

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

**ANIMAR: DESENVOLVIMENTO DE UMA FERRAMENTA
PARA CRIAÇÃO DE ANIMAÇÕES COM REALIDADE
AUMENTADA E INTERFACE TANGÍVEL**

RICARDO FILIPE REITER

BLUMENAU
2018

RICARDO FILIPE REITER

**ANIMAR: DESENVOLVIMENTO DE UMA FERRAMENTA
PARA CRIAÇÃO DE ANIMAÇÕES COM REALIDADE
AUMENTADA E INTERFACE TANGÍVEL**

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(a). Dalton Solano dos Reis, M.Sc. - Orientador

**BLUMENAU
2018**

**ANIMAR: DESENVOLVIMENTO DE UMA FERRAMENTA
PARA CRIAÇÃO DE ANIMAÇÕES COM REALIDADE
AUMENTADA E INTERFACE TANGÍVEL**

Por

RICARDO FILIPE REITER

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(a). Dalton Solano dos Reis, M.Sc. – Orientador, FURB

Membro: _____
Prof(a). Everaldo Artur Grahl, M. Eng. – FURB

Membro: _____
Prof(a). Mauricio Capobianco Lopes, Dr. – FURB

Blumenau, 11 de julho de 2018

Dedico este trabalho à minha família que sempre me apoiou desde o início da minha vida.

AGRADECIMENTOS

Aos meus pais por todo o incentivo durante o curso.

Ao meu orientador por toda a ajuda dada e o interesse pelo trabalho desde o início.

Ao professor Maurício Capobianco Lopes que disponibilizou parte da sua aula com a turma de pedagogia para testar a ferramenta.

Aos meus colegas de curso e de profissão que me acompanharam desde o começo do curso.

Nós só podemos ver um pouco do futuro, mas o suficiente para perceber que há muito a fazer.

Alan Turing

RESUMO

Este trabalho apresenta o desenvolvimento de uma ferramenta para criação de cenas animadas, utilizando os conceitos de Realidade Aumentada e Interfaces de Usuário Tangíveis. O aplicativo permite a criação e manipulação de cenários e objetos tridimensionais virtuais, sendo possível dar “vida” à cena ao utilizar Interfaces Tangíveis para a criação de animações dos objetos virtuais. A ferramenta foi desenvolvida com a *engine* Unity, em conjunto com a biblioteca Vuforia para a aplicação do conceito de Realidade Aumentada. O aplicativo utiliza o conceito de Interfaces Tangíveis com botões virtuais nos marcadores e com um marcador em formato de cubo, utilizado para mover, adicionar e remover objetos da cena, e para a captura de movimentos no momento da gravação das animações. Este trabalho também apresenta conceitos, técnicas e abordagens utilizadas para a implementação das funcionalidades necessárias. Para a realização dos testes, o aplicativo foi exportado para as plataformas Android e iOS, sendo testado com uma turma do curso de Pedagogia da FURB, que apesar de sentirem uma certa dificuldade no início para utilizar e entender a ferramenta, mostraram-se muito interessados e a maioria conseguiu concluir os objetivos propostos pela ferramenta dentro do tempo estipulado. A maior dificuldade percebida pelos usuários foi na parte de interação com a Interface Tangível, isto porque ainda há pouca familiaridade com este tipo de interface para a grande maioria das pessoas. Ainda assim, os testes realizados apontam que as funcionalidades desenvolvidas são de fácil uso, após o entendimento inicial de como a Interface de Usuário Tangível funciona.

Palavras-chave: Realidade aumentada. Interface de usuário tangível. Vuforia. Unity. Animação. Cenas animadas.

ABSTRACT

This work presents the development of a tool for creating animated scenes, using the concepts of Augmented Reality and Tangible User Interfaces. The application allows the creation and manipulation of virtual scenes and tridimensional objects, being possible to give “life” to the scene when using Tangible Interfaces for creating virtual objects animations. The tool was developed with Unity engine, together with the Vuforia framework for applying the concepts of Augmented Reality. The application uses the concept of Tangible Interfaces with virtual buttons on the markers and with a cuboid marker, used for moving, adding and removing objects from the scene, and for motion capture when recording an animation. This work also shows concepts, techniques and approaches used for implementing the necessary functionalities. For proceeding with the testing, the application was exported to Android and iOS platforms, being tested with a pedagogy class from FURB, that even though they felt some difficulties when using the tool in the beginning, they felt really interested and most of them could conclude the proposed tool objectives within the stipulated time. The most difficult part noticed by the users was when interacting with the Tangible Interface, and that is because there’s still an unfamiliarity with this type of interface for most people. However, tests show that the developed functionalities are easy to use after the initial understanding of how the Tangible User Interface Works.

Key-words: Augmented reality. Tangible user interfaces. Vuforia. Unity. Animation. Animated scenes.

LISTA DE FIGURAS

Figura 1 – Captura de movimentos utilizando as técnicas ópticas	16
Figura 2 – Aplicação URP.....	18
Figura 3 – Objeto sendo movido para a lixeira	19
Figura 4 – Interface de Usuário Tangível através do uso de um marcador de Cubo.....	20
Figura 5 – Marcador de simulação do Sistema Solar	21
Figura 6 – Criação de um novo objeto virtual	22
Figura 7 – Diagrama de Casos de Uso da aplicação.....	24
Figura 8 – Diagrama das principais Classes	27
Figura 9 – Diagrama de Classes de persistência.....	29
Figura 10 – Diagrama de Atividades	31
Figura 11 – Opção para ativar o Vuforia no projeto	33
Figura 12 – Adicionando o AR Camera no projeto	33
Figura 13 – Obtendo uma nova chave de licença para desenvolvimento no portal do Vuforia	34
Figura 14 – Página para adicionar um marcador ao <i>database</i>	35
Figura 15 – Página para adicionar um novo marcador	36
Figura 16 – Imagem de um quadrado com as características naturais destacadas	37
Figura 17 – Imagem de um círculo sem nenhuma característica natural detectada	37
Figura 18 – Imagem de um chão de pedras usada como marcador	38
Figura 19 – Imagem das características encontradas pelo Vuforia	38
Figura 20 – <i>Screenshot</i> da ferramenta AR Marker Generator.....	39
Figura 21 – Imagem utilizada para o marcador Seletor.....	40
Figura 22 – Características naturais encontradas pelo Vuforia no marcador seletor	40
Figura 23 – Marcador cubo	41
Figura 24 – Marcadores inspetor e lixeira	41
Figura 25 – Marcadores cena e gravador.....	42
Figura 26 – Árvore hierárquica da principal <i>scene</i> do AnimAR.....	43
Figura 27 – Objetos de realidade aumentada organizados na <i>scene</i>	44
Figura 28 – Inspector do <i>GameObject Scene</i>	45
Figura 29 – Inspector do <i>GameObject Selector</i>	47
Figura 30 – <i>BoxCollider</i> posicionado no cubo	50

Figura 31 – SphereCollider posicionado na lixeira.....	58
Figura 32 – Objetos do tipo SCENE_OBJECT e TAKE_OBJECT sendo movidos para a lixeira	58
Figura 33 – Menu inicial da aplicação.....	63
Figura 34 – Marcador Cena, visto pela visão da câmera	63
Figura 35 – Marcador Seletor no modo Fábrica de Objetos.....	64
Figura 36 – Trocando o modo de seleção do marcador Seletor.....	64
Figura 37 – Mudando a cena atual.....	65
Figura 38 – Adicionando um novo objeto na cena.....	66
Figura 39 – Adicionando um novo objeto na cena.....	66
Figura 40 – Gravando uma animação.....	67
Figura 41 – Marcador Inspetor inspecionando objetos da Cena e do Seletor	68
Figura 42 – Movendo objetos para a lixeira.....	68

LISTA DE QUADROS

Quadro 1 – Relação entre Requisitos Funcionais e Casos de Uso	26
Quadro 2 – Métodos de integração com o cubo da classe SceneController.....	45
Quadro 3 – Trecho de código da implementação da classe SelectorController	48
Quadro 4 – Trecho de código da implementação da classe ObjectFactorySelector ..	49
Quadro 5 – Trecho de código da implementação da classe CubeMarkerController.....	51
Quadro 6 – Métodos de gravação da classe AnimationController	53
Quadro 7 – Métodos ao finalizar gravação na classe AnimationController.....	55
Quadro 8 – Métodos de execução e parada das animações na classe AnimationController.....	56
Quadro 9 – Método invocado pelo marcador cubo na classe TrashBinController.....	57
Quadro 10 – Implementação do evento do Collider na classe InspectorController	59
Quadro 11 – Métodos que salvam e carregam os dados na classe PersistController .	60
Quadro 12 – Método da classe ChangeVuforiaConfig que configura a visualização do Vuforia.....	62
Quadro 13 – Comparativo entre os trabalhos correlatos	75

LISTA DE TABELAS

Tabela 1 – Perfil dos usuários.....	70
Tabela 2 – Passo a passo das funcionalidades básicas	71
Tabela 3 – Modo livre	72
Tabela 4 – Avaliação Geral	73

SUMÁRIO

1 INTRODUÇÃO.....	13
1.1 OBJETIVOS.....	13
1.2 ESTRUTURA.....	14
2 FUNDAMENTAÇÃO TEÓRICA	15
2.1 ANIMAÇÕES DIGITAIS	15
2.2 REALIDADE AUMENTADA	16
2.3 INTERFACE DE USUÁRIO TANGÍVEL.....	17
2.4 TRABALHOS CORRELATOS	18
2.4.1 VISEDU: Interface de Usuário Tangível utilizando Realidade Aumentada e Unity	18
2.4.2 Desenvolvimento de uma ferramenta para auxiliar no ensino do Sistema Solar utilizando Realidade Aumentada	19
2.4.3 Immersive Authoring of Tangible Augmented Reality applications	21
3 DESENVOLVIMENTO DA FERRAMENTA	23
3.1 REQUISITOS.....	23
3.2 ESPECIFICAÇÃO	24
3.2.1 Diagrama de Casos de Uso	24
3.2.2 Diagrama de classes	26
3.2.3 Diagrama de atividades	30
3.3 IMPLEMENTAÇÃO	32
3.3.1 Técnicas e ferramentas utilizadas.....	32
3.3.2 Operacionalidade da implementação	62
3.4 ANÁLISE DOS RESULTADOS	69
3.4.1 Metodologia	69
3.4.2 Experimentos e resultados.....	69
3.4.3 Comparativo com os trabalhos correlatos	74
4 CONCLUSÕES.....	76
4.1 EXTENSÕES	77
REFERÊNCIAS	78

1 INTRODUÇÃO

Animações digitais (CGI) e efeitos especiais (VFX) estão cada vez mais presentes no cotidiano. Filmes produzidos totalmente através de computação gráfica e outros com vários efeitos especiais embutidos, já são algo rotineiro nos dias atuais e isso só tende a crescer, conforme Diana Giorgiutti (2016, apud GIARDINA, 2016, tradução nossa), “Em Matrix foram produzidas 420 cenas de efeitos especiais. Hoje em dia, são em torno de 2000 cenas em um filme da Marvel, e eles lançam um filme por ano”. Entretanto, filmes não são os únicos a utilizar de animação digital, pois os jogos eletrônicos são conhecidos pelas suas animações dinâmicas.

Segundo Brandão (2012, p. 10), “Enquanto espectador, estamos sujeitos à montagem fixa e linear dos filmes, enquanto que nos jogos eletrônicos temos controle total ou parcial sobre o ângulo e a duração dos planos imagéticos.”. Neste contexto, o que também está se tornando popular é a Realidade Aumentada.

Kirner et al. (2006) definem a Realidade Aumentada (RA) como uma técnica para trazer o ambiente virtual ao ambiente físico do usuário, permitindo a interação com o mundo virtual, de maneira natural e sem necessidade de treinamento ou adaptação. Kirner et al. (2006, p. 22) também citam “Novas interfaces multimodais estão sendo desenvolvidas para facilitar a manipulação de objetos virtuais no espaço do usuário, usando as mãos ou dispositivos mais simples de interação.”, que é o caso de uma Interface de Usuário Tangível (IUT).

[...] interfaces Tangíveis, dão forma física para informações digitais, empregando artefatos físicos como **representações** e **controles** para mídias computacionais. TUIs combinam representações físicas (por exemplo, objetos físicos manipuláveis espacialmente) com representações digitais (por exemplo, visual e áudio), produzindo sistemas interativos que são mediados pelo computador, mas geralmente não identificáveis como “computadores”. (ULLMER, ISHII, 2001, p. 2, grifo do autor, tradução nossa).

Diante deste contexto, este trabalho desenvolveu um meio de se manipular um cenário virtual e criar animações com objetos virtuais, através do uso da Realidade Aumentada para visualização e feedback, e do uso de uma Interface de Usuário Tangível para o usuário manipular a cena (translação, rotação, escala, gravar, entre outros recursos).

1.1 OBJETIVOS

O objetivo deste trabalho é desenvolver uma ferramenta de criação de animações em 3D através de uma combinação de Interface de Usuário Tangível e Realidade Aumentada.

Os objetivos específicos são:

- a) disponibilizar uma ferramenta para criação de cenários;
- b) disponibilizar o uso pedagógico da ferramenta, para que possa auxiliar no desenvolvimento criativo das crianças;
- c) disponibilizar a utilização da ferramenta através de um head-mounted display (HMD), como o Cardboard.

1.2 ESTRUTURA

Este trabalho encontra-se dividido em quatro capítulos principais. O primeiro capítulo, tem como objetivo apresentar a introdução do trabalho e seus objetivos gerais e específicos. O segundo, apresenta a fundamentação teórica requerida para o entendimento e contextualização deste trabalho. O terceiro capítulo apresenta as etapas de desenvolvimento deste trabalho, mostrando a especificação do projeto com diagramas de classe, diagramas de caso de uso e de atividades. O terceiro capítulo também apresenta as técnicas e ferramentas utilizadas para a implementação do trabalho, contendo imagens e trechos de códigos para um entendimento geral da solução e, por fim, apresenta uma análise dos resultados das pesquisas realizadas com a utilização da ferramenta. O quarto e último capítulo apresenta uma conclusão do desenvolvimento deste trabalho, analisando a solução como um todo e os resultados atingidos, e sugestões para possíveis extensões deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 tem como objetivo apresentar os conceitos básicos de Animações Digitais. Na seção 2.2 serão apresentados os conceitos de Realidade Aumentada. Na seção 2.3 serão apresentados os conceitos de Interface de Usuário Tangível. Por fim, a seção 2.4 irá apresentar os trabalhos correlatos.

2.1 ANIMAÇÕES DIGITAIS

Segundo Parent (2001), animar significa literalmente “dar vida”.

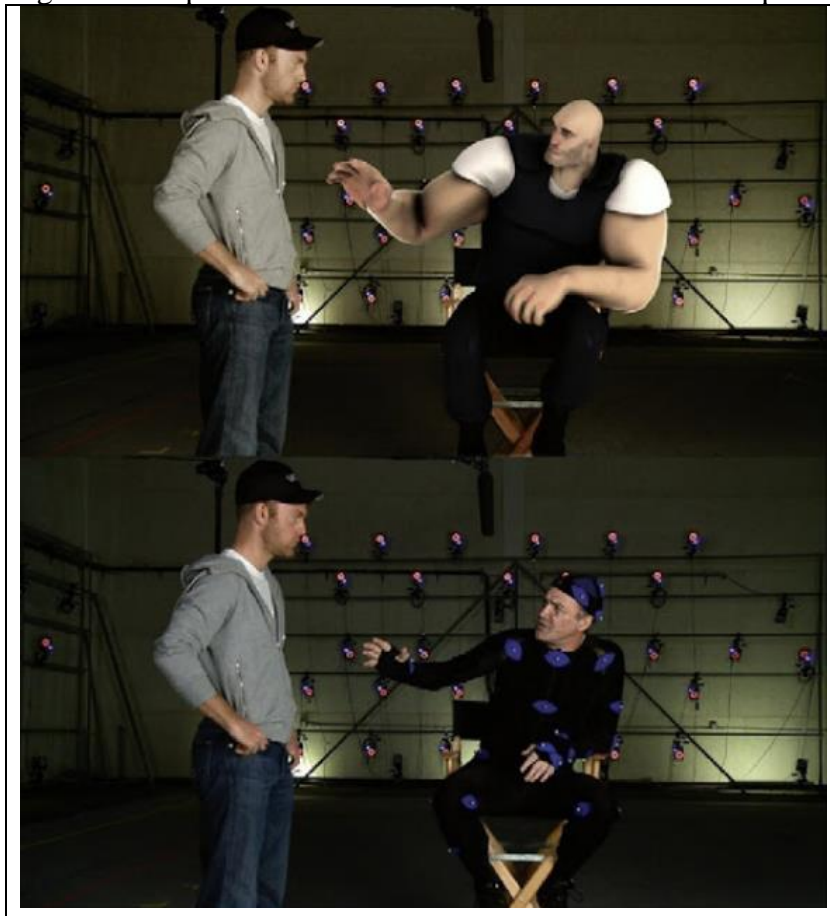
Animar é mover algo (ou fazer algo parecer que move) que não pode mover-se sozinho—seja ela um fantoche do King Kong, um desenho da Branca de Neve, os ponteiros de um relógio, ou uma imagem sintética de um brinquedo de cowboy de madeira. (PARENT, 2001, p. 16, tradução nossa)

Animações vêm sendo utilizadas para ensinar e entreter, desde a época dos fantoches até hoje em dia com filmes e jogos (PARENT, 2001, p. 16, tradução nossa). Parent (2001, p. 16) explica que as animações adicionam a dimensão de tempo para a computação gráfica, abrindo espaço para transmitir informação e conhecimento para o espectador, acendendo suas emoções e liberando a imaginação de criança de dentro de todos nós.

Uma das técnicas utilizadas para a criação de animações é a de *motion capture* que, segundo Menache (2011, p. 2), é um processo na qual se grava um movimento do mundo real, coletando-se informações de pontos chaves ao longo do tempo, para no final combiná-los em uma única representação tridimensional daquele movimento. “Resumindo, é a tecnologia que possibilita o processo de se traduzir uma performance do mundo real em uma performance digital.” (MENACHE, 2011, p. 2, tradução nossa). O objeto a ser capturado pode ser qualquer objeto do mundo real que contenha movimento, sendo os pontos chaves as áreas que melhor representam os movimentos das partes móveis do objeto como, por exemplo, as juntas dos ossos do corpo humano (MENACHE, 2011, p. 2).

Uma das formas de se capturar movimentos é através dos sistemas de captura de movimento ópticos, que utiliza várias câmeras especiais, conhecidas por Charge-Coupled Device (CCD), que captam ângulos diferentes do objeto em movimento, e um computador que obtêm as informações das câmeras. Com as informações obtidas das câmeras, um software gera no final um arquivo contendo as informações daquela movimentação, com a posição de cada ponto chave registrado ao longo do tempo (MENACHE, 2011, p. 17).

Figura 1 – Captura de movimentos utilizando as técnicas ópticas



Fonte: Menache (2011).

2.2 REALIDADE AUMENTADA

“Na década de 60, surgiram os consoles com vídeo, dando início às interfaces gráficas rudimentares.” (KIRNER; SISCOOTTO, 2007, p. 4), entretanto, foi com a popularização dos microcontroladores nas décadas de 70 e 80, que interfaces baseadas em comandos como o DOS surgiram e evoluíram a ponto de resultar no que conhecemos hoje como Windows. Apesar de a interface Windows ser interessante, ela fica restrita à limitação da tela do monitor e ao uso de representações como menus e ícones (KIRNER; SISCOOTTO, 2007, p. 4).

A realidade virtual surge então como uma nova geração de interface, na medida em que, usando representações tridimensionais mais próximas da realidade do usuário, permite romper a barreira da tela, além de possibilitar interações mais naturais. (KIRNER; SISCOOTTO, 2007, p. 4)

Apesar da Realidade Virtual (RV) possuir vantagens quando comparada à interface Windows, ela ainda necessita de equipamentos especiais como capacete, óculos estereoscópicos, entre outros, para que seja possível transportar o usuário para o espaço virtual da aplicação. Esses problemas acabaram por impedir a popularização da realidade virtual como uma nova interface (KIRNER; SISCOOTTO, 2007, p. 5).

Insley (2003 apud KIRNER; SISCOOTTO, 2007, p. 10) define a realidade aumentada como uma melhoria do mundo real com textos, imagens e objetos virtuais, gerados por computador. Kirner e Siscoutto (2007, p. 5) também descrevem a realidade aumentada como a sobreposição de objetos e ambientes virtuais com o ambiente físico, unindo os dois ambientes em um só, através de algum dispositivo tecnológico.

A realidade aumentada permite ter uma fácil utilização quando comparada à realidade virtual. Conforme Kirner e Siscoutto (2007, p. 5), “[...] o fato dos objetos virtuais serem trazidos para o espaço físico do usuário (por sobreposição) permitiu interações tangíveis mais fáceis e naturais, sem o uso de equipamentos especiais.”. A realidade aumentada vem sendo considerada uma possibilidade de ser a próxima geração de interface popular, isto porque a realidade aumentada dispensa o uso dos equipamentos especiais, e pode ser utilizada tanto em ambientes internos quanto externos (KIRNER; SISCOOTTO, 2007, p. 5).

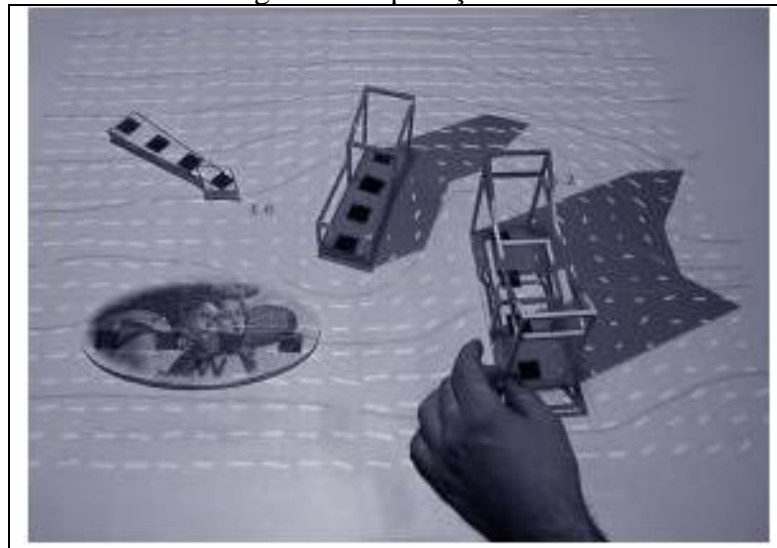
2.3 INTERFACE DE USUÁRIO TANGÍVEL

As interfaces de usuário gráficas modernas fazem uma grande distinção entre os “dispositivos de entrada” e os “dispositivos de saída”. Dispositivos como mouse e teclado são vistos somente como controles e dispositivos gráficos como monitores e head-mounted displays são vistos como representações visuais. As interfaces de usuário tangível exploram o conceito de espaço aberto pela eliminação desta distinção, juntando os controles e representações em uma coisa só (ULLMER; ISHII, 2001).

Ullmer e Ishii (2001) definem Interfaces de Usuário Tangíveis como representações físicas para informações digitais, permitindo que objetos físicos ajam como representações e controles para um mundo virtual. Diferentemente de teclados e mouses que também são objetos físicos, as formas e posições físicas dos objetos de interfaces tangíveis desempenham um importante papel para o mundo virtual (ULLMER; ISHII, 2001).

A Figura 2 mostra a aplicação URP para planejamento urbano, que utiliza uma interface de usuário tangível. Esta aplicação permite a manipulação direta de modelos físicos de construções para configurar e controlar uma simulação de um ambiente urbano (UNDERKOFFLER, 1993 apud ULLMER; ISHII, 2001). As sombras do prédio são projetadas de acordo com a posição do sol, que pode ser controlada girando os ponteiros de um relógio físico. Há também uma simulação de fluxo do vento, que pode ser controlado girando o ponteiro físico “controlador de vento”, assim, a direção do vento corresponderá à orientação física do ponteiro (ULLMER; ISHII, 2001).

Figura 2 – Aplicação URP



Fonte: Ullmer e Ishii (2001).

Quando se olha para um mouse, sua função é a de controlar o cursor gráfico da Interface de Usuário, o que pode ser realizado por quaisquer outros dispositivos de entrada como um *trackball*, joystick, caneta digitalizadora, etc. O exemplo da URP mostra como a interface está intimamente acoplada à identidade e configuração física dos artefatos físicos. (ULLMER; ISHII, 2001).

2.4 TRABALHOS CORRELATOS

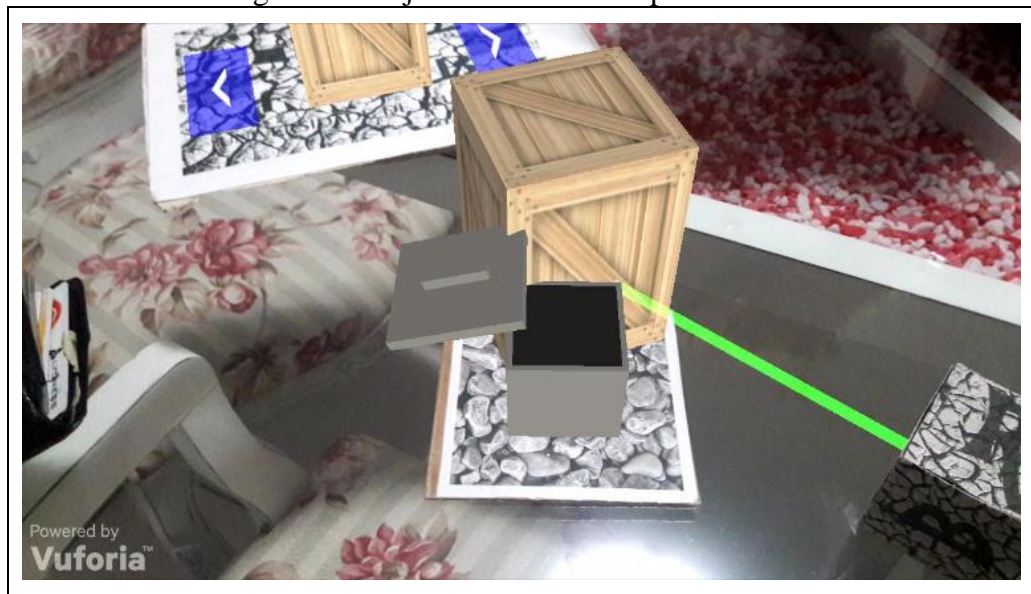
A seguir são apresentados trabalhos com características semelhantes ao objetivo do estudo proposto. O primeiro descreve um trabalho de conclusão de curso de Silva (2016) que desenvolveu um aplicativo para a plataforma Android que utiliza Interface de Usuário Tangível e Realidade Aumentada para a manipulação de objetos tridimensionais virtuais. O segundo é um trabalho de conclusão de curso de Schmitz (2017) o qual desenvolveu um aplicativo Android que utiliza Realidade Aumentada para auxiliar no ensino do Sistema Solar. Por fim, o artigo desenvolvido por Lee et al. (2004) que sugere uma nova abordagem para Sistemas de Criação, através de Realidade Aumentada e Interface de Usuário Tangível.

2.4.1 VISEDU: Interface de Usuário Tangível utilizando Realidade Aumentada e Unity

Silva (2016, p. 14) desenvolveu um aplicativo para a plataforma Android que teve como objetivo “[...] desenvolver uma interface de usuário tangível para manipular objetos tridimensionais virtuais utilizando realidade aumentada.”. Para o desenvolvimento, ele utilizou o motor de jogos Unity 3D e o Kit de desenvolvimento de Software (SDK) Vuforia para RA.

O trabalho utiliza marcadores para a visualização da cena e objetos virtuais, e a criação de uma interface de usuário tangível, por onde se faz possível a manipulação e interação com os objetos virtuais. Os marcadores são capturados através da câmera do smartphone, e os objetos são renderizados sobre a imagem dela para a criação de uma Realidade Aumentada. Ao identificar o marcador para criar objetos, o usuário pode então criar um objeto, escolhendo dentre algumas opções fornecidas pelo software. Com um marcador de cubo com seis imagens, se cria uma ferramenta para o usuário selecionar e movimentar os objetos, podendo adicioná-los às cenas, modificar os atributos do mesmo, ou até mesmo excluir o objeto, movendo-o para o marcador de lixeira. A Figura 3 mostra um objeto sendo movido com o marcador de cubo para a lixeira.

Figura 3 – Objeto sendo movido para a lixeira



Fonte: Silva (2016).

Silva (2016, p. 63) conclui que apesar de se perceber a não familiaridade dos usuários com a Interface de Usuário Tangível, “os objetivos propostos foram atingidos com resultados satisfatórios, com isso permitindo ao usuário um entendimento do conceito de Interface de usuário tangível” (SILVA, p. 63).

2.4.2 Desenvolvimento de uma ferramenta para auxiliar no ensino do Sistema Solar utilizando Realidade Aumentada

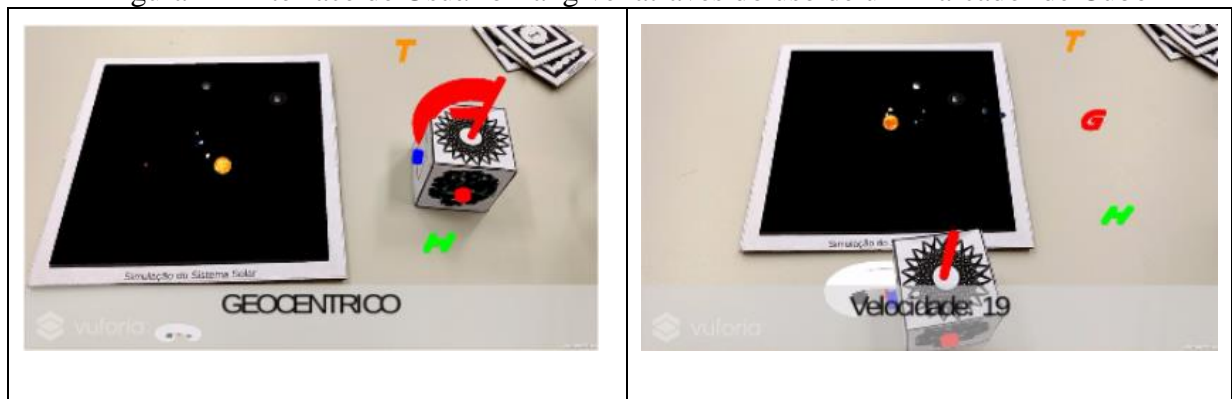
Schmitz (2017) teve como objetivo o desenvolvimento de uma ferramenta para auxiliar no ensino do Sistema Solar utilizando a RA. Para alcançar seu objetivo, ele desenvolveu um aplicativo para a plataforma Android utilizando o motor de jogos Unity 3D e o SDK Vuforia.

O trabalho foi dividido em dois módulos. Schmitz (2017, p. 31) define o primeiro como “[...] módulo de Dissecação do Sistema Solar onde o usuário pode ver os

planetas em detalhes, separadamente ou em conjunto, tendo uma escala real aplicada a eles, podendo mostrar a estrutura interna dos planetas e informações sobre eles.”, e o segundo “[...] Sistema Solar que apresenta todos os oito planetas orbitando o Sol, não observando escalas de tamanho e distância [...]” (SCHMITZ, 2017, p. 31). Schmitz (2017) descreve que ao apontar a câmera do celular para o marcador de Sistema Solar, é apresentada uma simulação do Sistema Solar, podendo ser uma das teorias, Heliocêntrica, Geocêntrica ou Geocêntrica, segundo Tycho Brahe.

Com a utilização do marcador de controle de Cubo, o usuário pode controlar a velocidade da simulação ou alterar a teoria que está sendo exibida. A Figura 4 mostra o Cubo interagindo com a interface tangível. Ao posicionar e girar o cubo sobre a propriedade, esta será alterada.

Figura 4 – Interface de Usuário Tangível através do uso de um marcador de Cubo



Fonte: Schmitz (2017).

Outro marcador de controle é o de Informações, que permite que sejam exibidas informações sobre a teoria que está sendo simulada no momento, ou informações específicas de um planeta. Por fim, o marcador de controle de Dissecação, que possibilita a visualização das camadas dos planetas. O uso em conjunto dos marcadores de Informação e Dissecação, permite que seja alterado o tipo de informação a ser exibido por um planeta. A Figura 5 mostra o marcador de simulação do Sistema Solar em funcionamento.

Figura 5 – Marcador de simulação do Sistema Solar



Fonte: Schmitz (2017).

Após os testes com usuários que Schmitz (2017) desenvolveu, foi percebido fadiga no braço ao utilizar o tablet segurando somente com uma mão após um longo tempo. Outro ponto relatado, foi o desempenho ruim do aplicativo devido à baixa configuração dos tablets testados. Apesar disto, Schmitz (2017, p. 79) explica “O objetivo de disponibilizar uma ferramenta com o intuito de ajudar a ensinar um conteúdo de Geografia, junto com os seus objetivos específicos, foram alcançados e os resultados dos testes comprovam isto.”.

2.4.3 Immersive Authoring of Tangible Augmented Reality applications

Lee et al. (2004) criaram uma nova abordagem para Sistemas de Autoria chamada de Immersive Authoring.

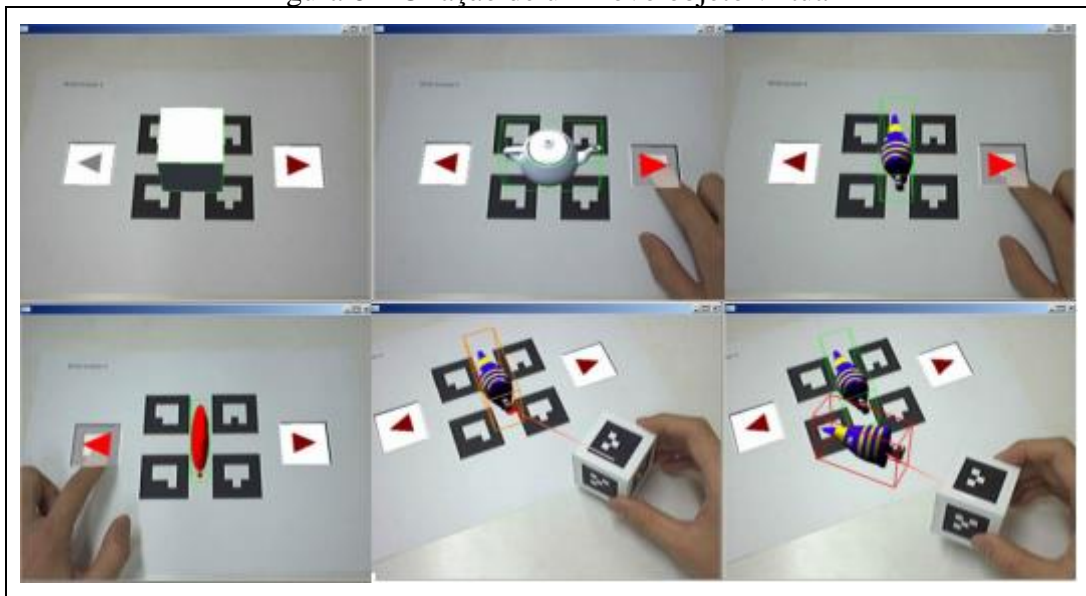
A abordagem permite o usuário desempenhar as tarefas de autoria dentro da aplicação de Realidade Aumentada sendo construída, de modo que o desenvolvimento e o teste da aplicação possam ser feitas simultaneamente ao longo do processo de desenvolvimento. (LEE et al., 2004, p. 1, tradução nossa).

Lee et al. (2004, p. 8) explicam que a aplicação desenvolvida tem como objetivo tornar fácil a criação de aplicações de Autoria com Realidade Aumentada, mesmo que a pessoa não possua experiência com programação. A característica mais importante de um sistema Immersive Authoring descrita por Lee et al. (2004, p. 4) é o conceito de “What You Fell Is What You Get (WYFIWYG).”, em português “O que você sente é o que você obtém” “Isto se refere à habilidade de se sentir todos os elementos sensoriais (visual, aural, e outros elementos se houver) do conteúdo final como se está sendo construído.” (LEE, et al., 2004, p. 4, tradução nossa). “O objetivo principal de Immersive Authoring é tornar possível se

experienciar os mundos virtuais enquanto eles são construídos.” (LEE, et al., 2004, p. 4, tradução nossa). A aplicação foi desenvolvida utilizando OpenGL como biblioteca gráfica e ARToolKit para a identificação e cálculo dos marcadores.

A aplicação faz uso de marcadores para a visualização de um cenário virtual e, através do marcador de controle de Cubo, é possível criar novos objetos, deletar ou manipular os objetos desse cenário como translação, escala e rotação, ou conectá-los a outros componentes como um motor ou som. A Figura 6 mostra o processo de criação de um novo objeto virtual, com a seleção do objeto da lista de objetos disponíveis e a seleção do objeto com o marcador Cubo.

Figura 6 – Criação de um novo objeto virtual



Fonte: Lee et al. (2004).

Lee et al (2004, p. 9, tradução nossa) concluem que “o sistema Immersive Authoring aparenta ser eficiente e fácil de usar, ainda assim viável o suficiente para criar variadas aplicações de RA Tangível”.

3 DESENVOLVIMENTO DA FERRAMENTA

Este capítulo tem como objetivo apresentar os passos do desenvolvimento da aplicação. A seção 3.1 apresenta os requisitos da aplicação. A seção 3.2 tem como objetivo apresentar a especificação do problema. Na seção 3.3 é apresentada de forma detalhada a implementação da ferramenta. Por fim, a seção 3.4 apresenta uma análise dos resultados obtidos.

3.1 REQUISITOS

A aplicação desenvolvida deverá atender os seguintes requisitos:

- a) disponibilizar um menu principal com a opção de iniciar o aplicativo (Requisito Funcional – RF);
- b) disponibilizar no menu principal uma opção para visualizar um pequeno tutorial de usabilidade (RF);
- c) disponibilizar no menu principal uma opção para excluir os dados salvos no dispositivo (RF);
- d) permitir a interação do usuário com o ambiente virtual através do uso de marcadores e as mãos (RF);
- e) permitir o usuário criar/editar/excluir cenas (RF);
- f) permitir o usuário criar/editar/excluir objetos dentro da cena através de um marcador cubo (RF);
- g) permitir o usuário criar/excluir animações para os objetos da cena através de um marcador cubo (RF);
- h) disponibilizar um marcador que é uma lista de objetos pré-definidos para o usuário escolher e utilizar em sua cena (RF);
- i) disponibilizar um marcador como um controle de gravação para cada cena de animação, em que usuário possa executar, pausar e gravar animações (RF);
- j) permitir a gravação e execução de animações simultâneas (RF);
- k) disponibilizar um marcador para inspecionar os objetos da cena, se possui animação ou não (RF);
- l) desenvolver para a plataforma Android (Requisito Não-Funcional – RNF);
- m) permitir a utilização de um head-mounted display, ou cardboard (RNF);
- n) utilizar a SDK Vuforia como biblioteca de Realidade Aumentada (RNF);
- o) desenvolver utilizando o Microsoft Visual Studio Ultimate 2013 como editor de scripts (RNF);

- p) utilizar o motor de jogos Unity 3D para desenvolver o projeto (RNF);
- q) utilizar o Adobe Photoshop CS6 como editor de imagens para os marcadores (RNF);
- r) utilizar a câmera do dispositivo para a captura de marcadores pré-definidos e a renderização do mundo virtual (RNF).

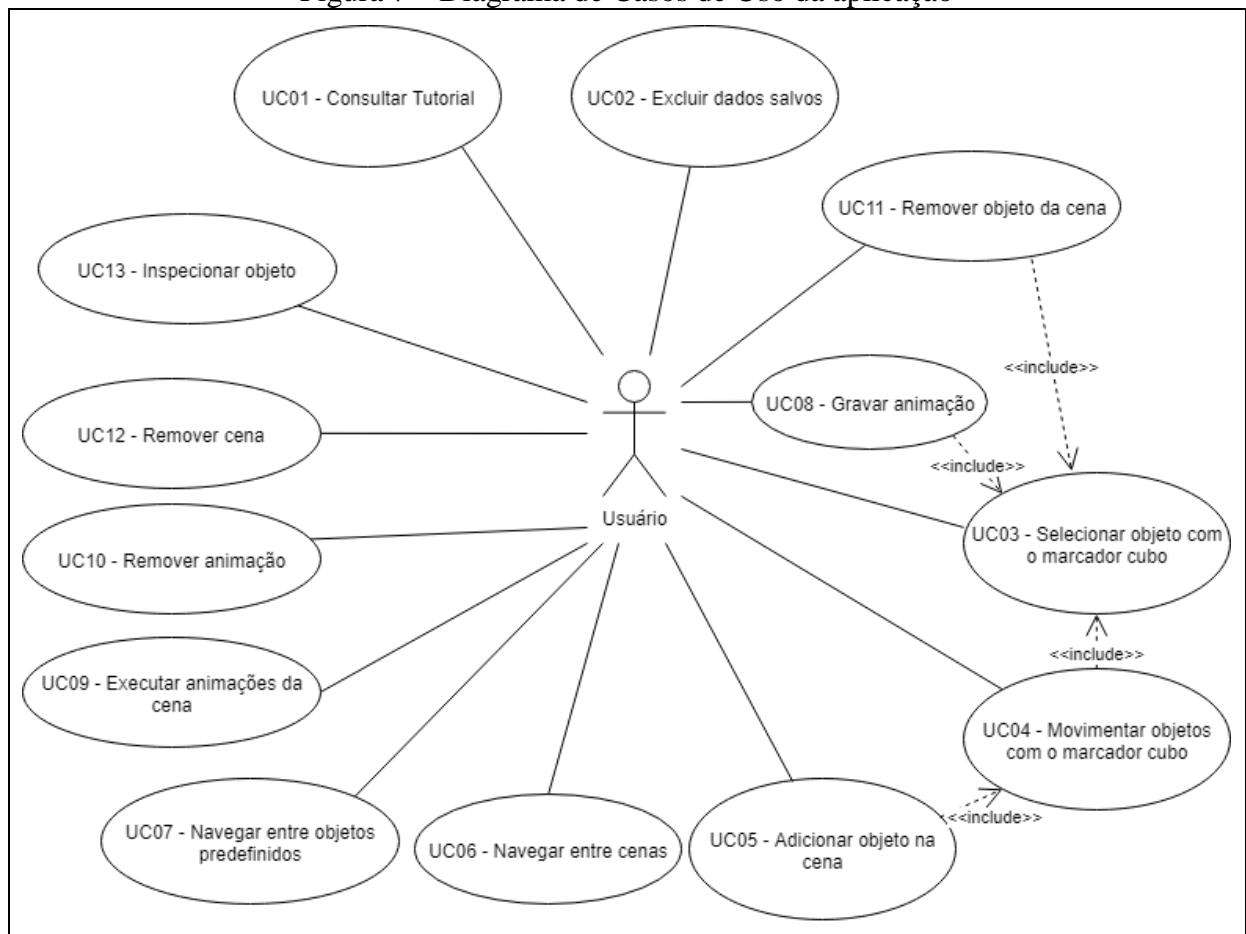
3.2 ESPECIFICAÇÃO

A aplicação foi especificada através da análise orientada a objetos, descrita na linguagem Unified Modeling Language (UML) e utilizando-se da ferramenta Draw.io para a elaboração dos respectivos diagramas.

3.2.1 Diagrama de Casos de Uso

A Figura 7 apresenta o diagrama de Casos de Uso desenvolvido com base nos requisitos funcionais desta aplicação.

Figura 7 – Diagrama de Casos de Uso da aplicação



Fonte: elaborado pelo autor.

No caso de uso UC01 - Consultar Tutorial, o usuário pressiona o botão de tutorial no menu principal, para dar início a um pequeno tutorial de usabilidade. O caso UC02 - Excluir dados salvos possibilita que o usuário remova os dados anteriormente salvos no dispositivo. UC03 - Selecionar objeto com o marcador cubo, neste caso, o usuário utiliza o marcador cubo e com sua ponta consegue selecionar objetos da cena, ou da Fábrica de Objetos. No UC04 - Movimentar objetos com o marcador cubo, o usuário pode movimentar livremente pelo mundo objetos selecionados pelo marcador cubo. O caso UC05 - Adicionar objeto selecionado na cena, o usuário adicionará o objeto atual selecionado na sua cena, exatamente no local desejado. No caso UC06 - Navegar entre cenas, o usuário utiliza o marcador de seletor de cenas para escolher a cena desejada das cenas já criadas, ou criando uma nova, caso a cena selecionada ainda não exista. No caso UC07 - Navegar entre objetos predefinidos, o usuário utiliza o marcador de seletor de objetos para navegar entre uma lista predefinida de objetos, que podem ser utilizados para construir a cena. No caso UC08 - Gravar animação, o usuário utiliza dos marcadores de gravação e do cubo para a gravação de animações, pressionando o botão de gravar, selecionando e movimentando o objeto conforme deseja animá-lo. O caso UC09 - Executar animações da cena, o usuário pressiona o botão de executar no marcador de gravação, executando todas as animações registradas para a cena atual. O caso UC10 - Remover animação, o usuário move, através do marcador cubo, o ícone da animação do marcador de seleção de animações para a lixeira, excluindo a animação selecionada, ou então, move diretamente o objeto anexado à animação para a lixeira, excluindo-a do mesmo jeito. No caso UC11 - Remover objeto da cena, o usuário utiliza o marcador cubo para mover um objeto da cena para lixeira, excluindo-o da mesma. No caso UC12 - Remover cena, o usuário utiliza o marcador cubo para mover o ícone da cena do marcador de seleção de cenas, para a lixeira, excluindo a cena selecionada e todos os seus objetos e animações. Por fim, no caso UC13 - Inspeccionar objeto, o usuário utiliza o marcador de inspeção para visualizar informações de um objeto, cena ou animação, apontando-o para o item desejado.

O Quadro 1 tem como objetivo facilitar a visualização da relação entre os requisitos e os casos de uso da aplicação.

Quadro 1 – Relação entre Requisitos Funcionais e Casos de Uso

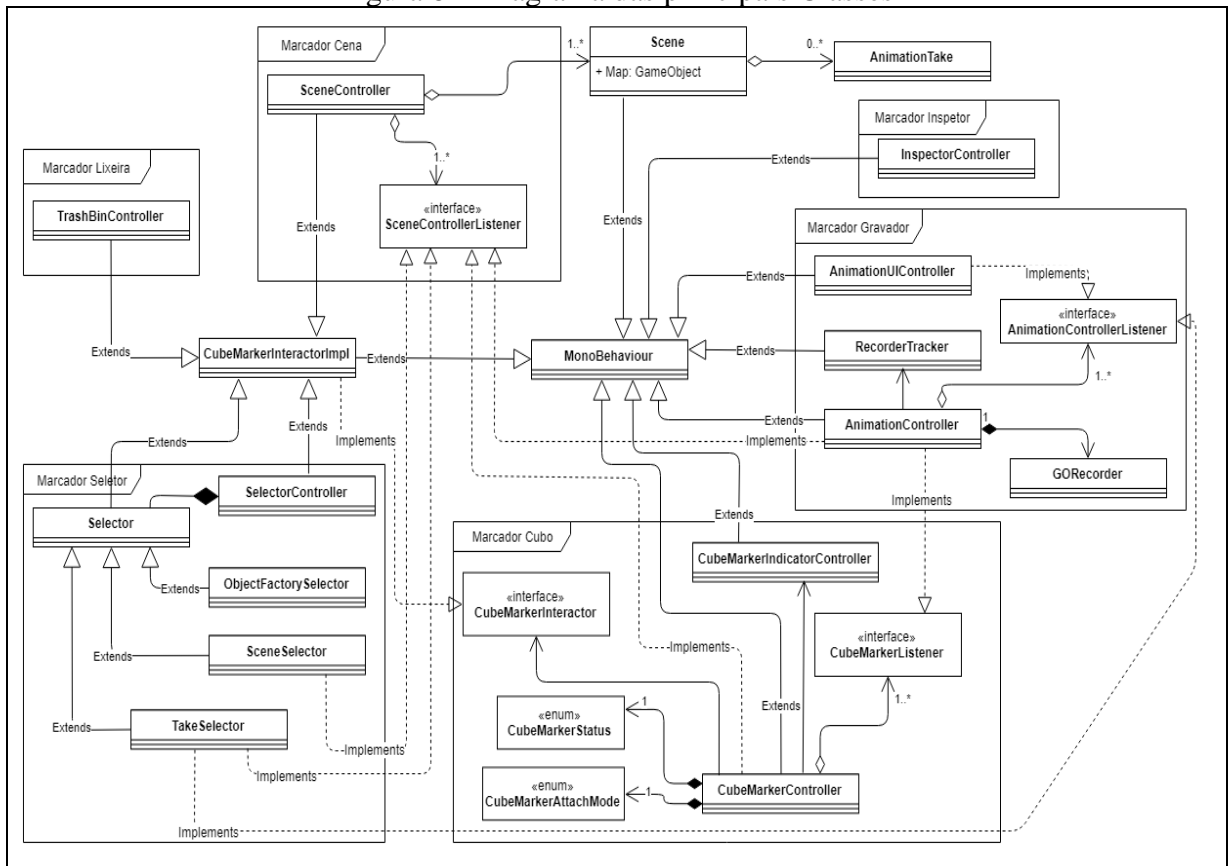
Casos de Uso \ RFs	Casos de Uso												
	UC01	UC02	UC03	UC04	UC05	UC06	UC07	UC08	UC09	UC10	UC11	UC12	UC13
b	X												
c		X											
d			X	X	X	X	X	X	X	X	X	X	X
e			X	X		X						X	
f			X	X	X						X		
g			X	X						X			
h							X						
i								X	X				
j								X	X				
k													X

Fonte: elaborado pelo autor.

3.2.2 Diagrama de classes

A Figura 8 apresenta o diagrama das principais classes do sistema, divididos entre os marcadores do sistema. Optou-se por esconder os campos e métodos para uma melhor visualização das classes.

Figura 8 – Diagrama das principais Classes



Fonte: elaborado pelo autor.

Os *scripts* estão divididos entre os marcadores dos quais fazem parte. Praticamente todos os *scripts* estendem da classe `MonoBehaviour`, isso porque o Unity possui uma estrutura de objetos chamada `GameObject`. Um `GameObject` nada mais é do que um objeto que faz parte do cenário, podendo possuir vários *scripts*, textura, renderizadores e outros. No caso dos *scripts* anexados à um `GameObject`, todos devem estender de `MonoBehaviour`. Faz-se o uso dessa classe, então, para anexar os *scripts* necessários aos `GameObjects`.

O *script* `SceneController`, que é anexado ao `GameObject` do marcador de cena, faz o controle da renderização da cena atual no marcador. Ele também guarda e controla todas as `Scene` do projeto. Uma `Scene`, que também é um `MonoBehaviour` dentro de um `GameObject`, possui uma lista com todos os objetos que compõem o mapa e todas as animações criadas para aquela cena.

O *script* `SelectorController`, responsável pela renderização e controle do marcador seletor, faz uso do padrão de projeto `Strategy` com a classe abstrata `Selector`, para controlar o seletor ativo. Este possui três implementações: o `ObjectFactorySelector`, que disponibiliza uma lista predefinida de objetos que podem ser adicionados em uma cena, o

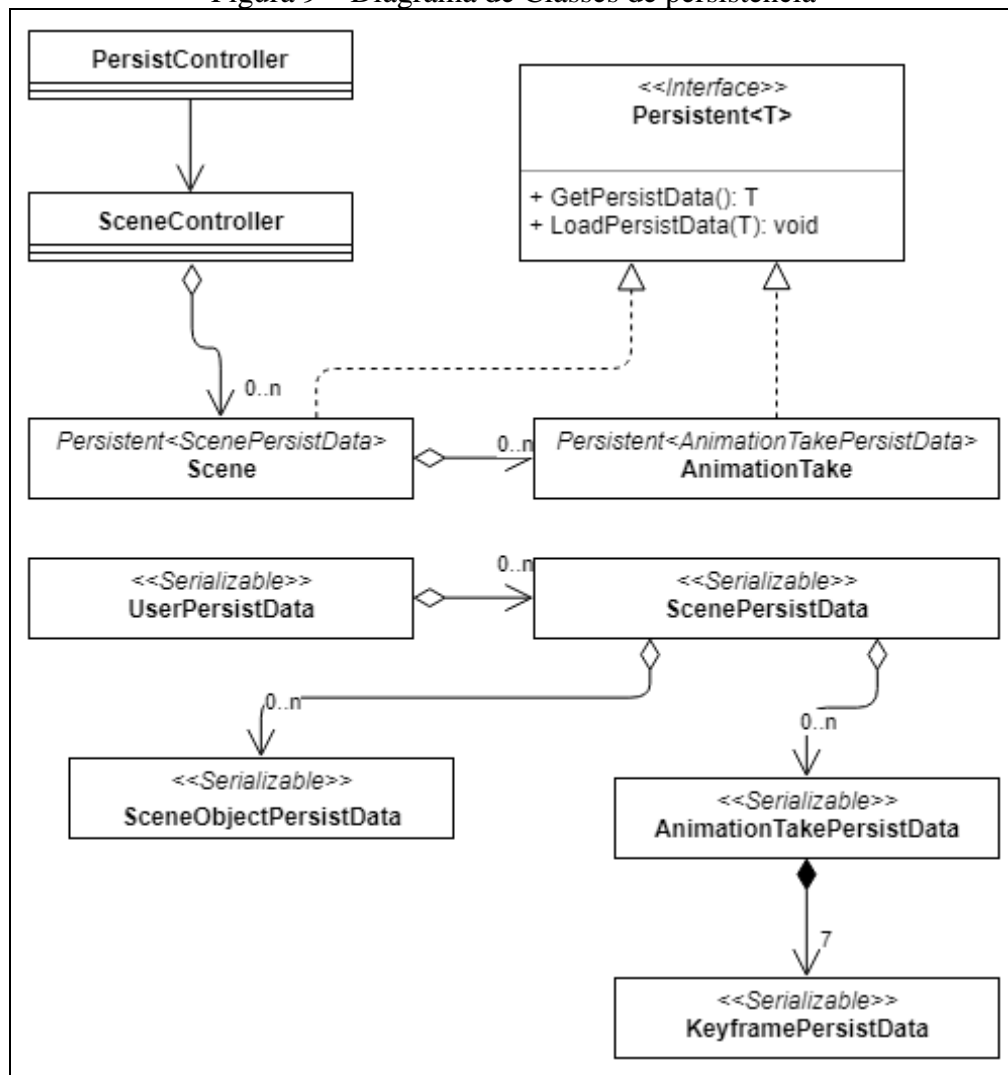
TakeSelector, que deixa o usuário navegar entre as animações criadas para aquela cena e, por último, o SceneSelector, que trata justamente de navegar entre as Scene do projeto.

O marcador cubo faz uso do *script* CubeMarkerController como principal ator, controlando a parte de seleção e movimentação dos objetos. Esse *script* também faz o uso da interface CubeMarkerInteractor, em que cada *script* que implemente esta interface, possui a habilidade de interagir com o marcador cubo. Neste caso, os *scripts* SelectorController, SceneController e TrashBinController, implementam o CubeMarkerInteractor. Isto ocorre, pois, estes marcadores possuem interação direta com o marcador cubo, como exemplo o SceneController, que interage com o cubo para adicionar, mover e remover objetos da cena.

Na parte de gravação, o AnimationController é o principal *script* do marcador gravador. Ele controla a interface do marcador e toda a parte de gravação e execução das animações. Ele trabalha em conjunto com o marcador de cubo para detectar o objeto que está sendo animado e com outras classes como o GOREcorder, que cuida da parte de salvar os *keyframes*, com a posição e rotação do objeto que está sendo gravado, gerando um AnimationClip quando a gravação termina.

Na Figura 9 demonstram-se as classes que fazem parte da persistência de dados da cena, objetos e animações. No Unity não há como serializar os GameObjects ou mesmo os MonoBehaviour, então, para alcançar uma forma de serializar os dados do projeto, criou-se classes C# puras e serializáveis, que guardam as informações necessárias para que se possa instanciar e recuperar os dados de uma sessão anterior, em uma nova sessão.

Figura 9 – Diagrama de Classes de persistência



Fonte: elaborado pelo autor.

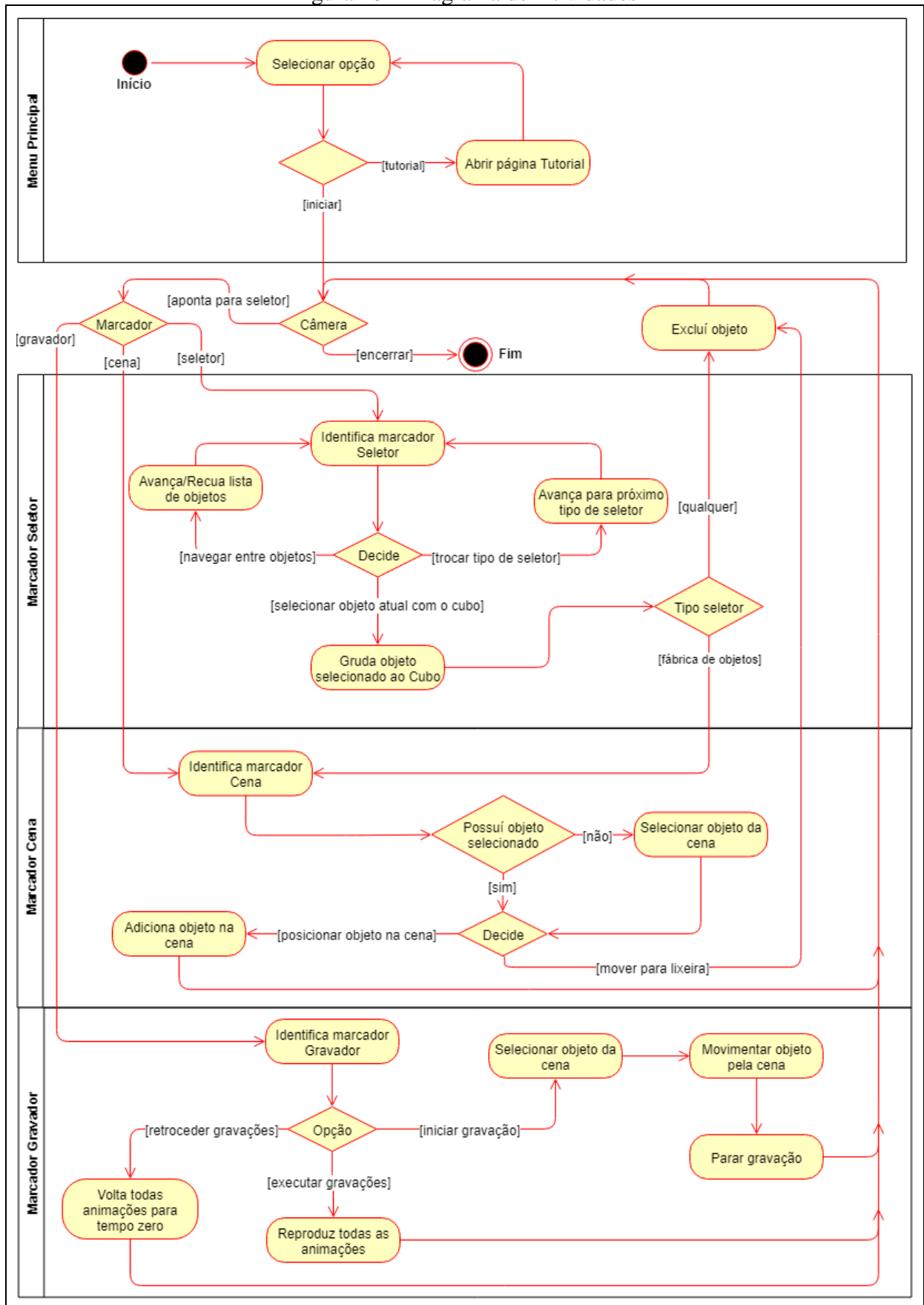
A interface `Persistent<T>` tem como objetivo tornar um `MonoBehaviour`, ou qualquer outra classe que a implemente, em um objeto serializável, implementando um método `GetPersistData` que retorna um objeto do tipo `T`, contendo todas as informações importantes daquela classe para que, futuramente, o método `LoadPersistData` carregue essas informações anteriormente salvas. Os dois principais *scripts* que salvam informações da cena e animações, a classe `Scene` e `AnimationTake`, implementam a interface `Persistent<ScenePersistData>` e `Persistent<AnimationTakePersistData>` respectivamente.

O *script* `PersistController` fica encarregado de coletar todas as informações das interfaces `Persistent<T>` e salvá-las no dispositivo do usuário, devendo também carregar essas informações salvas ao iniciar a aplicação.

3.2.3 Diagrama de atividades

A Figura 10 apresenta o diagrama de atividades das principais funções que o usuário realiza no programa. Aqui demonstra-se o caminho mais comum que os usuários realizam no sistema, como adicionar objetos na cena, movimentá-los e gravar animações. As atividades são em sua maioria controladas pela Interface de Usuário Tangível e o retorno é a exibição dos objetos virtuais na tela do dispositivo do usuário.

Figura 10 – Diagrama de Atividades



Fonte: elaborado pelo autor.

3.3 IMPLEMENTAÇÃO

Nesta seção são mostradas as técnicas e ferramentas utilizadas para o desenvolvimento deste projeto, juntamente com a operacionalidade da implementação.

3.3.1 Técnicas e ferramentas utilizadas

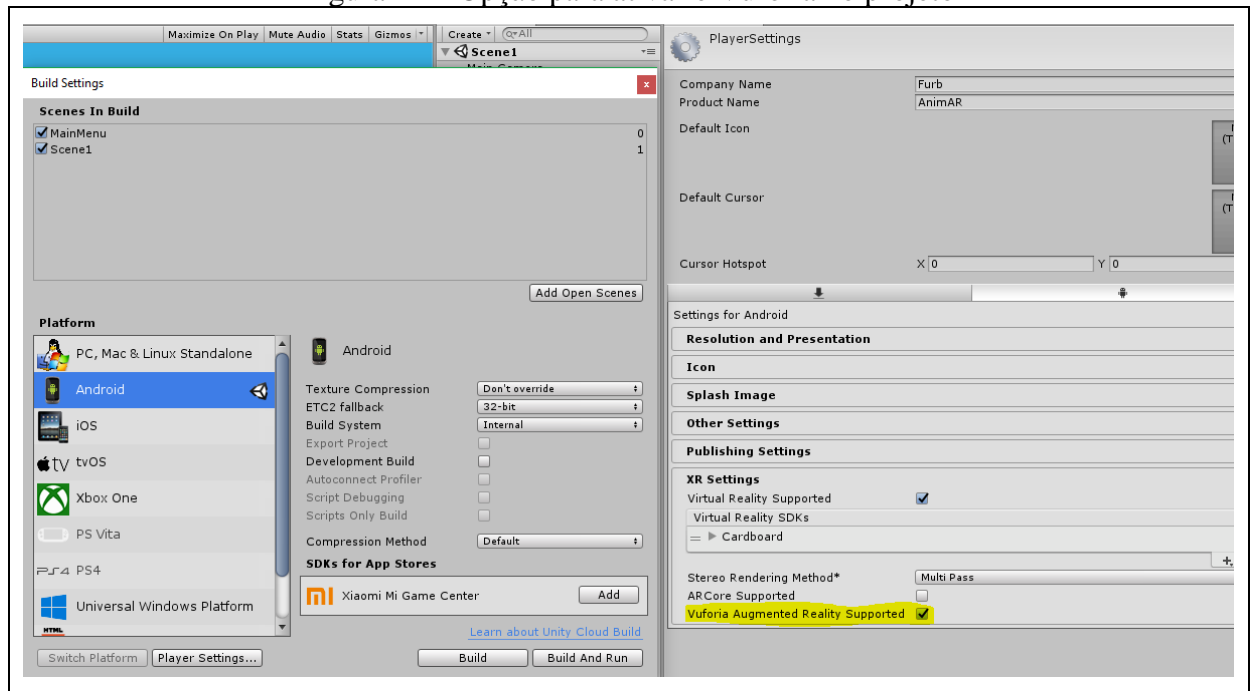
O aplicativo foi desenvolvido utilizando a *engine* gráfica Unity 2017.4.0f1 Personal em conjunto com o Microsoft Visual Studio Ultimate 2013 como editor de código. O Vuforia 7.0.47 foi utilizado como framework de realidade aumentada e já vem embutido com as versões mais recentes do Unity, que é o caso da utilizada para desenvolver este projeto. Para a criação dos marcadores, foi utilizada a ferramenta AR Marker Generator by Brosvision, que gera imagens aleatórias e otimizadas para reconhecimento de marcadores RA, em conjunto com a ferramenta Adobe Photoshop CS6 para edição das mesmas. Segundo Brosvision (2018), as imagens geradas pelo AR Marker Generator podem ser utilizadas e editadas livremente para qualquer uso.

Os testes durante todo o desenvolvimento foram realizados em um computador com processador i5-3570K, 16GB de Memória Ram, Placa de Vídeo MSI GTX 1060 6gb e Webcam Logitech C270 HD, e um celular Galaxy S8 com Android 8, 4 GB de Memória Ram e Câmera de 12 megapixels.

3.3.1.1 Utilização do Vuforia

A utilização do Vuforia no Unity ficou mais fácil a partir da versão 2017.2 do Unity, isto porque a partir desta versão, a SDK Vuforia vem integrada com o framework do Unity (VUFORIA, 2018). Para utilizar então o Unity no projeto, deve ser ativada a opção de suporte ao Vuforia nas configurações do projeto, pelo caminho `File, Build Settings, Player Settings`, expandir a aba `XR Settings` e ativar a opção `Vuforia Augmented Reality Supported` (Figura 11).

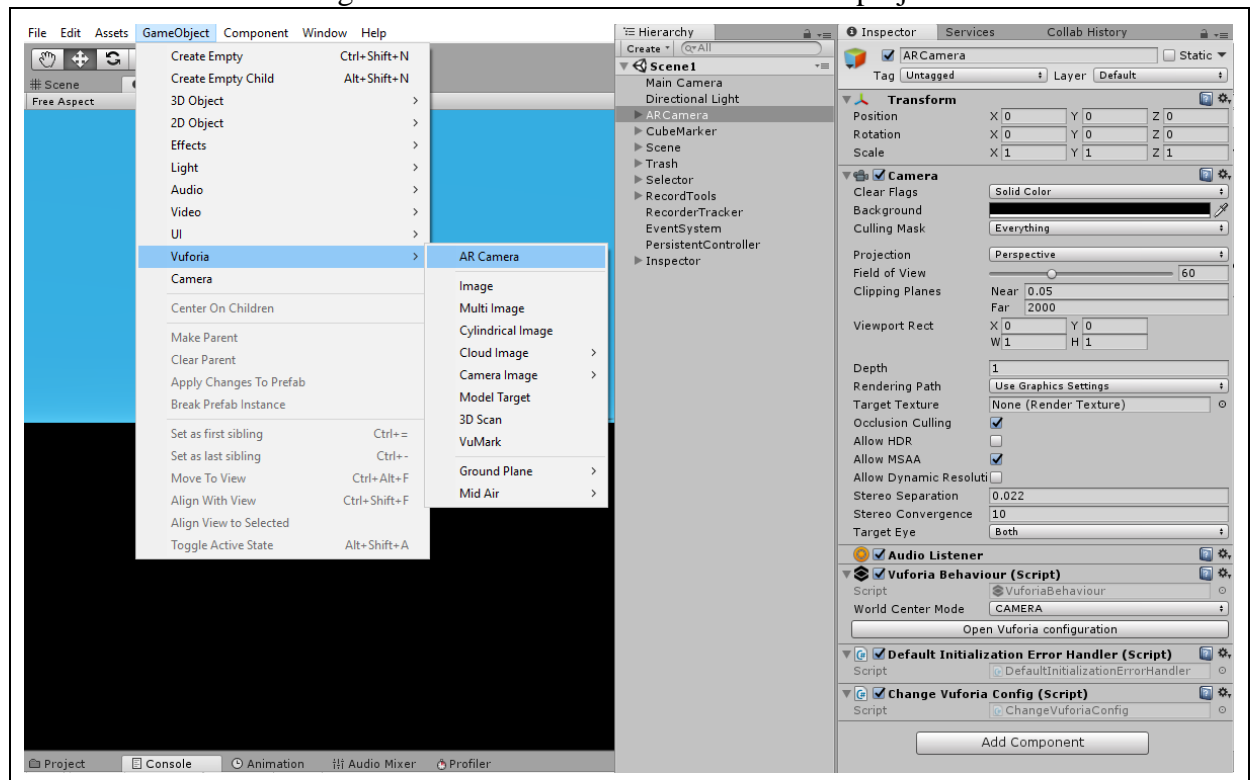
Figura 11 – Opção para ativar o Vuforia no projeto



Fonte: elaborado pelo autor.

A partir do momento em que o Vuforia está ativo, já se pode utilizar os `GameObjects` específicos do Vuforia. O principal `GameObject` que deve ser adicionado à cena é a `ARCamera` (Figura 12). Este `GameObject` possui os *scripts* necessários para a utilização da câmera, detecção de *targets* e renderização dos objetos na câmera.

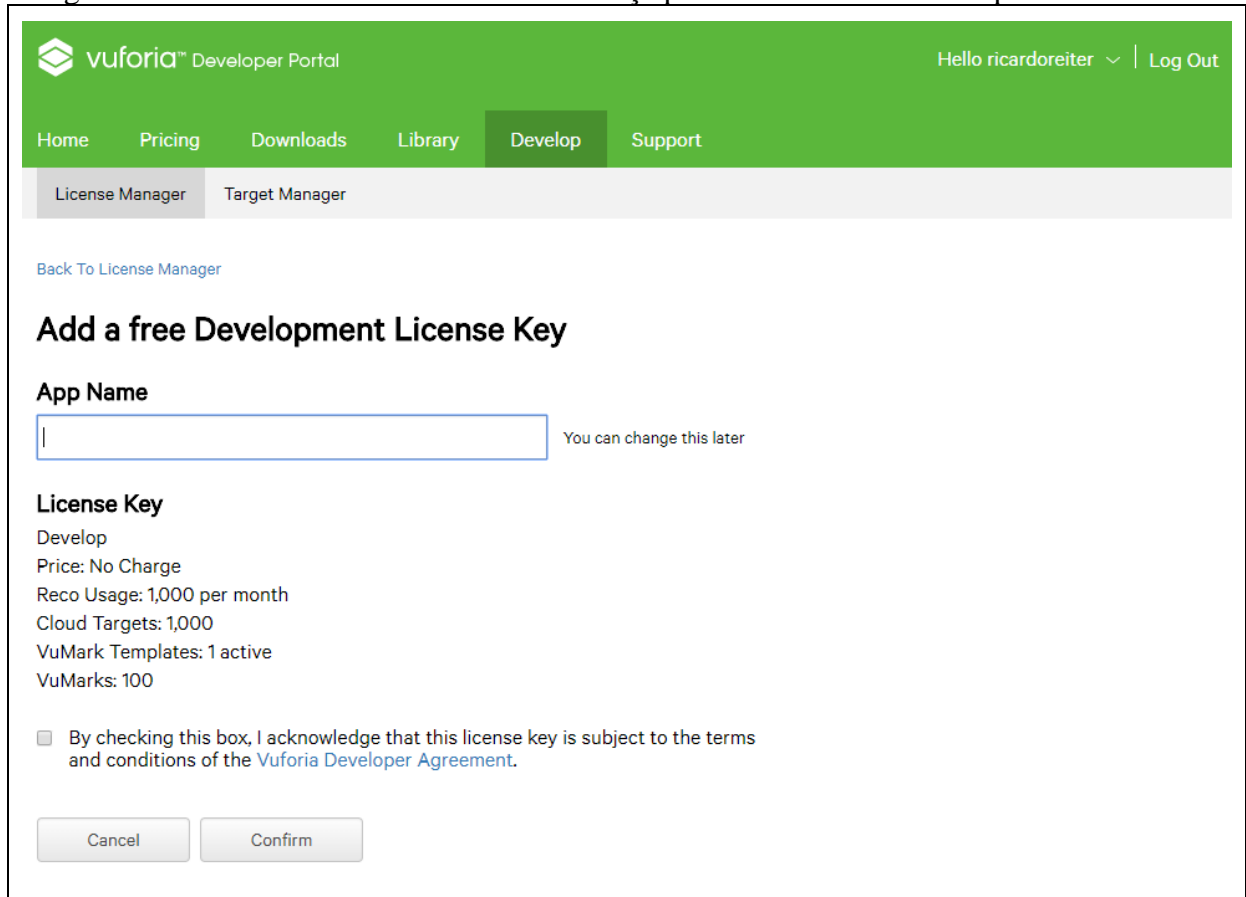
Figura 12 – Adicionando o AR Camera no projeto



Fonte: elaborado pelo autor.

Os últimos passos para a utilização do Vuforia é a obtenção de uma chave de licença do Vuforia e o cadastro dos marcadores no *database*. Para obter uma licença, deve-se cadastrar na página de desenvolvedores do Vuforia e solicitar uma nova chave de desenvolvimento.

Figura 13 – Obtendo uma nova chave de licença para desenvolvimento no portal do Vuforia



The screenshot shows the Vuforia Developer Portal interface. At the top, there is a green header with the Vuforia logo and the text 'vuforia™ Developer Portal'. On the right side of the header, it says 'Hello ricardoreiter' with a dropdown arrow and a 'Log Out' link. Below the header is a navigation menu with links for 'Home', 'Pricing', 'Downloads', 'Library', 'Develop' (which is highlighted), and 'Support'. Underneath the navigation menu, there are two sub-links: 'License Manager' and 'Target Manager'. The main content area has a link 'Back To License Manager' and a heading 'Add a free Development License Key'. Below the heading is a form with the following fields and text:

- App Name:** A text input field with a placeholder character '|'. To its right, it says 'You can change this later'.
- License Key:** A section listing the license details:
 - Develop
 - Price: No Charge
 - Reco Usage: 1,000 per month
 - Cloud Targets: 1,000
 - VuMark Templates: 1 active
 - VuMarks: 100
- By checking this box, I acknowledge that this license key is subject to the terms and conditions of the [Vuforia Developer Agreement](#).
- At the bottom, there are two buttons: 'Cancel' and 'Confirm'.

Fonte: elaborado pelo autor.

Após obtida a chave e adicionada às configurações do Vuforia do projeto, deve-se então criar um *database* de marcadores, cadastrando as imagens associadas aos marcadores que se deseja utilizar no projeto (Figura 14).

Figura 14 – Página para adicionar um marcador ao *database*

The screenshot shows the 'Target Manager' interface for 'AnimAR'. It features a breadcrumb trail 'Target Manager > AnimAR', the title 'AnimAR' with an 'Edit Name' link, and a 'Type: Device' label. Below this is a 'Targets (6)' section with an 'Add Target' button and a 'Download Database (All)' button. A table lists the following targets:

<input type="checkbox"/>	Target Name	Type	Rating	Status	Date Modified
<input type="checkbox"/>	inspector	Single Image	★★★★★	Active	May 09, 2018 00:57
<input type="checkbox"/>	trashbin	Single Image	★★★★★	Active	May 09, 2018 00:55
<input type="checkbox"/>	Scene_1	Single Image	★★★★★	Active	May 09, 2018 00:49
<input type="checkbox"/>	selector	Single Image	★★★★★	Active	May 09, 2018 00:44
<input type="checkbox"/>	recorder	Single Image	★★★★★	Active	May 09, 2018 00:44
<input type="checkbox"/>	CubeMarker	Cuboid	n/a	Active	May 09, 2018 00:26

Fonte: elaborado pelo autor.

Existem quatro tipos de *targets* que podem ser adicionadas ao *database* para utilizar no Vuforia. Single Image é uma simples imagem plana, o Cuboid em que se pode utilizar um cuboide para detecção de marcadores, como o marcador cubo utilizado neste projeto, Cylinder onde objetos cilíndricos podem ser detectados e, por último o 3D Object onde é possível utilizar objetos reais mais complexos, com várias faces (Figura 15).

Figura 15 – Página para adicionar um novo marcador

Add Target

Type:

Single Image Cuboid Cylinder 3D Object

File:

.jpg or .png (max file 2mb)

Width:

Enter the width of your target in scene units. The size of the target should be on the same scale as your augmented virtual content. Vuforia uses meters as the default unit scale. The target's height will be calculated when you upload your image.

Name:

Name must be unique to a database. When a target is detected in your application, this will be reported in the API.

Fonte: elaborado pelo autor.

Com os marcadores devidamente registrados, deve ser realizado o *download* do database no formato `unitypackage` e importado dentro do Unity. Após importado, pode-se adicionar `GameObjects` do tipo `Image Target` no projeto e a detecção daquele marcador estará configurada.

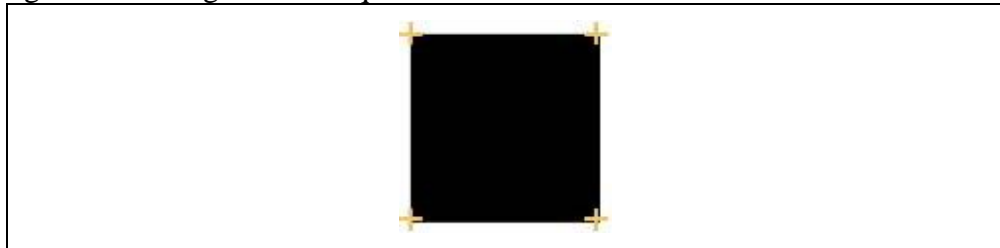
3.3.1.2 Criação dos Marcadores

A identificação dos marcadores no Vuforia se dá ao registrar uma imagem como marcador no *database*, então o Vuforia faz o processamento de imagem necessário da câmera para extrair informações daquela imagem e comparar com os marcadores registrados. Por conta disso, é importante analisar quais aspectos da imagem o Vuforia leva em consideração ao fazer o processamento, para então se obter uma boa identificação dos marcadores ao executar a aplicação.

O Vuforia tenta extrair da imagem, o que eles chamam de *features*, ou, características naturais. Uma característica natural é um detalhe nítido e pontiagudo da imagem, como por exemplo, os quatro cantos de um quadrado preto com o fundo branco (VUFORIA, 2018).

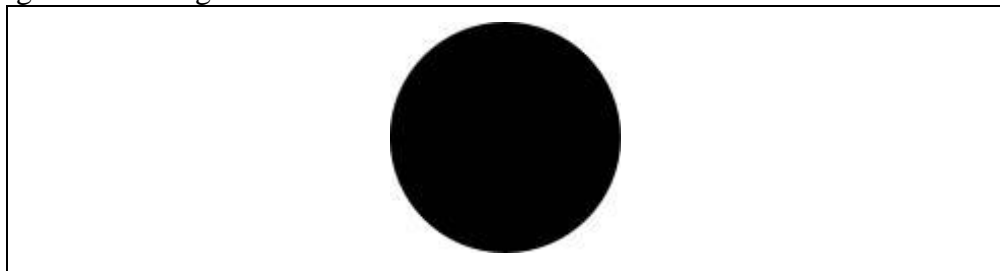
A Figura 16 mostra um exemplo de uma imagem usada como marcador em que o Vuforia identificou os quatro cantos como característica natural. Já a Figura 17 mostra um círculo em que o Vuforia não conseguiu encontrar nenhuma *feature* que ajude a identificação do marcador.

Figura 16 – Imagem de um quadrado com as características naturais destacadas



Fonte: Vuforia (2018).

Figura 17 – Imagem de um círculo sem nenhuma característica natural detectada



Fonte: Vuforia (2018).

Os principais pontos recomendados pelo Vuforia para o desenvolvimento de um bom marcador são: imagem rica em detalhes, como uma foto de um chão de pedras (Figura 18), ou um grupo de pessoas; que contenha bom contraste, regiões escuras e claras bem definidas e iluminadas; sem padrões repetitivos, como um tabuleiro de xadrez (VUFORIA, 2018).

Dependendo da quantidade de *features* encontradas e da não existência de padrões repetitivos, o Vuforia dá uma nota para a imagem, de zero a cinco, sendo zero a imagem não podendo ser reconhecida, e cinco a melhor imagem que se pode ter como marcador.

O tamanho do marcador também influencia no reconhecimento e identificação dos marcadores e, apesar de o tamanho ideal depender de foco de câmera, iluminação e da própria nota que a imagem recebeu, há uma recomendação do Vuforia para um tamanho mínimo ideal. A fórmula que deve ser utilizada é, dividir a distância da câmera até onde o marcador geralmente será utilizado por 10 (VUFORIA, 2018). Então, se um marcador pretende ser utilizado a uma distância de um metro da câmera, este deve possuir pelo menos 10 centímetros.

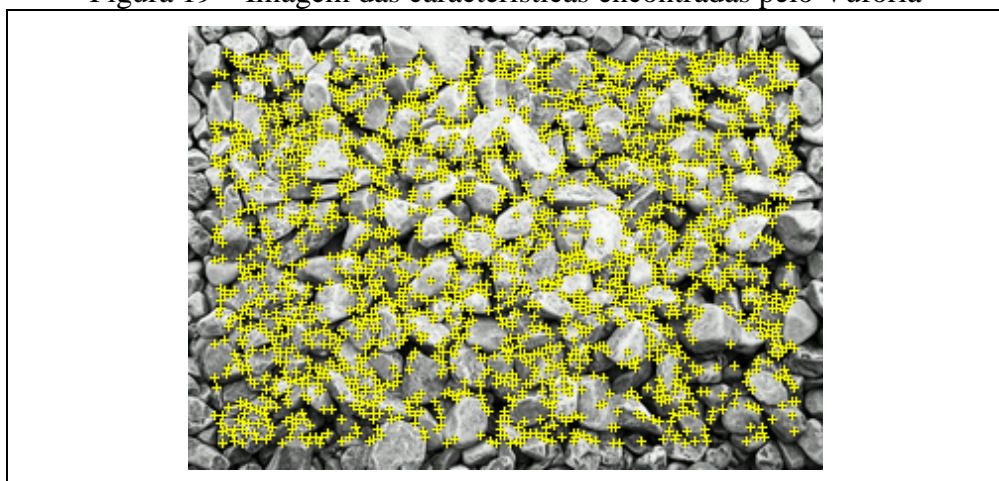
Figura 18 – Imagem de um chão de pedras usada como marcador



Fonte: Vuforia (2018).

A Figura 18 mostra uma imagem com as características descritas para um bom marcador. O resultado das *features* encontradas pelo Vuforia é mostrado na Figura 19, gerando um bom marcador de nota cinco.

Figura 19 – Imagem das características encontradas pelo Vuforia



Fonte: Vuforia (2018).

Inicialmente, tentou-se buscar fotos reais de pedras para a criação dos marcadores do Vuforia, porém, por falta de encontrar imagens livres de direitos autorais, acabou sendo encontrado uma solução melhor, utilizando um gerador de marcadores. O AR Marker

Generator by Brosvision, consegue gerar imagens randômicas embaralhando linhas, triângulos e quadrângulos, formando assim, uma imagem otimizada para o uso no Vuforia.

Figura 20 – Screenshot da ferramenta AR Marker Generator



Fonte: elaborado pelo autor.

Para os marcadores do AnimAR, as imagens foram geradas pela ferramenta geradora de marcadores e, então, editadas no Adobe Photoshop CS6. No Photoshop as imagens foram cortadas, transformadas em tons de cinza e adicionadas informações relevantes para o usuário

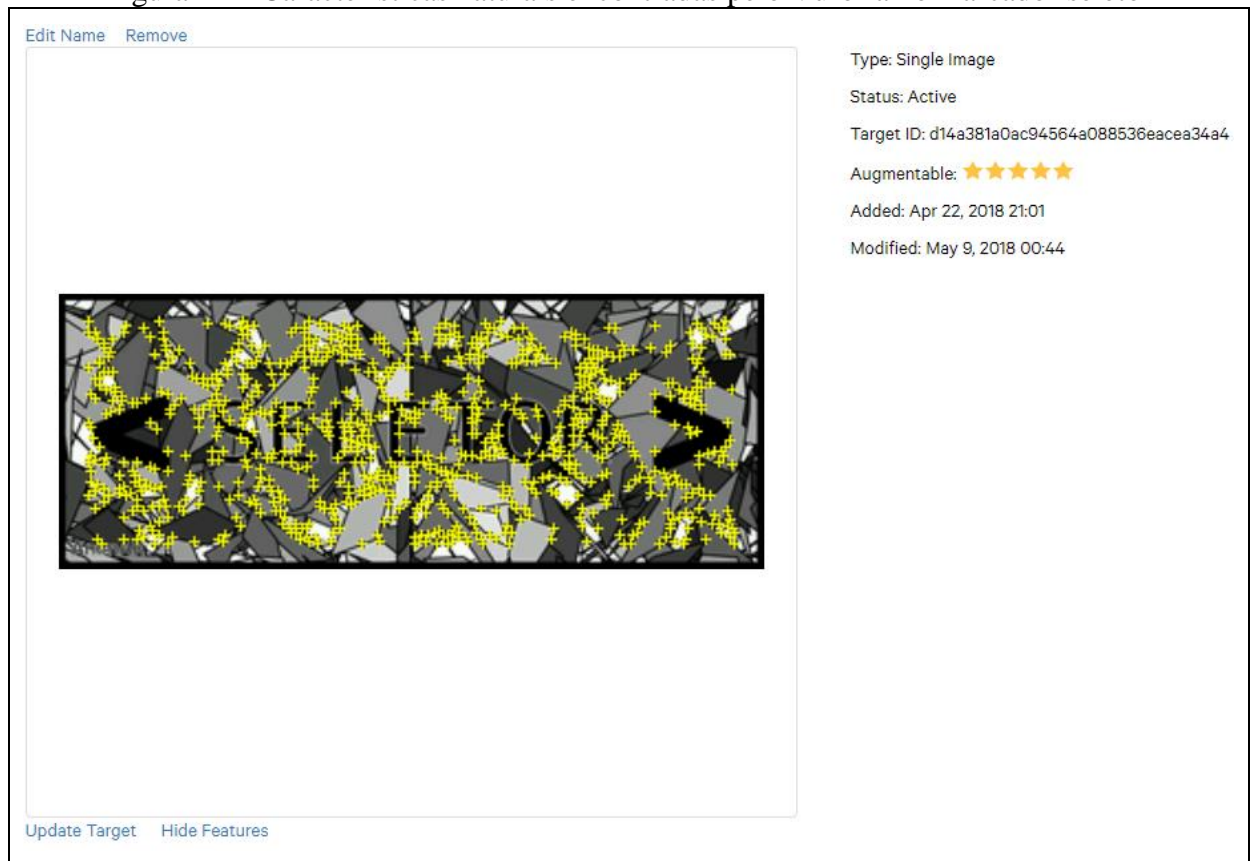
para a identificação dos marcadores, como o nome do marcador. A Figura 21 mostra o marcador criado para o seletor e a Figura 22 mostra as características naturais encontradas pelo Vuforia e sua respectiva nota de cinco estrelas.

Figura 21 – Imagem utilizada para o marcador Seletor



Fonte: elaborado pelo autor.

Figura 22 – Características naturais encontradas pelo Vuforia no marcador seletor

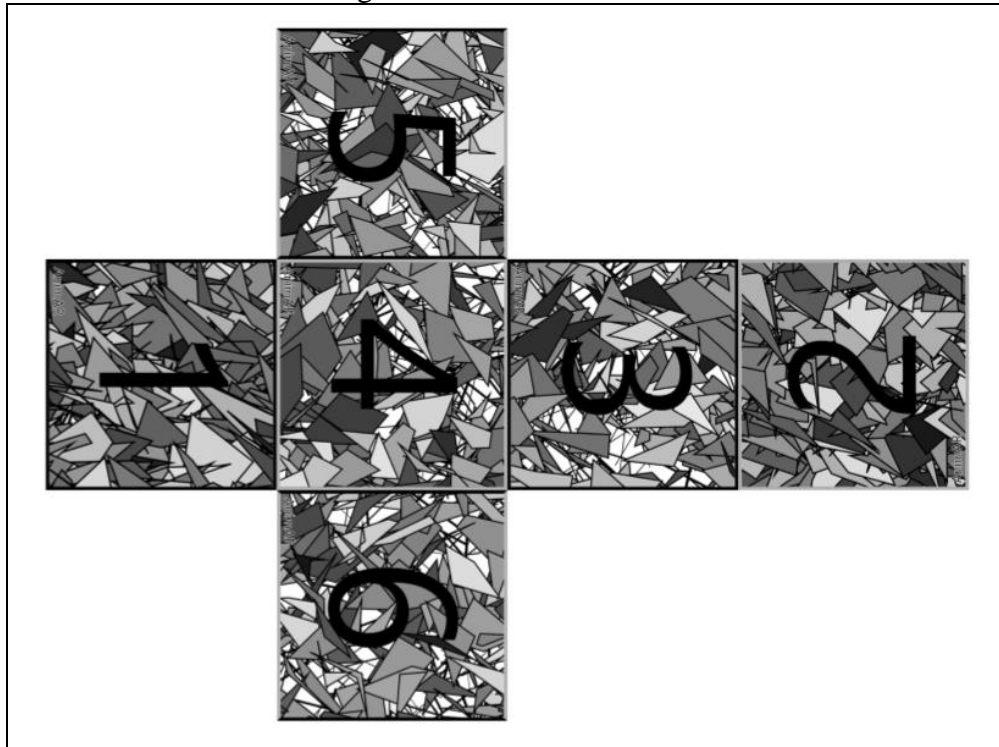


Fonte: elaborado pelo autor.

Neste trabalho, foram desenvolvidas 11 imagens, sendo cinco imagens simples de uma face e seis imagens para as seis faces do marcador cubo. Levando em consideração que o usuário utilizará a aplicação a uma distância média aproximada de 50 centímetros, os

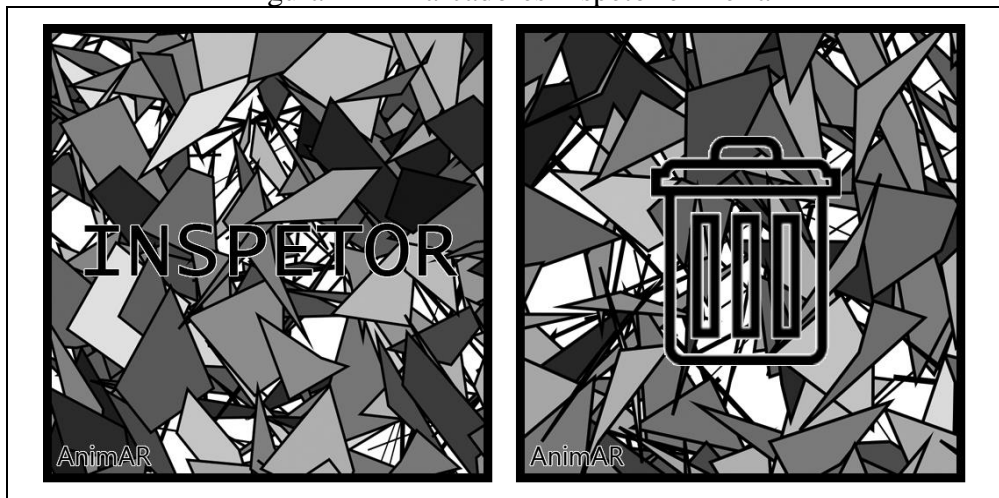
marcadores foram dimensionados a possuírem no mínimo cinco centímetros de altura e largura. O marcador cubo, que segue as mesmas regras das imagens simples, porém com seis faces, foi dimensionado no tamanho 5x5x5 centímetros, e suas faces estão representadas na Figura 23. Os marcadores seletor e gravador, possuem 17 centímetros de largura e 6,5 de altura. Os marcadores inspetor e lixeira (Figura 24), possuem 6x6 centímetros de dimensão. Por fim, o marcador cena, possui o tamanho de 15x15 centímetros.

Figura 23 – Marcador cubo



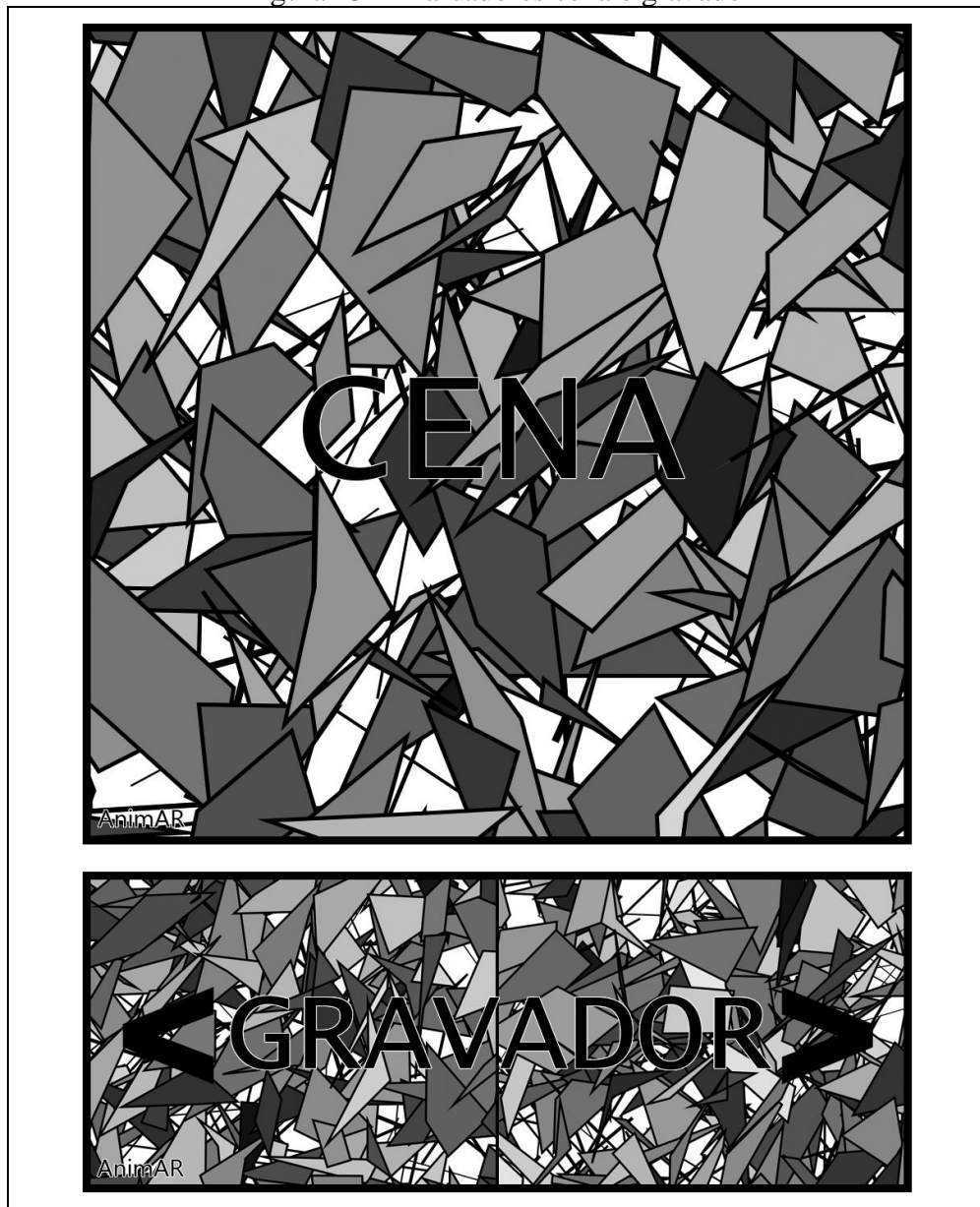
Fonte: elaborado pelo autor.

Figura 24 – Marcadores inspetor e lixeira



Fonte: elaborado pelo autor.

Figura 25 – Marcadores cena e gravador



Fonte: elaborado pelo autor.

3.3.1.3 Ferramenta

Um projeto Unity consiste de alguns conceitos próprios que são importantes destacar. Um projeto consiste de várias *scene*, que é uma cena da sua aplicação em si, como a cena do menu, a cena de um *level* do jogo, entre outros (não confundir com o marcador de cena do AnimAR). Uma *scene* consiste de vários *GameObjects* (GO) organizados em uma árvore hierárquica, que trazem vida à cena. Um GO pode ser um simples objeto gráfico como um boneco, uma árvore, o chão, ou então pode ser um objeto puramente com o objetivo de se fazer algum processamento. No GO, pode-se adicionar *scripts* que executarão enquanto o GO estiver ativo, podendo dar vida ao mesmo, sendo desta forma que se constrói uma aplicação

Unity. Outro conceito importante é o `prefab`, que nada mais é do que um `GameObject` pré-moldado que pode ser instanciado várias vezes dentro das `scenes`, realizando uma cópia do mesmo.

Com estes conceitos compreendidos, na Figura 26 tem-se representada a árvore hierárquica dos `GameObjects` da principal `scene` da aplicação, onde as principais funcionalidades existem. Objetos irrelevantes para o entendimento da implementação foram escondidos para facilitar a visualização e entendimento.

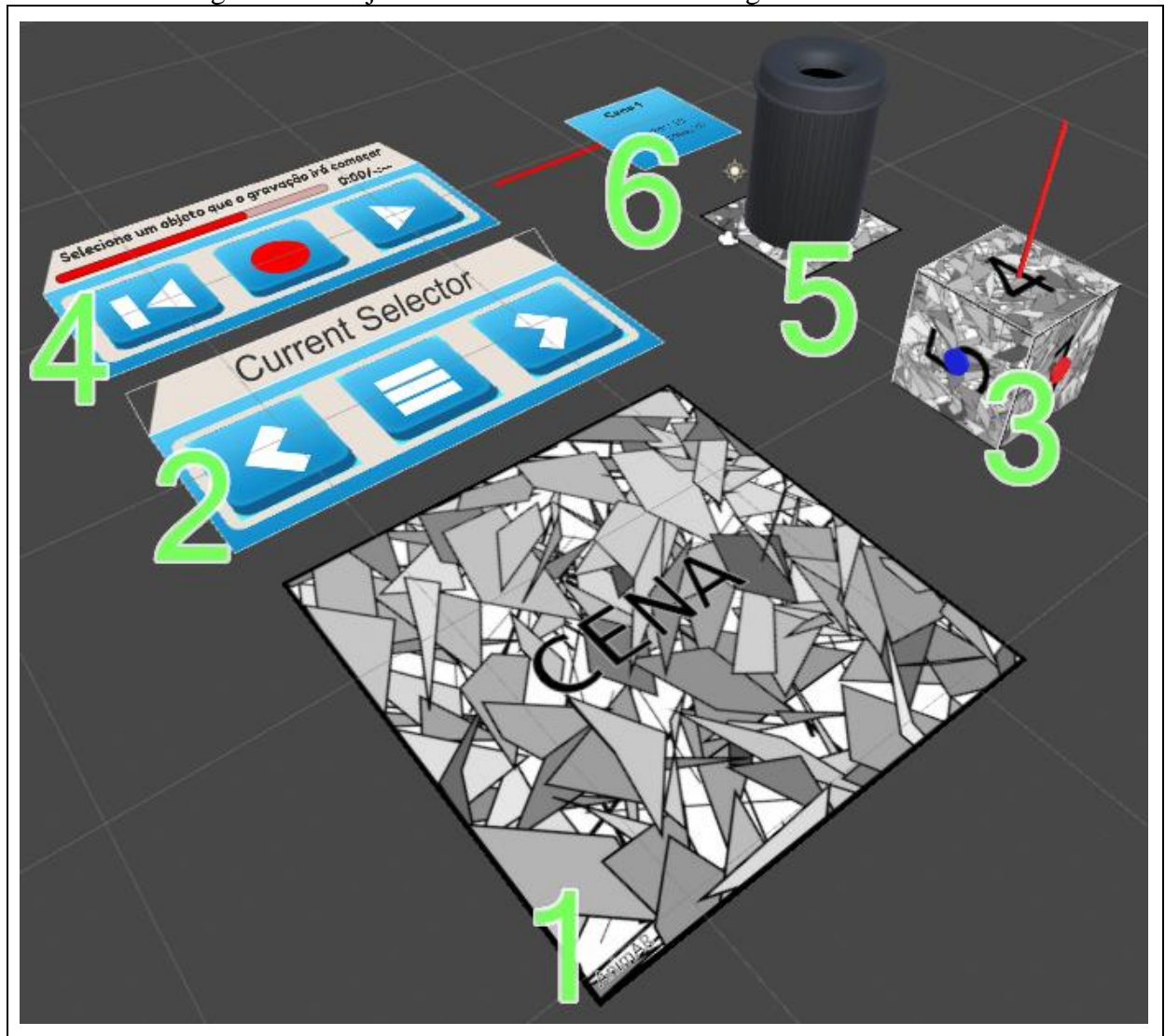
Figura 26 – Árvore hierárquica da principal `scene` do AnimAR



Fonte: elaborado pelo autor.

A Figura 27 mostra a representação gráfica dos `GOs` que fazem parte da realidade aumentada na `scene`. O item 1, que representa o `GO Scene` da Figura 26, é o marcador de cena. Este marcador tem como objetivo renderizar a cena desenvolvida pelo usuário, sendo o principal marcador onde o usuário pode manipular os objetos virtuais, gravar e reproduzir suas animações. O `script` que controla o marcador é o `SceneController` e está anexado ao `GO Scene`, sendo o seu objetivo manter uma lista de cenas do projeto, renderizar a cena atual e trabalhar em conjunto com o marcador cubo para adicionar e remover objetos da cena atual.

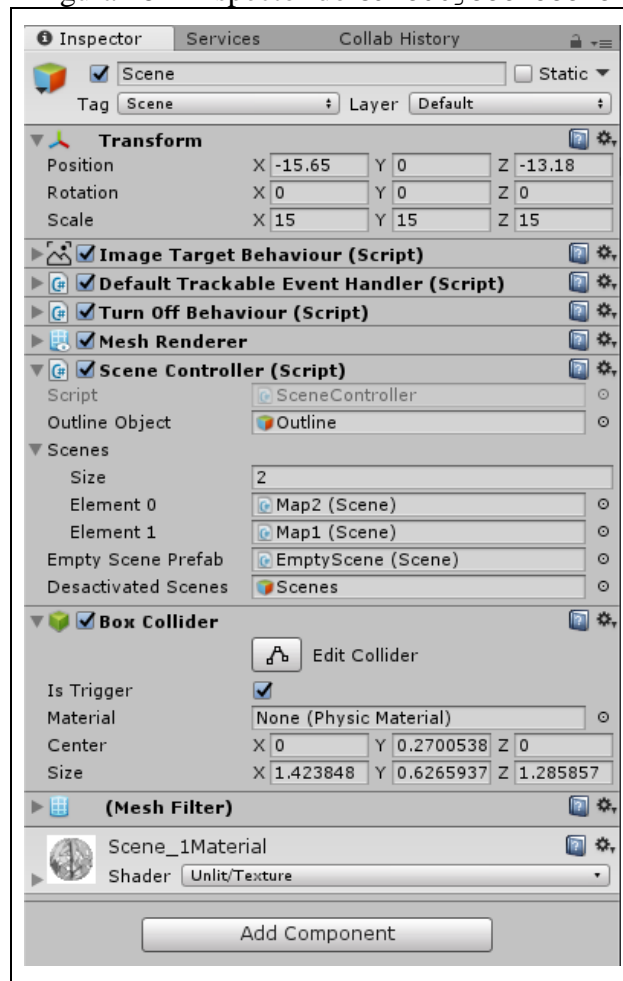
Figura 27 – Objetos de realidade aumentada organizados na scene



Fonte: elaborado pelo autor.

A Figura 28 mostra os atributos que compõem o `GameObject Scene`. `Transform` é um item presente em todo e qualquer `GO` do Unity e representa a matriz de transformação daquele objeto na cena, ou seja, sua translação, rotação e escala. `Image Target Behaviour`, `Default Trackable Event Handler` e `Turn Off Behaviour` são objetos próprios do Vuforia, que atuam para identificação e renderização dos marcadores. `Mesh Render` é um objeto do Unity que renderiza na cena uma *mesh*, ou malha gráfica. Neste caso, o objeto gráfico sendo renderizado é o marcador, item 1 mostrado na Figura 27, e que serve de auxílio durante o desenvolvimento. O `SceneController`, cujas funcionalidades já foram descritas, trabalha em conjunto com um `Box Collider` para realizar a integração com o marcador cubo. Quando o usuário entra dentro da área do `Box Collider` do objeto `Scene` com o marcador cubo, que também possui um `Collider`, a *script* detecta então que o usuário deseja adicionar ou remover algum objeto da cena.

Figura 28 – Inspector do GameObject Scene



Fonte: elaborado pelo autor.

O Quadro 2 mostra um trecho do código do SceneController que faz a integração com o cubo. Os métodos descritos são abstratos na classe CubeMarkerInteractorImpl e implementados pela classe SceneController. O objetivo é que a classe que os implementa, dizer como se deve comportar o objeto quando o marcador cubo o adiciona e remove da cena.

Quadro 2 – Métodos de integração com o cubo da classe SceneController

```

...
59 public override bool CanReceiveObject(MovableObject obj) {
60     return obj.type == MovableObject.TYPE.SCENE_OBJECT;
61 }
62
63 public override void ObjectReceived(MovableObject obj) {
64     obj.transform.parent = GetCurrentScene().Map.transform;
65     VuforiaUtils.EnableTargetObject(obj.gameObject);
66     PersistController.Instance.PersistEverything();
67 }
68
69 public override void ObjectRemoved(MovableObject obj) {
70 }
...

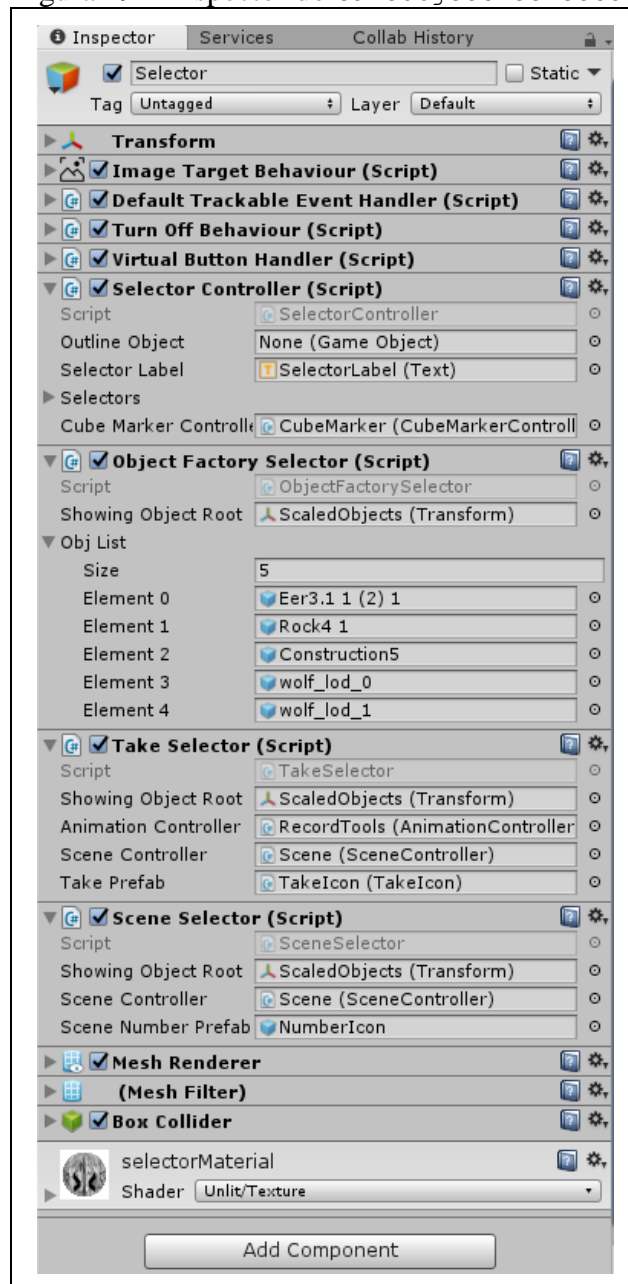
```

Fonte: elaborado pelo autor.

O método `CanReceiveObject` verifica se o objeto é aceitável dentro da cena atual, neste caso, o `SceneController` verifica se o tipo de objeto é um objeto de cena, como por exemplo, um boneco. `ObjectReceived` muda o `parent` do objeto recebido para a cena atual, isso significa mudar a posição do objeto recebido na árvore hierárquica da cena, para dentro do `GO` da cena atual. `GameObjects` sempre possuirão sua posição relativa ao seu `parent` de primeiro grau, significando que ele irá se mover sempre junto ao seu `parent`. Não há nada na implementação do `ObjectRemoved`, pois não é necessário realizar nenhuma tarefa no `SceneController` para isso, levando em consideração que o marcador cubo que irá modificar o `parent` do objeto removido.

O item 2, apresentado na Figura 27, é o marcador seletor, representado pelo `GameObject Selector` (Figura 29) da árvore hierárquica da `scene`. O marcador seletor possui dois botões para navegar entre a lista de seleção, e um botão para trocar o tipo de seletor. No projeto, foram desenvolvidos três tipos de seletor. O primeiro é a Fábrica de Objetos, controlado pelo *script* `ObjectFactorySelector`, tendo como objetivo disponibilizar uma lista de objetos predefinidos que o usuário pode adicionar às suas cenas. Outro tipo de seletor é o de seleção de animação, controlado pelo *script* `TakeSelector`. Possui o objetivo de navegar entre as animações criadas para aquela cena, visualizando o objeto que foi animado e sendo possível mover a animação para lixeira, excluindo-a. O último seletor é o de cena, sendo gerenciado pelo *script* `SceneSelector`. Este, permite navegar entre as cenas do projeto, criar e excluir.

Figura 29 – Inspector do GameObject Selector



Fonte: elaborado pelo autor.

A classe `SelectorController` utiliza o padrão de projeto `Strategy` para gerenciar o seletor ativo. Possuindo uma lista de `Selector` e o `Selector` ativo, ela repassa os métodos de navegação entre os objetos do seletor e de interação com o marcador cubo para o `Selector` atual. O Quadro 3 apresenta as principais partes da implementação do `SelectorController` e do padrão de projeto `Strategy`.

Quadro 3 – Trecho de código da implementação da classe SelectorController

```

...
10 public class SelectorController : CubeMarkerInteractorImpl,
ITrackableEventHandler, VirtualButtonListener {
...
13     public Selector[] Selectors;
...
17     private Selector currentSelector;
...
30     private void ChangeSelector(int index) {
31         currentSelectorIndex = Math.Abs(index % Selectors.Length);
32         CubeMarkerController.ResetAttached();
33         if (currentSelector) {
34             currentSelector.Desactive();
35         }
36         currentSelector = Selectors[currentSelectorIndex];
37         currentSelector.Active();
38         SelectorLabel.text = currentSelector.GetLabel();
39     }
40
41     public override bool CanReceiveObject(MovableObject obj) {
42         return currentSelector.CanReceiveObject(obj);
43     }
44
45     public override void ObjectReceived(MovableObject obj) {
46         currentSelector.ObjectReceived(obj);
47     }
48
49     public override void ObjectRemoved(MovableObject obj) {
50         currentSelector.ObjectRemoved(obj);
51     }
...
64     public void ButtonPressed(VirtualButtonBehaviour vb) {
65         switch (vb.VirtualButtonName) {
66             case "Next":
67                 currentSelector.Next();
68                 CubeMarkerController.ResetAttached();
69                 break;
70             case "Prev":
71                 currentSelector.Prev();
72                 CubeMarkerController.ResetAttached();
73                 break;
74             case "ChangeSelector":
75                 ChangeSelector(currentSelectorIndex + 1);
76                 break;
77         }
78     }
79
80 }
...

```

Fonte: elaborado pelo autor.

O método `ChangeSelector` muda o seletor ativo e chama os eventos de `Desactive` e `Active` dos respectivos seletores que foram desativados e ativados. Os métodos abstratos da classe `CubeMarkerInteractorImpl`, `CanReceiveObject`, `ObjectReceived` e `ObjectRemoved` são repassados para o seletor ativo, isto porque é o Selector atual que define o que deve acontecer quando há interação com o marcador cubo. Quando o usuário

pressiona os botões de avançar e/ou voltar no marcador seletor, são chamados então os métodos `Next` e/ou `Prev` do `Selector` ativo.

A implementação do `Selector ObjectFactorySelector` utiliza a lista de objetos do *script* `PersistentPrefabList`, representado pelo `GameObject PersistentController` na Figura 26, para mostrar os objetos disponíveis que pode se adicionar à cena. A lista de objetos da classe `PersistentPrefabList` também é utilizada para o instanciamento dos GOs das cenas anteriormente salvas, durante o carregamento do sistema. O Quadro 4 mostra trechos de código da parte que faz o carregamento dos objetos da `PersistentPrefabList`, e a interação com o marcador cubo, criando uma cópia do objeto atual mostrado no `ObjectFactorySelector`, quando o usuário o remove para adicionar à cena.

Quadro 4 – Trecho de código da implementação da classe `ObjectFactorySelector`

```

...
16 void Start() {
17     LinkedList<GameObject> newList = new LinkedList<GameObject>();
18     foreach (var prefabItem in PersistentPrefabList.Instance.list) {
19         var movable = prefabItem.GetComponent<MovableObject>();
20         if (movable && movable.type ==
MovableObject.TYPE.SCENE_OBJECT) {
21             newList.AddLast(movable.gameObject);
22         }
23     }
24     objList = newList.ToArray<GameObject>();
25     ChangeIndex(0);
26 }
...
65 public override void ObjectRemoved(MovableObject obj) {
66     if (currentObject == obj.gameObject) {
67         NewCurrentObject(obj.gameObject);
68     }
69 }
...
75 public void NewCurrentObject(GameObject objToCopy) {
76     var outliner = objToCopy.GetComponent<ObjectOutliner>();
77     if (outliner) {
78         outliner.SetEnabled(false);
79     }
80
81     currentObject = GameObject.Instantiate(objToCopy,
showingObjectRoot);
82     currentObject.transform.localPosition = new Vector3(0, 0, 0);
83
84     if (outliner) {
85         outliner.SetEnabled(true);
86     }
87 }
...

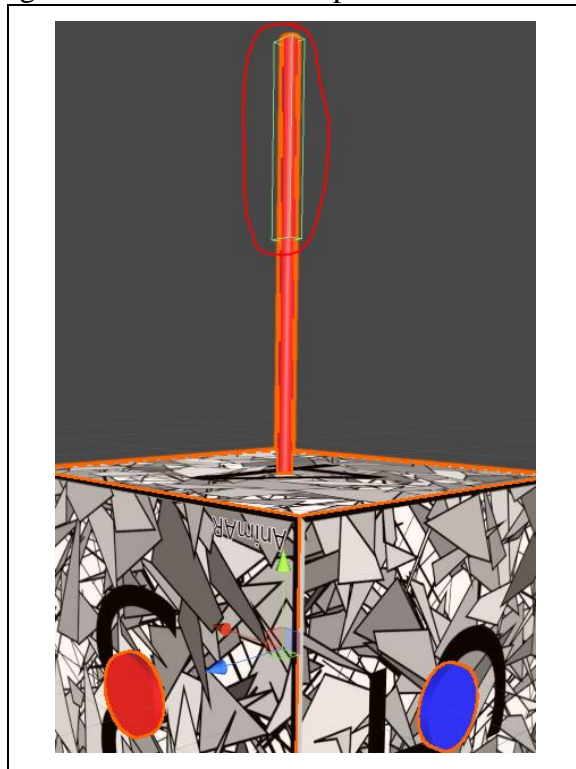
```

Fonte: elaborado pelo autor.

O item 3 da Figura 27 é a representação gráfica do GO `CubeMarker` da árvore hierárquica, que representa o marcador cubo. Além do *script* `CubeMarkerController` que controla o cubo, o GO possui também um `BoxCollider` e um `Rigidbody` para identificar

quando o cubo está tentando selecionar algum objeto. O `BoxCollider` (Figura 30) ativa o evento `OnTriggerEnter` no *script* `CubeMarkerController` para identificar quando há colisão com outro objeto do tipo `Collider`, porém, para que esta colisão seja detectada, é necessário que um dos `GOs` envolvidos na colisão possua um `Rigidbody`. Neste caso, o que mais fez sentido foi adicionar o `Rigidbody` ao marcador cubo, ao invés de adicionar em todos os possíveis objetos que colidirão com ele. Um `Rigidbody` tem a função de adicionar física à um `GameObject`, como massa e gravidade. No caso do marcador cubo, foi removida a gravidade do objeto, para que a *engine* de física não atue e conflite com o posicionamento do objeto feito pelo `Vuforia`.

Figura 30 – `BoxCollider` posicionado no cubo



Fonte: elaborado pelo autor.

Como pode ser observado na Figura 30, o `BoxCollider` fica posicionado na ponta do indicador do cubo e ativa o evento `OnTriggerEnter` no *script* `CubeMarkerController` quando colide com outros `Colliders`, conforme pode ser observado no Quadro 5.

Quadro 5 – Trecho de código da implementação da classe CubeMarkerController

```

...
42 void FixedUpdate() {
43     if (currentStatus == CubeMarkerStatus.MARKER_OVER_OBJECT ||
currentStatus == CubeMarkerStatus.OBJECT_ATTACHED) {
44         if (Math.Abs(Vector3.Distance(lastPos, transform.position)) >
movementTolerance) {
45             currentTime = secondsToHold;
46         } else {
47             currentTime -= Time.fixedDeltaTime;
48             if (currentTime <= 0) {
49                 if (currentStatus == CubeMarkerStatus.MARKER_OVER_OBJECT) {
50                     objectMarkerOver.currentInteractor.ObjectRemoved(objectMarkerOver);
51                     attachedObject = objectMarkerOver;
52                     attachedObjectInitialPos =
attachedObject.transform.localPosition;
53                     attachedObjectInitialRot =
attachedObject.transform.localRotation;
54                     if (attachMode == CubeMarkerAttachMode.RECORD_MODE) {
55                         indicatorDot.position = attachedObject.transform.position;
56                         indicatorDot.rotation = attachedObject.transform.rotation;
57                     } else {
58                         attachedObject.transform.parent = this.transform;
59                     }
60                     objectMarkerOver = null;
61                     NotifyObjectAttached(attachedObject);
62                 } else if (attachedObject && interactor != null && attachMode
== CubeMarkerAttachMode.NORMAL) {
63                     if (interactor.CanReceiveObject(attachedObject)) {
64                         attachedObject.currentInteractor = interactor;
65                         interactor.ObjectReceived(attachedObject);
66                         NotifyObjectDetached(attachedObject);
67                         attachedObject = null;
68                     }
69                 }
70                 UpdateStatus();
71                 currentTime = secondsToHold;
72             }
73         }
74     }
75     lastPos = transform.position;
76 }
77
78 void OnTriggerEnter(Collider other) {
79     var movable = other.GetComponent<MovableObject>();
80     var newInteractor = other.GetComponent<CubeMarkerInteractor>();
81
82     if (movable != null) {
83         if (currentStatus == CubeMarkerStatus.NOP) {
84             newInteractor = movable.currentInteractor;
85             objectMarkerOver = movable;
86             if (objectMarkerOver.outliner) {
87                 objectMarkerOver.outliner.SetColor(Color.red);
88                 objectMarkerOver.outliner.SetEnabled(true);
89             }
90             currentTime = secondsToHold;
91         }
92     }
93
94     if (newInteractor != null && currentStatus !=
CubeMarkerStatus.MARKER_OVER_OBJECT) {

```

```

95     if (interactor != null) interactor.OnCubeMarkerExit();
96         interactor = newInteractor;
97         interactor.OnCubeMarkerEnter();
98
99     }
100     UpdateStatus();
101 }
...

```

Fonte: elaborado pelo autor.

Conforme observado no Quadro 5, o método `OnTriggerEnter` verifica se o objeto colidido com o cubo é um objeto do tipo móvel, `MovableObject`, como objetos da cena, ou do tipo que interage com o cubo, `CubeMarkerInteractor`, como o marcador Cena ou marcador seletor. Caso seja do tipo `MovableObject`, atribuí o objeto então para o atributo `objectMarkerOver`, para futuramente verificar se deve selecionar este objeto ou não. O método `FixedUpdate` é um método do Unity que é chamado pela *engine* a cada N segundos. No caso do AnimAR, é chamado a cada 0,2 segundos, sendo o tempo padrão do Unity que pode ser configurado, se necessário. Na implementação do método na classe `CubeMarkerController`, é verificado se o cubo está sobre algum objeto ou se possui algum objeto já selecionado. Caso positivo, calcula-se então se o usuário está com o marcador cubo parado por alguns segundos, estipulados pelo atributo `secondsToHold`. Caso seja detectada uma movimentação maior que a tolerância definida pelo atributo `movementTolerance`, a contagem é resetada, caso contrário, diminui-se do atributo de contagem de tempo o `Time.fixedDeltaTime`, que é o tempo em segundos que desde a última passagem do método `FixedUpdate`. Desta forma, quando o contador chegar à zero, é considerado então que o marcador Cubo ficou imóvel pelo tempo necessário, e uma ação é tomada.

A ação a ser tomada depende do estado atual do marcador cubo. No caso de o cubo não possuir nenhum objeto selecionado e estiver com o indicador sobre um objeto (atributo `objectMakerOver`), remove o objeto do `currentInteractor` e anexa ao marcador cubo, levando em consideração que todos os objetos possíveis de se selecionar sempre estarão dentro de um `CubeMarkerInteractor`. Caso o marcador cubo possua algum objeto já selecionado/anexado no momento da ação a ser tomada, verifica-se então se o cubo está dentro de algum `CubeMarkerInteractor`, caso positivo, verifica se o mesmo pode receber aquele objeto, e caso positivo, atribui-se então o objeto ao `CubeMarkerInteractor`. Dependendo da implementação do `CubeMarkerInteractor`, diferentes ações podem ser tomadas com o objeto, como por exemplo, no caso do marcador cena, em que o objeto será adicionado à cena, e no caso do marcador lixeira, em que ele será destruído do projeto.

O item 4 da Figura 27 é o marcador de gravação, este sendo representado pelo `GameObject RecordTools` na árvore hierárquica. Os *scripts* `AnimationController` e `AnimationUIController` fazem parte do GO, sendo o primeiro o principal ator no controle de gravação, e o segundo controlando a parte de interface gráfica do marcador, como a linha do tempo e marcadores de tempo.

O *script* `AnimationController`, possui quatro possíveis estados: parado aguardando instruções, aguardando o usuário selecionar um objeto para começar a gravar, gravando e executando as animações.

Quadro 6 – Métodos de gravação da classe `AnimationController`

```

...
69 void LateUpdate() {
70     switch (status) {
71         case STATUS.RECORDING:
72             CurrentTime = recorder.currentTime;
73             recorder.TakeSnapshot (Time.deltaTime);
74             NotifyAnimationTimesChanged ();
75             break;
76         case STATUS.PLAYING:
77             CurrentTime = longestTake.Animation["clip"].time;
78             if (!longestTake.Animation.isPlaying) {
79                 CurrentTime = longestTake.Animation["clip"].length;
80                 Status = STATUS.IDLE;
81             }
82             NotifyAnimationTimesChanged ();
83             break;
84         }
85     }
...
248 public void PrepareForRecording(bool prepare) {
249     if (!prepare) {
250         Status = STATUS.IDLE;
251         cubeMarkerController.SetAttachMode (CubeMarkerAttachMode.NORMAL);
252     } else {
253         RewindAll ();
254         Status = STATUS.WAITING_OBJECT_TO_ATTACH;
255         cubeMarkerController
                .SetAttachMode (CubeMarkerAttachMode.RECORD_MODE);
256     }
257 }
...
265 public void ObjectAttached(MovableObject obj) {
266     if (this.status == STATUS.WAITING_OBJECT_TO_ATTACH && obj.type ==
MovableObject.TYPE.SCENE_OBJECT) {
267         this.recorderTracker.source = obj.transform;
268         Status = STATUS.RECORDING;
269         InternalPlayAll ();
270     }
271 }
...

```

Fonte: elaborado pelo autor.

O Quadro 6 apresenta os métodos de gravação da classe `AnimationController`. Quando o usuário pressiona o botão de gravar, o método `PrepareForRecording` é chamado e

o controlador ficará esperando o usuário selecionar algum objeto da cena com o marcador cubo para começar a gravar. Ao selecionar um objeto com o cubo, o método `ObjectAttached` irá ser chamado através da implementação da interface `CubeMarkerListener`, e começa a gravar utilizando a classe `GORecorder` para registrar em cada chamada do `LateUpdate`, a posição e rotação do objeto sendo gravado. `LateUpdate` é um método do Unity e é chamado após o processamento de cada *frame* da *engine*.

Inicialmente utilizou-se uma classe utilitária do Unity chamada `GameObjectRecorder` para gravar as informações dos objetos em animações. Com esta classe se informa o `GameObject` que deseja ser gravado e as propriedades que devem ser registradas, gerando um `AnimationClip` no final. Entretanto, esta é uma classe utilitária do `UnityEditor` e não está disponível em modo *runtime* ao compilar para alguma plataforma. Teve de ser criado do zero então, uma nova classe que obtivesse o mesmo comportamento, porém mais simples, para somente a gravação da matriz de transformação dos objetos, ao invés de qualquer propriedade. O `GORecorder` recebe o `GameObject` que se deseja gravar e, a cada chamada do `TakeSnapshot(time)`, cria um *frame* de animação no tempo `time` com as informações de posição e rotação do objeto. A classe então disponibiliza o método `SaveToClip` que salva todos os frames registrados em um `AnimationClip`.

O Quadro 7 mostra os métodos que trabalham na parte de finalizar a gravação. A gravação é parada no momento que o botão gravar é pressionado, invocando o método `StopRecording`, ou quando o marcador cubo não é mais identificado pela câmera do Vuforia, invocando o método `MarkerLost` da interface `CubeMarkerListener`. Esses dados são coletados, para no final da gravação serem populados em um novo `AnimationClip`, que é salvo junto com o `GO` do objeto gravado em um `AnimationTake` e adicionado na lista de animações da cena. O método `CreateNewTakeAtCurrentPos` faz a parte de criar o `AnimationClip` e o `AnimationTake` para adicionar na lista de `Takes` da cena atual.

Quadro 7 – Métodos ao finalizar gravação na classe AnimationController

```

...
87 public void CreateNewTakeAtCurrentPos () {
88     Animation animation =
        recorderTracker.source.GetComponent<Animation> ();
89     if (animation) {
90         animation.RemoveClip("clip");
91     } else {
92         animation =
            recorderTracker.source.gameObject.AddComponent<Animation> ();
93     }
94
95     var clip = new AnimationClip ();
96     clip.name = "clip";
97     recorder.SaveToClip(clip);
98
99     animation.playAutomatically = false;
100    animation.AddClip(clip, "clip");
101
102    var newTake = new AnimationTake(animation, clip,
        recorderTracker.source.gameObject, recorder.GetCurves ());
103
104    var animationIndex =
        SceneController.GetCurrentScene ().Takes.FindIndex (
            take => take.Animation == animation);
105    // O objeto gravado já pertence à alguma take: substitui-la deverá
106    if (animationIndex >= 0) {
107        SceneController.GetCurrentScene ().Takes[animationIndex] = newTake;
108    } else {
109        SceneController.GetCurrentScene ().Takes.Add(newTake);
110        NotifyTakeAdded(
            SceneController.GetCurrentScene ().Takes.Count () - 1);
111    }
112
113    CalculateClipTimes ();
114 }
...
234 public void StopRecording () {
235     if (status == STATUS.RECORDING) {
236         Status = STATUS.IDLE;
237         InternalStopAll ();
238         CreateNewTakeAtCurrentPos ();
239         this.recorderTracker.source = null;
240         recorder.ResetRecording ();
241         cubeMarkerController.SetAttachMode (CubeMarkerAttachMode.NORMAL);
242         RewindAll ();
243         NotifyAnimationTimesChanged ();
244     }
245 }
...
281 public void MarkerLost () {
282     StopRecording ();
283 }
...

```

Fonte: elaborado pelo autor.

Quadro 8 – Métodos de execução e parada das animações na classe `AnimationController`

```

...
128 public void PlayAll() {
129     cubeMarkerController.ResetAttached();
130     InternalPlayAll();
131     if (SceneController.GetCurrentScene().Takes.Count > 0) {
132         Status = STATUS.PLAYING;
133     }
134 }
135
136 private void InternalPlayAll() {
137     foreach (AnimationTake take in
138             SceneController.GetCurrentScene().Takes) {
139         if (recorderTracker.source != null
140             && recorderTracker.source.gameObject == take.GameObject) {
141             continue;
142         }
143         take.Animation.Play("clip");
144     }
145 }
146
147 public void StopAll() {
148     cubeMarkerController.ResetAttached();
149     InternalStopAll();
150     Status = STATUS.IDLE;
151 }
152
153 private void InternalStopAll() {
154     foreach (AnimationTake take in
155             SceneController.GetCurrentScene().Takes) {
156         take.Animation.Stop("clip");
157     }
158 }
...

```

Fonte: elaborado pelo autor.

No Quadro 8 temos os métodos de execução de animações e de parada. O método `PlayAll` é invocado pelo `AnimationUIController`, que manda executar todas as animações da cena. O método `StopAll` também é chamado pela interface gráfica e manda parar todas as animações.

O item 5 da Figura 27, o marcador lixeira, é representado pelo `GameObject Trash` da árvore hierárquica. O seu *script* é o `TrashBinController`, que estende da classe `CubeMarkerInteractorImpl`, pois interage com o marcador cubo. O GO possui um `SphereCollider` para identificar quando um objeto entra na área de descarte da lixeira, e o marcador cubo faz o seu trabalho para identificar se o usuário deseja mover o item para a ela, chamando assim o método `ObjectReceived` da classe `TrashBinController`. A implementação do método identifica o tipo de objeto móvel que está sendo descartado, podendo ser três tipos de objeto: `TAKE_OBJECT`, que representa um ícone de uma animação do seletor de animações, pois o usuário pode mover os ícones do seletor para excluir animações ou cenas; `SCENE_INFO_OBJECT`, que representa um ícone de uma cena no seletor

de cenas; `SCENE_OBJECT` que representa um objeto gráfico da cena, podendo conter uma animação vinculada ou não.

Dependendo do tipo de objeto, diferentes ações serão tomadas. Objetos que são ícones dos seletores irão excluir o tipo de informação à qual estão vinculados, ícones de animações irão excluir a animação selecionada, ícones de cena terão a sua cena excluída, já objetos da cena serão excluídos da cena e terão suas animações também removidas, caso existentes. O Quadro 9 apresenta a implementação do método `ObjectReceived` na classe `TrashBinController`, que é invocado pelo marcador cubo quando o usuário está com o objeto posicionado em cima da lixeira.

Quadro 9 – Método invocado pelo marcador cubo na classe `TrashBinController`

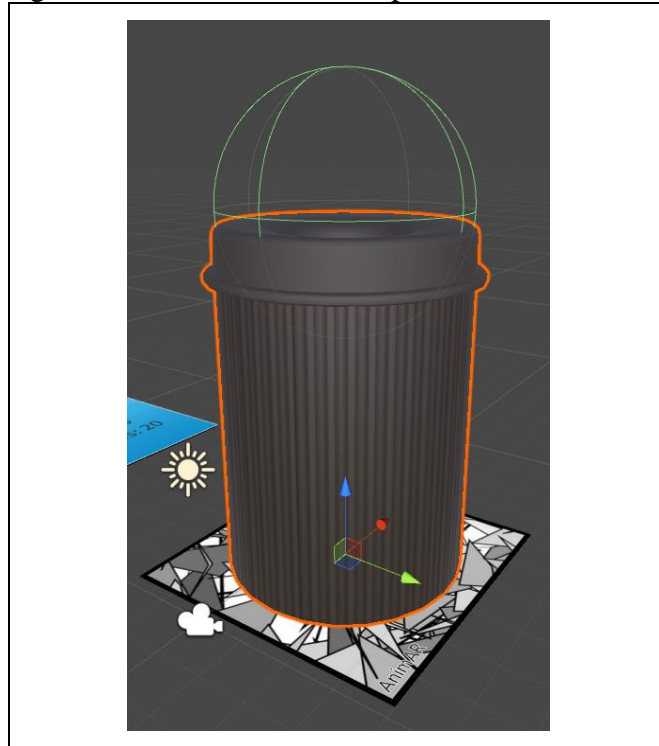
```

...
18 public override void ObjectReceived(MovableObject obj) {
19     var takeNumber = -1;
20     var sceneNumber = -1;
21     switch (obj.type) {
22         case MovableObject.TYPE.TAKE_OBJECT:
23             takeNumber = obj.GetComponent<NumberIcon>().Number;
24             break;
25         case MovableObject.TYPE.SCENE_OBJECT:
26             takeNumber = SceneController.GetCurrentScene().Takes.FindIndex(
                take => take.gameObject == obj.gameObject);
27             break;
28         case MovableObject.TYPE.SCENE_INFO_OBJECT:
29             sceneNumber = obj.GetComponent<NumberIcon>().Number;
30             break;
31     }
32     Destroy(obj.gameObject);
33
34     if (takeNumber > -1) {
35         AnimationController.RemoveTake(takeNumber);
36     } else if (sceneNumber > -1) {
37         SceneController.RemoveScene(sceneNumber);
38     }
39
40     var effectObj = GameObject.Instantiate(IncinerateEffectGO);
41     effectObj.transform.parent = this.transform;
42     effectObj.transform.localPosition = Vector3.zero;
43     effectObj.transform.localRotation =
        Quaternion.Euler(new Vector3(90, 0, 90));
44 }
...

```

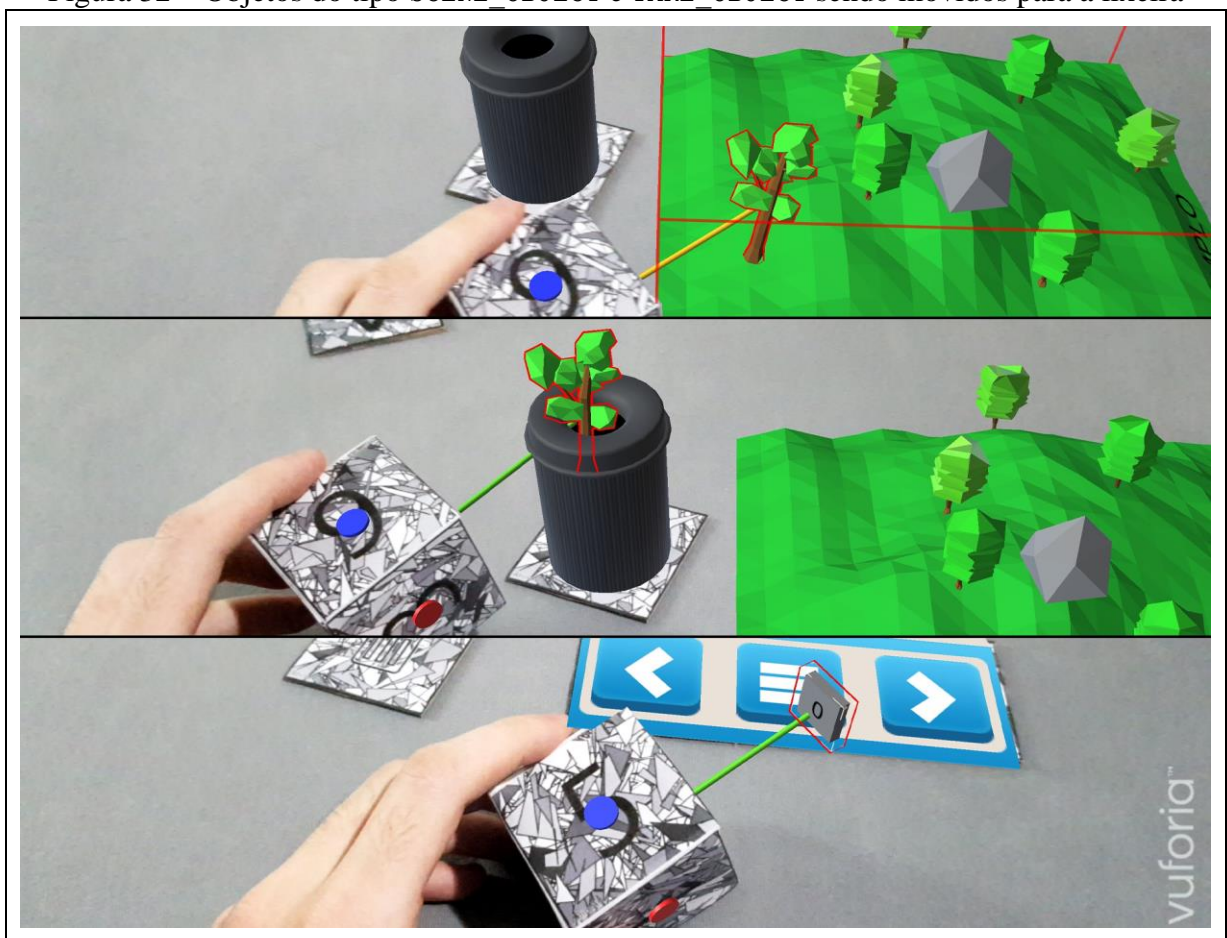
Fonte: elaborado pelo autor.

Figura 31 – SphereCollider posicionado na lixeira



Fonte: elaborado pelo autor.

Figura 32 – Objetos do tipo SCENE_OBJECT e TAKE_OBJECT sendo movidos para a lixeira



Fonte: elaborado pelo autor.

Por fim, o item 6 da Figura 27 representa o marcador inspetor. O `GameObject` responsável é o `Inspector` na árvore hierárquica da cena, possuindo o *script* `InspectorController` que cuida de toda a implementação do marcador. O `GO Inspector` possui um `BoxCollider` e um `Rigidbody` igualmente ao marcador cubo, porém aqui, ao identificar contato com outro objeto através do método `OnTriggerEnter`, é verificado se o objeto possui um *script* `MovableObject` e, assim, extrai as informações relevantes ao usuário para popular a interface gráfica do inspetor.

Quadro 10 – Implementação do evento do `Collider` na classe `InspectorController`

```

...
20 void OnTriggerEnter(Collider other) {
21     var movable = other.GetComponent<MovableObject>();
22     var titleString = "Sem informação";
23     var descriptionString = "Nenhuma informação encontrada para o objeto
selecionado";
24     if (movable != null) {
25         switch (movable.type) {
26             case MovableObject.TYPE.SCENE_OBJECT:
27                 titleString = movable.gameObject.name;
28                 var takes = SceneController.Instance.GetCurrentScene().Takes;
29                 var index = takes.FindIndex(
                                     t => t.GameObject == movable.gameObject);
30                 if (index >= 0) {
31                     descriptionString = "Animação: " + (index + 1);
32                     descriptionString += "\nDuração: " + takes[index].Clip.length;
33                 } else {
34                     descriptionString = "Sem animação";
35                 }
36                 break;
37             case MovableObject.TYPE.SCENE_INFO_OBJECT:
38                 var sceneNumber = other.GetComponent<NumberIcon>().Number;
39                 titleString = "Cena " + sceneNumber;
40                 var scene = SceneController.Instance.scenes[sceneNumber];
41                 var objectsQuantity =
                     scene.Map.GetComponentsInChildren<MovableObject>().Length;
42                 var takesQuantity = scene.Takes.Count;
43                 descriptionString = "Objetos: " + objectsQuantity;
44                 descriptionString += "\nAnimações: " + takesQuantity;
45                 break;
46             case MovableObject.TYPE.TAKE_OBJECT:
47                 var takeNumber = other.GetComponent<NumberIcon>().Number;
48                 titleString = "Animação " + takeNumber;
49                 var take = SceneController.Instance.GetCurrentScene()
                                     .Takes[takeNumber];
50                 descriptionString = "Duração: " + take.Clip.length;
51                 break;
52         }
53         title.text = titleString;
54         description.text = descriptionString;
55     }
56 }
...

```

Fonte: elaborado pelo autor.

3.3.1.4 Persistência

O Unity disponibiliza uma classe utilitária chamada `PlayerPrefs`. Ela é utilizada em forma de um mapa de chave-valor, salvando e recuperando dados que são persistidos no dispositivo do usuário, sendo multiplataforma. Utilizou-se então do `PlayerPrefs` para salvar as cenas e animações criadas pelo usuário. No entanto, o Unity não disponibiliza uma maneira fácil de serializar `GOs` ou `MonoBehaviors`, somente classes `C#` puras, marcadas com a anotação `Serializable`. Para que se tornasse possível salvar as cenas e animações, recuperando-as ao reiniciar a aplicação, criou-se então um controle de persistência para o projeto.

De maneira geral, os `GOs` que podem ser persistidos, devem implementar a interface `Persistent<T>` com os métodos `T:GetPersistData()` e `LoadPersistData(T)`, sendo `T` o tipo de dado que aquele `GO` trabalha. A persistência é controlada pelo script `PersistController`, que convoca o método `GetPersistData` das classes `Persistent`, transforma os dados em `JSON` e salva no `PlayerPrefs`. A mesma classe, ao iniciar a aplicação, carrega o `JSON` salvo do `PlayerPrefs` e invoca o método `LoadPersistData` das classes persistentes.

Quadro 11 – Métodos que salvam e carregam os dados na classe `PersistController`

```

25 public void PersistEverything() {
26     UserPersistData userData = new UserPersistData();
27     userData.Scenes = new ScenePersistData
28         [SceneController.Instance.scenes.Count()];
29     for (var i = 0; i < SceneController.Instance.scenes.Count(); i++) {
30         userData.Scenes[i] = SceneController.Instance.scenes[i]
31             .GetPersistData();
32     }
33     PlayerPrefs.SetString("data", JsonUtility.ToJson(userData, false));
34 }
35 public void LoadFromPersistedData() {
36     var jsonData = PlayerPrefs.GetString("data");
37     if (!String.IsNullOrEmpty(jsonData)) {
38         UserPersistData userData =
39             JsonUtility.FromJson<UserPersistData>(jsonData);
40         SceneController.Instance.scenes.Clear();
41         foreach (var scene in userData.Scenes) {
42             var newScene = new GameObject("Map").AddComponent<Scene>();
43             newScene.transform.parent =
44                 SceneController.Instance.DesactivatedScenes.transform;
45             newScene.transform.localPosition = Vector3.zero;
46             newScene.transform.localScale = new Vector3(0.06f, 0.06f, 0.06f);
47             newScene.LoadPersistData(scene);
48             SceneController.Instance.scenes.Add(newScene);
49             SceneController.Instance.CurrentScene = 0;
50         }
51     }
52 }

```

Fonte: elaborado pelo autor.

O Quadro 11 mostra a implementação dos métodos de salvamento e carregamento. O método `PersistEverything` irá invocar o método `GetPersistData` de todas as cenas do `SceneController` que, em sua implementação, irá salvar todos os objetos da cena e o `GetPersistData` de todas as animações presentes, gerando um `ScenePersistData` no final. Um `ScenePersistData` consiste da posição, rotação, escala e nome do `prefab` de todos os objetos da cena, e informações de todas as animações, representadas pela classe `AnimationTakePersistData`. O `ScenePersistData` de todas as cenas é então salvo em uma lista da classe `UserPersistData`, que é transformada em JSON e salva no `PlayerPrefs`. Ao iniciar a aplicação, o método `LoadFromPersistedData` irá carregar o JSON do `PlayerPrefs` e instanciar o `UserPersistData` através do mesmo. É criado então uma cena para cada `ScenePersistData` encontrado e invocado o método `LoadPersistData` passando como parâmetro o `ScenePersistData` carregado. A implementação do `LoadPersistData` na classe `Scene` irá ficar encarregada de instanciar novamente os `GameObjects` dos objetos do mapa e criar novamente os `AnimationClips` e `AnimationTakes` de todas as animações salvas.

A aplicação salva automaticamente os dados do usuário em alguns pontos de salvamento estratégicos, como na identificação de novos marcadores e criação de novas animações e cenas.

3.3.1.5 Cardboard

A ativação do suporte à Realidade Virtual é muito simples no Vuforia com Unity, praticamente não necessitando de programação. Foi ativado nas configurações do projeto a opção `Virtual Reality Supported` e, depois, no carregamento da cena do Vuforia, é configurado o `Eyewear Type` do Vuforia para `VideoSeeThrough` ou para `None`, dependendo se a opção de utilizar VR está ativada ou não. O Quadro 12 mostra a implementação do método de inicialização do *script* `ChangeVuforiaConfig`, que está presente no `GameObject ARCamera` da `scene` do Vuforia. Ele irá configurar o Vuforia para utilizar o Cardboard ou não, dependendo da seleção atual feita pelo usuário.

Quadro 12 – Método da classe `ChangeVuforiaConfig` que configura a visualização do Vuforia

```

12 void Start() {
13     var vuforiaType = PlayerPrefs.GetString("vuforiaType");
14     if (vuforiaType.Equals("vr")) {
15         DigitalEyewearARController.Instance.SetEyewearType(
16             DigitalEyewearARController.EyewearType.VideoSeeThrough);
17         DigitalEyewearARController.Instance.SetStereoCameraConfiguration(
18             DigitalEyewearARController.StereoFramework.Cardboard);
19     } else {
20         DigitalEyewearARController.Instance.SetEyewearType(
21             DigitalEyewearARController.EyewearType.None);
22     }
23 }

```

Fonte: elaborado pelo autor.

3.3.1.6 Objetos gráficos

Os objetos gráficos como, modelos 3D e elementos da interface do usuário, apesar de alguns terem sido criados pelas próprias ferramentas do Unity, grande parte fora obtido da loja de `Assets` do Unity, a `Asset Store`. Os modelos 3D utilizados para a criação das cenas foram obtidos do `Asset Low Poly Pack`. O efeito de incineração da lixeira foi obtido do `Asset Fire & Spell Effects`. Os elementos da interface gráfica, como ícones e fundos, foram obtidos do `Asset Simple UI`. Todos disponíveis gratuitamente na loja `Asset Store`.

Por fim, para a realização do efeito de borda realçada ou *outline*, quando o marcador Cubo está selecionado um objeto, fora utilizado o `Asset Quick Outline`, que está disponível gratuitamente na `Asset Store`.

3.3.2 Operacionalidade da implementação

Para a utilização da ferramenta, é necessário possuir em mãos o kit de marcadores da aplicação e um dispositivo móvel com o aplicativo instalado.

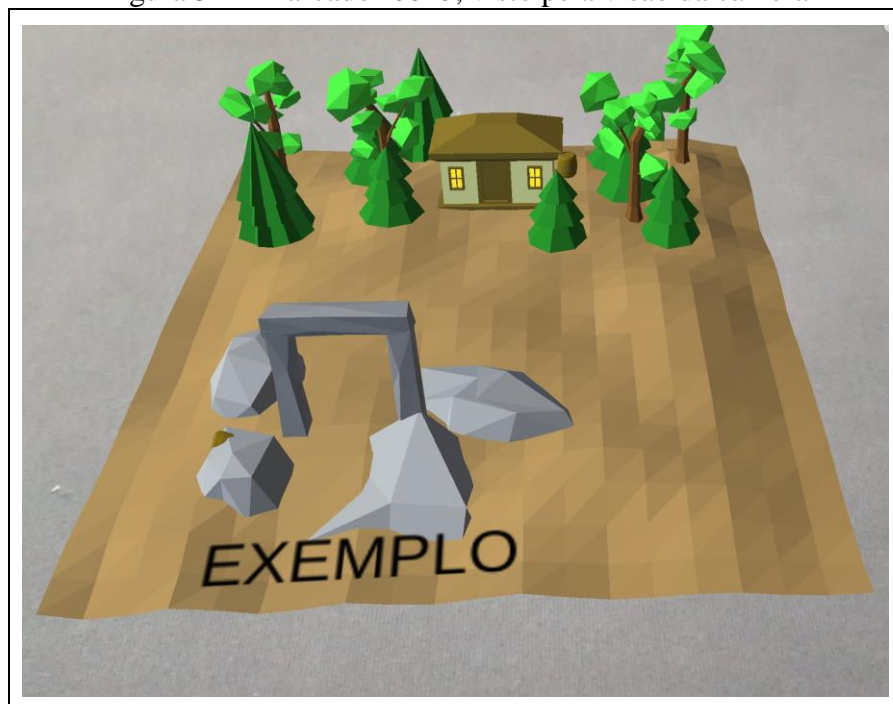
Após abrir a aplicação, no `Menu Inicial` (Figura 33) o usuário deve selecionar, ou não, se deseja utilizar o modo de realidade virtual com `Cardboard`, e clicar no botão `Iniciar` para abrir a visualização da câmera do dispositivo e começar a identificar os marcadores.

Figura 33 – Menu inicial da aplicação



Fonte: elaborado pelo autor.

Com a visão da câmera ativa, é possível então apontar a câmera para os marcadores e começar a utilizar a aplicação. Apontando a câmera para o marcador *Cena* da Figura 25, o usuário conseguirá visualizar a representação gráfica da cena atual, conforme a Figura 34.

Figura 34 – Marcador *Cena*, visto pela visão da câmera

Fonte: elaborado pelo autor.

Apontando a câmera para o marcador Seletor da Figura 21, o usuário visualizará um menu de seleção, que cria uma interface tangível para a interação com o usuário. A Figura 35 mostra o marcador Seletor, com o seletor no modo Fábrica de Objetos. Pressionando < ou >, o usuário navegará entre os objetos do seletor atual.

Figura 35 – Marcador Seletor no modo Fábrica de Objetos



Fonte: elaborado pelo autor.

Ao pressionar o botão de troca de seletor, o usuário poderá selecionar entre os seletores de Fábrica de Objetos, Seleção de Animação e Seleção de Cena, conforme a Figura 36.

Figura 36 – Trocando o modo de seleção do marcador Seletor



Fonte: elaborado pelo autor.

No Seletor de Cena, ao pressionar < ou > a cena atual no marcador Cena irá ser trocada automaticamente para a cena selecionada. Quando pressionado > e não houver mais cenas seguintes, uma nova cena em branco é criada.

Figura 37 – Mudando a cena atual

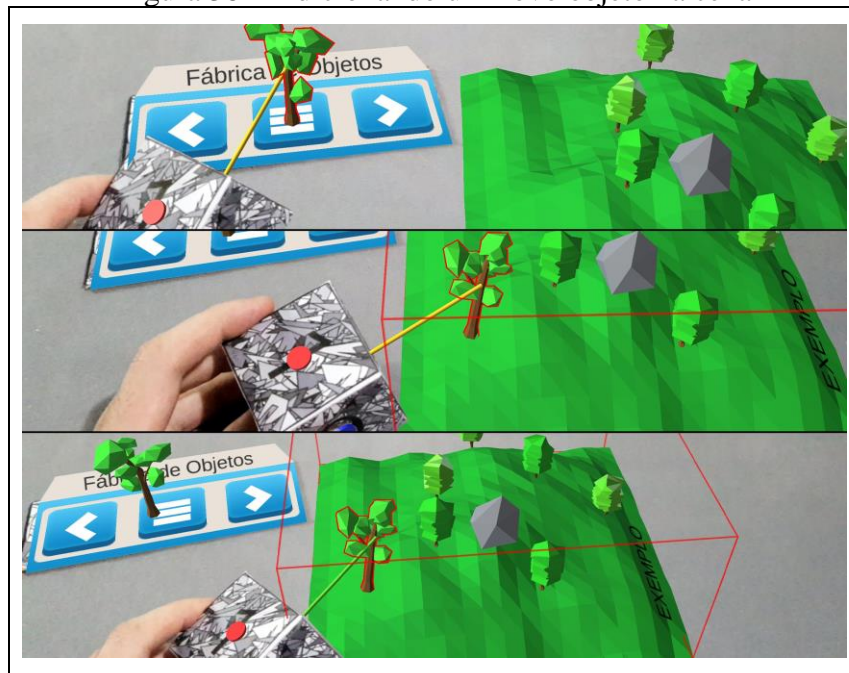


Fonte: elaborado pelo autor.

Utilizando do marcador Cubo, o usuário pode mover o objeto selecionado na Fábrica de Objetos para a cena, adicionando-o assim na mesma. Para selecionar com o Cubo, o usuário deve manter o indicador do cubo apontado para o objeto e aguardar dois segundos imóvel; Assim o objeto será selecionado pelo cubo, sendo o mesmo processo para adicionar o objeto em uma determinada posição da cena. O indicador do cubo ficará amarelo e o objeto que está sendo selecionado ficará realçado em vermelho, como forma de auxílio ao usuário.

A Figura 38 mostra o usuário selecionando um objeto da Fábrica de Objetos do marcador Seletor, com o marcador Cubo, e adicionando na cena atual no marcador Cena.

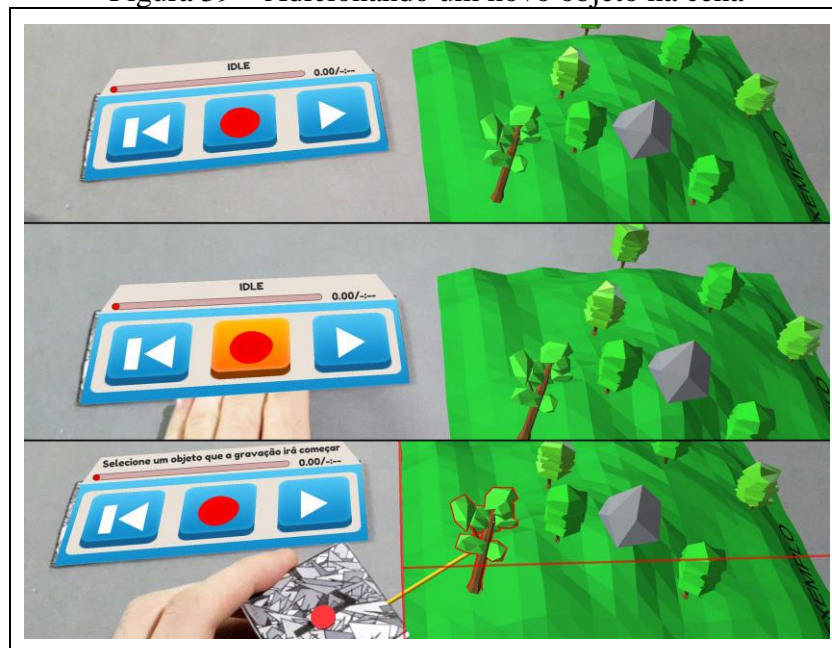
Figura 38 – Adicionando um novo objeto na cena



Fonte: elaborado pelo autor.

Ao apontar a câmera para o marcador *Gravador*, o menu de interface tangível de gravação será apresentado ao usuário com três botões, conforme Figura 39. O primeiro botão, retorna todas as animações para o início (no caso para o tempo zero da animação). O segundo é para iniciar a gravação de uma nova animação. O último, o botão de *play* e *pause*, executa ou para todas as animações da cena. A interface apresenta também uma linha do tempo das animações, mostrando onde cada animação acaba e qual o tempo atual e final, em segundos, da execução das animações.

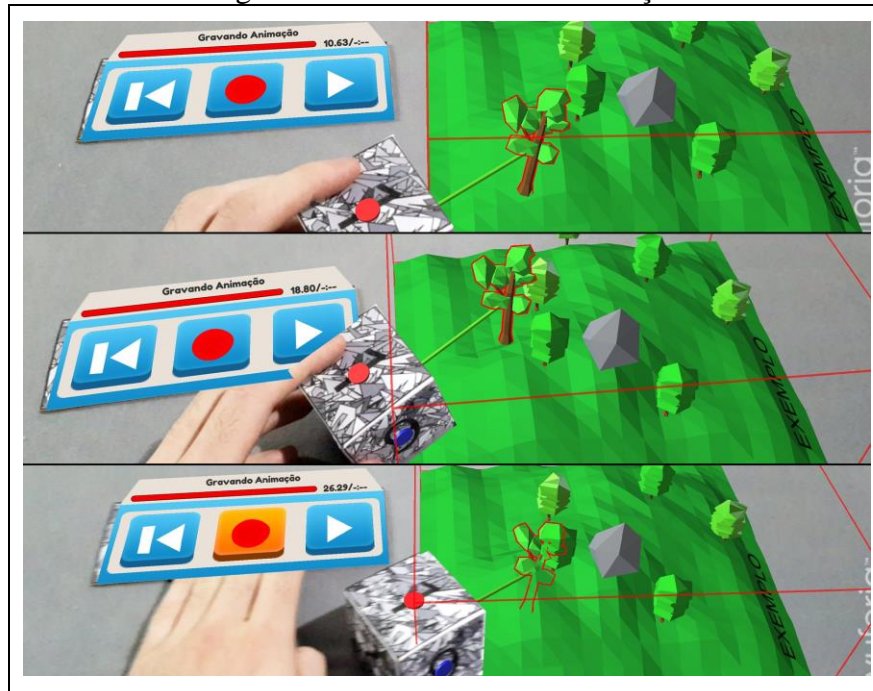
Figura 39 – Adicionando um novo objeto na cena



Fonte: elaborado pelo autor.

Ao pressionar o botão de gravação, o gravador entrará em estado de espera até que o usuário selecione o objeto da cena na qual deseja gravar uma animação. Quando selecionado, a gravação irá começar automaticamente e a movimentação do objeto a partir deste momento estará sendo gravada para no final gerar a animação. A gravação para automaticamente quando o marcador *Cubo* é perdido/escondido da visão da câmera ou quando pressionado o botão de gravar novamente.

Figura 40 – Gravando uma animação

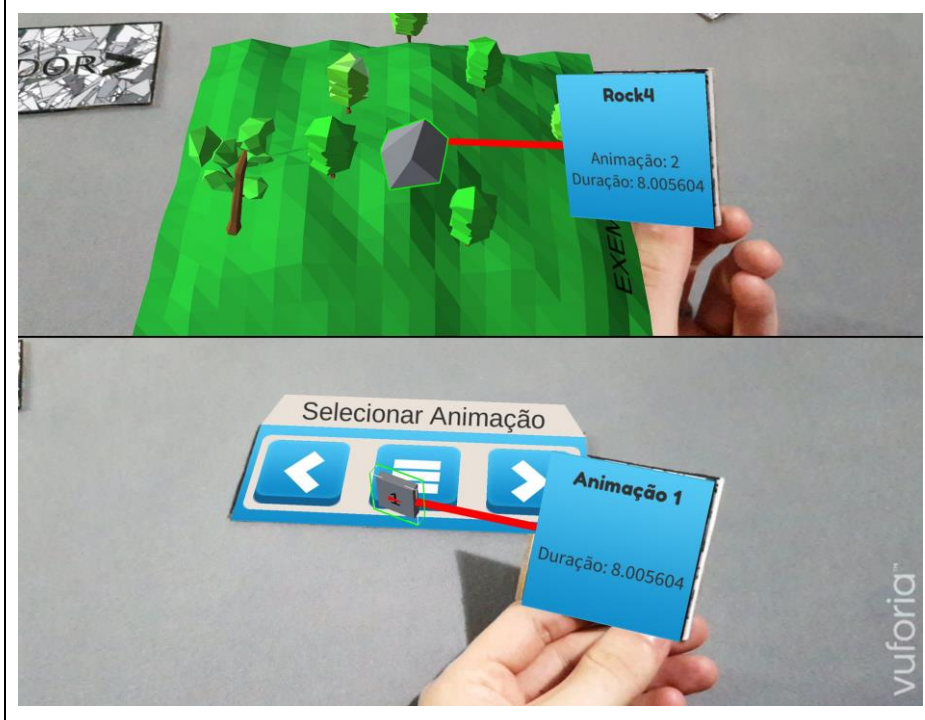


Fonte: elaborado pelo autor.

Novas animações são sempre geradas ao gravar um objeto que ainda não tenha sido gravado. Caso o usuário inicie uma nova gravação para um objeto que já está animado, a animação anterior será excluída.

Apontando a câmera para o marcador *Inspetor*, em conjunto com o marcador *Cena* ou *Seletor*, o usuário pode inspecionar os objetos para extrair informações relevantes. Na Figura 41, o usuário aponta o *Inspetor* para um objeto da cena, mostrando que o mesmo possui a animação de número dois da cena com a duração de oito segundos, em seguida, apontando para o ícone da animação no marcador *Seletor*, visualiza o tempo de duração da primeira animação.

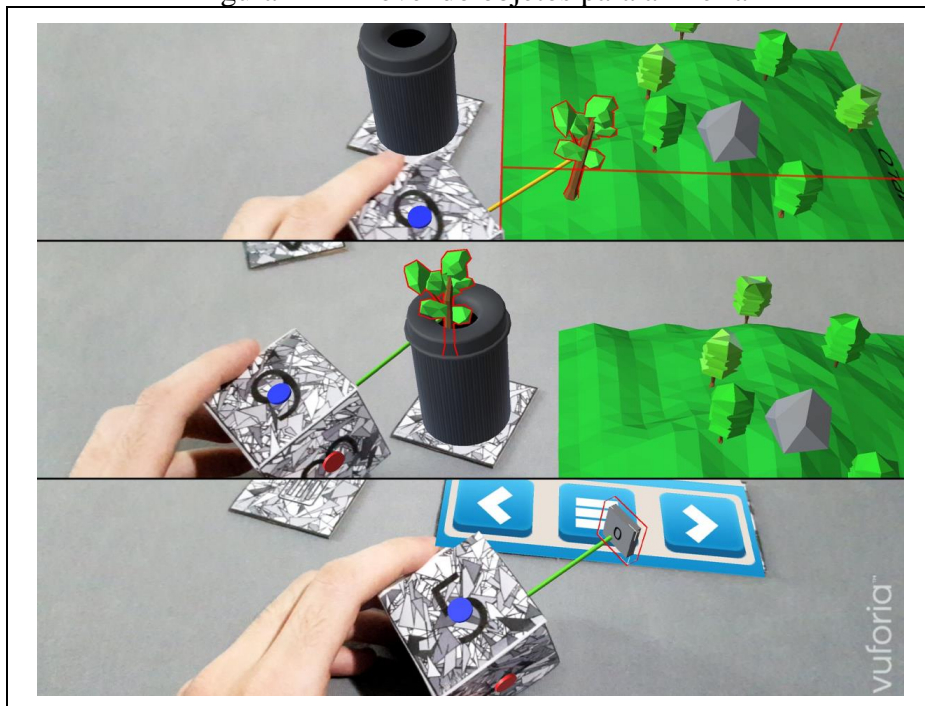
Figura 41 – Marcador Inspetor inspecionando objetos da Cena e do Seletor



Fonte: elaborado pelo autor.

Por fim, com a utilização do marcador *Lixeira*, o usuário pode mover objetos da cena, ícones de animações e ícones de cenas para a lixeira, excluindo-os. Caso o usuário mova um ícone de animação para a lixeira, a animação respectiva será excluída. Caso mova um ícone de cena, a cena selecionada será também excluída. A Figura 42 mostra o usuário movendo um objeto da cena para a lixeira.

Figura 42 – Movendo objetos para a lixeira



Fonte: elaborado pelo autor.

Para voltar ao Menu Principal, o usuário deve pressionar o botão de voltar do dispositivo móvel que está sendo utilizado.

3.4 ANÁLISE DOS RESULTADOS

Nesta seção, serão mostrados os experimentos realizados com a ferramenta.

3.4.1 Metodologia

Os testes foram realizados no mês de junho de 2018, no laboratório LIFE do campus 1 da Universidade Regional de Blumenau, com uma turma do curso de Pedagogia. Também estavam presentes o auxílio do autor, do orientador deste trabalho, e do professor da turma de Pedagogia, professor Maurício Capobianco Lopes. A turma foi dividida em oito ilhas, cada uma contendo um dispositivo *tablet* ou *smartphone*, um notebook com um formulário de pesquisa Google Forms aberto e um kit de marcadores. Cada ilha teve aproximadamente 30 minutos para utilizar a aplicação e responder o formulário em conjunto, possuindo em média três alunos por ilha. Os dispositivos utilizados foram dois *tablets* iPad, dois *smartphones* iPhone, dois *smartphones* Android MOTO G4 PLUS, um *smartphone* Android Samsung Galaxy S8 e um *smartphone* Android MOTO G6, emprestado de uma aluna de pedagogia. Optou-se por não realizar testes com o aplicativo no modo Cardboard, isto porque segundo avaliação prévia realizada pelo autor, identificou-se que a usabilidade com os tipos de *smartphones* e HMD disponíveis pareceu muito complicada, principalmente para pessoas sem um bom conhecimento da ferramenta e de Realidade Aumentada.

3.4.2 Experimentos e resultados

Com a turma dividida em oito grupos de, em média, três alunos, o professor e orientador deste trabalho, Dalton Solano dos Reis, fez uma breve apresentação sobre o trabalho e sobre o autor para os alunos de Pedagogia. Logo após isso, o autor, juntamente com o orientador, demonstrou a utilização do aplicativo em tempo real, passando pelas funcionalidades básicas dos marcadores, e espelhando a tela do dispositivo utilizado como exemplo em uma TV. Então, os alunos começaram a realização dos testes através do formulário de teste, em formato de passo a passo, para responderem um questionário de avaliação no final. A turma foi dividida em oito grupos, porém foram recebidas somente sete submissões do formulário, pois um grupo não conseguiu responder a tempo. A primeira etapa do formulário consiste em recolher dados sobre o perfil dos usuários, cujas respostas estão disponibilizadas na Tabela 1.

Tabela 1 – Perfil dos usuários

Idade	14,3% 20 anos
	28,6% 21 anos
	42,9% 22 anos
	14,3% 27 anos
Sexo	100% Feminino
Nível de escolaridade	100% Ensino superior incompleto
Utiliza dispositivos móveis com qual frequência	85,7% Frequentemente
	14,3% Às vezes
Já utilizou aplicações de Realidade Aumentada	57,1% Sim
	42,9% Não
Já utilizou aplicações de Realidade Virtual	71,4% Sim
	28,6% Não

Fonte: elaborado pelo autor.

Conforme observado, a maior parte dos alunos possuíam entre 20 e 22 anos de idade, sendo todos do sexo feminino. Como os testes foram realizados com uma turma de graduação, era de se esperar que todas possuiriam ao menos ensino superior incompleto. Grande parte já utilizava dispositivos móveis com bastante frequência, e já tinham experienciado aplicações de Realidade Virtual, porém, somente pouco mais da metade já tinha utilizado aplicações de Realidade Aumentada.

A segunda etapa do formulário é um passo a passo que demonstra todas as funcionalidades básicas da aplicação, como criar cenas, adicionar objetos e gravar animações. Para cada passo, os alunos deveriam informar se conseguiram realizar o passo com sucesso ou não, sendo que em grande parte dos grupos, o autor e o orientador deram eventuais suportes em determinados passos. Os resultados estão disponíveis na Tabela 2.

Tabela 2 – Passo a passo das funcionalidades básicas

Abra o aplicativo AnimAR e realize o pequeno tutorial dentro dele. O objetivo é aprender os comandos básicos.	100% Concluído
Aperte "Iniciar" no menu principal para iniciar o jogo. Utilize os marcadores "Cena" e "Seletor", tente fazer com que a câmera do dispositivo detecte-os.	100% Concluído
O modo padrão do marcador "Seletor" é o Fábrica de Objetos. Aperte o botão do meio para trocar o seletor, até chegar no modo seletor de cena.	100% Concluído
No modo "Seletor de Cena", você pode navegar entre as cenas da sua aplicação. Experimente navegar entre as cenas com os botões "<" e ">", visualizando a cena no marcador "Cena". Por padrão, o aplicativo vem com duas cenas de exemplo.	71,4% Concluído 28,6% Não conseguiu concluir
Mude a cena atual para a Cena 0. Mude o seletor atual para a "Fábrica de Objetos". Navegue entre a fábrica até encontrar uma pedra cinza. Com o marcador "Cubo", tente mover esse objeto da fábrica para dentro da Cena 0, em qualquer lugar da cena. Para selecionar o objeto antes de mover, você deve deixar o raio indicador do cubo dentro do objeto, parado por alguns segundos.	100% Concluído
Agora, pegue o marcador "Gravador". Pressione o botão do meio para iniciar uma gravação. Com o marcador "Cubo", selecione a pedra anteriormente adicionada à cena. Ao selecionar a pedra a gravação iniciará automaticamente. Movimente a pedra pela cena, de acordo como você deseja animá-la. Quando você terminar sua animação, aperte novamente o botão de gravar para parar a gravação, ou tampe/esconde o marcador "Cubo" para que ele não seja mais detectado e pare a gravação.	85,7% Concluído 14,3% Não conseguiu concluir
Pressione o botão de "Play" para então reproduzir a animação anteriormente gravada. Pronto, você produziu a sua primeira animação! Experimente gravar novas animações para outros objetos da mesma cena.	85,7% Concluído 14,3% Não conseguiu concluir
Pegue o marcador "Inspetor", plugue então inspetor no objeto que você acabou de animar, e verifique se aparecem informações sobre aquele objeto e sua animação.	100% Concluído
Pegue agora o marcador "Seletor" e mude o modo para o "Seletor de Animações". Verifique que agora a animação deve possuir a animação 0. Com o "Seletor de Animações" e o marcador de "Cena" abertos ao mesmo tempo, você pode visualizar na cena qual o objeto animado pela animação atual selecionada no seletor, sendo o objeto contornado em azul.	100% Concluído
Ative o marcador "Lixeira". Com o "Seletor de Animações" ainda aberto, pegue o marcador "Cubo" e tente mover o ícone da animação "0" para a lixeira. A animação então deverá ser excluída. Tente pressionar "Play" no marcador "Gravador", e verifique se a animação não está mais presente.	85,7% Concluído 14,3% Não conseguiu concluir

Fonte: elaborado pelo autor.

Conforme observado, em grande parte os objetivos conseguiram ser concluídos com facilidade, com exceção do quarto passo, onde 28,7% dos alunos tiveram dificuldades. Este foi o passo em que o auxílio foi mais requisitado. Observou-se ser mais difícil pois é o primeiro passo em que há uma interação mais profunda com a interface tangível, sendo necessário observar dois marcadores simultaneamente, o de seletor e o de cena. Observou-se

também, um certo desentendimento de como o seletor funciona e os três tipos diferentes de seleção. Após isso, outro ponto bastante requisitado de auxílio foi na parte de gravar a animação, o que já era esperado. Muitos alunos tiveram dificuldade em entender o passo a passo para gravar a animação e os estados de gravação, como quando se aperta o gravar e deve então selecionar um objeto para começar a gravar. Muitas vezes, o botão de gravar era pressionado duas vezes, anulando o início da gravação. Apesar de o marcador de gravação dar um feedback textual de que a gravação é iniciada, os alunos não conseguiram notar este tipo de feedback no começo.

A penúltima etapa do formulário de testes, consistia em trabalhar a parte criativa de construção com a aplicação, incentivando os usuários a adicionarem objetos e animações de acordo com a vontade deles. Nesta etapa, grande parte já tinha conseguido entender como a aplicação funciona, não sendo mais requisitado ajuda, porém, uma parte dos grupos não conseguiram concluir a tempo, deixando esta parte de lado. Os resultados podem ser observados na Tabela 3.

Tabela 3 – Modo livre

Com o "Seletor de Cenas", navegue até a Cena 2, que é uma cena completamente nova. Agora, fazendo uso dos marcadores ensinados, construa uma cena que contenha dois portais de pedra, um em cada extremidade do mapa. Crie uma "rua" entre esses dois portais, com árvores, pedras e outros objetos pelo caminho. No final, adicione um Lobo em um dos portais de pedra, e anime-o fazendo com que ele "ande" de um portal para o outro.	71,4% Concluído 28,6% Não conseguiu concluir
Se desejar, fique a vontade de criar novas animações para esta cena. Você pode por exemplo, adicionar novos lobos na cena e fazer uma matilha de lobos passeando pelo bosque.	71,4% Concluído 28,6% Não conseguiu concluir

Fonte: elaborado pelo autor.

Por fim, a última etapa consistia em uma avaliação geral do uso da ferramenta, avaliando diferentes critérios de usabilidade com notas de zero à cinco. Os resultados desta etapa, estão disponíveis na Tabela 4.

Tabela 4 – Avaliação Geral

Os passos desta pesquisa foram concluídos com facilidade?	14,3% - 2
	71,4% - 3
	14,3% - 4
Quantas tarefas você concluiu sem NENHUM auxílio externo?	14,3% - 2
	42,9% - 3
	28,6% - 4
	14,3% - 5
Como você classifica a usabilidade de se construir novas cenas/cenários?	14,3% - 1
	14,3% - 2
	57,1% - 4
	14,3% - 5
Como você classifica a usabilidade de se gravar animações?	42,9% - 2
	28,6% - 3
	14,3% - 4
	14,3% - 5
Como você classifica a usabilidade do AnimAR em Geral?	14,3% - 2
	14,3% - 3
	42,9% - 4
	28,6% - 5
Você acha que o AnimAR cumpriu seu objetivo de experimentar e fornecer uma nova maneira de se criar animações com facilidade?	14,3% - 2
	28,6% - 3
	28,6% - 4
	28,6% - 5

Fonte: elaborado pelo autor.

A maioria dos grupos acharam que somente metade dos passos foram fáceis, entretanto, 50% dos grupos conseguiram concluir ao menos metade das tarefas sem auxílio externo e 37,5% conseguiram concluir todas ou quase todas as tarefas sem auxílio. A maioria achou fácil a construção de cenários, porém já ao gravar animações, a maioria achou ruim e difícil. A parte de gravar animações foi onde se observou maior dificuldade e necessidade de auxílio do autor e do orientador, observou-se desentendimento em como a parte de gravar e parar funciona, ficando claro que esta parte da ferramenta não ficou de fácil entendimento e usabilidade para usuários que não estão acostumados com este tipo de aplicação. Entretanto, as avaliações da usabilidade em geral da ferramenta foram satisfatórias, sendo que 71,5% dos grupos concluíram que a aplicação possui uma usabilidade boa ou ótima.

Em geral, os resultados foram satisfatórios, entretanto, observou-se uma certa dificuldade no começo ao utilizar aplicação, isso porque a maioria dos alunos nunca utilizaram aplicações de Realidade Aumentada com Interfaces Tangíveis, sendo uma tecnologia ainda não muito utilizada nos dias atuais. A utilização do cubo e dos botões virtuais foram a principal dificuldade no início. Ao utilizar o cubo, não se entendia como deveria selecionar os objetos, pois é necessário manter o cubo parado por alguns segundos até que o objeto fosse selecionado. Com os botões, não se entendia como eles funcionavam exatamente, sendo que em muitas vezes, os alunos passavam a mão ou braço sem querer por

cima de um marcador com botões e acabavam ativando botões indesejados. Após um melhor entendimento de como funcionavam os marcadores e os botões virtuais, observou-se uma melhoria na usabilidade da ferramenta.

Apesar das dificuldades na utilização no início, os alunos logo no começo demonstraram interesse na ferramenta, inclusive no final, alguns alunos perguntaram se poderiam instalar o aplicativo em seus próprios dispositivos. Um ponto interessante observado com a utilização do marcador cubo, foi um grupo em que ao ter um objeto selecionado pelo marcador e não entendendo como descartar o mesmo, um dos alunos tentou chacoalhar o cubo com as mãos, numa tentativa de tirar o objeto “preso” ao mesmo. É um ponto interessante e de possível estudo de uma futura melhoria para aplicação.

Testes de utilização com o Cardboard não foram realizados com uma quantidade significativa de pessoas, pois o uso do mesmo não se mostrou muito eficiente. O problema encontrado foi que, por limitações de baixa resolução oferecidas pelo dispositivo testado, a dificuldade em focar as lentes do Cardboard na tela do dispositivo e a necessidade de fazer uma abertura na parte de trás do Cardboard para abrir o espaço da lente da câmera, tornaram a utilização com o Cardboard muito crua e difícil. O próprio autor encontrou dificuldades na utilização da ferramenta neste modo. Acredita-se que com a utilização de dispositivos dedicados para esta funcionalidade como um optical head-mounted display (OHMD), em que se permite a visualização de projeções virtuais dentro do mundo real, a interação seria muito mais rica e produtiva.

3.4.3 Comparativo com os trabalhos correlatos

O Quadro 13 mostra um comparativo da aplicação desenvolvida com os trabalhos correlatos. As colunas representam os trabalhos e as linhas as características. As características utilizadas para a comparação foram escolhidas de acordo com suas relevâncias para o projeto desenvolvido.

Quadro 13 – Comparativo entre os trabalhos correlatos

Correlatos	Silva (2016)	Schmitz (2017)	Lee et al. (2004)	AnimAR
Características				
realidade aumentada	X	X	X	X
interface tangível	X	X	X	X
manipulação de objetos virtuais	X		X	X
criação de animações			X	X
API de Realidade Aumentada	Vuforia	Vuforia	ARToolKit	Vuforia
motor gráfico	Unity	Unity	OpenGL	Unity
plataforma	Android	Android	Windows	Android/iOS
suporte a cardboard				X

Fonte: elaborado pelo autor.

É possível observar que todos os trabalhos possuem Realidade Aumentada através de uma câmera e uma Interface de Usuário Tangível para a facilitar a utilização e interação com o mundo virtual. Os projetos de Silva, Lee et al. e o AnimAR, focam na manipulação dos objetos virtuais através da Interface de Usuário Tangível, seus objetivos são a criação de conteúdo digital ao mesmo tempo em que se visualiza e experencia o resultado. Porém, somente o trabalho de Lee et al. e o AnimAR permitem a criação de cenas animadas, uma forma de se contar uma história através da criação de cenas com objetos animados.

Em termos de metodologia e ferramentas utilizadas para o desenvolvimento, Silva, Schmitz e o AnimAR utilizam o Vuforia para a captura dos marcadores e posicionamento dos objetos 3D em tempo real, enquanto Lee et al. utilizam ARToolKit para o mesmo propósito. Lee et al. desenvolveu a parte de renderização gráfica utilizando OpenGL, Silva, Schmitz e o AnimAR desenvolveram através do motor de jogos Unity3D, que abstrai a maior parte do trabalho de renderização, então o desenvolvedor pode se focar somente no desenvolvimento do projeto em si. Nenhum dos projetos além do AnimAR, oferecem suporte à Cardboard ou algum tipo de head-mounted display.

4 CONCLUSÕES

Este trabalho mostrou o desenvolvimento de uma ferramenta para a criação de cenários animados, utilizando Realidade Aumentada e Interfaces Tangíveis, em que objetos do mundo real interagem com objetos virtuais, permitindo o usuário utilizá-los para manipular o cenário virtual.

O objetivo de disponibilizar uma nova maneira de se criar animações, junto com seus objetivos específicos, foram atingidos e comprovados com a realização dos testes, com exceção do objetivo de disponibilizar a aplicação com o uso do Cardboard. Mesmo que o objetivo de se disponibilizar o uso do Cardboard tenha sido atingido, os resultados não se mostraram satisfatórios com os equipamentos de Cardboard utilizados.

Os testes foram realizados com um grupo pequeno de alunos, mas apesar disso foi possível obter resultados satisfatórios. Os alunos mostraram-se interessados no funcionamento da aplicação, conseguindo realizar os objetivos propostos pela ferramenta, ainda que com alguma dificuldade. A Realidade Aumentada juntamente com a Interface Tangível, possibilitaram a criação de cenários e animações, permitindo um novo nível de interação do usuário com a ferramenta. Este tipo de interação, evidenciou-se ser mais fácil para usuários não técnicos, que não possuem conhecimento de ferramentas de modelagem 3D e animação, a criarem mundos virtuais e dar vida aos mesmos através de pequenas animações.

As ferramentas e bibliotecas utilizadas no desenvolvimento desta ferramenta mostraram-se adequadas. A *engine* Unity, bem consagrada no ramo de aplicações gráficas, mostra-se eficiente e de fácil entendimento, sendo o motor principal desta ferramenta, cuidando da parte de renderização e de dar vida à aplicação. Possui também um ótimo motor físico integrado, facilitando e muito a detecção de interação entre os marcadores. O Vuforia fez toda a integração da RA com o Unity, possuindo suporte nativo dentro do mesmo, cuidando de tudo, desde a utilização da câmera do celular, do reconhecimento dos marcadores, até a ativação da renderização dos objetos gráficos envolvidos com os marcadores. A aplicação AR Marker Generator by Brosvision mostrou-se eficiente e prática no auxílio da criação dos marcadores utilizado pelo AnimAR, gerando padrões com certa aleatoriedade, sendo perfeitos para o motor de reconhecimento do Vuforia.

A contribuição social deixada por este trabalho se dá pela possibilidade da utilização da ferramenta na área da Pedagogia, podendo ser trabalhada a criatividade em sala de aula, ajudando no desenvolvimento das crianças. Por fim, este trabalho deixa como contribuição científica a evolução tecnológica em relação ao trabalho de Lee et al., utilizando Unity e

Vuforia para a criação de aplicações de RA e IUT, soluções de código encontradas para resolver problemas como a de captura de movimentos para registrar uma animação e toda a fundamentação teórica sobre o assunto.

4.1 EXTENSÕES

Para trabalhos futuros de extensões e melhorias, sugere-se:

- a) salvar dados do projeto na nuvem, para que possa compartilhar as cenas e animações criadas com outros dispositivos;
- b) adicionar um controle de linha do tempo melhor, permitindo-se por exemplo, iniciar uma gravação a partir de um determinado tempo X;
- c) permitir excluir pedaços do começo e do final das animações gravadas;
- d) criar uma função para exportar a animação em um formato que possa ser utilizado em outros programas de animação, como o Blender, ou até mesmo o Unity;
- e) adicionar novos tipos de objetos virtuais para serem adicionados, como iluminação;
- f) adicionar novos tipos de terrenos para a criação das cenas;
- g) permitir a execução de todas as animações de todas as cenas sequencialmente, como se fosse um “filme” com várias cenas;
- h) permitir a gravação de sons e aplica-los nas animações;
- i) testar a utilização do modo de Realidade Virtual/Cardboard com outros dispositivos, específicos para o uso de Realidade Virtual.

REFERÊNCIAS

- BRANDÃO, Luis R. G. Jogos Cinematográficos ou Filmes Interativos? A semiótica e a interatividade da linguagem cinematográfica nos jogos eletrônicos. In: SIMPÓSIO BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL, 11., 2012, Brasília. **Anais...** Brasília: Centro de Convenções Ulysses Guimarães, 2012. p. 165-174.
- BROSVISION. **AUGMENTED REALITY MARKER GENERATOR**. 2018. Disponível em: < <http://www.brosvision.com/ar-marker-generator/>>. Acesso em: 04 jun. 2018.
- GIARDINA, Carolyn. **As the Demand for Visual Effects Grows, a Shortage of Artists Looms Ahead**. [S.l.], 2016. Disponível em: <<http://www.hollywoodreporter.com/behind-screen/as-demand-visual-effects-grows-888415>>. Acesso em: 07 set. 2017.
- KIRNER, Claudio et al. **Fundamentos e Tecnologia de Realidade Virtual e Aumentada**. Belém, PA: [s.n.], 2006.
- KIRNER, Claudio; SISCOOTTO, Robson. **Realidade Virtual e Aumentada: Conceitos, Projeto e Aplicações**. Petrópolis, RJ: [s.n.], 2007.
- LEE, Gun A et al. **Immersive Authoring of Tangible Augmented Reality Applications**. Washington, DC, U.S.A: IEEE Computer Society. Washington, 2004. Disponível em: <https://ir.canterbury.ac.nz/bitstream/handle/10092/2309/12594683_2004-ISMAR-Iatar.pdf>. Acesso em: 07 set. 2017.
- MENACHE, Alberto. **Understanding Motion Capture for Computer Animation**. Burlington, Massachusetts, Morgan Kaufmann Publishers, 2011.
- PARENT, Rick. **Computer Animation: Algorithms and Techniques**. San Francisco, Morgan Kaufmann Publishers, 2001. Disponível em: <https://s3.amazonaws.com/academia.edu.documents/40786681/Computer_Animation-Algorithms_and_Techniques_.pdf?AWSAccessKeyId=AKIAIWOWYYGZ2Y53UL3A&Expires=1505093950&Signature=Hxl1OX6bCJBsAAVAKYOETbVDkik%3D&response-content-disposition=inline%3B%20filename%3DComputer_Animation_Algorithms_and_Techni.pdf>. Acesso em: 10 set. 2017.
- SCHMITZ, Evandro M. **Desenvolvimento de uma ferramenta para auxiliar no Ensino do Sistema Solar utilizando Realidade Aumentada**. 2017. 94f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.
- SILVA, Antônio M. da. **VISEDU: Interface de Usuário Tangível utilizando Realidade Aumentada e Unity**. 2016. 75f. Trabalho de Conclusão de Curso (Bacharel em Ciência da Computação) – Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.
- ULLMER, Brygg.; ISHII, Hiroshi. Emerging frameworks for tangible user interfaces. In: CARROL, John M. (Ed.). **Human-Computer Interaction in the New Millenium**. Ann Arbor, MI, U.S.A: University of Michigan. Ann Arbor, 2001. p. 579-601.
- VUFORIA. **Getting Started with Vuforia in Unity**. 2018. Disponível em: <<https://library.vuforia.com/articles/Training/getting-started-with-vuforia-in-unity.html>>. Acesso em: 04 jun. 2018.

VUFORIA. **Optimizing Target Detection and Tracking Stability**. 2018. Disponível em: <<https://library.vuforia.com/articles/Solution/Optimizing-Target-Detection-and-Tracking-Stability.html>> Acesso em: 04 jun. 2018.