloY)
315 {
316
317   if (AnguloY != 0) {
318
319     FRotator AnguloAtualGarra =
          ReferenciaGarraDireita->GetRelativeRotation();
320     FRotator RotacaoVertical =
          FRotator::MakeFromEuler(FVector(0.f, AnguloY *
          SensibilidadeMouse, 0.f));
321     FRotator RotacionadoVerticalmente = RotacaoVertical +
          AnguloAtualGarra;
322
323     ReferenciaGarraDireita->SetRelativeRotation(RotacionadoVerticalmente);
324
325
326   };
327
328 }
329 //-------------------------------------------------------------------------
330
331 //Movimento Camera
332 void AAtorGarra::DeslocamentoProfundidadeCamera(float
       EixoProfundidadeCamera)
333 {
334   if (EixoProfundidadeCamera != 0) {
335     FVector PosicaoAtualCamera = Camera->GetRelativeLocation();
336     FVector Deslocamento = FVector(EixoProfundidadeCamera *
          SensibilidadeProfundidade, 0.f, 0.f);
337     FVector PosicaoFinalGarra = Deslocamento + PosicaoAtualCamera;
338
339     Camera->SetRelativeLocation(PosicaoFinalGarra);
340   };
341 }
342
343 void AAtorGarra::DeslocamentoYCamera(float EixoY)
344 {
345   if (EixoY != 0) {
346     FVector PosicaoAtualCamera = Camera->GetRelativeLocation();
347     FVector Deslocamento = FVector(0.f, EixoY *
          SensibilidadeProfundidade, 0.f);
348     FVector PosicaoFinalGarra = Deslocamento + PosicaoAtualCamera;
349
350     Camera->SetRelativeLocation(PosicaoFinalGarra);
```

```
351   };
352 }
353
354 void AAtorGarra::DeslocamentoZCamera(float EixoZ)
355 {
356   if (EixoZ != 0) {
357     FVector PosicaoAtualCamera = Camera->GetRelativeLocation();
358     FVector Deslocamento = FVector(0.f, 0.f, EixoZ *
          SensibilidadeProfundidade);
359     FVector PosicaoFinalGarra = Deslocamento + PosicaoAtualCamera;
360
361     Camera->SetRelativeLocation(PosicaoFinalGarra);
362   };
363 }
364
365 void AAtorGarra::RotacaoXCamera(float AnguloX)
366 {
367
368   if (AnguloX != 0) {
369
370     FRotator AnguloAtualGarra = Camera->GetRelativeRotation();
371     FRotator RotacaoHorizontal =
          FRotator::MakeFromEuler(FVector(AnguloX *
          SensibilidadeCamera, 0.f, 0.f));
372     FRotator RotacionadoHorizontalmente = AnguloAtualGarra +
          RotacaoHorizontal;
373     Camera->SetRelativeRotation(RotacionadoHorizontalmente);
374     FRotator AnguloFinalGarra = Camera->GetRelativeRotation();
375
376     //UE_LOG(LogTemp, Warning, TEXT("***** %s \n"),
          *AnguloFinalGarra.ToString());
377
378
379   };
380
381 }
382
383 void AAtorGarra::RotacaoYCamera(float AnguloY)
384 {
385
386   if (AnguloY != 0) {
387
388     FRotator AnguloAtualGarra = Camera->GetRelativeRotation();
389     FRotator RotacaoHorizontal =
          FRotator::MakeFromEuler(FVector(0.f, AnguloY *
          SensibilidadeCamera, 0.f));
390     FRotator RotacionadoHorizontalmente = AnguloAtualGarra +
          RotacaoHorizontal;
391     Camera->SetRelativeRotation(RotacionadoHorizontalmente);
392     FRotator AnguloFinalGarra = Camera->GetRelativeRotation();
393
```

```
394     //UE_LOG(LogTemp, Warning, TEXT("***** %s \n"),
           *AnguloFinalGarra.ToString());
395
396
397   };
398
399 }
400
401 void AAtorGarra::RotacaoZCamera(float AnguloZ)
402 {
403
404   if (AnguloZ != 0) {
405
406     FRotator AnguloAtualGarra = Camera->GetRelativeRotation();
407     FRotator RotacaoHorizontal =
           FRotator::MakeFromEuler(FVector(0.f, 0.f, AnguloZ *
           SensibilidadeCamera));
408     FRotator RotacionadoHorizontalmente = AnguloAtualGarra +
           RotacaoHorizontal;
409     Camera->SetRelativeRotation(RotacionadoHorizontalmente);
410     FRotator AnguloFinalGarra = Camera->GetRelativeRotation();
411
412     //UE_LOG(LogTemp, Warning, TEXT("***** %s \n"),
           *AnguloFinalGarra.ToString());
413
414
415   };
416
417 }
418
419 //-----------------------------------------------------------------------
420
421
422 void AAtorGarra::RotacaoPontaGarra(float AnguloPontaGarra)
423 {
424
425   if (AnguloPontaGarra != 0) {
426
427     FRotator AnguloAtualPontaGarra =
           PontaGarra->GetRelativeRotation();
428     FRotator Rotacao = FRotator::MakeFromEuler(FVector(0.f, 0.f,
           AnguloPontaGarra * SensibilidadeRotacaoPontaGarra));
429     FRotator Rotacionado = Rotacao + AnguloAtualPontaGarra;
430
431     PontaGarra->SetRelativeRotation(Rotacionado);
432   };
433
434 }
435
436 void AAtorGarra::RotacaoApoioPinca(float AnguloApoioPinca)
437 {
438
```

```
439   if (AnguloApoioPinca != 0) {
440
441     FRotator AnguloAtualApoioPinca =
          ApoioPinca->GetRelativeRotation();
442     FRotator Rotacao =
          FRotator::MakeFromEuler(FVector(AnguloApoioPinca *
          SensibilidadeRotacaoPontaGarra, 0.f, 0.f));
443     FRotator Rotacionado = Rotacao + AnguloAtualApoioPinca;
444
445     if (Rotacionado.Roll < -75.f) {
446
447       Rotacionado.Roll = -75.f;
448
449     };
450
451     if (Rotacionado.Roll > 75.f) {
452
453       Rotacionado.Roll = 75.f;
454
455     };
456
457     ApoioPinca->SetRelativeRotation(Rotacionado);
458     FRotator AnguloFinalApoioPinca =
          ApoioPinca->GetRelativeRotation();
459
460
461     //UE_LOG(LogTemp, Warning, TEXT("***** %s \n"),
          *AnguloFinalApoioPinca.ToString());
462   };
463
464 }
465
466 void AAtorGarra::RotacaoPincasIndividuais(float
      AnguloPincasIndividuais)
467 {
468
469   if (AnguloPincasIndividuais == 1) {
470
471     if ((OverlapContato1 == 1) && (OverlapContato2 == 1)) {
472
473       Argola->DetachFromComponent(FDetachmentTransformRules::KeepWorldTransf
474       Argola->AttachToComponent(Referencia,
            FAttachmentTransformRules::KeepWorldTransform);
475       Argola->SetSimulatePhysics(true);
476       //UE_LOG(LogTemp, Error, TEXT("ENTROU NO DEATACH
            ----------------------------------------------------------------
            \n"));
477
478
479     };
480
481
```

```
482     FRotator AnguloAtualPinca1 =
           ReferenciaRotacaoPinca1->GetRelativeRotation();
483     FRotator AnguloAtualPinca2 =
           ReferenciaRotacaoPinca2->GetRelativeRotation();
484     FRotator Rotacao1 = FRotator::MakeFromEuler(FVector(0.f,
           (AnguloPincasIndividuais)*SensibilidadeAbrieEFecharGarras,
           0.f));
485
486     //if (OverlapContato1 == 1) {
487
488     //Rotacao1 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
489
490     //};
491
492     FRotator Rotacionado1 = Rotacao1 + AnguloAtualPinca1;
493
494     FRotator Rotacao2 = FRotator::MakeFromEuler(FVector(0.f,
           (-AnguloPincasIndividuais) *
           SensibilidadeAbrieEFecharGarras, 0.f));
495
496     //if (OverlapContato2 == 1) {
497
498     //Rotacao2 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
499
500     //};
501
502     FRotator Rotacionado2 = Rotacao2 + AnguloAtualPinca2;
503
504
505     if (Rotacionado1.Pitch > 45.f) {
506
507       Rotacionado1.Pitch = 45.f;
508
509     };
510
511     if (Rotacionado2.Pitch < -45.f) {
512
513       Rotacionado2.Pitch = -45.f;
514
515     };
516
517
518     ReferenciaRotacaoPinca1->SetRelativeRotation(Rotacionado1);
519     FRotator AnguloFinalPontaGarra1 =
           ReferenciaRotacaoPinca1->GetRelativeRotation();
520
521     ReferenciaRotacaoPinca2->SetRelativeRotation(Rotacionado2);
522     FRotator AnguloFinalPontaGarra2 =
           ReferenciaRotacaoPinca2->GetRelativeRotation();
523
524
```

```
525     //UE_LOG(LogTemp, Warning, TEXT("PINCA_01**: %s \n"),
            *AnguloFinalPontaGarra1.ToString());
526     //UE_LOG(LogTemp, Warning, TEXT("PINCA_02**: %s \n"),
            *AnguloFinalPontaGarra2.ToString());
527
528   };
529
530   if (AnguloPincasIndividuais == -1) {
531
532     FRotator AnguloAtualPinca1 =
            ReferenciaRotacaoPinca1->GetRelativeRotation();
533     FRotator AnguloAtualPinca2 =
            ReferenciaRotacaoPinca2->GetRelativeRotation();
534
535     FRotator Rotacao1 = FRotator::MakeFromEuler(FVector(0.f,
            (AnguloPincasIndividuais)*SensibilidadeAbrieEFecharGarras,
            0.f));
536
537     if (OverlapContato1 == 1) {
538
539       Rotacao1 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
540
541     };
542
543     FRotator Rotacionado1 = Rotacao1 + AnguloAtualPinca1;
544     FRotator Rotacao2 = FRotator::MakeFromEuler(FVector(0.f,
            (-AnguloPincasIndividuais) *
            SensibilidadeAbrieEFecharGarras, 0.f));
545
546     if (OverlapContato2 == 1) {
547
548       Rotacao2 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
549
550     };
551
552     FRotator Rotacionado2 = Rotacao2 + AnguloAtualPinca2;
553
554     if ((OverlapContato2 != 1)) {
555       if (Rotacionado1.Pitch < 0.f) {
556
557         Rotacionado1.Pitch = 0.f;
558
559       };
560     };
561
562     if ((OverlapContato1 != 1)) {
563       if (Rotacionado2.Pitch > 0.f) {
564
565         Rotacionado2.Pitch = 0.f;
566
567       };
568     };
```

```
569
570     ReferenciaRotacaoPinca1->SetRelativeRotation(Rotacionado1);
571     FRotator AnguloFinalPontaGarra1 =
           ReferenciaRotacaoPinca1->GetRelativeRotation();
572
573     ReferenciaRotacaoPinca2->SetRelativeRotation(Rotacionado2);
574     FRotator AnguloFinalPontaGarra2 =
           ReferenciaRotacaoPinca2->GetRelativeRotation();
575
576
577     //UE_LOG(LogTemp, Warning, TEXT("PINCA_01: %s \n"),
           *AnguloFinalPontaGarra1.ToString());
578     //UE_LOG(LogTemp, Warning, TEXT("PINCA_02: %s \n"),
           *AnguloFinalPontaGarra2.ToString());
579
580   };
581 }
582
583
584
585 //////////GARRA 2///////////
586
587 void AAtorGarra::DeslocamentoProfundidade_2(float EixoProfundidade)
588 {
589   if (EixoProfundidade != 0) {
590     FVector PosicaoAtualGarra =
           StaticMeshGarra_2->GetRelativeLocation();
591     FVector DeslocamentoX = FVector(0.f, 0.f, EixoProfundidade *
           SensibilidadeProfundidade);
592     FVector PosicaoFinalGarra = DeslocamentoX + PosicaoAtualGarra;
593
594     StaticMeshGarra_2->SetRelativeLocation(PosicaoFinalGarra);
595
596   };
597 }
598
599
600 void AAtorGarra::RotacaoXGarra_2(float AnguloX)
601 {
602
603   if (AnguloX != 0) {
604
605     FRotator AnguloAtualGarra =
           ReferenciaGarraEsquerda->GetRelativeRotation();
606     FRotator RotacaoHorizontal =
           FRotator::MakeFromEuler(FVector(AnguloX *
           SensibilidadeMouse, 0.f, 0.f));
607     FRotator RotacionadoHorizontalmente = AnguloAtualGarra +
           RotacaoHorizontal;
608     ReferenciaGarraEsquerda->SetRelativeRotation(RotacionadoHorizontalmente)
609
610
```

```
611
612   };
613
614 }
615
616 void AAtorGarra::RotacaoYGarra_2(float AnguloY)
617 {
618
619   if (AnguloY != 0) {
620
621     FRotator AnguloAtualGarra =
          ReferenciaGarraEsquerda->GetRelativeRotation();
622     FRotator RotacaoVertical =
          FRotator::MakeFromEuler(FVector(0.f, AnguloY *
          SensibilidadeMouse, 0.f));
623     FRotator RotacionadoVerticalmente = RotacaoVertical +
          AnguloAtualGarra;
624
625     ReferenciaGarraEsquerda->SetRelativeRotation(RotacionadoVerticalmente);
626
627
628   };
629
630 }
631
632 void AAtorGarra::RotacaoPontaGarra_2(float AnguloPontaGarra)
633 {
634
635   if (AnguloPontaGarra != 0) {
636
637     FRotator AnguloAtualPontaGarra =
          PontaGarra_2->GetRelativeRotation();
638     FRotator Rotacao = FRotator::MakeFromEuler(FVector(0.f, 0.f,
          AnguloPontaGarra * SensibilidadeRotacaoPontaGarra));
639     FRotator Rotacionado = Rotacao + AnguloAtualPontaGarra;
640
641     PontaGarra_2->SetRelativeRotation(Rotacionado);
642   };
643
644 }
645
646 void AAtorGarra::RotacaoApoioPinca_2(float AnguloApoioPinca)
647 {
648
649   if (AnguloApoioPinca != 0) {
650
651     FRotator AnguloAtualApoioPinca =
          ApoioPinca_2->GetRelativeRotation();
652     FRotator Rotacao =
          FRotator::MakeFromEuler(FVector(AnguloApoioPinca *
          SensibilidadeRotacaoPontaGarra, 0.f, 0.f));
653     FRotator Rotacionado = Rotacao + AnguloAtualApoioPinca;
```

```
654
655     if (Rotacionado.Roll < -75.f) {
656
657       Rotacionado.Roll = -75.f;
658
659     };
660
661     if (Rotacionado.Roll > 75.f) {
662
663       Rotacionado.Roll = 75.f;
664
665     };
666
667     ApoioPinca_2->SetRelativeRotation(Rotacionado);
668     FRotator AnguloFinalApoioPinca =
          ApoioPinca_2->GetRelativeRotation();
669
670
671     //UE_LOG(LogTemp, Warning, TEXT("***** %s \n"),
          *AnguloFinalApoioPinca.ToString());
672   };
673
674 }
675
676 void AAtorGarra::RotacaoPincasIndividuais_2(float
      AnguloPincasIndividuais)
677 {
678
679   if (AnguloPincasIndividuais == 1) {
680
681     if ((OverlapContato1_2 == 1) && (OverlapContato2_2 == 1)) {
682
683       Argola->DetachFromComponent(FDetachmentTransformRules::KeepWorldTransf
684       Argola->AttachToComponent(Referencia,
            FAttachmentTransformRules::KeepWorldTransform);
685       Argola->SetSimulatePhysics(true);
686       //UE_LOG(LogTemp, Error, TEXT("ENTROU NO DEATACH
            ----------------------------------------------------------------
            \n"));
687
688
689     };
690
691
692     FRotator AnguloAtualPinca1 =
          ReferenciaRotacaoPinca1_2->GetRelativeRotation();
693     FRotator AnguloAtualPinca2 =
          ReferenciaRotacaoPinca2_2->GetRelativeRotation();
694     FRotator Rotacao1 = FRotator::MakeFromEuler(FVector(0.f,
          (AnguloPincasIndividuais)*SensibilidadeAbrieEFecharGarras,
          0.f));
695
```

```
696     //if (OverlapContato1 == 1) {
697
698     //Rotacao1 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
699
700     //};
701
702     FRotator Rotacionado1 = Rotacao1 + AnguloAtualPinca1;
703
704     FRotator Rotacao2 = FRotator::MakeFromEuler(FVector(0.f,
            (-AnguloPincasIndividuais) *
            SensibilidadeAbrieEFecharGarras, 0.f));
705
706     //if (OverlapContato2 == 1) {
707
708     //Rotacao2 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
709
710     //};
711
712     FRotator Rotacionado2 = Rotacao2 + AnguloAtualPinca2;
713
714
715     if (Rotacionado1.Pitch > 45.f) {
716
717       Rotacionado1.Pitch = 45.f;
718
719     };
720
721     if (Rotacionado2.Pitch < -45.f) {
722
723       Rotacionado2.Pitch = -45.f;
724
725     };
726
727
728     ReferenciaRotacaoPinca1_2->SetRelativeRotation(Rotacionado1);
729     FRotator AnguloFinalPontaGarra1 =
            ReferenciaRotacaoPinca1_2->GetRelativeRotation();
730
731     ReferenciaRotacaoPinca2_2->SetRelativeRotation(Rotacionado2);
732     FRotator AnguloFinalPontaGarra2 =
            ReferenciaRotacaoPinca2_2->GetRelativeRotation();
733
734
735     //UE_LOG(LogTemp, Warning, TEXT("PINCA_01**: %s \n"),
            *AnguloFinalPontaGarra1.ToString());
736     //UE_LOG(LogTemp, Warning, TEXT("PINCA_02**: %s \n"),
            *AnguloFinalPontaGarra2.ToString());
737
738   };
739
740   if (AnguloPincasIndividuais == -1) {
741
```

```
742     FRotator AnguloAtualPinca1 =
           ReferenciaRotacaoPinca1_2->GetRelativeRotation();
743     FRotator AnguloAtualPinca2 =
           ReferenciaRotacaoPinca2_2->GetRelativeRotation();
744
745     FRotator Rotacao1 = FRotator::MakeFromEuler(FVector(0.f,
           (AnguloPincasIndividuais)*SensibilidadeAbrieEFecharGarras,
           0.f));
746
747     if (OverlapContato1_2 == 1) {
748
749       Rotacao1 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
750
751     };
752
753     FRotator Rotacionado1 = Rotacao1 + AnguloAtualPinca1;
754     FRotator Rotacao2 = FRotator::MakeFromEuler(FVector(0.f,
           (-AnguloPincasIndividuais) *
           SensibilidadeAbrieEFecharGarras, 0.f));
755
756     if (OverlapContato2_2 == 1) {
757
758       Rotacao2 = FRotator::MakeFromEuler(FVector(0.f, 0.f, 0.f));
759
760     };
761
762     FRotator Rotacionado2 = Rotacao2 + AnguloAtualPinca2;
763
764     if ((OverlapContato2_2 != 1)) {
765       if (Rotacionado1.Pitch < 0.f) {
766
767         Rotacionado1.Pitch = 0.f;
768
769       };
770     };
771
772     if ((OverlapContato1_2 != 1)) {
773       if (Rotacionado2.Pitch > 0.f) {
774
775         Rotacionado2.Pitch = 0.f;
776
777       };
778     };
779
780     ReferenciaRotacaoPinca1_2->SetRelativeRotation(Rotacionado1);
781     FRotator AnguloFinalPontaGarra1 =
           ReferenciaRotacaoPinca1_2->GetRelativeRotation();
782
783     ReferenciaRotacaoPinca2_2->SetRelativeRotation(Rotacionado2);
784     FRotator AnguloFinalPontaGarra2 =
           ReferenciaRotacaoPinca2_2->GetRelativeRotation();
785
```

```
786
787     //UE_LOG(LogTemp, Warning, TEXT("PINCA_01: %s \n"),
            *AnguloFinalPontaGarra1.ToString());
788     //UE_LOG(LogTemp, Warning, TEXT("PINCA_02: %s \n"),
            *AnguloFinalPontaGarra2.ToString());
789
790   };
791 }
792 //-------------------------------------------------------------------------
793
794 //Contato Garra Direita
795
796 void AAtorGarra::ColisaoContatoPinca1(UPrimitiveComponent*
      OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*
      OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
      FHitResult& SweepResult)
797 {
798
799   if ((OtherComp == Argola))
800   {
801     OverlapContato1 = 1;
802
803     if ((OverlapContato1 == 1) && (OverlapContato2 == 1)) {
804
805       FVector PosicaoAtualArgola = Argola->GetComponentLocation();
806       FRotator AnguloAtualArgola = Argola->GetComponentRotation();
807       FVector ScalaAtualArgola = Argola->GetComponentScale();
808
809       Argola->AttachToComponent(ContatoPinca1,
            FAttachmentTransformRules::KeepRelativeTransform);
810       Argola->SetSimulatePhysics(false);
811       Argola->SetWorldLocation(PosicaoAtualArgola);
812       Argola->SetWorldRotation(AnguloAtualArgola);
813       Argola->SetWorldScale3D(ScalaAtualArgola);
814       //UE_LOG(LogTemp, Error, TEXT("AS DUAS PINCAS ESTAO
            TOCANDO"));
815     }
816
817     //UE_LOG(LogTemp, Warning, TEXT("OCORREU O CONTATO COM A
          PIN A 1:  %f \n"), OverlapContato1);
818   }
819
820 }
821
822 void AAtorGarra::ColisaoContatoPinca2(UPrimitiveComponent*
      OverlappedComponent, AActor* OtherActor, UPrimitiveComponent*
      OtherComp, int32 OtherBodyIndex, bool bFromSweep, const
      FHitResult& SweepResult)
823 {
824
825   if ((OtherComp == Argola))
826   {
```

```
827     OverlapContato2 = 1;
828 
829     if ((OverlapContato1 == 1) && (OverlapContato2 == 1)) {
830 
831       FVector PosicaoAtualArgola = Argola->GetComponentLocation();
832       FRotator AnguloAtualArgola = Argola->GetComponentRotation();
833       FVector ScalaAtualArgola = Argola->GetComponentScale();
834 
835       Argola->AttachToComponent(ContatoPinca1,
             FAttachmentTransformRules::KeepRelativeTransform);
836       Argola->SetSimulatePhysics(false);
837       Argola->SetWorldLocation(PosicaoAtualArgola);
838       Argola->SetWorldRotation(AnguloAtualArgola);
839       Argola->SetWorldScale3D(ScalaAtualArgola);
840       //UE_LOG(LogTemp, Error, TEXT("AS DUAS PINCAS ESTAO
             TOCANDO"));
841 
842     }
843 
844     //UE_LOG(LogTemp, Warning, TEXT("OCORREU O CONTATO COM A
          PIN A 2: %f \n"), OverlapContato2);
845   }
846 
847 }
848 
849 void AAtorGarra::ColisaoSeparacaoPinca1(UPrimitiveComponent*
    OverlappedComp, AActor* OtherActor, UPrimitiveComponent*
    OtherComp, int32 OtherBodyIndex)
850 {
851 
852 
853 
854 
855   if ((OtherComp == Argola)) {
856 
857     OverlapContato1 = 0;
858     //Argola->AttachToComponent(Referencia,
          FAttachmentTransformRules::KeepRelativeTransform);
859 
860     //UE_LOG(LogTemp, Warning, TEXT("CONTATO 1 SEPAROU DA ARGOLA:
          %f \n"), OverlapContato1);
861   }
862 }
863 
864 void AAtorGarra::ColisaoSeparacaoPinca2(UPrimitiveComponent*
    OverlappedComp, AActor* OtherActor, UPrimitiveComponent*
    OtherComp, int32 OtherBodyIndex)
865 {
866 
867 
868   if ((OtherComp == Argola)) {
869 
```

```
870     OverlapContato2 = 0;
871     //Argola->AttachToComponent(Referencia,
           FAttachmentTransformRules::KeepRelativeTransform);
872     //UE_LOG(LogTemp, Warning, TEXT("CONTATO 2 SEPAROU DA ARGOLA:
           %f \n"), OverlapContato1);
873   }
874 }
875
876 int AAtorGarra::MovimentoCorpoGarra(FVector VetorPosicaoCubo,
       FVector VetorPosicaoReferencia)
877 {
878
879   float DistanciaProdundidadeNova =
         FVector::Distance(VetorPosicaoCubo, VetorPosicaoReferencia);
880
881   float DeltaDistancias = DistanciaProfundidadeAntiga -
         DistanciaProdundidadeNova;
882
883   DistanciaProfundidadeAntiga = DistanciaProdundidadeNova;
884
885   if (DeltaDistancias > 0) {
886     return 1;
887   }
888
889   else if (DeltaDistancias < 0) {
890     return -1;
891   }
892
893   else return 0;
894
895
896 }
897
898
899 //-------------------------------------------------------------------------
900
901 //Contato Garra Esquerda
902
903 //PRECISO ARRUMAR O CONTATO DA GARRA ESQUERDA
```