Pier Luigi Nakai Ricchetti

Blender Recommender System: Um sistema inteligente voltado para o ensino de computação gráfica

São Paulo, SP

2023

Pier Luigi Nakai Ricchetti

Blender Recommender System: Um sistema inteligente voltado para o ensino de computação gráfica

Trabalho de conclusão de curso apresentado ao Departamento de Engenharia de Computação e Sistemas Digitais da Escola Politécnica da Universidade de São Paulo para obtenção do Título de Engenheiro.

Universidade de São Paulo – USP

Escola Politécnica

Departamento de Engenharia de Computação e Sistemas Digitais (PCS)

Orientador: Prof. Dr. Ricardo Nakamura, Prof. Dr. Fabrizio Lamberti, Prof. Alberto Cannavò

São Paulo, SP

2023

Catalogação-na-publicação

Ricchetti, Pier Luigi Nakai
Blender recommender system: um sistema inteligente voltado para o ensino de computação gráfica / P. L. N. Ricchetti -- São Paulo, 2023.
52 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Computação e Sistemas Digitais.

1.Blender 2.Computação Gráfica 3.Sistema de recomendação 4.Sistema de logging I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Computação e Sistemas Digitais II.t.

Este trabalho é dedicado àqueles que ousam sonhar, aos que persistem diante das adversidades e aos que continuam acreditando no poder transformador do conhecimento.

Agradecimentos

Gostaria de expressar minha sincera gratidão a todos que contribuíram para a realização deste trabalho, tornando esta jornada acadêmica uma experiência enriquecedora e significativa.

Aos meus orientadores, prof. Dr. Ricardo Nakamura, prof. Dr. Fabrizio Lamberti e prof. Alberto Cannavò, que não apenas guiaram este trabalho com expertise e paciência, mas também me inspiraram a alcançar padrões mais elevados de excelência acadêmica. Suas orientações foram fundamentais para o desenvolvimento desta monografia.

À Escola Politécnica da USP, agradeço pela oportunidade de ingressar no Politécnico di Torino pelo programa de Dupla Graduação, uma experiência que enriqueceu profundamente minha formação, não só acadêmica, mas também pessoal. Ambas as instituições desempenharam papéis cruciais no meu percurso acadêmico, oferecendo suporte, desafios e oportunidades únicas. Agradeço sinceramente por essa colaboração acadêmica enriquecedora.

À minha amada família, cujo amor incondicional, compreensão e apoio constante foram a força motriz por trás dessa jornada. Agradeço por serem minha fonte de inspiração e por compartilharem comigo os desafios e triunfos desta jornada acadêmica.

Este trabalho não teria sido possível sem o apoio e contribuições valiosas de cada um de vocês. Estou profundamente grato por fazerem parte desta jornada.

Resumo

Este trabalho relata o desenvolvimento da primeira fase de um sistema inteligente de recomendação a ser desenvolvido para o Blender, cujo trabalho completo será apresentado como tese (*Master's Thesis*) na instituição Politecnico di Torino, com o objetivo de otimizar a experiência de aprendizado de estudantes de computação gráfica que fazem o uso deste programa; um *software* de modelagem 3D *open-source* amplamente utilizado. Essa primeira parte do projeto dedica-se a estabelecer uma arquitetura de base sólida, modular e escalável a fim de capturar, traduzir e processar eficientemente ações de usuários no Blender. A segunda fase de desenvolvimento explorará algoritmos avançados utilizando as ações do usuário capturadas para fornecer recomendações relevantes dentro do ambiente do Blender através da implementação de um sistema de recomendação. Assim, a arquitetura desenvolvida neste trabalho estabelece uma base sólida, robusta, modular e escalável para avanços, futuramente, deste sistema de recomendação. O sistema descrito não apenas demonstrou sua capacidade de capturar efetivamente as operações do usuário de forma transparente e confiável, como também, dada a sua praticidade de instalação sem a necessidade de configurações iniciais, a sua viabilidade e potencial para desenvolvimentos complexos. As áreas identificadas para aprimoramento, principalmente a expansão das operações reconhecidas e a detecção de ações que ainda podem ter impactos significativos nos resultados dos registros, oferecem direções claras para futuras melhorias. Assim, a arquitetura proposta atende não apenas às atuais necessidades do projeto, mas também fornece uma base para trabalhos futuros. Sua modularidade e integração profunda com a API do Blender a tornam uma solução versátil, permitindo o desenvolvimento de sistemas com funcionalidades semelhantes em cenários de modelagem 3D.

Palavras-chave: blender. python. sistema de recomendação. computação gráfica. modelagem 3D. educação. open-source.

Abstract

This work reports on the development of the first phase of an intelligent recommendation system to be further expanded for Blender, with the complete work to be presented as a Master's Thesis at the Politecnico di Torino. The primary goal is to optimize the learning experience for computer graphics students using this widely utilized open-source 3D modeling software. This initial project phase focuses on establishing a solid, modular, and scalable architecture to efficiently capture, translate, and process user actions within Blender. The subsequent development phase will explore advanced algorithms, utilizing the captured user actions to provide relevant recommendations within the Blender environment through the implementation of the recommender system. The architecture developed in this work lays a robust, modular, and scalable foundation for future advancements of this recommendation system. The described system not only demonstrated its ability to transparently and reliably capture user operations but also, due to its user-friendly installation without the need for initial configurations, proved its feasibility and potential for complex developments. Identified areas for improvement, particularly the expansion of recognized operations and the detection of actions that may still have significant impacts on log results, provide clear directions for future enhancements. Therefore, the proposed architecture not only meets the current project needs but also provides a groundwork for future endeavors. Its modularity and deep integration with the Blender API make it a versatile solution, enabling the development of systems with similar functionalities in 3D modeling scenarios.

Keywords: blender. python. recommendation system. computer graphics. 3D modeling. education. open-source.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

API	Application Programming Interface
PoliTo	Politecnico di Torino

Sumário

1 Introdução

Nos últimos anos, a demanda por habilidades em computação gráfica tem experimentado um notável crescimento, refletindo a importância cada vez maior dessa área no cenário tecnológico atual. Setores como entretenimento, design, mercado financeiro, publicidade e engenharia impulsionam a necessidade de profissionais qualificados, resultando em uma demanda cada vez maior por ensino nesse campo.

No entanto, o aprendizado dessa área pode ser muito difícil para alguns estudantes dada a complexidade dos conceitos envolvidos na área. Além disso, disciplinas que envolvem a modelagem tridimensional exigem que os alunos tenham uma noção espacial considerável e conhecimento (mesmo que mínimo) de algumas operações e transformações em três dimensões, conceitos que nem sempre são triviais e podem levar algum tempo para serem dominados.

Nesse contexto, surge a necessidade de abordagens mais inovadoras e escaláveis a fim de atender à crescente demanda educacional e às dificuldades envolvidas no aprendizado de programas de modelagem tridimensional (HE; ZHAO, 2012). O Blender Recommender System, um projeto único nascido do trabalho de tese (Master's Thesis) do aluno Pier Luigi Nakai Ricchetti em conjunto com a Escola Politécnica da USP e a Politecnico di Torino, orientado pelos professores drs. Ricardo Nakamura (Poli-USP), Fabrizio Lamberti (PoliTo) e pelo professor Alberto Cannavò (PoliTo), visa redefinir a maneira como os estudantes abordam cursos de computação gráfica usando o Blender - um popular *software open-source* utilizado para modelagem 3D, animação, renderização, composição, edição de vídeo e outras tarefas relacionadas à criação de conteúdo visual, usado tanto por artistas iniciantes quanto por profissionais em diversas áreas, como jogos, filmes, animação, design e engenharia – mediante a identificação dessas dificuldades comuns entre os alunos.

Este sistema inovador visa utilizar o poder da inteligência artificial, criando um agente virtual inteligente como um complemento em Python para o Blender a fim de ajudar os estudantes em sua jornada de aprendizado, fornecendo tutoriais personalizados, detectando erros e oferecendo recomendações valiosas e inteligentes. De fato, diversos estudos apontam para a eficacia deste tipo de sistema no aprendizado e o seu grande potencial na área de educação, como: (SHAW et al., 2000), (HJERT-BERNARDI; MELERO; HERN'NDEZ-LEO, 2012), (SCHROEDER; ADESOPE; GILBERT, 2013) e (RICKEL, 2001).

Dada a complexidade do trabalho, o projeto foi dividido em dois ciclos de desenvolvimento: o primeiro terá como objetivo o desenvolvimento da arquitetura do sistema bem como o da lógica de captura de ações e operações do usuário, enquanto que o segundo

terá como objetivo a implementação do *Recommender System* utilizando como base a arquitetura do primeiro ciclo.

O trabalho aqui desenvolvido é referente ao primeiro ciclo, a ser apresentado em formato de monografia para o trabalho de conclusão de curso da Escola Politécnica da USP. O trabalho completo (i.e., com o primeiro e o segundo ciclo completos) será apresentado como trabalho de tese do curso de engenharia de computação da Politecnico di Torino, concluindo assim, a dupla graduação do aluno.

1.1 Motivação

A motivação por trás do Sistema de Recomendação para o Blender surge do desejo de elevar a experiência educacional para estudantes envolvidos em cursos de computação gráfica que utilizam o Blender. Reconhecendo o amplo uso deste software de modelagem 3D, o projeto final busca facilitar a aprendizagem, implementando um sistema inteligente de recomendação. Ao criar uma arquitetura modular e escalável, o projeto visa atender às diversas necessidades dos usuários e estabelecer as bases para uma ferramenta educacional abrangente.

É relevante destacar que o estado da arte na pesquisa relacionada ao uso de *Recommender Systems* para casos de computação gráfica revela uma notável escassez de estudos abordando essa temática específica. Esta lacuna ressalta a necessidade de iniciativas inovadoras, como o Sistema de Recomendação para o Blender, que visa preencher esse vazio de conhecimento e proporcionar uma abordagem única para aprimorar a aprendizagem de computação gráfica, já que o Blender tem o potencial de auxiliar no aprendizado de conceitos importantes desta área (KADAM et al., 2013). Algumas pesquisas abordaram jeitos alternativos de aprimorar o ensino de computação gráfica, como por exemplo (YUAN; XU; ZHAO, 2010) e (LOWTHER; SHENE, 2000) mas nenhuma delas considera a possibilidade de um agente virtual. A falta de pesquisas existentes destaca ainda mais a relevância e a originalidade do projeto, que busca não apenas melhorar a experiência do usuário no Blender, mas também contribuir significativamente para o avanço do conhecimento nesse campo específico.

1.2 Objetivos

O objetivo desta primeira parte do projeto é o de desenvolver uma solução que permita que as ações de um usuário sejam identificadas e processadas em tempo real de forma rápida, eficiente e transparente dentro do Blender, construindo para isso uma arquitetura que seja sólida, modular e escalável. Devido a essas características, o sistema a ser desenvolvido se assemelha muito a um *logger*: uma ferramenta ou componente utilizado

para registrar informações relevantes durante a execução de um programa ou sistema, que neste caso serve para rastrear atividades e monitorar as ações do usuário.

Com estes avanços, espera-se obter uma base sólida para que a segunda parte do projeto possa ser desenvolvida, relacionada ao desenvolvimento do sistema de recomendação, que fará o uso desta arquitetura e das informações processadas pelo *logger* aqui desenvolvido a fim de fornecer respostas e recomendações inteligentes aos usuários do sistema.

O trabalho final completo, portanto, tem como objetivo aprimorar a experiência de aprendizado para estudantes de computação gráfica que utilizam o Blender, o que será alcançado por meio da personalização do aprendizado, oferecendo orientação específica, simplificando a criação de tutoriais e promovendo um ambiente de aprendizado mais interativo e envolvente.

1.3 Justificativa

O Sistema de Recomendação para o Blender possui o potencial de melhorar significativamente a experiência educacional para estudantes e professores de computação gráfica. O Blender é uma ferramenta proeminente nesse domínio, e ao introduzir um sistema inteligente de recomendação, o projeto visa fornecer orientação personalizada, simplificar a criação de tutoriais para educadores e, em última análise, promover um ambiente de aprendizado mais eficiente e envolvente. A escassez de pesquisas relacionadas ao uso de Recommender Systems para casos de computação gráfica enfatiza a importância de iniciativas inovadoras, como esta.

Esta primeira parte do trabalho desenvolvido é, por conseguinte, de igual importância, pois trata do desenvolvimento de toda a arquitetura e de seus componentes, bem como da implementação do *logger* do sistema, que servirá de base para a elaboração da segunda parte deste trabalho: o sistema de recomendação propriamente dito.

1.4 Organização do Trabalho

A organização desta monografia compreende seis capítulos distintos: Introdução, Definição de Termos, Método do Trabalho, Especificação de Requisitos, Desenvolvimento do Trabalho e Considerações Finais. Alguns desses capítulos estão divididos em seções e outros também em subseções.

O primeiro capítulo (Introdução) tem como propósito apresentar o tema desta monografia, a motivação por trás deste trabalho, contextualizar a relevância deste estudo, definir os seus objetivos e delinear a sua estrutura, fornecendo uma visão geral do que será abordado ao longo deste documento.

O capítulo 2 visa abordar os conceitos que são necessários ao entendimento deste documento pois são utilizados frequentemente, principalmente no capítulo 5 (Desenvolvimento do Trabalho), e encontra-se dividido em nove subseções, cada uma com a explicação do conceito em questão.

Já no capítulo 4, são descritos os requisitos do sistema para que o trabalho aqui descrito possa ser reproduzido, possuindo subseções descrevendo tanto os requisitos funcionais quanto os não funcionais.

No capitulo 5, é discutido o desenvolvimento do trabalho de forma detalhada e está dividido em cinco subseções principais: 5.1, que explica as tecnologias utilizadas no desenvolvimento, 5.2, que contém informações sobre a fase de estudo da ferramenta (Blender), 5.3 que explica o desenvolvimento da arquitetura do sistema, 5.4, que inclui mais oito subseções, cada uma referente a um componente da arquitetura e, por fim, 5.5, que descreve os testes, avaliações realizadas sobre o sistema e seus resultados, contendo mais três subseções, cada uma relacionada a um teste ou conjunto de testes diferente.

Finalmente, por último, o capítulo 6 que inclui as considerações finais e conclusões do autor sobre o projeto.

2 Definição de termos

2.1 Blender

O Blender é um software open-source que disponibiliza ferramentas para modelagem 3D, animação, renderização, simulação, composição, edição de vídeo e outras aplicações relacionadas à criação de conteúdo digital. O software também possui uma API descrita em python que permite que os usuários mais avançados possam customizar e desenvolver novas ferramentas para o aplicativo, que podem, inclusive, ser incluídas nas versões futuras (conferir a seção 2.9).

Uma dúvida comum que pode surgir principalmente em pessoas da área de engenharia é a de como o Blender e outros *softwares* similares diferem dos *softwares* do tipo CAD (*Computer-Aided Design*). Essa pergunta é válida pois a resposta ajuda a entender melhor como o Blender funciona.

O Blender, assim como outros aplicativos com o mesmo propósito, é um software de modelagem poligonal. Isso significa que os objetos são descritos por meio de uma malha composta por vértices, arestas e faces (conferir a seção 2.4). Um cilindro no Blender, por exemplo, não é de fato um cilindro perfeitamente redondo, mas sim uma composição de vértices, arestas e faces que, arranjados de forma específica, aproximam um cilindro. Assim, quanto maior a resolução geométrica, mais próxima do objeto em questão a malha fica (a custo, claro, de maior necessidade de processamento). Em *softwares* do tipo CAD, a abordagem é diferente pois os objetos neles descritos são na verdade equações matemáticas no espaço tridimensional (i.e., utiliza operações paramétricas e representações baseadas em sólidos reais). Desta forma, o usuário pode aproximar o quanto desejar da forma geométrica que esta nunca perderá a sua "resolução geométrica".

Justamente pelo fato de o Blender adotar uma abordagem poligonal e os aplicativos CAD usarem uma abordagem matemática, os *softwares* deste último tipo são utilizados com frequência em contextos que exigem precisão técnica (como engenharia, arquitetura, etc), enquanto que o primeiro é mais utilizado em contextos artísticos e mais versáteis, possuindo, comumente, tecnologias de renderização mais complexas para representações artísticas mais realistas.

2.2 Objeto ativo

No Blender, o conceito de "objeto ativo"refere-se ao objeto selecionado que está ativo para realizar operações ou modificações. Quando o usuário trabalha com múltiplos objetos

na cena 3D, ele pode selecionar vários ao mesmo tempo, mas apenas um é considerado ativo. O objeto ativo é geralmente destacado de alguma forma, como por uma cor diferente ou um contorno mais proeminente. No caso do Blender, o objeto ativo é indicado por uma cor de contorno mais clara.

Este conceito é importante pois algumas operações dentro do programa não podem afetar mais de um objeto por vez e, nestes casos, o objeto escolhido como alvo é sempre o ativo na cena.

2.3 Modo de objeto e modo de edição

O Blender oferece dois modos principais para interagir com os objetos: o "Modo de Objeto"e o "Modo de Edição".

No Modo de Objeto é possível selecionar, mover, escalar e rotacionar objetos como um todo. É ideal para operações que afetam o objeto como uma entidade única, ou seja, que não mudem de forma significativa a configuração relativa da malha tridimensional do objeto.

Já no Modo de Edição, é possível editar de forma detalhada os elementos dentro de um objeto, como vértices, arestas e faces. No Modo de Edição, o usuário pode realizar tarefas como modelagem, escultura e ajustes finos da malha. É neste modo portanto que de fato ocorre o processo de "modelagem"propriamente dito, em que o usuário pode criar, remover e mudar os vértices que compõe a estrutura tridimensional do objeto sem limitações.

2.4 *Mesh*

No Blender, uma "*mesh*"(ou malha) representa a estrutura tridimensional de um objeto e é composta por vértices, arestas e faces. A modelagem no Blender envolve a manipulação direta desses elementos para esculpir a forma desejada do objeto. Artistas podem adicionar, modificar e excluir vértices, arestas e faces, utilizando ferramentas de modelagem que incluem extrusão, subdivisão, deleção, etc. A versatilidade da malha permite a criação de uma ampla gama de objetos, desde simples formas geométricas até modelos complexos de personagens e cenários. Quando renderizado, o Blender pode transformar essa representação em imagens realistas, possibilitando a produção de animações, jogos e efeitos visuais.

2.5 Operações

Operações no Blender referem-se a ações que o usuário executa no software para realizar tarefas específicas. Isso pode incluir a criação de objetos, manipulação de geometria, aplicação de efeitos, entre outras ações. Operações também podem ser realizadas através de atalhos de teclado, menus ou painéis específicos no ambiente do Blender.

Como elas representam ações, muitas das operações possuem valores que podem ser escolhidos pelo próprio usuário durante as suas execuções. Por exemplo, a operação de mover um objeto de um ponto A para um ponto B envolve computar os valores de translação no formato x, y e z do objeto em questão.

2.6 Modificadores

No Blender, os "modifiers"(ou modificadores) são ferramentas que permitem aos usuários aplicar transformações não destrutivas em objetos. Esses modificadores são empregados para alterar a geometria, aparência ou comportamento de uma *mesh* (2.4) sem modificar diretamente sua estrutura original, que só ocorre quando o usuário decide aplicar as alterações. Cada modificador possui funcionalidades específicas, como suavização, subdivisão, deformação, entre outras. A aplicação deles é flexível e permite aos artistas iterar facilmente nas alterações, desativando ou ajustando os modificadores conforme necessário de forma dinâmica. Essa abordagem não destrutiva é valiosa para experimentação e refinamento de projetos no Blender, proporcionando maior controle e eficiência no processo de modelagem das malhas.

2.7 *Modal Operator*

Um "*Modal Operator*"no Blender se refere a uma operação que permanece ativa e aguarda alguma entrada do usuário enquanto está em execução. Com as informações dos *inputs* realizados pelo usuário, esse operador pode implementar alguma lógica específica relacionada à essas entradas. Essas operações, portanto, são "modais"porque alteram o comportamento do Blender até que o usuário conclua a operação ou a cancele. Um exemplo comum é a operação de transformação (mover, escalar, rotacionar) que continua ativa até que o usuário confirme a transformação ou a cancele.

Segundo a própria API do Blender, esse tipo de operador é adequado para a criação de ferramentas interativas dentro do *software* justamente por continuar a sua execução continuamente até que seja sinalizado a parar.

2.8 Contextos

No Blender, "contextos"referem-se ao ambiente ou situação específica em que o software está operando. Diferentes contextos podem ter diferentes configurações e opções disponíveis. Por exemplo, o contexto pode mudar quando o usuário está no Modo de Objeto versus o Modo de Edição. Conhecer o contexto em que o usuário está ajuda a entender quais operações são aplicáveis em determinadas situações, uma vez que as operações têm contextos específicos em que podem ser executadas.

É por essa razão que, na seção 5.4.3, uma "substituição de contexto"é necessária, pois assim a operação que se deseja executar fica disponível no Blender.

2.9 *Addons*

No Blender, os *addons* são módulos adicionais que estendem as capacidades do software, proporcionando funcionalidades e ferramentas extras por meio de *scripts* em Python que se comunicam diretamente com a API do Blender. Esses *addons* possibilitam uma personalização significativa, permitindo que os usuários escolham, ativem e desenvolvam ferramentas específicas de acordo com suas necessidades e preferências.

Comumente são desenvolvidos pela comunidade e disponibilizados para *download*, abrangendo uma variedade de áreas; desde automação de tarefas até integração com softwares externos. Sua instalação e ativação são processos simples, oferecendo aos usuários a flexibilidade de adaptar o Blender ao seu fluxo de trabalho específico. Essa modularidade contribui para a versatilidade do Blender, capacitando os usuários a aprimorar suas experiências de modelagem, animação e renderização, além de promover uma comunidade ativa de desenvolvimento e compartilhamento de *addons*.

3 Método do trabalho

A metodologia utilizada durante a maior parte das fases de desenvolvimento deste trabalho foi a conhecida como "ScrumBan". O ScrumBan é uma metodologia ágil que permite a escolha de características tanto do Scrum quanto do Kanban (ALQUDAH; RAZALI, 2018), podendo por exemplo considerar a utilização do quadro do primeiro para a visualização do fluxo de trabalho, mantendo iterações curtas chamadas *Sprints*, e a incorporação de elementos de priorização contínua, oferecendo uma abordagem potencialmente flexível que combina a estrutura do primeiro com a adaptabilidade do segundo, sendo adequado para ambientes que demandam previsibilidade e flexibilidade.

Para isso, utilizou-se uma ferramenta *online* chamada "Notion"que permite organizar projetos de forma personalizada. A figura 1 mostra uma captura de tela de um quadro Kanban utilizado na fase de desenvolvimento da arquitetura do sistema. A primeira coluna contém todas as atividades que ainda não haviam sido iniciadas, a do meio contém as que estavam em andamento e a última contém todas as que já haviam sido concluídas no momento da captura. Ao final de toda a semana, uma atualização dos avanços era discutida e novos objetivos eram definidos para a próxima semana (ou ciclo de desenvolvimento).

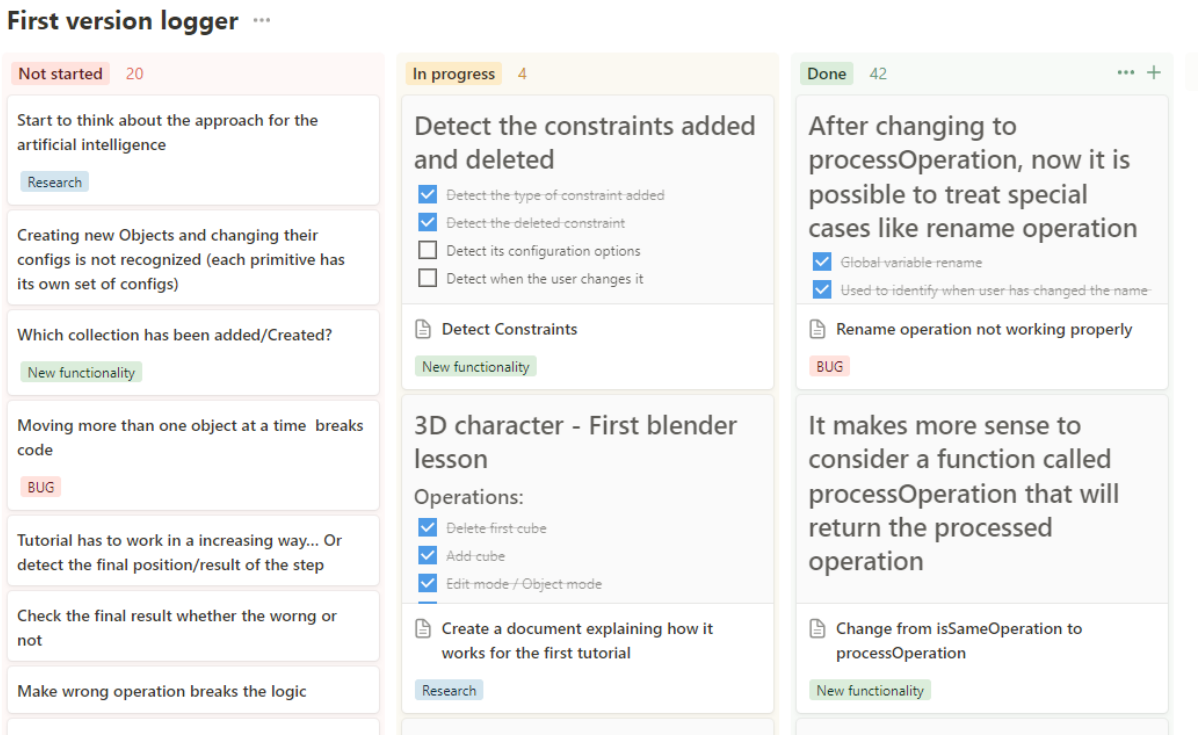

Figura 1 – Captura de tela do quadro Kanban utilizado durante a fase de desenvolvimento na ferramenta Notion

O desenvolvimento do trabalho começou com uma fase de pesquisa a fim de se entender o estado da arte das tecnologias de sistemas de agentes inteligentes. Como a decisão de dividir o desenvolvimento em duas partes ainda não havia sido tomada, as pesquisas se concentraram majoritariamente em torno deste tema e não da elaboração do *logger* em si, objetivo desta primeira fase de desenvolvimento do trabalho. Por conta disso, o capítulo de Desenvolvimento do Trabalho (5) não inclui esta seção.

Passada a fase de pesquisa, iniciou-se a fase de estudo da ferramenta. Mais especificamente, o Blender (ferramenta explicada na seção 2.1). Nesta fase, o foco foi entender como o objetivo do trabalho poderia ser alcançado através da API do programa, ou seja, envolveu o estudo da documentação do Blender a fim de se entender quais ferramentas e métodos poderiam ser utilizados para resolver o problema proposto. Isso também incluiu o estudo de *addons* (conceito explicado na seção 2.9) criados pela comunidade a fim de verificar se alguma solução já não existia.

Com um maior entendimento sobre o funcionamento interno do Blender e a sua API, iniciou-se a fase de desenvolvimento da arquitetura do *logger* e de seus componentes. Durante esta fase, a metodologia ScrumBan discutida anteriormente foi usada, o que permitiu uma flexibilidade dos avanços bem como uma visão geral do andamento do projeto.

Por fim, a fase de testes e avaliações teve parte do seu período interconectado com a fase anterior, dado que, ao final de determinados *sprints*, alguns testes e avaliações eram realizados a fim de validar a qualidade do que havia sido desenvolvido. Para os testes que envolveram números, como por exemplo o teste de performance abordado na seção 5.5.1, realizou-se uma análise quantitativa através da média de pelo menos cem ocorrências seguidas de um arredondamento em duas casas decimais. Já para os testes que não envolveram números, como por exemplo o teste abordado na seção 5.5.3, uma análise mais qualitativa foi realizada, analisando-se a resposta do sistema tendo em vista o seu funcionamento teórico.

4 Especificação de Requisitos

4.1 Introdução

O propósito deste capítulo é delinear as especificações detalhadas e requisitos para o desenvolvimento da arquitetura e do *logger* do Sistema de Recomendação do Blender, garantindo uma compreensão clara das funcionalidades e recursos esperados em cada componente do sistema.

4.1.1 Escopo

O Sistema de Recomendação do Blender visa otimizar a experiência de aprendizado para estudantes de computação gráfica que utilizam o Blender. Este documento foca especificamente nos requisitos para a primeira fase do projeto, que inclui o desenvolvimento de uma arquitetura modular e escalável, abrangendo vários componentes (explicados em detalhes no capítulo 5), como Modal Operator, Tutorial, Utils, Cache, Operations, UI, IO, e estabelecendo as bases para o Componente de Recomendação.

4.2 Requisitos Funcionais

A arquitetura do sistema desenvolvido, como será visto na seção 5.3, foi dividida em diversos componentes modulares que implementam funções específicas e bem delimitadas. Nas subseções seguintes, os requisitos funcionais de cada componente serão descritos. Para se obter uma visão geral da organização do sistema e facilitar o entendimento destes requisitos, recomenda-se conferir a figura 2 que se encontra no capítulo 5.

4.2.1 Requisitos Componente Modal Operator

- Interação Contínua com o Usuário

O Componente Modal Operator deve utilizar os operadores modais do Blender (2.7) para capturar e rastrear continuamente as entradas do usuário, incluindo movimentos e cliques do mouse, garantindo o monitoramento em tempo real das ações do usuário.

4.2.2 Requisitos Componente Tutorial

- Salvar no log, carregar tutoriais predefinidos e calcular progresso

O componente deve ter a capacidade de salvar todas as ações do usuário no log do sistema como uma série de operações registradas. Os dados registrados devem representar com precisão as etapas tomadas pelo usuário no ambiente Blender.

O componente deve também suportar o carregamento de tutoriais predefinidos, validando as ações do usuário em relação às etapas do tutorial. A porcentagem de progresso deve ser calculada e exibida.

4.2.3 Requisitos Componente Utils

- Processar Operações
- Identificar Novas Ações do Usuário
- Obter informações sobre objetos

O Componente Utils deve requisitar a tradução e processar as operações para uma linguagem universal, garantindo consistência e compatibilidade entre diferentes componentes do sistema. Além disso, o Componente Utils deve também determinar se uma operação capturada pelo "Modal Operator"representa uma nova ação ou operação do usuário, evitando redundância desnecessária nos dados e ser capaz de obter informações sobre os objetos da cena a serem utilizados pelo cache do sistema.

4.2.4 Requisitos Componente Cache

- Atualizações Dinâmicas e dicionário estruturado

O Componente Cache deve atualizar dinamicamente os objetos da cena através da utilização de um dicionário estruturado, fornecendo uma representação precisa e em tempo real das suas propriedades, incluindo escala, localização, rotação e vértices.

4.2.5 Requisitos Componente Operations

- Lidar de forma personalizada com várias operações
- Saída de operações em Linguagem Universal (tradução)

O Componente Operations deve abrigar funções correspondentes as várias operações, lidando efetivamente com operações de valor único, operações com vários parâmetros e operações complexas compostas de várias etapas, traduzindo-as para o formato universal utilizado pela arquitetura de acordo com cada especificidade.

4.2.6 Requisitos Componente UI

- Exibir Botões

O Componente UI deve implementar a lógica para exibir botões na interface do Blender, incluindo um para iniciar o sistema e outro para o finalizar.

4.2.7 Requisitos Componente IO

- Salvar Dados de Log

O Componente IO deve ser responsável por lidar eficientemente com o salvamento dos dados de log em formato txt, garantindo acessibilidade e armazenamento da maneira desejada.

4.2.8 Requisitos Componente de Recomendação

- Espaço para Desenvolvimento Futuro

O Componente de Recomendação, embora não implementado nesta fase, deve ser considerado para desenvolvimento futuro e integração com os componentes existentes.

4.3 Requisitos Não Funcionais

4.3.1 Detecção de Ação do Usuário

Os requisitos não funcionais relacionados à detecção de ações do usuário para este trabalho são:

- Rastreamento Preciso
- Desempenho
- Transparência

O sistema deve ser capaz de detectar e rastrear precisamente as ações do usuário, garantindo exatidão e prevenindo detecção duplicada, demonstrando desempenho eficiente, além de ser capaz também de lidar com diversas operações sem comprometer a responsividade e transparência do sistema para o usuário.

4.3.2 Arquitetura como um todo

Para a arquitetura, os requisitos não funcionais são:

- Escalabilidade
- Capacidade de manutenção
- Desempenho
- Confiabilidade
- Integração profunda com a API do Blender 3.2.2
- Desenvolvimento em Python

O sistema desenvolvido deve possuir uma arquitetura que ofereça escalabilidade e capacidade de manutenção, uma vez que constantemente são adicionadas, modificadas ou removidas operações em atualizações do Blender. Além disso, o desempenho e a confiabilidade da solução desenvolvida devem sempre ser prioridade justamente por se tratar de um *logger*, ou seja, está continuamente sendo executado em paralelo ao Blender.

Como a implementação da arquitetura é feita em formato de *addon* (conceito explicado em 2.9), ela deve possuir uma integração profunda com a API do Blender (o que, inclusive, interfere em seu desempenho) e por isso ser desenvolvida em Python e deve considerar a versão 3.2.2, pois foi a versão escolhida para o desenvolvimento do trabalho.

5 Desenvolvimento do Trabalho

5.1 Tecnologias Utilizadas

Neste trabalho, toda a arquitetura do sistema foi desenvolvida para ser lida pelo próprio Blender como um *addon* (2.9) a ser instalado pelos usuários. Portanto, utilizou-se Python como a linguagem de programação - já que é a linguagem oficial da API do Blender - e, claro, o próprio Blender.

A fim de auxiliar na programação, o software "Visual Studio Code"foi utilizado, pois torna a depuração de código muito mais fácil e intuitiva do que o próprio editor de texto do Blender, enquanto que para a criação de diagramas a ferramenta online "Diagrams.net"foi utilizada.

5.2 Estudo da ferramenta

Como discutido no capítulo 3, a fase de pesquisa se concentrou no tema de agentes virtuais inteligentes a serem utilizados como uma ferramenta de auxílio ao aprendizado. Ficou claro durante esta primeira fase que a solução mais adequada ao problema que o trabalho completo deseja resolver seria a de utilizar um *Recommender System*. No entanto, para que isso fosse possível, seria necessário antes desenvolver um método que permitisse o *logging* das operações dentro do Blender (software de modeagem 3D, explicado em detalhes na seção 2.1) já que o *Recommender System* necessita de dados extraídos do usuário.

Assim, surgiu a ideia de se dividir o trabalho em duas fases de desenvolvimento: a primeira relacionada à implementação da arquitetura do sistema sobre a qual o sistema de recomendação irá trabalhar, bem como implementar o *logger*, e a segunda relacionada ao desenvolvimento do sistema de recomendação propriamente dito.

Com isso, iniciou-se então a fase de estudo do Blender e as suas ferramentas de desenvolvimento a fim de se obter conhecimento de suas capacidades. Para isso, a API ofifical do Blender foi extensamente consultada e estudada, já que, para o desenvolvimento de *addons* (explicação na seção 2.9), é necessário fazer o seu uso.

Além disso, também foram estudados *addons* desenvolvidos pela própria comunidade a fim de entender se alguma solução para um *logger* do Blender já não havia sido feita. De fato, muitos desses códigos encontrados haviam implementações que poderiam ser tomadas como base para o desenvolvimento inicial do sistema, mas nenhuma delas apresentava a complexidade necessária para este trabalho como um todo. Ainda, devido à dificuldade de se encontrar pesquisas relacionadas a um sistema de *logging* para cenários de computação

gráfica, toda a arquitetura do sistema teve que ser desenvolvida por completo, desde a sua concepção mais ampla até os mínimos detalhes da implementação de cada um dos seus componentes e a resolução de seus eventuais problemas. O desenvolvimento da arquitetura está detalhada na próxima seção (5.3).

5.3 A arquitetura do sistema

Por se tratar de um sistema que deve ser capaz de identificar as ações do usuário de forma rápida, eficiente e transparente, a arquitetura do sistema deve ser organizada tendo estes objetivos como prioridade. No entanto, além desses objetivos, a arquitetura deve também proporcionar uma certa escalabilidade e flexibilidade para que atualizações e adições de novos recursos possam ser feitas. Isso se deve ao fato de que a API do Blender é frequentemente atualizada e novas operações podem eventualmente surgir, mudar ou deixar de existir. Levando todos estes requisitos em consideração, decidiu-se que a arquitetura seria dividida em diversos módulos que possuíssem funções bem delimitadas e que pudessem se comunicar entre si facilmente. Desta forma, a manutenção do sistema diante atualizações eventuais que o Blender possa receber fica mais intuitiva, melhorando a escalabilidade do sistema. O diagrama da arquitetura pode ser conferido na figura 2.

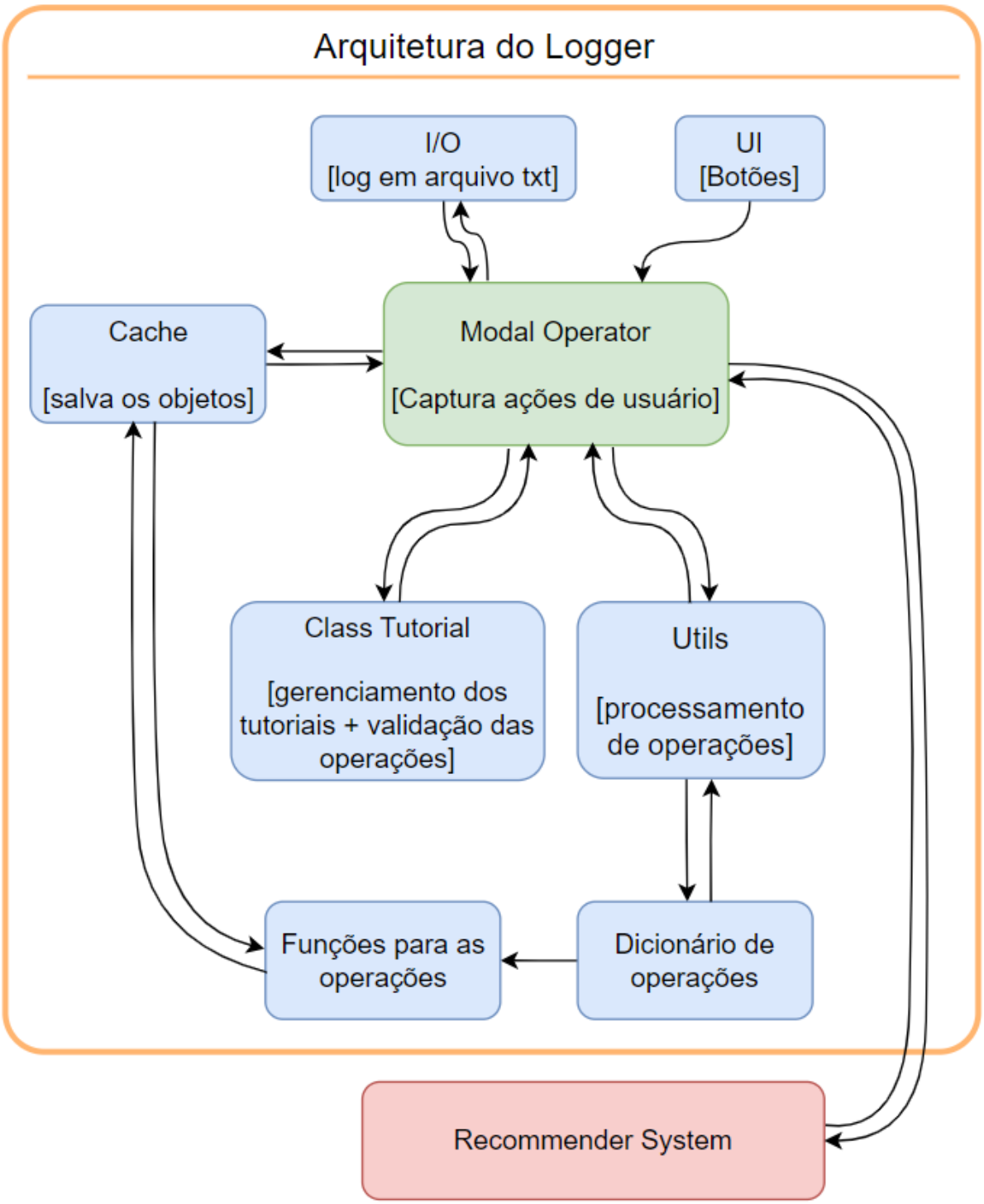

Figura 2 – Arquitetura do sistema de *logging* desenvolvida

Nota-se no diagrama da arquitetura desenvolvida que o componente "Recommender System"não assume uma posição central (o que, de primeira, pode parecer intuitivo dado que ele representa a inteligência do sistema), mas na verdade o componente "Modal Operator"é que o faz. Isso permite que uma ação detectada pelo sistema realizada pelo usuário seja rapidamente capturada, traduzida (como será visto em "Utils component"5.4.3) e

interpretada através da ativação rápida dos componentes necessários para tais processamentos. Se o componente de recomendação fosse o principal do sistema, o componente de captura ("Modal Operator") teria que enviar constantemente as suas detecções - que podem ou não representar ações do usuário - para o "Recommender"que só então deveria se conectar com os outros módulos para que o processamento das ações fosse realizada, ou seja, existiria uma camada a mais na comunicação, o que poderia prejudicar não só a eficiência do sistema, mas também a do próprio componente em questão.

É importante destacar que, futuramente, esta configuração de arquitetura pode mudar com o desenvolvimento da lógica do componente de recomendação (que será feita na segunda parte deste trabalho), mas, para o objetivo desta primeira parte, esta configuração representa a mais otimizada.

Nas subseções seguintes, cada módulo será explicado em detalhes.

5.4 Implementação dos componentes

5.4.1 Componente Modal Operator

Este componente, como mencionado anteriormente, representa a peça central do sistema e é o primeiro a ser "ativado"quando o usuário o inicia. A sua função principal é a de detectar a interação do usuário com o Blender através do movimento do mouse e de teclas pressionadas a fim de entender quando operações novas são feitas no Blender. Além disso, ele deve também solicitar a tradução da operação realizada ao componente "Utils"(5.4.3) da arquitetura a fim de salvar a operação no log do sistema, se comunicando para isso com o componente "Tutorial"do sistema e outros componentes que serão explicitados adiante.

Para que tudo isso seja possível, o componente deve considerar uma série de cenários e condições para a sua inicialização a fim de garantir seu correto funcionamento:

- Primeiramente, o componente inicializa duas variáveis internas - uma que salva a operação atual e outra que guarda uma referência ao componente "Tutorial"(5.4.2) para fins de comunicação - que serão utilizadas posteriormente para a tradução e para salvar no log as operações,
- Depois, o componente se comunica com o "Cache"(5.4.5) a fim de carregá-lo com todas as informações sobre os objetos presentes na cena no momento de sua inicialização, o que será útil para determinados tratamentos de operações. Aqui, é importante que o componente seja capaz de identificar se o usuário fez a inicialização do sistema no modo de edição ou de objeto (conceito explicado na seção 2.3) pois é necessário que todos os vértices do objeto, caso tenha se inicializado o sistema no modo edição, sejam salvos no cache para que as próximas operações feitas em seguida neste

modo possam usar com confiança os dados da configuração tridimensional da *mesh* (conceito explicado na seção 2.4) em questão,

- Em seguida, o componente atualiza uma variável global que contém o número de todas as operações realizadas anteriormente à inicialização do sistema. Esta informação será útil posteriormente para checar se houve ou não a realização de uma nova operação pelo usuário,

- Por fim, o componente deve ser capaz também de identificar se existe ou não algum objeto presente na cena, já que toda a lógica de detecção leva como premissa a existência de algum objeto sobre o qual a operação será realizada. Para isso, o componente considera um passo de segurança antes do início da detecção que consiste em verificar a existência de objetos na cena ou de um objeto ativo (2.2), permanecendo em *loop* até que esta condição seja modificada e exista algum objeto na cena.

Feita a inicialização correta do componente, este está pronto para realizar seu trabalho principal: detectar e processar de forma eficiente, rápida e transparente as operações realizadas pelo usuário dentro do Blender através da coordenação dos demais componentes.

Como comentado na seção 2.7 (aspectos conceituais), por se tratar de um *Modal Operator*, este componente é capaz de detectar continuamente *inputs* de usuário e executar uma lógica associada à estes eventos até que uma sinalização de parada seja executada, o tornando altamente adequado para o objetivo deste componente.

O primeiro passo para a implementação de sua lógica principal é justamente definir o conjunto de eventos relacionados aos *inputs* do usuário que devem ativar o seu processamento, já que, detectar a todo o momento qualquer *input*, seria altamente ineficiente. Tendo isso em vista, a combinação escolhida foi:

- **Todas as teclas e botões do mouse:** Escolhidos para serem detectados imediatamente pois representam interações que são utilizadas muito frequentemente para diversas operações dentro do programa. Além disso, muitas operações possuem *shortcuts* (combinações de teclas) que representam a sua execução. Portanto, se torna necessário o processamento imediato delas.

- **"INBETWEEN_MOUSEMOVE":** É um evento identificado pelo *Modal Operator* quando existe um movimento com o mouse que excede a velocidade de captura do evento "MOUSEMOVE"(que é detectado a cada tick do programa enquanto o mouse está em movimento). A razão pela qual o "MOUSEMOVE"não é utilizado e sim o "INBETWEEN_MOUSEMOVE"é justamente pelo fato de este último ser

detectado menos frequentemente do que o primeiro mas ainda em um taxa aceitável para que outras ações sejam identificadas a tempo. A necessidade da detecção do movimento do mouse será explicada adiante, já que envolve algumas considerações adicionais.

- **Exclusão do botão do meio do mouse e dos *timers*:** Assim como existem eventos que são necessários incluir, existem os eventos que devem ser excluídos pois são desnecessários para o escopo do projeto e poderiam se tornar uma potencial fonte de ineficiência. Este é o caso dos temporizadores internos do Blender e da detecção de *inputs* relacionada aos botões do meio do mouse. Portanto, estes eventos são ignorados pelo componente.

Pelo fato de o Blender oferecer diversas alternativas para que uma mesma operação seja realizada (seja por meio de *shortcuts* ou por meio de opções selecionáveis em sua rica interface) e de que estas teclas e botões do mouse capturados não permitem detectar todas as operações que possam ser executadas pelo usuário, surge a necessidade de desenvolver um método que permita essa detecção sem que para isso o componente tenha que processar desnecessariamente de forma contínua todos os *inputs* que usuário faz a cada *tick* de execução do programa. Assim, surge a ideia de usar o evento "INBETWEEN_MOUSEMOVE"como um *trigger* secundário do componente, o que oferece um interessante equilíbrio de performance uma vez que o seu acionamento não ocorre em todo movimento do mouse (o que poderia afetar a performance da solução) e sim em movimentos eventualmente mais abruptos. Desta forma, se um usuário realiza uma operação dentro do programa que não é imediatamente capturada pelo operador, assim que houver um movimento do mouse mais acelerado, o sistema será acionado e a ação será devidamente detectada. Outro ponto interessante desta solução é o de ela permite que mais um problema identificado seja resolvido: quando o usuário clica com o botão do mouse em algumas opções dentro da área de visualização 3D do Blender, o evento "LEFTMOUSE"não é detectado, mas observou-se que o "INBETWEEN_MOUSEMOVE"é sempre detectado. Assim, esta solução permite que estas operações sejam também detectadas imediatamente.

É importante destacar que assumir que um "movimento abrupto"do mouse irá eventualmente existir (após a realização de uma operação não detectada imediatamente pelo componente) é válido uma vez que, além de o Blender não detectar este evento apenas quando o movimento do mouse é realmente muito devagar, esta operação não detectada de imediato muito provavelmente terá sido executada através da interface do Blender (i.e., selecionando com o mouse alguma opção dentro da interface 3D) o que, como discutido anteriormente, sempre ativa o evento "INBETWEEN_MOUSEMOVE"e portanto será captado pelo componente imediatamente.

Após este filtro inicial dos *inputs* do usuário, o módulo envia a operação realizada,

extraída da própria API do Blender, para uma função no componente "Utils"(5.4.3) responsável por processar esta estrutura de dados e retornar a operação "traduzida"(a implementação da lógica de processamento das operações será explicada na seção do próprio componente) que então será utilizada para ser enviada ao componente "Tutorial"que irá salvar as operações como um log.

A estrutura interna deste componente pode ser conferida na figura 3 onde os retângulos pontilhados indicam outros componentes. No caso, "UI", como discutido anteriormente, é quem inicia o componente de fato, o qual mais adiante se comunica com os componentes "Utils"e "Tutorial".

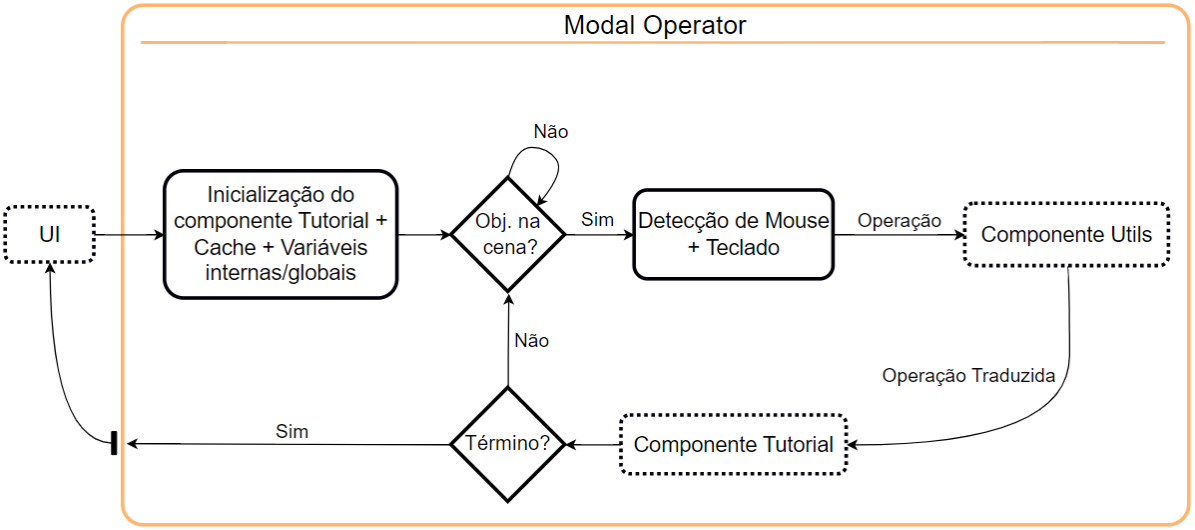

Figura 3 – Arquitetura interna do componente Modal Operator

5.4.2 Componente Tutorial

O principal objetivo deste componente é o de tratar de toda a lógica relacionada aos tutoriais que o usuário poderá carregar para utilizar o sistema. Para o desenvolvimento desta primeira parte do trabalho, este componente se comunica apenas com o "Modal Operator"(5.4.1), no entanto, futuramente irá certamente se comunicar com o "Recommender System"(5.4.8) já que as informações que são obtidas serão de extrema importância para a inteligência do sistema.

A função mais importante deste componente no estágio de desenvolvimento atual é o de receber do "Modal Operator"a operação realizada pelo usuário já processada e traduzida para que esta seja salva como log do sistema, através da utilização de uma informação que está disponível globalmente e que será utilizada pelo componente "IO"(5.4.7). Ao receber a operação já processada, esse componente também a salva em uma variável interna que possui o formato de um tutorial pré-carregado. Isso foi feito para permitir que, futuramente,

para facilitar a criação de tutoriais, o usuário possa "gravar"as operações realizadas em tempo real e transformá-las intuitivamente em um tutorial de forma automática. Aqui percebe-se a importância da tradução das operações para um formato comum, o que permite que elas sejam salvas e posteriormente lidas de forma bem mais eficiente e padronizada, possibilitando que comparações sejam feitas entre diversas operações.

Outras funções que este componente possui podem ser conferidas a seguir:

- **Carregar um tutorial:** Esta função permite que um tutorial seja carregado para ser utilizado no sistema. A estrutura de dados do tutorial segue o mesmo padrão traduzido das operações que a arquitetura usa, permitindo que a sequencia de operações salva como log possa ser utilizada como um tutorial dentro do próprio sistema, viabilizando assim, a leitura de uma sequência de operações capturadas pelo sistema em tempo real como um novo tutorial.
- **Obter próximo passo e validação do passo atual:** O componente também é capaz de guardar informações sobre o passo atual em que o usuário se encontra no tutorial e portanto pode também obter o próximo passo para avisá-lo. Além disso, o componente também é capaz de validar (ainda que de forma simples) a operação feita pelo usuário em relação à operação que deveria ser feita no passo atual do tutorial, o que é possível graças ao formato padrão utilizado pela arquitetura.
- **Obter Progresso:** Com as informações do passo atual e do número total de passos do tutorial carregado, o componente é capaz também de informar ao usuário sobre o seu progresso atual.

Como forma de provar que realmente o componente é capaz de validar (de forma simples para esta fase), carregar, obter a próxima etapa e obter o progresso de um tutorial, foram geradas instruções para a criação de uma pirâmide simples formada por cubos que envolvem operações básicas como translação e mudança de tamanho utilizando a própria captura destes passos pelo próprio sistema. A figura 4 mostra a captura de tela da pirâmide resultante do tutorial. Este tutorial pode ser carregado pelo usuário através de um botão da interface (renderizada pelo componente "UI"explicado na seção 5.4.6) e faz com que o sistema se comunique com o usuário atualmente pelo terminal do Blender (mas futuramente pela própria interface que será comandada pelo sistema de recomendação) inserindo informações sobre o próximo passo a ser executado, o progresso atual no tutorial e uma mensagem de erro quando o usuário erra um passo. Um exemplo desta comunicação pode ser vista na figura 5. Note que a comunicação é bem básica já que quem efetivamente se comunicará com o usuário de fato será o agente inteligente a ser desenvolvido no próximo ciclo do trabalho, bem como todo o processamento de tutoriais e o seu aprofundamento.

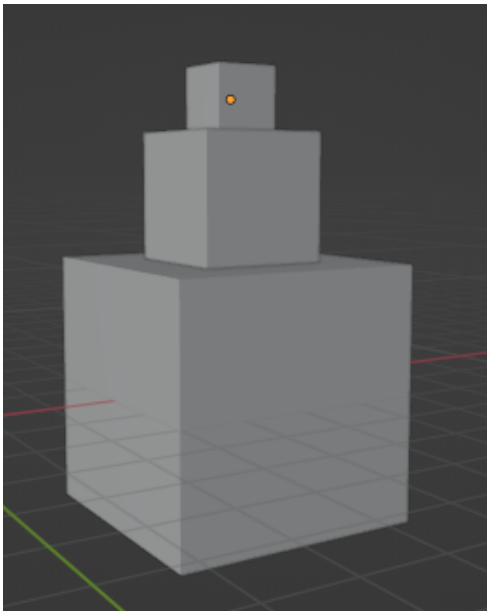

Figura 4 – Captura de tela da pirâmide gerada pelo tutorial de teste

```
============================ NEXT STEP: Perform the following operation:  ['Add', 'Cube', 'Cube']
========================================== TUTORIAL STARTED ==========================================
============================ Correct operation!
============================ Your progress:  10.0  %

============================ NEXT STEP: Perform the following operation:  ['Move', [5.0, 0.0, 0.0], 'Cube']
============================ Correct operation!
============================ Your progress:  20.0  %

============================ NEXT STEP: Perform the following operation:  ['Resize', [2.0, 2.0, 2.0], 'Cube']
============================ Correct operation!
============================ Your progress:  30.0  %

============================ NEXT STEP: Perform the following operation:  ['Add', 'Cube', 'Cube.001']
============================ Correct operation!
============================ Your progress:  40.0  %

============================ NEXT STEP: Perform the following operation:  ['Move', [0.0, 0.0, 3.0], 'Cube.001']
============================ WRONG OPERATION!,  ['Move', [3.0, 0.0, 0.0], 'Cube.001']
============================ Expected operation:  ['Move', [0.0, 0.0, 3.0], 'Cube.001']
============================ Correct operation!
============================ Your progress:  50.0  %

============================ NEXT STEP: Perform the following operation:  ['Move', [5.0, 0.0, 3.0], 'Cube.001']
============================ Correct operation!
============================ Your progress:  60.0  %

============================ NEXT STEP: Perform the following operation:  ['Add', 'Cube', 'Cube.002']
============================ Correct operation!
============================ Your progress:  70.0  %

============================ NEXT STEP: Perform the following operation:  ['Resize', [0.5, 0.5, 0.5], 'Cube.002']
============================ Correct operation!
============================ Your progress:  80.0  %

============================ NEXT STEP: Perform the following operation:  ['Move', [0.0, 0.0, 4.5], 'Cube.002']
============================ Correct operation!
============================ Your progress:  100  %

============================ NEXT STEP: Perform the following operation:  ['Move', [5.0, 0.0, 4.5], 'Cube.002']
============================ Tutorial Finished!
```

Figura 5 – Captura de tela do terminal da execução do tutorial de teste

É importante evidenciar que justamente pelo fato de este componente estar fortemente relacionado ao "Recommender System"(5.4.8), a sua implementação ainda está em

desenvolvimento e será completada em conjunto com ele na segunda fase de desenvolvimento do trabalho. Portando, atualmente, como dito anteriormente, a principal funcionalidade utilizada é a de salvar como log todas as operações capturadas pela arquitetura e não a de carregar e validar tutoriais.

5.4.3 Componente Utils

Este componente possui diversas funções que são utilizadas tanto pelo componente "Modal Operator"(5.4.1) quanto pelo componente "Operations"(5.4.4) do sistema, atuando na filtragem, processamento e tradução das operações, funções que um sistema típico de *logging* deve possuir (ALSPAUGH et al., 2014) para que seu *output* possa ser analisado eficientemente no futuro, além de possuir algoritmos para identificar informações sobre os objetos da cena.

Primeiramente, este componente é acionado pelo "Modal Operator"(quando um determinado *input* válido é identificado) e recebe uma estrutura de dados contendo diversas informações sobre a última operação feita pelo usuário. Por exemplo: se a operação realizada foi a de mover algo, esta estrutura de dados irá conter o nome interno da operação, o valor de translação realizado nos eixos x, y e z e outras muitas informações que podem ou não ser úteis para o *logger*. Neste caso, o objetivo do componente "Utils"é justamente processar essa estrutura de dados, filtrando as informações necessárias e transformando-as em valores que possam ser facilmente lidos por outros componentes do sistema, ou seja, atuando também como um tradutor das operações recebidas.

Entretanto, antes de começar todo o processamento das operações, o componente deve possuir um algoritmo capaz de identificar se a operação recebida é de fato uma nova operação ou se trata-se de uma operação repetida. Isso é necessário pois a detecção das interações do usuário são frequentemente capturadas múltiplas vezes de forma duplicada, uma vez que dado um *input* válido, o "Modal Operator"acessa a operação ativa (que contém sempre a última operação realizada) e a envia para o componente "Utils", sem detectar se a operação é realmente uma nova. Por exemplo: um clique de mouse é detectado três vezes pelo operador devido à forma de funcionamento interna do Blender. Assim, como a API do Blender não possui por *default* um método que permita diferenciar duas operações iguais mas que foram executadas duas vezes, torna-se necessário desenvolver um método para identificar quando houve realmente a execução de uma operação dentro do programa, caso contrário, o sistema processaria múltiplas vezes a mesma operação, o que o tornaria altamente ineficiente.

A interface do Blender é composta por diversas áreas dinâmicas (i.e., o usuário pode criar novas áreas, destruí-las, modificar os seus tamanhos etc), cada uma com suas propriedades e funções. Uma destas áreas é chamada de "Info", que contém o registro de todas as operações que foram realizadas, informações que o Blender deseja comunicar

para o usuário, algumas modificações que não são descritas como operações internamente no Blender mas que podem ser feitas pelo usuário e assim por diante. Utilizando as informações desta área, é possível filtrar as operações e finalmente inferir, através da comparação do número de ocorrências, se uma nova operação foi realizada, uma vez que, diferentemente do "Modal Operator", nesta área não existem operações duplicadas. No entanto, para que isso seja possível, a área deve sempre estar acessível na interface, ou seja, o usuário seria obrigado a operar no Blender sempre com esta área aberta, o que tornaria essa solução imprática. Para contornar este problema, utilizou-se um método chamado de substituição de contexto, que consiste em substituir uma propriedade do contexto (2.8) atual do Blender por outra que possua as informações que se deseja obter. Neste caso, a ideia é substituir uma área da interface pela área "Info"(caso esta não esteja presente) e depois re-substituir a mesma área pela anterior depois de todos os dados da área "Info"terem sido copiados para serem posteriormente filtrados. Para que este método funcione corretamente, alguns cuidados foram tomados:

- **Substituição da área ativa:** Um problema que surge naturalmente com esta solução é o de substituir uma área na qual o usuário encontra-se no momento. Isto é problemático pois como o método de substituição de contexto é utilizado, a área é rapidamente substituída por outra e depois pela mesma que estava anteriormente. Mesmo que imperceptível, isso já é suficiente para atrapalhar os *inputs* que o usuário venha a realizar na área em questão (já que o "Modal Operator"irá detectar e chamar novamente o componente "Utils"que irá novamente substituir a área e assim por diante). A solução para este problema foi implementar um algoritmo que utiliza as informações sobre a posição do mouse na interface, identificando sobre qual área o mouse encontra-se em cima e marcando esta área "ativa"como proibida de ser substituída. Desta forma, o problema é resolvido.

- **Só existe uma área:** Pode ocorrer (muito improvável) de o usuário trabalhar no Blender com apenas uma área na interface. Mesmo que seja raro, deve-se considerar este caso pois usuários iniciantes, por falta de conhecimento, podem acabar trabalhando desta forma acidentalmente. Como dito no ponto anterior, substituir a área "ativa"não é uma boa ideia e, portanto, neste caso, a solução foi a de justamente criar uma nova área na interface. Para que a experiência do usuário não seja afetada significativamente por essa solução, a área é criada sempre com o tamanho mínimo possível que é permitido pelo Blender e é mantida aberta durante toda a execução. Mesmo que o usuário feche a área, o algoritmo a re-abre novamente.

Com as informações devidamente copiadas da área "Info", agora é possível filtrar estas informações para que somente operações ou ações relevantes sejam consideradas. Como estes registros contém sempre um formato padrão, é possível diferenciar os dois de

forma rápida e direta e portanto calcular o número total dessas informações. Juntamente com a informação disponível globalmente sobre o número anterior de registros de operações (inicializada pelo "Modal Operator"), é possível inferir, através da diferença com o número atual calculado, se uma nova operação foi realizada. Em caso afirmativo, esta nova operação é então extraída e traduzida pelo próprio componente.

Um questionamento válido que pode surgir diante deste método é o de como o componente se comportaria caso a diferença entre os registros de operações atual e o anterior disponível globalmente fosse maior do que um. De fato, isto pode ocorrer em algumas operações específicas como a de deleção de um objeto (o que insere duas informações na lista extraída: uma referente à operação em si e outra um texto de comunicação para o usuário) por exemplo. A solução para este problema foi a de ignorar certos registros extraídos (já que, como dito anteriormente, eles possuem estruturas padronizadas) e, caso mesmo assim, alguma informação indesejada não seja ignorada, considerar a última operação válida capturada, obtendo assim, apenas uma operação a ser traduzida.

O processo de tradução é então solicitado através da comunicação com o componente "Operations"(5.4.4) que irá retornar a operação enviada traduzida em um formato padrão, que então será enviada de volta ao componente "Modal Operator"que poderá agora utilizar a operação processada. As especifidades do formato e do método de tradução serão explicados na seção do próprio componente em questão.

Outra funcionalidade importante do componente "Utils"é a de obter informações importantes sobre os objetos da cena que são frequentemente utilizadas pelo componente "Operations". O objetivo é justamente extrair de todos os objetos da cena todas as suas informações, não só em modo objeto (nome, posição no espaço 3D, tamanho e rotação), mas também em modo edição, ou seja, as informações sobre todos os seus vértices e as suas respectivas posições no espaço também são extraídas. Como será visto posteriormente (5.4.4), estas informações serão importantes para o tratamento de algumas operações.

A arquitetura interna deste componente pode ser conferida na figura 6. Detalhes muito específicos discutidos anteriormente foram omitidos da arquitetura para que o diagrama não ficasse poluído e muito carregado de informações. Assim como na figura 3, os retângulos com linhas pontilhadas representam outros componentes da arquitetura.

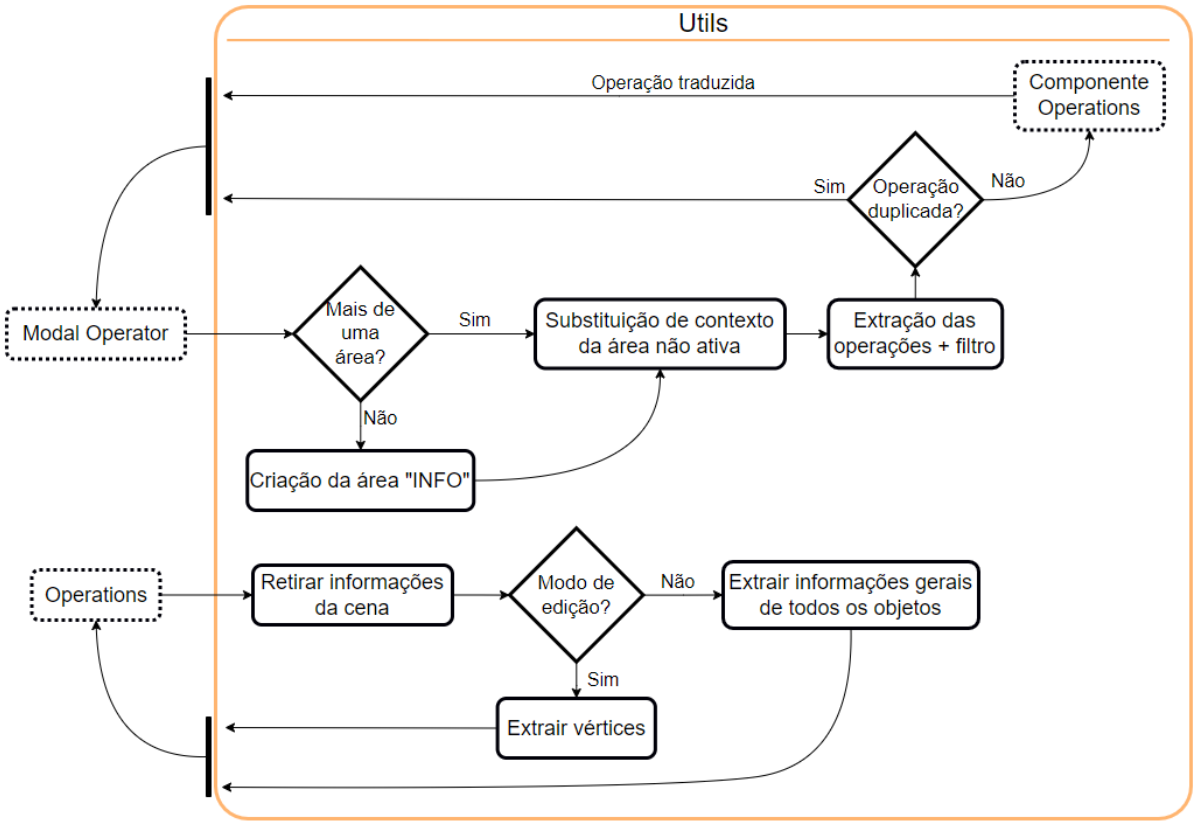

Figura 6 – Arquitetura interna do componente Utils

5.4.4 Componente Operations

Este componente é dividido em dois módulos: o dicionário de operações e as funções correspondentes das operações. O principal objetivo do componente é o de traduzir as operações recebidas pelo "Utils"(5.4.3) utilizando para isso tratamentos específicos para cada uma.

Devido à forma como o Blender descreve a estrutura das operações internamente, surge a necessidade de um tratamento específico para cada uma, já que os seus parâmetros comumente possuem nomes e tipos de dados diferentes. Assim, para cada operação a ser detectada, o componente deve ter conhecimento do método a ser utilizado para a sua tradução, conhecendo os nomes dos parâmetros relevantes e os tipos de dados que eles oferecem.

Para isso, o módulo de dicionário das operações foi criado. Este módulo possui a função de associar o nome de uma operação ao seu correspondente tratamento, que é implementado no módulo de funções do componente. A separação em dois módulos é justificável já que existem diversas operações que podem receber a mesma função de tratamento e portanto devem acionar a mesma lógica de tratamento. Como o numero

de operações é grande e um dos objetivos da arquitetura como um todo é manter a alta *performance*, a solução de fazer o uso do dicionário se torna interessante justamente pelo fato de o seu tempo de acesso ser constante independentemente do número de associações, o que não seria verdade caso fosse escolhida a solução de utilizar diversos *if-else* na implementação. Além disso, essa abordagem também viabiliza uma melhor escalabilidade do sistema uma vez que adicionar novas operações se torna mais intuitivo.

O segundo módulo (que contém as funções de tratamento das operações) possui uma lógica mais complexa, pois descreve para toda operação, um método de tradução específico. Para compreender a sua implementação, é necessário antes explicar o formato escolhido para ser usado universalmente pela arquitetura desenvolvida, i.e., como deve ser o formato das operações e seus atributos após o processo de tradução. Como esse processo deve permitir que esse novo formato seja utilizado por todo o sistema para a validação e comparação das operações, torna-se necessário incluir: o nome da operação executada, já que cada uma possui uma estrutura de dados diferente, os valores utilizados pelo usuário, pois assim o sistema é capaz de validar a operação através da comparação dos valores, e por fim, incluir qual é o objeto alvo da operação, uma vez que a operação pode ser executada em diversos objetos.

Por conseguinte, o formato padrão desenvolvido considera, respectivamente, os campos: nome da operação, valores da operação e objeto alvo.

No entanto, assim como para os outros componentes, alguns casos devem ser considerados para que se garanta o correto funcionamento do sistema desenvolvido:

- **Modo de edição e modo de objeto**: Operações que são feitas no modo de objeto (2.3) podem ser descritas de forma mais simplificada uma vez que não alteram a estrutura interna da malha. Entretanto, o mesmo não pode ser dito sobre operações executadas em modo de edição, já que informações adicionais devem ser incluídas para que a validação se torne viável. Desta forma, o módulo deve identificar se o objeto encontra-se no modo de edição e, em caso afirmativo, este deve salvar no campo de valores da operação um dicionário contendo informações sobre a configuração nova de todos os vértices do objeto em questão, quais vértices foram selecionados para a execução da operação e os eventuais atributos ou valores adicionais da operação. Assim, a estrutura ainda é mantida (o segundo campo continua contendo os valores da operação), preservando a compatibilidade em relação às outras operações.

- **Mais de um parâmetro**: Algumas operações possuem mais de um único parâmetro importante a ser guardado. Nestes casos a solução adotada é análoga à descrita no item anterior; um dicionário é criado contendo todos os valores importantes e é inserido no segundo campo a fim de se manter a compatibilidade da tradução.

- **Nem toda operação possui valores**: Existem algumas operações que não possuem valores a serem escolhidos. Por exemplo, as operações de deleção ou mudança do tipo de *shading* não possuem qualquer atributo, são operações "diretas", diferente por exemplo da operação de mover um objeto, no qual o usuário pode escolher os valores do movimento nos eixos x, y e z. Portanto, nestes casos, a operação traduzida contém somente dois campos: o nome da operação e o objeto alvo.

- **Nem sempre existe um objeto alvo**: Em alguns casos, não existe um objeto alvo. Por exemplo, se o usuário mudar o tipo de *shading* da cena (não confundir com a operação de troca de *shading* de um objeto), o "alvo"seria a própria cena e não um objeto do cenário. Por conta disso, adotou-se um padrão para operações que não possuem um alvo, que consiste em incluir no campo do objeto alvo uma *string* com o valor "Scene". Mesmo que o alvo não seja a cena, essa *string* indica ao sistema que simplesmente não existe um objeto alvo.

- **Operação não reconhecida**: A fim de tornar a identificação das operações mais flexível, desenvolveu-se um método de distinguir operações que ainda não são processadas pela arquitetura. Neste caso, a operação traduzida devolve apenas um campo contendo uma *string* indicando que a operação não foi reconhecida e logo em seguida o nome da operação (que sempre está disponível pela API do Blender). Esta solução é interessante pois permite que a execução do programa continue sem interrupções e pode servir posteriormente para identificar operações frequentemente utilizadas que ainda não são processadas pelo *logger*, servindo portanto como um meio de depuração para implementações futuras.

- **Outros casos específicos**: Além desses casos mencionados, existem diversos outros casos particulares que, devido às suas especificidades, não vale destacá-los em detalhes. Em geral, são casos relacionados à objetos que não são malhas (2.4), ou operações que necessitam de processamentos adicionais dado a forma como o Blender os descreve em sua estrutura interna.

Como é de se esperar, é justamente devido ao primeiro item descrito anteriormente (o caso do modo de objeto e de edição) que este componente deve se comunicar com o componente "Utils"(5.4.3) a fim de se obter as informações sobre os objetos (não só as suas propriedades em modo de objeto, mas também todas as informações sobre todos os seus vértices), dado que algumas operações não fornecem tais dados. Com isso, torna-se fundamental a comunicação deste componente com o "Cache"(5.4.5), a fim de atualizar os seus dados sobre os objetos da cena já que é através da comparação destes dados que se viabiliza, por exemplo, a identificação dos vértices selecionados para uma dada operação realizada no modo de edição.

Tipo de operação	Nome da operação	Valor(es) da operação	Objeto alvo
Comum com uma propriedade	String	String, lista, Boolean	String
Comum com n propriedades	String	Dicionário de tamanho n	String
Sem propriedade	String	– Não possui o campo –	String
Sem objeto alvo com uma propriedade	String	String, Lista, Boolean	String = "Scene"
Sem objeto alvo com n propriedades	String	Dicionário de tamanho n	String = "Scene"
Não reconhecida	String	– Não possui o campo –	– Não possui o campo –

Tabela 1 – Formato das operações traduzidas de acordo com os casos específicos

Na tabela 1, um resumo de como fica o resultado da tradução pode ser conferido. Nela, os casos específicos estão descritos na primeira coluna (da esquerda para a direita) e os tipos dos dados assumidos por cada campo nas demais colunas. Este formato desenvolvido permite que o log das operações seja muito flexível e portanto evita que futuras modificações sejam necessárias, dado que, segundo (YUAN; PARK; ZHOU, 2012), 98% das modificações realizadas após a criação dos métodos de *logging* são relacionadas ao conteúdo das mensagens de log, ou seja, o nível de detalhe, texto estático e variáveis.

No atual estágio de desenvolvimento, já foram cobertas 36 operações, todas com tratamentos e lógicas específicas para as suas traduções. Estas podem ser conferidas na tabela 2 que foi dividida em duas para melhor visualização no documento.

Nome da operação	Função correspondente
Run Script	runScriptOp
Toggle Edit Mode	editModeOp
Rename	renameOp
(De)select All	selectAllOp
Subdivision Set	subDivisionSetOp
New Collection	newCollectionOp
Move to Collection	moveToCollectionOp
Add Modifier	addModifierOp
Remove Modifier	removeModifierOp
Add Constraint	addConstraintOp
Delete Constraint	deleteConstraintOp
Resize	transformOp
Move	transformOp
Rotate	transformOp
Delete	deleteOp
Mirror	mirrorOp
Shade Smooth	shadeChangeOp
Shade Flat	shadeChangeOp
Recalculate Normals	recalcNormalsOp
Clear Scale	clearOp
Clear Location	clearOp
Clear Rotation	clearOp
Duplicate Objects	duplicateObjOp
Snap Selection to Cursor	snapOp
Snap Selection to Active	snapOp
Snap Selection to Grid	snapOp
Snap Cursor to Active	snapOp
Snap Cursor to Selected	snapOp
Snap Cursor to World Origin	snapOp
Snap Cursor to Grid	snapOp
Subdivide	subdivideOp
Extrude Region and Move	extrudeOp
Inset Faces	insetOp
Merge by Distance	mergeByDistanceOp
Add	defaultCase
Change scene shading	changeShading

Tabela 2 – Operações reconhecidas pelo componente e suas respectivas funções de tratamento

5.4.5 Componente Cache

O "Cache"é um componente essencial na arquitetura do sistema pelo fato de guardar informações importantes sobre o ambiente do Blender. Como discutido anteriormente, o componente "Operations"(5.4.4) está em constante comunicação com o cache a fim de comparar os valores nele guardados com os valores obtidos em tempo real.

A implementação do cache é, de certa forma, simples e segue uma estrutura que consiste em um dicionário aninhado e possui funções encapsuladas que permitem acessar ou modificar os seus valores de forma segura (utilizando *getters e setters*).

Como pode ser visto na figura 7, o cache possui de início duas chaves de acesso: "Todos os objetos"e "Todos os modificadores". O primeiro possui mais três dicionários aninhados enquanto que o segundo já acessa o valor final que consiste em uma lista dos nomes de todos os modificadores (confira a seção 2.6 para a explicação do conceito) de todos os objetos da cena. No diagrama, os retângulos contornados em azul representam as

chaves dos dicionários enquanto que os em vermelho representam os valores finais. Assim, nota-se que dado um nome de um objeto na cena, é possível acessar diretamente os valores do seu tamanho, posição e rotação (todos no formato x, y e z) além de poder acessar um outro dicionário contendo todos os vértices do objeto em questão e também as suas respectivas posições (novamente no formato dos eixos 3D).

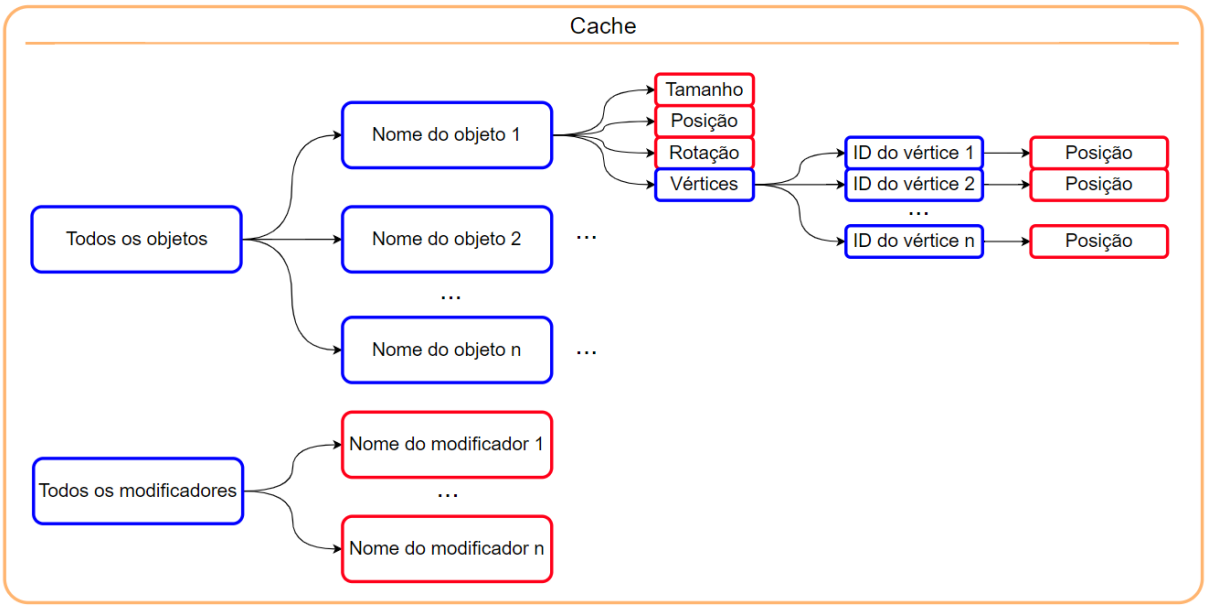

Figura 7 – Estrutura do cache

Assim, comparações de operações que modifiquem os estados dos objetos ou os nomes dos modificadores são viabilizadas e podem ser feitas utilizando as funções de acesso deste componente. Como visto anteriormente, é justamente isso que o componente "Operations"faz para extrair informações relacionadas aos objetos e as suas mudanças sofridas.

5.4.6 Componente UI

Assim como o nome sugere, este componente é responsável por *renderizar* os botões na interface gráfica do Blender para que o usuário possa inicializar o sistema desenvolvido mais facilmente. No atual estágio de desenvolvimento, este componente só implementa três botões: "*Log User Actions*"(inicia o *logger*), "*Load cube pyramid tutorial*"(carrega um tutorial simples somente de demonstração) e "*Stop logging the actions*"(para o *logger*), já que este é o objetivo desta primeira fase de desenvolvimento do projeto. Posteriormente, espera-se que a interface permita que o usuário possa carregar um tutorial de uma lista de tutoriais e possa também criar um tutorial, o que em teoria já é possível dada a arquitetura aqui desenvolvida e o formato traduzido das operações (como discutido em 5.4.2).

O local na interface do Blender escolhido para renderizar a interface foi o painel de propriedades da área de visualização 3D, que fica posicionado no canto direito da tela e pode ser facilmente aberto com um clique ou com a *shortcut* "N". Uma nova aba neste painel é criada chamada de "Blender Logger"que, quando acessada, mostra os três botões mencionados no parágrafo anterior.

O primeiro botão ("*Log User Actions*") só pode ser clicado pelo usuário quando o *logger* ainda não foi iniciado. No contexto da arquitetura desenvolvida esta verificação é importante pois, caso contrário, existiria um risco de o usuário iniciar mais de uma vez o *logger* o que criaria múltiplas instâncias do componente "Modal Operator"(5.4.1). Devido ao seu funcionamento, isso faria com que todos os *inputs* do usuário fossem capturados múltiplas vezes repetidamente, o que definitivamente sobrecarregaria o sistema. Como um dos objetivos principais do trabalho é justamente garantir a performance, isso não pode ser permitido.

O segundo botão ("*Load cube pyramid tutorial*"), assim como o primeiro, também só pode ser clicado quando o sistema ainda não foi iniciado (devido aos mesmos motivos) e o faz de forma análoga ao primeiro modo, com a diferença de que o arquivo txt de log não será gerado, mas o sistema estará capturando as ações do usuário exatamente do mesmo modo. Dessa forma, caso qualquer um destes primeiros botões seja clicado, os dois passam a ser exibidos com uma cor mais escura indicando que estão desabilitados. Nas figuras 8 e 9 são mostradas capturas de tela da interface quando nenhum dos dois botões haviam sido clicados (ou seja, o sistema ainda não havia sido ativado) e quando algum dos dois havia sido clicado pelo usuário, respectivamente.

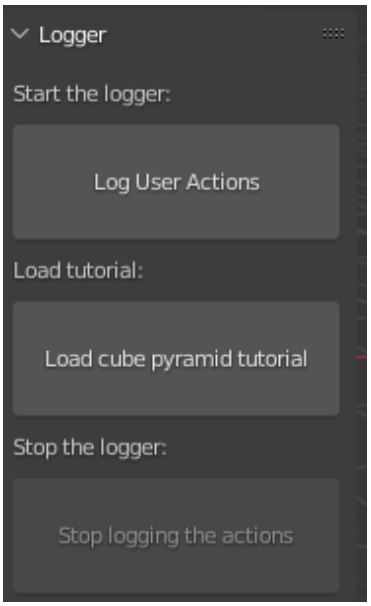

Figura 8 – Botões do sistema antes de iniciá-lo

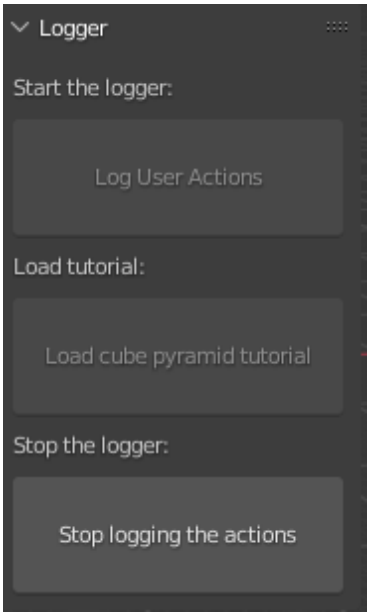

Figura 9 – Botões do sistema após a sua inicialização

Assim que o *logger* é iniciado, uma chamada para invocar o "Modal Operator"é criada o que inicia todo o processo de *logging* descrito até aqui.

Quando o botão "Stop logging the actions"é clicado pelo usuário, o sistema sinaliza para o "Modal Operator"que este deve interromper a sua execução e logo em seguida se comunica com o componente "IO"(5.4.7) para solicitar a criação de um arquivo txt de log contendo todas as operações realizadas pelo usuário enquanto o sistema estava ativo caso o sistema tenha sido inicializado pelo primeiro botão. Assim como discutido na seção 5.4.2, o arquivo de log contém as operações exatamente no mesmo formato traduzido que a arquitetura utiliza, mantendo a compatibilidade entre os registros e as operações dentro do sistema. Assim como o primeiro botão, este botão só é "ativado"quando o *logger* está em execução; não porque clicar múltiplas vezes para desativá-lo seja um problema do ponto de vista do funcionamento do sistema, mas porque desta forma o usuário tem um *feedback* da interface, que deixa claro para ele que o sistema foi de fato desativado.

5.4.7 Componente IO

Este componente é responsável por tratar das operações de entrada e saída de arquivos relacionados ao sistema desenvolvido.

Assim como alguns outros componentes da arquitetura, para esta primeira fase de desenvolvimento, a sua implementação ainda não está completa. Para fins de implementação do *logger*, este componente já é capaz de criar um arquivo txt de log contendo todas as operações e seus respectivos valores executados pelo usuário durante o tempo em que o sistema estava ativo, já no formato traduzido que é utilizado pela arquitetura. Neste

arquivo, cada linha é associada a uma única operação. Futuramente, este componente será responsável por ler estes arquivos de log e transformá-los em uma lista de operações para serem carregadas como um tutorial pelo componente "Tutorial"(5.4.2).

5.4.8 Componente Recommender

O objetivo deste componente é o de implementar a "inteligência"do sistema, i.e., o sistema de recomendação inteligente diante de todas as informações recebidas sobre as operações realizadas pelo usuário, seus erros e acertos.

Este componente está reservado para ser desenvolvido na segunda parte deste trabalho. Como a sua implementação é complexa e potencialmente extensa, optou-se por reservar um semestre inteiro somente para o seu desenvolvimento, sendo esta primeira parte reservada para a criação de toda a base necessária para este componente poder ser elaborado.

5.5 Testes e Avaliação

Para a avaliação da arquitetura e do sistema desenvolvido, foram considerados alguns cenários que visam validar os objetivos propostos para esta primeira fase de desenvolvimento do trabalho: eficiência, transparência, confiabilidade e capacidade de reportar as operações de forma correta.

5.5.1 Teste de performance, eficiência e transparência do sistema

Para validar estes três pontos (performance, eficiência e transparência), deve-se primeiro identificar qual é o possível fator limitante mais relevante da arquitetura segundo estes critérios. Devido à estrutura da arquitetura, o conceito de transparência está intimamente relacionada à eficiência e performance que o sistema possui, já que, se os componentes trabalhando em conjunto não forem eficientes em seus trabalhos, é esperado que se identifique uma perda de performance geral, o que, por consequência, acarretaria em uma perda de transparência ao usuário já que isso seria refletido diretamente na performance de execução do Blender.

De acordo com o funcionamento dos componentes da arquitetura e tomando como base o diagrama mostrado na figura 2, é possível, de primeira, excluir dos "candidatos"à limitantes os componentes "IO"e "UI". Isso se deve ao fato de que estes são os únicos componentes que não são continuamente utilizados pela arquitetura durante a sua execução, e sim, somente para iniciar ou parar o processo de *logging*.

Excluídos os dois componentes, o próximo componente que pode ser um bom candidato à limitante de performance é o "Modal Operator"dada a sua posição central

no sistema. De fato, o componente em questão pode afetar de forma significativa a eficiência do sistema uma vez que é o responsável por detectar continuamente as entradas do usuário durante a execução do sistema. No entanto, como a sua funcionalidade, de forma simplificada, se resume à isso, conferindo a tarefa de processamento das operações detectadas ao componente "Utils", a sua otimização se resumiria em simplesmente não sobrecarregar este último componente. Em outras palavras, a sua eficiência se resume em detectar somente entradas necessárias, descartar as desnecessárias e evitar detecção duplicada, medidas que já são tomadas (como descrito na seção 5.4.1) com exceção da última, puramente por uma limitação do próprio funcionamento interno do Blender. Por conseguinte, pode-se supor que, na realidade, o componente limitante da arquitetura seja o próprio "Utils". Os demais componentes também podem ser descartados já que realizam funções mais diretas e, com exceção do componente "Tutorial", só são ativados após todo o trabalho de processamento realizado por ele.

De acordo com a implementação do "Utils", este componente é o responsável não só por oferecer funções que permitem extrair informações sobre os diversos objetos da cena, como também por filtrar e processar as operações recebidas pelo "Modal Operator". É justamente nesta última funcionalidade que o fator limitante pode aparecer, mais especificamente, na eficiência da extração do número de operações realizadas através do método de substituição de contexto, como discutido em 5.4.3. Dado que uma área da interface é ativamente substituída por outra para então todo o seu conteúdo ser copiado e por fim ser substituída novamente pela anterior pode, em um cenário de estresse, ter a sua performance reduzida, o que acarretaria na perda de transparência, já que o usuário possivelmente veria a substituição ocorrendo. Além disso, dada a forma como o método é implementado, a performance tende a diminuir com o aumento da quantidade de operações realizadas pelo usuário, já que mais conteúdo deve ser copiado da área em questão.

Tendo estes pontos em vista, desenvolveu-se uma série de cenários de estresse onde diferentes quantidades de operações foram consideradas a fim de se analisar o comportamento do sistema. Ainda, a operação escolhida foi a de translação, que é uma das operações que mais possui informações a serem copiadas.

Na tabela 3, é possível conferir os valores utilizados e a resposta do sistema em milissegundos. Nota-se que para valores abaixo ou iguais a 6.000 operações, o tempo de processamento ainda é pequeno o suficiente para garantir uma funcionalidade transparente. No entanto, para quantidades de operações iguais ou acima de 8.000, o tempo de processamento começa a atrapalhar (mesmo que de forma sutil) a experiência do usuário.

Na figura 10, é possível conferir a relação entre o tempo necessário e o número de operações. Nota-se que, com o aumento do número de operações, o tempo de processamento tende a aumentar de forma aproximadamente linear para o intervalo escolhido.

Tempo (ms) arredondado	Número de operações	Quantidade de caracteres
0,55	1	465
9,50	2.000	921.122
17,61	4.000	1.844.758
24,53	6.000	2.766.091
35,40	8.000	3.686.698
46,71	10.000	5.063.626

Tabela 3 – Parâmetros dos testes de estresse realizados

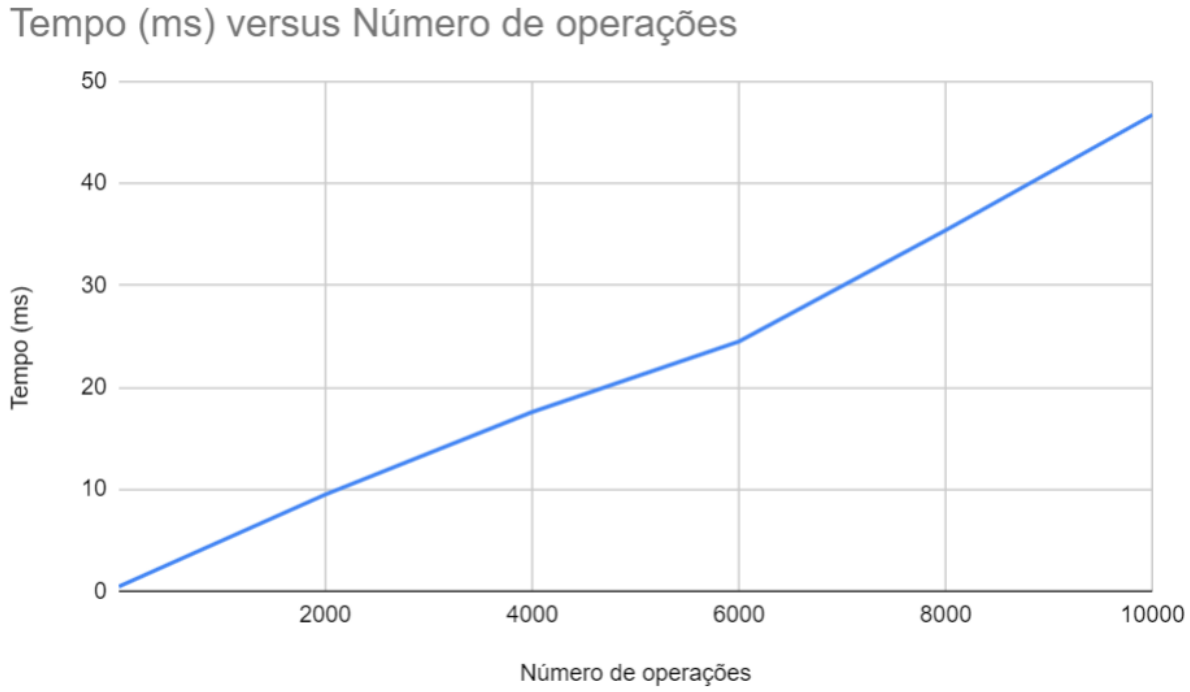

Figura 10 – Gráfico da relação entre o número de operações e o tempo de processamento

Para se obter uma noção de quantas operações um usuário normalmente realiza durante uma sessão de utilização do Blender, os arquivos de logs obtidos com os testes executados na seção 5.5.2 foram levados em consideração juntamente com um teste extra de utilização pelo próprio autor através da modelagem de um edifício de 25 andares (ou seja, um teste que envolve operações mais complexas e numerosas). Os resultados mostram que, em nenhum dos casos considerados, em uma sessão de utilização de uma hora, as operações ultrapassam a marca de mil. No teste do caso extra, por exemplo, onde se considerou a modelagem do edifício, em quarenta minutos foram realizadas 523 operações no total. Além disso, também deve-se levar em consideração que o número de caracteres a serem processados é significativamente menor do que o considerado nos cenários de estresse (para as 523 operações, foram 124.668 caracteres detectados contra aproximadamente 241.000

caracteres esperados considerando uma aproximação linear) dado que, como discutido anteriormente, a operação de translação é a que foi considerada, que é uma das que mais possui caracteres em sua descrição.

Portanto, levando em consideração que muito dificilmente um usuário irá operar com mais de 6.000 operações e 2.766.091 caracteres salvos em cache no Blender, pode-se dizer que a arquitetura desenvolvida é capaz de processar os dados de forma eficiente e performática (o sistema passa por todo o tratamento complexo das operações e ainda assim é capaz de transferir e operar com mais de dois milhões e meio de caracteres em apenas aproximadamente 24,5 milissegundos) mantendo ainda a transparência ao usuário. Mesmo que, em um cenário altamente improvável, o sistema seja utilizado com mais de 6.000 operações, ainda assim a arquitetura continuará funcionando da forma esperada, operando apenas com alguns atrasos, provavelmente menores do que meio segundo.

5.5.2 Teste de logging em cenário real

Outro aspecto importante que deve ser avaliado é a confiabilidade do sistema. Isso significa que, durante a sua execução, deve ser garantido que o sistema irá capturar todas as operações realizadas pelo usuário sem que isso resulte em problemas (como por exemplo algum erro de código, ou faça com que o Blender feche inesperadamente) e que também, ao finalizar a sua execução, o arquivo de log seja salvo com todas estas informações.

O método adotado foi o de distribuir uma versão do *logger* como um *addon* para o Blender a ser instalado por alguns alunos do Politecnico di Torino. Esta solução se demonstrou muito eficaz não só para analisar o funcionamento do sistema em diferentes contextos, mas também para detectar problemas e falhas do *logger*. Após as correções destas falhas, o sistema se demonstrou estável e capaz de gerar corretamente os arquivos de log contendo todas as operações executadas pelos usuários, que não encontraram mais problemas.

Na figura 11, é possível ver a captura de tela de um log que foi obtido por um dos alunos. É importante explicitar que quando a captura foi realizada, o código estava em uma versão mais desatualizada do que se encontra atualmente, o que significa que o número de operações que agora podem ser processadas pela arquitetura é muito maior do que na versão mostrada. Nota-se que, como discutido em 5.4.4, todas as operações seguem corretamente a estrutura padrão desenvolvida para ser usada universalmente pela arquitetura, mesmo nos casos em que a operação realizada não havia sido processada pelo sistema, que neste caso, insere uma linha com a descrição '*Not recognized operation*' e logo em seguida o nome da operação em questão.

Figura 11 – Captura de tela de um dos logs de um dos alunos que testaram o sistema

De acordo com os resultados obtidos e os testes realizados, é possível afirmar que, para situações e cenários comuns, o sistema deve ser estável e confiável, podendo ser executado sem apresentar falhas. Cenários mais específicos ou casos limite serão discutidos na próxima subseção (5.5.3).

5.5.3 Teste de cenários limites e não convencionais

Para finalizar a avaliação da solução desenvolvida, deve-se considerar também casos limite e não convencionais. Isso valida o funcionamento do sistema mesmo em situações fora do padrão, o que é especialmente relevante para este trabalho dado que grande parte dos usuários serão provavelmente inexperientes.

A seguir, cada item representa um teste realizado e contém breves explicações de como o sistema lida nesses cenários específicos:

- **Iniciar o logger sem nenhum objeto na cena** : Apesar de não parecer um caso limite, quando se leva em consideração que todas os projetos no Blender começam com um cubo, uma câmera e uma fonte de luz por *default* na cena, apagar todos os objetos para depois inicializar o logger não parece um cenário comum. Para casos como este, em que não existem objetos ativos na cena (conceito explicado em 2.2), o sistema é capaz de detectar este fenômeno e impedir que o componente

"Modal Operator"se comunique com o "Utils", já que este tentaria processar a última operação realizada que poderia ter sido dependente de um objeto ativo.

- **Deletar todos os objetos da cena** : Este cenário é análogo ao descrito anteriormente, pois ao deletar todos os objetos da cena, o conceito de objeto ativo deixa de fazer sentido e portanto o sistema poderia falhar. Neste cenário a solução implementada é a mesma do caso anterior.

- **Renomear algum objeto para o mesmo nome de antes** : Este cenário é um caso limite para o sistema devido à forma como esta operação é processada pela arquitetura, que realiza uma comparação dos nomes dos objetos salvos no cache com os atuais a fim de identificar qual objeto teve o seu nome mudado, já que a API do Blender não oferece uma forma direta de extração dessa informação. Assim, renomear um objeto para o mesmo nome de antes é problemático pois a comparação não acha diferenças e portanto retorna um valor nulo, que acaba sendo computado como uma operação possuindo esse valor. O sistema resolve esse problema considerando uma espécie de *flag* global que indica quando uma operação de troca de nome foi executada. Diante desta informação, o sistema fica ciente de que o usuário usou esta operação e aguarda por uma mudança de nome até que um tipo de evento específico é capturado, o que indica que o usuário terminou a operação. Caso nenhum nome tenha se alterado, o sistema simplesmente descarta a operação.

- **Clicar múltiplas vezes no botão de iniciar o logger** : Se o usuário por algum motivo fosse capaz de clicar várias vezes no botão de iniciar o logger, isso iniciaria vários componentes do tipo "Modal Operator"o que faria com que as ações do usuário fossem capturadas de forma duplicada diversas vezes. A solução adotada pelo sistema para este cenário foi abordada na seção 5.4.6, que consiste, resumidamente, em desabilitar o botão caso o logger já tenha sido iniciado.

- **Parar a execução do logger sem ter feito nenhuma operação** : Assim como para alguns outros casos específicos descritos anteriormente, este também tem a ver com a forma em que a arquitetura é construída. Como o sistema sempre salva o arquivo de log em formato txt após o termino de sua execução, caso nenhuma operação tenha sido executada, não haverá nada para salvar e consequentemente o Blender fechará com a mensagem de erro de escrita de arquivo. Para contornar este erro, o sistema simplesmente mantém o arquivo de log (se existir um) anterior e cancela a operação de escrita de arquivo pelo componente "IO".

- **Usar o logger somente com uma área na interface** : Como o sistema utiliza a técnica de substituição de contexto, é necessário que o usuário tenha pelo menos duas áreas abertas na interface. Este problema bem como a sua solução foram discutidas em detalhes na seção 5.4.3.

Dado que o sistema não apresentou problemas em nenhum caso comum e em nenhum dos casos limite, é possível afirmar que, dada a robustez da arquitetura construída, o sistema desenvolvido é confiável e dificilmente apresentará problemas aos usuários durante a sua execução.

6 Considerações Finais

6.1 Conclusões do Projeto de Formatura

Neste trabalho, foi descrito todo o processo de desenvolvimento de uma arquitetura no Blender capaz não só de capturar as operações realizadas pelo usuário e salvá-las como um arquivo de log, como também estabelecer uma base sólida para a segunda etapa de desenvolvimento do projeto: a implementação do sistema de recomendação.

No capítulo 5, ficou claro que o sistema se demonstrou capaz de atingir os objetivos aqui propostos - confiabilidade, transparência, desempenho e eficiência - dado o seu comportamento nos diversos testes realizados. Como foi desenvolvido em formato de *addon* para o Blender, ele é capaz de ser instalado por qualquer usuário que possua o Blender (na versão 3.2.2 ou uma versão próxima) e utilizado de forma simples e intuitiva, sem a necessidade de qualquer configuração inicial. Isso mostra que é possível desenvolver um *logger* no Blender que exija uma complexidade considerável e que ainda assim mantenha as características aqui atingidas. Além disso, também mostra que muito provavelmente o desenvolvimento do sistema de recomendação é viável e possível.

No entanto, nota-se que o sistema ainda deve aumentar a quantidade de operações que podem ser reconhecidas e processadas, bem como implementar a detecção de ações que podem afetar de forma significativa o resultado final dos arquivos de log. Estes pontos que não são suportados ainda pela solução aqui descrita não foram explorados pois provavelmente mudariam alguns aspectos fundamentais da arquitetura desenvolvida e, tendo em vista os objetivos desta primeira fase de desenvolvimento, não mudariam as conclusões do trabalho.

6.2 Contribuições e Trabalhos Futuros

Os avanços apresentados neste trabalho têm o potencial de fornecer uma arquitetura robusta e modular para capturar ações dentro do Blender de maneira eficiente e transparente. Como detalhado nos capítulos 4 (Especificação de Requisitos) e 5 (Desenvolvimento do Trabalho), cada componente do sistema, como o Modal Operator, Tutorial, Utils, Cache, Operations, UI, e IO, desempenha um papel crucial na criação de uma solução completa. O *design* modular adotado facilita o gerenciamento independente de cada componente, permitindo que equipes ou desenvolvedores individuais se concentrem em áreas específicas do sistema.

Dessa forma, a abordagem detalhada para cada componente não apenas fornece

clareza e foco, mas também estabelece as bases para futuros trabalhos. A descrição minuciosa dos requisitos funcionais de cada módulo, como a detecção precisa de ações do usuário, a eficiência na substituição de áreas ativas e a lógica para manipulação de operações complexas, cria uma fundação sólida para a expansão e evolução do sistema.

Além disso, os avanços conquistados na primeira parte deste trabalho estabelecem uma sólida base para o desenvolvimento da segunda fase, conforme discutido anteriormente. Na segunda parte, a arquitetura apresentada será utilizada como fundação para a implementação do Sistema de Recomendação (*Recommender System*). Aprofundando-se na inteligência do sistema, a segunda fase visa incorporar algoritmos e lógicas avançadas para analisar as ações do usuário registradas e fornecer recomendações relevantes no contexto do Blender. A estrutura modular permitirá que o foco seja no desenvolvimento específico do componente de Recomendação, aproveitando os dados capturados e processados de forma eficiente pela arquitetura existente. Esse enfoque incremental reflete a natureza escalável do projeto, permitindo uma evolução contínua do Sistema de Recomendação com base nos alicerces sólidos estabelecidos na primeira fase.

Portanto, a arquitetura proposta não só atende às necessidades presentes do projeto, mas também oferece uma estrutura sobre a qual trabalhos futuros podem se basear. Sua modularidade e integração profunda com a funcionalidade interna do Blender tornam-na uma solução versátil, abrindo caminho para o desenvolvimento contínuo e aprimoramento de sistemas que exijam funcionalidades semelhantes no contexto de design gráfico e modelagem 3D.

Referências

ALQUDAH, M.; RAZALI, R. An empirical study of scrumban formation based on the selection of scrum and kanban practices. *Int. J. Adv. Sci. Eng. Inf. Technol*, v. 8, n. 6, p. 2315–2322, 2018. Citado na página 20.

ALSPAUGH, S. et al. Analyzing log analysis: An empirical study of user log mining. In: *LISA14*. [S.l.: s.n.], 2014. p. 62–77. Citado na página 35.

HE, Y.; ZHAO, Y. Reform and exploration of the computer graphics. In: IEEE. *2012 IEEE Asia-Pacific Services Computing Conference*. [S.l.], 2012. p. 403–405. ISBN 978-0-7695-4897-5. Citado na página 12.

HJERT-BERNARDI, K.; MELERO, J.; HERN'NDEZ-LEO, D. Comparing the effects on students' behavior of two hint techniques embedded in a digital game-based learning tool. In: *2012 IEEE 12th International Conference on Advanced Learning Technologies*. [S.l.: s.n.], 2012. p. 138–140. Citado na página 12.

KADAM, K. et al. Integration of blender 3d in basic computer graphics course. In: *Proceedings of the International Conference on Computer Support for Collaborative Learning*. [S.l.: s.n.], 2013. p. 477–486. Citado na página 13.

LOWTHER, J. L.; SHENE, C.-K. Rendering + modeling + animation + postprocessing = computer graphics. *Computer Graphics (ACM SIGGRAPH)*, ACM, v. 34, n. 4, p. 134–135, 2000. Disponível em: <https://dl.acm.org/doi/10.1145/369215.369224>. Citado na página 13.

RICKEL, J. Intelligent virtual agents for education and training: Opportunities and challenges. In: . [S.l.: s.n.], 2001. p. 15–22. ISBN 978-3-540-42570-0. Citado na página 12.

SCHROEDER, N. L.; ADESOPE, O. O.; GILBERT, R. B. How effective are pedagogical agents for learning? a meta-analytic review. *Journal of Educational Computing Research*, v. 49, n. 1, p. 1–39, 2013. Citado na página 12.

SHAW, E. et al. Building a case for agent-assisted learning as a catalyst for curriculum reform in medical education. 06 2000. Citado na página 12.

YUAN, D.; PARK, S.; ZHOU, Y. Characterizing logging practices in open-source software. In: *2012 34th International Conference on Software Engineering (ICSE)*. [S.l.: s.n.], 2012. p. 102–112. Citado na página 41.

YUAN, G.-w.; XU, D.; ZHAO, Y. Teaching reform of computer graphics experiments. In: *The 5th International Conference on Computer Science Education*. Hefei, China: Yunnan University, 2010. Citado na página 13.