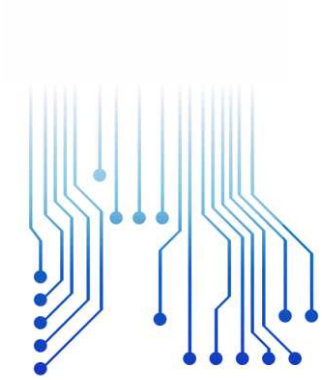
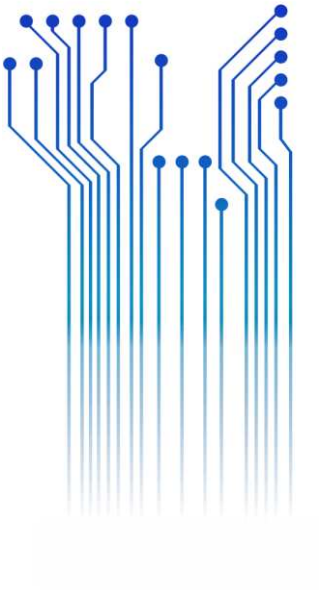


Curso de Graduação em Engenharia Elétrica

Trabalho de Conclusão de Curso

Desenvolvimento de Plataforma Virtual 3D para Realização de Ensaios em Transformadores

Natã de Macêdo Silva



Campina Grande - PB

Junho de 2024

Natã de Macêdo Silva

Desenvolvimento de Plataforma Virtual 3D para Realização de Ensaios em Transformadores

Trabalho de Conclusão de Curso submetido à Coordenação do Curso de Graduação em Engenharia Elétrica da Universidade Federal de Campina Grande como parte dos requisitos necessários para a obtenção do grau de Bacharel em Ciências no Domínio da Engenharia Elétrica.

Área de Concentração: Eletrotécnica

Edson Guedes da Costa, D.Sc.
Orientador

Campina Grande, PB
Junho de 2024

Natã de Macêdo Silva

Desenvolvimento de Plataforma Virtual 3D para Realização de Ensaios em Transformadores

Aprovado em _____ /_____ /_____

_____

Prof ° Jalberth Fernandes de Araújo, D.Sc.
Universidade Federal de Campina Grande
Avaliador

_____

Prof ° Edson Guedes da Costa, D.Sc.
Universidade Federal de Campina Grande
Orientador

Campina Grande, PB
Junho de 2024

Dedico este trabalho à minha mãe, Joelma de Macêdo Silva.

AGRADECIMENTOS

Agradeço, primeiramente, à minha mãe Joelma, mulher guerreira que sempre me incentivou a buscar nos estudos e no trabalho, uma oportunidade de satisfação própria e crescimento pessoal.

À minha noiva, Juanne, que sempre acreditou em mim e no meu potencial. Que me alegrou quanto estive triste, e que me motivou quando estive sem ânimo, ofereceu amor e carinho e hoje constitui um pilar essencial em minha vida.

A meu primo, Adson, que com certeza foi uma grande inspiração, com seus significativos conhecimentos em tecnologias.

À minha família residente em Campina Grande, em especial aos meus tios, Ancilete e Edson, que ofereceram todo apoio necessário para que eu pudesse me estabilizar na cidade e fazer a graduação.

À família da minha noiva, em especial ao meu sogro Mercus e minha sogra Solange, que me acolheram com todo carinho, e me apoiaram durante minha jornada acadêmica.

Aos meus colegas de curso, em especial Daniel, por todas as brincadeiras, anedotas intelectuais e trocadilhos, que tornaram a graduação bem mais leve e divertida.

Ao professor Edson, que me orientou e guiou, durante a elaboração deste trabalho.

E ao excelente curso de Engenharia Elétrica de graduação oferecido na UFCG, que apesar de bastante complicado, me possibilitou ver o mundo de uma forma mais crítica, me trouxe conhecimentos extremamente valiosos e me tornou sobretudo um bom “empreendedor” para a vida.

“Sem o elemento do prazer, não vale a pena tentar se destacar em nada”

Magnus Carlsen.

RESUMO

O curso de Engenharia Elétrica na UFCG inclui a disciplina de Equipamentos Elétricos, que possui parte teórica e a parte laboratorial. Esta disciplina aborda diversos equipamentos, como disjuntores e transformadores, essenciais para a formação de engenheiros. Dentre os diversos experimentos, o de transformadores de potência destaca-se, por ser o mais longo e com maior quantidade de ensaios realizados. A partir disto, o objetivo deste estudo é desenvolver para o corpo discente do curso de Engenharia Elétrica, uma plataforma virtual que possibilite não só a interação do aluno com um transformador de potência, assim como também a realização de simulação de ensaios de "tipo" e de "rotina". Para isso, utilizou-se o *software* Unity 3D como base, pois possui características de facilidade no desenvolvimento de estruturas, objetos e mapas 3D. A criação da plataforma virtual abrangeu a modelagem de um cenário similar ao laboratório físico em que o experimento era realizado, assim como do transformador, personagem, instrumentos de medição, e demais objetos gráficos necessários para simular a realização dos ensaios. O desenvolvimento possibilitou realização de ensaios como determinação de polaridade, ensaio de levantamento de curva de saturação, medição da resistência de isolamento, medição da relação de transformação, operação em curto-circuito e operação em vazio. Além disso o procedimento de inspeção visual foi implementado oferecendo a opção ao usuário de visualizar o transformador ensaiado, com seus respectivos dados de placa, e também um modelo de transformador sem a carcaça possibilitando a identificação das partes internas. Assim, o desenvolvimento do ambiente virtual dentro da Unity 3D, conseguiu representar de maneira realista o equipamento elétrico, a sala onde os experimentos eram realizados, e os principais instrumentos empregados na realização dos diferentes tipos de ensaios abordados. Com isso, foi disponibilizada a comunidade acadêmica uma plataforma que permite acesso a um laboratório com diversas ferramentas que possibilitam o aprofundamento nos estudos sobre transformadores, sem restrições de tempo, possível de ser acessada remotamente e que visa oferecer acessibilidade aos seus usuários que não possuem acesso ao laboratório físico.

Palavras-chave: Plataforma virtual, laboratório, equipamentos elétricos.

ABSTRACT

The Electrical Engineering course at UFCG includes the subject of Electrical Equipment, which has a theoretical part and a laboratory part. This subject covers various equipment, such as circuit breakers and transformers, essential for the training of engineers. Among the various experiments, the one on power transformers stands out, as it is the longest and has the largest number of tests carried out. Based on this, the objective of this study is to develop for the student body of the Electrical Engineering course, a virtual platform that allows not only the student to interact with a power transformer, but also to carry out "type" test simulations. and "routine". For this, Unity 3D software was used as a base, as it has characteristics that make it easy to develop structures, objects and 3D maps. The creation of the virtual platform included modeling a scenario similar to the physical laboratory in which the experiment was carried out, as well as the transformer, character, measuring instruments, and other graphic objects necessary to simulate the tests. The development made it possible to carry out tests such as polarity determination, saturation curve survey testing, insulation resistance measurement, transformation ratio measurement, short-circuit operation and no-load operation. Furthermore, the visual inspection procedure was implemented offering the user the option of viewing the tested transformer, with its respective plate data, and also a transformer model without the housing, enabling the identification of the internal parts. Thus, the development of the virtual environment within Unity 3D managed to realistically represent the electrical equipment, the room where the experiments were carried out, and the main instruments used to carry out the different types of tests covered. With this, a platform was made available to the academic community that allows access to a laboratory with various tools that enable in-depth studies on transformers, without time restrictions, possible to be accessed remotely and which aims to offer accessibility to its users who do not have access to the physical laboratory.

Keywords: Virtual platform, laboratory, electrical equipment**.**

LISTA DE FIGURAS

SUMÁRIO

1. INTRODUÇÃO

O curso de Engenharia Elétrica (EE) na Universidade Federal de Campina Grande (UFCG), possui em sua grade curricular a disciplina de Equipamentos Elétricos, que contempla carga horária de 75 h, sendo elas divididas entre a disciplina teórica com 60 h e o laboratório integrado com 15 h, período considerado curto para a realização de todos os experimentos.

Diversos equipamentos são abordados ao longo da disciplina, tanto teoricamente, quanto em experimentos laboratoriais, como, disjuntores, transformadores de instrumentos, transformadores de potência, para-raios, chaves seccionadoras e entre outros, que agregam conhecimentos e experiência imprescindíveis para a formação profissional de engenheiros, principalmente na ênfase de Eletrotécnica.

Dentre os diversos experimentos, o de transformadores de potência destaca-se, por ser o mais longo e com maior quantidade de ensaios realizados. A partir disto, visando uma melhor compreensão de todo o experimento, e uma maior flexibilidade de tempo em sua realização, faz-se necessário o desenvolvimento de uma plataforma virtual, possibilitando a realização dos ensaios de maneira remota. A plataforma virtual pode servir como ferramenta complementar, com finalidade de preparação para o experimento presencial, agregando maior aprendizado, assim como também pode ser útil para alunos de cursos de Engenharia Elétrica de outras instituições que não possuam acesso ao laboratório. Em conclusão, agregando de maneira positiva na formação acadêmica dos alunos da formação acadêmica em Engenharia Elétrica.

Considerando que as atividades e os experimentos do curso de Equipamentos Elétricos demandam tempo, foi pretendido direcionar os estudos no equipamento mais importante de uma subestação, o transformador de potência.

Transformadores de potência são equipamentos elétricos essenciais para a viabilidade e o funcionamento do sistema elétrico em corrente alternada, pois possibilita adequação aos níveis de tensão e corrente, de acordo com a necessidade e aplicação. Segundo Frontin (2013), um transformador é um equipamento destinado a transmitir energia elétrica ou potência elétrica de um circuito a outro, transformando tensões e correntes em um circuito de corrente alternada, ou a modificar os valores das impedâncias de um circuito elétrico.

Conforme Gervásio (2021) antes de deixar a linha de produção, os transformadores são rigorosamente testados e submetidos a vários tipos de ensaios. Para certificação do equipamento, faz-se necessário a realização de ensaios, podendo ser considerados ensaios de “tipo” e de “rotina”. Os ensaios de rotina contemplam os ensaios de resistência elétrica dos enrolamentos, relação de tensões, resistência do isolamento, polaridade, deslocamento angular

e sequência de fases, perdas em vazio e em carga, corrente de excitação, inerência de curto-circuito e ensaios dielétricos. Os ensaios de tipo que são os de elevação de temperatura, tensão suportável nominal de impulso atmosférico, nível de ruído, ensaios no óleo isolante, entre outros (ABNT 5356, 1993).

Diante disso, o presente estudo tem como finalidade desenvolver um ambiente virtual, em software, capaz de simular ensaios em transformadores de potência, permitindo assim, o seu uso como laboratório virtual de Equipamentos Elétricos, isto é, possibilitando a emulação de alguns ensaios de tipo e de rotina, otimizando a aprendizagem do assunto da disciplina. Assim, a plataforma virtual possibilitará aos discentes da disciplina de Equipamentos Elétricos do curso de Engenharia Elétrica da UFCG assim como também de outras instituições, a realização dos experimentos e emulações dos ensaios de forma virtual, como preparação e a consolidação dos conceitos envolvidos nos experimentos.

1.1 Objetivos

1.1.1 Objetivo Geral

O objetivo geral deste trabalho é desenvolver uma plataforma virtual 3D para realização de ensaios em transformadores, que possibilite a interação do aluno da disciplina de Equipamentos Elétricos da Universidade Federal de Campina Grande, com um transformador de potência emulando os ensaios de “tipo” e de “rotina".

1.1.2 Objetivos Específicos

Para alcançar o objetivo geral, faz-se necessário o cumprimento de etapas como:

1. Desenvolver a interface gráfica em modelo 3D do laboratório, transformador com instrumentos de medição;
2. Desenvolver virtualmente procedimento de inspeção visual;
3. Desenvolver virtualmente o ensaio de polaridade;
4. Desenvolver virtualmente ensaio de saturação para determinação de curva λmax versus Imax;
5. Desenvolver virtualmente ensaio de corrente em vazio;
6. Desenvolver virtualmente ensaio de relação de transformação;
7. Desenvolver virtualmente ensaio de operação em curto-circuito;
8. Desenvolver virtualmente ensaio de medição da resistência de isolamento;

1.2 Estrutura do Relatório

O presente estudo é composto de cinco capítulos que descrevem a trajetória detalhada deste trabalho. Assim, a organização está definida por:

- Capítulo 1: Introdução sobre o estudo e os objetivos desenvolvidos para a realização deste trabalho;
- Capítulo 2: Embasamento teórico a respeito de transformadores, os tipos de ensaios realizados nestes equipamentos, e as tecnologias utilizadas no desenvolvimento da plataforma virtual.
- Capítulo 3: Metodologia explicando os procedimentos computacionais utilizados no desenvolvimento da plataforma;
- Capítulo 4: Apresentação dos resultados obtidos no desenvolvimento e na execução do projeto, assim como das dificuldades encontradas;
- Capítulo 5: Considerações finais acerca da temática apresentada.

2. FUNDAMENTAÇÃO TEÓRICA

2.1 Transformadores

Com a descoberta da eletricidade, a possibilidade de transmitir eletricidade a longas distâncias foi considerada crucial. Os avanços tecnológicos na geração, transmissão e utilização final de energia permitem atualmente transformar áreas desabitadas ou subdesenvolvidas em polos industriais e grandes centros urbanos. Este processo é viabilizado com o uso de transformadores para realizar a adequação dos níveis de tensão e corrente de um circuito para outro, com baixas perdas

Do ponto de vista da segurança humana, é necessário a utilização de níveis baixos de tensão durante a entrega aos consumidores. Por outro lado, visando alcançar a alta capacidade de potência com perdas relativamente baixas, durante a transmissão de energia elétrica por meio de longas distâncias dos geradores até os centros consumidores, a utilização de altos níveis de tensão é mais adequada.

O transformador é um equipamento elétrico que funciona a partir da ação de um campo magnético e capaz de converter entre seus terminais de nível de tensão e corrente elétrica, para um outro nível, mantendo a potência elétrica durante esta conversão. Devido a sua grande importância, é possível afirmar que estes são os principais equipamentos encontrados nas subestações, sendo assim, a realização de análises, ensaios e manutenções são essenciais para evitar paradas não programadas, que podem causar diversos prejuízos.

2.2 Ensaios em Transformadores

Os ensaios realizados nos transformadores asseguram que o equipamento possa suportar as condições de operação previstas ao longo do tempo, da vida estimada, ajudando a prevenir interrupções no fornecimento de energia, prolongando a vida útil do transformador e reduzindo custos de manutenção. Os ensaios podem ser divididos em ensaios de “tipo” e ensaios de “rotina”. Ambos são essenciais para garantir que os transformadores operem de maneira segura e eficiente ao longo de sua vida útil.

Os ensaios de “rotina”, são realizados em todos os transformadores fabricados prontos para entrega, objetivando verificar se o equipamento específico está em condições adequadas de funcionamento e em conformidade com as especificações, para assegurar a qualidade consistente da produção

Diferentes ensaios são realizados nos transformadores antes de serem conectados ao sistema elétrico e são considerados ensaios de rotina, como:

1. Resistência dos Enrolamentos;
2. Relação de Tensões;
3. Verificação da Resistência de Isolamento;
4. Polaridade;
5. Deslocamento Angular;
6. Sequência de Fases;
7. Perdas em Vazio e Corrente de Excitação;
8. Perdas em Carga e Corrente de Curto-Circuito.

Por outro lado, os ensaios de "tipo" são realizados para validar o projeto do transformador e suas características, a fim de garantir que eles atendam aos requisitos especificados pelas normas técnicas. Estes ensaios são geralmente conduzidos em um protótipo ou em uma unidade de produção inicial. Os ensaios de tipo são mais criteriosos que os de ensaios de rotina e são listados abaixo:

1. Tensão Suportável à Frequência Industrial
2. Tensão Induzida;
3. Descargas Parciais;
4. Tensão Nominal Suportável de Impulso Atmosférico;
5. Impulso de Manobra;
6. Estanqueidade e Resistência à Pressão Interna e Estanqueidade a Quente;
7. Elevação de Temperatura.

2.3 Tecnologias utilizadas

2.3.1 Unity

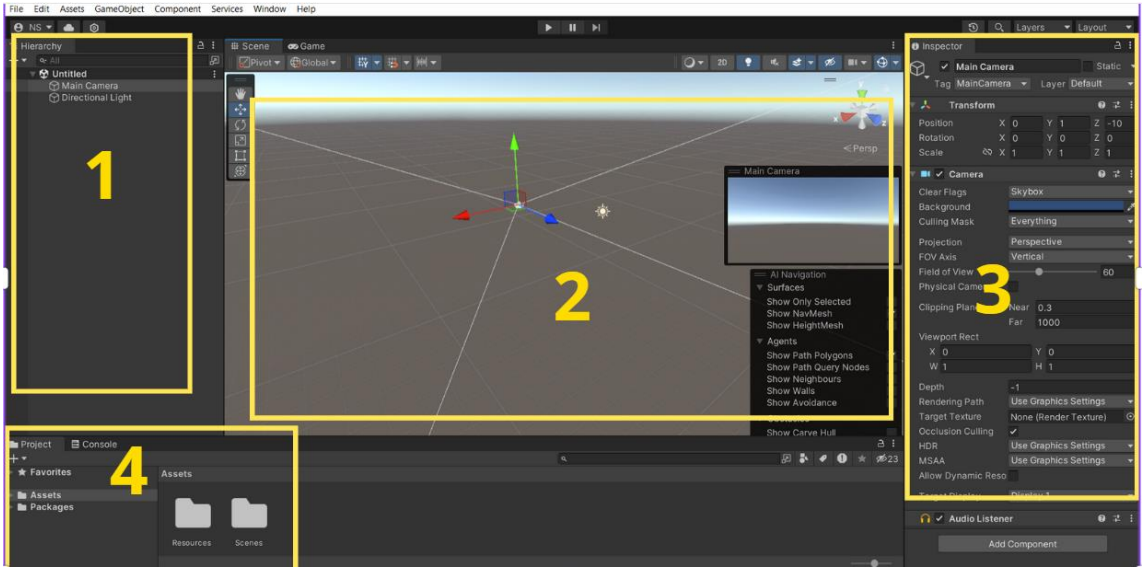
A Unity é uma plataforma de desenvolvimento de jogos amplamente utilizada que oferece um ambiente integrado para a criação de jogos 2D e 3D, aplicativos interativos, simuladores e outras aplicações. Lançada pela Unity Technologies, é conhecida por sua acessibilidade e versatilidade, permitindo que tanto iniciantes, quanto profissionais, desenvolvam projetos de alta qualidade. O Unity Editor, interface gráfica intuitiva da plataforma, facilita a criação e manipulação de objetos dentro de uma cena, oferecendo

ferramentas para importar e gerenciar ativos, configurar cenas, programar comportamentos e testar jogos em tempo real.

A Unity possui diversas ferramentas e telas para as mais diferentes funcionalidades. Na Figura 1, é mostrada tela inicial da Unity no formato padrão, nela em destaque são mostradas as abas/janelas mais utilizadas.

A janela 1 é chamada de *Hierarchy*, onde são armazenados todos os objetos que pertencem ao jogo, como por exemplo câmeras, iluminação, objetos 3D criados pelo usuário ou importados etc. Na Janela 2, chamada *Scene*, ocorre a alocação dos objetos que compõem o jogo, nela é feita a disposição dos objetos gráficos em um espaço tridimensional para posterior execução. Nesta mesma janela, há outra aba nomeada como *game*, onde se mostra o resultado do jogo após sua compilação e durante sua execução. Na janela 3, conhecida como *Inspector* são realizadas as configurações básicas dos componentes pertencentes aos objetos selecionados na janela *Hierarchy*. Onde são controladas as lógicas de funcionamento dos objetos, rotinas computacionais, características gráficas dos objetos entre diversas outras funcionalidades. Na Janela 4, são encontradas duas abas, a aba *project* que contém as pastas e arquivos que fazem parte do jogo, e a aba *console*, onde os resultados de compilação dos códigos, erros e avisos são mostrados.

Figura 1. Tela inicial padrão da Unity.



Fonte: Autor (2024).

A Unity possui como fator de destaque, o fato de possuir suporte multiplataforma, permitindo que os desenvolvedores criem um jogo uma vez e o exportem para diversas plataformas, como PC, consoles, dispositivos móveis, web e realidade virtual/aumentada. Isso

amplia o alcance potencial dos jogos e reduz o tempo e esforço necessário para lançá-los em múltiplos sistemas.

Além disso, a possibilidade de realizar importação de ativos, ou seja, recursos gráficos como por exemplo objetos, texturas e materiais, torna a Unity mais versátil. Por meio da AssetStore que é um marketplace oficial da Unity, os usuários podem comprar e vender ativos nomeados de *assets*, acelerando o desenvolvimento de projetos e fomentando uma rica base de recursos e tutoriais disponíveis para todos os níveis de habilidade.

A principal linguagem de programação suportada pela Unity é o C#, que permite definir o comportamento dos objetos no jogo, interações do usuário, física, inteligência artificial, entre outros. A Unity utiliza o motor PhysX para simulações de física em 3D, proporcionando um comportamento realista dos objetos. Além disso, a plataforma apresenta um motor gráfico poderoso que suporta a renderização em tempo real de gráficos 2D e 3D, incluindo recursos como iluminação, partículas e pós-processamento, garantindo visuais interessantes e desempenho otimizado.

2.3.2 C#

A linguagem *C Sharp*, comumente representada como C# é uma linguagem de programação moderna, orientada a objetos e de propósito geral, desenvolvida pela Microsoft como parte de sua plataforma .NET. Desde sua criação, C# se destacou por sua simplicidade, robustez e versatilidade, sendo amplamente utilizada no desenvolvimento de aplicações desktop, web, móveis e, especialmente, jogos eletrônicos.

C# é uma linguagem fortemente tipada, o que significa que o tipo de cada variável deve ser declarado explicitamente, proporcionando maior segurança e redução de erros em tempo de compilação. A linguagem suporta conceitos avançados de programação orientada a objetos, como herança, polimorfismo e encapsulamento, além de oferecer recursos modernos como LINQ (Language Integrated Query), delegados, eventos, e async/await para programação assíncrona. C# também possui uma sintaxe clara e intuitiva, facilitando a aprendizagem e a escrita de código eficiente e legível.

As vantagens de utilizar C# são inúmeras. Primeiramente, a integração com a plataforma .NET permite acesso a uma vasta biblioteca de classes e métodos, acelerando o desenvolvimento de aplicações complexas. C# é altamente eficiente em termos de performance e oferece excelente suporte para multithreading, permitindo a execução de múltiplas tarefas simultaneamente. A linguagem também possui um forte suporte da comunidade e uma ampla

documentação, facilitando a resolução de problemas e o aprendizado contínuo. Além disso, o uso de *garbage collection* automatiza a gestão de memória, reduzindo a incidência de *bugs* relacionados a vazamentos de memória.

Por outro lado, C# apresenta algumas desvantagens. A principal delas é a dependência da plataforma .NET, que, embora seja multiplataforma, ainda possui limitações em comparação com outras linguagens que não possuem tais dependências. Além disso, a curva de aprendizado pode ser íngreme para iniciantes, especialmente aqueles sem experiência prévia em linguagens orientadas a objetos. O gerenciamento automatizado de memória, apesar de suas vantagens, pode conduzir a problemas de desempenho em aplicações críticas de tempo real se não for utilizado corretamente.

A integração de C# com a Unity é um dos aspectos mais poderosos e atrativos para desenvolvedores de jogos. Unity utiliza C# como sua principal linguagem de *script*, permitindo que os desenvolvedores criem lógica de jogo, controladores de comportamento, e interações de usuário de forma eficiente. A combinação de C# com Unity oferece um ambiente de desenvolvimento robusto, onde é possível criar desde jogos simples até projetos complexos e imersivos. Assim, Unity fornece um editor intuitivo e visual, onde *scripts* C# podem ser facilmente anexados a objetos de jogo, facilitando a implementação de funcionalidades e a realização de testes em tempo real.

2.3.3 Visual Studio Code

Visual Studio Code também conhecido como VS Code é um editor de código-fonte desenvolvido pela Microsoft que se tornou uma das ferramentas preferidas de desenvolvedores ao redor do mundo. Lançado em 2015, VS Code é um editor gratuito e de código aberto, disponível para diversas plataformas, incluindo Windows, macOS e Linux. Uma das características mais notáveis do VS Code é sua leveza e rapidez, o que o torna uma excelente escolha tanto para projetos pequenos quanto para grandes aplicações. Sua interface moderna e intuitiva facilita a navegação e a edição de código, oferecendo uma experiência de desenvolvimento agradável e eficiente.

Uma das maiores vantagens do Visual Studio Code é sua extensibilidade. Através de sua vasta biblioteca de extensões, desenvolvedores podem adicionar funcionalidades específicas para praticamente qualquer linguagem de programação ou *framework*. Extensões para Python, JavaScript, C#, Java, e muitas outras linguagens estão facilmente disponíveis no marketplace integrado. Além disso, o VS Code oferece suporte integrado para controle de

versão com Git, permitindo que os desenvolvedores gerenciem repositórios diretamente do editor. A integração com plataformas como GitHub e Azure DevOps também facilita o trabalho colaborativo e a integração contínua.

Outro ponto forte do VS Code é o seu robusto conjunto de ferramentas de depuração. O editor permite que desenvolvedores inspecionem variáveis, configurem pontos de interrupção, e executem códigos passo a passo, tudo dentro do próprio ambiente de desenvolvimento. Essas ferramentas são essenciais para identificar e corrigir erros de maneira eficiente, aumentando a produtividade e a qualidade do código. A possibilidade de depurar diretamente no editor, combinada com a integração com várias linguagens e plataformas, faz do VS Code uma escolha poderosa para desenvolvedores que buscam um ambiente de desenvolvimento integrado (IDE) leve, mas altamente funcional.

Por fim, o Visual Studio Code também se destaca pelo suporte ao desenvolvimento remoto e pelo trabalho colaborativo em tempo real. Recursos como o Live Share permitem que múltiplos desenvolvedores trabalhem no mesmo projeto simultaneamente, compartilhando seu ambiente de desenvolvimento e colaborando como se estivessem na mesma sala. Esse recurso é particularmente útil em tempos de trabalho remoto, permitindo que equipes distribuídas mantenham uma colaboração eficaz. Além disso, o suporte para desenvolvimento remoto permite que desenvolvedores trabalhem em máquinas virtuais, contêineres Docker, ou servidores remotos diretamente do VS Code, proporcionando uma flexibilidade sem precedentes no desenvolvimento de software.

3. METODOLOGIA

Para o desenvolvimento da plataforma virtual e sua interface gráfica, utilizou-se o *software* Unity 3D como base, por possuir características de facilidade de programação e desenvolvimento de estruturas, objetos e mapas 3D. Além disso, pelo fato de se tratar de um *game engine,* ou seja, um motor de jogo, possui diversas funcionalidades implementadas, voltadas para simulação de fenômenos físicos por exemplo como gravidade, colisões etc.

Na loja da unity 3D chamada *AssetStore* (https://assetstore.unity.com/) assim como em outros bancos imagens e arquivos 3D como Warehouse 3D (https://3dwarehouse.sketchup.com/) serão importadas texturas e objetos pré-dimensionados, disponibilizados pela plataforma ou por outros usuários, os quais serão utilizados quando possível no processo de implementação do cenário do laboratório. Objetos mais específicos como transformadores e instrumentos de medição, serão desenvolvidos dentro do próprio Unity 3D, com auxílio de editores de imagens como por exemplo o programa Blender e o Corel Draw com a finalidade de criar modelos mais elaborados.

A lógica dos experimentos, cálculos de variáveis, e estrutura do projeto, serão desenvolvidos com auxílio do software Visual Studio Code, responsáveis pelo processo de codificação, depuração e compilação das rotinas necessárias para o funcionamento da plataforma pretendida

Dentro do visual Studio, será utilizada a linguagem de programação C#, para elaboração dos códigos e rotinas computacionais associadas ao funcionamento dos ensaios e estrutura da plataforma virtual, tendo em vista que se trata de uma linguagem robusta e compatível para interação com o Unity 3D.

Para simular de maneira mais adequada a realização dos ensaios no transformador, utilizou-se valores típicos de variáveis, obtidos no experimento do período corrente da turma de equipamentos elétricos, assim como dados dos períodos passados, quando disponíveis, na implementação das rotinas, a fim de definir faixas de valores possíveis como resultado para cada uma das medições nos ensaios.

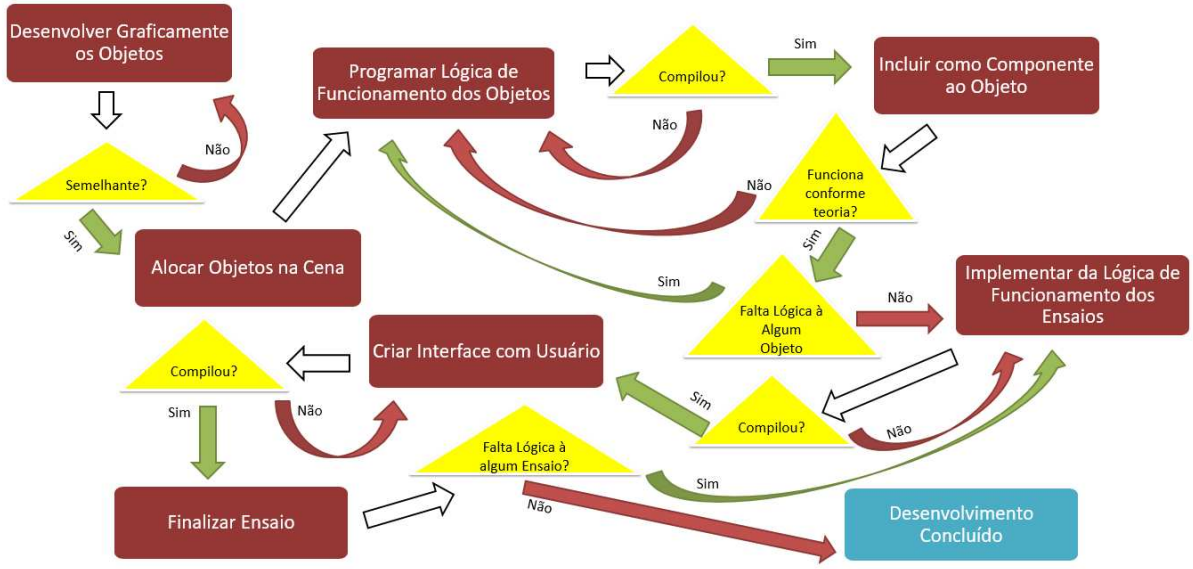
Na Figura 2, mostra-se o fluxograma para desenvolvimento da plataforma virtual pretendida. Para correto funcionamento o fluxograma é dividido em etapas. A primeira etapa consiste em desenvolver graficamente os objetos. Para isso usa-se *softwares* de modelagem 3D como por exemplo o Blender, afim de criar modelos mais complexos para os objetos da plataforma. Através do Mixamo o avatar será implementado, assim como suas animações. Para importação de ativos para o jogo como por exemplo texturas e materiais, utiliza-se a loja

AssetStore. Alguns objetos poderão ser importados diretamente de banco de imagens e modelos 3D como por exemplo o 3D *Warehouse.*

A segunda etapa consiste em alocar os objetos na cena, sendo assim todos os objetos modelados serão configurados e colocados em proporção para fazerem parte do ambiente criado. Com os objetos devidamente alocados, inicia-se a etapa de oferecer lógica aos objetos, através de *scripts* na linguagem C#, serão implementadas rotinas computacionais que representem bem os equipamentos e instrumentos utilizados na plataforma. Quando estas rotinas e códigos funcionam corretamente, são incluídas aos objetos para então iniciar a fase de testes das novas funcionalidades implementadas.

Com todos os objetos alocados no canário e suas devidas lógicas sido implementadas. Inicia-se o processo de implantação da lógica de funcionamento dos ensaios, aplicando-se restrições ao uso incorreto dos equipamentos e instrumentos, aplicando-se valores para as grandezas elétricas conforme observados no laboratório. Validado o funcionamento coerente dos ensaios, a interface com o usuário se faz um ponto crucial, pois possibilitará que o usuário possa controlar o personagem, os instrumentos e os equipamentos através de um computador, por meio de seus terminais (teclado e mouse). Assim poderá ser realizada a conclusão dos ensaios, e a partir disto o desenvolvimento da plataforma será efetuado.

Figura 2. Fluxograma para desenvolvimento da plataforma.



Fonte: Autor (2024).

4. RESULTADOS E DISCUSSÃO

4.1 Desenvolvimento da plataforma

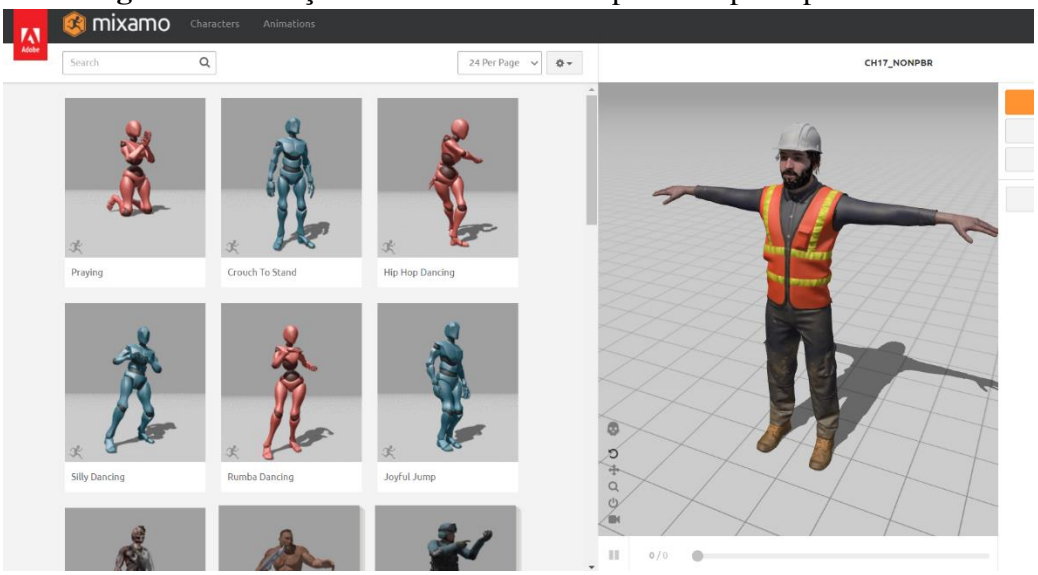
Durante o processo de desenvolvimento da plataforma diversas escolhas foram tomadas. Foi planejada a criação de um ambiente similar ao ambiente físico, em que o experimento de transformadores da disciplina de Equipamentos Elétricos da UFCG disponibiliza. Fotografias foram registradas, e medições *in loco* foram realizadas visando alocar virtualmente os objetos de uma maneira mais fiel a realidade, em termos de proporção e localização espacial.

Além disso, a plataforma objetiva que no processo de realização dos ensaios seja possível que o usuário faça a ligação dos circuitos de maneira simples e intuitiva com animações 3D associadas. Como resultado dos ensaios obteve-se valores, resultados e proporções análogas aos ensaios feitos presencialmente na disciplina de Equipamentos Elétricos.

4.1.1 Implementação do Personagem

A primeira etapa consistiu na importação para a unity de um avatar para simular o aluno ou engenheiro que realiza o ensaio no equipamento elétrico. Para isso foi utilizado a plataforma Mixamo que disponibiliza diversos modelos de personagens, além de animações que podem ser utilizadas. Buscou-se um personagem com características que remetessem a um engenheiro, com os devidos EPIs (equipamentos de proteção individual). Foi feito o download do avatar escolhido no formato (.fbx) que pode ser lido pela Unity em diferentes animações, que remetessem a possíveis movimentos esperados do avatar, como por exemplo andar, ficar parado, pegar um fio, colocar um fio, e manusear um objeto. Na Figura 3 é possível observar o avatar escolhido para compor a plataforma.

Figura 3. Ilustração do avatar escolhido para compor a plataforma.

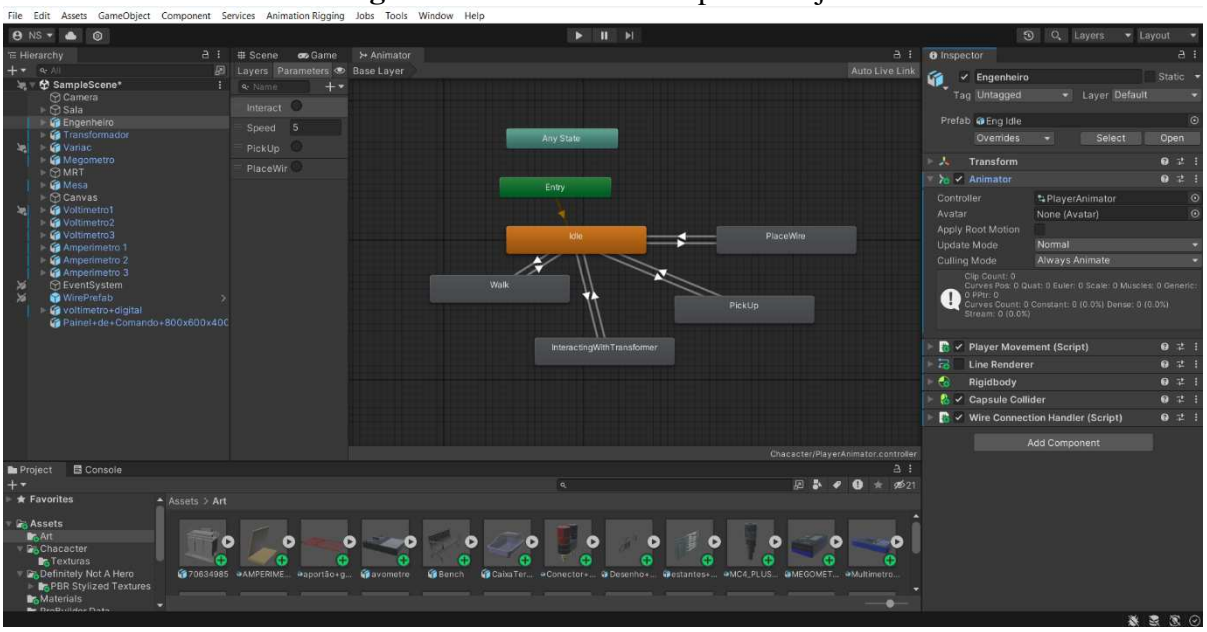


Fonte: Autor (2024).

Sendo importado para a Unity, foi criado um objeto na aba *Hierarchy,* nomeado Engenheiro, e a este foi adicionado um componente *Animator* para possibilitar a utilização de animação do personagem. A partir disto foi realizada a configuração da animação do personagem na janela de animação da Unity, utilizando variáveis tipo *trigger* e *float*, posteriormente utilizadas para controlar o personagem via *script*.

Para o processo de animação foi necessário criar diferentes estados para o objeto, um para cada tipo de animação desejada. Na Figura 4 se mostram os estados *Idle* que deve ficar ativo quando o personagem não receber comandos do usuário, *walk* usado para representar o avatar andando, *InteractingWithTransformer* representando a interação do personagem com o transformador, *PickUp* com a animação de pegar um objeto e *PlaceWire* para animação de colocar um fio. A partir disto foi possível a lógica de ativação para as transações de estado de acordo com as variáveis criadas anteriormente.

Figura 4. Diversos estados para o objeto.



Fonte: Autor (2024).

Em Anexo A é possível identificar o *script* em C# utilizado para controlar o avatar.

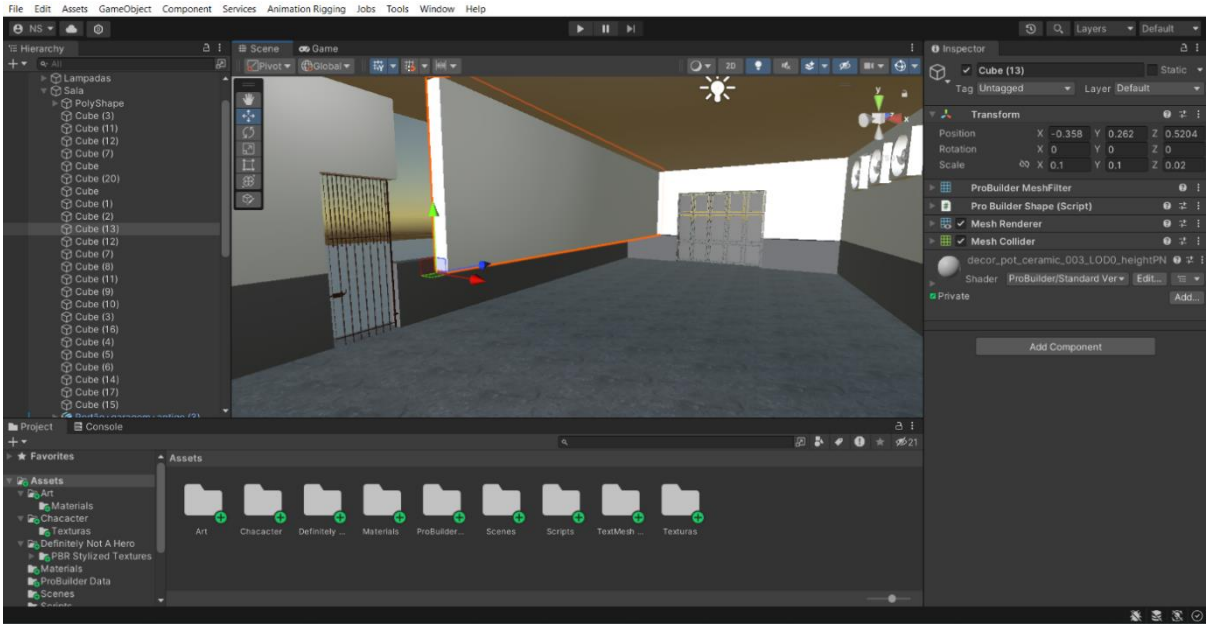
4.1.2 Implementação de Cenário

A implementação de cenário considerou a estrutura da sala onde os experimentos de transformadores eram realizados no Laboratório de Equipamentos Elétricos da UFCG.

Para criar a sala do laboratório virtual, utilizou-se a ferramenta ProBuilder da Unity, uma poderosa extensão que permite a modelagem e edição de geometria diretamente dentro do ambiente de desenvolvimento. Com o ProBuilder, foi possível esculpir a estrutura do laboratório de maneira eficiente, utilizando formas básicas como cubos e planos. Essas formas foram manipuladas para definir as paredes, o chão e o teto da sala, permitindo um controle preciso sobre as dimensões e a disposição dos elementos. Para os objetos criados dentro da Unity, a importação de texturas e materiais ocorreram por meio da *AssetStore*. Na Figura 5 mostra-se a alocação de um dos objetos cúbicos para composição da parede da sala de experimentos.

Os demais objetos como portões, mesas e prateleiras, por exemplo, foram importados de site de modelos 3D, como, por exemplo, o Warehouse 3D, e modificados dentro da Unity, para se adequar aos tamanhos e dimensões adequados.

Figura 5. Exemplo de alocação de um objeto que compõe a parede da sala de experimentos.



Fonte: Autor (2024).

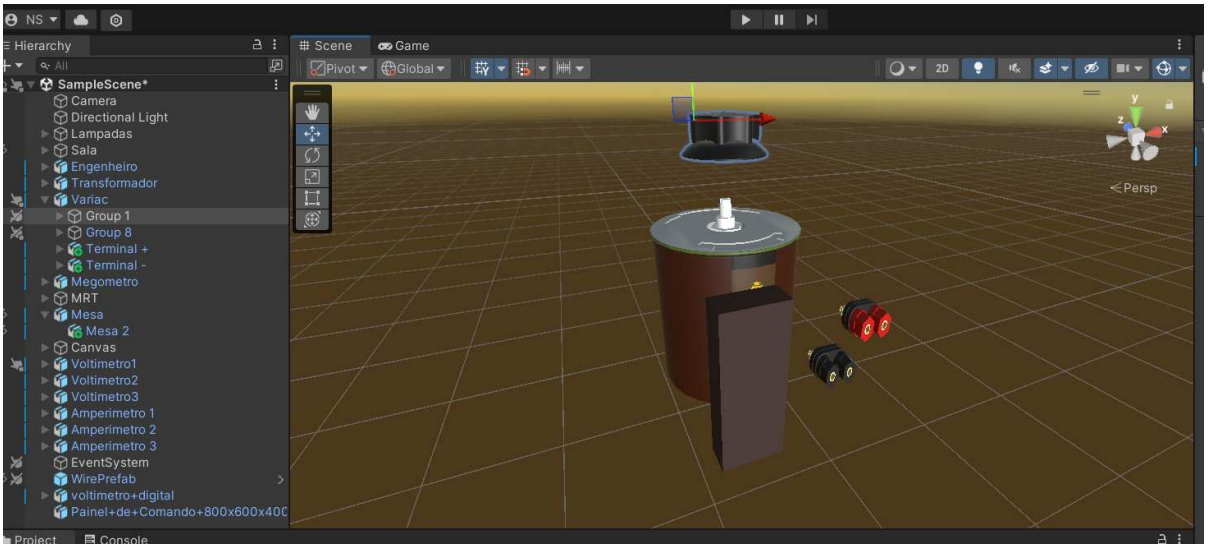
4.1.3 Implementação de Equipamentos e Instrumentos

Para a realização dos ensaios, é crucial que a plataforma possua objetos que representem os instrumentos de medição, como por exemplos, amperímetros, voltímetros, MRT, a fonte de tensão variável e principalmente, o equipamento que vai ser ensaiado, que é o transformador.

Alguns instrumentos de medição, como por exemplos, o multímetro e megômetro, além do modelo do transformador, foram obtidos a partir de banco de imagens e objetos 3D, como por exemplo, o Warehouse 3D e importados para o projeto. Demais instrumentos tiveram que ser ajustados, montados e até criados totalmente com auxílio da ferramenta ProBuilder. Na Figura 6, é possível observar o exemplo de um instrumento montado parcialmente na unity, utilizando diferentes objetos em sua composição.

Tendo em vista que o objetivo da plataforma é realizar ensaios com os transformadores, para o processo de ligação elétrica entre os instrumentos e o equipamento, foi implementado em todos os instrumentos, terminais de conexão, onde os cabos farão a conexão elétrica de maneiras variadas, a depender do ensaio que o transformador é submetido.

Figura 6. Exemplo de um instrumento utilizando diferentes objetos em sua composição.



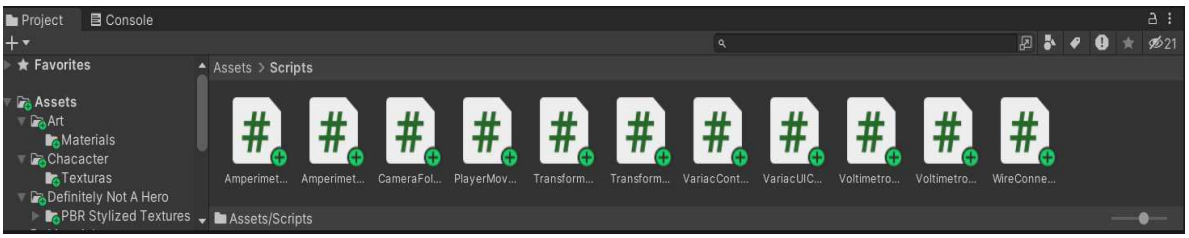
Fonte: Autor (2024).

4.1.4 Elaboração de *Scripts*

As lógicas de funcionamento dos elementos da plataforma essencialmente são definidas por *scripts*, ou seja, um arquivo de código que define um conjunto de instruções a serem executadas. Esses *scripts* são utilizados para adicionar comportamentos, interações e lógica aos objetos dentro da aplicação. Em essência, um *script* pode ser visto como um "comando" que dita como um objeto deve se comportar em resposta a eventos ou mudanças no ambiente. Estes códigos comandam desde a parte gráfica da plataforma, até a realização de cálculos de variáveis relacionadas as grandezas elétricas associadas aos ensaios.

Com o objetivo de realizar os diversos ensaios abordados pela plataforma, e proporcionar uma abordagem 3D interativa e dinâmica, foi necessário criar diversos *scripts*. Estes *scripts* foram desenvolvidos para controlar tanto o personagem quanto o ajuste da câmera, além de definir o comportamento dos diferentes instrumentos e equipamentos elétricos. A pasta com alguns *scripts* desenvolvidos para a plataforma pode ser visualizada na Figura 7.

Figura 7. Pasta com alguns exemplos de *scripts* desenvolvidos para a plataforma.



Fonte: Autor (2024).

Cada um dos códigos possui um conjunto de funções específicas e são obrigatórios para o funcionamento da plataforma. É possível citar alguns dos principais *scripts* gerados para a plataforma, como por exemplo os códigos:

Player Movement: Como a plataforma busca oferecer uma imersão na realização dos ensaios, foi objetivado que o personagem conseguisse se mover dentro do ambiente da sala de experimentos. Então foi programado o *script* com a finalidade de propiciar ao usuário a movimentação do personagem de maneira livre dentro do cenário, condicionado a interagir com os objetos ou obstáculos, a fim de detectar a colisão com objetos e não os atravessar. Além disso, o *script* é responsável por todas as animações que o personagem possui, desde andar até interagir com os equipamentos, garantindo uma movimentação suave e responsiva. A movimentação do personagem ocorre através do clique em qualquer uma das teclas W movimentando personagem para frente, A movimentando para esquerda, S para trás ou D para a direita. No anexo A, mostra-se o *script Player Movement* desenvolvido na linguagem C# e adicionado como componente ao objeto Engenheiro.

Camera Follow: Na unity para possibilitar a visualização enquanto o projeto é executado, é necessário o uso de câmeras. Foi planejado que a câmera deveria seguir a posição do personagem do jogo à medida que ele se move pelo ambiente. Além disso, para oferecer uma maior visualização por parte do usuário, também foi implementada a funcionalidade de a câmera rotacionar o campo de visão em torno do personagem, seguindo a orientação relativa do ponteiro do mouse na tela. O *script Camera Follow* desenvolvido foi adicionado como componente ao objeto *Camera.*

Wire Connection Handler: A plataforma permite que o usuário monte circuitos para ensaios no transformador, manuseando os cabos que interligam o circuito. O usuário pode realizar conexões nos terminais elétricos do transformador, variac e instrumentos através de cliques com o botão esquerdo do mouse. Ao clicar em um terminal, inicia-se a criação e renderização de um novo cabo que segue a posição do personagem em tempo real até ser conectado no terminal de destino com um segundo clique. Caso a conexão não seja desejada, o cabo pode ser excluído com a tecla K. Os cabos gerados simulam o efeito da gravidade com uma curva parabólica, e o código identifica os terminais conectados, fornecendo a lógica das conexões elétricas para os demais *scripts*. No anexo B, está o *script Wire Connection Handler* programado e adicionado ao objeto Engenheiro.

Transformer Controller: O equipamento ensaiado contempla um *script* responsável pelos cálculos das grandezas elétricas associadas ao transformador, e as interações destas

grandezas com os demais instrumentos disponíveis na plataforma. Este script foi utilizado como componente para objeto Transformador.

Variac Controller: Os ensaios realizados necessitam de uma fonte de tensão variável, portanto foi criado um *script* capaz de simular o funcionamento de um variac. Para manter a noção do giro para alterar os níveis de tensão, tal como são encontrados nos variac reais em suas manipulas, adotou-se que a tensão do variac deveria ser ajustada com o giro do *scroll* (botão de rolagem) do mouse.

Voltimetro Controller: Define terminais de um multímetro para simular a leitura de tensões, comandando a lógica de funcionamento simplificada de um voltímetro ideal e definindo as interações dos demais equipamentos. As lógicas de funcionamento dos instrumentos de medição possuem similaridades entre si, e suas diferenciações ocorrem no princípio de cálculo das variáveis internas e tipos de interação possíveis com os demais instrumentos ou equipamentos. Exemplificando estes *scripts,* no Anexo C demonstra-se o código gerado para comandar os multímetros na função voltímetro.

Amperimetro Controller: Configura os terminais de um multímetro para simular a leitura de correntes, comandando a lógica de funcionamento simplificada de um amperímetro ideal e definindo as interações dos demais equipamentos.

Megometro Controller: Acrescenta o objeto megômetro, a lógica do princípio básico de funcionamento do instrumento, definindo seus terminais para a realização de leitura de resistências no transformador.

MRT Controller: Comanda a lógica de funcionamento de um MRT, para medir relação de transformação de cada uma das fases do transformador.

Além disso, para representar valores das grandezas elétricas atualizadas na tela da plataforma, *scripts* foram criados. Estes foram nomeados como:

- *Variac UI Controller*: Mostra tensão do variac, ativado ou desativado com clique na tecla X*;*
- *Amperimetro UI Controller*: Mostra leitura dos multímetros configurados como amperímetros, ativado ou desativado com clique na tecla C;
- *Voltímetro UI controller:* Mostra leitura dos multímetros configurados como voltímetros, ativado ou desativado com clique na tecla V;
- *Wattimetro UI controller*: Mostra leitura dos wattímetros, ativado ou desativado com clique na tecla B;

- *Megometro UI Controller:* Mostra leitura do megômetro, ativado com clique na tecla M;
- *MRT controller*: Mostra a leitura do instrumento medidor de relação de transformação, ativado ou desativado com clique na tecla N.

Na Figura 8, pode-se observar as teclas do teclado e os botões do mouse que podem ser utilizadas pelo usuário para interagir com a plataforma.

Figura 8. Teclas do teclado e botões do mouse que podem ser utilizadas pelo usuário na plataforma.

Fonte: Autor (2024).

4.2 Execução do projeto

A realização deste trabalho de conclusão de curso envolveu o desenvolvimento de uma plataforma 3D interativa, projetada para simular ensaios em transformadores. A seguir, são apresentados os resultados obtidos durante a execução do projeto, abrangendo desde a verificação da parte gráfica do ambiente virtual, implementação dos *scripts* até a validação das funcionalidades da plataforma.

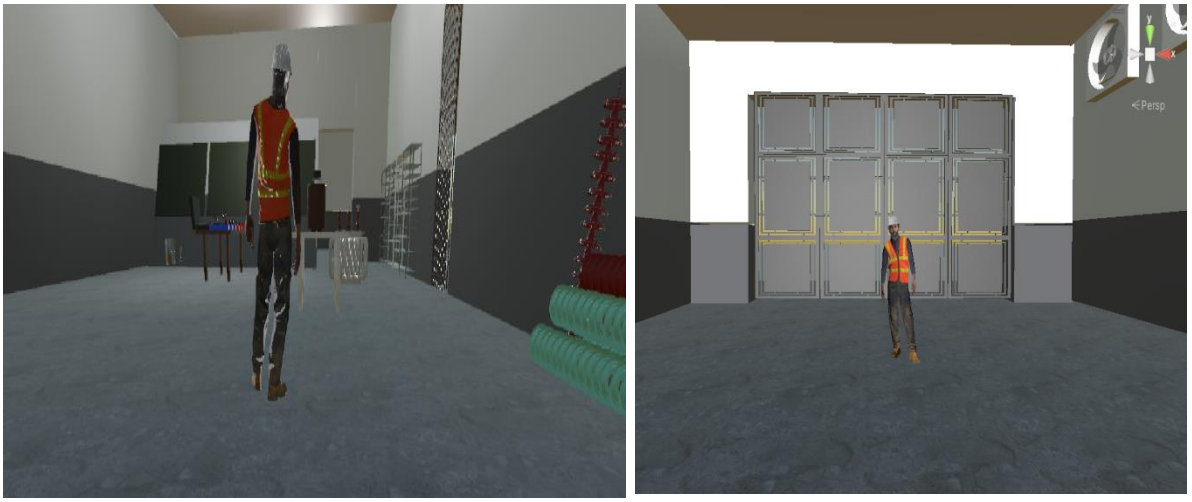
4.2.1 Ambiente Virtual

O primeiro resultado significativo foi a implementação bem-sucedida da plataforma 3D, que integra todos os componentes necessários para a simulação de ensaios em transformadores. Utilizando o Unity, foi possível criar um ambiente virtual que replica com precisão os elementos de um laboratório de ensaios elétricos. Na Figura 9, mostra-se fotografias realizadas *in loco* no laboratório, e na Figura 10, é mostrada vistas da tela da plataforma que busca replicar os principais detalhes da sala do laboratório.

Figura 9. Fotografias realizadas *in loco* no laboratório da UFCG.

Fonte: Autor (2024).

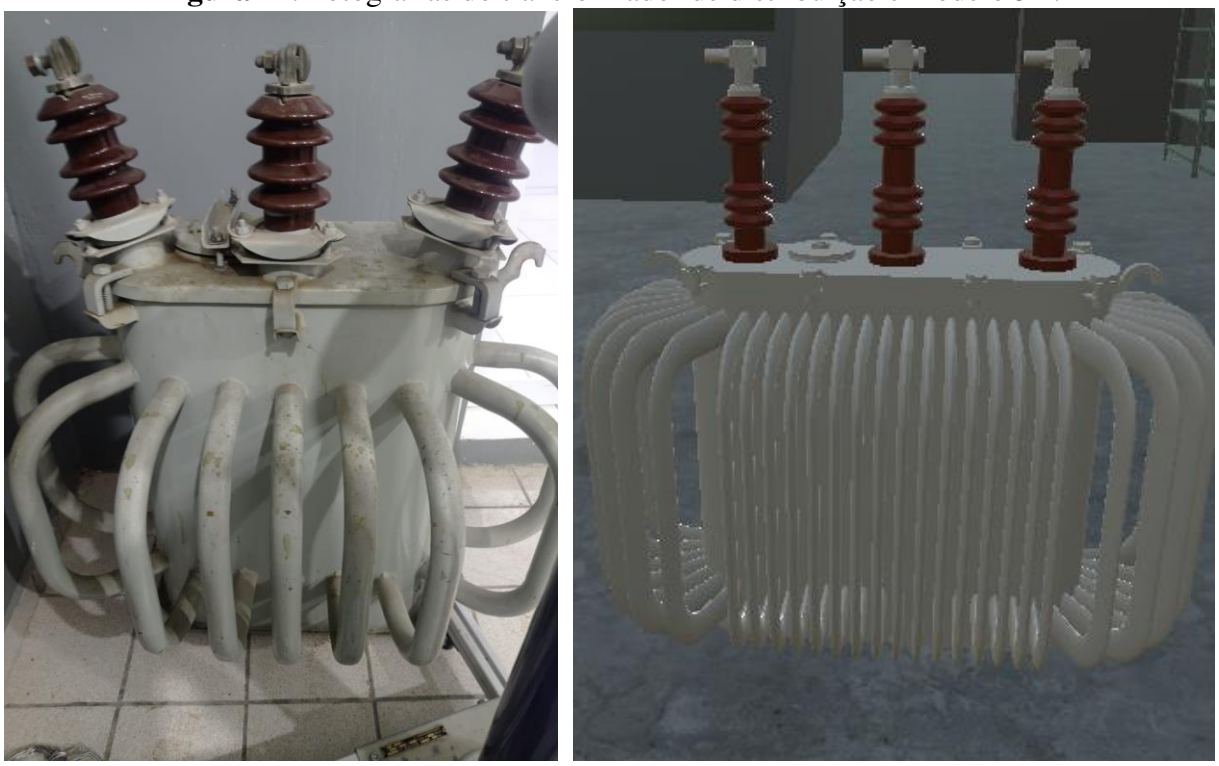
Figura 10. Captura de tela da plataforma 3D desenvolvida.



Fonte: Autor (2024).

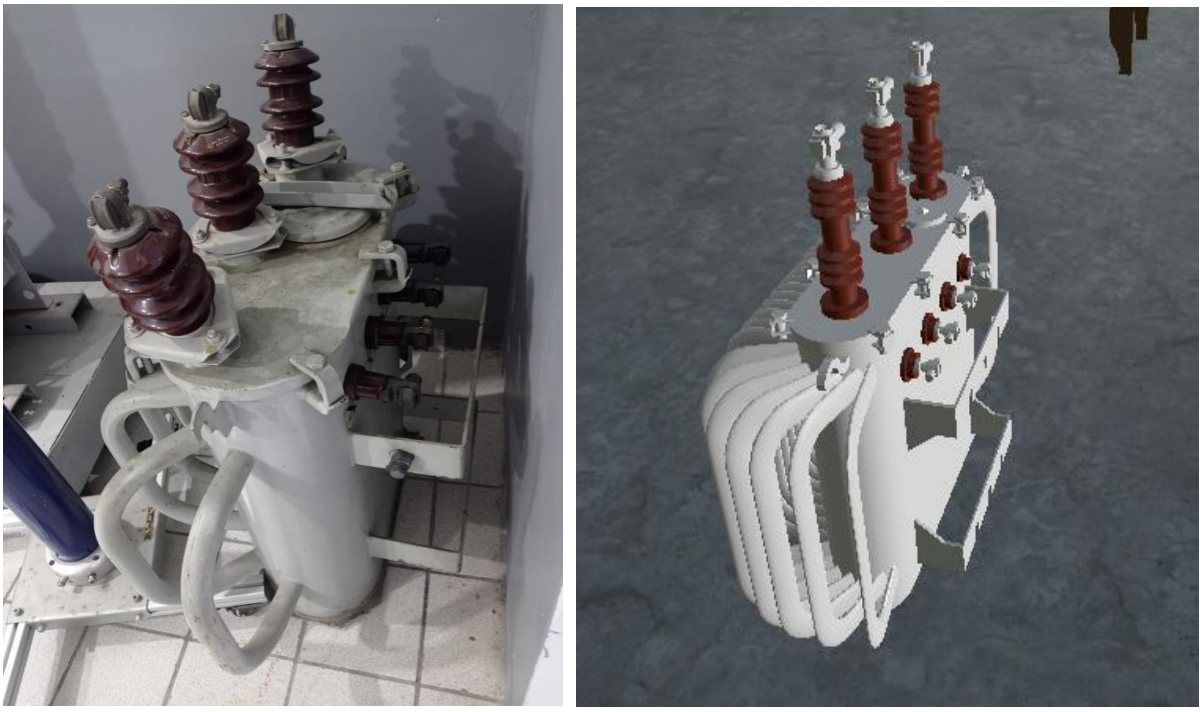
O transformador utilizado na plataforma também replica o transformador utilizado para realização dos ensaios na UFCG. Na Figura 11, mostra-se fotografias do transformador de distribuição de 15kVA com tensões de 0.38/13.8 kV e seu respectivo modelo 3D, e na Figura 12 é possível observar o transformador real e o modelo 3D gerado para a utilização na plataforma.

Figura 11. Fotografias do transformador de distribuição e modelo 3D.



Fonte: Autor (2024).

Figura 12. Transformador real e modelo 3D gerado para a utilização na plataforma.



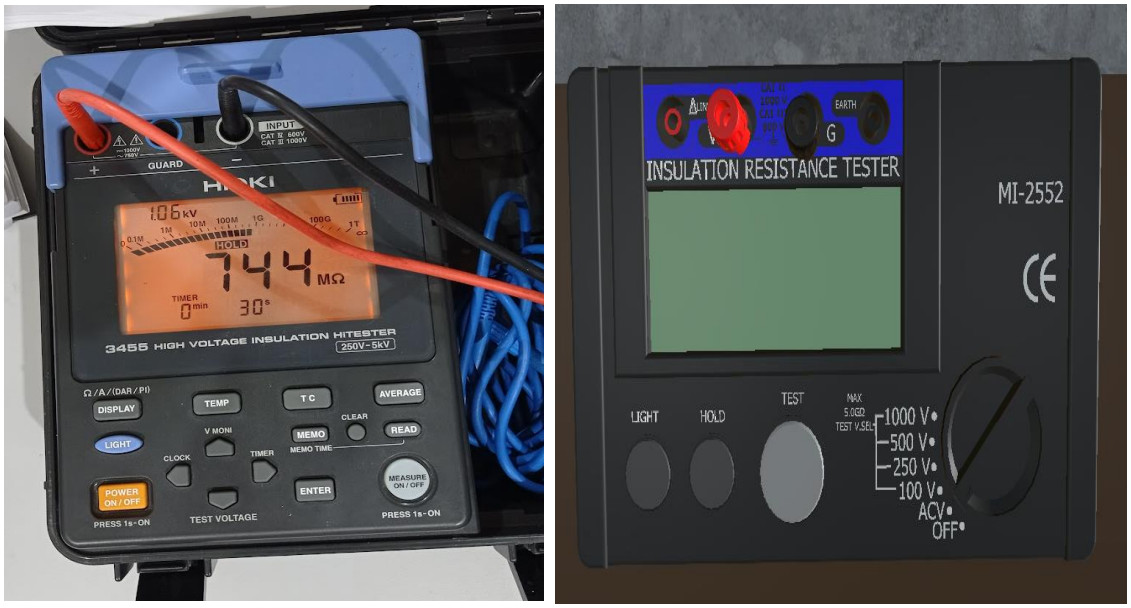
Fonte: Autor (2024).

Além disso, os instrumentos de medição implementados na plataforma basearam-se nos modelos reais utilizados.

Figura 13. Multímetros reais e modelos 3D de multímetros.

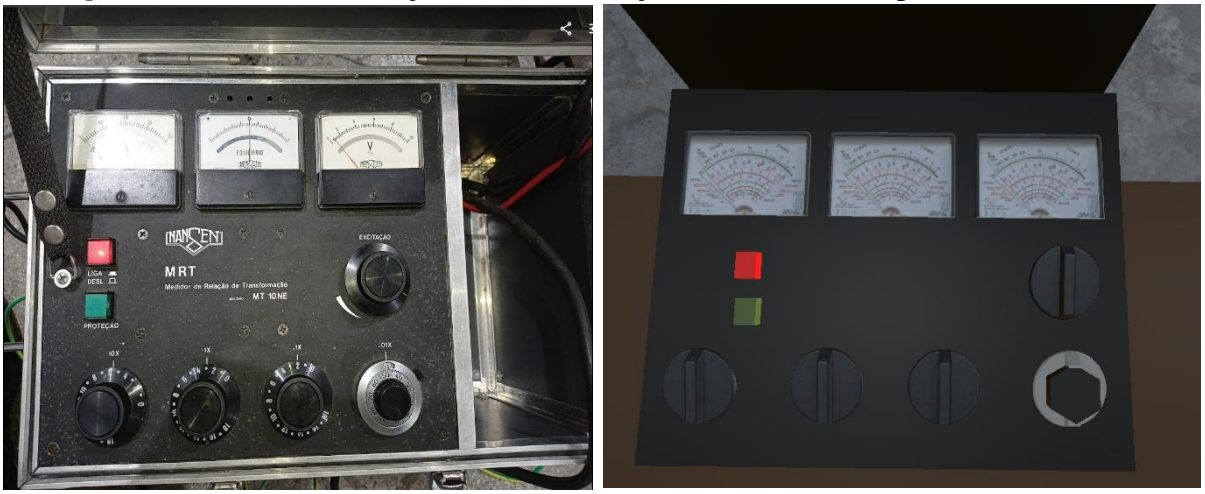
Fonte: Autor (2024).

Figura 14. Megômetro real e modelo 3D de megômetro.



Fonte: Autor (2024).

Figura 15. Medidor de relação de transformação (MRT) e seu respectivo modelo 3D.



Fonte: Autor (2024).

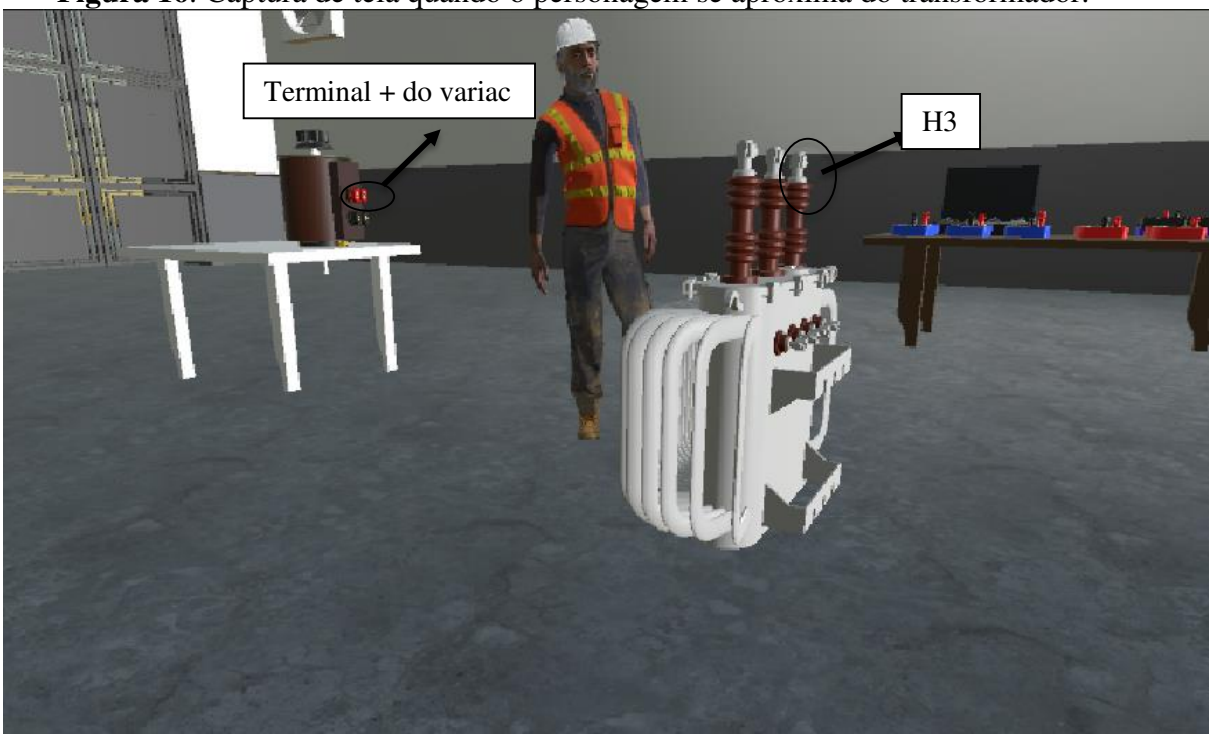
4.2.2 Integração dos *Scripts* com Objetos

Diversos *scripts* foram desenvolvidos para gerenciar as interações e comportamentos dentro da plataforma. Entre os principais *scripts*, destacam-se o *Wire Connection Handler*, responsável por possibilitar conexão dos terminais dos equipamentos e instrumentos. Este código abrange diversas funcionalidades a fim de tornar a interação com o usuário mais fluída e realista, tornando o *script* maior e consequentemente mais complexo. O seu correto funcionamento, entretanto, é fundamental pois é o pilar para a realização de todos os ensaios, pelo fato de conter a lógica por trás da montagem dos circuitos.

Quando a plataforma foi executada, foi observado que as estratégias definidas tiveram êxito (estratégias de conexões e desconexões a partir de conectores implementado em cada um dos instrumentos e equipamentos), e atenderam às expectativas de comportamento. Para um exemplificar um dos processos mais básicos da plataforma, foi pretendido realizar uma única conexão do terminal H3 do transformador, até o terminal positivo do variac. Nas Figuras 16, 17, 18 , 19 e 20, é possível identificar o funcionamento da plataforma em diferentes etapas na criação dos condutores.

Na Figura 16, mostra-se uma captura de tela quando o personagem se aproxima do transformador de distribuição para realizar a conexão do terminal H3 do transformador, ao terminal positivo do variac.

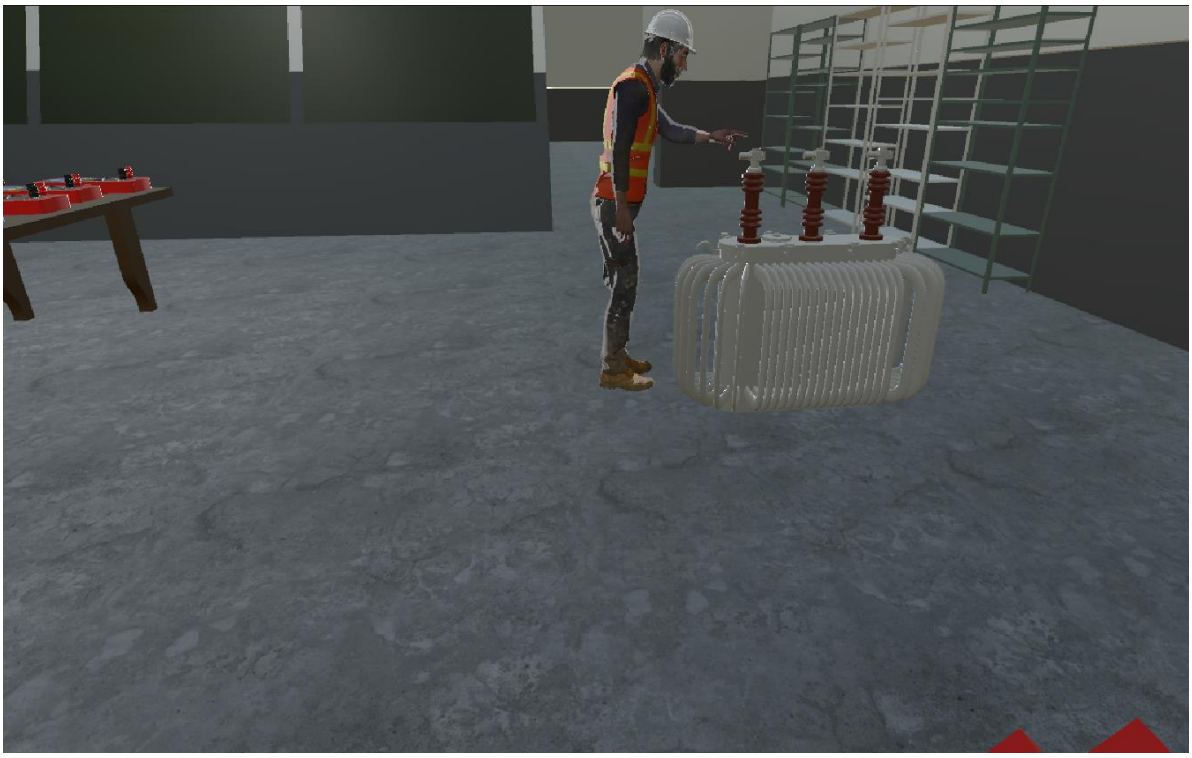
Figura 16. Captura de tela quando o personagem se aproxima do transformador.



Fonte: Autor (2024).

Na Figura 17, mostra-se o funcionamento da animação do personagem ao tocar o terminal H3 do transformador. Este procedimento é controlado pelo usuário ao clicar com o botão esquerdo sobre o terminal em questão, que acarreta a identificação de comando para criação de uma nova conexão, e início de um novo cabo.

Figura 17. Funcionamento da animação ao tocar o terminal H3 do transformador.



Fonte: Autor (2024).

Após clicar no terminal do transformador, é criado um objeto fio na hierarquia do projeto. Assim, uma extremidade do fio é fixada no terminal H3. A outra extremidade do cabo conforme planejado, segue a posição do personagem de acordo com a movimentação dele pelo cenário, simulando o manuseio desta segunda extremidade do cabo.

A simulação dos cabos elétricos com efeitos de gravidade foi um destaque do projeto. Utilizando animações que replicam o comportamento físico dos cabos, foi possível aumentar o realismo da plataforma. Os cabos se curvam de forma aproximadamente parabólica, proporcionando uma experiência visual mais fiel à realidade. Este realismo contribui para uma melhor imersão do usuário no ambiente virtual.

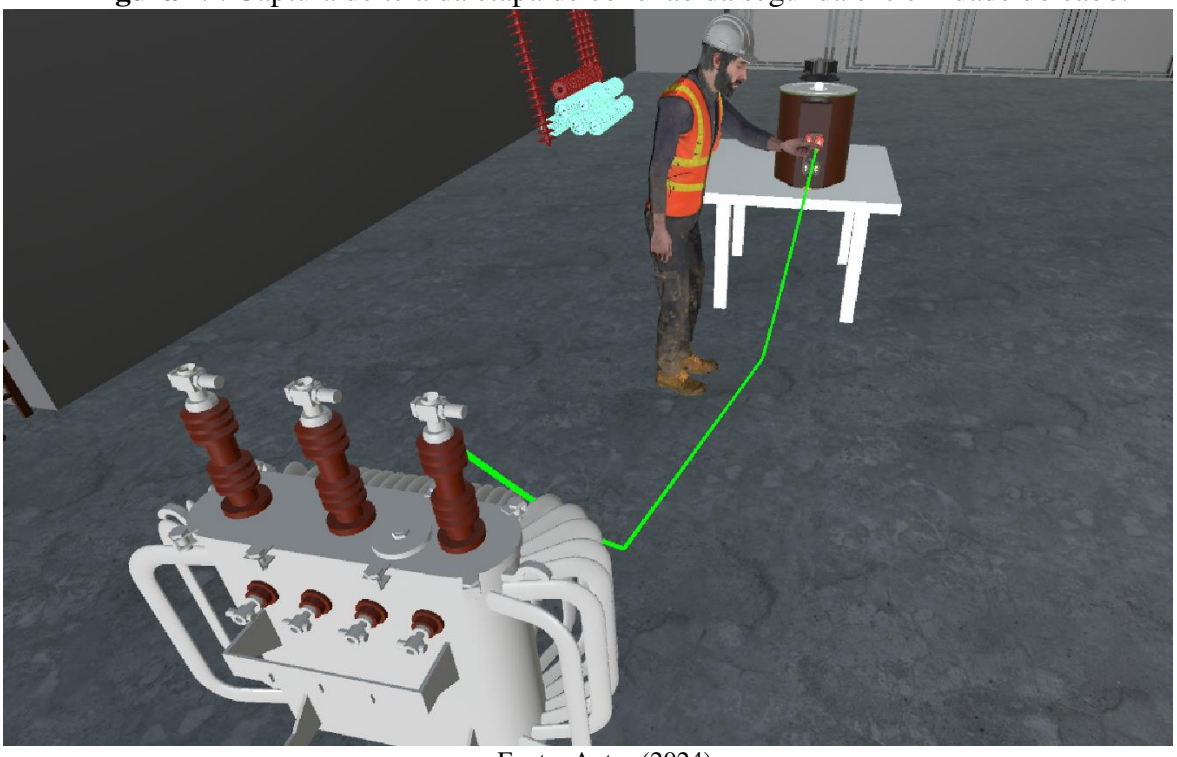
Para simular o efeito da gravidade nos cabos, foram implementados trechos de código a fim de alterar as alturas (coordenadas no eixo Y) de cada ponto que constitui o objeto, possibilitando com que fiquem dispostos de parabólica no cenário, mas com a limitação de não haver pontos com alturas negativas, impedindo que a parábola atravesse o plano do chão. A Figura 18 mostra uma captura de tela durante esta etapa.

Figura 18. Captura de tela durante a etapa de transporte da segunda extremidade do cabo.

Fonte: Autor (2024).

A partir de um segundo clique do mouse sobre algum outro terminal disponível na plataforma, no exemplo adotado como o terminal positivo do variac, é fixado a outra extremidade do cabo.

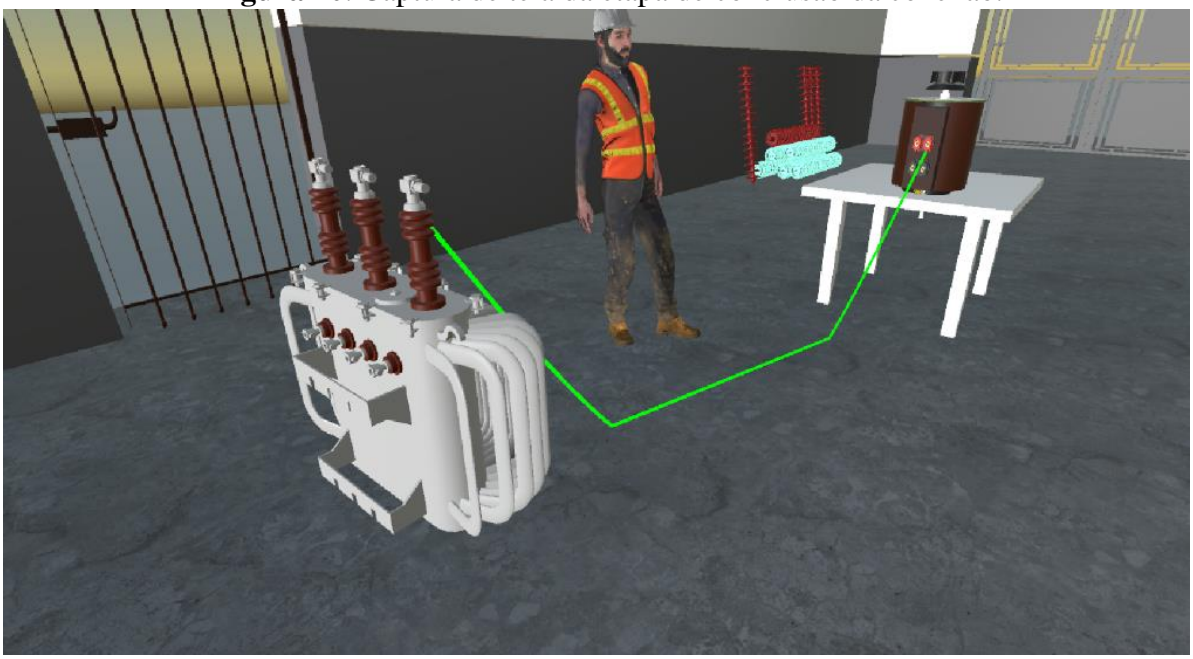
Figura 19. Captura de tela da etapa de conexão da segunda extremidade do cabo.



Fonte: Autor (2024).

Na Figura 20, mostra-se a etapa da conclusão da conexão, demonstrando o funcionamento esperado da plataforma para este procedimento.

Figura 20. Captura de tela da etapa de conclusão da conexão.



Fonte: Autor (2024).

Com a conexão efetivada entre algum terminal de equipamento ou instrumento a outro terminal, exemplificado pela ligação do terminal H3 do transformador, com o terminal positivo do variac, foi possível demonstrar o funcionamento conjunto de diferentes *scripts*, como o *Player Movement, Camera Follow, Wire Connection Handler* de maneira adequada.

A partir destas interações os outros *scripts* destes objetos foram ativados e interligados. Então, a partir das lógicas de funcionamento programadas, dependendo do cenário e estímulos, pôde-se gerar respostas como atualização de variáveis que representam grandezas elétricas, por exemplo igualar a tensão dos terminais interligados pelas duas extremidades, considerando, portanto, o modelo ideal para fio ou cabo.

4.2.3 Realização de Ensaios

A plataforma possibilita a realização de diferentes tipos de ensaios em transformadores, como por exemplo, ensaio para determinação de polaridade, ensaio de saturação, entre outros assim como também permite realização de procedimentos como inspeção visual. Para realizar os ensaios foi desejado que o usuário tivesse o máximo de liberdade possível, e que este não necessite seguir um caminho pré-definido cronologicamente para realização de conexões, desconexões, leituras etc. sendo assim, ao invés de criar fases separadas, restringindo as

funcionalidades totais da plataforma de acordo com o tipo de ensaio escolhido, optou-se por deixar disponíveis todas as funcionalidades ao usuário, sendo que este pode usufruir destas, a qualquer momento. Portanto a plataforma 3D desenvolvida não faz diferenciação do cenário, ou objetos entre os diferentes ensaios, ficando à cargo do usuário decidir a sequência de passos de deseja seguir.

Um dos objetivos do projeto era proporcionar uma experiência interativa para o usuário. Com os *scripts* implementados, os usuários podem facilmente conectar e desconectar cabos, ajustar a posição da câmera e manipular os instrumentos elétricos. Esta interatividade é fundamental para o aprendizado prático, permitindo que os usuários compreendam melhor os conceitos teóricos ao visualizar os resultados em tempo real.

A plataforma foi criada de tal forma que possibilitou para além da realização além dos ensaios, também o processo de inspeção visual do transformador. Pelo fato de a plataforma oferecer um elevado grau de detalhamento de seus objetos que o compõem, foi oferecido ao usuário uma visão detalhada e tridimensional das principais partes que constituem um transformador de distribuição.

Através da inspeção visual do transformador, foi possível identificar as principais partes que constituem o transformador. Foi desenvolvido um comando especial, por meio da interação do usuário ao clicar na tecla i, em que era mostrada placa de identificação do transformador, um dos principais pontos que precisava ser abordado durante a inspeção visual. Além disso, por trabalhar com modelos tridimensionais bem detalhados e realistas, com o uso da plataforma foi possibilitado que o usuário visualizasse componentes como buchas de alta tensão, bucha de baixa tensão, gancho de suspensão, suporte para fixação do transformador ao poste, base de apoios, tanque, radiadores entre outras partes importantes.

Este tipo de informação oferecida pela plataforma, visou promover discussões sobre o tema, e oferecer uma noção melhor ao aluno sobre as partes que constituem o transformador, e suas características técnicas a partir da placa de identificação.

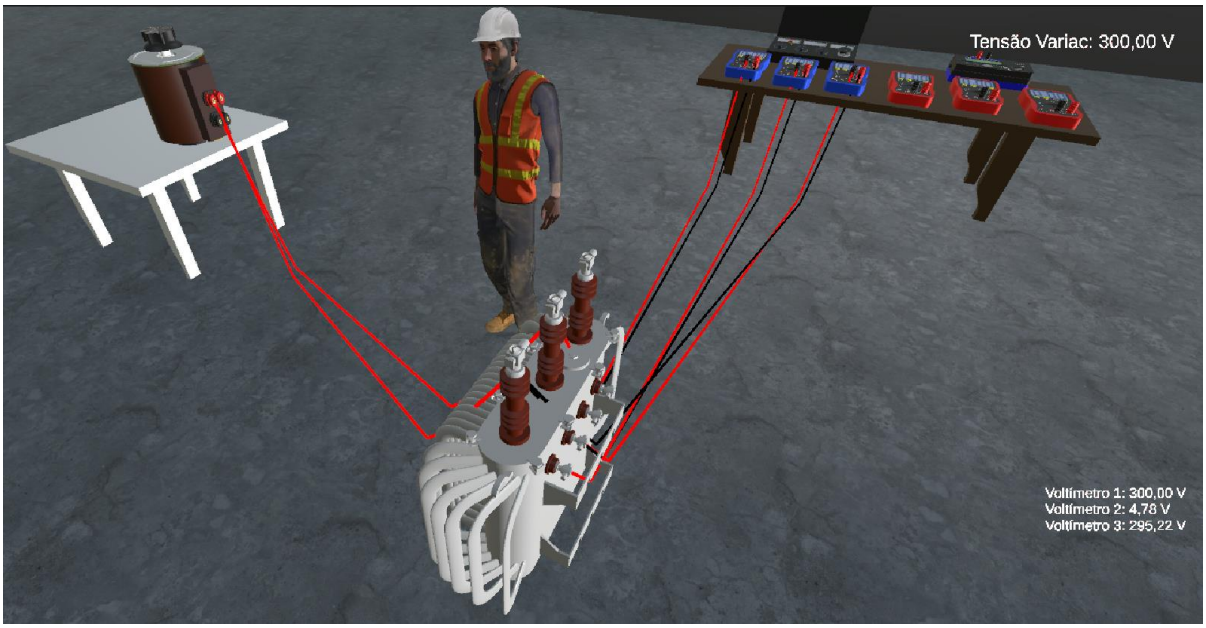
Figura 21. Captura de tela do processo de inspeção visual do transformador.

Fonte: Autor (2024).

Os *scripts* desenvolvidos para o transformador permitiram a modificação e o monitoramento das tensões e correntes em tempo real. Ao conectar os terminais do transformador a outras fontes de tensão, como o variac, foi possível observar a alteração dos valores de tensão e corrente nos terminais correspondentes. Este ajuste dinâmico é crucial para a realização de ensaios precisos e educativos. Um dos exemplos de ensaios que a plataforma possibilita realização, é o ensaio de determinação de polaridade.

Na Figura 22, mostra-se uma das montagens relacionadas a realização do ensaio de determinação da polaridade de cada uma das fases. Como no experimento além do transformador, utilizou voltímetros e variac, através dos comandos adequados configurou-se a tela principal do jogo para mostrar os valores destas principais grandezas envolvidas no ensaio.

Figura 22. Captura de tela da realização do experimento de determinação de polaridade.



Fonte: Autor (2024).

Para validar os resultados obtidos na plataforma, foram realizados testes comparativos entre as medições simuladas e os valores teóricos esperados. Os resultados demonstraram uma alta exatidão dos *scripts* implementados, com diferenças mínimas entre os valores simulados e os teóricos. Foram realizadas modelagens matemáticas utilizando os dados reais dos ensaios realizados presencialmente, afim de realizar aproximações que resultassem em valores simulados muito próximos dos obtidos em laboratório. Esta validação é essencial para confirmar a eficácia e a confiabilidade da plataforma desenvolvida.

Para um funcionamento adequado, compatível com os resultados obtidos em ensaios reais, foram utilizados dados reais obtidos do experimento de transformadores da disciplina de Equipamentos Elétricos, que ocorreram no ano de 2023 de maneira presencial. A partir destes dados foram realizadas aproximações e modelagens, para posterior implementação nos *scripts* dos objetos da plataforma. Um exemplo deste tipo de situação, ocorre no ensaio de saturação do transformador, utilizado para determinação da curva λmax-Imax. Para este ensaio houve utilização de métodos numéricos para correlacionar as variáveis de maneira adequada.

Os dados reais da medição realizada no experimento real são mostrados na Tabela 1.

Tabela 1. Dados reais da medição realizada.

V (V)	A1 (A)	A2 (A)	A3 (A)
0	0	0	0
25	0,013	0,018	0,024
50	0,028	0,037	0,047
100	0,058	0,086	0,096
125	0,083	0,126	0,14
150	0,156	0,203	0,237
175	0,333	0,366	0,431
200	0,705	0,657	0,92
220	1,215	1,16	1,714
250	3,798	4,155	5,21
275	11,728	11,081	13,07

Fonte: Autor (2024).

A partir destes dados, foi realizada interpolação a partir de um modelo exponencial afim de determinar equações que relacionassem a tensão V aplicada a partir do variac entre os terminais de baixa tensão do transformador, com as correntes medidas em cada uma das fases.

Com auxílio de planilha no Microsoft Excel, foram identificadas equações que representaram as correntes em cada uma das fases, com relação a sua respectiva tensão de fase. Os ajustes tiveram coeficiente de determinação R^2 superior 99% para todas as fases do transformador, o que significa um bom ajuste dos dados. Como resultado foram utilizadas Equações 1, 2 e 3 no *script Transformer Controller*.

$$I1 = 6{,}63 * 10^{-4} * e^{-0{,}0353 * Vx10} \quad (1)$$

$$I2 = 1{,}26 * 10^{-3} * e^{-0{,}0328 * Vx20} \quad (2)$$

$$I3 = 1{,}84 * 10^{-3} * e^{-0{,}0321 * Vx30} \quad (3)$$

Em que:

I1 - corrente no terminal X1 (A);

I2 - corrente no terminal X2 (A);

I3 - corrente no terminal X3 (A);

Vx10 - tensão entre o terminal X1 e neutro (V);

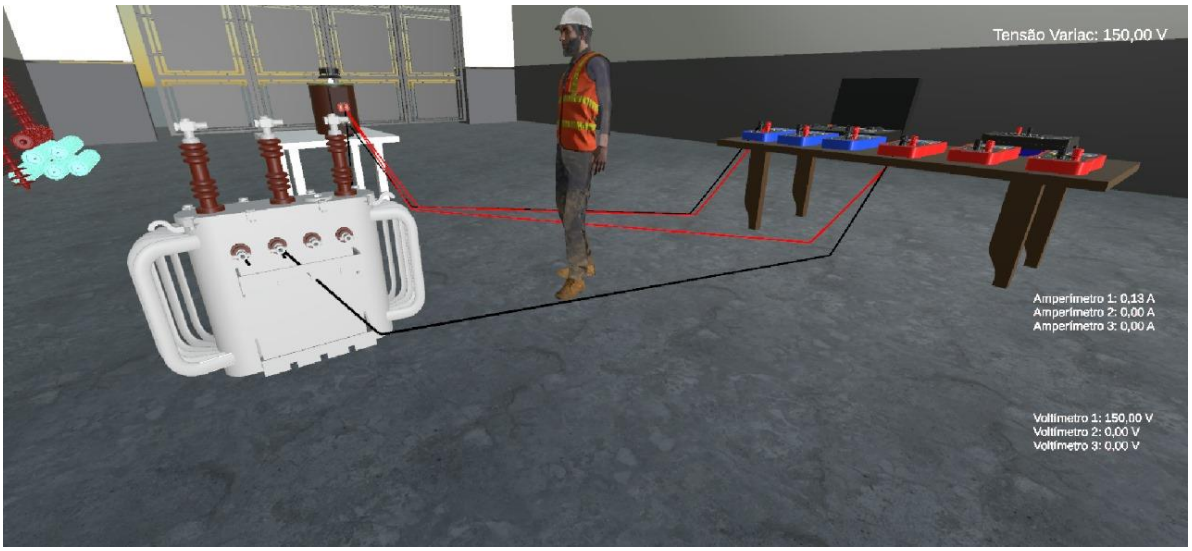
Vx20 - tensão entre o terminal X1 e neutro (V);

Vx30 - tensão entre o terminal X1 e neutro (V).

Nas Figuras 23, 24 e 25, podem ser verificados três momentos durante a realização do ensaio de saturação do transformador. O arranjo foi disposto com alimentação do variac entre os terminais X1 e X0, sendo realizada a medição da corrente de saturação do transformador por meio do amperímetro 1, e da tensão aplicada nos terminais do transformador por meio do voltímetro 1.

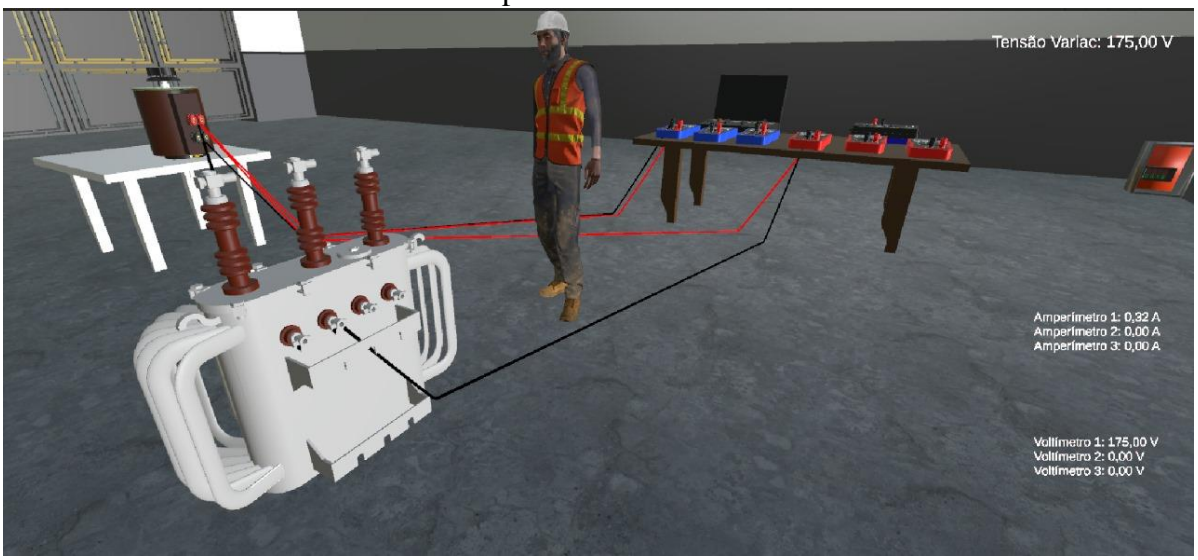
Observa-se o funcionamento correto da plataforma, ao perceber que os valores de correntes medidos para três níveis de tensões aplicadas, são muito próximos dos valores medidos presentes na Tabela 1 para a fase 1.

Figura 23. Captura de tela de realização de ensaio de saturação de transformador para tensão aplicada de 150 V.



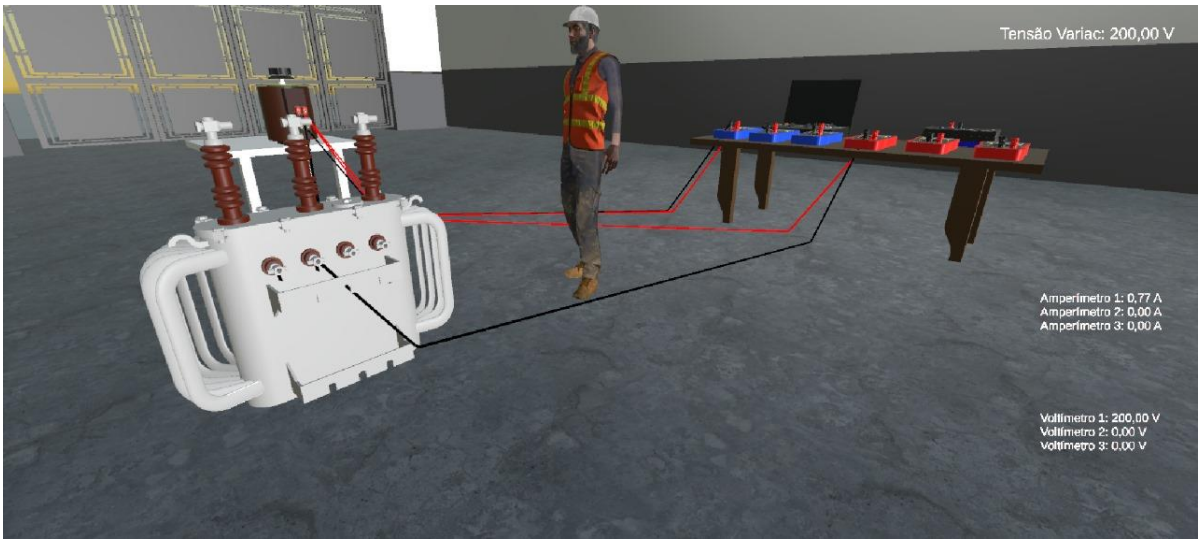
Fonte: Autor (2024).

Figura 24. Captura de tela de realização de ensaio de saturação de transformador para tensão aplicada de 175 V.



Fonte: Autor (2024).

Figura 25. Captura de tela de realização de ensaio de saturação de transformador para tensão aplicada de 200 V.



Fonte: Autor (2024).

Diante do exposto, observa-se o potencial de funcionamento da plataforma para alguns ensaios dentre os tipos possíveis de serem emulados. Para além dos ensaios e procedimentos abordados, outros tipos de ensaios conseguem ser realizados na plataforma desenvolvida, tais como o ensaio de corrente a vazio e medição de perdas, relação de transformação, operação em curto-circuito e medição de resistência de isolamento.

4.2.4 Dificuldades Encontradas

Ao longo do processo de desenvolvimento da plataforma 3D algumas dificuldades foram encontradas. Inicialmente os maiores desafios encontrados foram no processo de familiarização com a Unity. Mesmo sendo uma plataforma intuitiva, pelo fato de serem utilizadas múltiplas janelas e abas, a percepção rápida de onde se localizavam as diferentes ferramentas foi prejudicada. Esta dificuldade se tornou menos relevante à medida que o projeto era desenvolvido e mais experiência era adquirida pelo aluno.

Um dos principais fatores que dificultaram a elaboração do ambiente virtual foi a escassez de ativos, como por exemplo, objetos para importação para a Unity. A *AssetStore* mesmo possuindo uma diversidade de modelos 3D disponíveis, quando se tratou de itens voltados para áreas mais específicas como equipamentos elétricos e instrumentos de medição, a disponibilidade é bastante limitada e em alguns casos inexistentes. Para suprir esta demanda o aluno teve que se dedicar em processo de modelagem 3D, dentro da própria Unity para objetos

mais simples ou utilizando softwares como o Blend para criar alguns objetos mais complexos e que foram posteriormente incorporados à plataforma.

Exemplo deste tipo de solução foi para criar o modelo 3D do Medidor de Relação de Transformação MRT. Outra forma de contornar este impasse foi encontrar outras fontes para os ativos 3D, como foi o caso do site Warehouse 3D, pois mesmo não encontrando objetos completos já dimensionados, como por exemplo o variac, foi possível encontrar partes que puderam ser unidas para montar o modelo completo do equipamento.

Outro desafio encontrado foi o desenvolvimento e a integração dos diferentes *scripts*, aos objetos do jogo. Pelo fato de se utilizar diversas funcionalidades para emular os ensaios, foi necessário trabalhar com múltiplos códigos que em grande parte eram extensos e consequentemente ocasionaram com mais frequência erros de compilação e depuração.

5. CONCLUSÃO

Diante do que foi exposto, constatou-se que os objetivos foram atingidos com sucesso. Foi desenvolvida uma plataforma virtual interativa que simulou ensaios em transformadores, destinada primordialmente aos alunos do curso de Engenharia Elétrica da Universidade Federal de Campina Grande, embora também tenha o potencial de beneficiar também estudantes de outras instituições. A implementação deste laboratório virtual em software ofereceu uma ferramenta educativa e avançada para a disciplina de Equipamentos Elétricos, permitindo a emulação de experimentos em um ambiente interativo e dinâmico. Esta abordagem não só possibilita a compreensão mais aprofundada dos conceitos teóricos de seus usuários, como também otimiza o processo de aprendizagem ao simular ensaios de tipo e de rotina.

A plataforma cumpriu seus objetivos, ao conseguir implementar com êxito uma interface gráfica tridimensional e realística que representou um laboratório equipado com um transformador de distribuição, um variac utilizado como fonte de tensão e instrumentos de medição como amperímetros, voltímetros, megômetro etc. Aliado a isto, também foram elaboradas lógicas de funcionamento por meio de *scripts* que foram consistentes em seus resultados e cálculos realizados. A plataforma atendeu as expectativas e permitiu realização de procedimentos técnicos, como por exemplo inspeção visual de um transformador, além de diversos ensaios específicos como o de polaridade, saturação para determinar a curva de λmax versus Imax, corrente em vazio, relação de transformação, operação em curto-circuito e medição da resistência de isolamento.

Como diferencial a montagem individual dos arranjos na plataforma virtual permitiu uma compreensão integral dos procedimentos por parte do aluno, diferentemente do ambiente presencial, onde a divisão de tarefas entre vários alunos muitas vezes impede que cada um experimente todas as etapas do ensaio. A plataforma também ajudou seus usuários na questão da limitação de tempo nos experimentos presenciais, oferecendo mais flexibilidade para a realização dos ensaios sem pressão temporal. Por meio das simulações disponíveis, esperou-se que os alunos aprimorassem tanto a sua compreensão teórica quanto suas habilidades práticas relacionadas aos equipamentos elétricos, reforçando assim a qualidade do ensino técnico e prático oferecido pela instituição.

Conforme apresentado, a experiência de elaborar o projeto da plataforma 3D não apenas aprimorou o caráter autodidata do aluno autor deste trabalho, acerca do assunto de ensaios em transformadores, como também proporcionou uma compreensão aprofundada da *game engine* Unity, além de permitir a prática efetiva da linguagem de programação C#. Essencialmente,

este projeto reforçou significativamente a capacidade de resolução de problemas, habilidade fundamental para superar os diversos desafios encontrados durante o processo de desenvolvimento.

REFERÊNCIAS

ABNT – Associação Brasileira de Normas Técnicas. NBR 5356: Transformadores de Potência, dezembro de 2007.

ABNT – Associação Brasileira de Normas Técnicas. NBR 5380: Transformador de Potência – Método de Ensaio. Maio de 1993.

FERREIRA, H. A. Relatório de Estágio. Universidade Federal da Paraíba. Campina Grande, Paraíba. Março de 2000.

FILHO, J. M. Manual de Equipamentos Elétricos. Volume 1. 2a Edição. Livros Técnicos e Científicos Editora. Rio de Janeiro, 1994.

FRONTIN, S. O. e et al. - Equipamentos de Alta Tensão; Prospecção e Hierarquização de Inovações Tecnológicas, 1a. Edição, Brasília, 2013.

GERVÁSIO, L., da S., L. C., de J., A. M., & B., E. A. Transformadores de Potência: ensaios e proteção. **Brazilian Journal of Development**, 7 (6), 59954-59975, 2021.

LUCIANO, B. A. Apostila sobre Transformadores. Universidade Federal da Paraíba. Campina Grande, Paraíba.

MEGGER. Operating Instructions – Major MEGGER Tester. 3rd Edition. England.
NANSEN S. A. Medidor de Relação de Espiras de Transformadores MT 10NE – Manual de Instruções. 3a Edição. Minas Gerais, 1995.

OLIVEIRA, J. C. de, C., J. R. e ABREU, J. P. G. de. Transformadores: Teoria e Ensaios. Editora Edgard Blucher LTDA. São Paulo, 1984.

SOUSA, A. A. Relatório de Estágio. Universidade Federal da Paraíba. Campina Grande, Paraíba. Junho de 1998.

SOUZA, de L. S. Desenvolvimento do ambiente virtual do experimento transformadores de Potência. Trabalho de Conclusão de Curso de Engenharia Elétrica. 95p. Campina Grande - PB, 2021.

Unity 3D. Disponível em: https://unity.com/pt. Acesso em: maio de 2024.

ANEXO A

```
1  using UnityEngine;
2
3  public class PlayerMovement : MonoBehaviour
4  {
5      public float speed = 5.0f; // Ajuste conforme necessário
6      public float rotationSpeed = 720.0f; // Velocidade de rotação
7      public float rayDistance = 0.5f; // Distância do raycast para
         detecção de obstáculos
8      private Animator animator;
9      public WireConnectionHandler wireConnectionHandler; // Referência
         ao objeto WireConnectionHandler
10
11     private bool isMovingToTarget = false;
12     private Vector3 targetPosition;
13     private string walkAnimationTrigger = "Walk";
14     private float fixedY; // Coordenada Y fixa do avatar
15     private Vector3 movement;
16
17     void Start()
18     {
19         animator = GetComponent<Animator>();
20         fixedY = transform.position.y; // Armazena a coordenada Y
             inicial
21     }
22
23     void Update()
24     {
25         float moveHorizontal = Input.GetAxis("Horizontal");
26         float moveVertical = Input.GetAxis("Vertical");
27
28         // Definir o movimento com base no input do jogador
29         Vector3 inputMovement = new Vector3(moveHorizontal, 0.0f,
             moveVertical).normalized;
30
31         // Parar o movimento se não houver input
32         if (inputMovement == Vector3.zero)
33         {
34             movement = Vector3.zero;
35             animator.SetFloat("Speed", 0f);
36             return;
37         }
38
39         // Verificar se há um obstáculo em frente usando raycast
40         if (!IsObstacleInFront(inputMovement))
41         {
42             movement = inputMovement * speed * Time.deltaTime;
43
44             // Mover o personagem usando Transform
45             Vector3 newPosition = transform.position + movement;
46             newPosition.y = fixedY; // Manter a posição Y fixa
47             transform.position = newPosition;
48
49             // Rotacionar o personagem na direção do movimento
```

```
50              Quaternion toRotation = Quaternion.LookRotation
                  (inputMovement);
51              transform.rotation = Quaternion.RotateTowards
                  (transform.rotation, toRotation, rotationSpeed *
                  Time.deltaTime);
52
53              animator.SetFloat("Speed", 1f); // Ativar a animação de
                  andar
54          }
55          else
56          {
57              animator.SetFloat("Speed", 0f); // Parar a animação de
                  andar se houver obstáculo
58          }
59
60          // Verificar clique do mouse para animação de interação
61          if (Input.GetMouseButtonDown(0))
62          {
63              animator.SetTrigger("Interact");
64          }
65
66          // Verificar a entrada do teclado para a animação de pegar fio
67          if (Input.GetKeyDown(KeyCode.Z))
68          {
69              animator.SetTrigger("PickUp");
70          }
71
72          // Verificar a entrada do teclado para a animação de colocar
              fio
73          if (Input.GetKeyDown(KeyCode.X))
74          {
75              animator.SetTrigger("PlaceWire");
76          }
77
78          // Mover o personagem para a posição alvo se necessário
79          if (isMovingToTarget)
80          {
81              MoveToTarget();
82          }
83      }
84
85      void LateUpdate()
86      {
87          // Garantir que a rotação nos eixos X e Z permaneça 0
88          Vector3 rotation = transform.rotation.eulerAngles;
89          rotation.x = 0;
90          rotation.z = 0;
91          transform.rotation = Quaternion.Euler(rotation);
92
93          // Garantir que a posição Y permaneça fixa
94          Vector3 position = transform.position;
95          position.y = fixedY;
96          transform.position = position;
```

```
 97      }
 98
 99      public void MoveToPosition(Vector3 position, string walkTrigger)
100      {
101          targetPosition = position;
102          walkAnimationTrigger = walkTrigger;
103          isMovingToTarget = true;
104          animator.SetTrigger(walkAnimationTrigger);
105      }
106
107      private void MoveToTarget()
108      {
109          Vector3 targetPositionAdjusted = new Vector3(targetPosition.x,
               fixedY, targetPosition.z);
110          Vector3 direction = (targetPositionAdjusted -
               transform.position).normalized;
111
112          if (!IsObstacleInFront(direction))
113          {
114              Vector3 movement = direction * speed * Time.deltaTime;
115              Vector3 newPosition = transform.position + movement;
116              newPosition.y = fixedY; // Manter a posição Y fixa
117              transform.position = newPosition;
118
119              // Rotacionar o personagem na direção do movimento
120              if (direction != Vector3.zero)
121              {
122                  Quaternion toRotation = Quaternion.LookRotation
                       (direction);
123                  transform.rotation = Quaternion.RotateTowards
                       (transform.rotation, toRotation, rotationSpeed *
                       Time.deltaTime);
124              }
125
126              if (Vector3.Distance(transform.position,
                   targetPositionAdjusted) < 0.1f)
127              {
128                  isMovingToTarget = false;
129                  animator.ResetTrigger(walkAnimationTrigger);
130              }
131          }
132      }
133
134      private bool IsObstacleInFront(Vector3 direction)
135      {
136          RaycastHit hit;
137          Vector3 rayOrigin = transform.position + Vector3.up * 0.5f; //
               Ajustar a origem do raycast para estar na altura do
               personagem
138          if (Physics.Raycast(rayOrigin, direction, out hit,
               rayDistance))
139          {
140              return hit.collider != null;
```

```
141             }
142             return false;
143         }
144     }
145
```

ANEXO B

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

public class WireConnectionHandler : MonoBehaviour
{
    public Camera mainCamera; // A câmera principal que segue o personagem
    public Transform player; // O transform do jogador
    public GameObject wirePrefab; // Prefab do LineRenderer para representar o fio
    public float wireThickness = 0.02f; // A espessura dos fios
    public int segments = 20; // O número de segmentos da linha para simular a curva parabólica
    public float maxReachDistance = 3.0f; // Distância máxima que o avatar pode alcançar para conectar o fio
    public TransformerController transformerController; // Referência ao script TransformerController
    public VariacController variacController; // Referência ao script VariacController
    public VoltimetroController voltimetroController1; // Referência ao script VoltimetroController do primeiro multímetro
    public VoltimetroController voltimetroController2; // Referência ao script VoltimetroController do segundo multímetro
    public VoltimetroController voltimetroController3; // Referência ao script VoltimetroController do terceiro multímetro
    public AmperimetroController amperimetroController1; // Referência ao script AmperimetroController do primeiro amperímetro
    public AmperimetroController amperimetroController2; // Referência ao script AmperimetroController do segundo amperímetro
    public AmperimetroController amperimetroController3; // Referência ao script AmperimetroController do terceiro amperímetro

    private List<LineRenderer> wires = new List<LineRenderer>(); // Lista para armazenar os fios criados
    private List<(Transform, Transform)> wireConnections = new List<(Transform, Transform)>(); // Lista para armazenar as conexões de fios
    private Vector3 initialPoint; // Ponto inicial do fio
    private bool isConnecting = false; // Flag para determinar se estamos conectando um fio
    private LineRenderer currentLineRenderer; // Referência ao LineRenderer do fio atual
    private Transform terminalInicial; // O terminal inicial conectado

    void Update()
    {
        // Detectar clique do mouse para iniciar ou finalizar a conexão do fio
        if (Input.GetMouseButtonDown(0))
        {
            Ray ray = mainCamera.ScreenPointToRay(Input.mousePosition);
            RaycastHit hit;
            if (Physics.Raycast(ray, out hit))
```

```
36              {
37                  if (Vector3.Distance(hit.point, player.position) <=
                      maxReachDistance)
38                  {
39                      if (!isConnecting)
40                      {
41                          // Iniciar a conexão do fio no ponto de conexão
                         pré-definido
42                          initialPoint = hit.collider.bounds.center; //
                        Usar o centro do colisor como ponto inicial
43                          StartConnectingWire(initialPoint);
44
45                          // Verificar se o ponto inicial é um terminal
                        do Variac
46                          if (hit.collider.transform ==
                        variacController.terminalPositivo)
47                          {
48                              Debug.Log("Conectando fio ao terminal + do
                        Variac");
49                              terminalInicial = hit.collider.transform;
50                          }
51                          else if (hit.collider.transform ==
                        variacController.terminalNegativo)
52                          {
53                              Debug.Log("Conectando fio ao terminal - do
                        Variac");
54                              terminalInicial = hit.collider.transform;
55                          }
56                          // Verificar se o ponto inicial é um terminal
                        do transformador
57                          else if (IsTransformadorTerminal
                        (hit.collider.transform))
58                          {
59                              Debug.Log($"Conectando ao terminal
                        {hit.collider.transform.name} do Transformador");
60                              terminalInicial = hit.collider.transform;
61                          }
62                          // Verificar se o ponto inicial é um terminal
                        do voltímetro
63                          else if (IsVoltimetroTerminal
                        (hit.collider.transform))
64                          {
65                              Debug.Log($"Conectando ao terminal
                        {hit.collider.transform.name} do Voltímetro");
66                              terminalInicial = hit.collider.transform;
67                          }
68                          // Verificar se o ponto inicial é um terminal
                        do amperímetro
69                          else if (IsAmperimetroTerminal
                        (hit.collider.transform))
70                          {
71                              Debug.Log($"Conectando ao terminal
                        {hit.collider.transform.name} do Amperímetro");
```

```
72                          terminalInicial = hit.collider.transform;
73                      }
74                  }
75                  else
76                  {
77                      // Finalizar a conexão do fio no ponto clicado
78                      Vector3 endPoint =
                    hit.collider.bounds.center; // Usar o centro do
                    colisor como ponto final
79                      UpdateLineRenderer(currentLineRenderer,
                    initialPoint, endPoint);
80                      isConnecting = false;
81
82                      // Verificar se o ponto final é um terminal do
                    Variac
83                      if (hit.collider.transform ==
                    variacController.terminalPositivo)
84                      {
85                          Debug.Log("Conectando fio ao terminal + do
                    Variac");
86                          transformerController.ConectarTerminal
                    (terminalInicial, variacController.VPositivo);
87                          transformerController.ConectarTerminal
                    (hit.collider.transform, variacController.VPositivo);
88                          ConectarTerminalVoltimetros
                    (terminalInicial, variacController.VPositivo);
89                          ConectarTerminalVoltimetros
                    (hit.collider.transform, variacController.VPositivo);
90                          ConectarTerminalAmperimetros
                    (terminalInicial, hit.collider.transform,
                    variacController.VPositivo);
91                          wireConnections.Add((terminalInicial,
                    hit.collider.transform));
92                      }
93                      else if (hit.collider.transform ==
                    variacController.terminalNegativo)
94                      {
95                          Debug.Log("Conectando fio ao terminal - do
                    Variac");
96                          transformerController.ConectarTerminal
                    (terminalInicial, variacController.VNegativo);
97                          transformerController.ConectarTerminal
                    (hit.collider.transform, variacController.VNegativo);
98                          ConectarTerminalVoltimetros
                    (terminalInicial, variacController.VNegativo);
99                          ConectarTerminalVoltimetros
                    (hit.collider.transform, variacController.VNegativo);
100                         ConectarTerminalAmperimetros
                    (terminalInicial, hit.collider.transform,
                    variacController.VNegativo);
101                         wireConnections.Add((terminalInicial,
                    hit.collider.transform));
102                     }
```

```
103                         // Verificar se o ponto final é um terminal do transformador
104                         else if (IsTransformadorTerminal(hit.collider.transform))
105                         {
106                             Debug.Log($"Conectando ao terminal {hit.collider.transform.name} do Transformador");
107                             float tensaoTerminalInicial = GetPotencialDoTerminal(terminalInicial);
108                             transformerController.ConectarTerminal(hit.collider.transform, tensaoTerminalInicial);
109                             ConectarTerminalVoltimetros(hit.collider.transform, tensaoTerminalInicial);
110                             ConectarTerminalAmperimetros(terminalInicial, hit.collider.transform, tensaoTerminalInicial);
111                             wireConnections.Add((terminalInicial, hit.collider.transform));
112                         }
113                         // Verificar se o ponto final é um terminal do voltímetro
114                         else if (IsVoltimetroTerminal(hit.collider.transform))
115                         {
116                             Debug.Log($"Conectando ao terminal {hit.collider.transform.name} do Voltímetro");
117                             float tensaoTerminalInicial = GetPotencialDoTerminal(terminalInicial);
118                             ConectarTerminalVoltimetros(hit.collider.transform, tensaoTerminalInicial);
119                             ConectarTerminalAmperimetros(terminalInicial, hit.collider.transform, tensaoTerminalInicial);
120                             wireConnections.Add((terminalInicial, hit.collider.transform));
121                         }
122                         // Verificar se o ponto final é um terminal do amperímetro
123                         else if (IsAmperimetroTerminal(hit.collider.transform))
124                         {
125                             Debug.Log($"Conectando ao terminal {hit.collider.transform.name} do Amperímetro");
126                             float tensaoTerminalInicial = GetPotencialDoTerminal(terminalInicial);
127                             ConectarTerminalAmperimetros(terminalInicial, hit.collider.transform, tensaoTerminalInicial);
128                             wireConnections.Add((terminalInicial, hit.collider.transform));
129                         }
130                         currentLineRenderer = null;
131                     }
```

```
132                 }
133             }
134         }
135 
136         // Atualizar a posição do segundo ponto do fio enquanto está conectando
137         if (isConnecting && currentLineRenderer != null)
138         {
139             Vector3 playerPosition = player.position + Vector3.up * 1.0f; // Adicionar offset de 1 unidade no eixo Y
140             UpdateLineRenderer(currentLineRenderer, initialPoint, playerPosition);
141         }
142 
143         // Atualizar as tensões dos terminais em tempo real
144         UpdateWireConnections();
145 
146         // Detectar a tecla K para remover o último fio criado
147         if (Input.GetKeyDown(KeyCode.K))
148         {
149             RemoveLastWire();
150         }
151     }
152 
153     private void StartConnectingWire(Vector3 startPoint)
154     {
155         GameObject wireObject = Instantiate(wirePrefab); // Instanciar o prefab do fio
156         currentLineRenderer = wireObject.GetComponent<LineRenderer>();
157         currentLineRenderer.positionCount = segments + 1; // Ajustar o número de segmentos
158         currentLineRenderer.startWidth = wireThickness; // Ajustar a espessura do fio
159         currentLineRenderer.endWidth = wireThickness; // Ajustar a espessura do fio
160         currentLineRenderer.material = new Material(Shader.Find("Sprites/Default")); // Usar um material padrão para cores
161         currentLineRenderer.startColor = GetColor(terminalInicial); // Atribuir uma cor baseada no terminal inicial
162         currentLineRenderer.endColor = currentLineRenderer.startColor; // Atribuir a mesma cor ao final do fio
163         currentLineRenderer.SetPosition(0, startPoint);
164         isConnecting = true;
165         wires.Add(currentLineRenderer);
166     }
167 
168     private void UpdateLineRenderer(LineRenderer lineRenderer, Vector3 startPoint, Vector3 endPoint)
169     {
170         if (lineRenderer == null) return; // Verificar se o LineRenderer ainda existe
171 
```

```
172         float distance = Vector3.Distance(startPoint, endPoint);
173
174         if (distance > 1.5f)
175         {
176             // Criar fio com forma parabólica
177             lineRenderer.positionCount = segments + 1;
178             for (int i = 0; i <= segments; i++)
179             {
180                 float t = i / (float)segments;
181                 Vector3 point = Vector3.Lerp(startPoint, endPoint, t);
182                 float parabolicHeight = Mathf.Sin(Mathf.PI * t) *
                      (distance / 2); // Simular gravidade com a distância
                      real
183                 point.y -= parabolicHeight;
184                 if (point.y < 0.5f) point.y = 0.5f; // Garantir que a
                      altura mínima seja 0.5
185                 lineRenderer.SetPosition(i, point);
186             }
187         }
188         else
189         {
190             // Criar fio reto
191             lineRenderer.positionCount = 2;
192             lineRenderer.SetPosition(0, startPoint);
193             lineRenderer.SetPosition(1, endPoint);
194             for (int i = 0; i < lineRenderer.positionCount; i++)
195             {
196                 Vector3 position = lineRenderer.GetPosition(i);
197                 if (position.y < 0.5f) position.y = 0.5f; // Garantir
                      que a altura mínima seja 0.5
198                 lineRenderer.SetPosition(i, position);
199             }
200         }
201     }
202
203     private Color GetColor(Transform terminal)
204     {
205         if (terminal == variacController.terminalPositivo || terminal
              == voltimetroController1.terminalPositivo || terminal ==
              voltimetroController2.terminalPositivo || terminal ==
              voltimetroController3.terminalPositivo || terminal ==
              amperimetroController1.terminalPositivo || terminal ==
              amperimetroController2.terminalPositivo || terminal ==
              amperimetroController3.terminalPositivo)
206         {
207             return Color.red;
208         }
209         else if (terminal == variacController.terminalNegativo ||
              terminal == voltimetroController1.terminalNegativo ||
              terminal == voltimetroController2.terminalNegativo ||
              terminal == voltimetroController3.terminalNegativo ||
              terminal == amperimetroController1.terminalNegativo ||
              terminal == amperimetroController2.terminalNegativo ||
```

```
            terminal == amperimetroController3.terminalNegativo)
        {
            return Color.black;
        }
        else
        {
            return Color.green; // Fios conectados aos terminais do transformador
        }
    }

    private void RemoveLastWire()
    {
        if (isConnecting)
        {
            // Cancelar o fio incompleto
            isConnecting = false;
            if (currentLineRenderer != null)
            {
                wires.Remove(currentLineRenderer);
                Destroy(currentLineRenderer.gameObject);
                currentLineRenderer = null;
            }
        }
        else if (wires.Count > 0)
        {
            LineRenderer wireToRemove = wires[wires.Count - 1];
            wires.RemoveAt(wires.Count - 1);

            // Remover a conexão correspondente
            if (wireConnections.Count > 0)
            {
                (Transform terminalA, Transform terminalB) = wireConnections[wireConnections.Count - 1];
                wireConnections.RemoveAt(wireConnections.Count - 1);

                // Atualizar a tensão do terminal removido
                transformerController.DesconectarTerminal(terminalA);
                transformerController.DesconectarTerminal(terminalB);
                DesconectarTerminalVoltimetros(terminalA);
                DesconectarTerminalVoltimetros(terminalB);
                DesconectarTerminalAmperimetros(terminalA, terminalB);
            }

            Destroy(wireToRemove.gameObject);
        }
    }

    private bool IsTransformadorTerminal(Transform terminal)
    {
        foreach (Transform t in transformerController.altaTerminais)
        {
            if (t == terminal) return true;
```

```
260         }
261         foreach (Transform t in transformerController.baixaTerminais)
262         {
263             if (t == terminal) return true;
264         }
265         return false;
266     }
267 
268     private bool IsVoltimetroTerminal(Transform terminal)
269     {
270         return terminal == voltimetroController1.terminalPositivo ||
              terminal == voltimetroController1.terminalNegativo ||
271                terminal == voltimetroController2.terminalPositivo ||
                 terminal == voltimetroController2.terminalNegativo ||
272                terminal == voltimetroController3.terminalPositivo ||
                 terminal == voltimetroController3.terminalNegativo;
273     }
274 
275     private bool IsAmperimetroTerminal(Transform terminal)
276     {
277         return terminal == amperimetroController1.terminalPositivo ||
              terminal == amperimetroController1.terminalNegativo ||
278                terminal == amperimetroController2.terminalPositivo ||
                 terminal == amperimetroController2.terminalNegativo ||
279                terminal == amperimetroController3.terminalPositivo ||
                 terminal == amperimetroController3.terminalNegativo;
280     }
281 
282     private float GetPotencialDoTerminal(Transform terminal)
283     {
284         if (terminal == variacController.terminalPositivo)
285         {
286             return variacController.VPositivo;
287         }
288         if (terminal == variacController.terminalNegativo)
289         {
290             return variacController.VNegativo;
291         }
292 
293         if (IsTransformadorTerminal(terminal))
294         {
295             if (terminal == transformerController.altaTerminais[0])
                  return transformerController.VH1;
296             if (terminal == transformerController.altaTerminais[1])
                  return transformerController.VH2;
297             if (terminal == transformerController.altaTerminais[2])
                  return transformerController.VH3;
298             if (terminal == transformerController.baixaTerminais[0])
                  return transformerController.VX0;
299             if (terminal == transformerController.baixaTerminais[1])
                  return transformerController.VX1;
300             if (terminal == transformerController.baixaTerminais[2])
                  return transformerController.VX2;
```

```
301             if (terminal == transformerController.baixaTerminais[3])
                  return transformerController.VX3;
302         }
303
304         return 0f;
305     }
306
307     private void ConectarTerminalVoltimetros(Transform terminal, float
          tensao)
308     {
309         if (terminal == voltimetroController1.terminalPositivo ||
              terminal == voltimetroController1.terminalNegativo)
310         {
311             voltimetroController1.ConectarTerminal(terminal, tensao);
312         }
313         if (terminal == voltimetroController2.terminalPositivo ||
              terminal == voltimetroController2.terminalNegativo)
314         {
315             voltimetroController2.ConectarTerminal(terminal, tensao);
316         }
317         if (terminal == voltimetroController3.terminalPositivo ||
              terminal == voltimetroController3.terminalNegativo)
318         {
319             voltimetroController3.ConectarTerminal(terminal, tensao);
320         }
321     }
322
323     private void DesconectarTerminalVoltimetros(Transform terminal)
324     {
325         if (terminal == voltimetroController1.terminalPositivo ||
              terminal == voltimetroController1.terminalNegativo)
326         {
327             voltimetroController1.DesconectarTerminal(terminal);
328         }
329         if (terminal == voltimetroController2.terminalPositivo ||
              terminal == voltimetroController2.terminalNegativo)
330         {
331             voltimetroController2.DesconectarTerminal(terminal);
332         }
333         if (terminal == voltimetroController3.terminalPositivo ||
              terminal == voltimetroController3.terminalNegativo)
334         {
335             voltimetroController3.DesconectarTerminal(terminal);
336         }
337     }
338
339     private void ConectarTerminalAmperimetros(Transform
          terminalInicial, Transform terminalFinal, float tensao)
340     {
341         if (IsAmperimetroTerminal(terminalInicial) &&
              IsTransformadorTerminal(terminalFinal))
342         {
343             AmperimetroController amperimetro =
```

```
                GetAmperimetroController(terminalInicial);
344             if (amperimetro != null)
345             {
346                 amperimetro.ConectarTerminal(terminalInicial, tensao,
                      transformerController.GetCorrenteDoTerminal
                      (terminalFinal));
347             }
348         }
349         else if (IsAmperimetroTerminal(terminalFinal) &&
              IsTransformadorTerminal(terminalInicial))
350         {
351             AmperimetroController amperimetro =
                  GetAmperimetroController(terminalFinal);
352             if (amperimetro != null)
353             {
354                 amperimetro.ConectarTerminal(terminalFinal, tensao,
                      transformerController.GetCorrenteDoTerminal
                      (terminalInicial));
355             }
356         }
357     }
358
359     private void DesconectarTerminalAmperimetros(Transform terminalA,
          Transform terminalB)
360     {
361         if (IsAmperimetroTerminal(terminalA))
362         {
363             AmperimetroController amperimetro =
                  GetAmperimetroController(terminalA);
364             if (amperimetro != null)
365             {
366                 amperimetro.DesconectarTerminal(terminalA);
367             }
368         }
369         if (IsAmperimetroTerminal(terminalB))
370         {
371             AmperimetroController amperimetro =
                  GetAmperimetroController(terminalB);
372             if (amperimetro != null)
373             {
374                 amperimetro.DesconectarTerminal(terminalB);
375             }
376         }
377     }
378
379     private AmperimetroController GetAmperimetroController(Transform
          terminal)
380     {
381         if (terminal == amperimetroController1.terminalPositivo ||
              terminal == amperimetroController1.terminalNegativo)
382         {
383             return amperimetroController1;
384         }
```

```
385         if (terminal == amperimetroController2.terminalPositivo ||
              terminal == amperimetroController2.terminalNegativo)
386         {
387             return amperimetroController2;
388         }
389         if (terminal == amperimetroController3.terminalPositivo ||
              terminal == amperimetroController3.terminalNegativo)
390         {
391             return amperimetroController3;
392         }
393         return null;
394     }
395
396     private void UpdateWireConnections()
397     {
398         foreach (var connection in wireConnections)
399         {
400             Transform terminalA = connection.Item1;
401             Transform terminalB = connection.Item2;
402
403             float potencialA = GetPotencialDoTerminal(terminalA);
404             float potencialB = GetPotencialDoTerminal(terminalB);
405
406             if (terminalA == variacController.terminalPositivo ||
                  terminalB == variacController.terminalPositivo)
407             {
408                 if (IsTransformadorTerminal(terminalA))
409                 {
410                     transformerController.ConectarTerminal(terminalA,
                      variacController.VPositivo);
411                 }
412                 if (IsTransformadorTerminal(terminalB))
413                 {
414                     transformerController.ConectarTerminal(terminalB,
                      variacController.VPositivo);
415                 }
416                 ConectarTerminalVoltimetros(terminalA,
                      variacController.VPositivo);
417                 ConectarTerminalVoltimetros(terminalB,
                      variacController.VPositivo);
418                 ConectarTerminalAmperimetros(terminalA, terminalB,
                      variacController.VPositivo);
419             }
420             else if (terminalA == variacController.terminalNegativo ||
                  terminalB == variacController.terminalNegativo)
421             {
422                 if (IsTransformadorTerminal(terminalA))
423                 {
424                     transformerController.ConectarTerminal(terminalA,
                      variacController.VNegativo);
425                 }
426                 if (IsTransformadorTerminal(terminalB))
427                 {
```

```
428                     transformerController.ConectarTerminal(terminalB,
                    variacController.VNegativo);
429                 }
430                 ConectarTerminalVoltimetros(terminalA,
                    variacController.VNegativo);
431                 ConectarTerminalVoltimetros(terminalB,
                    variacController.VNegativo);
432                 ConectarTerminalAmperimetros(terminalA, terminalB,
                    variacController.VNegativo);
433             }
434             else
435             {
436                 // Atualizar conexões entre terminais do transformador
                    e voltímetro/amperímetro
437                 if (IsTransformadorTerminal(terminalA))
438                 {
439                     if (IsVoltimetroTerminal(terminalB))
440                     {
441                         voltimetroController1.ConectarTerminal
                    (terminalB, potencialA);
442                         voltimetroController2.ConectarTerminal
                    (terminalB, potencialA);
443                         voltimetroController3.ConectarTerminal
                    (terminalB, potencialA);
444                     }
445                     if (IsAmperimetroTerminal(terminalB))
446                     {
447                         ConectarTerminalAmperimetros(terminalB,
                    terminalA, potencialA);
448                     }
449                 }
450                 if (IsTransformadorTerminal(terminalB))
451                 {
452                     if (IsVoltimetroTerminal(terminalA))
453                     {
454                         voltimetroController1.ConectarTerminal
                    (terminalA, potencialB);
455                         voltimetroController2.ConectarTerminal
                    (terminalA, potencialB);
456                         voltimetroController3.ConectarTerminal
                    (terminalA, potencialB);
457                     }
458                     if (IsAmperimetroTerminal(terminalA))
459                     {
460                         ConectarTerminalAmperimetros(terminalA,
                    terminalB, potencialB);
461                     }
462                 }
463             }
464         }
465
466         // Atualizar tensões em tempo real
467         transformerController.AtualizarTensoes();
```

```
468     }
469 }
470
471
```

ANEXO C

```
using UnityEngine;

public class VoltimetroController : MonoBehaviour
{
    public Transform terminalPositivo;
    public Transform terminalNegativo;

    private float tensaoTerminalPositivo;
    private float tensaoTerminalNegativo;

    public float TensaoVoltimetro
    {
        get { return Mathf.Abs(tensaoTerminalPositivo -
          tensaoTerminalNegativo); }
    }

    public void ConectarTerminal(Transform terminal, float tensao)
    {
        if (terminal == terminalPositivo)
        {
            tensaoTerminalPositivo = tensao;
        }
        else if (terminal == terminalNegativo)
        {
            tensaoTerminalNegativo = tensao;
        }
    }

    public void DesconectarTerminal(Transform terminal)
    {
        if (terminal == terminalPositivo)
        {
            tensaoTerminalPositivo = 0f;
        }
        else if (terminal == terminalNegativo)
        {
            tensaoTerminalNegativo = 0f;
        }
    }
}

```