UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

VISEDU-CG 4.0: VISUALIZADOR DE MATERIAL EDUCACIONAL

WILLIAM FERNANDES KOEHLER

BLUMENAU
2015

2015/1-29

WILLIAM FERNANDES KOEHLER

VISEDU-CG 4.0: VISUALIZADOR DE MATERIAL EDUCACIONAL

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dalton Solano dos Reis - Orientador

BLUMENAU
2015

2015/1-29

VISEDU-CG 4.0: VISUALIZADOR DE MATERIAL EDUCACIONAL

Por

WILLIAM FERNANDES KOEHLER

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Dalton Solano dos Reis, M. Sc. – Orientador, FURB
Membro:	Prof. Alexander Roberto Valdameri, M. Sc. – FURB
Membro:	Prof. Matheus Carvalho Viana, Doutor – FURB

Blumenau, 09 de julho de 2015

Dedico este trabalho àqueles que me apoiaram durante o desenvolvimento do mesmo

AGRADECIMENTOS

Aos familiares que estiveram presentes.

Aos colegas, de trabalho e universidade, pelo apoio que prestaram nesses anos.

Ao meu orientador, Dalton Solano dos Reis, por ter acreditado na conclusão desse trabalho.

Se o conhecimento pode criar problemas, não é através da ignorância que podemos solucioná-los

Isaac Asimov

RESUMO

O VisEdu-CG 3.0 é uma ferramenta voltada para o ensino de computação gráfica para estudantes desta disciplina. Após três versões, a ferramenta passou a apresentar alguns obstáculos ao tentar estender a mesma, sendo o principal destes o baixo reaproveitamento dos comportamentos presentes na ferramenta. A solução alcançada para este problema foi portar o VisEdu-CG para um motor de jogos, ferramenta para criação e manutenção dos elementos de um jogo. Assim, este trabalho teve como objetivos disponibilizar um modo de visualização tridimensional em WebGL para um motor de jogos orientado a componentes, portar o VisEdu-CG para essa nova ferramenta e ajustá-la para melhorar sua usabilidade. A biblioteca Three.js, utilizado pelo motor para abstração do WebGL, proveu uma forma fácil de criar e manipular os elementos gráficos. O visualizador conta com HTML, Javascript para montar sua interface de usuário, enquanto usa JQuery para definir os comportamentos mais complexos da mesma. A avaliação de operacionalidade da aplicação foi feita a partir de um teste de experiência executado pelos alunos de computação gráfica. O trabalho também aborda os resultados obtidos a partir de testes de performance e memória realizados em alguns dos principais navegadores do mercado.

Palavras-chave: Computação gráfica. Three.js. WebGL. JQuery.

ABSTRACT

The VisEdu 3.0 is a tool for teaching computer graphics to students of this discipline. After three versions, the tools showed some troubles when trying to extend it, being the lack of reuse of its behaviors the major issue. The solution reached was to port the VisEdu-CG to a game engine, tool for creation and maintenance of game elements. Therefore, this work has aimed to provide a tridimensional viewing mode in WebGL to a component oriented game creating tool, port the VisEdu-CG to this new tool and adjust it to improve its usability. The Three.js library, used to abstract WebGL, provide an easy way to create and handle graphical elements. The Viewer relies on HTML and Javascript to assemble the user interface, while using JQuery to define its more complex behaviours. The application's operability evaluation was done after an experience test done by computer graphics students. This work also introduces to the results of performance and memory done in some of the major browsers.

Key-words: Computer graphics. Three.js. WebGL. JQuery.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

2D – 2 Dimensões

3D – 3 Dimensões

API – *Application Programming Interface*

CSS – *Cascading Style Sheets*

DOM – *Document Object Model*

FPS – *Frames per Second*

FURB – Fundação Universidade Regional de Blumenau

HTML – Hyper Text Markup Language

JSON – JavaScript Object Notation

JVM – *Java Virtual Machine*

OpenGL ES – Open Graphics Library for Embedded Systems

RF – Requisito Funcional

RNF – Requisito Não Funcional

STEP – *Scheller Teacher Education Program*

UC – *Use Case*

UML – *Unified Modeling Language*

WebGL – *Web Graphics Library*

SUMÁRIO

1 INTRODUÇÃO

Uma característica comum entre as grandes inovações tecnológicas é que elas permitiram ou mudaram a forma como as pessoas se conectam e compartilham informação. Nesse contexto, a Internet se destaca como uma inovação que alterou a própria natureza e diversidade de nossas interações. Não por acaso, a intimidade da w*eb* com a geração e compartilhamento de informação fez com que diversas pessoas tentassem introduzi-la no ambiente educacional. Dessas tentativas, nasceu o conceito de Objetos de Aprendizagem (HODGINS, 2002).

Segundo Polsani (2003), as definições e práticas de Objetos de Aprendizagem eram confusas e arbitrárias, fazendo que não pudessem se beneficiar da flexibilidade e escalabilidade oferecidas pela *web*. A complexidade tecnológica foi outro fator que dificultou a adoção do paradigma no mercado, visto que rotinas gráficas, fundamentais para os objetos, estavam amarradas à tecnologias complexas e pouco práticas.

Com o passar do tempo, a demanda por tecnologias mais práticas e eficientes aumentou e novos recursos surgiram para os desenvolvedores, como foi o caso da *Application Programming Interface* (API) *Web Graphics Library* (WebGL). O WebGL é uma biblioteca gráfica em Javascript que utiliza a já consolidada API do Open *Graphics Library for Embedded Systems* (OpenGL ES) 2.0 para executar rotinas gráficas em ambiente *web* (ARORA, 2014). Apesar de ser uma ferramenta eficiente, o WebGL trabalha exclusivamente com o ambiente gráfico, não fazendo o controle de física da aplicação. Assim, caso o programa exija comportamentos mais realísticos, como controle de colisão ou animações de objetos, essas rotinas deverão ser implementadas pelos desenvolvedores ou executadas em conjunto com outras bibliotecas (PARISI, 2014).

Segundo Boyle (2013), um dos aspectos mais importantes para os Objetos de Aprendizagem é a reusabilidade dos mesmos. Uma vez em que eles estejam projetados para isso, a tarefa de criação de novos objetos e compartilhamento dos já existentes torna-se menos complexa e facilita na utilização e crescimento do sistema como um todo.

Um dos projetos a usar Objetos de aprendizagem em ambiente *web* foi o *framework* VisEdu-CG. Este, atualmente em sua terceira versão, é uma aplicação educacional voltada à disciplina de computação gráfica. Por meio de HTML5 e WebGL, o *framework* possibilita o usuário a interagir através da *web* com vários conceitos comuns da disciplina, como: adicionar e remover objetos ao espaço gráfico, rotacionar, deslocar elementos, aplicar texturas e iluminação, entre outros (NUNES, 2014).

Diante do exposto, propõe-se continuar o desenvolvimento do framework VisEdu-CG, refatorando a arquitetura do projeto visando deixá-la mais desacoplada à uma tecnologia de renderização e com comportamentos mais componentizados, além de buscar uma nova solução para manutenção dos espaços bidimensionais da interface. Essas mudanças visam a exposição do usuário a um novo conceito gráfico, buscando assim o avanço no processo de ensino à disciplina, bem como facilitar trabalhos de extensão sobre o VisEdu-CG.

1.1 OBJETIVOS

O objetivo do projeto é fazer a integração do trabalho intitulado "VisEdu-CG 3.0: Aplicação Didática para Visualizar Material Educacional - Módulo de Computação Gráfica" desenvolvido por Nunes (2014), com o projeto "Motor para Jogos 2D Utilizando HTML5" de Harbs (2013) e acrescentar novas funcionalidades para o visualizador.

Os objetivos específicos do trabalho são:

a) portar o VisEdu-CG para a arquitetura do motor de jogos;
b) disponibilizar ao motor a API WebGL para suportar a renderização em 3D;
c) elaborar uma nova forma de representar os elementos gráfico bidimensionalmente quando em modo de renderização 2D.

1.2 ESTRUTURA

Essa monografia está estruturada em quatro capítulos. O primeiro é destinado a introdução e apresentação da estrutura do mesmo.

No segundo é tratada a fundamentação teórica necessária para compreensão dos temas da implementação.

O terceiro capítulo apresenta as etapas de desenvolvimento do motor de jogos e do visualizador de material educacional, com seus devidos diagramas de caso de uso, de pacote e de classes. Contém também os resultados e discussões sobre os resultados obtidos.

O quarto capítulo apresenta as considerações finais e as conclusões do trabalho, complementando com sugestões de extensões para futuros trabalhos.

2 FUNDAMENTAÇÃO TEÓRICA

A Seção 2.1 apresenta uma breve conceitualização a respeito de Objetos de Aprendizagem e os princípios que formam os mesmos. A Seção 2.2 é dedicada ao motor de jogos HTML5. A Seção 2.3 trata do Projeto VisEdu-CG em sua versão 3.0. Por fim, a Seção 2.4 trata dos trabalhos correlatos.

2.1 OBJETOS DE APRENDIZAGEM

Chegar a uma definição concreta para Objetos de Aprendizagem (OA) não é uma tarefa prática. Geralmente, esse processo se compromete ao tornar-se demasiadamente abrangente e se mostrar mais complexo à medida que se tenta formar o conceito a partir de considerações subjetivas. Apesar disso, uma definição conceitual pode ser traçada a partir de determinados princípios aplicados ao conteúdo digital: aprendizagem e reusabilidade (POLSANI, 2013).

Ainda, para Polsani (2013), o princípio da aprendizagem apresenta os aspectos de forma, que contextualizará o conteúdo de modo a passar um entendimento ao usuário, e narrativa, que deve conduzir o conhecimento da melhor maneira e meio (visual, textual ou qualquer outro disponível) para o mesmo.

Enquanto o princípio da aprendizagem fornece um mecanismo para a constituição interna do OA, a reusabilidade atribuí um valor para ele. O Objeto pode utilizar de flexibilidade, escalabilidade e adaptabilidade apenas quando estiver disponível a ser utilizando por diversos desenvolvedores e em diferentes contextos educacionais. A reusabilidade consiste na separação do processo de desenvolvimento do Objeto com o uso instrucional do mesmo (POLSANI, 2003).

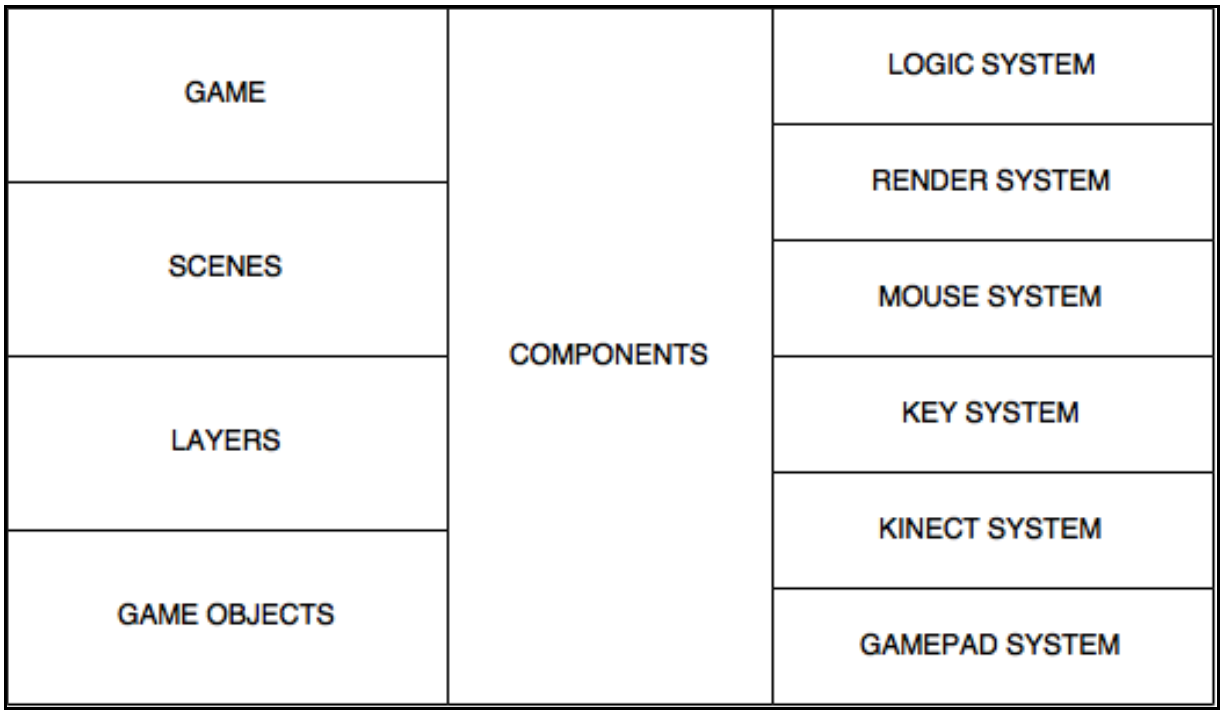
Para Boyle (2003), o desacoplamento de OA é um desafio considerável. O Objeto deve ter um objetivo específico e evitar ao máximo a dependência com outros OA. Os *designers* do sistema de aprendizagem devem pensar no impacto geral na aprendizagem e se esforçar para alcançar uma experiência de aprendizagem integrada completa. Caso sejam implementados seguindo essas características, os OAs estarão prontos para ser integrados no sistema de aprendizagem educacional e atender à diversos perfis de estudantes.

Segundo Polsani (2003), os OAs têm muito a evoluir, e cabe aos desenvolvedores, *designers*, acadêmicos e profissionais multimídia envolverem-se com esse modelo multidisciplinar e colaborativo para criação de conteúdo apropriado para a sociedade que acompanhamos atualmente.

2.2 MOTOR DE JOGOS 2D UTILIZANDO HTML5

O Motor de Jogos 2D Utilizando HTML5 é um motor voltado a criação de jogos baseado nas tecnologias Javascript e HTML5. O projeto é acompanhado por uma ferramenta de edição de jogos que visa possibilitar um modo de criação mais visual para os jogos (HARBS, 2013). A Figura 1 apresenta a arquitetura do motor de jogos e do editor.

Figura 1 - Arquitetura do editor e motor de jogos



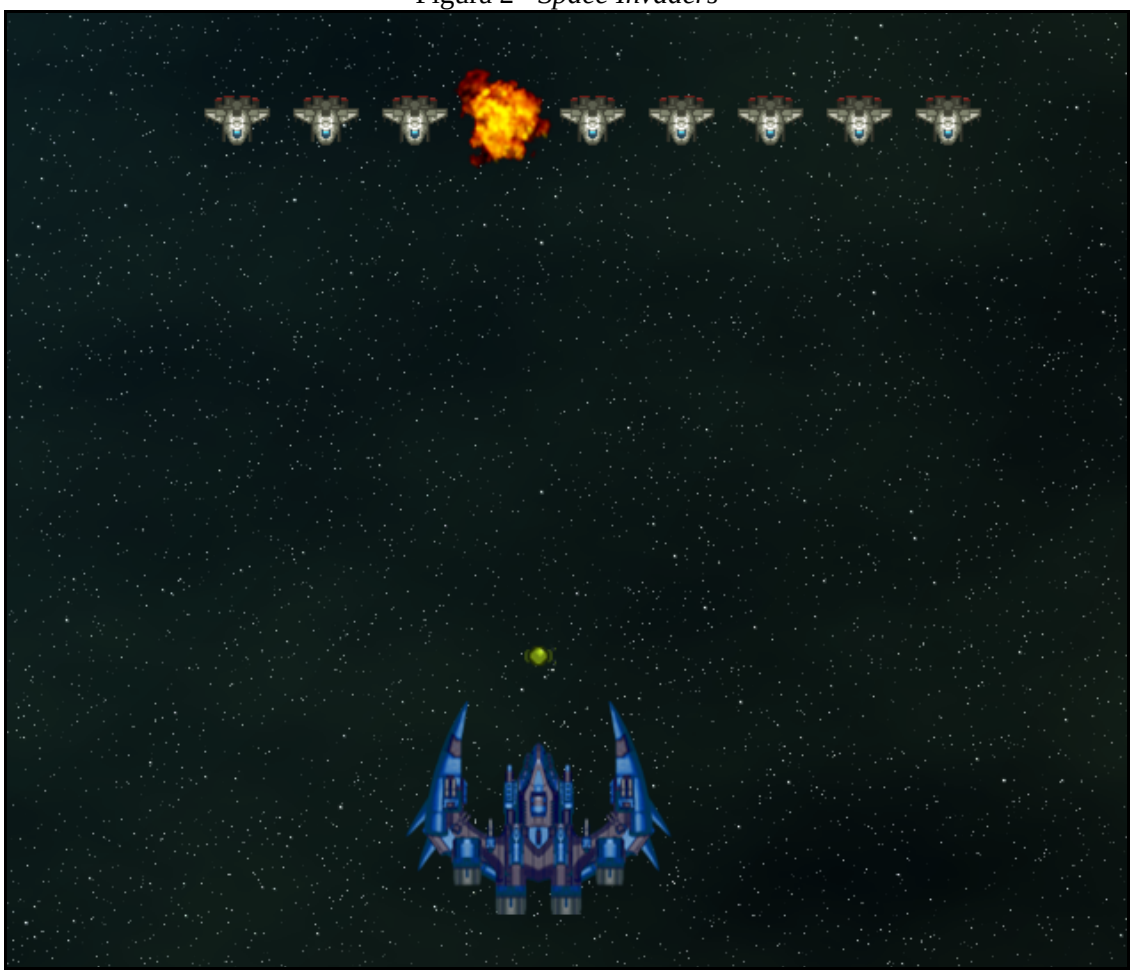
Fonte: Harbs (2013).

Segundo Harbs (2013), para desenhar as cenas o motor faz uso do *canvas*, uma novidade do HTML5 que fornece a potência gráfica para implementar de gráficos simples até vídeo games. Para Harbs (2013), um dos diferenciais da ferramenta é a arquitetura orientada a componentes, que permite que ao invés de fazer uso de herança para definição de novos tipos de comportamento, é feito o uso de componentes para estender os comportamentos deste objeto. Assim, esses componentes podem ser adicionados ou removidos dos objetos gráficos quando deseja-se alterar a forma como estes se comportam

O motor de jogos é composto por um conjunto de cenas. Uma cena é responsável por agrupar as camadas e limitar a área espacial em que os objetos irão responder à física do jogo. Cada cena deve conter ao menos uma camada. Caso tenha mais de uma, serão desenhadas na ordem em que forem adicionadas à cena. Os objetos, que são adicionados à camada, são elementos que são manipulados pelo usuário ou computador ou então são utilizados para

compor os cenários do jogo. A Figura 2 mostra um jogo de S*pace Invaders* desenvolvido a partir do motor de jogos. Neste caso, são trabalhadas funcionalidades como renderização de imagens, adição e remoção de elementos em tempo de execução e integração com o *joystick*.

Figura 2 - *Space Invaders*



Fonte: Harbs (2013).

O jogo também pode conter um conjunto de câmeras, ainda que apenas uma pode estar ativa por vez. Pode também conter os *assets*, arquivos de multimídia configurados pelo usuário e que podem ser acessados a qualquer momento. Os componentes podem ser associados aos objetos, camada ou cena. Além dos presentes no motor de jogos, o usuário do mesmo pode criar novos componentes para suprir as demandas do jogo a ser desenvolvido (Harbs, 2013).

2.3 TECNOLOGIAS USADAS NO DESENVOLVIMENTO DO MOTOR DE JOGOS

Esta seção descreve as tecnologias adotadas no desenvolvimento do VisEdu-CG 4.0. São elas HTML5, Javascript, JQuery e Three.js.

2.3.1 HTML5

Para Harbs (2013), o HTML é uma linguagem de marcação de texto usada para criação de páginas *web*. A linguagem foi desenvolvida originalmente para descrição semântica de trabalhos científicos. Com o passar dos anos, seu *design* e adaptações permitiram que a tecnologia ganhasse espaço e fosse usada para descrever outros tipos de documentos, como as páginas *web*. O HTML5 é a mais nova versão do HTML, trazendo novos recursos e padronizando comportamentos que já eram usados em larga escala pelos desenvolvedores (2010).

Essa nova versão do HTML também permitiu a execução de rotinas gráficas dentro dos navegadores. O WebGL, baseado nas especificações do OpenGL ES 2.0, permitiu que os programas renderizassem cenas gráficas complexas de forma nativa, sem a necessidade de contar com *plug-ins* externos (NUNES, 2014).

2.3.2 Javascript

A linguagem Javascript surgiu pela necessidade de manipular elementos visuais da página HTML, como campos de formulários e imagens, adicionando um dinamismo para telas até o momento estáticas. A necessidade cada vez maior de interação entre o usuário e página garantiram a popularidade da linguagem (HARBS, 2013). Atualmente, a linguagem se encontra na versão 1.8.5, homologada em julho de 2010 (W3SCHOOLS, 2015).

Hoje, o Javascript é uma linguagem amplamente disseminada no desenvolvimento *web*, estando assim presente na maioria dos navegadores presentes no mercado, inclusive nos de *smarthphones* e *tablets* (CANTOR; JONES, 2012).

A linguagem pode ser descrita como leve e dinâmica, permitindo que seu código seja inserido no meio das páginas *web.* Além disso, é considerada como tendo uma baixa curva de aprendizagem (W3SCHOOLS, 2015).

2.3.3 JQuery

O JQuery é uma biblioteca Javascript de código aberto e compacto, criada com o objetivo de facilitar a programação em Javascript. O JQuery adiciona expressividade ao Javascript, pois permite que com poucas linhas de código seja possível efetuar uma séria de operações que custariam horas de programação e centenas de linhas, caso a biblioteca não fosse utilizada (MCFARLAND, 2011).

Um dos recursos mais populares da linguagem é o uso de seletores, da mesma forma que o CSS, para selecionar os elementos HTML. O JQuery é amplamente utilizado para fazer a manipulação dos campos da tela, podendo alterar as propriedades dos mesmos, injetar conteúdo e fazer requisições para o servidor (W3SCHOOLS, 2015).

Além dos recursos padrões da biblioteca, é possível encontrar diversos *plug-ins* desenvolvidos para estender sua capacidade. Esses *plug-ins* podem ser usados para adicionar novos comportamentos nos elementos da tela, mudar a apresentação da interface do usuário, tratamento de imagens, aplicar efeitos de animação, entre outros (JQUERY, 2015).

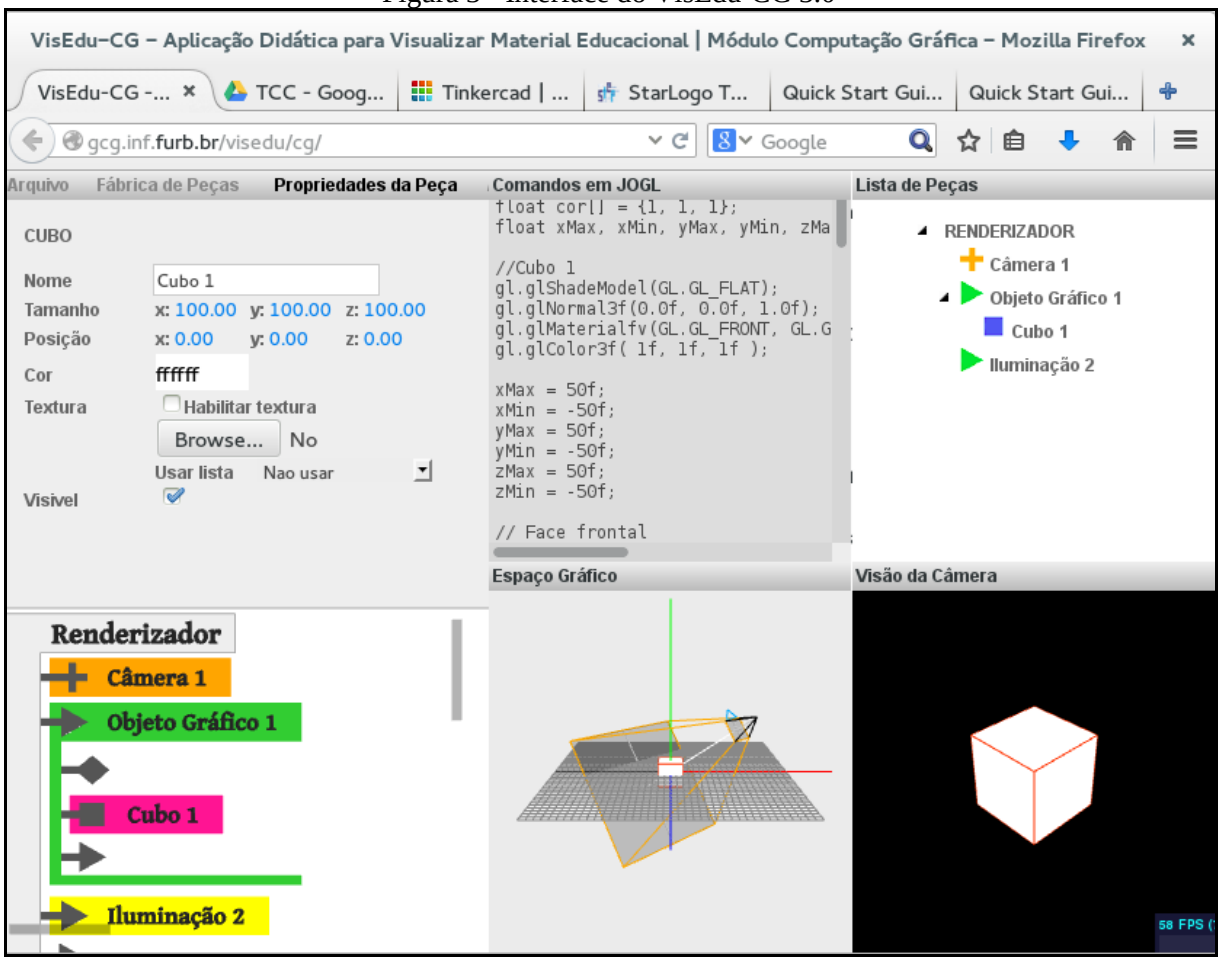
2.3.4 Three.js

O WebGL é uma API voltada para a criação de gráficos de alta performance em 2D e 3D. Com ela, é possível aproveitar as capacidades da placa de vídeo do usuário sem ter que abandonar ambiente *web*. Apesar do poder da tecnologia, a mesma pode se mostrar problemática, uma vez em que a criação e manutenção de seus elementos, além da depuração de seu código não são tarefas simples. O Three.js surgiu como uma alternativa para abstração das tecnologias de renderização. Uma das APIs gráficas suportadas pelo Three.js é o WebGL. A abstração que a biblioteca fornece para o WebGL facilita a atividade de criar figuras tridimensionais, animar objetos pela cena, aplicar texturas e materiais, fazer a carga de modelos 3D, entre outras coisas. O Three.js é uma biblioteca Javascript, permitindo assim desfrutar de benefícios da linguagem como a fácil integração com o código das páginas HTML e o aproveitamento de outras bibliotecas Javascript, como o JQuery (DIRSKEN, 2013).

2.4 VISEDU-CG 3.0: APLICAÇÃO DIDÁTICA PARA VISUALIZAR MATERIAL EDUCACIONAL – MODULO DE COMPUTAÇÃO GRÁFICA

O VisEdu é um *framework* voltado para o aprendizado da disciplina de computação gráfica. O ambiente utiliza da plataforma w*eb* para demonstrar funções comuns entre as bibliotecas gráficas, como a transformação de objetos (rotação, translação e escala), uso da câmera e texturas (NUNES, 2014). A interface do VisEdu-CG em sua versão 3.0 pode ser vista na Figura 3.

Figura 3 - Interface do VisEdu-CG 3.0



Fonte: Nunes (2014).

O *framework* conta com uma janela denominada `Fábrica de peças`, onde o usuário pode adicionar ao *`Renderizador`* alguma peça que representa um elemento dentro de uma cena, como a `câmera`, `iluminação`, `transformação` ou `objeto gráfico`, por exemplo. Cada peça tem suas propriedades, que podem ser vistas e modificadas na janela de `Propriedades da Peça`. Cada tipo de peça possui propriedades distintas: um `cubo` tem, dentre outras, cor e textura, enquanto a `câmera` tem os valores de `near` e `far`. As peças adicionadas ao renderizador são desenhadas na janela `Espaço Gráfico`. Além dessas, o programa também apresenta as janelas de *`Arquivo`*, *`Lista de peças`*, *`Comandos em JOGL`* e `Visão da Câmera`, que são responsáveis por, respectivamente, exportar ou abrir um arquivo representando uma cena, listar as peças ativas de forma hierárquica, exibir o código fonte necessário para reproduzir o resultado gráfico da peça selecionada e mostrar a cena a partir da câmera (NUNES, 2014). A primeira versão do trabalho, denominada AduboGL, desenvolvida por Araújo (2012), consistia numa aplicação *desktop* em C++ que apresentou a mecânica de usar as peças para demostrar conceitos da computação gráfica. Essa versão trazia apenas as

peças de `Cubo`, `Rotação`, `Translação` e `Escala`, mas estabeleceu a base dos trabalhos que o seguiram.

A segunda versão do trabalho foi feita por Montibeler (2014). O projeto foi responsável por trazer a aplicação para o contexto *web*, já com o uso da biblioteca Three.js, e adicionar as peças de `Câmera` e `Objeto Gráfico`.

Para a versão 3.0 do projeto, Nunes (2014) implementou três novos tipos de peças para a fábrica: *spline*, polígono e luz. Também foi adicionado o comportamento de poder selecionar a figura a partir do espaço 3D, mas devido à forma como foi implementado, o recurso não teve a operacionalidade esperada. Por fim, foi disponibilizado um ambiente 2D para trabalhar, contendo os mesmos recursos do ambiente tridimensional, mas que ainda precisa ser melhorado. Nunes chegou a sugerir algumas extensões para o trabalho, entre elas estão a disponibilização de um recurso de animação na cena, ajustes de usabilidade e performance do `Polígono` e ajustes na usabilidade dos elementos quando em modo 2D.

2.5 TRABALHOS CORRELATOS

Estão disponíveis uma série de trabalhos dedicados à criação e manipulação de cenas gráficas. Desses, é possível destacar algumas ferramentas voltadas para a área de aprendizagem. Para que consigam motivar e incentivar os usuários iniciantes, essas ferramentas permitem que com poucos cliques o usuário consiga compor seu cenário e aplicar os conceitos da computação gráfica, como transformações e iluminação, sobre os objetos presentes.

Na Seção 2.5.1 abordada o TinkerCad (TINKERCAD, 2014), software voltado para a criação de cenários e objetos tridimensionais através de um ambiente *web*. A Seção 2.5.2 se dedica ao StarLogo TNG (STEP, 2014), software educacional voltado para construção de jogos.

2.5.1 TINKERCAD

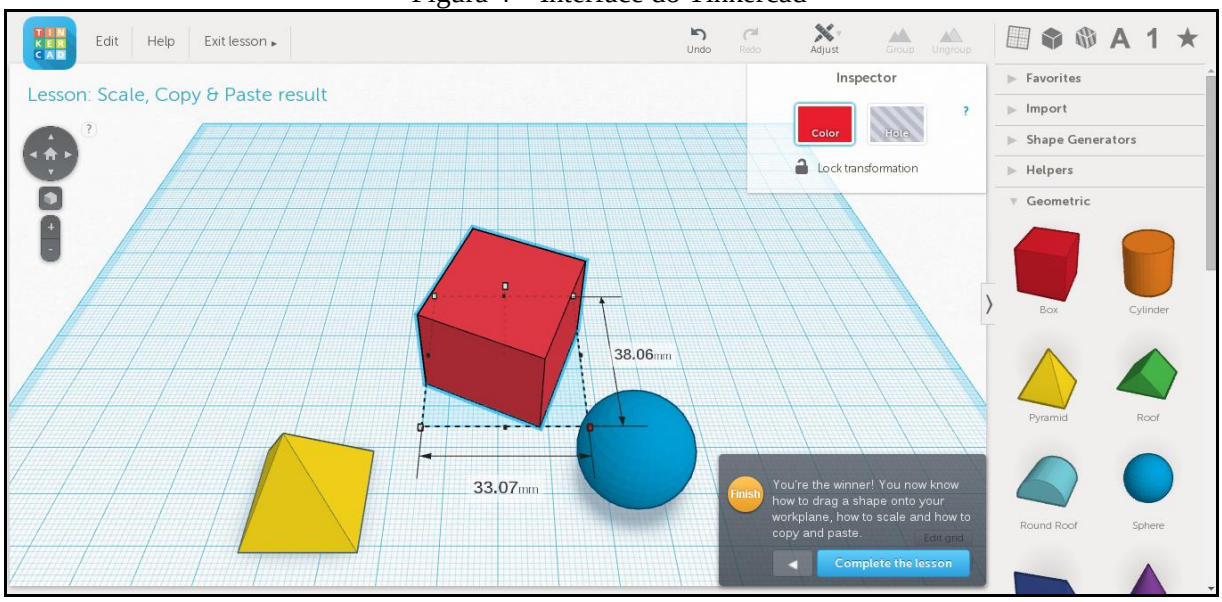
O Tinkercad (Figura 4) é uma ferramenta que permite que o usuário monte um cenário 3D adicionando e modelando objetos pré-definidos, como cubos, pirâmides e esferas. Além disso, o programa possibilita que objetos sejam combinados, gerando novos elementos gráficos (TINKERCAD, 2014).

A tecnologia escolhida para o desenvolvimento da parte gráfica da ferramenta foi o WebGL, pois por meio deste API seria possível utilizar a ferramenta a partir dos navegadores mais comuns no mercado. Outras características é a capacidade de funcionar como interface

para impressoras 3D. Esses fatores contribuíram em tornar o Tinkercad popular entre estudantes de computação gráfica ou design, artistas e em empresas menores (TINKERCAD, 2014).

A Figura 4 mostra a interface do programa durante o redimensionamento de um componente, na qual é possível observar os elementos adicionados à cena, as propriedades do objeto a ser editado e alguns tipos de figuras que podem ser adicionadas.

Figura 4 – Interface do Tinkercad



Fonte: Tinkercad (2014).

2.5.2 STARLOGO TNG

O StarLogo TNG é uma ferramenta voltada para modelagem e simulação desenvolvida pelo MIT's S*cheller Teacher Education Program* (STEP). O mesmo apresenta uma ferramenta que possibilita uma programação visual, ou programação baseada em blocos. O usuário consegue arrastar para a tela personagens e terrenos, e sobre eles adicionar ações, texturas ou diversos outros recursos (STEP, 2014).

Além dos blocos para construção do ambiente gráfico propriamente dito, também estão disponíveis peças para se fazer o controle lógico do jogo, assim como controles para fazer a integração entre elementos lógicos e físicos do cenário.

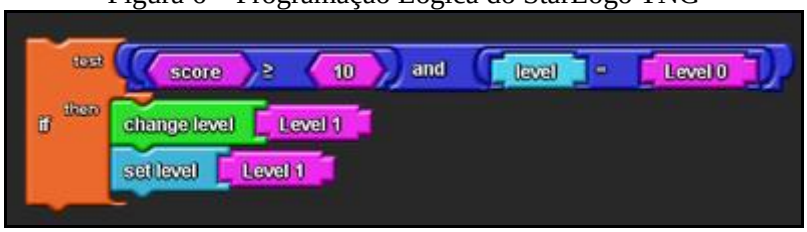
A Figura 5 mostra a interface de programação baseada em blocos em conjunto com a tela do cenário do jogo montado.

Figura 5 – Interface de programação do StarLogo TNG

Fonte: Letopsi (2015).

Também é possível combinar diversos comandos para gerar procedimentos que podem ser reaproveitados em outras rotinas (STEP, 2014). A Figura 6 mostra um exemplo de programação lógica da fase quando o jogador atingir 10 pontos e o nível for 0, será feita a troca para o nível 1.

Figura 6 – Programação Lógica do StarLogo TNG



Fonte: STEP (2014).

O StarLogoTNG foi desenvolvido em Java e, dessa forma, pode ser executado a partir de qualquer plataforma que tenha uma implementação da *Java Virtual Machine* (JVM).

3 DESENVOLVIMENTO

Neste capítulo são abordadas as etapas da integração entre o motor de jogo e do visualizador de material educacional. Aqui são apresentados os requisitos, especificação, detalhes de implementação e resultados e discussão.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O motor de jogos deve:

a) trabalhar em espaços 3D ou 2D (Requisito Funcional - RF);
b) desenhar os tipos de objetos já existentes no motor em 3D (RF);
c) desenhar componentes do tipo Polígono e *Spline* (RF);
d) desenhar componentes responsável por renderizar luz (RF);
e) desenhar *Helpers* dos componentes de luz, câmera, *grid* e *axis* (RF);
f) permitir que a aplicação que o utiliza a customize os métodos de renderização (Requisito não Funcional - RNF).

O visualizador deve:

a) implementar controle orbital de câmera (RF);
b) permitir a transição entre 2D e 3D sem apagar as peças presentes (RF);
c) implementar o menu da aplicação utilizando as tecnologias HTML, Javascript e JQuery (RNF).

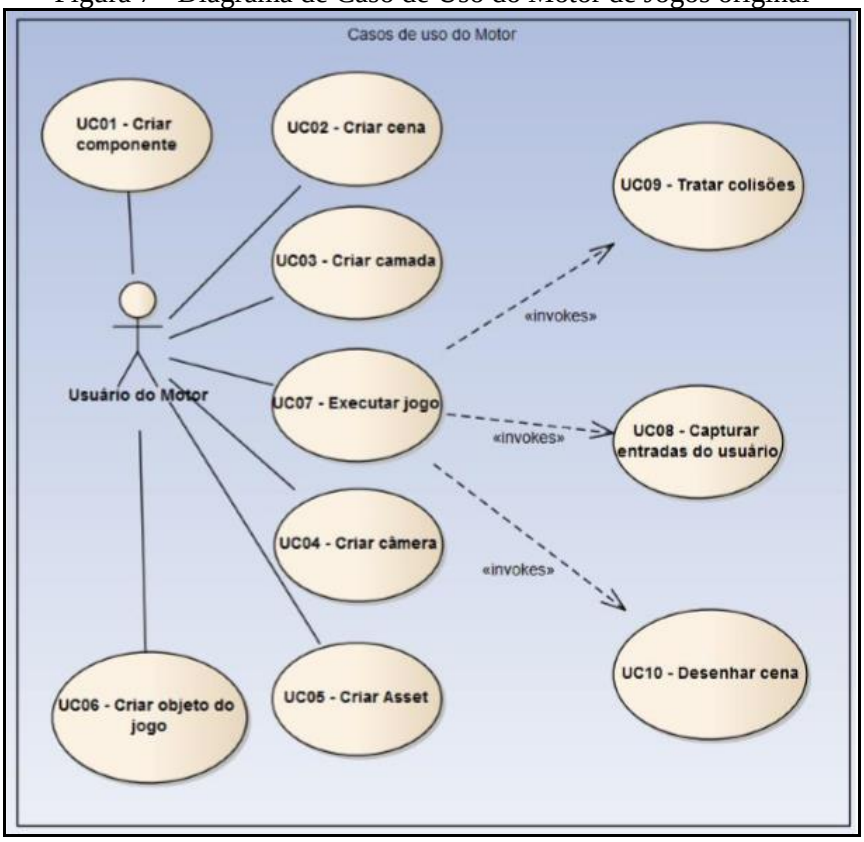
3.2 ESPECIFICAÇÃO

Para especificar esse projeto, foi feito uso de diagramas da *Unified Modeling Language* (UML), desenhados através da ferramenta Draw.io. Nas seções seguintes são apresentados os diagramas de caso de uso, de pacote, de classe e de sequência que representam a arquitetura do trabalho.

3.2.1 Casos de uso do motor de jogos

Esta seção é dedicada a descrever os casos de uso das funcionalidades presentes no motor de jogos. O diagrama original do motor pode ser observado através da Figura 7.

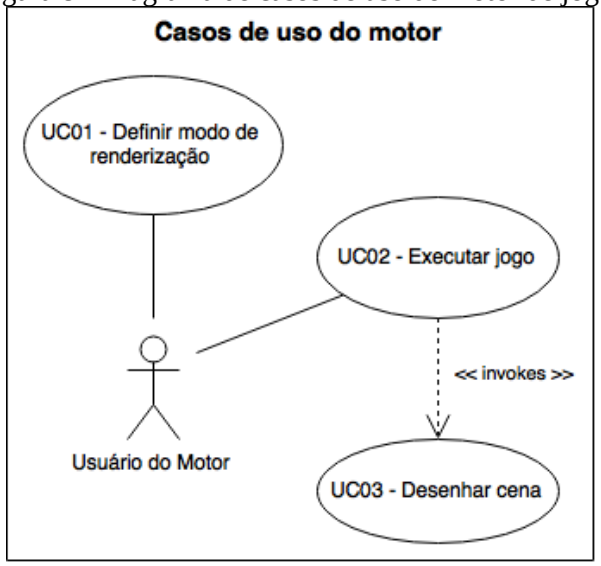
Figura 7 - Diagrama de Caso de Uso do Motor de Jogos original



Fonte: Harbs (2013).

Os casos de uso definidos na primeira implementação do motor já abrangem os casos necessários para este trabalho. Assim, o foi adicionado um novo caso de uso referente à definição do modo de renderização. Também é apresentado o caso `UC03 - Desenhar cena` pois o mesmo foi alterado para se adequar aos objetivos deste trabalho. O caso `UC02`, presente nos casos de uso originais, é mostrado para contextualizar o `UC03`. O diagrama de casos de uso do motor de jogos pode ser visto na Figura 8.

Figura 8 - Diagrama de casos de uso do motor de jogos



O Quadro 1 apresenta os casos de uso do motor de jogos que foram criados ou alterados e os requisitos funcionais referentes a cada caso.

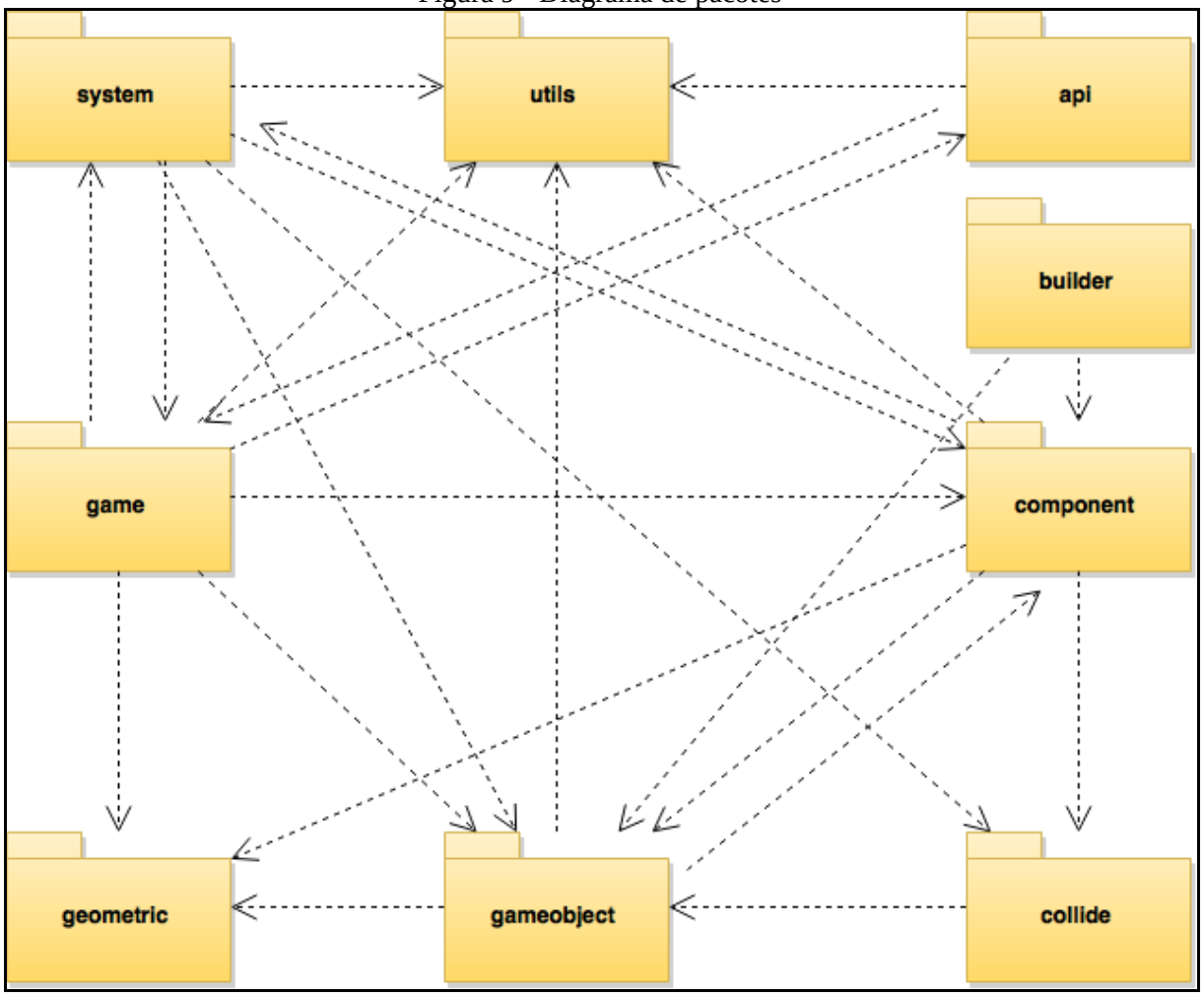
Quadro 1 - Relação entre requisitos e casos de uso do motor de jogos

Caso de uso	Requisitos do motor de jogos
UC01 – Definir modo de renderização	• permitir que a aplicação que o utiliza a customize os métodos de renderização
UC03 - Desenhar cena	• desenhar os tipos de objetos já existentes no motor em 3D; • desenhar componentes do tipo Polígono e *Spline*; • desenhar componentes responsável por renderizar luz; • desenhar *Helpers* dos componentes de luz, câmera, *grid* e *axis*; • permitir que a aplicação que o utiliza a customize os métodos de renderização.

3.2.2 Diagramas do motor de jogos

Nesta seção são apresentados os pacotes utilizados pelo motor de jogos. A Figura 9 representa o diagrama de pacote do motor sem o detalhamento das classes, pois estas serão descritas nas seções subsequentes.

Figura 9 - Diagrama de pacotes

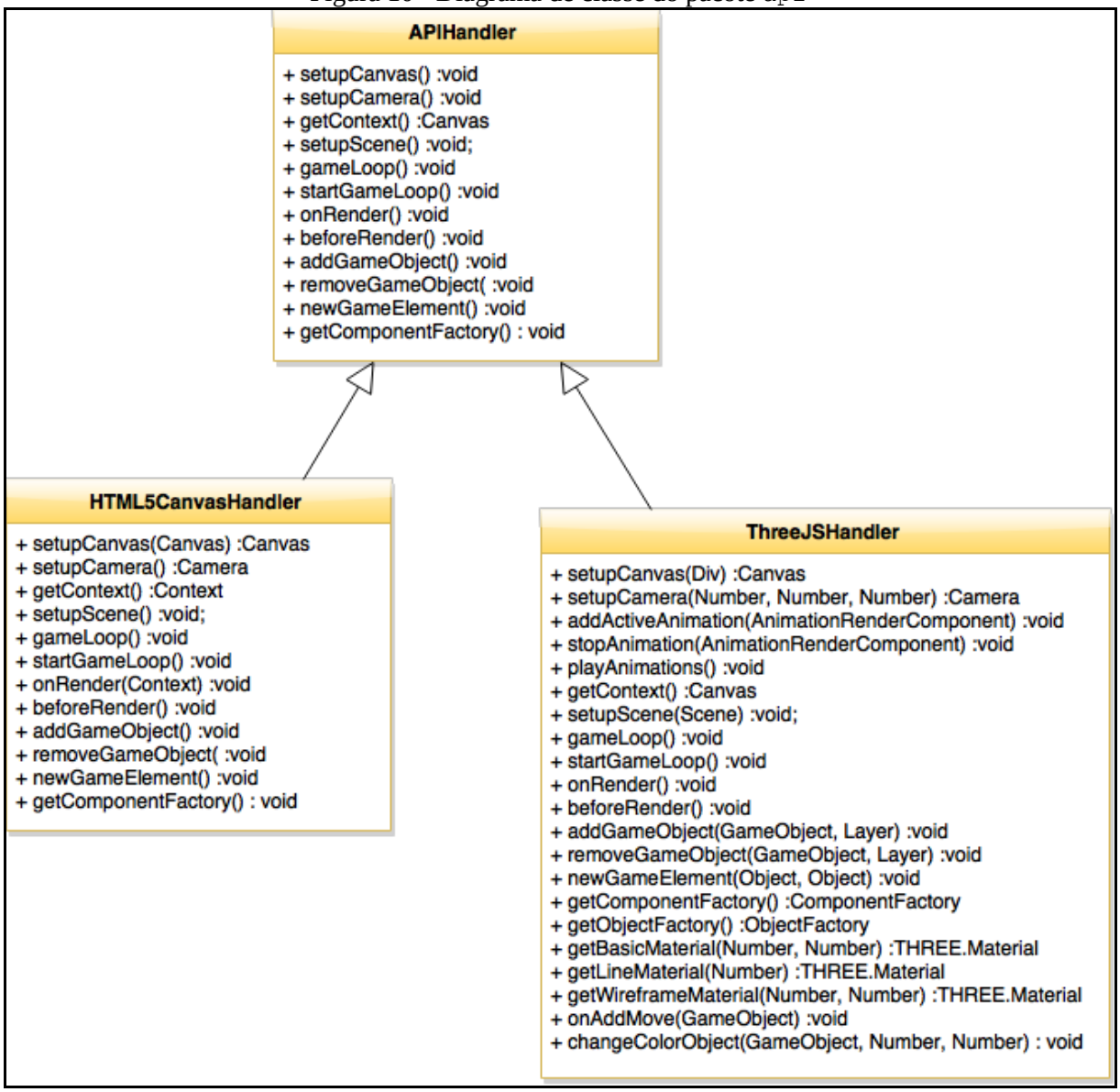


Os pacotes `api` e `builder` foram desenvolvidos neste projeto, sendo o resto importado do motor de jogos original (HARBS, 2013). Ainda assim, o único deles que não passou por nenhuma modificação foi o `collide`. As seções a seguir descrevem as classes que compõe os pacotes manipulados para este projeto.

3.2.2.1 Pacote `api`

O pacote de `api` foi criado para agrupar e isolar as rotinas que manipulam diretamente os métodos específicos de alguma API. Além disso, foi criada a classe `APIHandler`, que exercerá o papel de interface para as funções com objetivos comuns dentro de seu contexto gráfico. As classes presentes no pacote podem ser conferidas na Figura 10.

Figura 10 - Diagrama de classe do pacote `api`



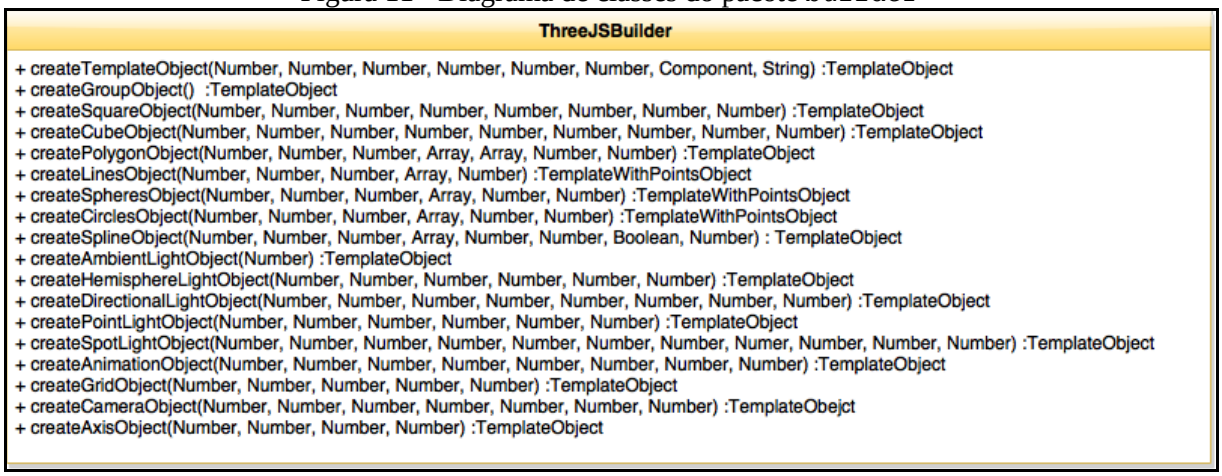
Foi criada uma classe para cada API gráfica, além da interface implementada por elas. Sendo assim, `HTML5CanvasHandler` e `ThreeJSHandler` possuem os métodos que vão consumir o *canvas* do HTML5 e o WebGL do ThreeJS, respectivamente.

Além das funções declaradas na interface, o `ThreeJSHandler` possui algumas funções mais específicas que visam atender necessidades exclusivas do contexto WebGL, como o tratamento de materiais.

3.2.2.2 Pacote `builder`

O pacote `builder` comporta uma classe criada para auxiliar o processo de criação de novos objetos. O diagrama do pacote pode ser observado a partir da Figura 11.

Figura 11 - Diagrama de classes do pacote `builder`

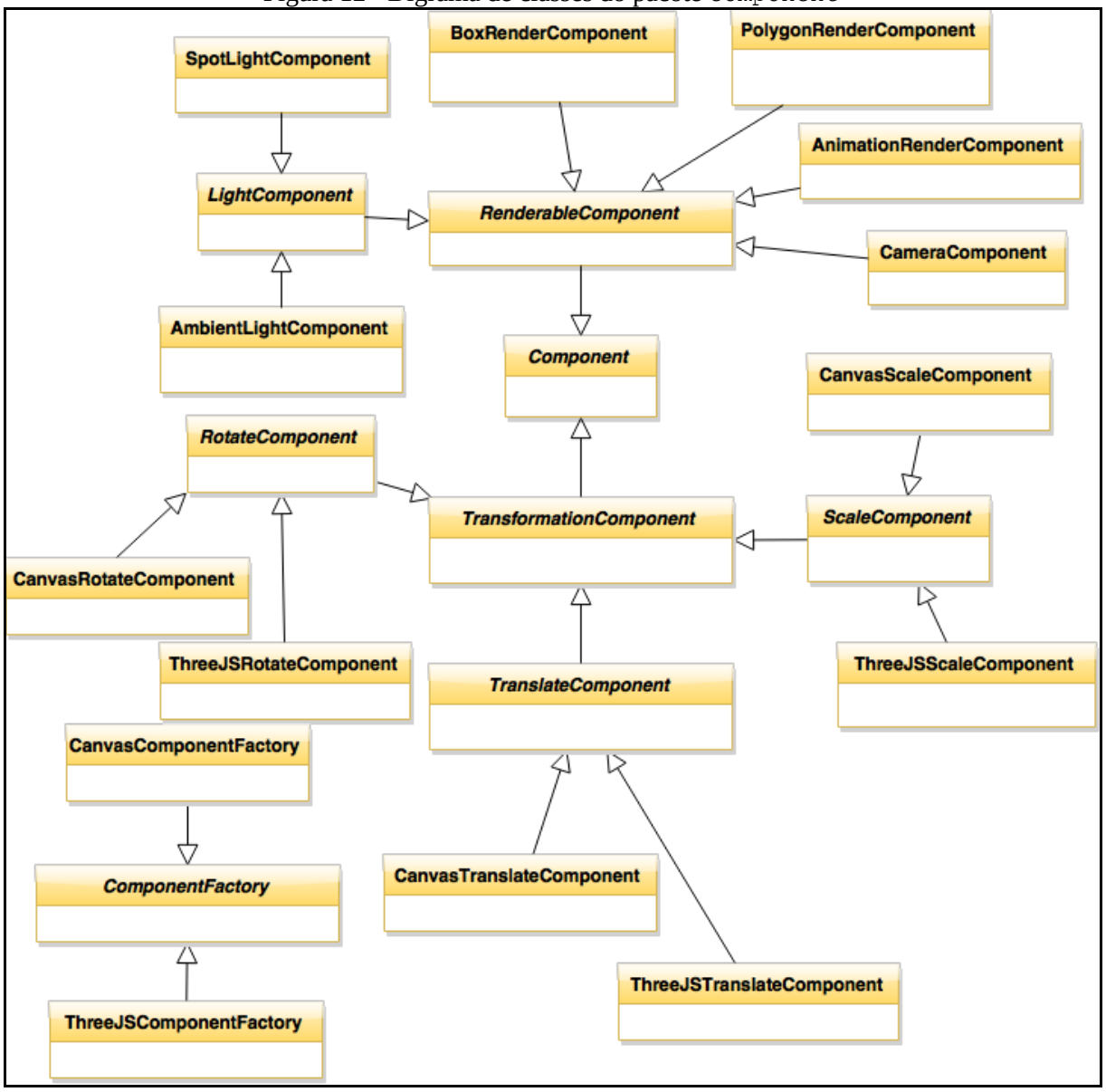


A classe `ThreeJSBuilder` foi criada ao observar a demanda por uma maneira facilitada de criar os objetos usados para compor as cenas do motor gráfico. O método `createTemplateObject` recebe os dados espaciais, ou seja, as coordenadas e dimensão, o componente a ser utilizado e uma etiqueta para o mesmo. O resultado é o componente encapsulado num objeto com capacidade de já ser adicionado à cena e ser submetido à transformações geométricas de escala, rotação e translação. Os outros métodos da classe são responsáveis por criar objetos para seus respectivos componentes, fazendo uso do `createTemplateObject`.

3.2.2.3 Pacote `component`

Seguindo os conceitos estabelecidos no desenvolvimento original do motor de jogos, o pacote `component` contém os componentes utilizados para desenhar e interagir com os elementos gráficos do jogo. Também disponibiliza os recursos necessários para que o usuário possa desenvolver seus próprios componentes e uma fábrica utilizada para criar os componentes de transformações espaciais de acordo com o contexto da aplicação. A Figura 12 apresenta algumas das classes presentes.

Figura 12 - Digrama de classes do pacote `component`



Os componentes que estendem da interface `RenderableComponent` são os responsáveis por desenhar os elementos na tela. Cada um destes componentes representa uma forma de figura, o que pode ser observado no diagrama por meio do `BoxRenderCompoent`, `AnimationRenderComponent` e `PolygonRenderComponent`, encarregados de desenhar retângulos, animações e polígonos, respectivamente. Além destes presentes no diagrama, o pacote também contém os seguintes componentes gráficos: `AxisRenderComponent`, `CircleRenderComponent`, `CirclesRenderComponent`, `CubeRenderComponent`, `GridRenderComponent`, `GroupComponent`, `ImageRenderComponent`, `LinesRenderComponent`, `SpheresRenderComponent` e `SplineComponent`. Estendendo do `RenderableComponent`, também temos os componentes de luz. Para estes, o `LightComponent` funciona como uma classe abstrata, interfaceando as funções que as

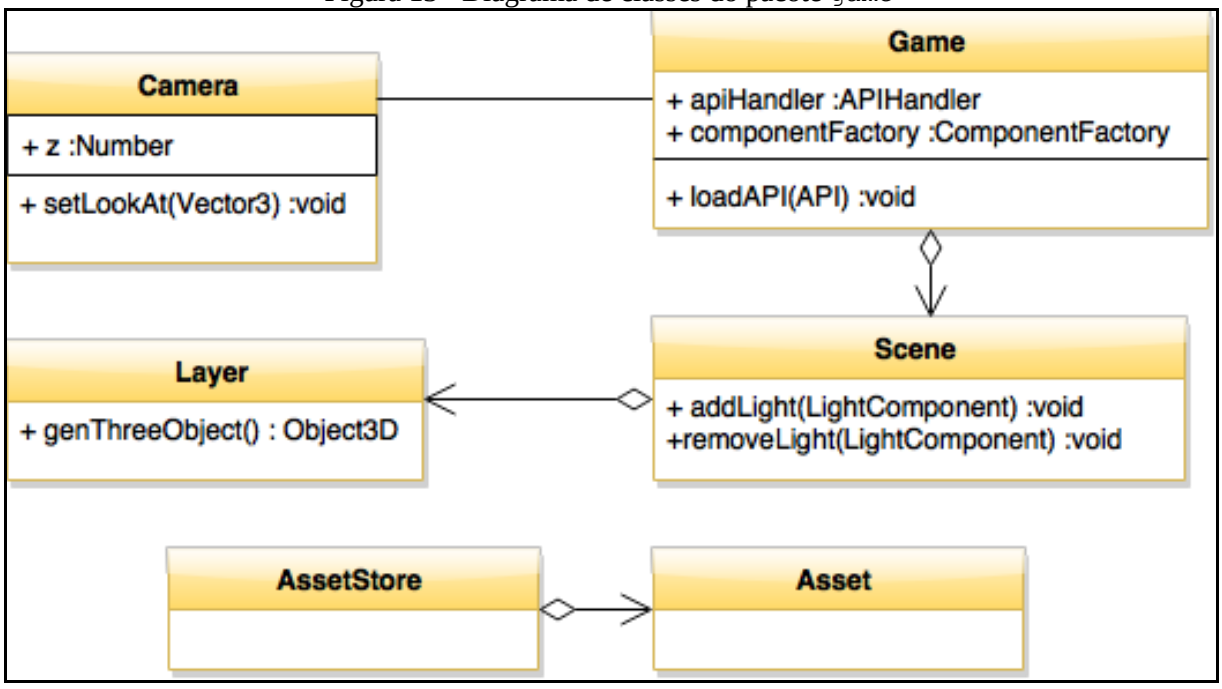
especializações devem implementar. Os componentes especializados de luz são o `AmbientLightComponent` e `SpotLightComponent`, presentes no diagrama, mais o `PointLightComponent` e `DirectionalLightComponent`. Os componentes gráficos implementam o método `genThreeObject`, criado para que eles montem um objeto tridimensional por meio do Three.js. Alguns componentes de luz e o de câmera também implementam o método `genHelper`, que cria objetos visuais que mostram o alcance e limites de seus elementos gráficos.

Também se encontram no pacote os componentes responsáveis por tratar as transformações, derivados da classe `TransformationComponent`. O `RotateComponent`, `TranslateComponent` e `ScaleComponent`, presentes na implementação original do motor, perderam seu papel de classes especializadas para se tornarem interfaces. Assim, cada transformação ganhou uma implementação destinada para o *canvas* do HTML5 e outra para o Three.js. Para controlar qual destes deve ser instanciado, foram criadas as classes `ComponentFactory`, `CanvasComponentFactory` e `ThreeJSComponentFactory`. Esta fábrica abstrata é instanciada uma vez para então suprir os objetos que requerem transformações com o componente adequado para o contexto.

3.2.2.4 Pacote `game`

O pacote game armazena as classes necessárias para a execução do jogo. O diagrama do pacote é apresentado pela Figura 13.

Figura 13 - Diagrama de classes do pacote `game`

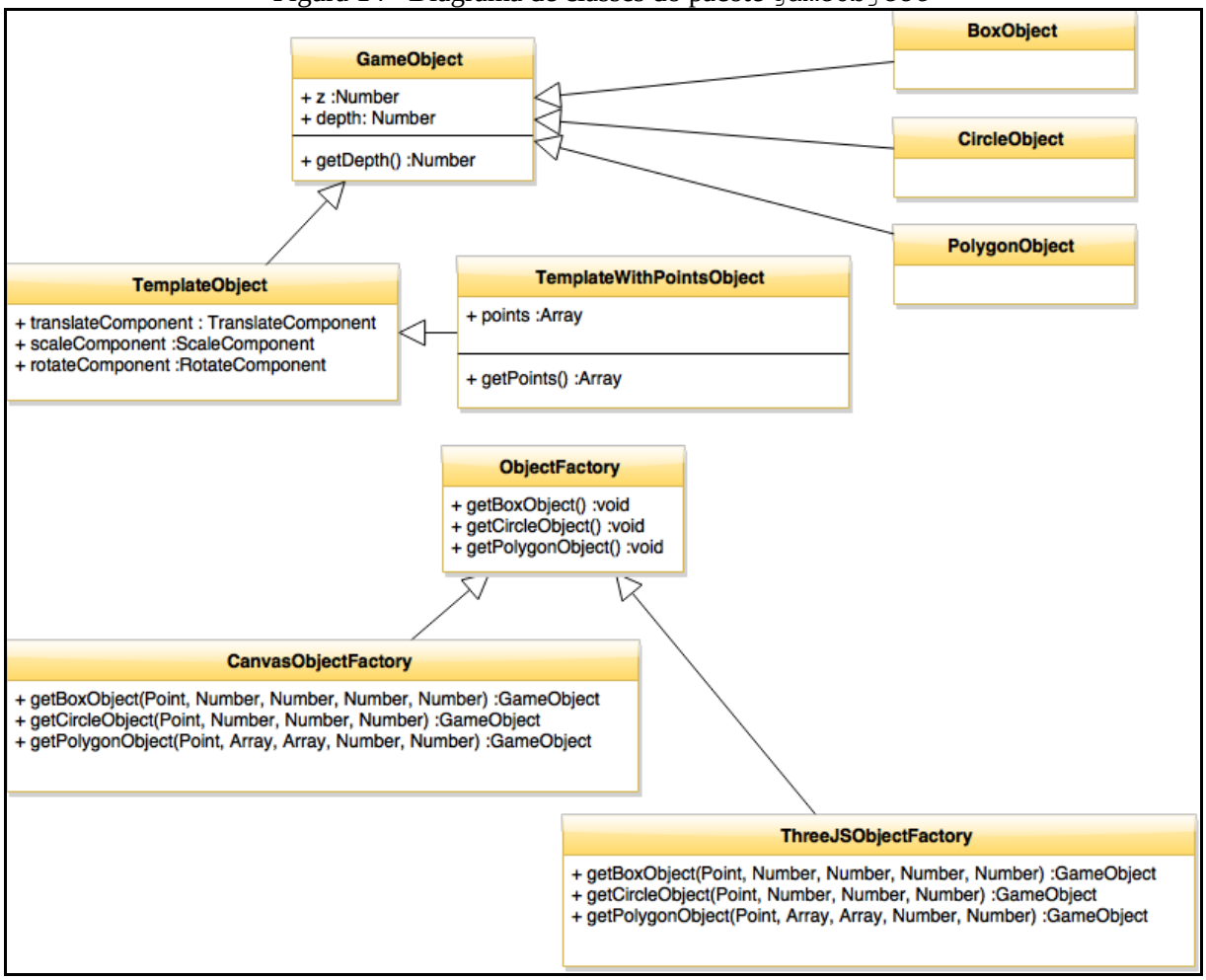


As classes desse pacote permaneceram as mesmas do motor original, mas algumas classes passaram por alterações. A `Camera` passou a suportar a terceira dimensão e o método `setLootAt`, capaz de controlar para onde a câmera irá apontar. A classe `Game`, responsável por gerenciar a execução do jogo, teve a adição dos atributos `apiHandler`, `componentFactory` e `objectFactory`, populados no também novo método `loadAPI`. A classe `Layer`, de forma semelhante aos componentes renderizáveis, ganhou o método `genThreeObject` para gerar um objeto capaz de fazer o controle de camadas no Three.js.

3.2.2.5 Pacote `gameobject`

O pacote `gameobject` possui a classe básica de um objeto capaz de ser interpretado pelo motor, junto de algumas especializações que visam integrar o objeto com componentes gráficos. Também está presenta uma fábrica abstrata capaz de gerar esses objetos especializados para garantir a compatibilidade entre o contexto do *canvas* e do Three.js. Diagrama de classes disponível na Figura 14.

Figura 14 - Diagrama de classes do pacote `gameobject`



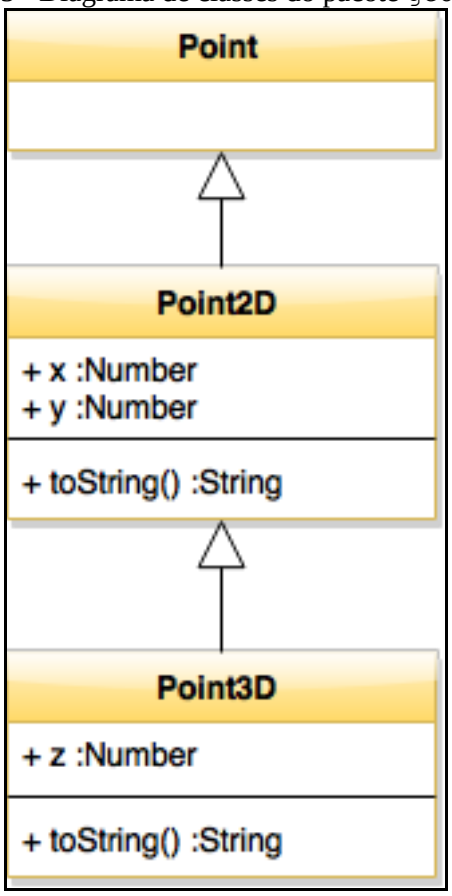
A classe `GameObject` passou a trabalhar com a terceira dimensão, ganhando assim os atributos `z` e `depth` e o método `getDepth` que retorna esta última variável. Não foram adicionados novos objetos específicos para comportar os componentes, como os já existentes BoxObject, CircleObject e PolygonObject, mas foram criadas as classes `TemplateObject` e, estendendo desta, `TemplateWithPointsObject`. O `TemplateObject` é um objeto com capacidades espaciais já preparadas e que pode receber um componente renderizável informado pelo usuário. O `TemplateWithPointsObject` se diferencia do `TemplateObject` pois visa comportar componentes baseados em pontos, como o `PolygonRenderComponent`.

A classe `ObjectFactory` foi criada para servir de interface para criação dos objetos específicos. Seguindo o padrão de projeto *abstract factory*, a fábrica é estendida por `TemplateObject` e `ThreeJSFactory`. O controle de qual dessas será utilizada é feito na criação do `Game`.

3.2.2.6 Pacote `geometric`

O pacote `geometric` possui classes que armazenam informações espaciais. Diagrama de classes pode ser visto na Figura 15.

Figura 15 - Diagrama de classes do pacote `geometric`

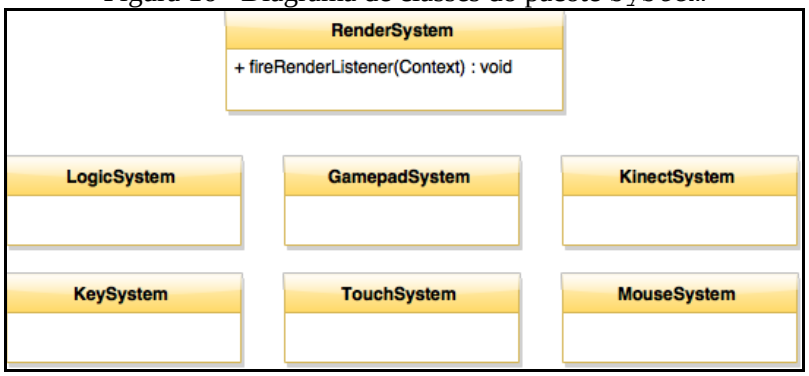


A classe `Point` atua apenas como interface para funções que são chamadas tanto no contexto bidimensional quanto no tridimensional. O `Point2D`, representa coordenadas para o contexto 2D, enquanto o `Point3D` atende ao contexto 3D.

3.2.2.7 Pacote `system`

O pacote `system` agrupa as classes responsáveis por disparar os eventos do motor. A Figura 16 mostra do diagrama de classes do pacote.

Figura 16 - Diagrama de classes do pacote `system`

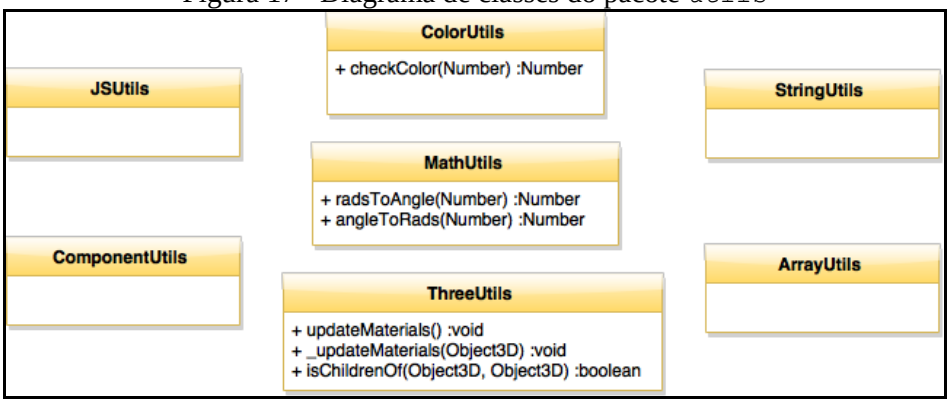


A única classe do pacote que foi alterada neste projeto foi o `RenderSystem`, fazendo que seu método `fireRenderListener` dispare os métodos de renderização do `apiHandler` da classe `Game`.

3.2.2.8 Pacote `utils`

O pacote `utils` contém as classes utilitárias do motor. O diagrama de classes do pacote pode ser acompanhado na Figura 17.

Figura 17 - Diagrama de classes do pacote `utils`



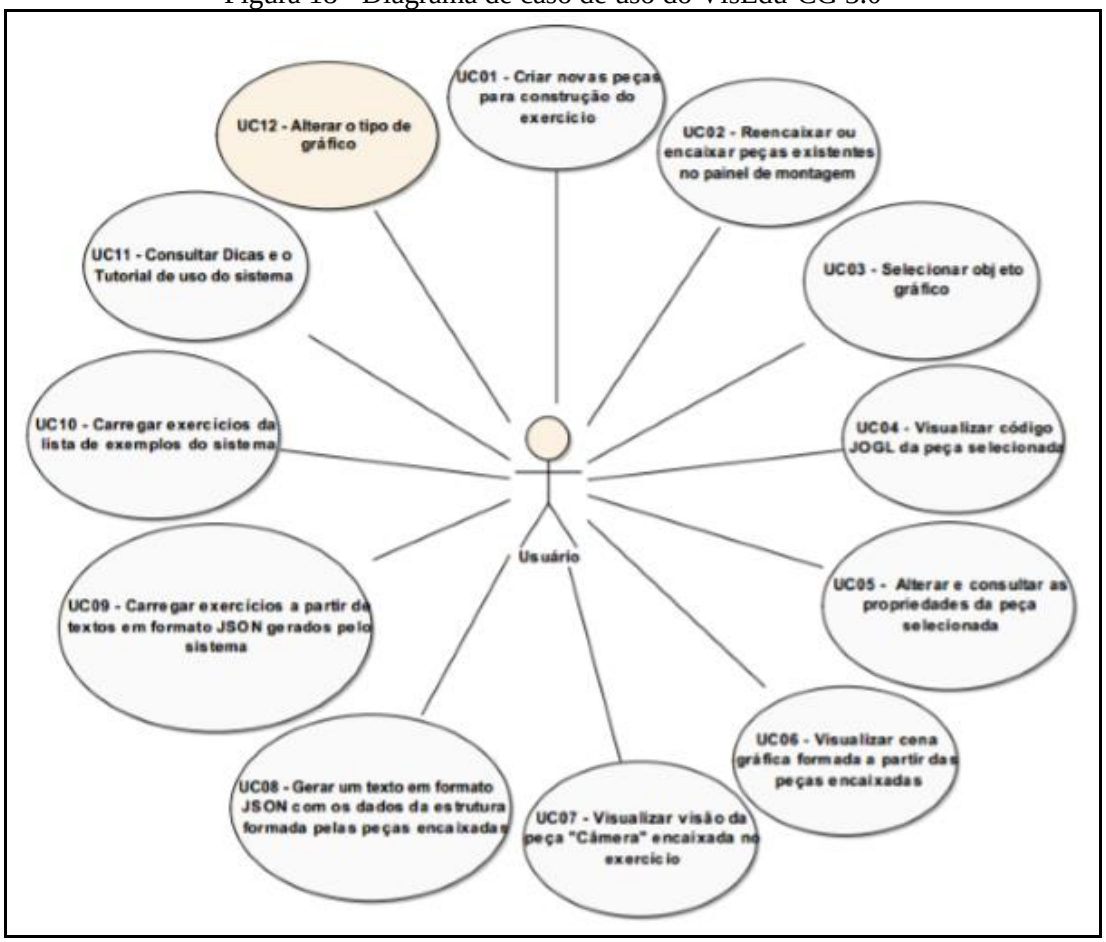
Durante a realização desse projeto, três novos utilitários foram adicionados. O primeiro é o `ColorUtils`, cujo método `checkColor` verifica um valor representa uma cor válida e, caso contrário, retorna uma cor padrão. O segundo seria o `MathUtils`, que possui métodos para converter ângulo para radianos e radianos para ângulo. O último novo utilitário é o

`ThreeUtils`, com o método `isChildrenOf`, que busca através de um algoritmo recursivo verificar se um objeto do Three.js é filho de outro, `_updateMaterials`, que atualiza o material de um objeto e todos os filhos (de forma também recursiva) e o método `updateMaterials`, que atualiza o material de todos os objetos em cena.

3.2.3 Casos de uso do visualizador de material educacional

Esta seção visa descrever os principais casos de uso do visualizador de material educacional. Todos os casos aqui apresentados foram trazidos da terceira versão do VisEdu-CG, por Nunes (2014b), que pode ser visto na Figura 18.

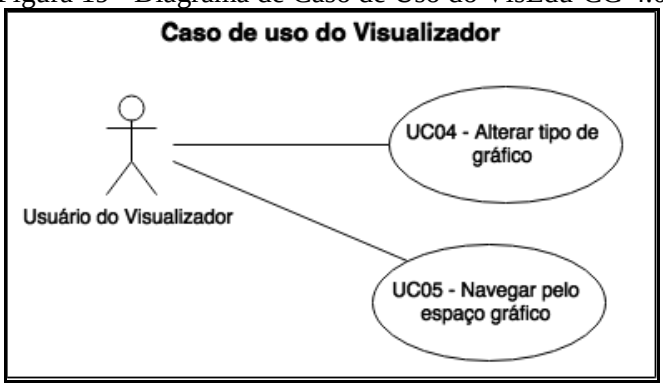
Figura 18 - Diagrama de caso de uso do VisEdu-CG 3.0



Fonte: Nunes (2014).

O caso de uso UC04 foi descartado pois o painel relacionado a ele foi removido para valorizar o espaço gráfico. O UC12, destacado na Figura 18, descreve a interação do usuário com os modos de renderização em 3D e 2D e sofreu por alterações. Também foi adicionado um caso de uso para descrever a ação do usuário manipular a câmera do espaço gráfico. O detalhamento dos casos e uso do visualizador pode ser visto no Apêndice A, enquanto o diagrama de casos de uso do VisEdu-CG 4.0 pode ser visto na Figura 19.

Figura 19 - Diagrama de Caso de Uso do VisEdu-CG 4.0



O Quadro 2 descreve a relação entre os casos de uso e os requisitos do visualizador de material educacional.

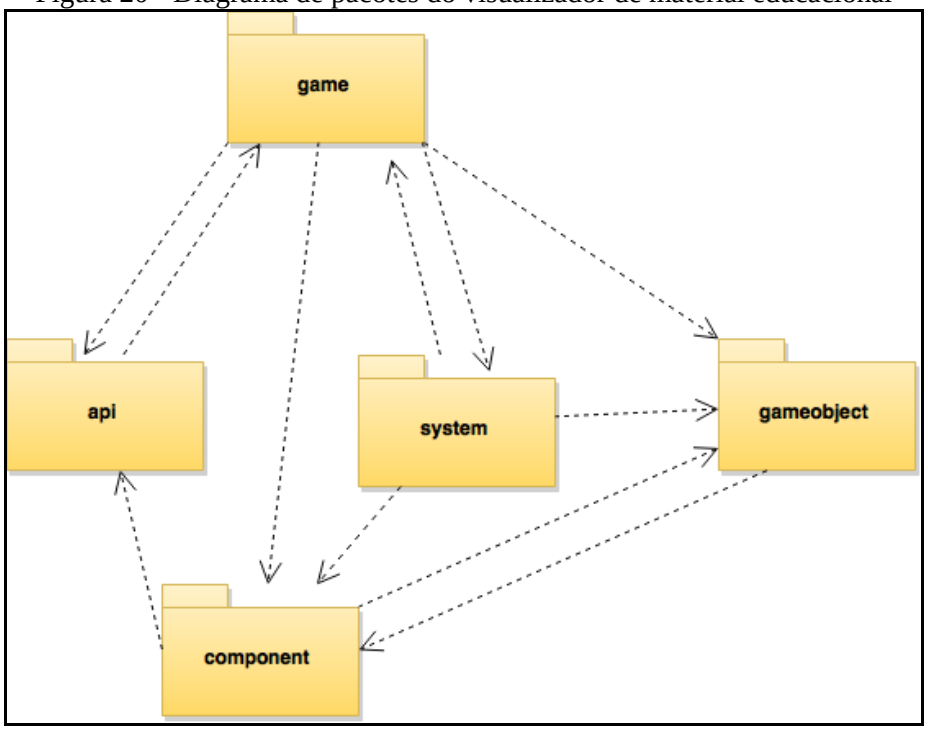
Quadro 2 - Relação entre casos de uso e requisitos do visualizador de material educacional

Caso de uso	Requisitos do visualizador de material educacional
UC04 – Definir modo de renderização	• permitir a transição entre 2D e 3D sem apagar as peças presentes;
UC05 – Navegar pelo espaço gráfico	• implementar controle orbital de câmera;

3.2.4 Diagrama de classes do visualizador de material educacional

Esta seção tem como objetivo apresentar os pacotes do visualizador de material educacional, assim como a relação e comunicação entre estes. O diagrama de pacotes do visualizador, contendo os pacotes mais relevantes do motor, pode ser visto na Figura 20.

Figura 20 - Diagrama de pacotes do visualizador de material educacional

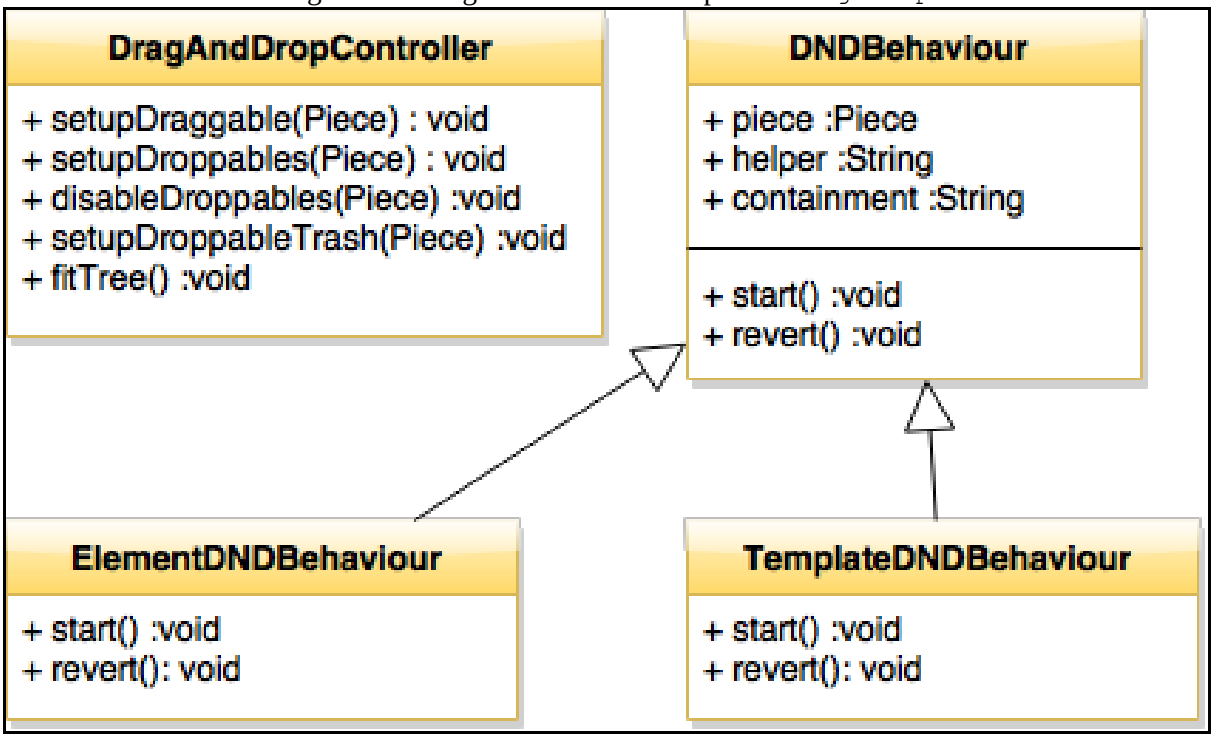


As seções subsequentes destinam-se a mostrar as classes destes pacotes e a relação mantida estre as mesmas.

3.2.4.1 Pacote `dragndrop`

O pacote `dragndrop` contém as classes necessárias para controlar o comportamento de arrastar e soltar uma peça, além de disparar os eventos da árvore quando isso acontecer. O diagrama das classes deste pacote pode ser vista na Figura 21.

Figura 21 - Diagrama de classes do pacote `dragndrop`

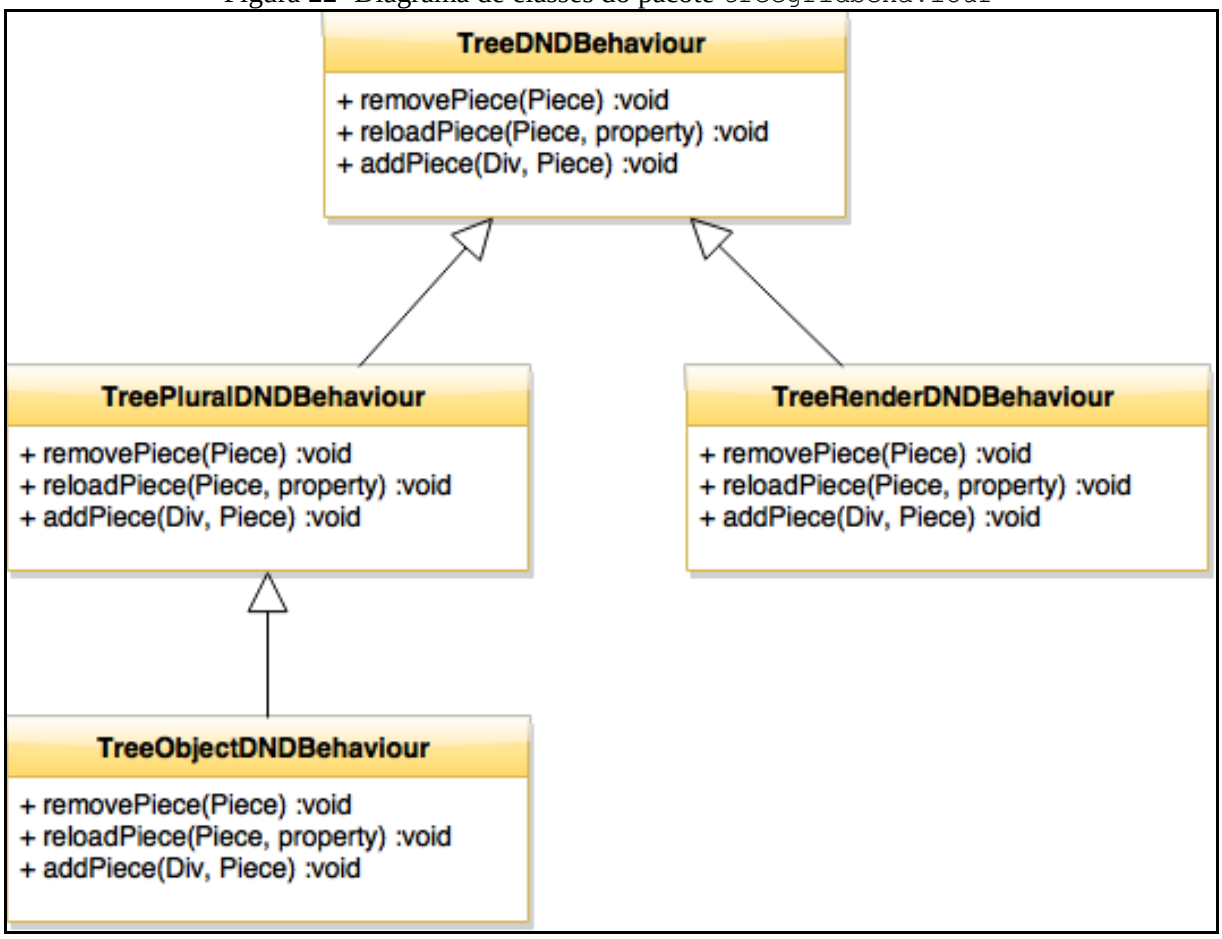


Este pacote contém a classe abstrata `DNDBehaviour`, que declara os valores de `helper` (o efeito da peça ao ser arrastada) e `containment` (os limites de até onde a peça pode ser arrastada), e as funções de `start` e `revert`, disparadas quando o evento de arrastar começa e quando o evento é cancelado, respectivamente. O `DNDBehaviour` é estendido por duas classes: `TemplateDNDBehaviour`, que representa as peças disponíveis na fábrica de peças, e `ElementDNDBehaviour`, representando as que já estão presentes na árvore de peças e podem ser alocadas para novos conectores ou serem removidas. Também está presente o `DragAndDropController`, um utilitário que possui as funções que habilitam e desabilitam o comportamento de arrastar e soltar das peças e disparar os eventos da árvore quando uma peça for encaixada em um novo conector.

3.2.4.2 Pacote `treebehaviour`

O pacote `treebehaviour` tem como objetivo controlar os comportamentos da árvore de peças quando uma destas é adicionada, realocada ou removida da mesma, e disparar os eventos referentes ao contexto gráfico. O diagrama representado pela Figura 22 demonstra as classes presentes no pacote.

Figura 22- Diagrama de classes do pacote `treegridbehaviour`

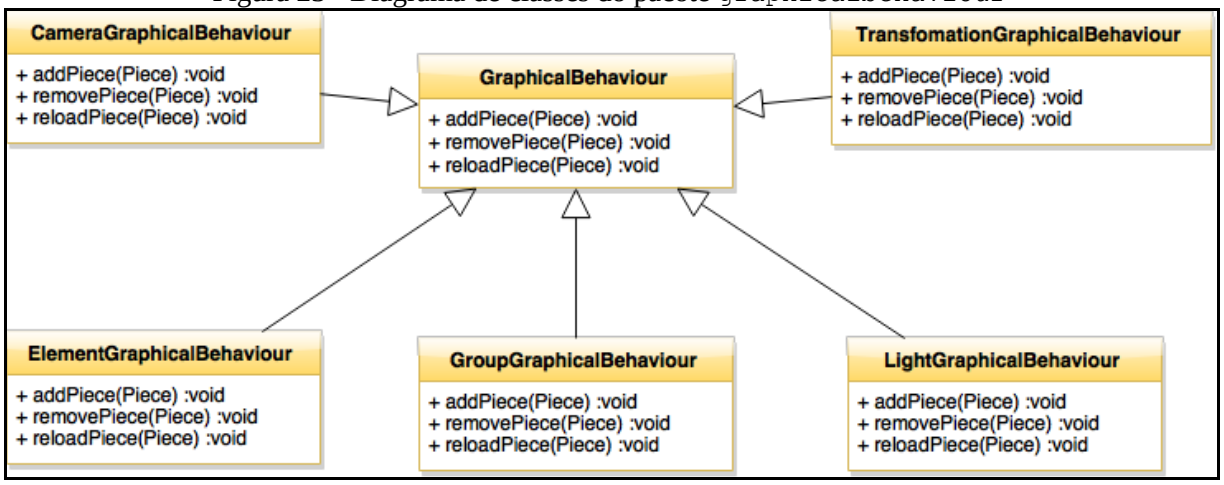


O `TreeDNDBehaviour` define os comportamentos básicos para as peças. O `TreePluraDNDBehaviour` estende esse comportamento básico, fazendo com que ao adicionar uma peça a um nó da árvore, este irá ser clonado, gerando assim um novo nó funcional à disposição do usuário. O `TreeObjectDNDBehaviour` estende a `TreePluraDNDBehaviour`, de forma que além de clonar a peça, é criado mais um nó dentro da mesma, permitindo a adição de diversas peças. O `TreeRenderDNDBehaviour` é destinado a peça de render que, apesar de não implementar eventos de arrastar ou soltar, deve disparar o evento de recarga.

3.2.4.3 Pacote `graphicalbehaviour`

As classes do pacote `graphicalbehaviour` são encarregadas de tratar os eventos gráficos de forma a adicionar, recarregar e remover os componentes da tela. O diagrama de classe deste pacote pode ser visto na Figura 23.

Figura 23 - Diagrama de classes do pacote `graphicalbehaviour`

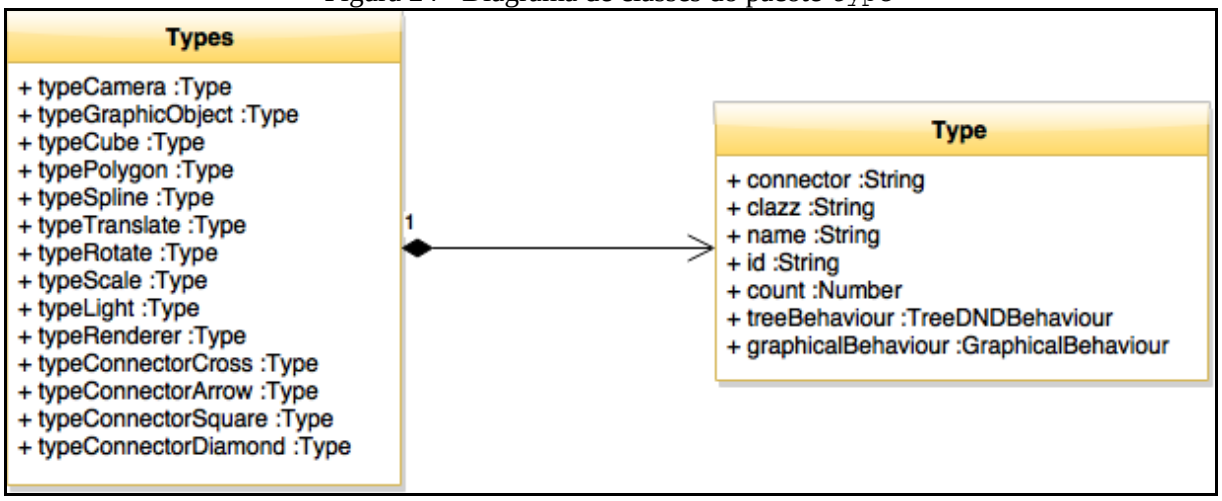


A classe `GraphicalBehaviour` fornece uma interface com os métodos de adicionar, remover e recarregar um elemento gráfico na cena. O `TransformationGraphicalBehaviour` implementa a interface para que a mesma trabalhe com componentes de transformação. O `LightGraphicalBehaviour` implementa a interface para que a mesma trabalhe com componentes de luz. O `GroupGraphicalBehaviour` implementa a interface de modo que a mesma trabalhe com elementos de camadas. O `ElementGraphicalBehaviour` implementa a interface para trabalhar com os componentes renderizáveis do visualizador, no caso, cubo, polígono e *spline*. O `ComeraGraphicalBehaviour` implementa a interface para suportar a manipulação de um componente de câmera.

3.2.4.4 Pacote `type`

O pacote de `type` é responsável por criar e manter a definição de um tipo de peça. Essa definição é composta por atributos como o nome deste tipo, o `TreeDNDBehaviour` e `GraphicalBehaviour` da peça. O diagrama de classes pode ser visto na Figura 24.

Figura 24 - Diagrama de classes do pacote `type`

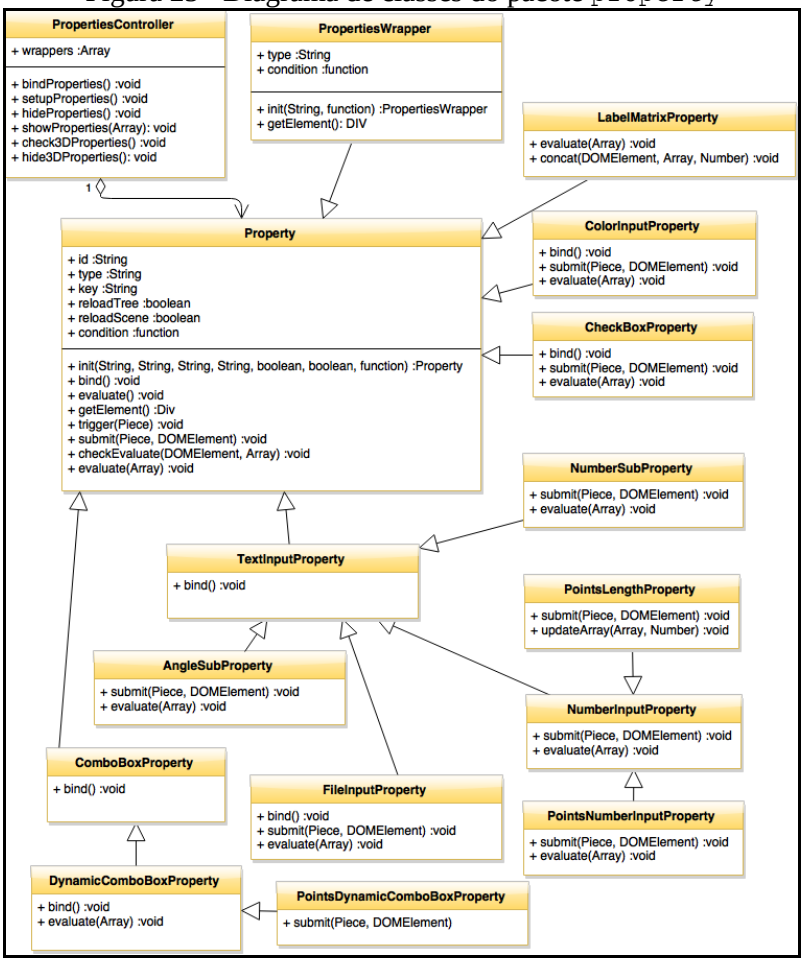


A classe `Type` guarda os atributos pertinentes a um tipo de peça. O atributo `connector` representa o tipo de encaixe da peça. O `clazz` representa a classe CSS utilizada pela peça. O `name` é o nome padrão da peça. O id é a identificação de peça no contexto HTML. O atributo `count` contabiliza quantas peças do respectivo tipo já foram criadas. Os atributos `treeBehaviour` e `graphicalBehaviour` contém os comportamentos de árvore e gráficos do tipo de peça, respectivamente. A classe `Types` opera como um enumerador, inicializando e disponibilizando os tipos de peça para o programa.

3.2.4.5 Pacote `property`

O pacote `property` comporta as classes que tem como objetivo gerenciar as propriedades das peças e manter os elementos visuais da aba de propriedades do visualizador. O diagrama de classes está presente na Figura 25.

Figura 25 - Diagrama de classes do pacote `property`



A classe `Property` trabalha como uma classe abstrata para as demais propriedades do visualizador. Seus métodos e atributos trabalham fazendo a manutenção das propriedades da peça, de sua criação, criação de eventos e passagem de valor entre objeto ao elemento HTML.

A `LabelMatrixProperty` manuseia propriedades cujo valores sejam matrizes que devem ser exibidas como texto. Enquanto a classe `CheckBoxProperty` manuseia propriedades booleanas representadas em tela como um *checkbox*. Já a classe`ColorInputProperty` manuseia propriedades de valor numérico representados em tela por um `input` do tipo `color`.

O `TextInputProperty` manuseia propriedades que será representado na tela com um elemento de texto simples. O `AngleSubProperty`, que estende do `TextInputProperty`, manuseia propriedades que são armazenadas em radianos mas exibidas em ângulo. O `FileInputProperty`, que estendo o `TextInputProperty`, manuseia propriedades contendo arquivos e são informados através de um `input` do tipo `file`. O `NumberInputProperty` manuseia propriedades numéricas representadas na tela por um campo numérico. O `PointsNumberInputProperty`, estendendo `NumberInputProperty`, representa valores

numéricos que se encontram dentro de um vetor de pontos, e será representado em tela como um campo numérico.

O `NumberSubProperty`, estendendo `TextInputProperty`, representa valores numéricos que se encontram dentro de um vetor, e será representado em tela como um campo numérico. Enquanto o `PointsLengthProperty`, estendendo `NumberInputProperty`, manuseia propriedades de vetores, alterando o tamanho dos mesmos, e representa na tela como um `spiner`. A classe `ComboBoxProperty` manuseia propriedades cujo valores são índices e exibe um `select` de acordo com o mesmo. Já a classe `DynamicComboBoxProperty`, estendendo `ComboBoxProperty`, manuseia propriedades que serão representadas em campos do tipo `select` que são montados dinamicamente. O Points`DynamicComboBoxProperty`, estendendo o `DynamicComboBoxProperty`, representa propriedades cujo valores são vetores de pontos e serão representados em tela por campos do tipo `select` montados dinamicamente.

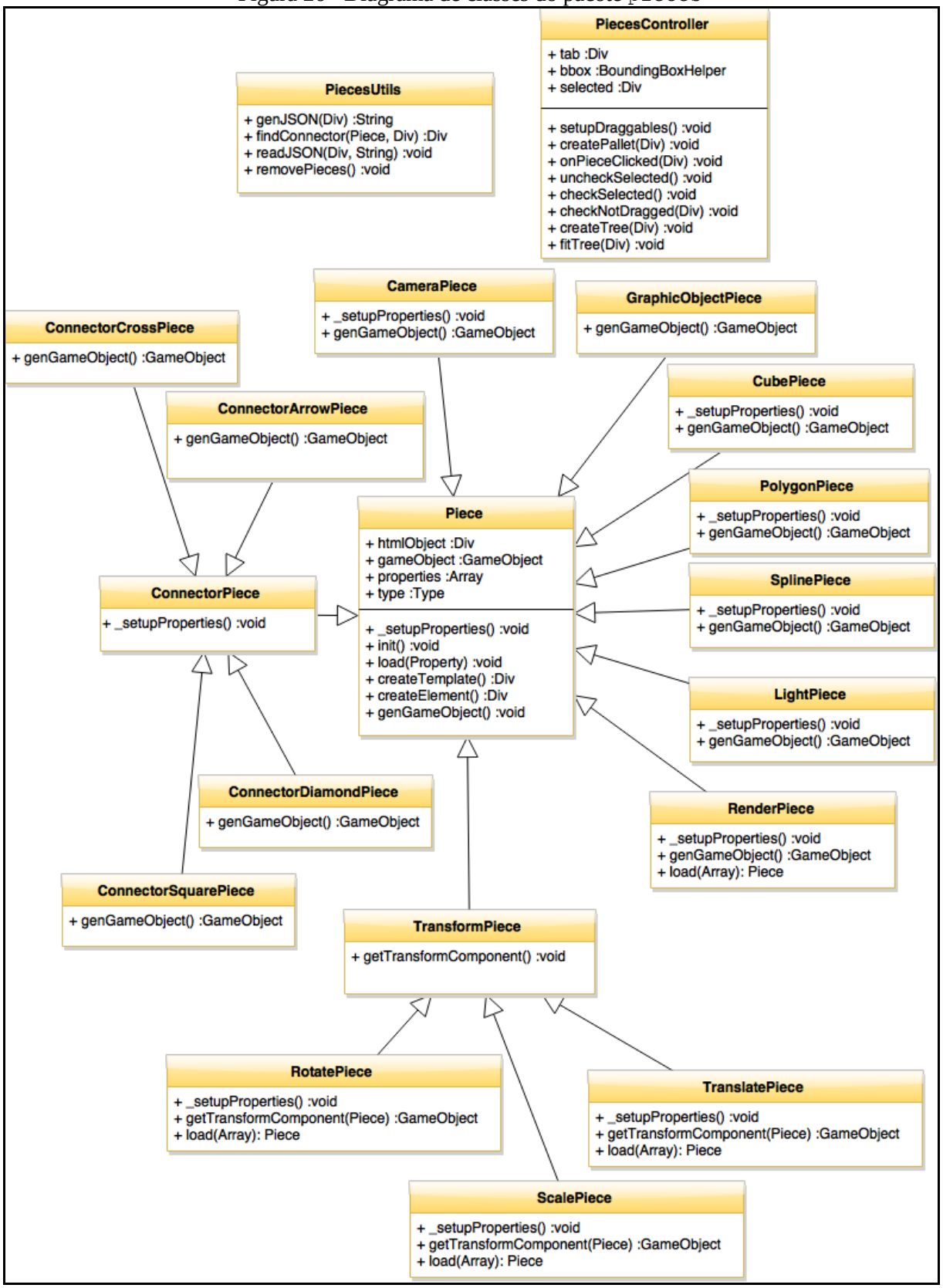
O `PropertiesWrapper` compartilha o mesmo comportamento das outras propriedades, com exceção de que não possui valores e de que é representado em tela por um campo que comporte outras propriedades. Assim, acaso passe por alguma situação onde sua visibilidade for afetada, as outras propriedades contidas nele também sofreram o efeito.

Por fim, o `PropertiesController` utiliza os tipos de propriedades descritos para criar as propriedades que serão ligados com os campos da tela. A classe também possui o método `setupProperties`, chamado toda vez em que uma peça é selecionada em cena ou na árvore, que faz o trabalho de verificar quais propriedades devem ser exibidas e qual devem ser ocultadas.

3.2.4.6 Pacote `piece`

O diagrama de classes do pacote `pieces` pode ser visto na Figura 26.

Figura 26 - Diagrama de classes do pacote `pieces`



O pacote `piece` possui as classes responsáveis por representar as peças do jogo e os utilitários e controladores necessários para a manutenção destas. A classe `Piece` opera como

uma classe abstrata para as demais peças. Ela contém atributos para o elemento HTML (`htmlObject`), elemento de cena (`gameObject`), as propriedades (`properties`) e o tipo (`type`) da mesma.

A `CameraPiece`, `GraphicObjectPiece`, `CubePiece`, `PolygonPiece`, `SplinePiece` e `LightPiece` são os componentes responsáveis, respectivamente, pelas peças de câmera, objeto gráfico, cube, polígono, *spline* e luz.

As peças responsáveis pelas transformações gráficas, `RotatePiece`, `TranslatePiece` e `ScalePiece`, estendem a interface `TransformPiece` e implementam o método `getTransformComponent` que retorna, a partir do objeto gráfico referenciado, o componente de transformação adequado para o tipo de peça.

Também estão presentes no pacote as peças responsáveis pelos conectores, implementando a interface `ConnectorPiece`, `ConnectorCrossPiece`, `ConnectorSquarePiece`, `ConnectorDiamondPiece` e `ConnectorArrowPiece`.

O `PiecesController` controla qual aba do visualizador estará à mostra, controla o contorno mostrado em torno da peça selecionada, cria a palheta de peças disponíveis na fábrica, controla os eventos de clique sobre as peças e cria a árvore de peças.

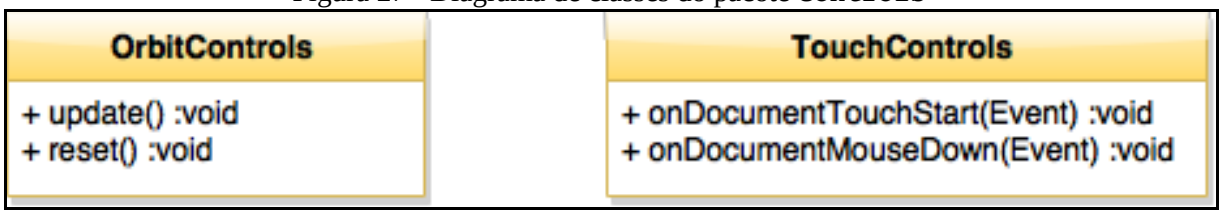
O método `genJSON` da classe `PieceUtils` gera um JSON representando o estado das peças presentes na cena. A função `findConnector` encontra um conector disponível para uma peça determinada. A função `readJSON` monta a árvore de componentes a partir de um JSON. O método `removePieces` limpa a árvore de componentes de todas suas peças.

Por último, o `RenderPiece` atua como a raiz da cena. Todas as peças adicionadas ao jogo pertencerão à hierarquia do dele, direta ou indiretamente.

3.2.4.7 Pacote `controls`

O pacote `controls` contém as classes responsáveis por permitir a interação direta entre o usuário do visualizador e a câmera da cena do programa. O diagrama de classes deste pacote segue na Figura 27.

Figura 27 - Diagrama de classes do pacote **`controls`**



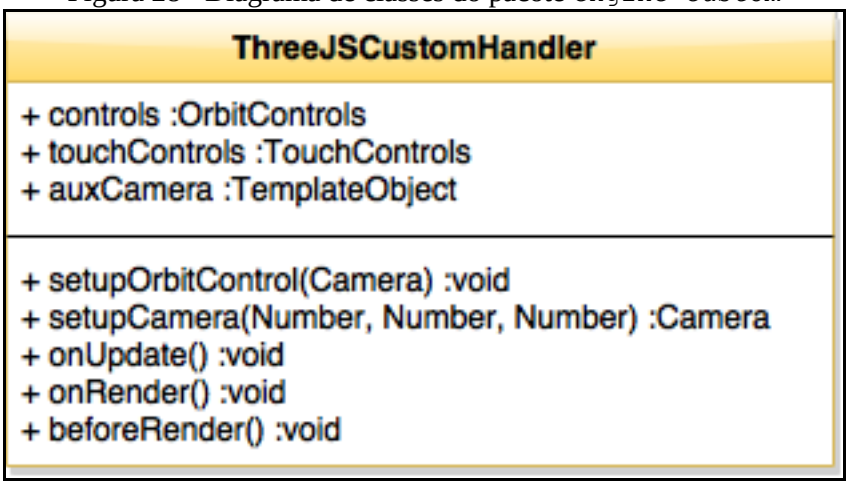
A classe `ObitControls` é responsável pelo controle da câmera principal do visualizador, incluindo rotação, zoom e deslocamento. A `TouchControls` é responsável por permitir a seleção das peças da cena quando o usuário efetua um duplo clique no mesmo.

Ambas classes são derivadas de exemplos disponíveis no site oficial do Three.js (THREE.JS, 2015).

3.2.4.8 Pacote `engine-custom`

O pacote `engine-custom` contém uma extensão do `ThreeJSHandler`, customizado para atender as necessidades do manipulador. O diagrama de classes pode ser visto na Figura 28.

Figura 28 - Diagrama de classes do pacote `engine-custom`

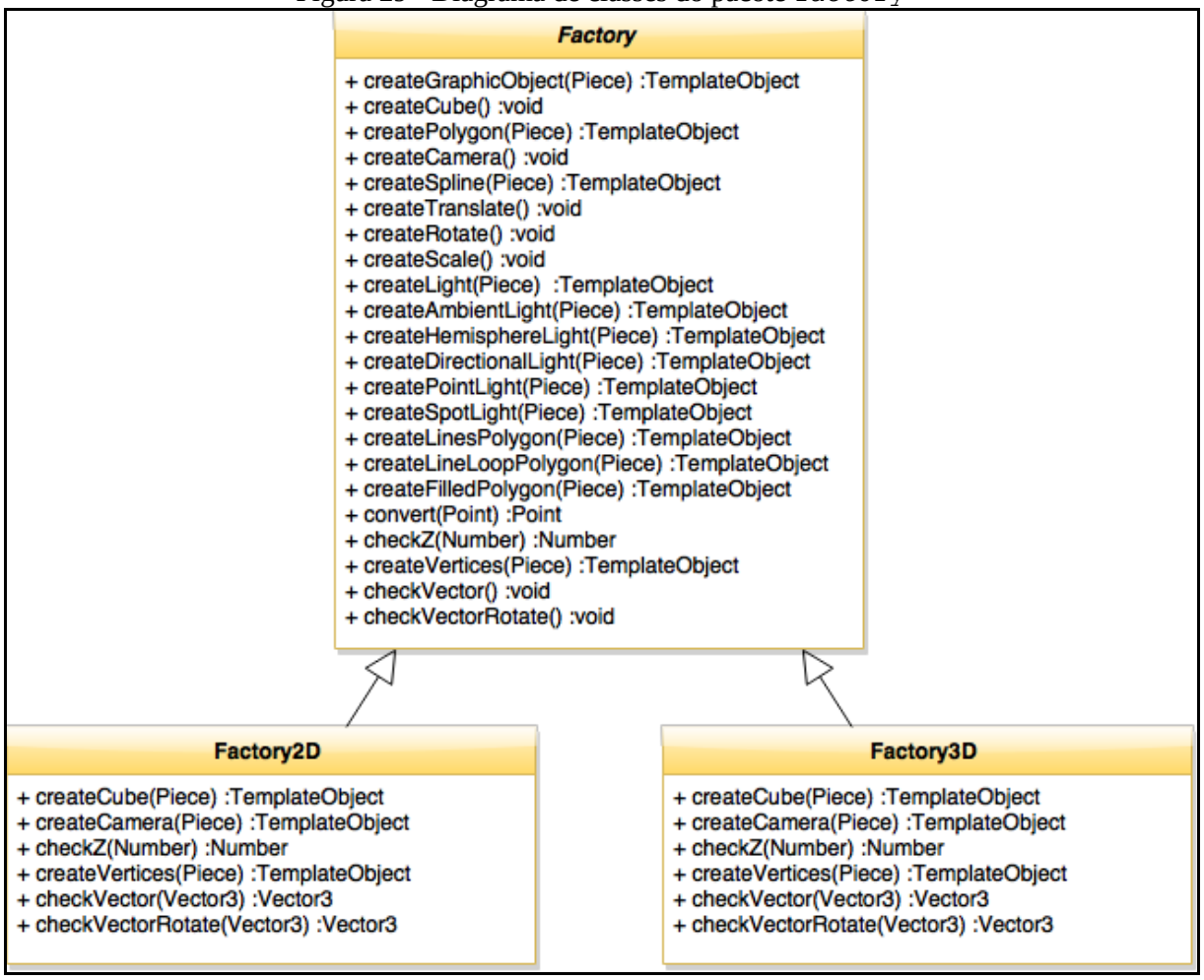


A classe `ThreeJSCustomHandler` é responsável por adicionar os controles de câmera e clique da cena, habilitar a segunda câmera que pode ser adicionada a partir das peças e criar o medidor de quadros por segundo, gerado através da biblioteca `Stats`. A classe também customiza o evento de renderização (`onRender`) para possibilitar a renderização do sistema de referência universal e atualizar o medidor.

3.2.4.9 Pacote `factory`

O pacote `factory` contém as classes necessárias para criação e manutenção das peças a partir de suas propriedades para o contexto 2D e 3D. O diagrama de classes deste pacote pode ser visto na Figura 29.

Figura 29 - Diagrama de classes do pacote `factory`

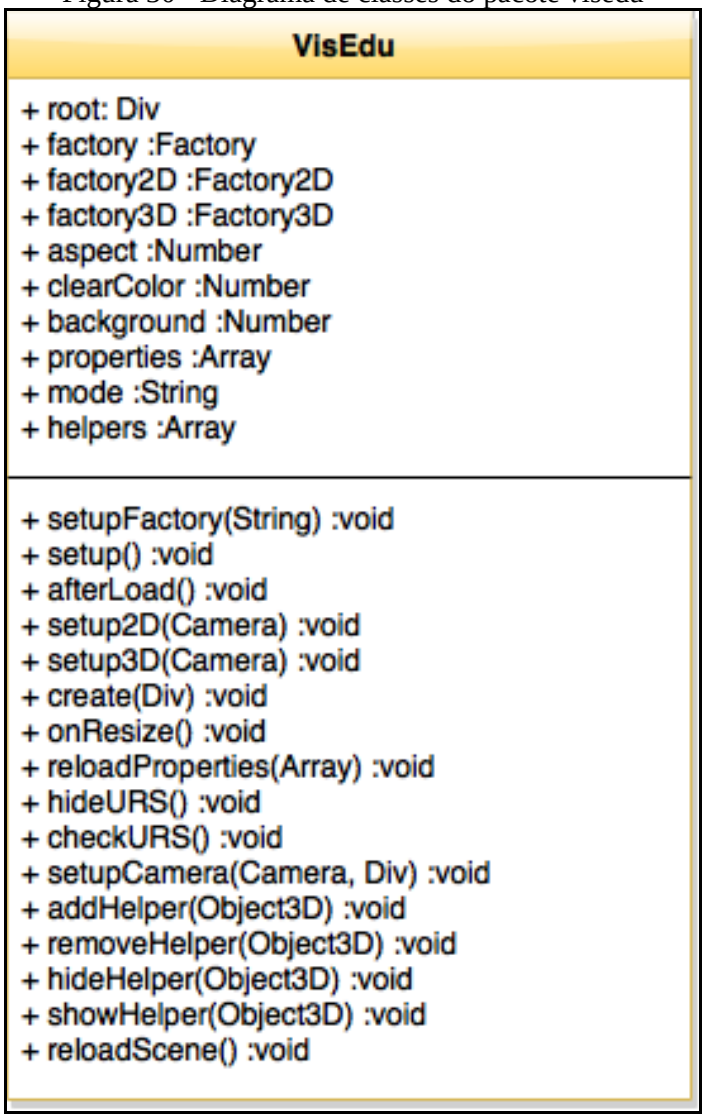


A classe `Factory` atua se baseando no padrão de projeto de fábrica abstrata (porém sem o uso de interfaces, visto que o Javascript não as possui). A mesma é responsável por gerar os objetos que são adicionados à cena quando uma peça é adicionada na árvore. O `Factory2D` estende a `Factory`, de forma a gerar os objetos específicos para o contexto 2D. O `Factory3D` possui as implementações específicas para o modo 3D.

3.2.4.10 Pacote `visedu`

O pacote `visedu` possui a classe central na utilização do visualizador. O diagrama de classes pode ser visto na Figura 30.

Figura 30 - Diagrama de classes do pacote visedu

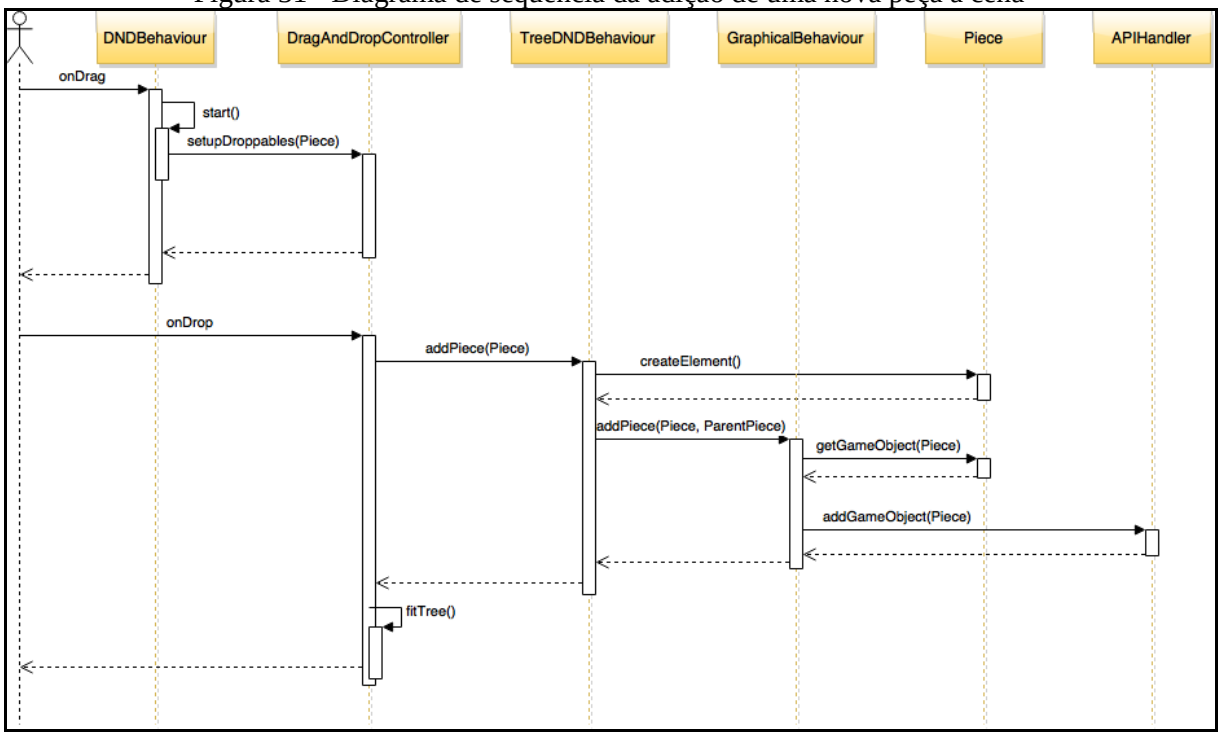


A classe `VisEdu` é responsável por inicializar o `ThreeJSCustomHandler` que será utilizado pelo resto do programa, gerenciar o modo de renderização (2D ou 3D), manter a segunda câmera, controlar as propriedades que são mantidas pela peça `Renderer`.

3.2.5 Diagrama de sequência

Esta seção apresenta o diagrama de sequência do visualizador de material educacional, VisEdu-CG, no processo de adicionar uma peça de elemento gráfico, como um cubo, polígono ou `spline`. O diagrama pode ser visto na Figura 31.

Figura 31 - Diagrama de sequência da adição de uma nova peça à cena

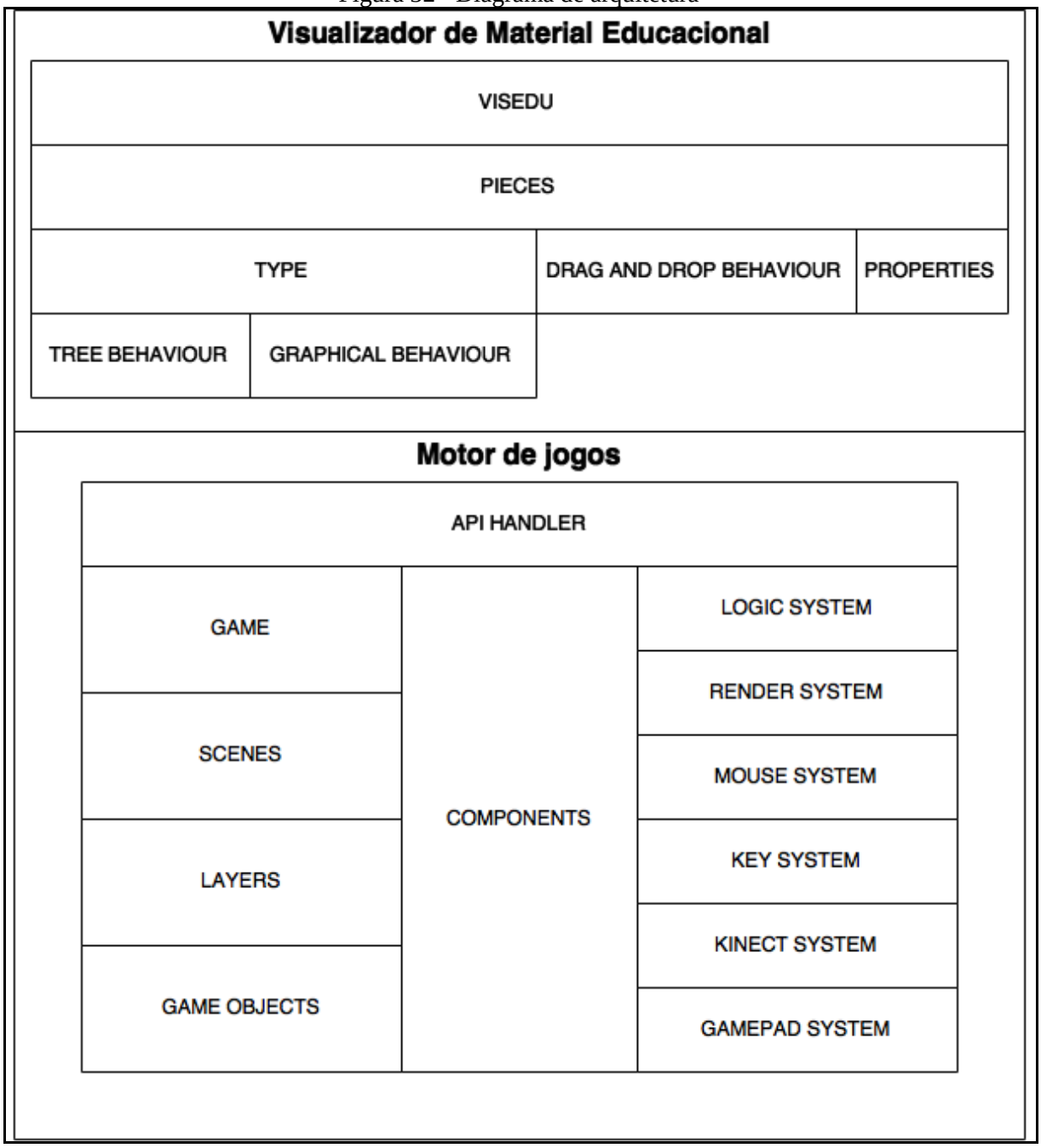


O processo inicia no momento em que o usuário começa a arrastar uma peça. Com esse movimento, ele disparará os eventos da peça e invoca a função start do `DragAndDropController`. Nesse momento, o controlador irá habilitar os conectores a receberem a peça em questão e programar os `callbacks` dos mesmos. O próximo passo ocorre quando a peça for salta em cima de um desses conectores e, assim, disparar o `callback` deixado no `DragAndDropController`. Esse `callback` invocará o método `addPiece` do `TreeDNDBehaviour` associado ao objeto `type` da peça. Essa função chamará o `createElement`, criando assim o elemento gráfico na árvore de peças. Em seguida o método `addPiece` do `GraphicalBehaviour` é invocado, gerando assim um Objeto do jogo. Por último, esse objeto é adicionado à cena quando chamado o método `addGameObject` do `APIHandler`.

3.2.6 Diagrama da arquitetura

Esta Seção apresenta o diagrama de arquitetura do motor de jogos após passar pelas modificações aplicadas para suportar o modo de renderização tridimensional, junto com a arquitetura utilizada pelo visualizador do material educacional (Figura 32).

Figura 32 - Diagrama de arquitetura



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas, ferramentas utilizadas e a operacionalidade da integração do motor de jogos com o visualizador de material educacional.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento do motor de jogos (HARBS, 2013) fez uso da linguagem Javascript e a biblioteca Three.js, para abstração do WebGL.

O visualizador de material educacional foi desenvolvido com as linguagens HTML5, Javascript e JQuery. O JQuery foi utilizado para definir os comportamentos de arrastar e soltar das peças, além de fazer o controle de abas e diálogos do visualizador.

O ambiente de desenvolvimento adotado para ambos foi o Eclipse 4.4.1 (ECLIPSE, 2015). Os navegadores utilizados durante o desenvolvimento foram o Google Chrome 43.0.2357.130 m, o Internet Explorer 11.0.9600.17842 e o Mozilla Firefox 38.0.5.

Para execução do visualizador, foi usado um computador *desktop* Windows 8.1, com processador AMD FX 8150, 8 GB DDR3 de memória RAM e placa de vídeo AMD Radeon HD 7950.

3.3.2 O motor de jogos

Esta seção descreve os conceitos utilizados na implementação realizada sobre a arquitetura do motor de jogos para viabilizar a renderização em modo tridimensional e com a nova tecnologia gráfica.

3.3.2.1 Arquitetura do motor de jogos

Para disponibilizar a nova forma de renderização com o Three.js foi necessário fazer diversas alterações na arquitetura do motor de jogos. O primeiro passo, após estudar as diferenças e semelhanças da nova tecnologia com o *canvas* do HTML5, foi identificar o código específico para manutenção do contexto da API do *canvas*, isolá-lo em uma classe própria e disponibilizar o acesso a mesma a partir da classe central do motor, o `Game`. Os trechos de código identificados foram aqueles responsáveis pela manutenção do elemento de *canvas* do HTML, configuração da câmera do jogo, criação da cena principal, inicialização do *loop* do jogo, obtenção do contexto do *canvas*, renderização dos objetos, adição e remoção de objetos da cena. Com o código isolado, os diversos pontos do sistema que consumiam a API do *canvas* diretamente passaram a invocar os métodos da classe `HTML5CanvasHandler`. Com o código específico isolado, foi criada a interface `APIHandler` especificando os métodos presentes na `HTML5CanvasHandler`. A interface pode ser visualizada no Quadro 3.

Quadro 3 - Interface de manuseador de API

```
01 /**
02  Interface de manipulação de APIs
03  *
04  */
05 function APIHandler() {}
06
07 APIHandler.prototype.setupCanvas = function () {}
08 APIHandler.prototype.setupCamera = function () {}
09 APIHandler.prototype.setupScene = function () {}
10 APIHandler.prototype.startGameLoop = function () {}
11 APIHandler.prototype.getContext = function () {}
12 APIHandler.prototype.render = function () {}
13 APIHandler.prototype.onRender = function () {}
14 APIHandler.prototype.beforeRender = function () {}
15 APIHandler.prototype.addGameObject = function () {}
16 APIHandler.prototype.removeGameObject = function () {}
17 APIHandler.prototype.newGameElement = function () {}
18 APIHandler.prototype.getComponentFactory = function () {}
19 APIHandler.prototype.getObjectFactory = function () {}
```

Com essa interface definida, foi criada a classe `ThreeJSHandler`, uma classe que implementava a `APIHandler` e que seria responsável por manter o ciclo de vida do Three.js. O resultado esse processo gerou duas classes, cada uma especializada para atender uma API, e uma interface que assegurava que estas iriam implementar os métodos necessários para determinados contextos do sistema. Com isso, a classe `Game` passou a permitir que o usuário defina qual API será usada para renderização a partir do método `loadAPI`. O método `loadAPI` permite que o usuário do motor passe como argumento uma `String` ou um objeto do tipo `APIHandler`, podendo este ser um objeto customizado pelo usuário do motor. Caso o usuário não chame o método, a função `init` da classe game o invocará, passando como parâmetro o `HTML5CanvasHandler`. Sendo assim, caso haja intenção de usar um manuseador que não o do *canvas*, o mesmo deve ser informado antes de chamar o `init` da classe Game. Os métodos `loadAPI` e `init` podem ser vistos no Quadro 4.

Quadro 4 - Método `loadAPI` e `init` da classe `Game`

```
01 /**
02 * Classe que representa o jogo.
03 *
04 * @author Marcos Harbs
05 * @class Game
06 * @static
07 */
08 var Game = new function(){
09
10       ...
11       this.apiHandler = null;
12       ...
13
14       this.loadAPI = function (api) {
15             if (api) {
16                   if (api == 'ThreeJS') {
17                         Game.apiHandler = new ThreeJSHandler();
18                   } else {
19                         Game.apiHandler = api;
20                   }
21             } else {
22                   Game.apiHandler = new HTML5CanvasHandler();
23             }
24             Game.componentFactory =
25 Game.apiHandler.getComponentFactory();
26             Game.objectFactory = Game.apiHandler.getObjectFactory();
27       }
28
29       /**
30       * Método que inicializa o jogo.
31       *
32       * @author Marcos Harbs
33       * @method init
34       * @param {Element} element
35       * @param {Scene} scene
36       */
37       this.init = function(element, scene){
38             if (!this.apiHandler) {
39                   this.loadAPI();
40             }
41             this.canvas = Game.apiHandler.setupCanvas(element);
42             this.context = Game.apiHandler.getContext();
43             this.camera = Game.apiHandler.setupCamera();
44             this.loadGame();
45             this.setScene(scene);
46             this.lastUpdateTime = 0;
47             this.frameRate = 60;
48             this.apiHandler.startGameLoop();
49             this.running = true;
50             this.paused = false;
51       }
...
```

Com a lógica de renderização funcionando, o próximo passo foi modificar os componentes atuais para que estivessem aptos à serem renderizados pelo Three.js. Para isso os componentes encarregados de gerar elementos gráficos passaram a implementar o método `genThreeObject`, que retorna um objeto tridimensional montado pela API do Three.js. As

classes de camada (`Layer`), cena (`Scene`) e câmera (`Camera`) também tiveram a adição desse mesmo método. Os componentes encarregados de aplicar transformações espaciais (`RotateComponent`, `ScaleComponent` e `TranslateComponent`) foram alteradas de forma diferentes; os mesmos passaram a fazer o papel de interfaces e cada um ganhou duas especializações, sendo uma para atender a API do `canvas` e outro a do `Three.js`. Para controlar qual dessas especializações são carregadas, foi criada a fábrica `ComponentFactory`, que pode ser vista no Quadro 5.

Quadro 5 - Interface da fábrica abstrata de componentes

```
01 /**
02  * Fábrica que controla a criação de componentes que estão
03 especializados para alguma API específica
04  */
05
06 function ComponentFactory() {}
07
08 ComponentFactory.prototype.getRotateComponent = function() {}
09 ComponentFactory.prototype.getScaleComponent = function() {}
10 ComponentFactory.prototype.getTranslateComponent = function() {}
```

Assim, toda vez que for requisitado um componente de transformação, ele deve ser obtido através da fábrica, que deve ser acessada através do método `getComponentFactory` declarado na `APIHandler`.

Ocasionalmente, com a mudança de tecnologia e demandas do visualizador gráfico, surgiu a necessidade de criar algumas funções específicas para o Three.js apenas, como o `getBasicMaterial`. Essas rotinas foram alocadas para ficar junto com as outras funções do `ThreeJSHandler`. Caso em futuras implementações surja a necessidades de usar funções semelhantes no *canvas* ou então for adicionada uma nova API gráfica que compartilhe esses comportamentos, esses métodos específicos devem ser declarados pela interface `APIHandler`.

3.3.2.2 Novos componentes do motor de jogos

As exigências do visualizador de material educacional resultaram na criação de novos componentes para o motor de jogos. Para a composição do sistema de referência universal, foi criada classe `AxisRenderComponent`. Também foi criado o um componente para desenhar o grid, `GridRenderComponent`.

Para desenhar as peças com encaixe quadrado, foram adicionados os componentes `CircleRenderComponent`, `CubeRenderComponent`, `LinesRenderComponent`, `SplineRenderComponent` e `SpheresRenderComponent`.

Para dar a funcionalidade desejada à peça de `Objeto Gráfico` foi criado o componente `GroupObject`.

Para possibilitar a segunda câmera do visualizador foi criada a classe `CameraComponent`.

Para permitir o manuseio de luz, foi criada a classe `LightComponent` e suas especializações: `AmbientLightComponent`, `SpotLightComponent`, `PointLightComponent` e `DirectionalLightComponent`.

Durante o desenvolvimento do sistema foi sentida a necessidade de associar esses componentes a um objeto gráfico com capacidades espaciais de uma forma mais prática, pois o `GameObject` não contém essas capacidades por padrão e os outros objetos já presentes no motor são específicos para um tipo de componente. Assim, criou-se a classe `TemplateObject`. Esta, que estende o `GameObject`, adiciona automaticamente os componentes de rotação, escala e translação, além de um quarto componente que será passado como parâmetro.

3.3.3 O visualizador de material educacional

O visualizador de material educacional, VisEdu-CG, é uma ferramenta que permite o usuário a montar uma cena gráfica, composta por elementos de cubo, polígono, *spline* e luz, arrastando peças que representam esses conceitos para a árvore de renderização. O visualizador também permite que as propriedades das peças sejam alteradas, transformações espaciais sejam aplicadas sobre as peças, alternar o modo de visualização entre 2D e 3D, exportar as peças para um arquivo em formato JSON e importá-las via arquivo ou texto neste mesmo formato.

Para o desenvolvimento deste trabalho, a ferramenta foi completamente reformulada. Essa decisão foi tomada para que o visualizador estivesse completamente adequado com o motor de jogos. Uma das maiores diferenças do visualizador foi alterá-lo para que apenas o painel da cena e da segunda câmera sejam desenhados através do Three.js. O restante da tela usa apenas HTML, Javascript e JQuery.

O Visualizador conta com quatro abas: `Arquivos`, `Fábrica de peças`, `Propriedades da Peça` e `Ajuda`. Cada uma delas é uma `div`, disposta em forma de aba através da função `tab` do jQuery. Todos os painéis presentes já estavam presentes na versão 3.0 do VisEdu-CG. Com a versão 4.0, a maioria destes passaram por algumas alterações, e os painéis de `Comandos em JOGL` e `Lista de Peça`s deixaram de compor a aplicação.

Uma peça é um objeto Javascript da classe `Piece` que possui um tipo (da classe `Type`), que gerencia informações como nome, classe CSS e comportamentos de árvore e cena, uma

referência para um elemento HTML, sendo este a representação visual da peça, um `GameObject` do motor de jogos, uma lista de propriedades e um objeto de comportamento gráfico. O ciclo de vida da peça começa quando a mesma é criada e adicionada em tela, através do método `genElement`, que pode ser visto no Quadro 6.

Quadro 6 - Método **`genElement`** da classe **`Piece`**

```
01 Piece.prototype.genElement = function(isTemplate) {
02       var str = "<div class='piece resting ";
03       var txt;
04       var name = this.properties['name'];
05
06       if (name) {
07             txt = "<div class='txt'>";
08       }
09
10       if (isTemplate) {
11             str += "template ";
12             if (txt) {
13                   txt += this.type.name;
14             }
15       } else {
16             str += "element ";
17             if (txt) {
18                   txt += name;
19             }
20       }
21
22       str += this.type.clazz + " " + this.type.connector +
23             "' >";
24
25       if (this.type.connector) {
26             str +="<img/>";
27       }
28
29       if (txt) {
30             str += txt + "</div>";
31       }
32
33       str += "</div>";
34       this.htmlObject = $(str).data('piece', this);
35       if (isTemplate) {
36             this.onDragBehaviour = new TemplateDNDBehaviour();
37       } else {
38             this.onDragBehaviour = new ElementDNDBehaviour();
39             this._addMouseEvent();
40       }
41
42       this.onDragBehaviour.piece = this;
43       return this.htmlObject;
44 }
```

O valor `isTamplate` será verdadeiro caso a peça esteja sendo criada para ser um modelo na fábrica de peças ou falso caso seja um elemento da árvore de renderização. Neste método, é feita a montagem do elemento HTML da peça e é definido o `onDragBehaviour` (do tipo DNDBehaviour), que define como um objeto se comporta quando for arrastado.

Após criado, o elemento HTML da peça é configurado para que o mesmo possa ser arrastado pela tela. Essa configuração é feita com o JQuery, a partir do método `draggable`. Quando a peça começa a ser arrastada, serão buscados os conectores livres (ou seja, sem peça associada) e eles serão configurados para que as peças possam ser soltas neles, com o método `droppable` do JQuery.

Por meio do padrão de projeto `Strategy`, os comportamentos de árvore e da cena gráfica de cada tipo são organizados e armazenados nos atributos `treeBehaviour` e `graphicalBehaviour` da classe `Type`. Quando uma peça é solta no conector, ela buscará o comportamento de árvore de seu tipo e o disparará. Isso fará com que outra peça de mesmo tipo seja criada e adicionada a árvore e, dependendo do comportamento do tipo, clonará o conector agora ocupado.

Depois de fazer as devidas operações na árvore, será invocado o comportamento gráfico relativo ao tipo da peça que foi encaixada. Esse comportamento poderá ser de câmera (`CameraGraphicalBehaviour`), de grupo (`GroupGraphicalBehaviour`), de elemento gráfico (`ElementGraphicalBehaviour`), de transformação (`TransformationGraphicalBehaviour`) ou de luz (`LightGraphicalBehaviour`). Caso o comportamento seja do tipo de transformação, ele trabalhará com o componente de transformação respectivo presente na peça de `Objeto Gráfico` em que a peça estiver encaixada (visto que as transformações só podem ser encaixadas dentro desse tipo). Caso contrário, será instanciado um novo `GameObject` através da fábrica de componentes disponibilizada pela classe `VisEdu`.

Da mesma forma em que a peça pode disparar um comportamento de árvore e encadear um evento gráfico caso seja adicionada a um novo conector, esse comportamento também opera quando a peça é removida (arrastada para a lixeira) ou recarregada (caso seja alocada para outro conector, suas propriedades mudem ou o modo de visualização seja alterado).

As propriedades de uma peça são armazenadas num vetor em cada objeto da classe `Piece`. Os objetos do tipo `Property`, inicializados na classe `PropertyController`, são responsáveis por popular os campos da aba `Propriedades da Peça` com os valores da peça selecionada no visualizador, detectar quando o valor de uma propriedade é alterado e atualizar esse dado na peça, controlar a visibilidade dos campos, apresentar o valor de forma que seja mais legível para o usuário e armazenar o valor da forma mais prática computacionalmente. Na página HTML, as propriedades possuem uma identificação própria e uma segunda para indicar o grupo a que pertencem. Esses dados são comparados com a peça, quando for

selecionada, para definir se a propriedade deve ser exibida. O Quadro 7 mostra o método que verifica a visibilidade das peças.

Quadro 7 - Método `setupProperties` da classe `PropertiesController`

```
this.setupProperties = function() {
            var selected = PiecesController.selected;
            var piece = $(selected).data('piece');
            if (selected) {
                  var properties = piece.properties;
                  var fields = $('.property.'+piece.type.id);
                  this.showProperties(fields);
                  $.each(fields, function(index, item) {
                        $.each($(item).find('.prop-field'),
function(index, field) {

      $(field).data("property").checkEvaluate(item, properties);

                        });
                  });
                  $.each(PropertiesController.wrappers, function(index,
item) {
                        item.check(properties);
                  });
            } else {
                  this.hideProperties();
            }
            PropertiesController.check3DProperties();
      }
```

As propriedades também carregam um sinalizador que indica quais peças são exclusivas do contexto tridimensional. A classe `VisEdu` tem como principais objetivos armazenar as propriedades que são informadas através da peça `Renderer`, que também é a raiz da árvore de peças, e manter o modo e fábrica de objetos do visualizador. Quando o modo for 2D, o atributo `factory` apontará para o que referência a Factory2D. Quando o modo for 3D, o atributo `factory` apontará para a Factory3D. Essas fábricas trabalham seguindo o padrão de projeto de *Abstract Factory*. Caso a propriedade seja alterada, a cena será recarregada, mantendo as peças com as mesmas propriedades porém desenhando no novo formato. Esse controle pode ser acompanhado no Quadro 8.

Quadro 8 - Método `setupFactory` da classe `VisEdu`

```
01 this.setupFactory = function(mode) {
02             var current = this.factory;
03             if (mode == '3D') {
04                   this.factory = this.factory3D;
05                   this.setup3D();
06             } else if (mode == '2D') {
07                   this.factory = this.factory2D;
08                   this.setup2D();
09
10             } else {
11                   console.log('[VisEdu] no factory for mode: '+mode);
12             }
13             this.mode = mode;
14             PropertiesController.check3DProperties();
15             if (current && this.factory != current) {
16                   this.reloadScene();
17             }
18       }
```

A aba de arquivo apresenta as funcionalidades de exportação e importação da cena. Caso opte-se pela exportação, será gerado um arquivo em formato JSON, contendo o valor obtido do método `getJSON`, da classe `PiecesUtil`, passando como parâmetro o elemento raiz da árvore de renderização. Caso decida-se importar, aparecerá uma janela (criada através da função `dialog` do JQuery) onde um arquivo ou texto em formato JSON serão lidos e a cena será carregada passando esses dados para o método `readJSON` da classe `PiecesUtils`.

A aba de `Ajuda` apresenta uma janela com informações de como utilizar o programa. A janela é montada através do JQuery, com o método `dialog`.

O visualizador também conta com um manuseador de API customizado. Ao invés de usar o `ThreeJSHandler` disponibilizado pelo motor, o mesmo foi estendido pelo `ThreeJSCustomHandler`. Essa estratégia foi adotada para permitir que alguns comportamentos gráficos próprios do visualizador pudessem ser definidos sem correr o risco de afetar outras aplicações que fizessem uso do motor de jogos. Desses comportamentos, o mais complexo é o tratamento da segunda câmera, feito no método `beforeRender`, que pode ser visto no Quadro 9.

Quadro 9 - Método `beforeRender` da classe `ThreeJSCustomHandler`

```
01 ThreeJSCustomHandler.prototype.beforeRender = function() {
02       var left   = 0;
03       var bottom = Game.canvas.heigh;
04       var width  = Game.canvas.width;
05       var height = Math.floor(Game.canvas.height/2);
06
07       this.renderer.setClearColor(0);
08       if (this.auxCamera) {
09             VisEdu.defLight.threeObject.visible = false;
10             VisEdu.hideURS();
11             VisEdu.hideHelpers();
12             this.renderer.setViewport( left, bottom, width, height );
13             this.renderer.setScissor( left, bottom, width, height );
14             this.renderer.enableScissorTest ( true );
15             this.renderer.setClearColor(VisEdu.backgroundColor);
16             this.renderer.render(Game.scene.threeObject,
17 this.auxCamera.threeObject);
18             VisEdu.defLight.threeObject.visible = true;
19             VisEdu.showHelpers();
20       }
21       VisEdu.checkURS();
22       height = bottom = Math.round(Game.canvas.height/2);
23       this.renderer.setViewport( left, bottom, width, height );
24       this.renderer.setScissor( left, bottom, width, height );
25       this.renderer.enableScissorTest ( true );
26       this.renderer.setClearColor(VisEdu.clearColor);
27       Game.camera.threeObject.aspect = width / height;
28       Game.camera.threeObject.updateProjectionMatrix();
29
   }
```

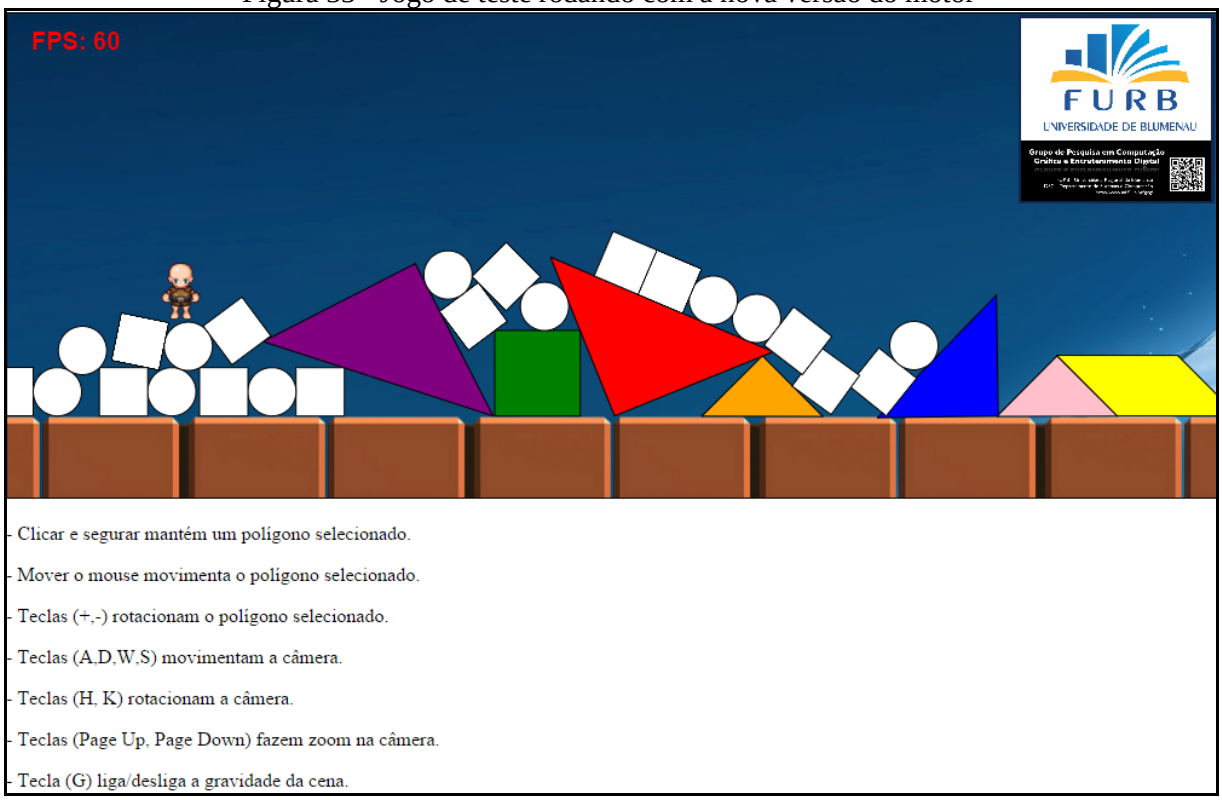
3.3.4 Operacionalidade da implementação

As seções a seguir descrevem os detalhes e funcionalidades de operacionalidade do programa VisEdu-CG, com foco para as diferenças desse trabalho em relação à versão de Nunes (2014).

3.3.4.1 Motor de jogos

Com o desenvolvimento deste trabalho, o motor de jogos passou a ter a capacidade de trabalhar com objetos e cenas tridimensionais através da biblioteca Three.js. Ainda assim, a possibilidade de desenhar com a API do `canvas` do HTML5 continua sendo relevante. Para garantir que a capacidade 2D não foi comprometida, uma aplicação de testes desenvolvida por Harbs (2013) foi executada a partir da nova versão do motor. Essa aplicação específica foi escolhida pois trabalha com diversos conceitos importantes do motor, como a física 2D, tratar a entrada do usuário por teclado (movimento do personagem) e mouse (clique para adicionar elementos aleatórios na tela e arrastar as peças), geração estática e dinâmica de objetos e *assets*. O resultado do teste pode ser visto na Figura 33.

Figura 33 - Jogo de teste rodando com a nova versão do motor



O código do programa de teste teve que sofrer algumas adaptações. A primeira foi a chamada do método `loadAPI` ao fazer a carga do programa, pois este vai determinar a API a ser usada. A segunda alteração foi a utilização da `ObjectFactory` para criação dos objetos gráficos. Essa alteração foi necessária pois a forma de criação dos objetos foi alterada devido aos atributos específicos do contexto tridimensional. Os trechos de código são exibidos no Quadro 10.

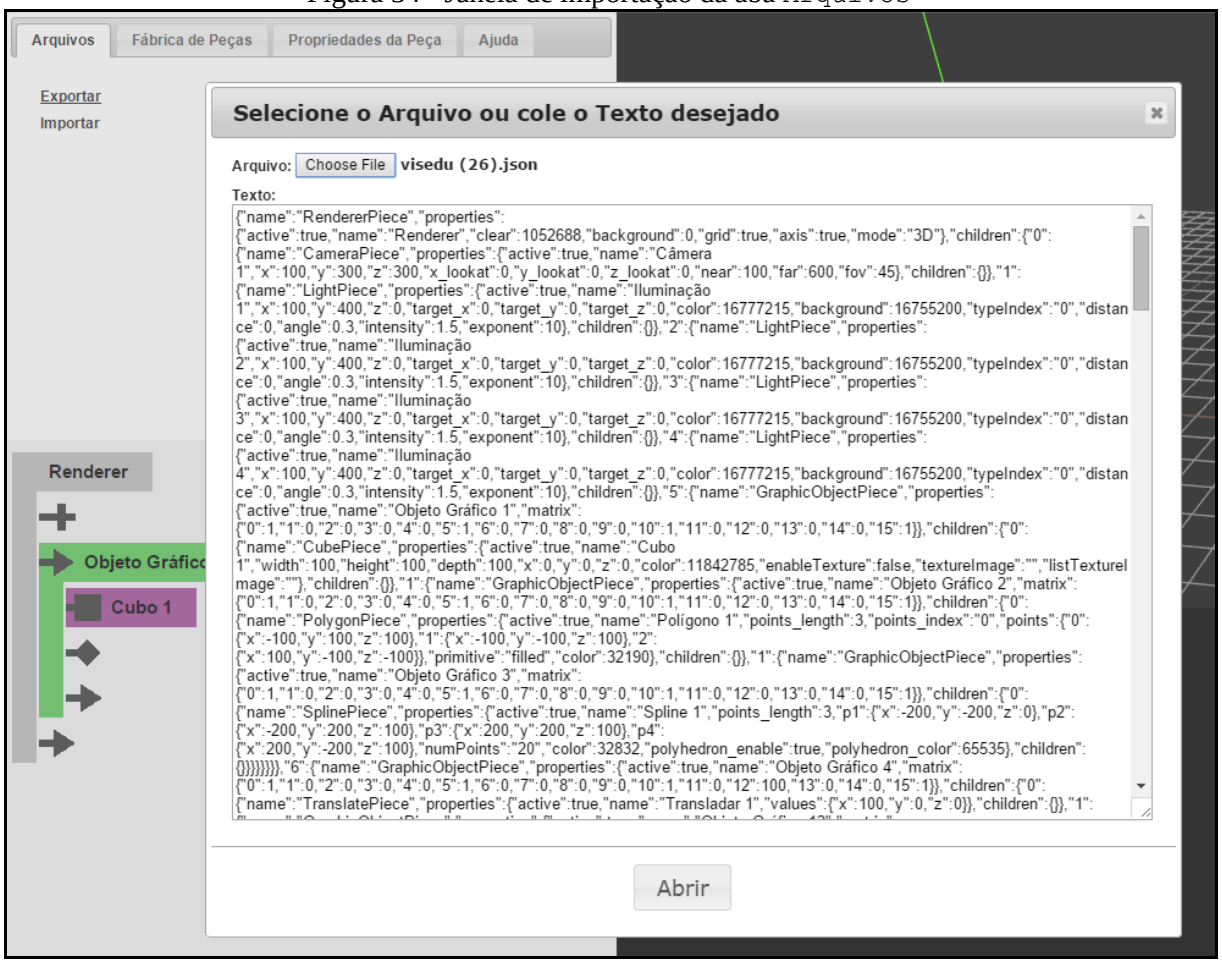
Quadro 10 - Trecho da criação do jogo com a nova versão do motor

```
01 function buildGame(){
02     Game.loadAPI();
03     …
04     var p1 = Game.apiHandler.getObjectFactory().getPolygonObject(new
05 Point2D().initialize(430, 200), [new Point2D().initialize(-50, 100),
06 new Point2D().initialize(-50, -100), new Point2D().initialize(50, 0)],
07 null, "purple", "black");
08
09     var p2 = Game.apiHandler.getObjectFactory().getPolygonObject(new
10 Point2D().initialize(500, 200), [new Point2D().initialize(-100, -50),
11 new Point2D().initialize(100, -50), new Point2D().initialize(0, 50)],
12 null,"red", "black");
13
14     var p3 = Game.apiHandler.getObjectFactory().getPolygonObject(new
15 Point2D().initialize(600, 200), [new Point2D().initialize(-25, 0), new
16 Point2D().initialize(25, -50), new Point2D().initialize(25, 50)], null,
17 "pink", "black");
18 …
```

3.3.4.2 Painel de `Arquivos`

O painel de `Arquivos` manteve a interface de uso presente nas versões anteriores do visualizador de material educacional. Ao clicar em `exportar`, o navegador baixará um arquivo JSON contendo o estado das peças presentes no jogo. Ao clicar em `importar`, uma janela é exibida (Figura 34). Nesta, o usuário pode selecionar um arquivo exportado, carregando assim seu conteúdo no editor de texto da janela, ou escrever manualmente esse conteúdo (seguindo a estrutura dos arquivos JSON). Quando o conteúdo do editor for submetido, o mesmo será convertido novamente em peças e adicioná-las à árvore de peças.

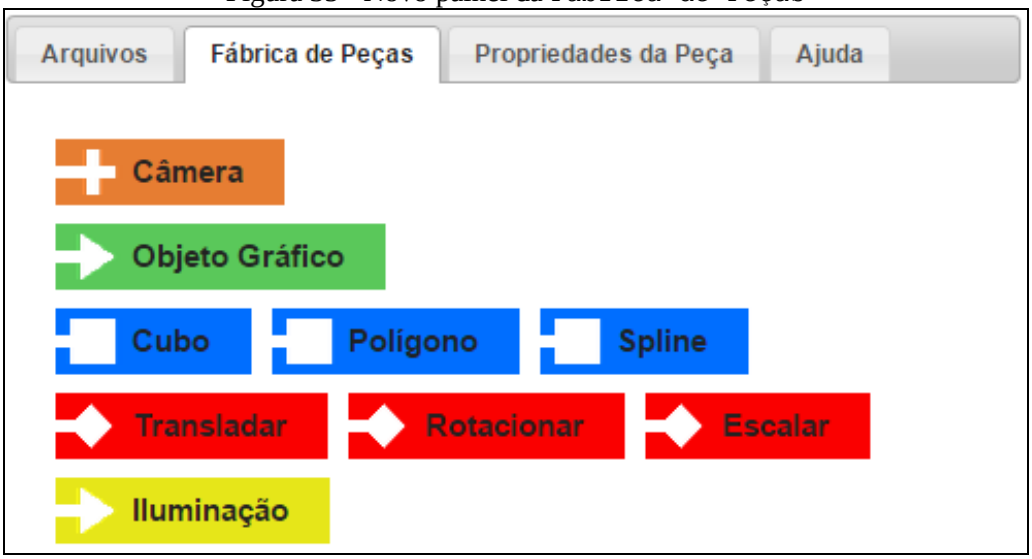
Figura 34 - Janela de importação da aba `Arquivos`



3.3.4.3 Painel `Fábrica de Peças`

As peças apresentadas nesse painel já estavam persentes na versão 3.0 do VisEdu-CG. A lixeira que permite remover os componentes da árvore foi realocada desta aba para o painel da árvore de peças. A Figura 35 ilustra o painel em questão.

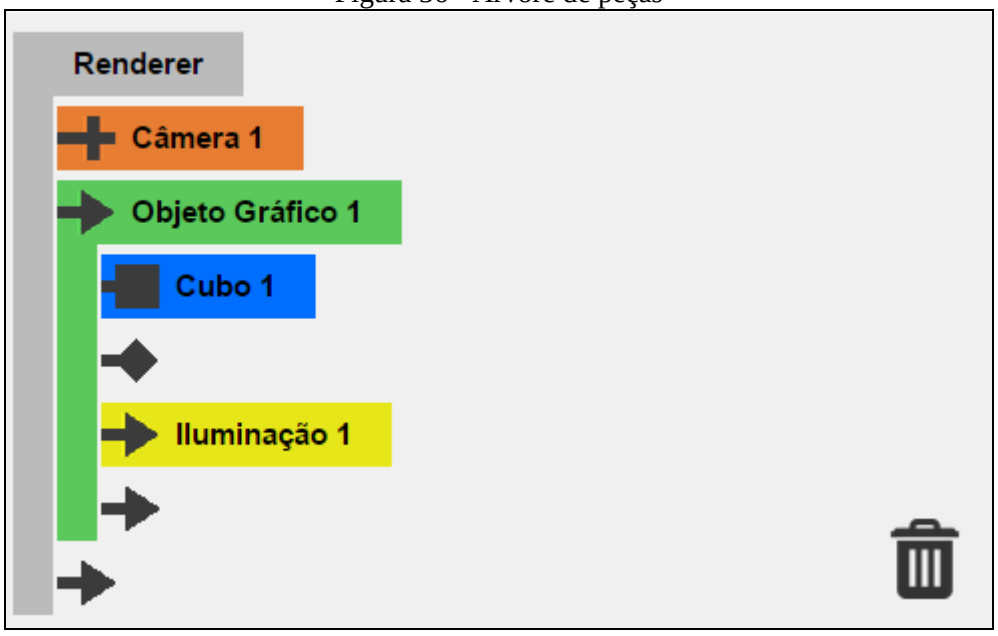
Figura 35 - Novo painel da `Fábrica de Peças`



3.3.4.4 Painel da árvore de componentes

A árvore de componentes tem como raiz a peça `Renderer`. A árvore disponibiliza um conector com encaixe de cruz e um de flecha, que servirão como base para a cena montada pelo usuário através das peças da fábrica. O painel também passou a comportar a lixeira usada para excluir as peças. Por meio da peça `Renderer` é possível ter acesso à algumas propriedades da cena, como a visibilidade da grade e do sistema de referência universal, a cor de fundo da cena e da segunda câmera e o modo de renderização (2D ou 3D). A Figura 36 contém a figura da arvore com algumas peças adicionadas e com a lixeira à mostra.

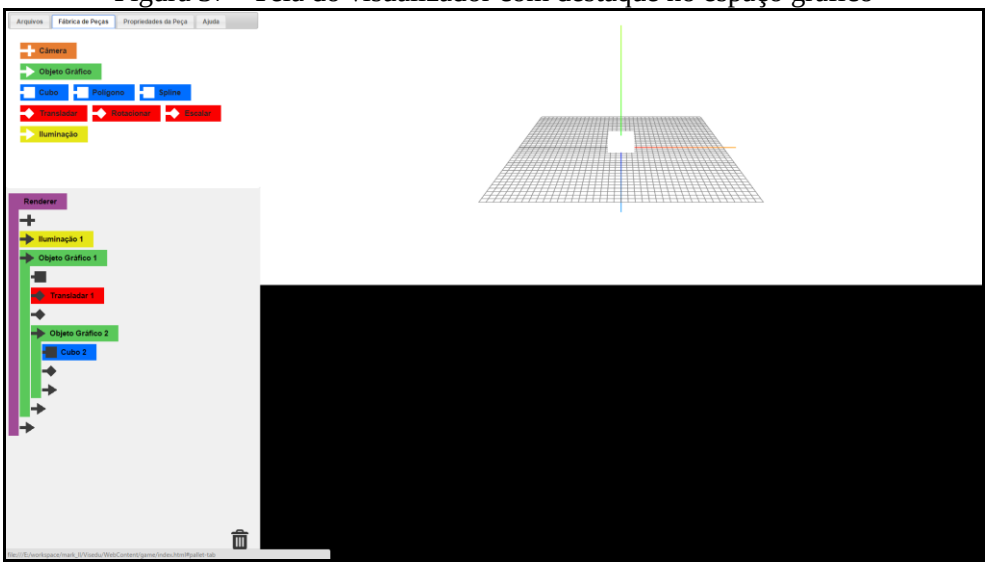
Figura 36 - Árvore de peças



3.3.4.5 Painel do Espaço Gráfico

Com a remoção dos painéis de `Comandos em JOGL` e `Lista de Peças` outros painéis passaram a ocupar mais espaço na tela. Assim, o espaço gráfico duplicou de tamanho e passou a ocupar o espaço superior da tela. Outra alteração é que a cor preta foi escolhida para ser o padrão do fundo deste espaço. A Figura 37 ilustra a tela do programa com o espaço gráfico com fundo branco para destacar sua visualização.

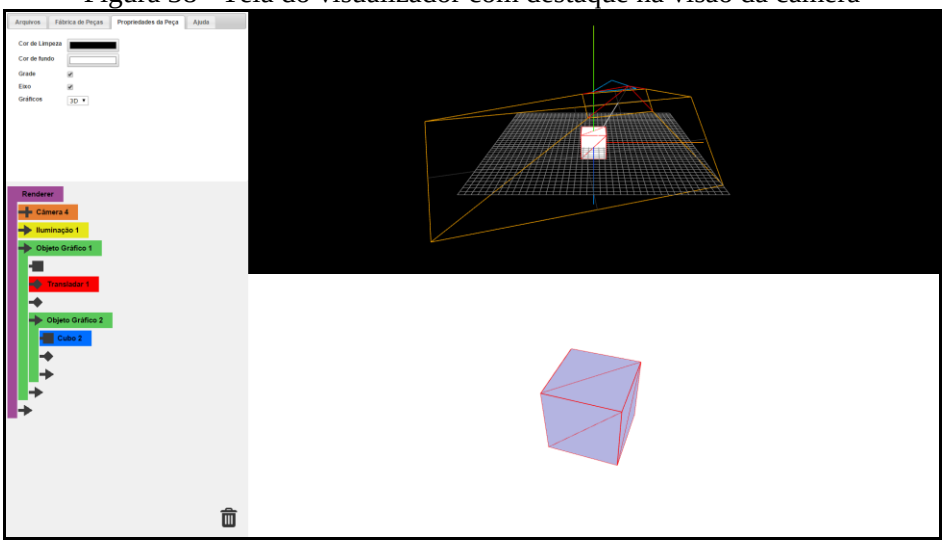
Figura 37 - Tela do visualizador com destaque no espaço gráfico



3.3.4.6 Painel da visão da câmera

O painel da segunda câmera, assim como o do espaço gráfico, passou a ocupar o dobro em relação a verão anterior do sistema. A Figura 38 ilustra este painel, com o fundo em branco para destacá-lo do espaço gráfico.

Figura 38 - Tela do visualizador com destaque na visão da câmera



3.3.4.7 Painel de `Propriedades da Peça`

O painel de Propriedades da Peça, apesar da refatoração, passou por poucas mudanças visuais ou de usabilidade em relação ao trabalho de Nunes (2014). Os campos numéricos por responsáveis pelas coordenadas e tamanho da peça, além dos valores das peças de transformação, agora são apresentados através de um `spinner`, permitindo assim que o usuário use as setas do teclado para aumentar ou diminuir o valor da propriedade (Figura 39). Os campos de cor também mudaram, sendo exibidos através de um `input` do tipo `color` (Figura 40). Esse campo se comportou como esperado nos navegadores Mozilla Firefox e Google Chrome, mas não abriu a janela de seleção de cor no Internet Explorer, obrigando então o usuário a inserir os dados manualmente em hexadecimal.

Figura 39 - Campo de `spinner` das propriedades no Google Chrome

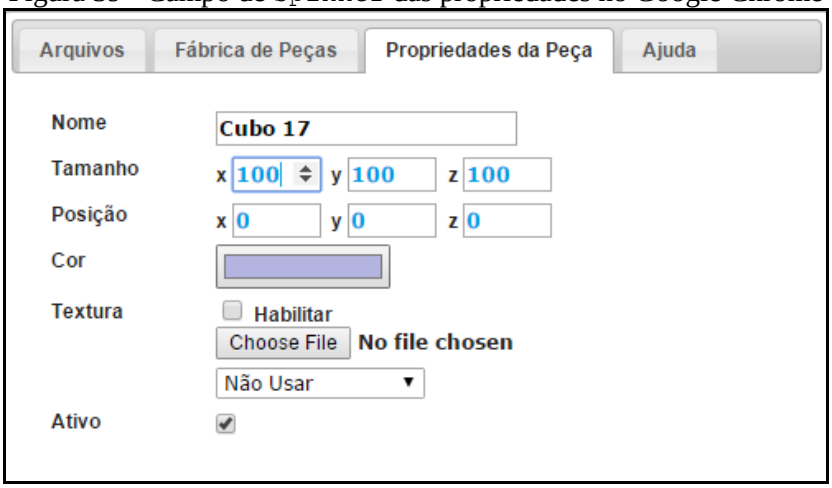
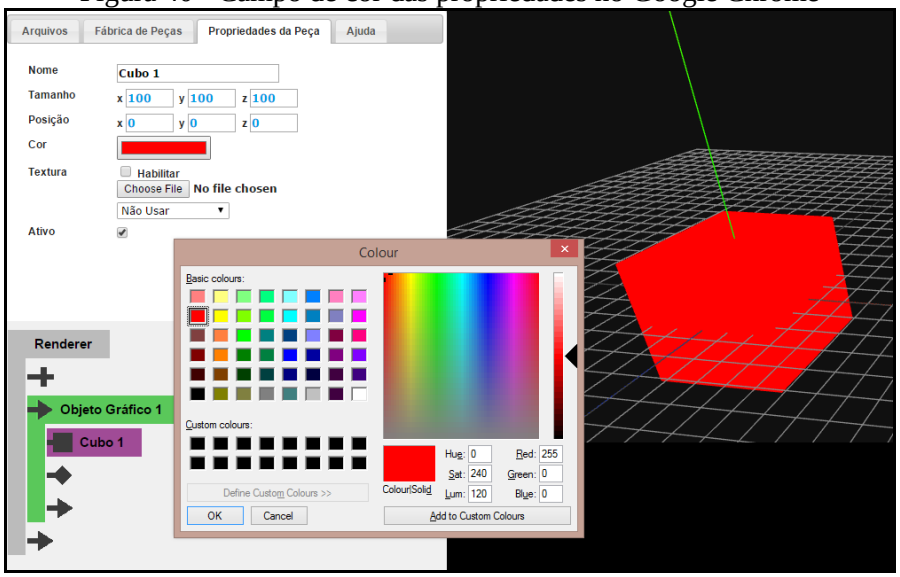


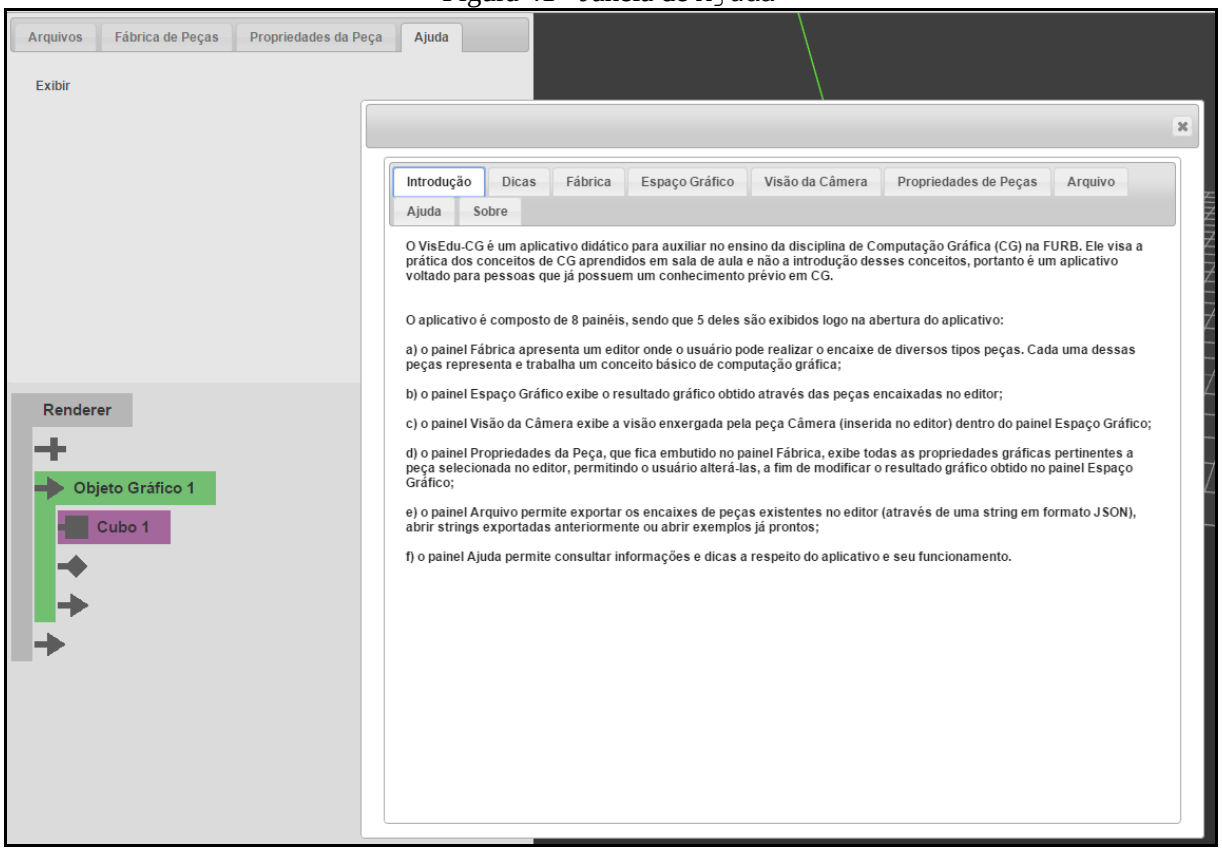
Figura 40 - Campo de cor das propriedades no Google Chrome



3.3.4.8 Painel de `Ajuda`

O painel de A`juda` é responsável por abrir uma janela com instruções de usabilidade do sistema. Ao abrir a aba, o usuário se depara com a opção `Exibir`. Clicando nesse item, a janela é apresentada e o usuário pode navegar entre seus itens. Esta janela é ilustrada na Figura 41.

Figura 41 - Janela de `Ajuda`



3.4 RESULTADOS E DISCUSSÕES

Esse trabalho teve como objetivo disponibilizar um novo modo de renderização para o motor de e jogos HTML desenvolvido por Harbs (2013b) e fazer a integração deste com o VisEdu-CG 3.0, desenvolvido por Nunes (2014b).

O trabalho começou com a adaptação do motor de jogos para permitir a utilização do Three.js para renderização de suas peças, enquanto mantinha o suporte ao HTML5. A maior dificuldade encontrada neste momento foi a diferença em como cada uma das tecnologias gerenciam os elementos em cena; o HTML5 exige que eles sejam redesenhados a cada *loop* do jogo, enquanto o WebGL, abstraído pelo Three.js, mantém esses objetos em memória, exigindo que o usuário apenas aplique as modificações desejadas.

Todos os componentes visuais presentes no motor de jogos já possuíam algum equivalente na biblioteca de objetos do Three.js. Dessa forma, deixar os componentes compatíveis com ambas tecnologias gráficas foi uma tarefa simples.

A decisão de refazer completamente o visualizador de material educacional aumentou significativamente o volume de programação para concluir este trabalho, mas também permitiu uma fácil integração com o motor de jogos e neutralizou o risco de incompatibilidade entre os programas. Além disso, permitiu o uso de tecnologias mais difundidas (HTML, CSS, Javascript e JQuery) fossem utilizadas para criação e manutenção das abas, janelas e árvore de peças do sistema.

A remoção do painel de `Comandos em JOGL` e `Lista de Peças` permitiu que a visualização do espaço gráfico, foco da aplicação, fosse melhorado.

Uma alteração que não estava prevista foi o novo modo de controle de câmera. O visualizador foi então desenvolvido para usar um controle orbital de câmera.

Com o objetivo de levantar com precisão os resultados da aplicação foram realizados testes de desempenho e operacionalidade. Os resultados obtidos nestes podem ser acompanhados nas seções seguintes.

3.4.1 Teste de desempenho

O processo de avaliação de desempenho do programa foi realizado adicionando elementos gráficos na cena do jogo. Os testes foram realizados nos navegadores: Google Chrome, Mozilla Firefox e Internet Explorer.

Os critérios analisados para medir a performance do visualizador foram o consumo de memória, monitorado através do gerenciador de tarefas do Windows, e a taxa de quadros por segundo, acompanhado através do monitor gerado pela biblioteca `Stats`.

Os cenários de testes são semelhantes aos usados no desenvolvimento do VisEdu 3.0 por Nunes (2014b). Foram criados seis cenários, cada um com determinada quantidade de peças dos tipos `Renderer`, `Câmera`, `Iluminação`, `Objeto Gráfico`, `Transladar`, `Cubo`, `Polígono e Spline`. A Tabela 1 descreve a configuração das peças em cada cenário.

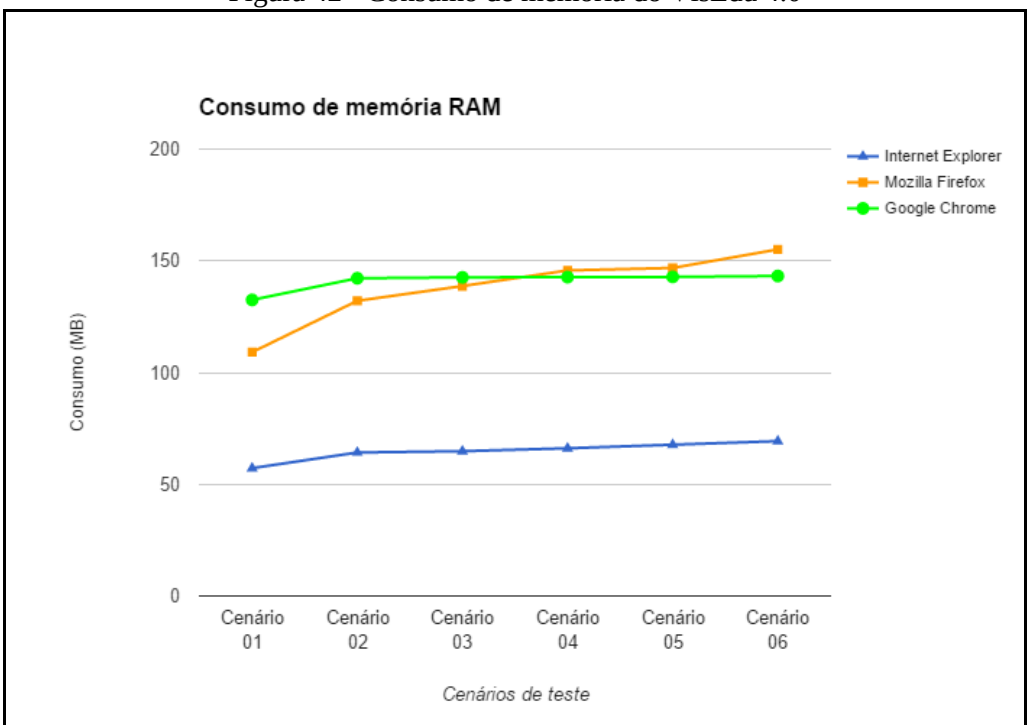
Tabela 1 - Cenários de teste

Peça	Cenário 01	Cenário 02	Cenário 03	Cenário 04	Cenário 05	Cenário 06
Renderer	1	1	1	1	1	1
Câmera	1	1	1	1	1	1
Iluminação	1	1	1	1	1	2
Objeto Gráfico	3	6	9	11	12	15
Transladar	0	6	9	11	12	15
Cubo	1	2	3	3	4	5
Polígono	1	2	3	4	4	5
Spline	1	2	3	4	4	5
	9	21	30	36	39	49

Para cada cenário montado no visualizador foi exportado um arquivo em formato JSON através da aba `Arquivos`. Para cada navegador foram importados estes arquivos para monitorar o consumo de memória e taxa de quadros por segundo.

Com relação ao consumo de memória, foi observado que o Internet Explorer foi o navegador que obteve o menor consumo de memória em todos os cenários. Até o terceiro cenário, o Firefox consumiu menos que o Chrome, mas o navegador da Google mostrou um consumo mais constante entre os testes, e a partir do quarto cenário o Firefox passou a ser o que mais consumia memória. A Figura 42 mostra o consumo dos navegadores de acordo com o cenário.

Figura 42 - Consumo de memória do VisEdu 4.0



	Cenário 01	Cenário 02	Cenário 03	Cenário 04	Cenário 05	Cenário 06
Internet Explorer	57.2	64.2	64.8	66.1	67.7	69.3
Mozilla Firefox	109.1	132.1	138.7	145.7	146.8	155.1
Google Chrome	132.5	142.2	142.6	142.7	142.8	143.2

Com relação aos quadros por segundo, os testes realizados resultaram na mesma qualidade de processamento em todos os cenários e para todos os navegadores: 60 quadros por segundo constantes. Esse resultado diverge bastante o que foi reportado no trabalho de Nunes (2014b). Para fins de comparação, os cenários foram reproduzidos novamente na versão 3.0 do VisEdu-CG, e este também alcançou 60 quadros por segundo. Esse resultado pode ser atribuído a diferenças na capacidade de processamento das máquinas, otimizações feitas nas últimas versões do Three.js ou em melhorias na forma como os navegadores implementam o WebGL.

3.4.2 Teste de Operacionalidade

Para fazer a avaliação da operacionalidade do visualizador, foi realizado um exercício com a turma de computação gráfica da FURB em conjunto com o professor da disciplina. A atividade consistia numa lista de 5 questões que requeriam que o estudante montasse cenas utilizando as peças do programa. A lista de questões está disponível no Apêndice B. Uma vez que os alunos estão no final da disciplina, assume-se que estes conhecem os conceitos de computação gráfica abordados pelo VisEdu-CG 4.0. Ao finalizar a atividade, os alunos foram convidados a responder um questionário com perguntas relacionados a experiência de uso da aplicação. As perguntas presentes no questionário podem ser vistas no Quadro 11.

Quadro 11 - Questionário de operacionalidade

Questão	Opção			
Qual o seu nível de conhecimento sobre conceitos de "Computação Gráfica"?	Nenhum	Ruim	Bom	Ótimo
O que você achou do "layout" do aplicativo?	Inadequado	Ultrapassado	Normal	Moderno
O que você achou da "usabilidade" do aplicativo?	Difícil, nada intuitivo	Mediano, mas pouco intuitivo	Mediano, mas intuitivo	Fácil e intuitivo
O que você achou do tamanho e disponibilidade das "janelas"?	Péssimo	Ruim	Bom	Ótimo
O comportamento de arrastar e encaixar as peças funcionou apropriadamente?	Sim	Houve poucos problemas	Houve vários problemas	Não funcionou
A disponibilidade e usabilidade das propriedades das peças estão adequadas?	Sim	Parcialmente		Não
O que você achou do controle da câmera	Bom	Regular		Ruim
O aplicativo ajudou a entender a aplicação de como usar os objetos gráficos "Cubo", "Polígono" e "Spline"?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo ajudou a entender o conceito de "Câmera Sintética" e suas propriedades (posição, look at, near, far e fov)?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo ajudou a entender o conceito de "Grafo de Cena" (uso do glPushMatrix/glPopMatrix)?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo ajudou a entender a aplicação de "texturas" nos objetos gráficos?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo ajudou a entender a aplicação de "iluminação" nos objetos gráficos?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
O aplicativo possibilitou entender em 2D os mesmos conceitos gráficos apresentados no espaço 3D?	Não ajudou	Ajudou pouco	Ajudou	Ajudou muito
Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo?	Descritiva			
Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo?	Descritiva			
Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?	Descritiva			

Ao todo, doze alunos responderam os questionamentos. Destes 58,3% dos alunos responderam que consideram seu nível de conhecimento em computação gráfica bom, enquanto os outros 41,7% consideraram ruim. Metade dos alunos classificaram o layout do aplicativo como moderno, a outra metade classificou como normal. Já 75% dos alunos consideraram a usabilidade como mediana, mas intuitiva, e 25% como fácil e intuitiva. As respostas às três primeiras perguntas podem ser vistas na Figura 43, Figura 44 e Figura 45.

Figura 43 - Gráfico das respostas da pergunta 1

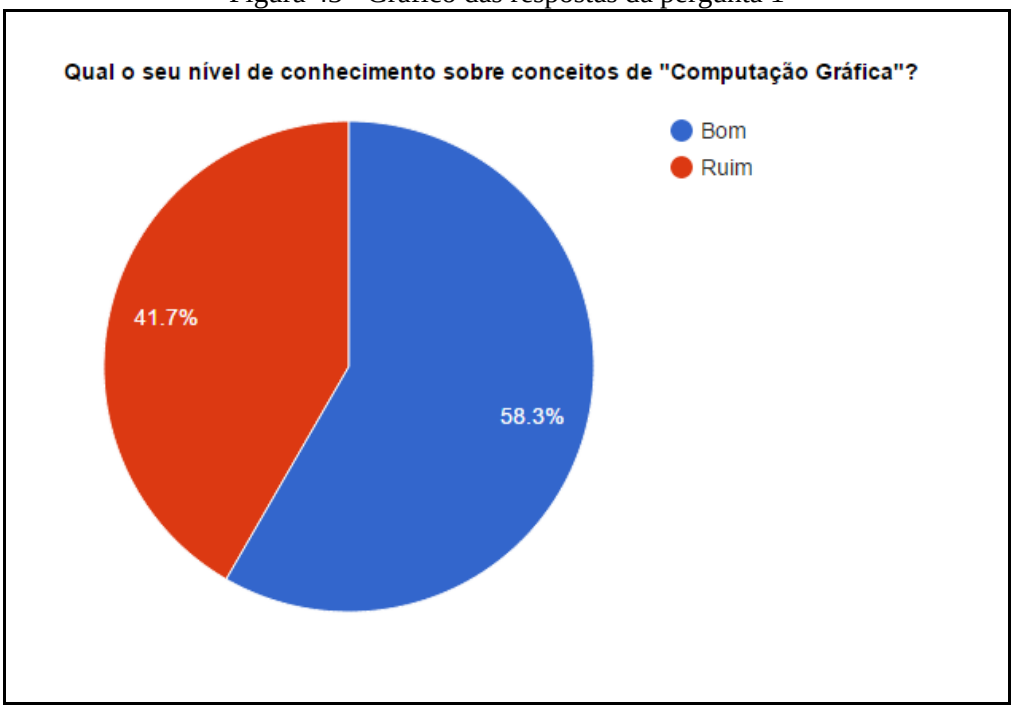


Figura 44 - Gráfico das respostas da pergunta 2

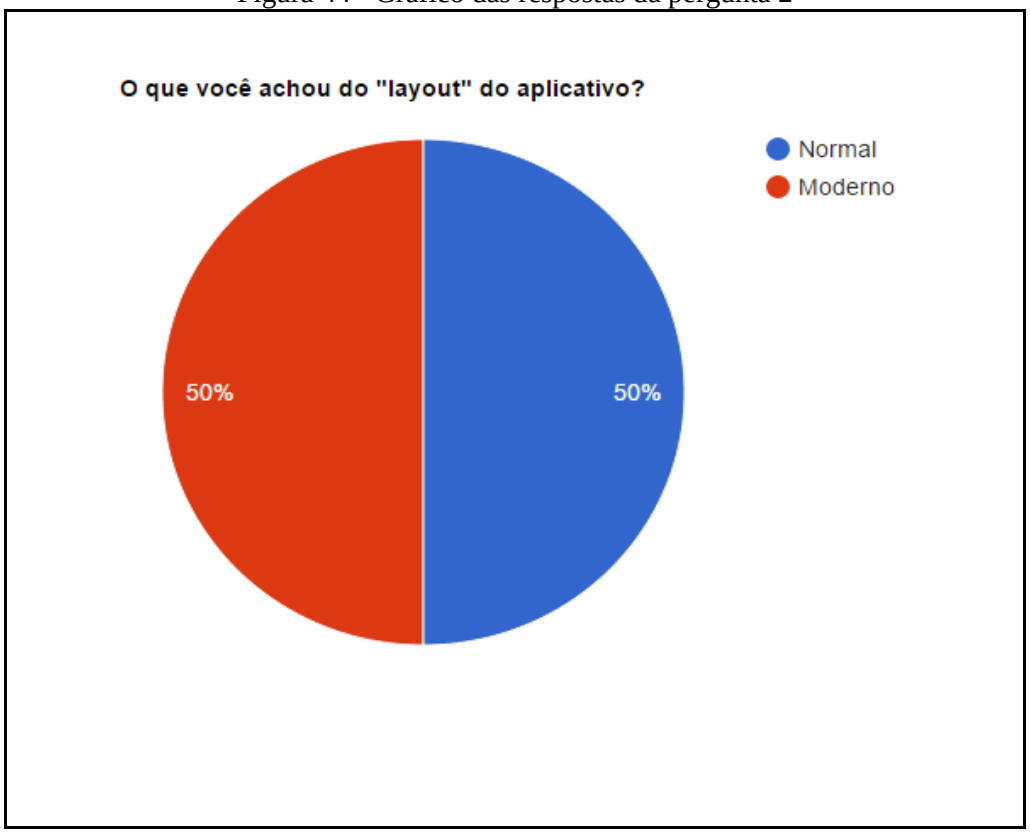
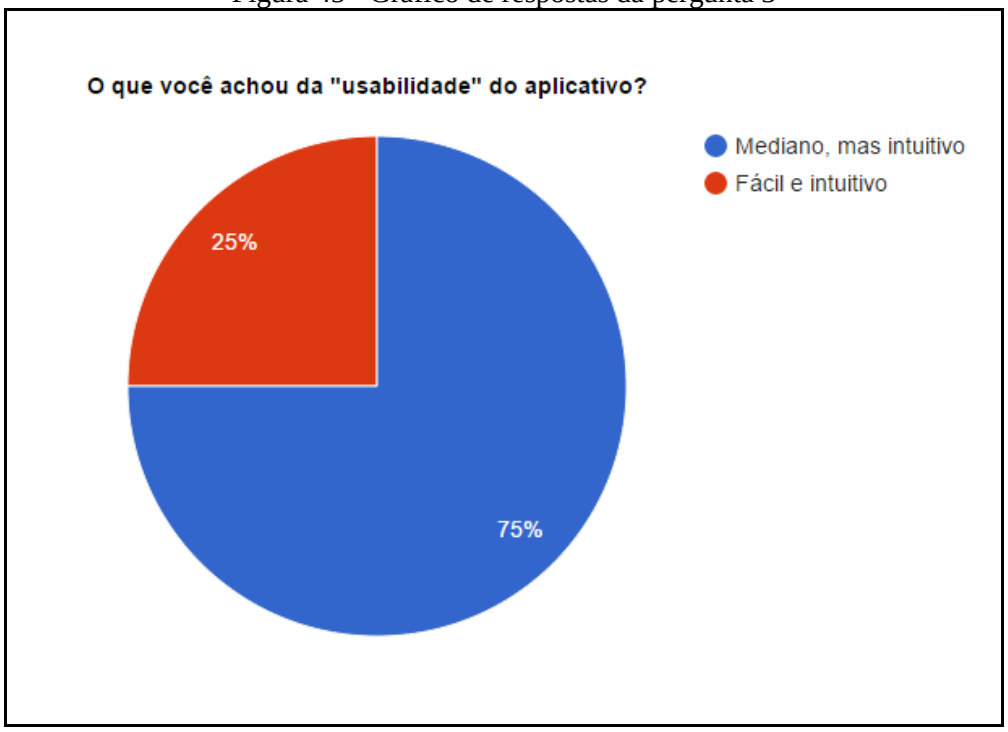


Figura 45 - Gráfico de respostas da pergunta 3



Ainda, 66,7% consideraram o tamanho e disponibilidade das janelas bom, 25% consideraram ótima e 8.3% achou ruim (Figura 46). Também 83,3% relataram que o comportamento de arrastar e encaixar as peças funcionou apropriadamente, 16,7% afirmaram que encontraram poucos problemas (Figura 47). Ainda, 58,3% consideraram a usabilidade das propriedades adequada contra 41.5% que achou apenas parcialmente adequada (Figura 48).

Figura 46 - Gráfico de respostas da pergunta 4

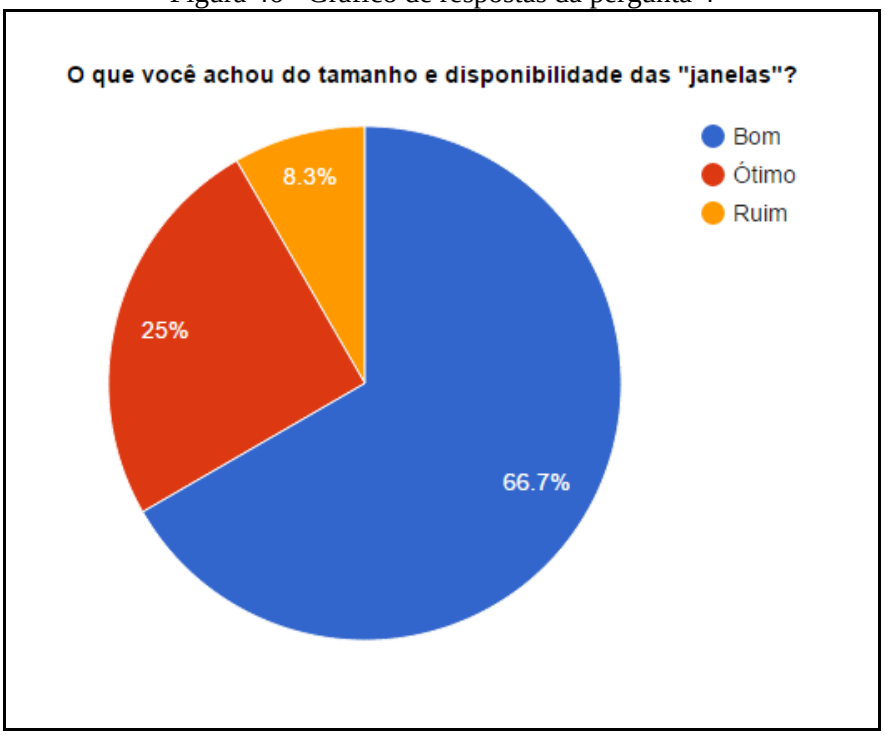


Figura 47 - Gráfico de respostas da pergunta 5

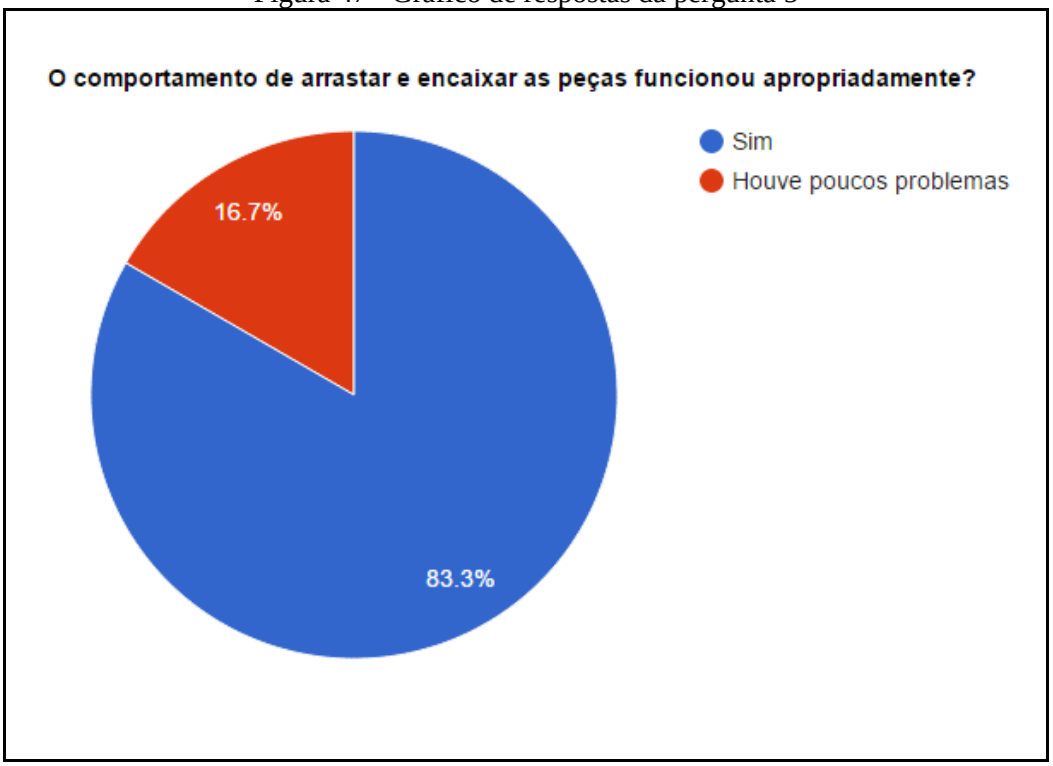
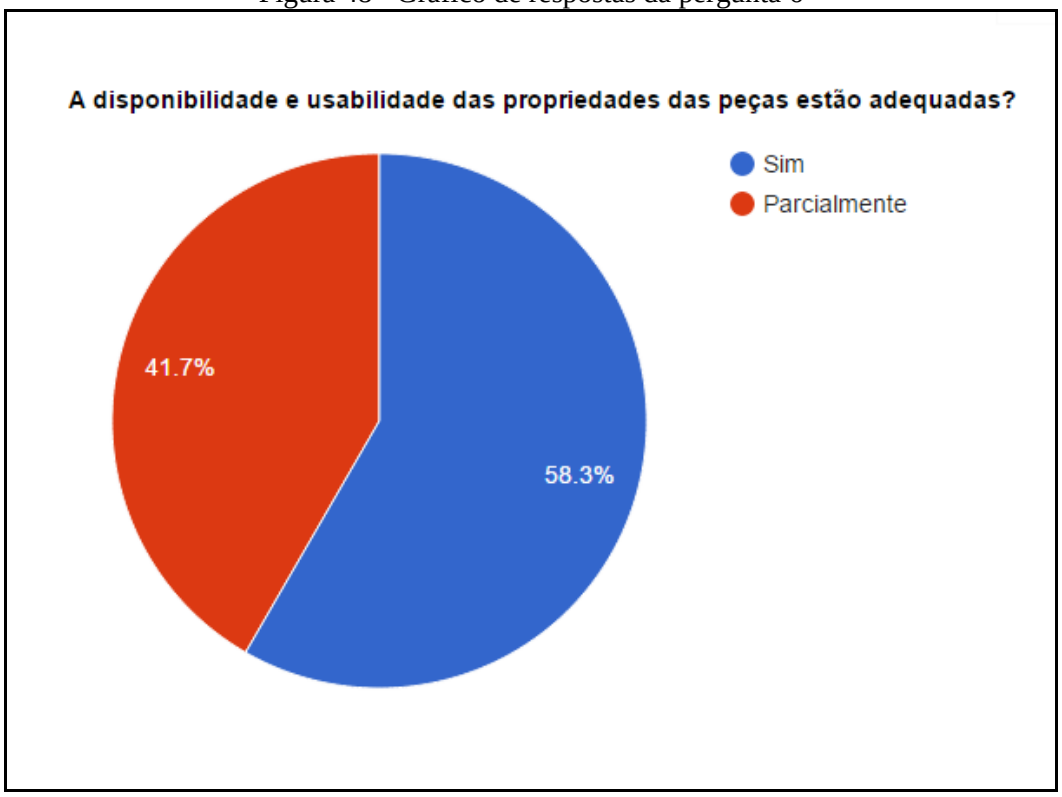


Figura 48 - Gráfico de respostas da pergunta 6



A avaliação também apontou que 66,7% dos avaliadores acharam o controle da câmera bom, enquanto 16,7% considerou mediano e outros 16,7% acharam ruim (Figura 49). Ainda, 75% afirmaram que a aplicação ajudou a entender os conceitos de cubo, polígono e *spline*, enquanto 25% disseram ajudar muito (Figura 50). Também, 50% afirmaram que o programa

ajudou a entender o conceito de câmera sintética e suas propriedades, 41,7% consideraram que ajudou muito e 8,3% diz não ter ajudado, conforme a Figura 51.

Figura 49 - Gráfico de respostas da pergunta 7

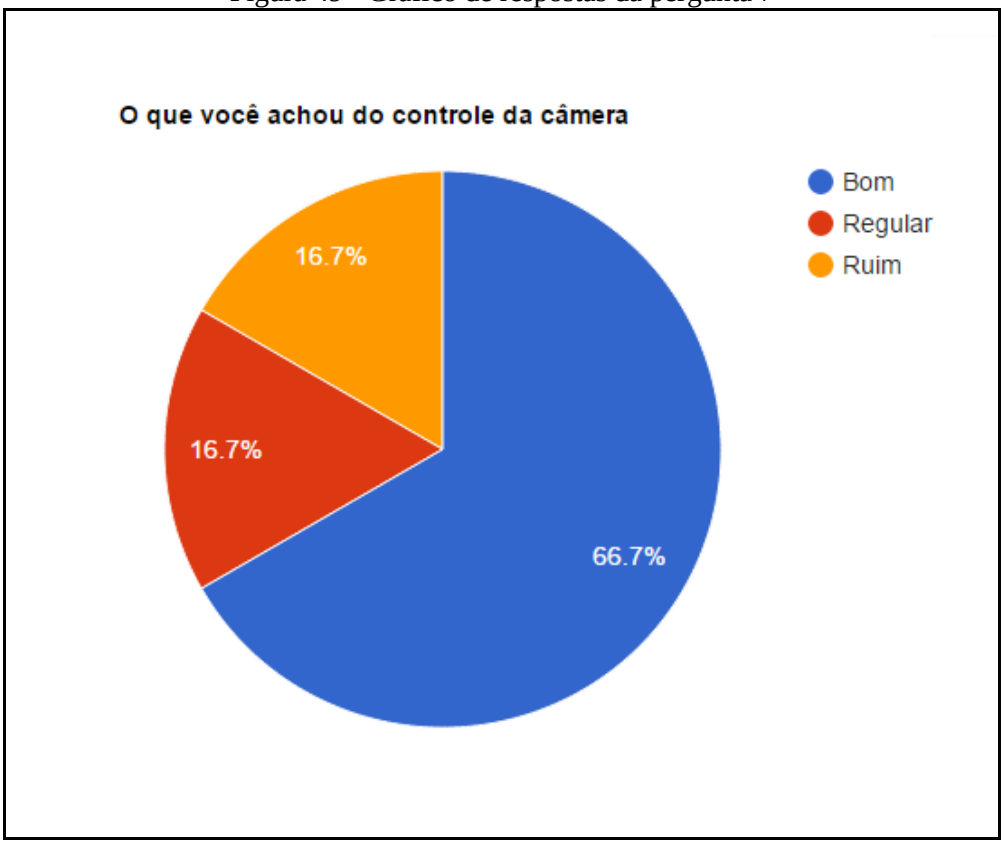
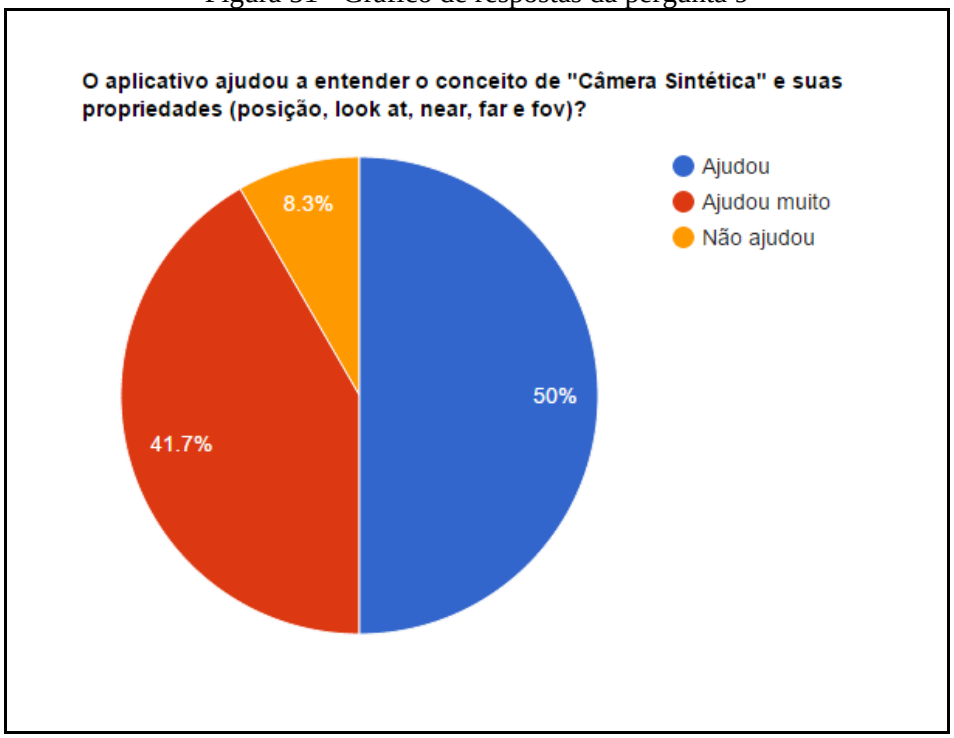


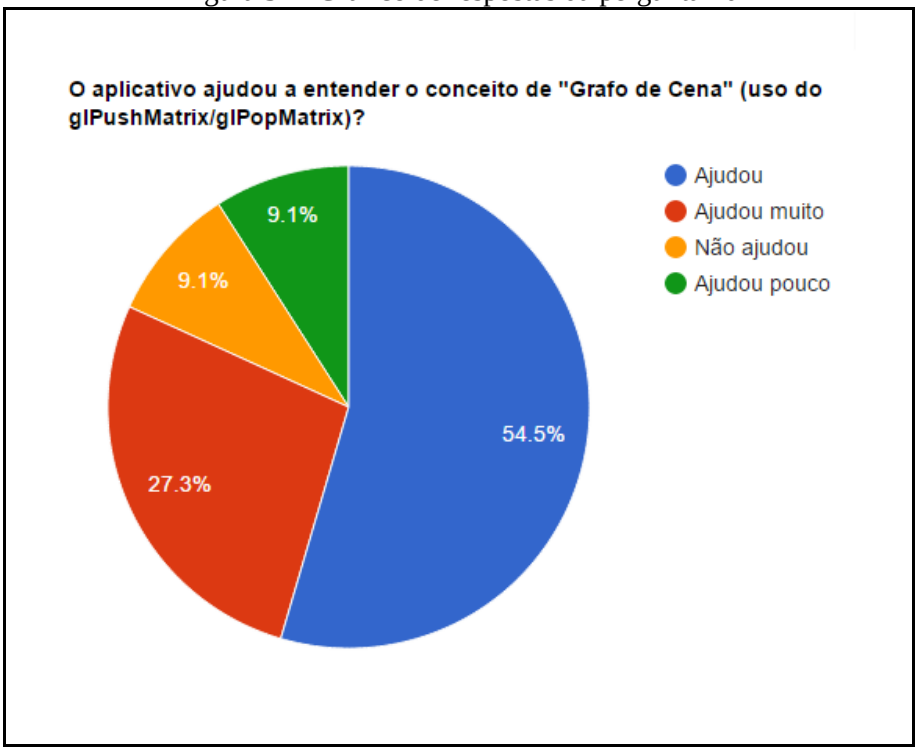
Figura 50 - Gráfico de respostas da pergunta 8

Figura 51 - Gráfico de respostas da pergunta 9



Entre os alunos, 54,5% afirmaram que a aplicação ajudou a entender os conceitos de Grafo de cena, 27,3% disseram ter ajudado muito, 9,1% disseram ter ajudado pouco e 9,1% diz não ter ajudado (Figura 52).

Figura 52 - Gráfico de respostas da pergunta 10



Dos alunos, 41,7% alegaram que o programa ajudou a entender o conceito de transformações geométricas, outros 41,7% disseram ter ajudado muito, 8,3% alegaram ter

ajudado pouco e 8,3% afirmaram não ter ajudado (Figura 53). Ainda, 58,3% disseram que o visualizador ajudou a entender o uso de texturas nos objetos, com 33,3% dizendo que ajudou muito e 8,3% alegando que ajudou pouco (Figura 54).

Figura 53 - Gráfico de respostas da pergunta 11

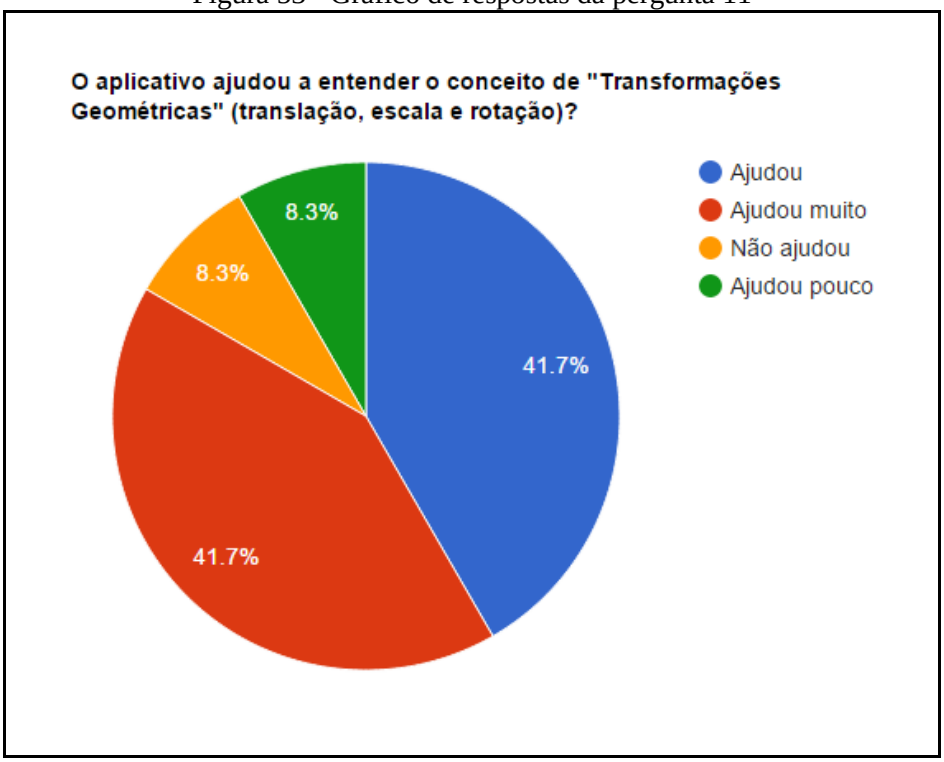


Figura 54 - Gráfico de respostas da pergunta 12

Dentre os avaliados, 50% afirmaram que a ferramenta ajudou a entender a aplicação de iluminação nos objetos gráficos, enquanto 41.7% afirmaram ter ajudado muito e 8.3% consideraram que ajudou pouco (Figura 55). Enfim, 58% afirmaram que o aplicativo ajudou a

entender em 2D os mesmos conceitos gráficos apresentados no espaço 3D, enquanto 33.3% alegaram que ajudou muito e 8.3% disseram que ajudou pouco (Figura 56).

Figura 55 - Gráfico de respostas da pergunta 13

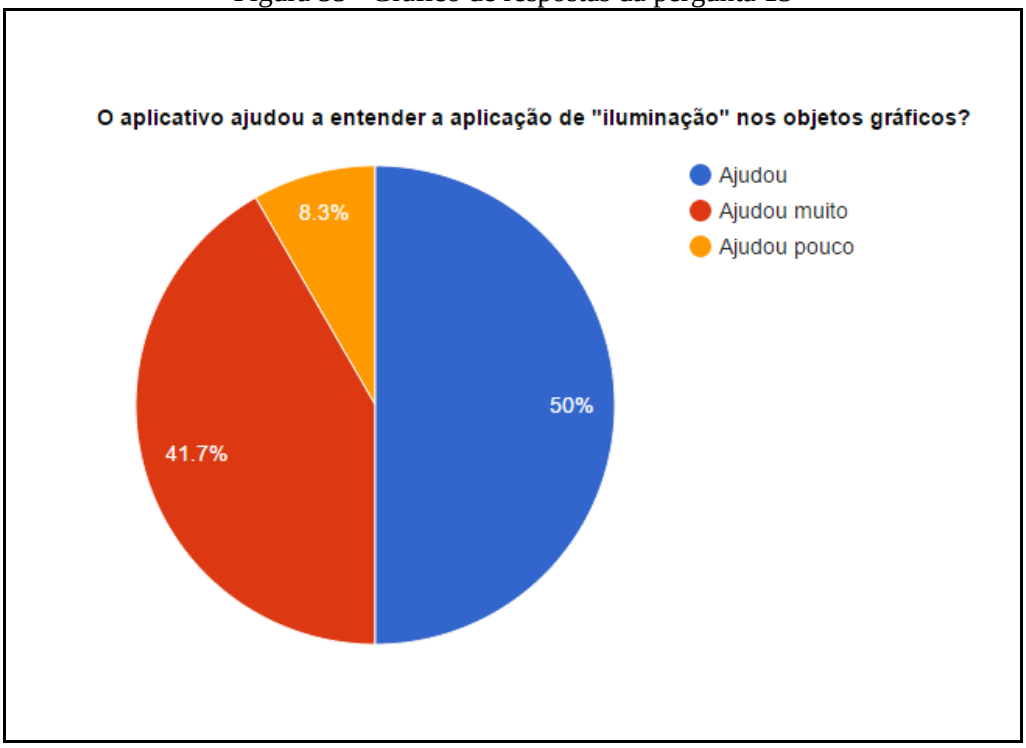
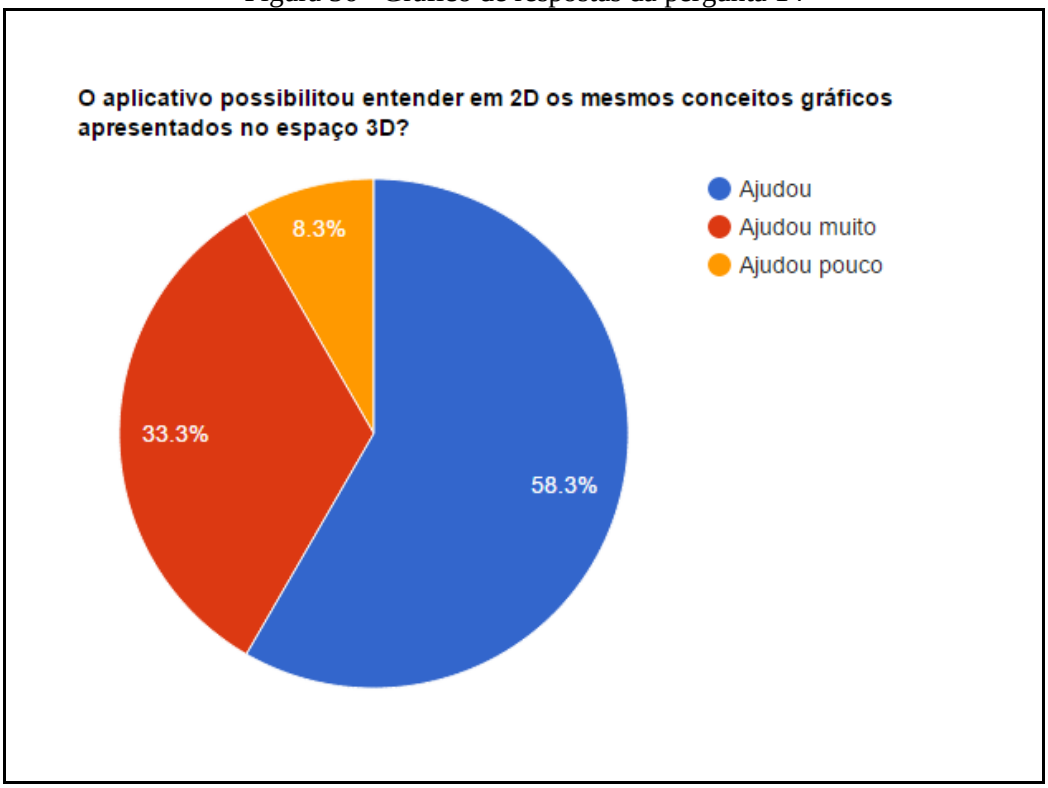


Figura 56 - Gráfico de respostas da pergunta 14



As respostas para as questões descritivas estão presentes no Apêndice C. Na questão referente aos pontos positivos da aplicação, a maior parte dos comentários estão relacionados

com a facilidade de usar a ferramenta. Outro ponto positivo recorrente é de conseguir cumprir seu papel de apresentar visualmente os conceitos aprendidos em sala.

Quanto aos pontos negativos do sistema, a crítica mais recorrente foi na usabilidade de algumas peças, especialmente a que controla a quantidade de pontos do polígono, e compatibilidade com navegadores.

Houve diversas sugestões de melhorias, como mais tipos de peças disponíveis, a volta do painel que mostrava o código gerado para a cena, sugestões de usabilidade, correções de compatibilidade, entre outras.

3.4.3 Comparativo entre o trabalho desenvolvido e os correlatos

Esta seção tem como objeto apresentar a discutir as principais características deste trabalho em comparação com os trabalhos correlatos. O Quadro 12 apresenta uma comparação entre estes.

Quadro 12- Características dos trabalhos correlatos e do projeto desenvolvido

Característica	TinkerCad	Motor de jogos HTML5	StarLogo TNG	VisEdu-CG 3.0	VisEdu-CG 4.0
Utiliza espaço gráfico 3D	Sim	Não	Sim	Sim	Sim
Suporta animações	Não	Sim	Sim	Não	Não
Faz uso de HTML5	Sim	Sim	Não	Sim	Sim
Faz uso de WebGL	Sim	Não	Não	Sim	Sim
Software educacional	Sim	Sim	Sim	Sim	Sim
Comportamentos reutilizáveis entre componentes	Não	Sim	Sim	Não	Sim

Conforme a comparação, é possível perceber que o visualizador desenvolvido não se enquadra em apenas um item: suporte a animações. Ainda que o motor do visualizador tenha essa funcionalidade, não foi implementada a peça que daria acesso desse recurso ao usuário. A implementação da peça de animação foi incluída na lista de sugestões de extensão do trabalho (seção 4.1).

O destaque deste trabalho foi manter uma definição de arquitetura de comportamentos reutilizáveis entre componentes combinada com a utilização do espaço gráfico 3D.

4 CONCLUSÕES

Essa monografia apresentou a integração de um visualizador de material educacional com um motor de jogos HTML5. Neste processo, o motor foi adaptado para suportar a biblioteca Three.js e assim fornecer as capacidades tridimensionais necessárias (conforme objetivo proposto) e o visualizador foi completamente refatorado para explorar outras técnicas, tecnologias e garantir que opere de forma eficiente junto ao motor.

O Three.js se provou uma ferramenta muito útil, capaz de desenhar cenas complexas através de uma API simples e uma ampla biblioteca de componentes e funcionalidades. Mesmo inserindo um grande número de elementos gráficos e testando em navegadores diferentes, a biblioteca se manteve estável e performática, mostrando que há espaço na web para aplicações gráficas cada vez mais exigentes.

Apesar da grande capacidade do Three.js, é importante analisar o custo e benefício do uso da tecnologia. A utilização de HTML, CSS, Javascript e JQuery permitiu um controle mais simples e eficaz sobre peças, abas e janelas.

Os objetivos propostos relacionados ao motor de jogos e o VisEdu-CG foram atingidos. O modo 2D, já presente na versão 3.0 (NUNES, 2014) foi melhorado para que os objetos renderizados nesse modo sejam elementos completamente bidimensionais. Outra melhoria relacionada aos modos de renderização foi permitir que o usuário troque o modo sem recarregar a página e perder o estado das peças. As principais melhorias que devem ser aplicadas ao visualizador incluem a criação de mais tipos peças, a fim de cobrir uma gama cada vez maior de conceitos gráficos, e a correção de alguns problemas que surgiram na utilização das propriedades das peças, principalmente com o polígono.

4.1 EXTENSÕES

Durante as etapas finais do desenvolvimento deste projeto, foram observados alguns itens a serem alterados e algumas sugestões de extensões. Estas são:

d) dinamizar a construção das propriedades das peças;
e) criar de um tipo de peça para animação dos `Objetos Gráficos`;
f) criar de um tipo de peça para manipulação dos *Shaders* dos `Objetos Gráficos`;
g) disponibilizar uma biblioteca de física 3D para permitir tratar a colisão dos objetos tridimensionais do motor;
h) disponibilizar uma forma para que um usuário do visualizador consiga, remotamente, visualizar e manipular a cena gráfica de outro usuário.

REFERÊNCIAS

ARAÚJO, Luciana P. de. **AduboGL**: aplicação didática usando a biblioteca OpenGL. 2012. 78 f. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

ARORA, Sumeet. **WebGL Game Development**. Birmingham: Packt Publishing Ltd, 2014. 418 p.

BOYLE, Tom. Design principles for authoring dynamic, reusable learning objects. **Australian Journal of Educational Technology.** [s.l.], p. 46-58. jan. 2003. Disponível em: <http://ascilite.org.au/ajet/submission/index.php/AJET/article/view/1690/787>. Acesso em: 08 set. 2014.

CANTOR, Diego; JONES, Brandon. **WebGL beginner's guide**. Birmingham: Packt

Publishing Ltd, 2012.

ECLIPSE. **Eclipse Foundation** [S.l.], 2015. Disponível em: <https://eclipse.org/org/foundation/>. Acesso em: 20 jan. 2015.

HARBS, Marcos. **Motor para Jogos 2D Utilizando HTML5**. 2013. 78 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências e Exatas Naturas, Universidade Regional de Blumenau, Blumenau.

HODGINS, H. Wayne. The future of learning objects. In: ECI Conference on e-Technologies in Engineering Education: Learning Outcomes Providing Future Possibilities, 2002, Davos. **Proceedings…** Bloomington, Agency for Instructional Technology, 2004. P. 671–678

LETOPISI. **StarLogo**. [S.l.], 2015. Disponível em: < http://letopisi.org/index.php/StarLogo>. Acesso em: 18 jul. 2015.

MCFARLAND, David S. **Javascript & jQuery**: the missing manual. 2. Ed. [S.l]: O'Reilly Media, Inc, 2011. 538 p.

MONTIBELER, James P. **VISEDU-CG**: visualizador de material educacional, módulo de computação gráfica. 2014a. 103 f. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

NUNES, S. A. **VisEdu-CG 3.0:** Aplicação didática para visualizar material educacional – Módulo de Computação Gráfica. 2014. 89 f. Trabalho de Conclusão de curso (Bacharelado em Ciência da Computação) – Centro de Ciências e Exatas Naturas, Universidade Regional de Blumenau, Blumenau.

PARISI, Tony. **Programming 3D Applications with HTML5 and WebGL:** 3D Animation and Visualization for Web Pages. [S.l]: O'Reilly Media, Inc, 2014. 404 p.

PILGRIN, Mark. **HTML5:** up and running. [S.l]: O'Reilly Media, Inc, 2010. 222 p.

POLSANI, Pithamber R. Use and abuse of reusable learning objects. **Journal of Digital information**, v. 3, n. 4, 2003. Disponível em: <https://journals.tdl.org/jodi/index.php/jodi/article/view/89/88 >. Acesso em: 8 set. 2014.

STEP. **StarLogo TNG**. [S.l.], 2013. Disponível em: <http://education.mit.edu/projects/starlogo-tng>. Acesso em: 8 set. 2014.

THREE.JS. **three.js**. [S.l.], 2015. Disponível em: < http://threejs.org/>. Acesso em: 20 jan. 2015.

TINKERCAD. **Create 3D digital designs with online CAD**. [S.l], 2014. Disponível em: <https://tinkercad.com/about/>. Acesso em: 8 set. 2014.

W3SCHOOLS. **HTML5 introduction**. [S.l.], 2015. Disponível em: <http://www.w3schools.com/html/html5_intro.asp>. Acesso em: 15 jun. 2015.

_____. **JavaScript introduction**. [S.l.], 2015. Disponível em: <http://www.w3schools.com/js/js_intro.asp>. Acesso em: 15 jun. 2015

_____. **jQuery introduction**. [S.l.], 2015. Disponível em: <http://www.w3schools.com/jquery/jquery_intro.asp >. Acesso em: 15 jun. 2015.

APÊNDICE A – Detalhamento dos casos de uso

Esse apêndice apresenta o detalhamento dos casos de isso do motor de jogos (Harbs, 2013) e do visualizador de material educacional.

Quadro 13 - Caso de uso 01

UC01 – Definir modo de renderização	
Descrição	Permite o usuário do sistema definir a API que será usada para renderização do jogo.
Cenário principal	1. O usuário determina a utilização do HTML5 2. O usuário determina a utilização do Three.js 3. O usuário cria sua própria rotina de renderização
Pós-Condição	O sistema passa a trabalhar com uma tecnologia de renderização específica.

Quadro 14 - Caso de uso 02

UC02 – Executar jogo	
Descrição	Permite que o Usuário do Motor customize a rotina de renderização.
Cenário principal	1. O `Usuário do Motor` chama o método `Game.init`(). 2. O motor começa o *loop* principal do jogo, chamando o UC03
Pós-Condição	O motor executará a rotina de renderização definida pelo usuário ao invés da rotina padrão.

Quadro 15 - Caso de uso 03

UC03 – Desenhar cena	
Descrição	Permite que o Usuário do Motor customize a rotina de renderização.
Cenário principal	1. O motor itera sobre os componentes presentes e desenha os mesmos, a partir do manuseador de API definido no sistema.
Pós-Condição	O motor desenhará os componentes de acordo com seu tipo.

Quadro 16- Caso de uso 04

UC04 – Alternar entre visualização em 2D e 3D	
Descrição	Permite mudar o modo de visualização das peças.
Cenário principal	1. Clicar na peça `Renderer`. 2. Mudar o valor do campo `Gráficos`.
Pós-Condição	Sistema irá alternar entre a visualização em 2D ou 3D sem perder o estado das peças.

Quadro 17 - Caso de uso 05

UC05 – Navegar pelo espaço gráfico	
Descrição	Permite navegar pelo espaço gráfico.
Cenário principal	1. O usuário clica com o botão esquerdo do mouse sobre o espaço gráfico e, com o botão apertado, arrasta o mouse. 2. A câmera do espaço gráfico se moverá, porém continuará a apontar para o mesmo ponto.
Fluxo Alternativo 1	1. O usuário clica com o botão direito do mouse sobre o espaço gráfico e, com o botão apertado, arrasta o mouse. 2. A câmera do espaço gráfico se moverá e o ponto para onde está apontando acompanhará esse movimento.
Fluxo Alternativo 2	1. O usuário gira a roda do mouse 2. O zoom do visualizador aumenta ou diminui, de acordo com o sentido do giro.
Pós-Condição	Sistema irá alternar entre a visualização em 2D ou 3D sem perder o estado das peças.

APÊNDICE B – Lista de exercícios do teste de operacionalidade

A atividade que foi aplicada no teste de operacionalidade da aplicação pode ser vista nos itens Figura 57, Figura 58, Figura 59, Figura 60 e Figura 61.

Figura 57 - Primeira página da atividade do teste de operacionalidade

DSC – Departamento de Sistemas e Computação (FURB)
Grupo de Pesquisa em Computação Gráfica, Processamento de Imagens e Entretenimento Digital
Disciplina: Computação Gráfica - prof. Dalton Solano dos Reis

Unidade 04 - Conceitos básicos de 3D

Informações utilizadas no exercício:
- aonde encontrar o VisEdu-CG: http://www.inf.furb.br/gcg/visedu/cg/
- os arquivos das respostas devem ser zipados e salvos no AVA em VisEdu-CG
- cada arquivo de resposta deve ser nomeado com o número do exercício em questão (1.json, 2.json, 3.json, 4.json e 5.json)
- após terminar o exercício FAVOR RESPONDER o questionário: https://docs.google.com/forms/d/1nJGC2UW8pVNWd9k6Y5dG_cH-lMKcaiegdz7mpXGM3eA/viewform

1) (peso 2,0) Crie uma cena utilizando o VisEdu-CG que **somente** contenha as peças "Camera", "Objeto Gráfico", "Cubo" (com textura "Logo Grupo CG") e "Iluminação". Após altere as "Propriedades" da peça "Camera 1" no "Renderer" para mudar o ponto de vista do observador na cena. A visualização do objeto "Cubo 1" deve ser a mais próxima possível ao exibido na imagem abaixo. Neste caso utilize os valores de "Near 10", "Far 300" e "FOV 45" também descrito na figura abaixo, mudando os valores de "Posicao" e "Look At". Os valores de "Tamanho" e "Posição" do "Cubo 1"não devem ser alterados.

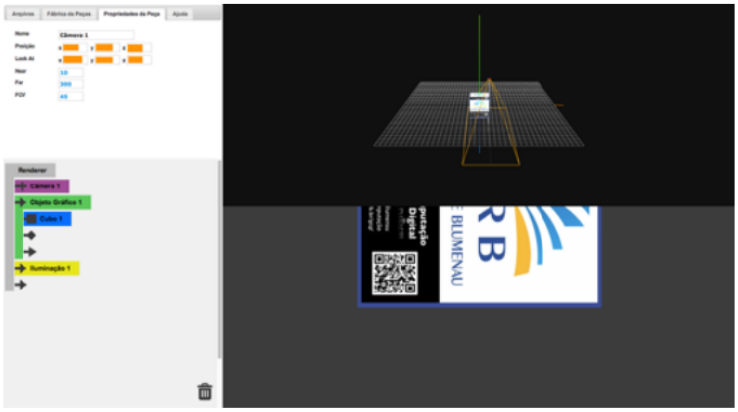


Figura 58 - Segunda página da atividade do teste de operacionalidade

2) (peso 2,0) Crie uma cena utilizando o VisEdu-CG que **somente** contenha as peças descritas na figura abaixo para que tem as mesmas texturas e representação visual. No caso o tamanho do objeto "cubo" foi mantido com 100 unidades e esta posicionado com a face de baixo (a do cubo) com valor no plano "zero" do eixo Z. Os Objetos "Polígono" e "Spline" tem a sua BBox com o mesmo tamanho do "cubo" e estão alinhado com o eixo Z em zero. O "cubo" tem a face da frente também alinhada com o eixo Z em zero.

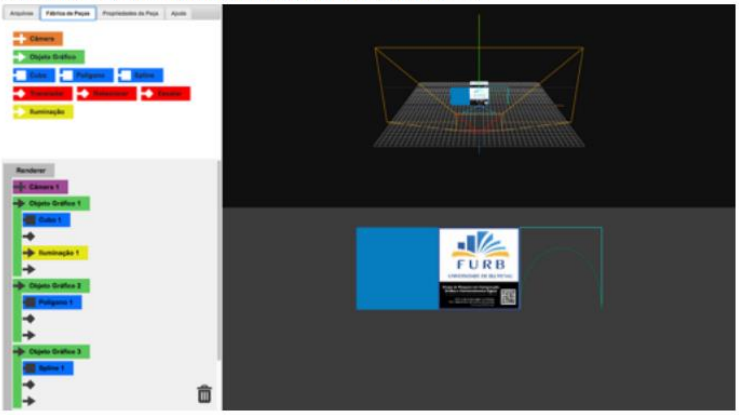


Figura 59 - Terceira página da atividade do teste de operacionalidade

3) (peso 2,0) Crie uma cena utilizando o VisEdu-CG para ter o resultado visual conforme figura abaixo. Não é permitido mudar os parâmetros dos objetos “Cubo”, somente o valor da sua textura. O objeto “Cubo 1” tem o seu centro com coordenadas (0,0,0) coincidindo com a origem, os outros dois “Cubos” sofreram transformação geométrica e se encontram adjacentes ao “Cubo 1”. Todos os objetos “Cubo” são ampliados somente na altura em duas vezes ao seu tamanho original usando uma transformação geométrica. Neste caso esta cena **somente** pode conter as peças: uma "Câmera", uma “Iluminação”, três "Objetos Gráficos", três "Cubo", uma "Escalar" e dois “Transladar”.

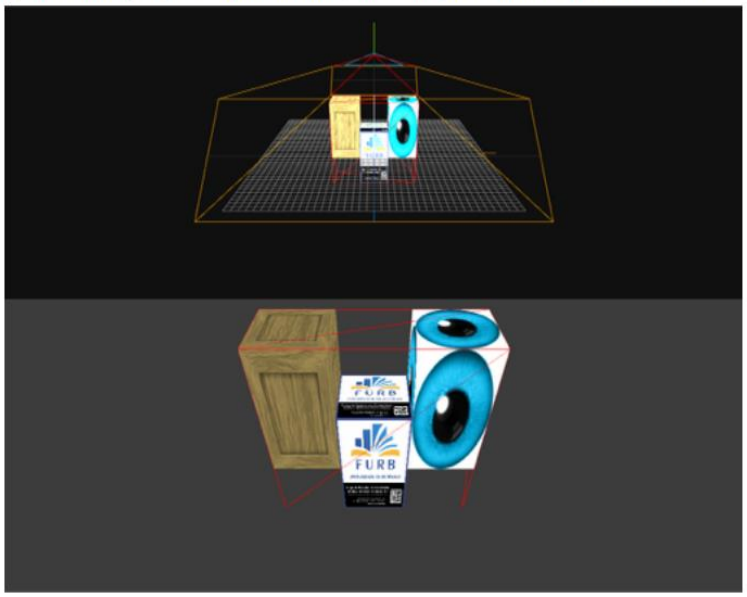


Figura 60 - Quarta página da atividade do teste de operacionalidade

4) (peso 2,0) Use o arquivo “CG-04_exer_04.json” (entrada) disponível em http://gcg.inf.furb.br/visedu/cg/exercicios/CG-04_exer_04.json e com **somente** as peças descritas na figura abaixo para reproduzir o efeito visual gerado pela iluminação.

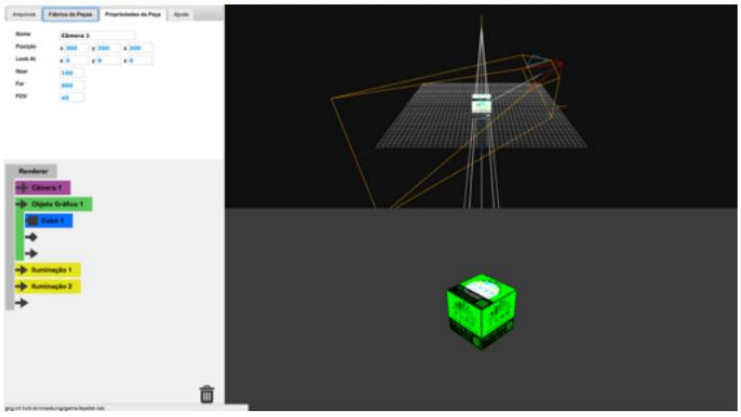
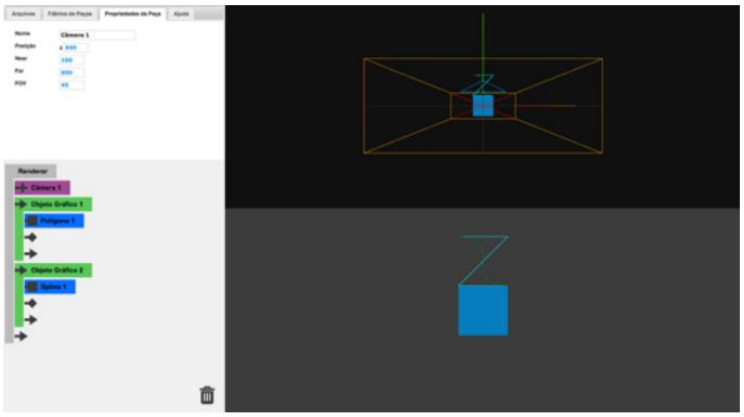


Figura 61 - Quinta página da atividade do teste de operacionalidade

5) (peso 2,0) Crie uma cena utilizando o VisEdu-CG usando **somente** as peças descritas na figura abaixo. O resultado visual deve ser idêntico ao exibido na figura. Mude opção do "Render"/"Gráficos" para usar **Espaço Gráfico 2D.** O Objeto "Polígono" tem uma largura e altura de 100 unidades e o seu centro está posicionado na origem. Já o objeto "Spline" tem a sua base inferior adjacente a base superior do "Polígono". Também observe que o poliedro de controle da "Spline" forma a letra "Z".



APÊNDICE C – Respostas das questões do questionário de operacionalidade

As respostas das questões referentes ao ponto positivo do uso do VisEdu-CG 4.0 estão descritas no Quadro 18.

Quadro 18 - Respostas da questão referente aos pontos positivos

Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo?
O layout é agradável e o aplicativo facilitou muito o entendimento de câmera sintética.
A facilidade para ver as transformações gráficas acontecendo e compreende-las de forma mais fácil.
Possibilita aplicar os conceitos apreendidos.
Drag and drop dos componentes visuais
Fácil de usar;
Online, não precisa de instalação. Intuito simples.
Ajuda a entender os conceitos basicos de OpenGl
A facilidade em adicionar objetos e poder visualizá-los tanto no modo de edição, quanto no modo "final".
Usabilidade
Torna mais visual os conceitos aprendidos em sala de aula, e de uma forma rápida e fácil disponibiliza para o aluno o controle da cena para que o mesmo amplie seus conhecimentos.
O aplicativo é intuitivo e possibilita testar o compartamento de cenas de forma rápida.

As respostas das questões referentes ao ponto positivo do uso do VisEdu-CG 4.0 estão descritas no Quadro 19.

Quadro 19 - Respostas das questão referente aos pontos negativos

Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo?
Ao alterar as propriedades a necessidade de dar enter é um incômodo.
Creio que ainda existam alguns bugs, por exemplo o botão para excluir objetos da cena não está funcionando.
O uso do componente polígono. Falta de aprofundamento com relação as funcionalidades dos componentes na opção de "Ajuda".
Algumas incompatibilidades com alguns navegadores;
Polígno não respeita a quantidade de pontos.
Apresentou problemas em alguns navegadores, problemas de compatibilidade.
O ponto negativo que vi, seria a dificuldade de remover objetos, tem momentos em que se seleciona o texto e não o item.
Faltou um pouco de intuitividade no uso do polígono.
Na mudança das propriedades dos objetos, o "enter" ou "tab" não é tão intuitivo, fazendo com que as vezes os valores não mudem apenas por esse detalhe, como no caso de trocar de campo utilizando o mouse.

As respostas das questões referentes ao ponto positivo do uso do VisEdu-CG 4.0 estão descritas no Quadro 20.

Quadro 20 - Respostas da questão referentes às sugestões

Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?
Deveria alterar as propriedades do objeto ao sair dos inputs e não ao dar enter.
Fazer um revisão nos pequenos bugs da interface com o usuário.
Estudar a possibilidade de alterar o valor z do polígono. Hoje, somente valores x e y podem ser alterados. Quando se cria mais pontos, estes são gerados com valor z zerados.
Resolver problemas de incompatibilidade;
Poder movimentar a câmera com mouse. Ajustar o poligno para 4 pontos, tive que edita-lo pelo json.
Poder alterar os objetos de forma visual, selecionando e movendo.
Aumentar ou fazer com que de alguma forma tenha-se mais espaço para movimentar. O outro seria poder alterar ou movimentar o objeto com o mouse e alterar os valores do cadastro automaticamente.
Ele já é ótimo, mas ampliar a quantidade de objetos seria legal.
Uma sugestão é a criação de um botão "Aplicar" na parte de alteração das propriedades dos objetos.
seria interessante mostar o código gerado para a cena.