UNIVERSIDADE FEDERAL DE SANTA CATARINA

Jaime Paz Lopes

Desenvolvimento de um jogo educacional de Realidade Aumentada

Florianópolis - SC

2013/1

UNIVERSIDADE FEDERAL DE SANTA CATARINA
CENTRO TECNOLÓGICO - CTC
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA - INE

Jaime Paz Lopes

Desenvolvimento de um jogo educacional de Realidade Aumentada

Trabalho de conclusão de curso apresentado como parte dos requisitos para obtenção do grau de Bacharel em Ciências da Computação.

Orientador
Professor José Eduardo de Lucca

Florianópolis - SC
2013/1

Índice de ilustrações

Sumário

RESUMO

O presente trabalho trata do desenvolvimento de um jogo educacional utilizando a Realidade Aumentada como interface homem-máquina, tendo como consequência despertar maior interesse no estudante, devido ao alto grau de interatividade proporcionada pelo uso desta tecnologia. Este jogo será incluído no projeto Univerciência, que é um mundo educacional virtual focado no Ensino Básico.

O desenvolvimento do projeto parte de uma pesquisa bibliográfica como técnica para a coleta de dados, através de um levantamento bibliográfico exploratório em diversas fontes, a fim de adquirir conhecimento da tecnologia da Realidade Aumentada, familiarizando-se, primeiramente, com a parte teórica desta tecnologia.

Desta forma, são apresentadas algumas considerações teóricas da Realidade Aumentada, tais como, definição, características, vantagens e diversidades de utilização, para se chegar a alguns aspectos de seu uso na área de jogos digitais, mais especificamente, os educacionais.

Após esta parte explanatória, são especificadas as etapas de desenvolvimento de um jogo digital educacional utilizando a Realidade Aumentada, ou seja, são relatadas as principais decisões tomadas no decorrer do processo de desenvolvimento do jogo, com suas devidas explicações, além de algumas descrições mais detalhadas a respeito dos elementos essenciais que compõem a sua arquitetura.

Palavras-chave: Realidade Aumentada, Desenvolvimento de Jogos, Jogos na Educação.

1 INTRODUÇÃO

O tema deste trabalho nasce do interesse de se incorporar recursos de Realidade Aumentada ao jogo Universo de Ciências (Univerciência), que é um projeto educacional centrado na educação, formal e não formal, de Ciências (Física, Biologia e Química) no Ensino Básico, direcionado a alunos de doze a quatorze anos.

O Univerciência vem sendo desenvolvido em parceria entre a empresa Mentes Brilhantes Brinquedos Inteligentes e o GeNESS, Centro de Geração de Novos Empreendimentos em Software e Serviços da Universidade Federal de Santa Catarina, com o apoio financeiro do CNPq, edital MCT/CNPq Nº 62/2009 RHAE – Pesquisador na Empresa, projeto 561723/2010-9.

O jogo Univerciência consiste em um mundo virtual formado por vários minijogos que serão explorados pelos alunos para reforçar a assimilação dos conteúdos apresentados em aula.

Há muito tempo que os jogos têm sido usados no campo da aprendizagem, seja jogos utilizando cartas, tabuleiros e dados das mais variadas complexidades até os jogos computadorizados (ZORZAL et al., 2005; ZORZAL et al., 2006). Eles podem ser utilizados como ferramenta de ensino em função do seu conhecido potencial para estimular o jogador a concentrar-se (MAZZAROTTO, 2009), despertando e incentivando o interesse pelo conteúdo didático, e, consequentemente, resultando em aprendizado.

O interesse por jogos digitais vem crescendo muito. Segundo Mota (2007), as crianças, desde a primeira infância, aprendem a manusear um computador, ou seja, bem antes de aprender a ler e escrever. Com o avanço da tecnologia, foram criadas muitas formas de interação com os computadores. Recentemente, vem sendo dado destaque a formas de interação que adotam técnicas de Realidade Aumentada (EXAME, 2013), onde o usuário recorre a objetos reais para interagir com o computador de uma forma simples e natural (SANTIN, 2004; KIRNER, 2013), transportando o ambiente virtual para o seu espaço físico. A interação por Realidade Aumentada acontece em tempo real, decorrente da combinação de elementos virtuais e reais (AZUMA, 1997).

Essa combinação de fatores transforma a Realidade Aumentada em uma excelente ferramenta para estimular a utilização de jogos. Por isso, a ideia de incorporar recursos de

Realidade Aumentada ao Univerciência, para atrair o aluno e facilitar a absorção do conteúdo didático.

Partindo desse interesse, este projeto consiste no desenvolvimento de um jogo digital educacional que utilize a tecnologia da Realidade Aumentada para torná-lo mais imersivo e interativo, atraindo a atenção dos estudantes, além de servir de subsídio para o uso dessa tecnologia no jogo Univerciência.

1.1 OBJETIVO GERAL

- Desenvolver um jogo educacional utilizando a tecnologia da Realidade Aumentada.

1.2 OBJETIVOS ESPECÍFICOS

- Pesquisar, identificar e estudar considerações teóricas e técnicas existentes relativas a Realidade Aumentada.
- Levantar e analisar tecnologias de Realidade Aumentada para verificação das possibilidades de utilização.
- Identificar ferramentas para o desenvolvimento de programas em Realidade Aumentada.
- Realizar testes para definir a biblioteca de Realidade Aumentada a ser utilizada.
- Demonstrar a implementação de um jogo educacional em Realidade Aumentada.

1.3 ESCOPO

Este trabalho consiste no estudo e utilização da Realidade Aumentada com a finalidade de desenvolver um jogo educacional.

Partindo do estudo da bibliografia existente e identificando suas diversas utilizações, chega-se à escolha de soluções e métodos que serão empregados no desenvolvimento do jogo aqui proposto, que terá uma temática educacional e adotará uma abordagem não-formal, ou seja, não relacionada a um tema curricular formal.

É importante destacar que o enfoque do desenvolvimento do jogo em questão é na programação de suas funcionalidades e na forma de utilização da Realidade Aumentada, pelo que os objetos tridimensionais e as imagens usadas têm apenas o objetivo de ilustrar o que é exposto.

1.4 METODOLOGIA

A metodologia utilizada será a de pesquisa bibliográfica do tipo exploratória, com o fim de se obter subsídios para o desenvolvimento de um jogo digital educacional utilizando-se a Realidade Aumentada.

Num segundo momento, será modelado e desenvolvido um jogo digital educacional para exploração prática dos conceitos levantados. Esse jogo será desenvolvido com base no método de criação de jogos descritos por Schuytema (2008): pré-produção, produção e pós-produção.

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os fundamentos teóricos identificados na etapa de pesquisa e que foram utilizados na formulação deste trabalho.

2.1 JOGOS

Segundo Schuytema (2008), um jogo é uma atividade lúdica composta por uma série de ações e decisões que levam ao final proposto. Ele é limitado por regras, que criam situações interessantes para desafiar o jogador, e pelo universo do jogo, que existe para proporcionar uma estrutura e um contexto para as ações do jogador.

No caso de jogos eletrônicos, as regras e o universo do jogo são apresentados e controlados por um programa que é desenvolvido para, a cada ação do jogador, responder com uma reação, desencadeando um novo desafio a ser superado.

2.1.1 Jogos na educação

Os jogos vêm sendo amplamente usados como ferramenta de apoio à educação, principalmente pelo seu potencial lúdico, motivacional e interativo (BORGES, 2005; SOUZA et al., 2010). São muito bem aceitos pelo público jovem, tanto para entretenimento quanto como ferramenta de educação, o que torna o processo de sua utilização na educação uma evolução natural (TIMM et al., 2008).

No entanto, sua evolução na educação não está livre de problemas. Alguns deles são: a falta de diversão dos jogos pedagógicos (COSTA, 2009) e a utilização incorreta do jogo pelo professor (PASSERINO, 1999).

Como forma de entretenimento, estão tão amplamente difundidos na sociedade que é quase certo que um aluno note se um jogo educacional cumpre minimamente o quesito diversão. Muitas vezes, um jogo educacional parece ser feito simplesmente para contemplar o conteúdo educacional e não para ser também atrativo para o aluno. Como consequência, ele sente-se desmotivado, o que acaba tanto por dificultar sua compreensão acerca do conteúdo, quanto por trazer efeitos negativos em relação à sua aprendizagem (COSTA, 2009). Por essa razão, um jogo educacional deve ser divertido, resultando em um melhor aproveitamento do conteúdo educacional a ser trabalhado.

Outro ponto importante é que um jogo educacional deve servir como ferramenta de apoio ao professor; sua utilização deve ser guiada por este, pois, alguns jogos podem exigir um grau de conhecimento prévio, ainda não desenvolvido pelo aluno, ou porque, simplesmente, dúvidas e dificuldades podem aparecer para este no seu decorrer (COSTA, 2009; PASSERINO, 1999).

2.1.2 Desenvolvimento de Jogos

Jogos podem ser desenvolvidos por grandes empresas ou até mesmo por uma pessoa em suas horas vagas, pois esse é um mercado gigante (THEESA, 2012) que aceita muito bem tanto os independentes (HUMBLEBUNDLE, 2013) como aqueles que já possuem uma marca consolidada, de empresas com renome mundial. Em alguns casos, os desenvolvedores não possuem o dinheiro necessário para criar o jogo e procuram publicadoras, que são as empresas responsáveis por gerenciar toda a parte de propaganda, publicação e distribuição de um jogo, e, algumas vezes, chegam a financiar parcial ou totalmente sua criação (IGN, 2006).

Esta seção descreve as etapas fundamentais do processo de desenvolvimento de um jogo, também adotadas para a elaboração deste trabalho. Geralmente, tal processo inclui três etapas: pré-produção, produção e pós-produção (SCHUYTEMA, 2008).

2.1.2.1 Pré-produção

O desenvolvimento de um jogo começa pela pré-produção, que é a elaboração da proposta do mesmo. No caso de jogos comerciais, as ideias que compõem a proposta do jogo geralmente nascem de uma dada necessidade do mercado, que pode ser para atender a demanda de um estilo específico de jogo, um novo nicho de público ou a pedido de alguma publicadora (CHANDLER, 2009).

Geralmente, é responsabilidade da Equipe de Game Design (grupo que descreve com detalhes como o jogo deve se comportar) criar o documento de Proposta do Jogo, que contém o seu conceito, explicação da jogabilidade, todas as funcionalidades do sistema, história, público-alvo, requisitos, levantamento de pessoal e custo estimado. Quando aprovado, esse documento é usado como base na elaboração do Game Design Document, que é o principal documento criado na produção de um jogo, pois nele se encontram todas as informações e decisões que foram tomadas na criação do jogo. Além disso, ele é considerado um documento "vivo", já que suas informações são constantemente alteradas, adicionadas ou retiradas de acordo com as necessidades do projeto durante todas as etapas de seu desenvolvimento (SCHUYTEMA, 2008).

Na etapa de pré-produção, também são criados pequenos protótipos que servem de prova de conceito do jogo. Essa experimentação é muito importante, pois nela se verifica a viabilidade das ideias criadas, podendo, assim, ser eventualmente alteradas ou retiradas (BRATHWAITE; SCHREIBER, 2009).

2.1.2.2 Produção

A segunda etapa de desenvolvimento do jogo é a produção, onde todo o seu conteúdo é criado: código fonte, imagens, texturas, modelos 3D, história, diálogos, animações, localização, mapas, músicas, entre outros (CHANDLER, 2009). Por isso, considera-se a produção como a etapa principal e a mais demorada de todo o processo. Outro ponto muito importante dessa fase

são os testes, que passam a ser feitos assim que se inicia a produção do jogo. Tudo é testado exaustivamente, mesmo que seja algo aparentemente muito simples, como a movimentação do personagem ou a navegação nos menus, tentando-se encontrar o maior número de problemas possível. Inicialmente, os testes praticamente não consomem tempo de desenvolvimento do jogo, mas essa situação pode se modificar à medida que a processo de produção se aproxima do fim, sobretudo na etapa de desenvolvimento de funcionalidades novas, que podem afetar alguma outra já desenvolvida (SHUYTEMA, 2008).

2.1.2.3 Pós-produção

A última etapa de desenvolvimento do jogo é a pós-produção, que começa com o seu lançamento. Mesmo depois de ser lançado, ele continua a ser desenvolvido, pois problemas ainda podem ser encontrados depois de seu lançamento, o que requer correções que devem ser disponibilizadas aos jogadores na forma de atualizações. Além disso, alguns jogos, por possuírem opção *online*, precisam de manutenção dos servidores para garantia de qualidade (MOORE; NOVAK, 2010).

Uma prática utilizada na tentativa de aumentar a vida útil do jogo, mesmo após seu lançamento, é a criação de uma equipe de desenvolvimento que continue adicionando novos elementos, como cenários ou histórias, lançados como atualizações, podendo ou não ser cobrados adicionalmente (SHUYTEMA, 2008).

2.2 REALIDADE AUMENTADA (RA)

Antes de definir Realidade Aumentada, é necessário explicar o contexto onde ela se encontra, que é o da Realidade Misturada, pois RA nada mais é do que um tipo de Realidade Misturada (ZORZAL et al, 2005).

Realidade Misturada é um tipo de ambiente que combna o mundo real com um mundo virtual. No caso da RA, o mundo predominante é o mundo real (MILGRAM, 1994). Quando o ambiente predominante é o virtual, classifica-se como Virtualidade Aumentada. A Ilustração 1 mostra o Diagrama de Realidade/Virtualidade Contínua.

Ilustração 1: Diagrama de Realidade/Virtualidade Contínua.
Fonte: Milgram (1994)

Esses ambientes de Realidade Misturada são computacionalmente produzidos utilizando-se *hardwares*, como câmeras ligadas a computadores, e *softwares*, que geram objetos tridimensionais virtuais sobrepostos nas imagens capturadas pela câmera, mostrando o resultado obtido para o usuário do sistema. Essas câmeras podem ser simples *webcams* integradas a um *notebook* até capacetes sofisticados, para aumentar a imersão no mundo gerado (AZUMA, 97).

A Ilustração 2 mostra um ambiente de Realidade Aumentada. As paredes, a mesa e o telefone são reais, foram capturados por uma câmera e enviados para o computador, que adicionou as duas cadeiras e o abajur.

Ilustração 2: Ambiente de Realidade Aumentada
Fonte: Azuma (1997)

Segundo Azuma (1997), um sistema de Realidade Aumentada deve possuir três características:

– combinar o ambiente real com elementos virtuais: o ambiente de Realidade Aumentada é gerado tendo como base o ambiente real, que é preenchido com objetos virtuais que o complementam;

– interativo em tempo real: esses ambientes devem oferecer meios para que o usuário possa realizar ações e observar as reações produzidas imediatamente; e

– apresentado em três dimensões: os objetos devem ser modelos tridimensionais que se mesclam da melhor forma possível no ambiente real, devendo aumentar, diminuir e girar de acordo com as necessidades do usuário.

2.2.1 Interação

Um dos principais diferenciais de desenvolver um programa com interface de comunicação em Realidade Aumentada é o ganho de interatividade que essa tecnologia proporciona. Com a RA, a interação com o sistema não é intermediada por teclado e *mouse*, mas

fundamentalmente através de uma câmera que passa as informações geradas pelo usuário para o computador (SANTIN, 2004).

A interação em RA pode ser implementada de diversas formas, sendo a mais comum a utilização de marcadores para permitir a interação do usuário com o ambiente de Realidade Aumentada. A Ilustração 3 mostra um exemplo de marcador contendo uma imagem em preto e branco, que, ao ser captada por uma câmera, modifica o ambiente de RA, de acordo com as ações predefinidas.

Ilustração 3: Exemplo de marcador
Fonte: NyARToolKit (2013)

Um dos usos mais comuns de marcadores é associá-los a um objeto virtual 3D, o que torna possível ao usuário alterar, por exemplo, as propriedades de posição, ângulo e proporção do objeto 3D no ambiente de RA, manipulando de forma correspondente a posição, ângulo e proporção do marcador no ambiente real. Outra prática comum é associar um marcador a uma função, que pode ser, por exemplo, a de destruir todos os modelos de objetos 3D no ambiente ou até a de fechar o programa (SANTIN, 2006). A Ilustração 4 exibe um exemplo de manipulação de um objeto virtual 3D com um marcador.

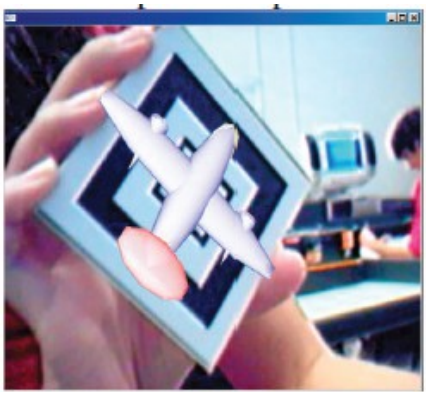


Ilustração 4: Manipulação de objeto virtual com marcador
Fonte: Santin (2004)

Para além do uso de marcadores, a interação em RA pode ser muito mais elaborada, tal como em alguns sistemas que podem oferecer luvas que captam os movimentos do usuário. Em outros ambientes, a forma de interação pode ser completamente livre de objetos físicos, o usuário, através de seus movimentos naturais, interage com os objetos virtuais (AZUMA, 1997).

2.2.2 Tipos de *Display*

Umas das possíveis classificações dos tipos de sistemas de Realidade Aumentada é quanto ao tipo de *Display* que é utilizado. Essa classificação baseia-se na forma de exibição do ambiente de Realidade Aumentada para o usuário, levando em conta a posição dos dispositivos usados e a forma que o usuário interage com eles.

2.2.2.1 *Display* em capacetes

Nesse tipo de visualização usa-se um capacete com uma câmera acoplada que projeta as imagens do ambiente diretamente na frente do usuário.

2.2.2.1.1 Sistemas de visão ótica direta

Utilizam lentes transparentes através das quais o usuário vê diretamente o mundo real e, simultaneamente, visualiza projeções de imagens virtuais. Desta forma, o que o usuário vê é o objeto virtual sobreposto ao ambiente real. A maneira mais comum de se conseguir esse efeito é usando uma lente inclinada que reflita as imagens do computador diretamente na visão do usuário (AZUMA et al., 2001). A Ilustração 5 esquematiza esse tipo de *Display*.

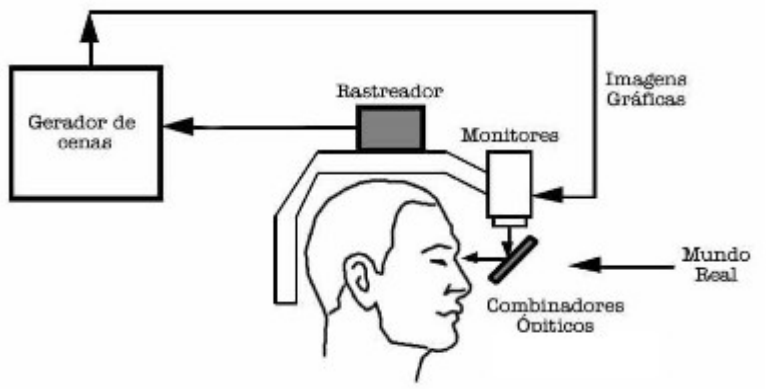


Ilustração 5: Diagrama de funcionamento do sistema de visão ótica direta.
Fonte: Azuma (1997) apud Kirner (2013)

2.2.2.1.2 Sistemas de visão direta por vídeo

Nesses sistemas, a câmera do capacete capta o mundo real, as cenas são computacionalmente misturadas com os elementos virtuais e exibidas em monitores que ficam bem na frente dos olhos do usuário (AZUMA et al. 2001), como mostrado na Ilustração 6.

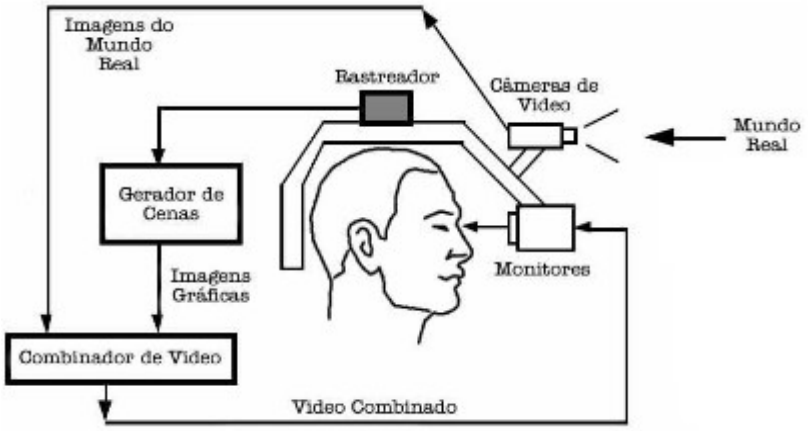


Ilustração 6: Diagrama de funcionamento do sistema de visão direta por vídeo.
Fonte: Azuma (1997) apud Kirner (2013)

2.2.2.2 *Displays* portáteis

Alguns sistemas preveem os dispositivos de tela e câmera em um mesmo suporte que o usuário pode mover para visualizar o mundo sob diferentes ângulos, de maneira similar a uma lupa (AZUMA et al. 2001). Nesse tipo de sistema é possível utilizar celulares com câmeras, como mostrado na Ilustração 7. Nesse caso, o copo é usado como marcador para que uma pessoa visualize uma animação em Realidade Aumentada através de um dispositivo móvel.

Ilustração 7: Animação em realidade aumentada em copo de café.
Fonte: Starbucks (2011)

2.2.2.3 Monitores

Sistemas utilizam uma *webcam* para capturar as imagens do mundo real, que são misturadas com elementos do virtual em um computador, mostrando o resultado em um monitor (AZUMA et al. 2001). A Ilustração 8 exibe um esquema do seu funcionamento.

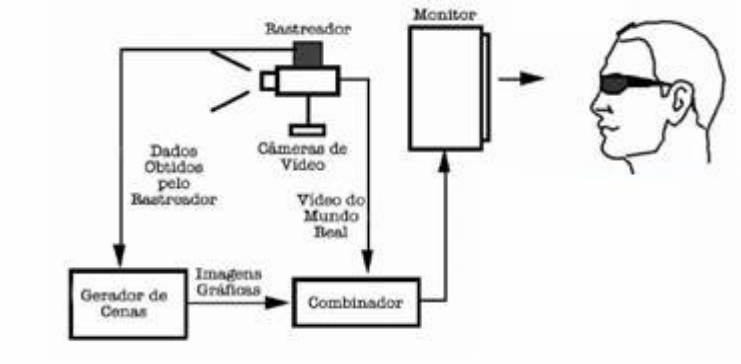


Ilustração 8: Diagrama de funcionamento utilizando monitores.
Fonte: Azuma (1997) apud Kirner (2013)

2.2.2.4 Projeção

Nesse tipo de *Display,* os elementos virtuais são projetados diretamente no mundo real. O projetor pode estar em um capacete projetando os elementos virtuais especificamente no campo de visão do usuário. Esses elementos devem responder de forma realística à movimentação do usuário, alterando sua rotação e escala de acordo com o ângulo que o usuário assume. De forma alternativa, o projetor pode ser estático, projetando sempre de um mesmo ângulo de visão (AZUMA et al., 2001).

2.2.3 Aplicações em Realidade Aumentada

Aqui são mostradas diversas aplicações existentes de Realidade Aumentada. O primeiro enfoque dedica-se a descrever aplicativos que usam essa tecnologia, e, posteriormente, são citados e detalhados jogos de Realidade Aumentada, que foram classificados em tradicionais e comerciais.

2.2.3.1 Aplicativos em Realidade Aumentada

Esta seção apresenta um conjunto de aplicativos que ilustra algumas possibilidades de utilização da RA.

2.2.3.1.1 Butlers ViewAR

A empresa ViewAR desenvolve soluções em Realidade Aumentada para *design* de interior e arquitetura. Em um de seus aplicativos para celulares e *tablets*, permite que o usuário tire fotos dos ambientes de uma casa e nelas posicione móveis virtuais. Em um cenário de uso desse aplicativo, o usuário pode criar modelos virtuais idênticos aos móveis de sua casa para planejar de que maneira deseja que determinado ambiente esteja organizado (VIEWAR, 2013). A Ilustração 9 mostra o resultado do posicionamento de um objeto virtual em um cômodo real.

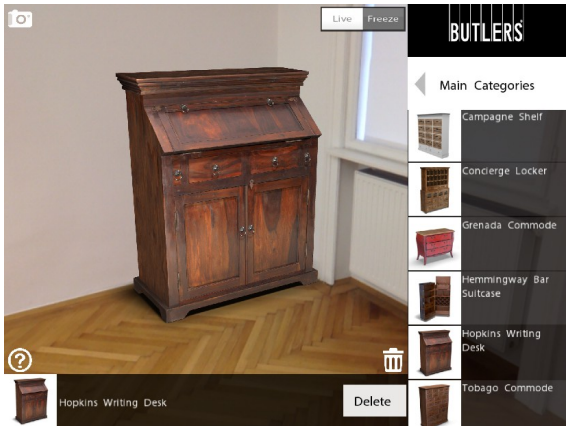


Ilustração 9: Posicionando um armário virtual em uma imagem de uma sala real.
Fonte: ViewAR (2013)

2.2.3.1.2 Lego Digital Box

A Lego, empresa que produz e vende blocos de montar, desenvolveu um sistema que usa a própria caixa do produto como marcador, tornando possível a visualização do brinquedo já montado antes da compra (INTEL, 2013). A Ilustração 10 mostra uma caixa de Lego que exibe como o brinquedo fica depois de montado.

Ilustração 10: Modelo virtual de um caminhão de Lego sendo visualizado
Fonte: Gizmodo (2013)

2.2.3.1.3 WebCam Social Shopper

O WebCam Social Shopper é um produto destinado a lojas de roupas *online*. Como não é possível provar uma roupa *online,* esse sistema faz uma simulação, sobrepondo uma roupa virtual no corpo do usuário (ZUGARA, 2013). A ilustração 11 mostra uma mulher provando uma peça de roupa; os botões na tela são sensíveis à movimentação do usuário, bastando posicionar a mão sobre eles.

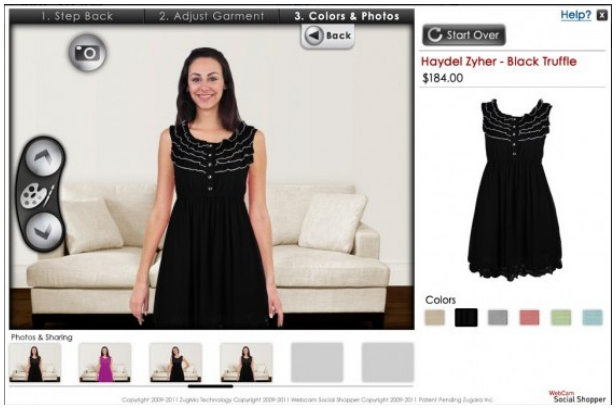


Ilustração 11: Modelo real provando uma roupa virtual
Fonte: Zugara (2013)

2.2.3.1.4 Letters Alive

É um programa que auxilia na alfabetização de crianças. Ele associa os cartões com letras a modelos tridimensionais de animais, que aparecem quando o cartão é mostrado para a câmera (LOGICALCHOICES, 2013). A Ilustração 12 exemplifica esse funcionamento. Esse sistema também exibe animações e emite os sons dos animais.

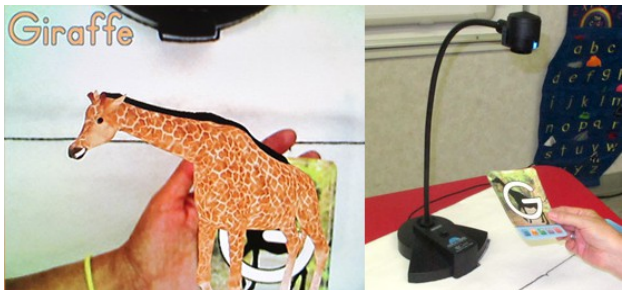


Ilustração 12: Cartão da girafa sendo mostrado para a câmera.
Fonte: LAWRENCEVILLE, 2010

2.2.3.1.5 New Horizon

Esse produto é um livro voltado ao ensino de inglês para japoneses. Algumas páginas das lições do livro possuem marcadores que dão ao aluno acesso a uma animação a respeito do assunto abordado (BANDAI, 2013). A Ilustração 13 mostra um usuário assistindo ao vídeo de uma lição em um celular. Ao fundo, é possível visualizar o marcador de cor azul com o texto "Try it".

Ilustração 13: Imagem de uma das animações do livro
Fonte: Bandai, 2013

2.2.3.2 Jogos em Realidade Aumentada

Esta seção apresenta jogos que utilizam a tecnologia de Realidade Aumentada, trazendo também informações sobre seu funcionamento.

2.2.3.2.1 Jogos tradicionais

Aqui são apresentadas versões em Realidade Aumentada de jogos tradicionais amplamente conhecidos.

2.2.3.2.1.1 Quebra-Cabeça 3D

De modo geral, um quebra-cabeça convencional em duas dimensões (2D) é formado por pequenas peças em formatos diferentes que, ao serem encaixadas umas às outras, formam uma imagem 2D.

No quebra-cabeça 3D em Realidade Aumentada, cada marcador simboliza uma parte de um objeto 3D computadorizado. O jogador deve organizar esses marcadores até que o objeto esteja completo (ZORZAL et al., 2005).

A Ilustração 14 mostra um exemplo de quebra-cabeça 3D em Realidade Aumentada, em que cada parte do modelo 3D está associada a um cubo, e cada face do cubo está associada a diferentes posições e tamanhos de uma parte do modelo 3D, tornando assim mais complexo o problema de montar o quebra-cabeça. A Ilustração 15 mostra o mesmo quebra-cabeça sendo montado.

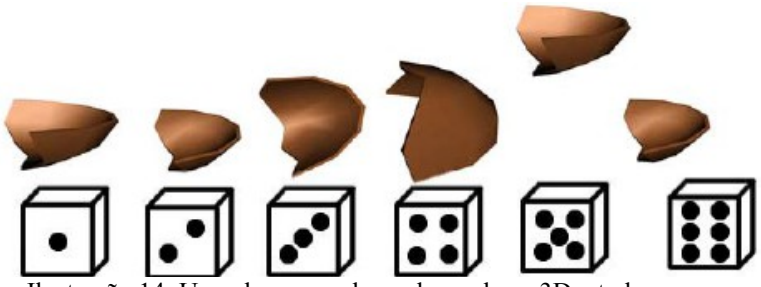


Ilustração 14: Uma das peças do quebra-cabeça 3D e todas as suas variações.
Fonte: Zorzal et al. (2005)

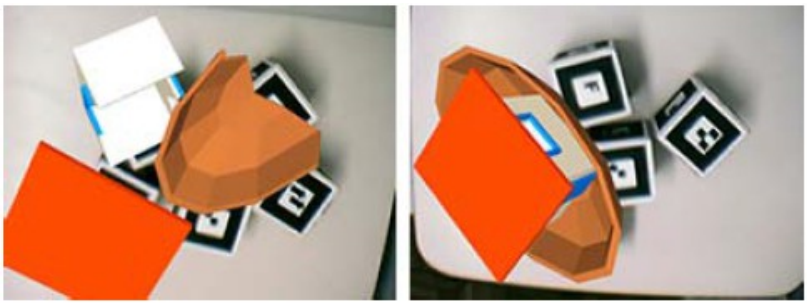


Ilustração 15: Quebra-Cabeça durante e após a montagem.
Fonte: Zorzal et al. (2005)

2.2.3.2.1.2 Quebra-Cabeça Ordenador

Neste tipo de quebra-cabeça as peças são embaralhadas em um quadrado, de tal forma que cada uma esteja cercada por outras quatro à sua volta (em cima, embaixo, à esquerda e à direita), com exceção daquelas que estejam nas bordas do quadrado. Em um dos cantos do quadrado não haverá peça, sendo este espaço utilizado pelo jogador para a movimentação de alguma peça adjacente. Para iniciar o jogo, desloca-se a peça adjacente para o espaço vazio, criando-se um novo espaço vazio e uma nova configuração de peças no quebra-cabeça; repete-se o passo anterior com outra peça adjacente ao novo espaço vazio, gerando uma nova configuração do quebra-cabeça e assim sucessivamente. O jogo continua até o jogador formar uma configuração do quebra-cabeça com a sequência de números ou símbolos correta.

Em sua versão em Realidade Aumentada, cada uma das peças consiste em um marcador que pode representar, por exemplo, letras, números ou pedaços de um objeto (ZORZAL et al., 2006). A Ilustração 16 mostra como é um quebra-cabeça ordenador sem os objetos virtuais e um quebra-cabeça ordenador em uso.

Ilustração 16: Funcionamento do quebra-cabeça ordenador.
Fonte: Zorzal et al., (2006)

2.2.3.2.1.3 Torre de Hanói

A Torre de Hanói é um problema de lógica clássico e também já tem uma versão desenvolvida em Realidade Aumentada. Esse problema é composto por três torres verticais e alguns discos de tamanhos diferentes, que são inicialmente ordenados em uma das torres, de tal forma que o maior disco fique embaixo e o menor em cima. O objetivo do jogador é mover todos esses discos para uma outra torre mantendo a mesma ordem do início, mas ele só pode mover um disco por vez e nunca pode deixar que um disco maior fique em cima de um disco menor.

Em sua versão em Realidade Aumentada o jogo da Torre de Hanói é constituído de quatro cubos, cada um com quatro marcadores, sendo que cada cubo representa um disco e cada marcador coloca o disco em uma altura diferente (ZORZAL et al., 2008), como mostrado na Ilustração 17. A Ilustração 18 mostra como o jogo é na sua posição inicial e depois durante seu desenvolvimento.

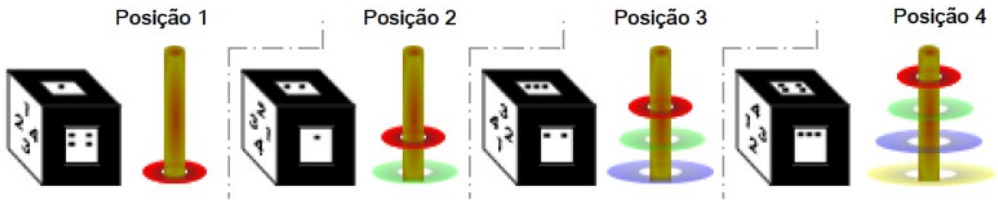


Ilustração 17: Posição do disco vermelho em relação a face do cubo.
Fonte: Zorzal et al. (2008)

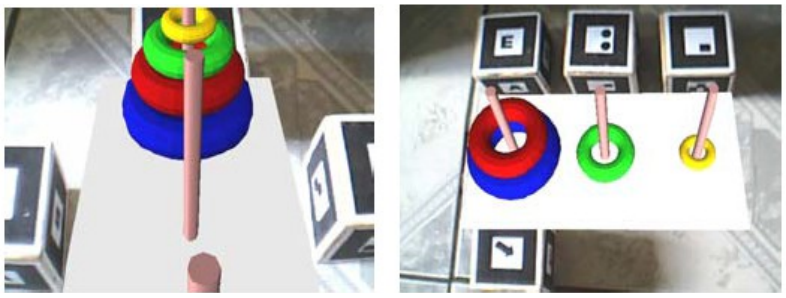


Ilustração 18: Jogo durante sua utilização
Fonte: Zorzal et al. (2008)

2.2.3.2.1.4 Cubo Mágico

Este brinquedo é composto por oito cubos interligados. Cada uma de suas faces possui um pedaço de uma imagem. O objetivo do usuário é manipular os cubos reposicionando-os de tal forma que as faces dos cubos unidas formem imagens diferentes. Cada uma dessas imagens é uma parte de uma história, assim, o jogador precisa também monta-las na ordem correta para compreender o roteiro.

Em sua versão em Realidade Aumentada, cada uma das imagens formadas pelo posicionamento dos cubos é um marcador, que ao ser mostrado para a câmera gera uma animação que representa uma parte da história. Em uma de suas versões, o cubo conta a história de Davi e Golias, e a cada novo posicionamento dos cubos o usuário visualiza um cenário e escuta uma passagem da história (ZORZAL, 2005). A Ilustração 19 mostra uma pessoa usando o cubo e uma das cena da história.

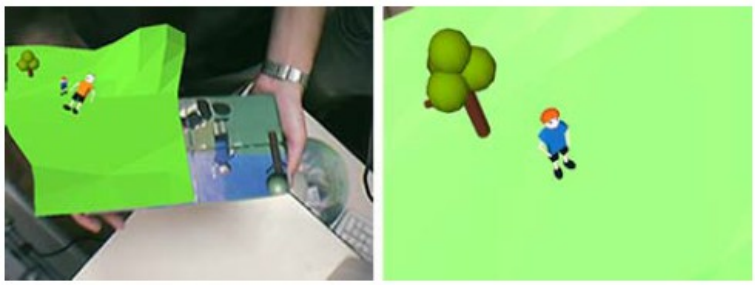


Ilustração 19: Cubo Mágico sendo utilizado.
Fonte: Zorzal (2005)

2.2.3.2.1.5 Jogo de Palavras

Neste jogo, o usuário tem à sua disposição um conjunto de letras que devem ser combinadas para formar palavras conhecidas de uma língua (ZORZAL, 2005).

Em uma versão em Realidade Aumentada, cada letra é um marcador. Quando uma combinação de letras/marcadores gera uma palavra conhecida na língua, e essa palavra está cadastrada no jogo, uma representação virtual dessa palavra é mostrada na tela (ZORZAL, 2005). Nesse jogo é utilizado um sistema de marcadores compostos, onde um marcador sozinho não é reconhecido pelo sistema, mas eles em conjunto possuem representação virtual. A Ilustração 20 mostra alguns exemplos de soluções para um jogo de palavras que considera a língua inglesa. A Ilustração 21 mostra uma pessoa jogando um jogo de palavras em Realidade Aumentada.

Ilustração 20: Exemplo de palavras existentes e seus modelos tridimensionais.
Fonte: Zorzal et al. (2008)

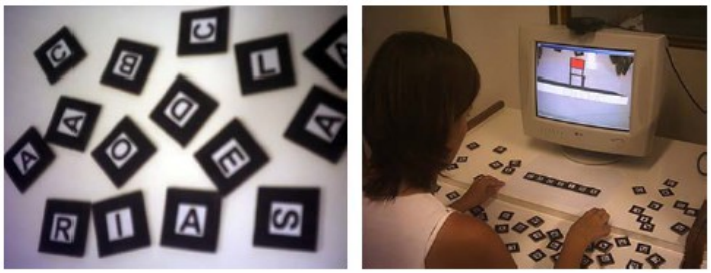


Ilustração 21: Demonstração de como o jogo é jogado.
Fonte: Zorzal et al. (2005)

2.2.3.2.1.6 Jogo da Memória

Na versão convencional deste jogo, diversos pares de peças são inicialmente embaralhados e virados para baixo, sendo que cada jogador, na sua vez, desvira duas peças tentando encontrar quais delas constituem um par.

Na sua versão em Realidade Aumentada todas as peças devem ser colocadas no campo de visão da câmera, sendo que, no lugar das imagens, cada peça possuirá um marcador que será analisado pelo sistema computacional que vai gerar o objeto que o representa (POVIDELO, 2004). A Ilustração 22 mostra alguns exemplos de marcadores e seus respectivos modelos tridimensionais para um jogo da memória em Realidade Aumentada.

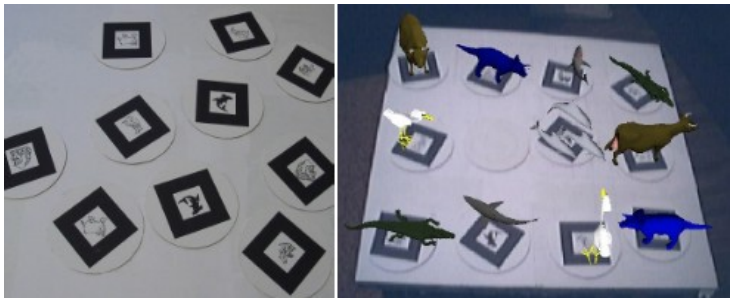


Ilustração 22: Peças do jogo para o jogo da memória em RA
Fonte: Povidelo, 2004

2.2.3.2.2 Jogos Comerciais

Como a indústria de jogos movimenta muito dinheiro e está sempre buscando novas técnicas e tecnologias para conquistar clientes (THEESA, 2012), é esperado que grandes empresas de desenvolvimento de jogos eletrônicos desenvolvam aplicações de Realidade Aumentada. Esta seção apresenta alguns desses jogos.

2.2.3.2.2.1 PS Vita

O PS Vita é um videogame portátil lançado em dezembro de 2011 pela Sony Computer Entertainment (PLAYSTATIONBLOG, 2011). Ele possui câmera embutida que pode ser usada durante os jogos, criando possibilidades de Realidade Aumentada.

Em um dos jogos para PS Vita, o TableTop Tanks (PLAYSTATION, 2013a), publicado pela Sony em maio de 2012, qualquer lugar pode se tornar uma arena de combate entre tanques de guerra. A Ilustração 23 mostra uma sessão de jogo em cima de uma mesa. Basta realizar o escaneamento da área desejada e quando o jogo começar, torna-se possível assumir a direção de um tanque de guerra cuja missão é destruir os outros tanques na arena.

Ilustração 23: Jogo TableTop Tanks em execução em um PS Vita
Fonte: Playstation (2013b)

Em outro jogo, chamado Little Deviants, o jogador deve destruir com um canhão os robôs que aparecem na tela. O mundo do jogo é o ambiente real capturado pela câmera do aparelho, no qual os robôs vão aparecendo aleatoriamente. Para acertá-los, o jogador deve mover o aparelho e posicionar a mira em cima dos alvos (PLAYSTATION, 2013c), tal como mostra a Ilustração 24.

Ilustração 24: Jogo Little Deviant.
Fonte: Playstation (2013c)

2.2.3.2.2.2 Playstation 3

O Playstation 3, lançado em novembro de 2006 pela Sony (ENGADGET, 2006), é capaz de executar jogos em Realidade Aumentada através de uma câmera que pode ser adicionada ao sistema. Um dos principais jogos em RA desse console é o EyePet, onde um animal de estimação é adicionado ao mundo real e o jogador pode usar marcadores para interagir com ele (Sony, 2013). Na Ilustração 25 o jogador usa o controle para mover uma cama elástica.

Ilustração 25: Exemplo de interação com o bicho de estimação no jogo EyePet.
Fonte: Playstation (2013d)

Um dos jogos mais recentes para Playstation 3 que usa a tecnologia de Realidade Aumentada é o Wonderbook: Book of Spells. Neste, o jogador torna-se participante da história apresentada por um livro dotado de marcadores para Realidade Aumentada, como mostrado na Ilustração 26, permitindo ao jogador explorar alguns recursos interativos, tais como magias, disponíveis em diferentes partes da história (PLAYSTATION, 2013e). A Ilustração 27 mostra uma das cenas do jogo.

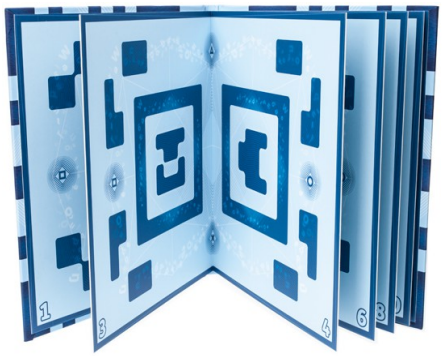


Ilustração 26: Livro usado no jogo
Wonderbook: Book of Spells
Fonte: PCMag (2013)

Ilustração 27: Demonstração de uma das magias do
livro do jogo Wonderbook: Book of Spells.
Fonte: PLAYSTATION, 2013e.

3 CRIAÇÃO DO JOGO

De acordo com o objetivo principal deste trabalho, neste capítulo são abordados os principais pontos sobre a criação de um jogo educacional, denominado Gangorra Espacial. Serão relatadas as principais decisões tomadas no decorrer do processo de desenvolvimento do jogo com suas devidas explicações, além de algumas descrições mais detalhadas a respeito dos elementos essenciais que compõem a sua arquitetura.

3.1 CARACTERÍSTICAS GERAIS DO JOGO

Conforme dito na introdução, o desenvolvimento do presente jogo educacional em Realidade Aumentada se insere na história do mundo virtual do jogo Universo de Ciências, que se passa em um futuro próximo, onde os humanos, depois de anos enviando satélites e naves para o espaço sem se preocupar com possíveis consequências, passam a enfrentar a constante ameaça do lixo espacial: materiais que ficam em órbita em volta da Terra e que, pela ação da gravidade, podem cair no nosso planeta e causar problemas mais sérios à vida humana. E, de fato, isso acaba acontecendo, alguns lixos espaciais caem do espaço gerando estragos e transtornos à vida na Terra, e é nesse ponto que o jogador entra, com a missão de ajudar alguns cientistas a recolher o lixo já caído e solucionar os problemas relacionados.

No intuito de cumprir com a sua missão principal no Univerciência, o jogador deve enfrentar uma série de minidesafios, sendo que um desses é o Gangorra Espacial, jogo desenvolvido neste trabalho. O Gangorra Espacial entra no mundo virtual do Univerciência no momento em que um cientista pede para o jogador estudar os planetas para melhor entender o sistema solar. Enfim, o que realmente se quer que o jogador entenda ao enfrentar o minidesafio Gangorra Espacial é a proporção entre os planetas do sistema solar, em termos de tamanho e peso. Tal proporção deve ser avaliada através de uma gangorra, em que é possível colocar os planetas em cada um de seus lados tentando mantê-la equilibrada. O Gangorra Espacial prevê

diferentes níveis de dificuldade com objetivo de equilibrar diferentes planetas, nos quais o jogador pode testar várias opções de equilíbrio. Com isso, pretende-se também que o jogador compreenda alguns fundamentos físicos a respeito do funcionamento de uma gangorra.

Os fundamentos físicos da gangorra que se deseja que o jogador entenda são que pesos parecidos se equilibram se colocados a uma mesma distância em cada um dos lados da gangorra. E também que é possível equilibrar algo muito pesado com algo muito leve, posicionando o objeto de maior peso próximo ao pivô e o de menor peso longe do pivô.

Uma descrição mais detalhada do Gangorra Espacial é apresentada no Game Design Document no Apêndice B deste trabalho.

3.1.1 Linguagem de Programação

A linguagem de programação adotada para o desenvolvimento do jogo Gangorra Espacial é a linguagem Java, pois esta é a usada no desenvolvimento do Univerciência. Isto permite uma integração mais simples e rápida do Gangorra Espacial como um minidesafio do Univerciência.

3.1.2 Realidade Aumenta aplicada ao jogo Gangorra Espacial

Esta seção descreve como a Realidade Aumentada funciona no jogo Gangorra Espacial.

3.1.2.1 Interação

A forma de interação escolhida para o jogo Gangorra Espacial foi a utilização de marcadores. Esta decisão se deve à facilidade de sua obtenção por alunos e professores no momento de sua utilização, bastando ter o arquivo digital com o desenho dos marcadores ou obter esse arquivo através da Internet. Assim, com o arquivo em mãos, o último passo é imprimir tantos marcadores quantos forem necessários. Outro ponto importante em relação à escolha da interação em Realidade Aumentada com marcadores é a facilidade de desenvolvimento do jogo, pois essa forma de interação é o padrão existente nas bibliotecas de RA, não sendo necessária uma implementação adicional.

3.1.2.2 Tipo de *Display*

Para esse projeto, o tipo de *Display* escolhido foi o monitor, pois ele facilita a criação de locais de jogo, sendo preciso apenas uma *webcam* conectada a um computador tradicional.

3.1.3 Considerações educacionais

O mundo virtual do Univerciência, em que o jogo desenvolvido neste trabalho se insere, é um mundo *online* onde vários estudantes jogam simultaneamente, de modo que o próprio Univerciência fornece opções para que os alunos conversem e troquem informações, tais como: o que acham do jogo, em que parte da história estão, de quais jogos gostaram mais e quais soluções encontraram para os jogos. Esse tipo de interação ajuda na motivação, pois o aluno não estará jogando sozinho, mas sempre acompanhado por seus colegas.

Um ponto muito importante do mundo virtual do Univerciência é que tanto seus cenários quanto os seus minidesafios são criados no intuito de deixar o jogo divertido e agradável para o jogador. Por exemplo, cada minidesafio tem um objetivo bem definido pela história do mundo virtual, o que torna enfrentá-lo muito mais interessante, sendo já apresentada a sua motivação inicial antes mesmo de começá-lo. Além disso, os minidesafios possuem um pequeno tutorial, evitando que o jogador fique perdido e não saiba o que fazer. Eles são divididos em várias fases pequenas, o que enfatiza a evolução dentro de cada um deles e dentro do mundo virtual como um todo.

O Univerciência foi pensado para ser usado, principalmente, como uma ferramenta de sala de aula na mão do professor, embora o aluno tenha liberdade dentro do mundo virtual para ir para os locais que quiser e enfrentar os minidesafios na ordem que desejar. De todo modo, é importante a presença do professor para auxiliar o aluno na compreensão do conteúdo e ajudar nos momentos em que aparecerem dúvidas.

No caso do jogo educacional desenvolvido neste trabalho, o Gangorra Espacial, a presença do professor pode ser ainda mais importante. Apesar de esse jogo possuir elementos de Realidade Aumentada, possibilitando maior interação e tornando-o mais atrativo para o aluno e, consequentemente, aumentando sua motivação em jogar. Por outro lado, ele pode causar alguma estranheza ou dificuldade de compreensão em virtude da sua forma de interação. Portanto, se um aluno não tiver o suporte inicial de um professor, aquele pode se desmotivar por não conseguir entender o jogo, gerando um efeito contrário ao desejado.

Além disso, é importante enfatizar que o objetivo do jogo Gangorra Espacial, além de mostrar os planetas do sistema solar, passando uma ideia de sua aparência e proporção de seus tamanhos, é que o aluno compreenda o funcionamento de uma gangorra. O jogo não tem o propósito de aprofundar os fundamentos teóricos nem de apresentar as fórmulas matemáticas e valores necessários para o aluno calcular os torques e posicionar os planetas, mas sim de servir de instrumento para que o professor aborde esses assuntos em sala de aula.

3.2 DESENVOLVIMENTO

Nesta seção é detalhado o desenvolvimento do jogo Gangorra Espacial utilizando-se o processo proposto por Schuytema (2008).

3.2.1 Pré-produção

Nesta etapa, foi realizado um estudo para verificar qual a biblioteca de Realidade Aumentada melhor se enquadra às necessidades do projeto do Gangorra Espacial. Depois de definida a biblioteca, foram feitos alguns testes para avaliar e validar seu desempenho. Por fim, foram criadas todas as ideias para o jogo e seu Game Design Document.

3.2.1.1 Bibliotecas

Como já foi dito anteriormente, o jogo Gangorra Espacial deve ser desenvolvido em Java, e uma série de bibliotecas deve ser escolhida para que o desenvolvimento do jogo em *software* seja iniciado. São elas: biblioteca de Realidade Aumentada, biblioteca de captura de câmera e biblioteca de visualização gráfica. A biblioteca de Realidade Aumentada gerencia todo o sistema de marcadores, suas posições e modelos relacionados; a biblioteca de captura de câmera é responsável pela obtenção das imagens do ambiente real; por fim, a biblioteca gráfica reproduz as imagens captadas pela câmera sobrepondo aos marcadores os seus modelos tridimensionais.

Geralmente, uma biblioteca de Realidade Aumentada já vem integrada com as bibliotecas de captura de câmera e visualização gráfica (SOCIALCOMPARE, 2013), pois aquela é dependente destas duas outras bibliotecas para o funcionamento pleno de suas aplicações.

Várias bibliotecas para desenvolvimento em Java foram encontradas, mas a maioria era focada em Java Mobile (SOCIALCOMPARE, 2013), como AndAR e DroidAR, utilizadas para desenvolver aplicações para aparelhos portáteis com Android como sistema operacional. Algumas também ofereciam opções para computadores de mesa, embora com número reduzido de funcionalidades. Por essas razões, essas opções de bibliotecas foram descartadas.

Uma biblioteca relativamente conhecida é a jARToolKit (JARTOOLKIT, 2013a), porém ela só permite que um único marcador seja detectado por vez (JARTOOLKIT, 2013b), impossibilitando vários tipos de aplicações e reduzindo suas possibilidades de uso. Assim, ela foi desconsiderada para adoção no desenvolvimento deste trabalho.

Outra biblioteca bastante explorada em aplicações de Realidade Aumentada desenvolvidas em Java é a NyARToolKit (NYATLA, 2013). Entretanto, ela foi desenvolvida no Japão, e só alguns documentos foram traduzidos oficialmente para o inglês, dificultando a compreensão da sua documentação. A biblioteca NyARToolKit foi criada em 2008 e vem sendo atualizada frequentemente. Sua última atualização foi no final de 2012.

Um problema enfrentado com a NyARToolKit é que, embora as opções de bibliotecas gráficas nativas sejam boas, elas não são as melhores opções quando o objetivo é desenvolver um jogo, pois não possuem diversas estruturas necessárias, que ficam a cargo de quem a utiliza. Por isso, foi criada a biblioteca ARMonkeyKit (ARMONKEYKIT, 2013a), que é uma adaptação da NyARToolKit para funcionar com jMonkeyEngine, uma poderosa ferramenta gráfica para criação de jogos. A biblioteca ARMonkeyKit foi criada em janeiro de 2010 e sendo atualizada até 2011, por essa razão ela usa versões antigas da NyARToolKit e da jMonkeyEngine. Em 2012, ela começou a ser reformulada para funcionar com versões mais atuais da NyARToolKit e jMonkeyEngine (ARMONKEYKIT, 2013b), porém nenhuma versão foi lançada até o momento da elaboração deste trabalho. Por essas razões, a ARMonkeyKit não foi usada neste projeto.

Como resultado, a NyARToolKit foi escolhida como a biblioteca de desenvolvimento de programas em Realidade Aumentada deste trabalho. Mas ainda assim outras escolhas precisam ser feitas, uma referente à biblioteca de captura de câmera, outra referente à visualização gráfica. A biblioteca NyARToolKit é compatível com duas bibliotecas de captura de câmera e duas de visualização gráfica.

A escolha de uma biblioteca para captura de câmera foi simples, as opções eram Java Media Framework (JMF) e QuickTime. O JMF foi descontinuado e por isso não funciona corretamente na maior parte dos sistemas operacionais atuais, logo ele não poderia ser utilizado. Já o QuickTime continua sendo atualizado e desenvolvido pela Apple. Assim, optou-se por utilizar o QuickTime no desenvolvimento deste projeto.

As opções para biblioteca gráfica eram Java3D e JOGL, sendo que a primeira a ser analisada foi a Java3D. Trata-se de uma biblioteca de alto nível, ou seja, abstrai vários elementos da manipulação de objetos tridimensionais e facilita o desenvolvimento. A Java3D possui também outras vantagens, como a criação de um grafo de cena, que simplifica a criação de dependências entre objetos e propaga as mudanças para todos os objetos que têm ligação com o objeto alterado. Outra vantagem é que ela possui uma ferramenta pronta para carregar os dados dos objetos tridimensionais para o cenário. Essa biblioteca começou a ser desenvolvida 1997 e continuou sendo atualizada até 2003. Em 2004 ela foi retomada como um projeto de código aberto, porém descontinuado. Apesar disso, possui uma versão bem evoluída e estável.

A JOGL é uma biblioteca de baixo nível, ou seja, seus comandos são primitivos e seus usuários devem criar suas próprias funções de alto nível. Ao contrário da Java3D, a biblioteca JOGL não tem a funcionalidade de abrir modelos tridimensionais, assim, essa função deve ser criada com base nos conjuntos de comandos existentes. Por outro lado, a JOGL é constantemente atualizada e possui diversos exemplos e tutoriais que são facilmente encontrados, razão pela qual ela foi a biblioteca gráfica escolhida para este trabalho.

Como foi dito, a JOGL não possui a função de carregar modelos tridimensionais, por isso este trabalho utiliza um carregador criado por Andrew Davison em seu livro Pro Java 6 3D Game Development (DAVISON, 2013). Essa ferramenta permite carregar modelos tridimensionais e texturas salvos no formato 3DS.

Em suma, como explicado neste subitem, este trabalho adota as seguintes bibliotecas Java para o desenvolvimento do jogo Gangorra Espacial em *software*: NyARToolKIT (Realidade Aumentada), QuickTime (captura da câmera) e JOGL (visualização gráfica). Adicionalmente, adota-se o carregador de modelos de Andrew Davison (2013).

3.2.1.2 Testes das Bibliotecas

Foram feitos quatro testes para verificar como funcionam as bibliotecas escolhidas e que tipo de resultados elas podem oferecer.

Esta seção apresenta o objetivo, a descrição detalhada e o resultado obtido em cada teste, e, no decorrer de sua explicação, é sempre indicada a linha corresponde do código fonte do teste em questão.

O primeiro teste consiste em verificar como é feita a captura de imagem da câmera, como detectar a presença de um marcador e como posicionar um objeto 3D simples em cima desse marcador. O código gerado nesse exemplo está no Apêndice A com o título Cube.java.

Este primeiro teste pode parecer muito simples, mas com ele já é possível identificar o funcionamento básico da NyARToolKit e identificar que por ser fortemente integrada as outras bibliotecas, ela torna o desenvolvimento muito mais rápido. Por exemplo, através deste primeiro teste, sabe-se que é a classe NyARQtCamera (especialização da NyARSensor, que gerencia o processamento e captura das imagens da câmera de uma forma mais geral), que oferece ao desenvolvedor a manipulação de imagens QuickTime. Para criar um objeto NyARQtCamera (linha 24) é preciso ter um objeto da classe QtCameraCapture (linha 25), que é a classe que realmente captura as imagens e as deixa prontas para serem acessadas pelo usuário através do método getSourceImage() (linha 39) da classe NyARQtCamera.

Outra classe muito importante nesse primeiro teste é a NyARGlMarkerSystem, que é responsável por gerenciar todos os marcadores e verificar se estão ou não sendo capturados. Ela é uma especialização da NyARMarkerSystem, mas já faz todos os tratamentos de criação, posição e rotação dos marcadores para ficar da forma que a JOGL entenda. Para usá-la basta criar um objeto NyARGlMarkerSystem (linha 26), adicionar os marcadores através do método addARMarker() (linha 29), atualizar a cada momento a imagem que ele recebe da câmera, passando um objeto da classe NyARQtCamera como parâmetro no método update() (linha 41) e verificar se existe ou não um marcador sendo capturado com o método isExistMaker() (linha 42), que tem como único parâmetro uma referência a um marcador específico e retorna um valor verdadeiro caso esse marcador esteja sendo capturado.

A classe que termina a sequência desse primeiro teste é a NyARGlMarkerSystemRender (linha 27), responsável por desenhar em cima da imagem que a câmera está capturando, adicionando nessa imagem os objetos associados aos marcadores. Basta usar o método loadMarkerMatrix() (linha 43) para carregar a posição do marcador e chamar os métodos que desenham o objeto que o representa (linha 44). Como resultado, esse objeto já estará na posição correta. No caso desse teste, o objeto associado ao marcador é um cubo colorido.

O resultado desse primeiro teste foi considerado bem sucedido, o objeto apareceu em cima do marcador, respondendo muito bem aos movimentos realizados. Na Ilustração 28 é possível ver qual foi o marcador utilizado e como o objeto aparece para o usuário durante a execução do programa de teste.

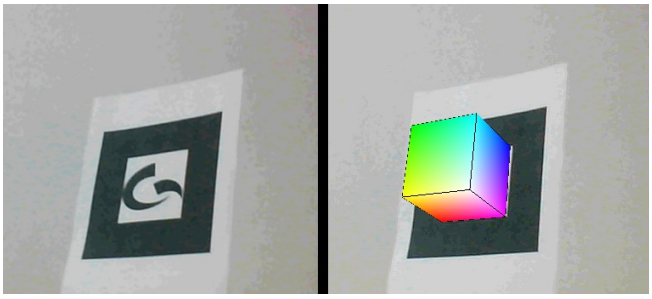


Ilustração 28: Marcador sem e com seu objeto associado

Um segundo teste realizado consiste em comprovar se realmente seria possível utilizar dois marcadores diferentes ao mesmo tempo. O código gerado nesse teste está no Apêndice A com o título Cubes.java.

O funcionamento básico desse teste é igual ao do teste anterior, com a única diferença de que dois marcadores foram adicionados no NyARGlMarkerSystem usando o método addARMarker() (linhas 30 e 31) para cada um dos marcadores, e duas checagens do método isExistMaker() (linhas 45 e 49) foram chamadas, uma para cada marcador.

Esse segundo teste também foi bem sucedido, sendo possível observar que com dois marcadores o sistema se comportou da mesma forma que o anterior com apenas um marcador. A Ilustração 29 mostra como cada um dos marcadores ficou durante a execução do programa.

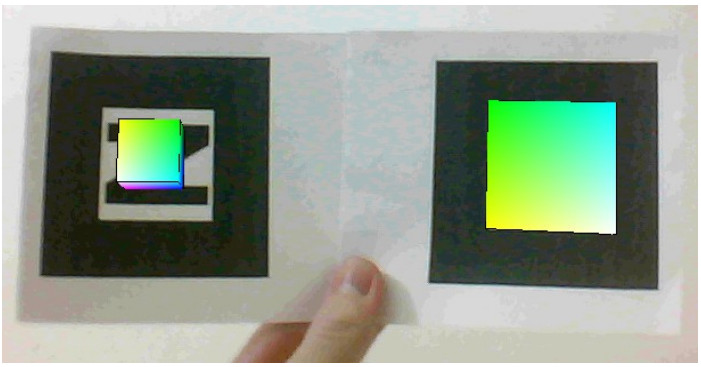


Ilustração 29: Teste com dois marcadores simultâneos

O terceiro teste realizado consiste em carregar um arquivo que descreve um objeto modelado e texturizado usando ferramentas para criação de objetos tridimensionais. Esse foi o teste usado para validar o carregador de objetos desenvolvido por Davison (2013). O código gerado nesse teste está no Apêndice A com o título ModelLoader.java.

Esse teste teve como base o código criado para o primeiro teste, com a adição da criação de um objeto Java da classe OBJModel (linha 34). Esse objeto representa um modelo 3D que é posicionado em cima do marcador pelo método loadMarkerMatrix() (linha 48) da classe NyARGlMarkerSystemRender e desenhado com o seu próprio método draw() (linha 49). A Ilustração 30 mostra o resultado desse teste.

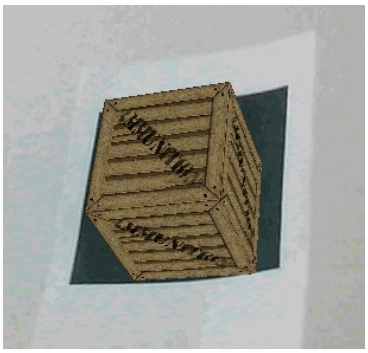


Ilustração 30: Objeto 3D
carregado de um arquivo .3DS

O quarto e último teste consiste em verificar como o sistema responderia a vários marcadores entrando e saindo da área de captura da tela. O código gerado nesse teste está no Apêndice A com o título Models.java.

Esse teste usou como base o teste anterior, mas com vários marcadores, cada um dos quais associado a um respectivo arquivo 3DS de modelo 3D com textura. Na Ilustração 31 é possível ver ao mesmo tempo na tela, cinco marcadores associados aos seus respectivos objetos 3D.

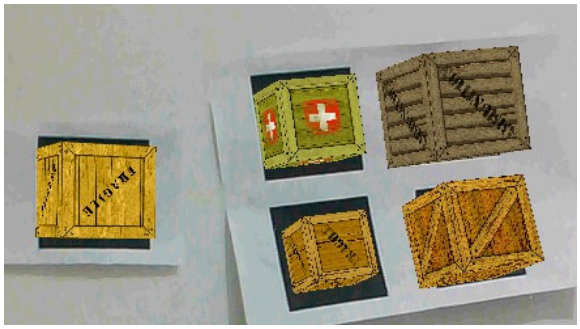


Ilustração 31: Cinco marcadores com seus modelos relacionados.

3.2.1.3 Game Design Document

Durante a pré-produção também foi desenvolvido o Game Design Document, que descreve os detalhes do jogo Gangorra Espacial. Esse documento se encontra no Apêndice B.

3.2.2 Produção

Esta é a principal etapa do desenvolvimento deste jogo, onde todas as suas funcionalidades e conteúdo foram criados.

3.2.2.1 Considerações quanto à Realidade Aumentada

No jogo Gangorra Espacial, a câmera deve ficar em uma posição elevada, aproximadamente vinte ou trinta centímetros acima da mesa, voltada para uma superfície plana e grande o suficiente para caber o marcador da gangorra. Esse marcador possui um grande espaço em branco do lado esquerdo e outro espaço em branco do lado direito para representar a área em que serão posicionados os marcadores dos planetas. Os marcadores dos planetas também devem ser posicionados na mesma superfície onde se localiza o marcador da gangorra. Para serem considerados em cima da gangorra os marcadores dos planetas devem ficar ao lado do marcador da gangorra, dentro da área em branco. A Ilustração 32 mostra como é o marcador da gangorra. A linha pontilhada nessa ilustração indica o local que deve ser recortado após o marcador ser impresso.

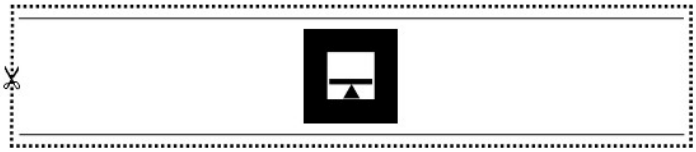


Ilustração 32: Marcador da gangorra

Os marcadores dos planetas possuem um espaço em branco em volta que deve ser usado para movimentá-lo em cima da gangorra sem atrapalhar sua captura pela câmera. A Ilustração 33 mostra como são os marcadores de dois planetas.

Ilustração 33: Marcadores de planetas

A Ilustração 34 mostra uma configuração possível dos marcadores durante o jogo. Na imagem, o jogador tentaria equilibrar o planeta Saturno, indicado pelo marcador com a letra 'S', com o planeta Terra, indicado pelo marcador com a letra 'T'.

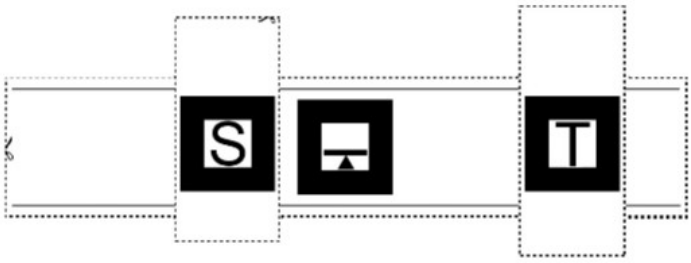


Ilustração 34: Demostração de posicionamento dos marcadores durante o jogo

Por uma questão de espaço foi optado por marcadores de quatro centímetros, grandes o suficiente para serem bem captados pela câmera e pequenos o suficiente para não atrapalharem a jogabilidade.

Os modelos tridimensionais dos planetas não são posicionados imediatamente acima do marcador para que não sejam impedidos de serem visualizados pelo modelo tridimensional da gangorra. A Ilustração 35 mostra como o planeta é posicionado em relação ao seu marcador. Alguns modelos de planetas podem parecer pequenos, mas o motivo disso é que eles estão em escala, sendo que Júpiter é muito maior em relação aos menores planetas do Sistema Solar, como Mercúrio.

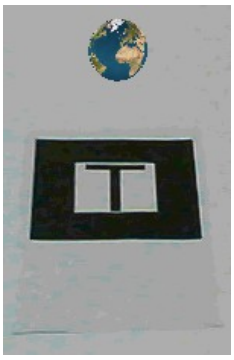


Ilustração 35:
Marcador da Terra e
seu modelo 3D

O Apêndice C contém todos os marcadores usados nesse jogo.

3.2.2.2 Simplificações Físicas

Para saber se a gangorra está equilibrada é feito o cálculo do torque para os seus dois lados, se o valor for igual, a gangorra está em equilíbrio, se ele for diferente, a gangorra está pendendo para o lado que apresenta maior torque. Os cálculos do torque de cada planeta sobre a gangorra compreendem as seguintes fórmulas (UFSM, 2013):

Torque = Peso do planeta X Distância do pivô da gangorra
Peso do planeta = Massa do planeta X gravidade

Os valores de massa de cada planeta envolvido no jogo são apresentados no Game Design Document, presente no Apêndice B deste trabalho. No momento da implementação algumas adaptações foram realizadas. Primeiro, no cálculo do peso do planeta, a gravidade foi ignorada, pois ela é constante e usada no cálculo do torque dos dois lados da gangorra. Logo, ao igualar as duas equações e verificar se ocorre o equilíbrio, essa nova equação pode ser dividida pela gravidade, eliminando-a da equação. Matematicamente:

$$m1 * g * d1 = m2 * g * d2$$

$$\frac{m1 * g * d1}{g} = \frac{m2 * g * d2}{g}$$

$$m1 * d1 = m2 * d2$$

Onde :
- $m1$ = massa do planeta 1
- g = gravidade
- $d1$ = distância entre planeta 1 e pivô
- $m2$ = massa do planeta 2
- $d2$ = distância entre planeta 2 e pivô

Outra simplificação foi feita em relação às massas dos planetas, já que os valores são muito altos para serem armazenados e utilizados em cálculos. Assim, para diminuir o valor das massas e facilitar o armazenamento e cálculo desse valores, todas as massas armazenadas foram previamente divididas pelo menor valor de massa entre todos os planetas (no caso, Mercúrio). Matematicamente :

$$m1 * d1 = m2 * d2$$

$$\frac{m1 * d1}{mM} = \frac{m2 * d2}{mM}, \quad \text{sendo } m1 = p1 * mM \text{ e } m2 = p2 * mM$$

$$\frac{p1 * mM * d1}{mM} = \frac{p2 * mM * d2}{mM}$$

$$p1 * d1 = p2 * d2$$

Onde :
- $m1$ = massa do planeta 1
- $d1$ = distância entre planeta 1 e pivô
- $m2$ = massa do planeta 2
- $d2$ = distância entre planeta 2 e pivô
- mM = massa de Mercúrio
- $p1$ = massa do planeta 1 proporcional a Mercúrio
- $p2$ = massa do planeta 2 proporcional a Mercúrio

Além disso, a distância calculada entre o pivô da gangorra e o marcador do planeta não é exata, pois os valores obtidos pela biblioteca de Realidade Aumentada ficam constantemente variando em torno de um intervalo de números não inteiros. Por esse motivo, foi preciso criar artifícios para que os cálculos de distância fossem feitos. Primeiro, os valores obtidos são transformados em valores inteiros para desconsiderar as pequenas variações que ocorrem; depois, no momento de verificar o equilíbrio dos planetas, foi considerado que cada planeta estava em um dado intervalo e não a uma distância específica.

Para o cálculo da distância do planeta em relação ao pivô da gangorra foi considerado que a posição zero da gangorra não é o meio do marcador da mesma, mas dois centímetros ao lado do marcador da gangorra, pois os marcadores não podem se sobrepor. Visto que um marcador mede quatro centímetros, e a posição de um planeta é calculada com base no meio do seu marcador, então, a menor distância possível entre o marcador de um planeta e o marcador da gangorra é de dois centímetros, ou seja, a distância entre o centro do marcador de um planeta para a borda do marcador da gangorra quando ambos os marcadores estão colados.

3.2.2.3 Modelos e texturas

Os modelos tridimensionais dos planetas consistem em esferas simples com diferentes texturas, modelados e texturizados no *software* Blender. Durante o jogo esses modelos aparecem de tamanhos diferentes por que foram redimensionados, respeitando a escala, no código-fonte. As texturas são aproximações de imagens reais dos planetas capturados por satélites (NASA, 2013; PLANETTEXTURE, 2013). A Ilustração 36 mostra todos os planetas capturados pela câmera ao mesmo tempo.

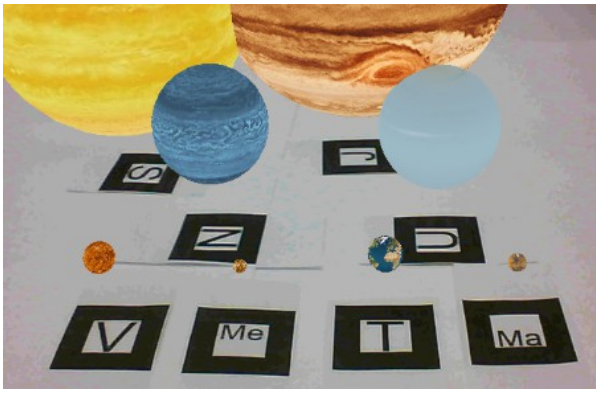


Ilustração 36: Todos os marcadores do jogo Gangorra Espacial e seus planetas

Para simular a gangorra foi utilizado um modelo tridimensional de uma barra com textura de espaço e uma linha branca, que indica o pivô da gangorra, que é importante para identificar a que distância estão os planetas.

3.2.2.4 Estado atual do jogo

O jogo já possui implementadas todas as funcionalidades que foram inicialmente levantadas, mas nada impede que novas funções sejam adicionadas mesmo após a conclusão deste trabalho. Em seu estado atual, o jogo possui um pequeno tutorial que orienta o jogador sobre onde posicionar a gangorra e os planetas para que tudo funcione corretamente. Esse é um ponto que provavelmente precisará de adaptações caso os testes indiquem que essa introdução à jogabilidade não é suficientemente clara. Na etapa do tutorial indicada na Ilustração 37 o jogador deve colocar o marcador da Terra sobre a gangorra.

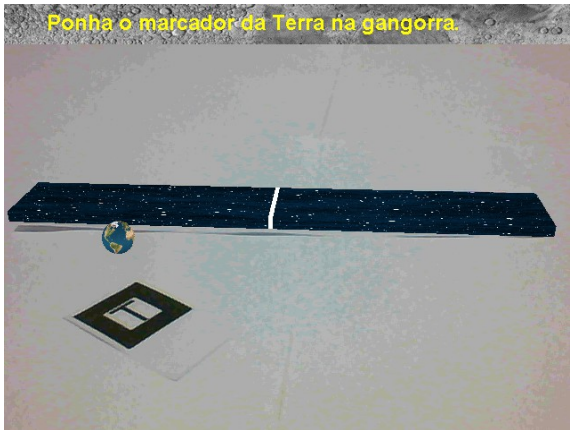


Ilustração 37: Uma das etapas do tutorial do jogo Gangorra Espacial

Após essa etapa inicial, o jogo realmente começa. O jogador deve equilibrar os planetas indicados pelo jogo, que fica constantemente testando o equilíbrio da gangorra. Quando a gangorra se mantiver equilibrada durante dois segundos é que o jogador passa para a próxima fase. Na Ilustração 38 o jogador deve encontrar a posição de equilíbrio entre a Terra e Vênus.

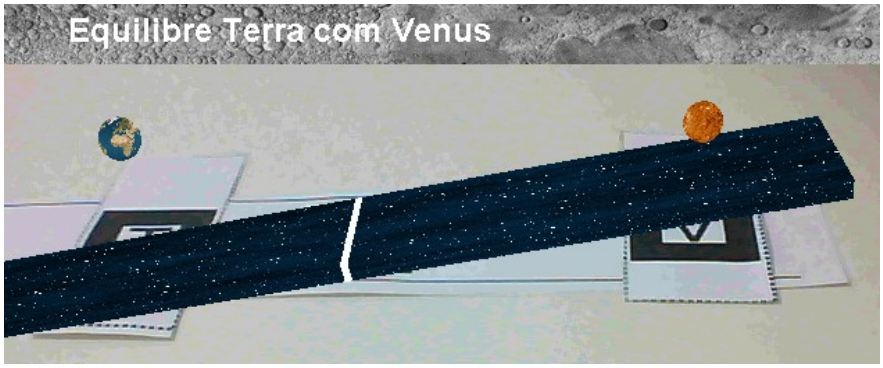


Ilustração 38: Uma das fases do jogo Gangorra Espacial

Ao completar todas as fases, o jogador continua no jogo e pode brincar com todos os planetas livremente, sem se preocupar com um objetivo predefinido. A Ilustração 39 mostra o momento em que o jogador completa todas as fases.

Ilustração 39: Fim do jogo Gangorra Espacial

3.2.3 Pós-produção

Ao final do seu presente processo de desenvolvimento, o jogo Gangorra Espacial ainda continuará na sua etapa de produção, já que sua publicação depende de outros fatores além da conclusão da implementação das funcionalidades aqui relatadas, tais como a adição dos modelos tridimensionais finais, a inserção dos sons e música ambiente, a realização de testes mais rígidos e a sua integração com o mundo virtual do Univerciência. Porém, esses são alguns dos desdobramentos futuros pensados para o desenvolvimento do jogo. A responsabilidade por essas etapas está fora do escopo deste trabalho.

4 CONCLUSÃO

Este trabalho tem como objetivo, além da criação de um jogo, gerar subsídio teórico e prático para que outros jogos de Realidade Aumentada possam ser criados e adicionados ao projeto Univerciência que vem sendo desenvolvido pelo GeNESS. Logo, tudo o que foi aprendido e desenvolvido nesse trabalho contribuirá diretamente na geração de outros jogos, onde os resultados aqui gerados serão reproduzidos.

No Capítulo 2, foi possível visualizar que os fundamentos teóricos que embasam a Realidade Aumentada já vêm sendo estudados há algum tempo e possuem diversas pesquisas que comprovam as vantagens de utilização desta tecnologia para diversas áreas de aplicação. Já existe muito material nessa área descrevendo várias formas de se interagir com um computador através da Realidade Aumentada. Foi observado também que já existem utilizações da Realidade Aumentada em várias áreas, como design de interiores, lojas digitais, educação e jogos. Esse último ramo já vem sendo explorado pela indústria de forma muito abrangente há alguns anos, e já foram criados jogos de grande porte baseados totalmente nas funções de Realidade Aumentada.

O Capítulo 3 apresenta o processo de desenvolvimento da prova de conceito de um jogo em RA, pondo em prática os conceitos correlatos levantados no capítulo anterior. Inicialmente, é demonstrado o levantamento das ferramentas para se desenvolverem aplicações de Realidade Aumentada em Java. Foi possível verificar que faltam soluções atualizadas e de fácil compreensão, ou seja, a maior parte das ferramentas estava desatualizada ou já havia sido abandonada há vários anos e a ferramenta mais adequada encontrada, e que foi usada na elaboração desse trabalho, possuía a maior parte da sua documentação em japonês. Apesar dessa barreira linguística, não foi difícil utilizá-la, pois ela já vinha completa e seus criadores tiveram o cuidado de criar dependências a mais de uma biblioteca. Assim, quando uma das opções era inadequada, existia outra opção pronta para ser usada. Essa biblioteca trata Realidade Aumentada através do uso de marcadores, o que cria mais uma dependência, além da já existente ao computador e a *webcam*, mas que é facilmente resolvida se existir um arquivo digital desses marcadores, permitindo ao usuário imprimir livremente quantos marcadores necessitar.

A partir dos resultados deste trabalho, considera-se que essa biblioteca (NyARToolKit) se mostrou eficiente e confirmou que é possível desenvolver jogos de Realidade Aumentada para computador na linguagem Java, pois, além de ter passado em todos os testes desenvolvidos, ela também permitiu que todas as ideias usadas na criação das funcionalidades fossem corretamente implementadas, como demonstrado na seção 3.2 deste trabalho. Assim, o jogo cumpriu com todas as especificações técnicas levantadas, o que pode ser verificado a partir do Game Design Document, documento com a descrição detalhada de todo o funcionamento do jogo, desenvolvido na etapa de pré-produção e disponível nos Apêndices desse trabalho. A real contribuição educacional do jogo poderá ser comprovada mediante aplicação de testes junto ao público-alvo, estudantes do Ensino Básico de doze a quatorze anos, e análise técnica de profissionais ligadas à educação.

4.1 TRABALHOS FUTUROS

Como já foi explicado anteriormente, o jogo desenvolvido neste trabalho ainda não está pronto, portanto, o primeiro passo é conclui-lo e todos os elementos faltantes, como: arte, som, testes e integração com o mundo do jogo.

Outro ponto a ser verificado é se o jogo precisa ter mais conteúdo, e se for o caso, será necessário desenvolvê-lo. Algumas das opções de evolução para esse jogo são:

- criação de mais planetas de fora do sistema solar ou adição dos planetas anões, como Plutão;
- aumentar as informações de cada um dos planetas, como: incliná-los no eixo de rotação e adicionar funções para que eles fiquem girando em uma velocidade proporcional a sua velocidade real;
- mudar o pivô da gangorra para criar mais desafios na hora de equilibrar os planetas. Já que o cálculo para verificar o equilíbrio da gangorra leva em conta a posição dos planetas em relação ao seu pivô, ao mudar a posição deste pivô, é possível criar novos desafios que não existiriam anteriormente.

REFERÊNCIAS BIBLIOGRÁFICAS

ARAÚJO, C, F, M., SIMON, E, B. "**Objetos de aprendizagem em educação a distância: uso de jogos educacionais no estilo RPG (Role-playing games) digitais**", Paidéi@ – Revista Científica de Educação a Distância, v. 1, n. 1, p. 1-32. 2008

ARMONKEYKIT. **ARMonkeyKit.** Disponível em: <http://armonkeykit.wordpress.com/>. Acessado em: Maio, 2013a.

ARMONKEYKIT. **Porting ARMonkeyKit to JME3 some questions.** Disponível em: <http://jmonkeyengine.org/forum/topic/porting-armonkeykit-to-jme3-some-questions/> Acessado em: Maio, 2013b.

AZUMA, R. et al. "**Recent Advances in Augmented Reality**". IEEE Computer Graphics and Applications 21, 6 (Nov/Dec 2001), p. 34-47.

AZUMA, T. "**A survey of Augmented Reality**", Presence: Teleoperators and Virtual Environments, p. 355-385, Agosto 1997

BANDAI. Disponível em: <http://www.tokyo-shoseki.co.jp/books/miraikei/>. Acessado em: Abril, 2013.

BORGES, C. J. **O Lúdico nas Interfaces das Relações Educativas**, Revista de Pedagogia, N 12 Vol. 6. 2005

BRATHWAITE, B., SCHREIBER, I. "**Challenges for Game Designers**". 2009.

CHANDLER, H. M. "**The Game Production Handbook**" 2nd ed. 2009.

DAVISON, Andrew. **Chapter 17. Picking the models.** Disponível em: <http://fivedots.coe.psu.ac.th/~ad/jg2/ch17/index.html> Acessado em: Maio, 2013.

COSTA, L. D. **O que os jogos de entretenimento têm que os jogos com fins pedagógicos não têm**. VIII Simpósio Brasileiro de *Games* e Entretenimento Digital (VIII SBGames), Rio de Janeiro, SBC – Sociedade Brasileira de Computação. 2009

ENGADGET. **Sony Playstation 3 launch detail.** Disponível em: <http://www.engadget.com/2006/05/08/sony-playstation-3-launch-details/> Acessado em: Junho, 2013.

EXAME. "**15 Campanhas Inteligentes com Realidade Aumentada. Fevereiro**", 2013. Disponível em: <http://exame.abril.com.br/marketing/noticias/16-usos-inteligentes-de-realidade-aumentada-em-campanhas> Acessado em: Maio, 2013

GIZMODO. **Lego's digital box show compelted 3D models with no construction needed.** Disponível em: <http://gizmodo.com/5138574/legos-digital-box-shows-completed-3d-models-with-no-construction-needed> Acessado em: Abril, 2013.

HUMBLEBUNDLE. **Prior Bundle Statistics.** Disponível em: <http://support.humblebundle.com/customer/portal/articles/281031-prior-bundle-statistics> Acessado em: Junho, 2013.

IGN. **The Economics of Game Publishing.** Disponível em: <http://www.ign.com/articles/2006/05/06/the-economics-of-game-publishing> Acessado em: Junho, 2013.

INTEL. **Intel labs: augmented reality lego display.** Disponível em: <http://www.intel.com/content/www/us/en/research/intel-labs-demo-augmented-reality-lego-display-video.html> Acessado em: Abril, 2013.

JARTOOLKIT. **JARToolKit.** Disponível em: <http://sourceforge.net/projects/jartoolkit/> Acessado em: Maio, 2013a.

JARTOOLKIT. **ReadMe.txt**. Disponível em: <http://sourceforge.net/projects/jartoolkit/files/latest/download?source=navbar> Acessado em: Maio, 2013b.

KIRNER, C., ZORZAL, E. R. "**Aplicações Educacionais em Ambientes Colaborativos com Realidade Aumentada**" Disponível em: <http://www.realidadeaumentada.com.br/artigos/13164.pdf>. Acessado em: Maio, 2013

LAWRENCEVILLE, GA. **Augmented reality- The new big wave of learning technologies is on the horizon.** Disponível em: <http://www.prweb.com/releases/LogicalChoiceTechnologies/AR_Alive/prweb4729474.htm>. Novembro, 2010. Acessado em: Abril 2013.

LOGICALCHOICES. **Logical Choice Letters Alive.** Disponível em: <http://www.logicalchoice.com/products/lettersalive/> Acessado em: Abril, 2013.

MAZZAROTTO, M., BATTAIOLA, A. L. "**Uma visão experiencial dos jogos de computador na Educação: A relação entre motivação e melhora do raciocínio no processo de aprendizagem.**" VIII Brazilian Symposium on Games and Digital Entertainment. Outubro, 2009.

MILGRAM, P. et al. "**Augmented Reality: A class of displays on the reality-virtuality continuum**", SPIE, V.2351, p. 282-292, 1994.

MOORE, M. E, Novak, J. "**Game Industry Career Guide**", Cengage Learning. 2010

MOTA, A. B. "**Criança e Mídia – O acesso ao computador e seus reflexos nos saberes da criança de educação infantil.**" 2007. 135f. Tese (Mestrado em Educação) Universidade Federal do Paraná, Curitiba, 2007.

NASA. **Solar System Simulator.** Disponível em: <http://maps.jpl.nasa.gov/> Acessado em: Maio, 2013.

NYARTOOLKIT. **NyARToolKitCPP/2.4.0.** Disponível em: <http://nyatla.jp/nyartoolkit/wp/?p=448> Novembro, 2009. Acessado em: Abril, 2013.

NYATLA.. **NyARToolKit**. Disponível em: <http://nyatla.jp/nyartoolkit/wp/>. Acessado em: Maio. 2013

PASSERINO, L., M. **Avaliação de jogos educativos computadorizados**. In: TISE ´98 – Taller Internacional de Software Educacional, Santiago, Chile. Disponível em: <http://www.c5.cl/ieinvestiga/actas/tise98/html/trabajos/jogosed/> Acessado em: Junho, 2013.

PCMAG. **Sony Wonderbook: Book of Spells Move bundle.** Diponível em: <http://www.pcmag.com/slideshow_viewer/0,3253,l=305883&a=305897&po=4,00.asp> Acessado em: Abril, 2013.

PLANETTEXTURE. **Planet Texture Maps.** Disponível em: <http://planetpixelemporium.com/planets.html> Acessado em: Maio, 2013.

PLAYSTATION. **Table Top Tanks.** Disponível em: <http://us.playstation.com/psvita/games-and-media/psv-table-top-tanks.html> Acessado em: Abril, 2013a.

PLAYSTATION. **AR Play: Augmented reality gaming on PS Vita system.** Disponível em: <http://us.playstation.com/psvita/apps/psvita-app-ar.html> Acessado em: Abril, 2013b.

PLAYSTATION. **Little Deviants: Touch, tap and tilt to save Little Deviants from the Evil Botz.** Disponível em: <http://us.playstation.com/psvita/games-and-media/psv-little-deviants.html> Acessado em: Abril, 2013c.

PLAYSTATION. **EyePet**. Disponível em: <https://www.youtube.com/watch?v=YZvxIjdyyII> Acessado em: Abril 2013d.

PLAYSTATION. **Wonderbook: Book of Spells.** Disponível em: <http://us.playstation.com/games/wonderbook-book-of-spells-ps3.html> Acessado em: Abril, 2013e.

PLAYSTATIONBLOG. **Get Ready: PS Vita is Coming February 22nd**. Disponível em: <http://blog.us.playstation.com/2011/10/18/get-ready-ps-vita-is-coming-february-22nd/> Acessado em: Junho, 2013

PROVIDELO, C. et al. "**Ambiente dedicado para aplicações educacionais interativas com realidade misturada**", VII Symposium on Virtual Reality, SP – Brazil, October, 2004.

REALIDADEAUMENTADA. **Realidade Aumentada** Disponível em:< http://realidadeaumentada.com.br/home/index.php?option=com_content&task=view&id=1&Itemid=27 >. Acesso em: junho 2012

SANTIN, M. "**Desenvolvimento de técnicas de interação para aplicações de realidade aumentada com ARToolKit**", I Workshop de realidade aumentada, 2004.

SANTIN, M. "**Técnicas para fixar objetos virtuais em ambientes de autoria com realidade aumentada**", III Workshop de realidade aumentada, 2006.

SANTIN, M. et al. "**Ações interativas em Ambientes de Realidade Aumentada com ARToolKit**", VII Symposium on Virtual Reality, SBC, p. 161-168. 2004.

SCHUYTEMA, P. "**Design de games: Uma abordagem prática**", Cengage Learning Editora. 2008.

SOCIALCOMPARE. **Augmented Reality SDK Comparison** <http://socialcompare.com/en/comparison/augmented-reality-sdks> Maio, 2013. Acessado em: Maio, 2013.

SONY. **Meet EyePet.** Disponível em: <http://www.eyepet.com/meet-eyepet/> Acessado em: Abril, 2013.

STARBUCKS. **Stakbucks cup magic.** Disponível em: <http://youtu.be/RWwQXi9RG0w>. Novembro, 2011. Acessado em: Abril, 2013.

THEESA. **The Entertainment Software Association – 2012 Annual Report**. Disponível em: <http://www.theesa.com/about/ESA_2012_Annual_Report.pdf> Acessado em: Junho, 2013

TIMM, M. I., RIBEIRO, L. O. M., LANDO, V. R., AZEVEDO, M. P., VIEIRA, E. **Game educacional: desafios da integração de elementos ficcionais, tecnológicos, cognitivos e de conteúdo.** *SBGames 2008 - VII Symposium on Computer Games and Digital Entertainment*, Belo Horizonte. 2008

UFSM. **Torque**. Grupo de Ensino de Física da Universidade de Santa Maria. Disponível em: <http://coral.ufsm.br/gef/Rotacoes/rotacoes04.pdf> Acessado em: Julho, 2013.

VIEWAR. **List of our Apps.** Disponível em: <http://www.viewar.com/site/apps> Acessado em: Abril, 2013

WANDERLEY, A. J.; MEDEIROS, A. F.; SILVA, K. S.; SILVA, M. F. **Aprendizagem Interativa: Uma Análise do Uso da Realidade Aumentada no Desenvolvimento de Jogos**

Educacionais. Disponível em: <http://www.die.ufpi.br/ercemapi2011/artigos/ST1_01.pdf> Acessado em: Maio, 2013.

ZORZAL, E. R., BUCCIOLI, A. A. B. and KIRNER, C. "**Desenvolvimento de Jogos em Ambiente de Realidade Aumentada**". SBGAMES – Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital, WJogos, USP, São Paulo, SP. 2005.

ZORZAL, E. R., et al. "**Aplicações de jogos educacionais c om realidade aumentada**" Revista Renote, v.6 nº 1, Julho, 2008

ZORZAL, E. R., et al. "**Usando realidade aumentada no desenvolvimento de quebra-cabeças educacionais**", VII Symposium on Virtual Reality, 2006

ZUGARA, **A truly interactive experience for today's digital shopper.** Disponivel em: <http://webcamsocialshopper.com/demos> Acessado em: Abril, 2013.

ZUGARA. **The world's best virtual dressing room.** Disponível em: <http://webcamsocialshopper.com/> Acessado em: Abril, 2013.

APÊNDICE A – CÓDIGO DOS TESTES

Cube.java

```
package testes;
import javax.media.opengl.*;
import jp.nyatla.nyartoolkit.jogl.sketch.GlSketch;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import jp.nyatla.nyartoolkit.markersystem.NyARMarkerSystemConfig;
import jp.nyatla.nyartoolkit.qt.utils.NyARQtCamera;
import jp.nyatla.nyartoolkit.qt.utils.QtCameraCapture;


public class Cube extends GlSketch
{
     private NyARQtCamera camera;
     private NyARGlMarkerSystem nyar;
     private NyARGlMarkerSystemRender render;

     private final static String ARCODE_FILE = "geness.pat";
     private int id;

     public void setup(GL gl)throws Exception
     {
       this.size(640,480);
       NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(640,480);
       this.camera=new                                                    NyARQtCamera(new
QtCameraCapture(config.getScreenSize(),30.0f));
       this.nyar=new NyARGlMarkerSystem(config);
       this.render=new NyARGlMarkerSystemRender(this.nyar);

       this.id=this.nyar.addARMarker(ARCODE_FILE,16,25,80);
       gl.glEnable(GL.GL_DEPTH_TEST);
       this.camera.start();
     }

     public void draw(GL gl)throws Exception
     {
       synchronized(this.camera)
       {
             try {
                   this.render.drawBackground(gl, this.camera.getSourceImage());
                   this.render.loadARProjectionMatrix(gl);
                   this.nyar.update(this.camera);
                   if(this.nyar.isExistMarker(this.id)){
                         this.render.loadMarkerMatrix(gl,this.id);
                         this.render.colorCube(gl,40,0,0,20);
                   }
                   Thread.sleep(1);
```

```
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
    }
    public static void main(String[] args)
    {
      new Cube().run();
      return;
    }

}
```

Cubes.java

```
package testes;

import javax.media.opengl.*;
import jp.nyatla.nyartoolkit.jogl.sketch.GlSketch;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import jp.nyatla.nyartoolkit.markersystem.NyARMarkerSystemConfig;
import jp.nyatla.nyartoolkit.qt.utils.NyARQtCamera;
import jp.nyatla.nyartoolkit.qt.utils.QtCameraCapture;

public class Cubes extends GlSketch
{
     private NyARQtCamera camera;
     private NyARGlMarkerSystem nyar;
     private NyARGlMarkerSystemRender render;

     private final static String ARCODE_FILE = "e.pat";
     private final static String ARCODE_FILE2 = "n.pat";
     private int[] ids=new int[2];

     public void setup(GL gl)throws Exception
     {
       this.size(640,480);
       NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(640,480);
       this.camera=new                                                NyARQtCamera(new
QtCameraCapture(config.getScreenSize(),30.0f));
       this.nyar=new NyARGlMarkerSystem(config);
       this.render=new NyARGlMarkerSystemRender(this.nyar);

       this.ids[0]=this.nyar.addARMarker(ARCODE_FILE2,16,25,80);
       this.ids[1]=this.nyar.addARMarker(ARCODE_FILE,16,25,80);

       gl.glEnable(GL.GL_DEPTH_TEST);
       this.camera.start();
     }


     public void draw(GL gl)throws Exception
     {
       synchronized(this.camera){
              try {
                     this.render.drawBackground(gl, this.camera.getSourceImage());
                     this.render.loadARProjectionMatrix(gl);
                     this.nyar.update(this.camera);
                     if(this.nyar.isExistMarker(this.ids[0])){
                            this.render.loadMarkerMatrix(gl,this.ids[0]);
                            this.render.colorCube(gl,20,0,0,20);
                     }
                     if(this.nyar.isExistMarker(this.ids[1])){
```

```
                        this.render.loadMarkerMatrix(gl,this.ids[1]);
                        this.render.colorCube(gl,40,0,0,20);
                  }
                  Thread.sleep(1);
            } catch (Exception e) {
                  e.printStackTrace();
            }
      }
    }
    public static void main(String[] args)
    {
      new Cubes().run();
      return;
    }
}
```

ModelLoader.java

```
package testes;

import javax.media.opengl.*;

import OBJLoader.OBJModel;
import jp.nyatla.nyartoolkit.jogl.sketch.GlSketch;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import jp.nyatla.nyartoolkit.markersystem.NyARMarkerSystemConfig;
import jp.nyatla.nyartoolkit.qt.utils.NyARQtCamera;
import jp.nyatla.nyartoolkit.qt.utils.QtCameraCapture;

public class ModelLoader extends GlSketch
{
     private NyARQtCamera camera;
     private NyARGlMarkerSystem nyar;
     private NyARGlMarkerSystemRender render;

     private final static String ARCODE_FILE_G = "g.pat";
     private int[] ids=new int[1];
     private OBJModel[] models = new OBJModel[1];

     public void setup(GL gl)throws Exception
     {
       this.size(640,480);
       NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(640,480);
       this.camera=new                                                NyARQtCamera(new
QtCameraCapture(config.getScreenSize(),30.0f));
       this.nyar=new NyARGlMarkerSystem(config);
       this.render=new NyARGlMarkerSystemRender(this.nyar);

       this.ids[0]=this.nyar.addARMarker(ARCODE_FILE_G,16,25,80);

       this.models[0] = new OBJModel("crate1", 50, gl, true);

       gl.glEnable(GL.GL_DEPTH_TEST);
       this.camera.start();
     }

     public void draw(GL gl)throws Exception
     {
       synchronized(this.camera){
              try {
                     this.render.drawBackground(gl, this.camera.getSourceImage());
                     this.render.loadARProjectionMatrix(gl);
                     this.nyar.update(this.camera);
                     if(this.nyar.isExistMarker(this.ids[0])){
                            this.render.loadMarkerMatrix(gl,this.ids[0]);
                            models[0].draw(gl);
```

```
                }
                Thread.sleep(1);
            } catch (Exception e) {
                e.printStackTrace();
            }
        }
    }
    public static void main(String[] args)
    {
        new ModelLoader().run();
        return;
    }
}
```

Models.java

```
package testes;

import javax.media.opengl.*;

import OBJLoader.OBJModel;
import jp.nyatla.nyartoolkit.jogl.sketch.GlSketch;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import jp.nyatla.nyartoolkit.markersystem.NyARMarkerSystemConfig;
import jp.nyatla.nyartoolkit.qt.utils.NyARQtCamera;
import jp.nyatla.nyartoolkit.qt.utils.QtCameraCapture;

public class Models extends GlSketch
{
     private NyARQtCamera camera;
     private NyARGlMarkerSystem nyar;
     private NyARGlMarkerSystemRender render;

     private final static String ARCODE_FILE_G = "g.pat";
     private final static String ARCODE_FILE_E = "e.pat";
     private final static String ARCODE_FILE_N = "n.pat";
     private final static String ARCODE_FILE_S = "s.pat";
     private final static String ARCODE_FILE_Ge = "geness.pat";
     private int[] ids=new int[5];
     private OBJModel[] models = new OBJModel[5];

     public void setup(GL gl)throws Exception
     {
       this.size(640,480);
       NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(640,480);
       this.camera=new                                                    NyARQtCamera(new
QtCameraCapture(config.getScreenSize(),30.0f));
       this.nyar=new NyARGlMarkerSystem(config);
       this.render=new NyARGlMarkerSystemRender(this.nyar);

       this.ids[0]=this.nyar.addARMarker(ARCODE_FILE_G,16,25,80);
       this.ids[1]=this.nyar.addARMarker(ARCODE_FILE_E,16,25,80);
       this.ids[2]=this.nyar.addARMarker(ARCODE_FILE_N,16,25,80);
       this.ids[3]=this.nyar.addARMarker(ARCODE_FILE_S,16,25,80);
       this.ids[4]=this.nyar.addARMarker(ARCODE_FILE_Ge,16,25,80);

       this.models[0] = new OBJModel("crate1", 50, gl, true);
       this.models[1] = new OBJModel("crate2", 60, gl, true);
       this.models[2] = new OBJModel("crate3", 70, gl, true);
       this.models[3] = new OBJModel("crate4", 80, gl, true);
       this.models[4] = new OBJModel("Crate_Fragile", 65, gl, true);

       gl.glEnable(GL.GL_DEPTH_TEST);
       this.camera.start();
```

```
	}

	public void draw(GL gl)throws Exception
	{
	  synchronized(this.camera){
		try {
			this.render.drawBackground(gl, this.camera.getSourceImage());
			this.render.loadARProjectionMatrix(gl);
			this.nyar.update(this.camera);
			if(this.nyar.isExistMarker(this.ids[0])){
				this.render.loadMarkerMatrix(gl,this.ids[0]);
				models[0].draw(gl);
			}
			if(this.nyar.isExistMarker(this.ids[1])){
				this.render.loadMarkerMatrix(gl,this.ids[1]);
				models[1].draw(gl);
			}
			if(this.nyar.isExistMarker(this.ids[2])){
				this.render.loadMarkerMatrix(gl,this.ids[2]);
				models[2].draw(gl);
			}
			if(this.nyar.isExistMarker(this.ids[3])){
				this.render.loadMarkerMatrix(gl,this.ids[3]);
				models[3].draw(gl);
			}
			if(this.nyar.isExistMarker(this.ids[4])){
				this.render.loadMarkerMatrix(gl,this.ids[4]);
				models[4].draw(gl);
			}
			Thread.sleep(1);
		} catch (Exception e) {
			e.printStackTrace();
		}
	  }
	}
	public static void main(String[] args)
	{
	  new Models().run();
	  return;
	}
}
```

APÊNDICE B – GAME DESIGN DOCUMENT

Game Design Document

Gangorra Espacial

Sumário

1 VISÃO GERAL

1.1 RESUMO

Nesse jogo de Realidade Aumentada o jogador usará marcadores que representam planetas do sistema solar e terá como objetivo posicionar estes planetas em cima de uma gangorra para que ela fique equilibrada. O jogo possuirá diversas fases, nas quais diferentes planetas devem ser equilibrados.

1.2 ASPECTOS FUNDAMENTAIS

O jogador controlará apenas os marcadores em uma superfície plana com o propósito de obter o equilíbrio da gangorra, ao alcançá-lo, o jogador passará para a fase seguinte.

1.3 PÚBLICO-ALVO

O jogo tem como público alvo adolescentes entre 12 e 14 anos.

1.4 GÊNERO

Este jogo é do tipo quebra-cabeças, pois o jogador deve, com base nos marcadores definidos pela fase e pelas noções de física que possui, encontrar a posição que equilibre a gangorra.

1.5 HISTÓRIA

Esse jogo será um dos muitos jogos de um mundo virtual. A história desse mundo virtual se passa em um futuro próximo, onde os humanos, depois de anos enviando satélites e naves para o espaço sem se preocupar com as consequências, têm um problema com o grande número de lixo espacial. Esse lixo fica em órbita em volta da Terra e é afetado pela sua gravidade, logo, é uma questão de tempo para que caía e cause estrago. É nesse ponto que o jogador entra, ele ajuda cientistas a recolher o lixo já caído e solucionar os problemas causados.

Todos os jogos desse mundo virtual estão relacionados com essa história e esse jogo não é diferente, pois ele entra em um momento em que um cientista pede para o jogador estudar os planetas para melhor entender o sistema solar.

2 GAMEPLAY

2.1 CÂMERA

Por este jogo ser de Realidade Aumentada é necessário que o usuário possua uma câmera conectada ao computador, que deve ser posicionada em algum lugar elevado e voltada

para uma superfície plana na horizontal. A câmera deve capturar a gangorra em uma posição central na tela, de forma que exista uma boa distância de cada lado da gangorra para posicionar os outros marcadores.

2.2 MARCADORES

Este jogo possui nove marcadores, um para a gangorra e oito que representam cada um dos planetas do sistema solar. O marcador da gangorra terá como símbolo uma pequena gangorra e duas linhas paralelas, uma em cima e outra embaixo, para ajudar o jogador a posicionar os marcadores dos planetas.

Marcador da gangorra

Os marcadores dos planetas devem possuir como símbolo a primeira letra do seu nome, por exemplo, a Terra terá como símbolo a letra “T”. No caso de Mercúrio e Marte, por possuírem a mesma letra no nome, serão usadas as duas primeiras letras.

2.3 PLANETAS

Os dados sobre os planetas usados nesse jogo foram tirados do site: http://astro.if.ufrgs.br/ssolar.htm

Os oito planetas do sistema solar devem ser colocados no jogo levando-se em conta dois de seus atributos físicos, o diâmetro e a massa. O diâmetro será usado para determinar o tamanho de seu modelo tridimensional, e, a massa para o cálculo de equilíbrio da gangorra. Abaixo, tabela contendo os dados de cada planeta.

Planeta	Diâmetro (km)	Massa (kg)
Mercúrio	4.878	$3,3x10^{23}$
Vênus	12.100	$4,87\times10^{24}$
Terra	12.756	$5,97\times10^{24}$
Marte	6.786	$6,42\times10^{23}$
Júpiter	142.984	$1,90\times10^{27}$
Saturno	120.536	$5,69\times10^{26}$
Urano	51.108	$8,70\times10^{25}$
Netuno	49.538	$1,03\times10^{26}$

No jogo, esses planetas devem ser modelos tridimensionais e estar em escala para passar ao jogador uma noção precisa da relação entre o tamanho de cada um.

2.4 FÍSICA

O jogo deve se basear na seguinte fórmula:

Torque = Peso do planeta X Distância do pivô da gangorra

Os casos abordados nesse jogo são fisicamente impossíveis de ocorrer, mas podem ser computacionalmente simulados para se ter uma noção da relação entre as massas dos planetas.

Para cálculo do peso dos planetas a fórmula a ser usada é:

Peso do planeta = Massa do planeta X gravidade

A gravidade da simulação é um valor aproximado da gravidade da Terra, o valor usado será 10.

Para saber se a gangorra esta equilibrada é feito o cálculo do torque para os seus dois lados, se o valor for igual a gangorra, está em equilíbrio, se ele for diferente, a gangorra está pendendo para o lado que possui maior torque.

2.5 GAMEPLAY

Com o marcador da gangorra posicionado, o jogador deve colocar os planetas indicados, de acordo com cada uma das fases, em cada um dos lados da gangorra e ir mexendo na posição dos marcadores até que eles fiquem equilibrados por 2 segundos. Se os marcadores atingirem o equilíbrio e permanecerem durante esse tempo, o jogador passa para a próxima fase.

A gangorra deve interagir a cada nova adição de planetas, se um dos lados ficar mais pesado, esse lado deve abaixar, elevando o lado mais leve da gangorra, e cada novo movimento dos marcadores deve ser recalculado pelo sistema e a gangorra assumirá a posição determinada por esse novo cálculo.

2.6 FLUXO DO JOGO

A primeira parte do jogo funciona como um tutorial, informando ao jogador todos os movimentos que ele deve fazer. Inicialmente, o jogo indicará que a gangorra deve ser colocada na

frente da câmera, depois o jogador deve colocar o planeta Terra em um dos lados da gangorra. Após essa etapa, o jogo começa, e o jogador deve, posicionando os marcadores dos planetas, procurar o equilíbrio da gangorra em cada uma das fases.

Após completar o jogo, ele não deve ser fechado, deve ser mantido aberto em um modo em que o jogador tenha liberdade de posicionar qualquer um dos planetas e ver como a gangorra se comporta.

3 CONTEÚDO ARTÍSTICO

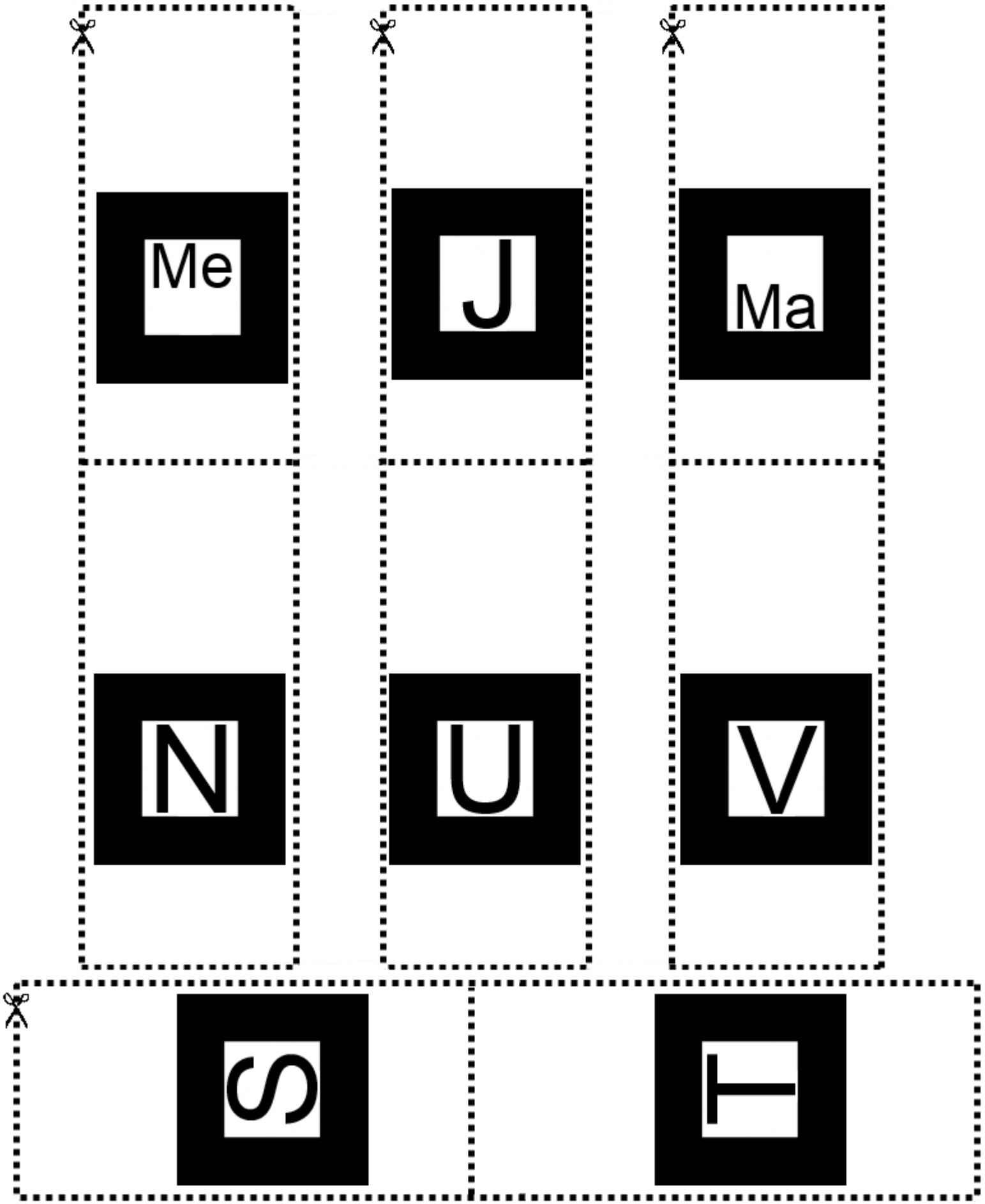
3.1 INTERFACE

O jogador deve visualizar na tela, durante todo o jogo, uma imagem que possua descrito o objetivo da fase em que ele está. O jogo não deve possuir nenhuma interface além do espaço necessário para posicionar o objetivo.

3.2 MODELOS

Todos os elementos do jogo, gangorra e planetas, devem ser modelos tridimensionais. Os planetas devem possuir uma escala similar a sua escala real e a textura deve ser o mais parecida possível com a do planeta real. A gangorra deve ser uma superfície plana e indicar claramente a posição de seu pivô.

APÊNDICE C – MARCADORES



APÊNDICE D – CÓDIGO FONTE

Classe GlSketch – Classe presente no NyARToolKit, mas sofreu pequenas alterações

```
import java.awt.Frame;
import java.awt.Insets;
import java.awt.event.MouseEvent;
import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import javax.media.opengl.GLCanvas;
import javax.media.opengl.GLEventListener;
import jp.nyatla.nyartoolkit.core.types.NyARIntSize;
import com.sun.opengl.util.Animator;

public abstract class GlSketch implements GLEventListener, MouseListener
,MouseMotionListener
{
     private Frame _frame;
     protected GLCanvas _canvas;
     boolean _is_setup_done=false;
     public GlSketch()
     {
     }

     public void run()
     {
       this._frame= new Frame("NyARTK Sketch");
       this._frame.addWindowListener(new WindowAdapter() {
              public void windowClosing(WindowEvent e)
              {
                     System.exit(0);
              }
       });
       this._canvas = new GLCanvas();
       this._canvas.addGLEventListener(this);
       this._canvas.addMouseListener(this);
       this._canvas.addMouseMotionListener(this);
       Insets ins = this._frame.getInsets();
       this._frame.setSize(320 + ins.left + ins.right,240 + ins.top + ins.bottom);
       this._canvas.setBounds(ins.left, ins.top,320,240);
       this._frame.add(this._canvas);
       this._frame.setVisible(true);
     }
     public void title(String i_title)
     {
       this._frame.setTitle(i_title);
     }
     public void size(NyARIntSize i_s)
```

```
    {
      this.size(i_s.w,i_s.h);
    }
    public void size(int i_w,int i_h)
    {
      Insets ins = this._frame.getInsets();
      this._frame.setSize(i_w + ins.left + ins.right,i_h + ins.top + ins.bottom);
      this._canvas.setBounds(ins.left, ins.top, i_w,i_h);
    }
    public final void init(GLAutoDrawable drawable)
    {
      try {
            GL gl=drawable.getGL();
            this.setup(gl);
            Animator animator = new Animator(drawable);
            animator.start();
            this._is_setup_done=true;
      } catch (Exception e) {
            e.printStackTrace();
      }
      return;
    }
    public final void reshape(GLAutoDrawable drawable, int x, int y, int width, int
height)
    {
      GL gl=drawable.getGL();
      gl.glClear(GL.GL_COLOR_BUFFER_BIT | GL.GL_DEPTH_BUFFER_BIT);
      gl.glViewport(0, 0, width, height);
      return;
    }
    public void display(GLAutoDrawable drawable)
    {
      try {
            if(this._is_setup_done){
                        this.draw(drawable.getGL());
            }
      }catch(Exception e){
            e.printStackTrace();
      }
    }
    public void displayChanged(GLAutoDrawable arg0, boolean arg1, boolean arg2)
    {
    }
    public boolean isSetupDone(){
      return _is_setup_done;
    }
    public void mouseClicked(MouseEvent arg0) {}
    public void mouseEntered(MouseEvent arg0) {}
    public void mouseExited(MouseEvent arg0) {}
    public void mousePressed(MouseEvent arg0) {}
    public void mouseReleased(MouseEvent arg0) {}
    public void mouseDragged(MouseEvent arg0) {}
    public void mouseMoved(MouseEvent arg0) {}
    public abstract void setup(GL i_gl) throws Exception;
    public abstract void draw(GL i_gl) throws Exception;
```

```
    public abstract void update();
}
```

Classe Game

```
import java.awt.Font;
import javax.media.opengl.GL;
import javax.media.opengl.GLAutoDrawable;
import com.sun.opengl.util.j2d.TextRenderer;
import jp.nyatla.nyartoolkit.jogl.sketch.GlSketch;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import jp.nyatla.nyartoolkit.markersystem.NyARMarkerSystemConfig;
import jp.nyatla.nyartoolkit.qt.utils.NyARQtCamera;
import jp.nyatla.nyartoolkit.qt.utils.QtCameraCapture;

public class Game extends GlSketch{

     protected NyARQtCamera camera;
     protected NyARGlMarkerSystem nyar;
     protected NyARGlMarkerSystemRender render;

     @Override
     public void setup(GL i_gl) throws Exception {

       this.size(640,480);
       NyARMarkerSystemConfig config = new NyARMarkerSystemConfig(640,480);
       this.camera=new                                                NyARQtCamera(new
QtCameraCapture(config.getScreenSize(),30.0f));
       this.nyar=new NyARGlMarkerSystem(config);
       this.render=new NyARGlMarkerSystemRender(this.nyar);

       i_gl.glEnable(GL.GL_DEPTH_TEST);
       this.camera.start();
     }

     @Override
     public void draw(GL i_gl) throws Exception {
       this.render.drawBackground(i_gl, this.camera.getSourceImage());
       this.render.loadARProjectionMatrix(i_gl);
       this.nyar.update(this.camera);
     }

     public void update() {
     }

     @Override
     public final void display(GLAutoDrawable drawable)
     {
       try {
             if(this.isSetupDone()){
                   synchronized(this.camera){
                         this.draw(drawable.getGL());
                         update();
                   }
             }
       }catch(Exception e){
```

```
            e.printStackTrace();
        }

        TextRenderer text;
          text = new TextRenderer(new Font("SansSerif", Font.BOLD, 14));
        text.beginRendering(100, 100);
        text.draw("teste", 100, 100);
        text.endRendering();
      }

}
```

Classe Level

```
import java.util.ArrayList;
import java.util.List;

public class Level {
     public List<String> side = new ArrayList<String>();
     public List<String> otherSide = new ArrayList<String>();

     public Level(String[] sideA, String[] sideB) {
       for (int i = 0; i < sideA.length; i++) {
             side.add(sideA[i]);
       }
       for (int i = 0; i < sideB.length; i++) {
             otherSide.add(sideB[i]);
       }
     }
}
```

Classe Main

```
public class Main {

     public static void main(String[] args) {
       new ScaleGame().run();
     }

}
```

Classe Object3D

```
import java.awt.Font;
import javax.media.opengl.GL;
import com.sun.opengl.util.GLUT;
import com.sun.opengl.util.j2d.TextRenderer;
import quicktime.qd3d.math.Vector3D;
import jp.nyatla.nyartoolkit.core.NyARException;
import jp.nyatla.nyartoolkit.core.types.matrix.NyARDoubleMatrix44;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;
import OBJLoader.OBJModel;

public class Object3D {

     protected int id;
     protected OBJModel model;
     protected String pattern;
     protected String name;

     protected int rotationAngle;
     protected Vector3D rotation;
     protected Vector3D scale;
     protected Vector3D position;
     protected Vector3D translation;

     protected GLUT glut;
     protected TextRenderer text;

     public Object3D(String modelName, String pattern, int size, GL gl) {
       model = new OBJModel(modelName, size, gl, false);
       name = modelName;
       this.pattern = pattern + ".pat";
       rotationAngle = 0;
       rotation = new Vector3D(0,0,0);
       scale = new Vector3D(1,1,1);
       translation = new Vector3D(0,0,0);
       position = new Vector3D(Integer.MAX_VALUE,Integer.MAX_VALUE,Integer.MAX_VALUE);
       text = new TextRenderer(new Font("SansSerif", Font.BOLD, 36));
       glut = new GLUT();
     }

     public int getID(){
       return id;
     }

     public String getPattern() {
       return pattern;
     }

     public void setID(int id) {
       this.id = id;
     }

     public void draw(NyARGlMarkerSystemRender render, GL gl) {
```

```
        try {
                render.loadMarkerMatrix(gl, getID());
        } catch (NyARException e) {
                e.printStackTrace();
        }
        gl.glRotated(rotationAngle, rotation.getX(), rotation.getY(), rotation.getZ());
        gl.glScaled(scale.getX(), scale.getY(), scale.getZ());
        gl.glTranslated(translation.getX(), translation.getY(), translation.getZ());
        model.draw(gl);
    }

    public void update(NyARGlMarkerSystem nyar) {
        try {
                NyARDoubleMatrix44 matrix = nyar.getMarkerMatrix(id);

                int newX = (int)Math.round(matrix.m03);

                if(newX % 2 == 1) newX += 1;
                position.setX(newX);

                float newY = (float)matrix.m13;
                if(Math.abs( newY - position.getY()) > 5)
                        position.setY(newY);

                float newZ = (float)matrix.m23;
                if(Math.abs(newZ - position.getZ()) > 5)
                        position.setZ(newZ);
        } catch (NyARException e) {
                e.printStackTrace();
        }
    }

    public float getX() {
        return position.getX();
    }

    public float getY() {
        return position.getY();
    }

    public float getZ() {
        return position.getZ();
    }
}
```

Classe Scale

```
import java.util.ArrayList;
import java.util.Calendar;
import java.util.List;
import javax.media.opengl.GL;
import OBJLoader.OBJModel;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystem;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;

public class Scale extends Object3D {

     private enum ScaleState{
       left, middle, right;
     }

     private ScaleState state = ScaleState.middle;
     private List<Weight> leftSide, rightSide;
     private long oldTime = Long.MAX_VALUE;
     private boolean stable = false;
     private OBJModel base;

     public Scale(String modelName, String pattern, int size, GL gl) {
       super(modelName, pattern, size, gl);

       rotation.setX(1);
       rotation.setY(0);
       rotation.setZ(0);

       scale.setX(2);
       scale.setY(15);
       scale.setZ(0.3f);

       leftSide = new ArrayList<Weight>();
       rightSide = new ArrayList<Weight>();
     }

     public ScaleState getState() {
       return state;
     }

     public void draw(NyARGlMarkerSystemRender render, GL gl) {
       super.draw(render, gl);
     }

     public void update(NyARGlMarkerSystem nyar) {

       super.update(nyar);

       int leftMin = 0, leftMax = 0;
       for(Weight obj : leftSide) {

              int aux = obj.getWeight() * (obj.getPosition() - 1);
              leftMin += aux;
              aux = obj.getWeight() * (obj.getPosition());
```

```
        leftMax += aux;
  }

  int rightMin = 0, rightMax = 0;
  for(Weight obj : rightSide) {
        int aux = obj.getWeight() * (obj.getPosition() - 1);
        rightMin += aux;
        aux = obj.getWeight() * (obj.getPosition());
        rightMax += aux;
  }

  if(leftMin > rightMax) {
        state = ScaleState.left;
        stable = false;
        oldTime = Long.MAX_VALUE;
  } else {
        if(rightMin > leftMax) {
              state = ScaleState.right;
              stable = false;
              oldTime = Long.MAX_VALUE;
        } else {
              state = ScaleState.middle;
              if(oldTime == Long.MAX_VALUE){
                    oldTime = Calendar.getInstance().getTimeInMillis();
              }
              long now = Calendar.getInstance().getTimeInMillis();
              if(now - oldTime > 2000) {
                    stable = true;
              }
        }
  }

  if(state == ScaleState.right)
        rotationAngle = -10;
  if(state == ScaleState.left)
        rotationAngle = 10;
  if(state == ScaleState.middle)
        rotationAngle = 0;
}

public void addLeftObject(Weight obj) {
  if(!leftSide.contains(obj))
        leftSide.add(obj);
  if(rightSide.contains(obj))
        rightSide.remove(obj);
}

public void addRightObject(Weight obj) {
  if(!rightSide.contains(obj))
        rightSide.add(obj);
  if(leftSide.contains(obj))
        leftSide.remove(obj);
}

public void removeObject(Weight obj) {
```

```
      if(leftSide.contains(obj))
            leftSide.remove(obj);
      if(rightSide.contains(obj))
            rightSide.remove(obj);
    }

    public boolean checkSolution(Level level) {
      if(!stable){
            return false;
      }

      if(!((level.side.size()  ==  leftSide.size()  &&  level.otherSide.size()  ==
rightSide.size()) ||
                  (level.side.size() == rightSide.size() && level.otherSide.size()
== leftSide.size())) )
      {
            return false;
      }


      int count1 = 0, count2 = 0;
      for(Weight obj : leftSide) {
            for(String s : level.side) {
                  if(obj.name.toLowerCase().equals(s.toLowerCase())){
                        count1++;
                  }
            }
            for(String s : level.otherSide) {
                  if(obj.name.toLowerCase().equals(s.toLowerCase())){
                        count2++;
                  }
            }
      }
      if(leftSide.size() != count1 && leftSide.size() != count2) {
            return false;
      }

      count1 = 0;
      count2 = 0;
      for(Weight obj : rightSide) {
            for(String s : level.side) {
                  if(obj.name.toLowerCase().equals(s.toLowerCase())){
                        count1++;
                  }
            }
            for(String s : level.otherSide) {
                  if(obj.name.toLowerCase().equals(s.toLowerCase())){
                        count2++;
                  }
            }
      }
      if(rightSide.size() != count1 && rightSide.size() != count2) {
            return false;
      }
      System.out.println("stable");
```

```
        stable = false;
        oldTime = Long.MAX_VALUE;
        return true;
      }
  }
```

Classe ScaleGame

```
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.color.ColorSpace;
import java.awt.image.BufferedImage;
import java.awt.image.ComponentColorModel;
import java.awt.image.DataBuffer;
import java.awt.image.DataBufferByte;
import java.awt.image.Raster;
import java.awt.image.WritableRaster;
import java.io.BufferedReader;
import java.io.File;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;
import java.nio.Buffer;
import java.nio.ByteBuffer;
import java.util.ArrayList;
import java.util.List;
import javax.imageio.ImageIO;
import javax.media.opengl.*;
import jp.nyatla.nyartoolkit.core.NyARException;

public class ScaleGame extends Game
{
     private Scale scale;
     private List<Weight> sceneObjects;
     private String fasesFile = "fases\\";
     private List<String> fases;
     private BufferedImage bar;
     private Image barImage;
     private WritableRaster raster;
     private List<Level> gameLevel;
     private int actualLevel = 0;
     private int tutorial = 0;
     private boolean gameOver = false;

     public void setup(GL gl)throws Exception
     {
       super.setup(gl);
       sceneObjects = new ArrayList<Weight>();

       scale = new Scale("space", "balanca", 20, gl);
       int id = nyar.addARMarker(scale.getPattern(),16,25,40);
       scale.setID(id);

       sceneObjects.add(new Weight("mercurio", "mercurio", 1 * 5, gl, 1));
       sceneObjects.add(new Weight("venus", "venus", (int) (2.5 * 5), gl, 15));
       sceneObjects.add(new Weight("terra", "terra", (int) (2.6 * 5), gl, 18));
       sceneObjects.add(new Weight("marte", "marte", (int) (1.4 * 5), gl, 2));
       sceneObjects.add(new Weight("jupiter", "jupiter", (int) (29 * 5), gl, 5758));
       sceneObjects.add(new Weight("saturno", "saturno", (int) (24.7 * 5), gl, 1724));
```

```
	sceneObjects.add(new Weight("urano", "urano", (int) (10.4 * 5), gl, 264));
	sceneObjects.add(new Weight("netuno", "netuno", (int) (10.1 * 5), gl, 312));

	for(Object3D obj : sceneObjects) {
		id = nyar.addARMarker(obj.getPattern(),16,25,40);
		obj.setID(id);
	}


	String file = fasesFile + "fases.txt";
	fases = new ArrayList<String>();

	gameLevel = new ArrayList<Level>();

	BufferedReader br;
	try {
		br = new BufferedReader( new FileReader(file) );
		String line = "";
		while (((line = br.readLine()) != null)) {
			fases.add(line);
			System.out.println(line);
			String[] sides = line.split("X");
			String[] sideA = sides[0].split(",");
			String[] sideB = sides[1].split(",");
			for (int i = 0; i < sides.length; i++) {
				System.out.println(sides[i]);
				String[] planets = sides[i].split(",");
				for (int j = 0; j < planets.length; j++) {
					System.out.println(planets[j]);
				}
			}
			Level level = new Level(sideA, sideB);
			gameLevel.add(level);
		}
	    br.close();
	} catch (FileNotFoundException e) {
		e.printStackTrace();
	} catch (IOException e) {
		e.printStackTrace();
	}

	File f = new File("interface\\moon.jpg");
	barImage = null;
	try {
		barImage = ImageIO.read(f);
	} catch (IOException e) {
		e.printStackTrace();
	}
	raster     =          Raster.createInterleavedRaster     (DataBuffer.TYPE_BYTE,
barImage.getWidth(null),
			barImage.getHeight(null), 4,null);
			   ComponentColorModel   colorModel   =   new   ComponentColorModel
(ColorSpace.getInstance(ColorSpace.CS_sRGB), new int[] {8,8,8,8},
		true, false, ComponentColorModel.TRANSLUCENT, DataBuffer.TYPE_BYTE);
	  bar =  new BufferedImage (colorModel, raster, false, null);
```

```
    }

    public void draw(GL gl)throws Exception
    {
      super.draw(gl);

      if(nyar.isExistMarker(scale.getID()))
            scale.draw(render, gl);

      for(Weight obj : sceneObjects) {
            if(nyar.isExistMarker(obj.getID())){
                  obj.draw(render, gl);
            }
      }

      Graphics2D g = bar.createGraphics();
        g.drawImage(barImage, 0, 0, null);
      Font font = new Font("Times", Font.BOLD, 24);
      g.setFont(font);

        switch (tutorial) {
        case 0:
      g.setColor(Color.white);
            g.drawString("Enquadre o marcador da gangorra.", 50, 30);
            break;
        case 1:
      g.setColor(Color.yellow);
            g.drawString("Ponha o marcador da Terra na gangorra.", 50, 30);
            break;
      default:
            if(!gameOver) {
                  if(actualLevel % 2 == 0){
                  g.setColor(Color.white);
                  } else {
                  g.setColor(Color.yellow);
                  }
                  Level l = gameLevel.get(actualLevel);
                  String sideA = "";
                  for(String s : l.side) {
                        sideA += s + " ";
                  }
                  String sideB = "";
                  for(String s : l.otherSide) {
                        sideB += s + " ";
                  }
                  g.drawString("Equilibre " + sideA + "com " + sideB, 50, 30);
            }else{
                  g.drawString("Parabéns, você completou todos os desafios!", 50,
30);
            }
            break;
      }

        DataBufferByte dukeBuf = (DataBufferByte)raster.getDataBuffer();
        byte[] bytes = dukeBuf.getData();
```

```
        Buffer barBuffer = ByteBuffer.wrap(bytes);
       gl.glDrawPixels(barImage.getWidth(null), barImage.getHeight(null), GL.GL_RGBA,
GL.GL_UNSIGNED_BYTE, barBuffer);
     }

     @Override
     public void update() {
       for(Weight obj : sceneObjects) {
             try {
                   if(nyar.isExistMarker(obj.getID())) {
                         obj.update(nyar);
                         if(isLeftSide(obj)) {
                               scale.addLeftObject(obj);
                               obj.setPosition((int) (scale.getX() - obj.getX() -
40));
                         } else {
                               if(isRightSide(obj)) {
                                     scale.addRightObject(obj);
                                     obj.setPosition((int)       (obj.getX()       -
scale.getX() - 40));
                               } else {
                                     scale.removeObject(obj);
                               }
                         }
                   } else {
                         scale.removeObject(obj);
                   }
             } catch (NyARException e) {
                   e.printStackTrace();
             }

       }

       try {
             if(nyar.isExistMarker(scale.getID())) {
                   scale.update(nyar);
             }
       } catch (NyARException e1) {
             e1.printStackTrace();
       }


       switch (tutorial) {
       case 0:
             try {
                   if(nyar.isExistMarker(scale.getID())) {
                         tutorial++;
                   }
             } catch (NyARException e) {
                   e.printStackTrace();
             }
             break;
       case 1:
             try {
                   if(nyar.isExistMarker(sceneObjects.get(2).getID())) {
```

```
                        if(isLeftSide(sceneObjects.get(2))                        ||
isRightSide(sceneObjects.get(2))){
                              tutorial++;
                        }
                  }
            } catch (NyARException e) {
                  e.printStackTrace();
            }
            break;
      default:
            if(!gameOver)
                  if(scale.checkSolution(gameLevel.get(actualLevel))){
                        actualLevel++;
                        if(actualLevel >= gameLevel.size()) {
                              gameOver = true;
                        }
                  }

            break;
      }

    }

    private boolean isRightSide(Weight obj) {

      if(positionProblem(obj)) {
            return false;
      }
      if(scale.getX() < obj.getX()) {
            return true;
      }

      return false;
    }

    private boolean isLeftSide(Weight obj) {
      if(positionProblem(obj)) {
            return false;
      }

      if(scale.getX() > obj.getX()){
            return true;
      }

      return false;
    }

    private boolean positionProblem(Weight obj) {

      if(Math.abs(obj.getY() - scale.getY()) > 20 ) {
            return true;
      }
      if(Math.abs(obj.getZ() - scale.getZ()) > 20 ) {
            return true;
      }
```

```
      return false;
    }
}
```

Classe Weight

```
import javax.media.opengl.GL;
import quicktime.qd3d.math.Vector3D;
import jp.nyatla.nyartoolkit.jogl.utils.NyARGlMarkerSystemRender;

public class Weight extends Object3D{

     public enum ScaleSide {
       left,
       right,
       none
     }

     private ScaleSide side;
     private int position;
     private int weight;

     public Weight(String modelName, String pattern, int size, GL gl, int weight) {
       super(modelName, pattern, size, gl);

       this.weight = weight;
       side = ScaleSide.none;
       translation = new Vector3D(0,0,size/2 + 30);
     }

     public void draw(NyARGlMarkerSystemRender render, GL gl) {
       super.draw(render, gl);

     }

     private ScaleSide getSide(){
       return side;
     }

     private void setSide(ScaleSide side){
       this.side = side;
     }

     public int getPosition(){
       return position;
     }

     public void setPosition(int position){
       this.position = position;
     }

     public int getTorque() {
       return weight * 10 * position;
     }

     public int getWeight(){
       return weight;
     }
}
```

APÊNDICE E – ARTIGO

Testes de Viabilidade de desenvolvimento de aplicações de Realidade Aumentada em Java

Jaime Paz Lopes

Departamento de Informática e Estatística – Universidade Federal de Santa Catarina (UFSC)

jaimelopes@inf.ufsc.br

Abstract. *This article seeks to validate the possibility of using Augmented Reality in ordcer to develop programs in Java. The importance of this study is to add Augmented Reality technics to "Univerciência" project (which is a virtual educational world focused on young students) because the use of Augmented Reality would be helpful in picking the young students attention with the high level of interactivity proportionate by this kind of technology.*

Resumo. *O presente artigo trata da validação da possibilidade de se desenvolver programas em Java utilizando a Realidade Aumentada como interface homem-máquina. Essa validação se faz necessária, pois se pretende adicionar ao projeto Univerciência (que é um mundo educacional virtual focado no Ensino Básico) funcionalidades em Realidade Aumentada, buscando ter como consequência, despertar maior interesse no estudante, devido ao alto grau de interatividade proporcionada pelo uso desta tecnologia.*

1. Introdução

O tema deste artigo nasce do interesse de se incorporar recursos de Realidade Aumentada ao jogo Universo de Ciências (Univerciência), que é um projeto educacional centrado

na educação, formal e não formal, de Ciências (Física, Biologia e Química) no Ensino Básico, direcionado a alunos de doze a quatorze anos.

O Univerciência vem sendo desenvolvido em parceria entre a empresa Mentes Brilhantes Brinquedos Inteligentes e o GeNESS, Centro de Geração de Novos Empreendimentos em Software e Serviços da Universidade Federal de Santa Catarina, com o apoio financeiro do CNPq, edital MCT/CNPq Nº 62/2009 RHAE – Pesquisador na Empresa, projeto 561723/2010-9.

O jogo Univerciência consiste em um mundo virtual formado por vários minijogos que serão explorados pelos alunos para reforçar a assimilação dos conteúdos apresentados em aula.

Com o avanço da tecnologia, foram criadas muitas formas de interação com os computadores. Recentemente, vem sendo dado destaque a formas de interação que adotam técnicas de Realidade Aumentada (EXAME, 2013), onde o usuário recorre a objetos reais para interagir com o computador de uma forma simples e natural (SANTIN, 2004; KIRNER, 2013), transportando o ambiente virtual para o seu espaço físico. A interação por Realidade Aumentada acontece em tempo real, decorrente da combinação de elementos virtuais e reais (AZUMA, 1997).

Essa combinação de fatores transforma a Realidade Aumentada em uma excelente ferramenta para estimular a utilização de jogos. Por isso, a ideia de incorporar recursos de Realidade Aumentada ao Univerciência, para atrair o aluno e facilitar a absorção do conteúdo didático.

Partindo desse interesse, este artigo consiste no desenvolvimento teste para validar se é possível utilizar a tecnologia da Realidade Aumentada para tornar o Univerciência mais imersivo e interativo, atraindo a atenção dos estudantes.

2. Realidade Aumentada

Antes de definir Realidade Aumentada, é necessário explicar o contexto onde ela se encontra, que é o da Realidade Misturada, pois RA nada mais é do que um tipo de Realidade Misturada (ZORZAL et al, 2005).

Realidade Misturada é um tipo de ambiente que combina o mundo real com um mundo virtual. No caso da RA, o mundo predominante é o mundo real (MILGRAM, 1994). Quando o

ambiente predominante é o virtual, classifica-se como Virtualidade Aumentada. A Ilustração 1 mostra o Diagrama de Realidade/Virtualidade Contínua.

Ilustração 1: Diagrama de Realidade/Virtualidade Contínua.
Fonte: Milgram (1994)

Esses ambientes de Realidade Misturada são computacionalmente produzidos utilizando-se *hardwares*, como câmeras ligadas a computadores, e *softwares*, que geram objetos tridimensionais virtuais sobrepostos nas imagens capturadas pela câmera, mostrando o resultado obtido para o usuário do sistema. Essas câmeras podem ser simples *webcams* integradas a um *notebook* até capacetes sofisticados, para aumentar a imersão no mundo gerado (AZUMA, 97).

A Ilustração 2 mostra um ambiente de Realidade Aumentada. As paredes, a mesa e o telefone são reais, foram capturados por uma câmera e enviados para o computador, que adicionou as duas cadeiras e o abajur.

Ilustração 2: Ambiente de Realidade Aumentada
Fonte: Azuma (1997)

Segundo Azuma (1997), um sistema de Realidade Aumentada deve possuir três características:

– combinar o ambiente real com elementos virtuais: o ambiente de Realidade Aumentada é gerado tendo como base o ambiente real, que é preenchido com objetos virtuais que o complementam;

– interativo em tempo real: esses ambientes devem oferecer meios para que o usuário possa realizar ações e observar as reações produzidas imediatamente; e

– apresentado em três dimensões: os objetos devem ser modelos tridimensionais que se mesclam da melhor forma possível no ambiente real, devendo aumentar, diminuir e girar de acordo com as necessidades do usuário.

Um dos principais diferenciais de desenvolver um programa com interface de comunicação em Realidade Aumentada é o ganho de interatividade que essa tecnologia proporciona. Com a RA, a interação com o sistema não é intermediada por teclado e *mouse*, mas fundamentalmente através de uma câmera que passa as informações geradas pelo usuário para o computador (SANTIN, 2004).

A interação em RA pode ser implementada de diversas formas, sendo a mais comum a utilização de marcadores para permitir a interação do usuário com o ambiente de Realidade Aumentada. A Ilustração 3 mostra um exemplo de marcador contendo uma imagem em preto e branco, que, ao ser captada por uma câmera, modifica o ambiente de RA, de acordo com as ações predefinidas.

Ilustração 3: Exemplo de marcador
Fonte: NyARToolKit (2013)

Um dos usos mais comuns de marcadores é associá-los a um objeto virtual 3D, o que torna possível ao usuário alterar, por exemplo, as propriedades de posição, ângulo e proporção do objeto 3D no ambiente de RA, manipulando de forma correspondente a posição, ângulo e proporção do marcador no ambiente real. Outra prática comum é associar um marcador a uma função, que pode ser, por exemplo, a de destruir todos os modelos de objetos 3D no ambiente ou até a de fechar o programa (SANTIN, 2006). A Ilustração 4 exibe um exemplo de manipulação de um objeto virtual 3D com um marcador.

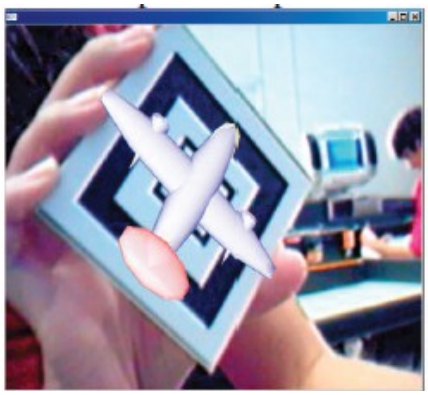


Ilustração 4: Manipulação de objeto virtual com marcador
Fonte: Santin (2004)

Para além do uso de marcadores, a interação em RA pode ser muito mais elaborada, tal como em alguns sistemas que podem oferecer luvas que captam os movimentos do usuário. Em outros ambientes, a forma de interação pode ser completamente livre de objetos físicos, o usuário, através de seus movimentos naturais, interage com os objetos virtuais (AZUMA, 1997).

3. Testes

Esta seção aborda, inicialmente, informações relevantes para a criação dos testes, e em uma etapa posterior, é descrito como os testes foram realizados e quais os resultados neles obtidos.

3.1. Linguagem de Programação

A linguagem de programação adotada para o desenvolvimento dos testes é a linguagem Java, pois esta é a usada no desenvolvimento do Univerciência.

3.2. Interação

A forma de interação escolhida foi a utilização de marcadores, pois esse tipo de interação é o que traz mais vantagens ao Univerciência. A principal é à facilidade de sua obtenção por alunos e professores no momento de sua utilização, bastando ter o arquivo digital com o desenho dos marcadores ou obter esse arquivo através da Internet. Assim, com o arquivo em mãos, o último passo é imprimir tantos marcadores quantos forem necessários. Outro ponto importante em relação à escolha da interação em Realidade Aumentada com marcadores é a facilidade de desenvolvimento, pois essa forma de interação é o padrão existente nas bibliotecas de RA, não sendo necessária uma implementação adicional.

3.3. Bibliotecas

Como já foi dito anteriormente, os testes devem ser desenvolvidos em Java, e uma série de bibliotecas deve ser escolhida para que eles sejam iniciados. São elas: biblioteca de Realidade Aumentada, biblioteca de captura de câmera e biblioteca de visualização gráfica. A biblioteca de Realidade Aumentada gerencia todo o sistema de marcadores, suas posições e modelos relacionados; a biblioteca de captura de câmera é responsável pela obtenção das imagens do ambiente real; por fim, a biblioteca gráfica reproduz as imagens captadas pela câmera sobrepondo aos marcadores os seus modelos tridimensionais.

Geralmente, uma biblioteca de Realidade Aumentada já vem integrada com as bibliotecas de captura de câmera e visualização gráfica (SOCIALCOMPARE, 2013), pois aquela é dependente destas duas outras bibliotecas para o funcionamento pleno de suas aplicações.

Várias bibliotecas para desenvolvimento em Java foram encontradas, mas a maioria era focada em Java Mobile (SOCIALCOMPARE, 2013), como AndAR e DroidAR, utilizadas para desenvolver aplicações para aparelhos portáteis com Android como sistema operacional.

Algumas também ofereciam opções para computadores de mesa, embora com número reduzido de funcionalidades. Por essas razões, essas opções de bibliotecas foram descartadas.

Uma biblioteca relativamente conhecida é a jARToolKit (JARTOOLKIT, 2013a), porém ela só permite que um único marcador seja detectado por vez (JARTOOLKIT, 2013b), impossibilitando vários tipos de aplicações e reduzindo suas possibilidades de uso. Assim, ela foi desconsiderada para adoção no desenvolvimento deste trabalho.

Outra biblioteca bastante explorada em aplicações de Realidade Aumentada desenvolvidas em Java é a NyARToolKit (NYATLA, 2013). Entretanto, ela foi desenvolvida no Japão, e só alguns documentos foram traduzidos oficialmente para o inglês, dificultando a compreensão da sua documentação. A biblioteca NyARToolKit foi criada em 2008 e vem sendo atualizada frequentemente. Sua última atualização foi no final de 2012.

Um problema enfrentado com a NyARToolKit é que, embora as opções de bibliotecas gráficas nativas sejam boas, elas não são as melhores opções quando o objetivo é desenvolver um jogo, pois não possuem diversas estruturas necessárias, que ficam a cargo de quem a utiliza. Por isso, foi criada a biblioteca ARMonkeyKit (ARMONKEYKIT, 2013a), que é uma adaptação da NyARToolKit para funcionar com jMonkeyEngine, uma poderosa ferramenta gráfica para criação de jogos. A biblioteca ARMonkeyKit foi criada em janeiro de 2010 e sendo atualizada até 2011, por essa razão ela usa versões antigas da NyARToolKit e da jMonkeyEngine. Em 2012, ela começou a ser reformulada para funcionar com versões mais atuais da NyARToolKit e jMonkeyEngine (ARMONKEYKIT, 2013b), porém nenhuma versão foi lançada até o momento da elaboração deste trabalho. Por essas razões, a ARMonkeyKit não foi usada neste projeto.

Como resultado, a NyARToolKit foi escolhida como a biblioteca de desenvolvimento de programas em Realidade Aumentada deste artigo. Mas ainda assim outras escolhas precisam ser feitas, uma referente à biblioteca de captura de câmera, outra referente à visualização gráfica. A biblioteca NyARToolKit é compatível com duas bibliotecas de captura de câmera e duas de visualização gráfica.

A escolha de uma biblioteca para captura de câmera foi simples, as opções eram Java Media Framework (JMF) e QuickTime. O JMF foi descontinuado e por isso não funciona corretamente na maior parte dos sistemas operacionais atuais, logo ele não poderia ser utilizado.

Já o QuickTime continua sendo atualizado e desenvolvido pela Apple. Assim, optou-se por utilizar o QuickTime no desenvolvimento deste projeto.

As opções para biblioteca gráfica eram Java3D e JOGL, sendo que a primeira a ser analisada foi a Java3D. Trata-se de uma biblioteca de alto nível, ou seja, abstrai vários elementos da manipulação de objetos tridimensionais e facilita o desenvolvimento. A Java3D possui também outras vantagens, como a criação de um grafo de cena, que simplifica a criação de dependências entre objetos e propaga as mudanças para todos os objetos que têm ligação com o objeto alterado. Outra vantagem é que ela possui uma ferramenta pronta para carregar os dados dos objetos tridimensionais para o cenário. Essa biblioteca começou a ser desenvolvida 1997 e continuou sendo atualizada até 2003. Em 2004 ela foi retomada como um projeto de código aberto, porém descontinuado. Apesar disso, possui uma versão bem evoluída e estável.

A JOGL é uma biblioteca de baixo nível, ou seja, seus comandos são primitivos e seus usuários devem criar suas próprias funções de alto nível. Ao contrário da Java3D, a biblioteca JOGL não tem a funcionalidade de abrir modelos tridimensionais, assim, essa função deve ser criada com base nos conjuntos de comandos existentes. Por outro lado, a JOGL é constantemente atualizada e possui diversos exemplos e tutoriais que são facilmente encontrados, razão pela qual ela foi a biblioteca gráfica escolhida para este trabalho.

Como foi dito, a JOGL não possui a função de carregar modelos tridimensionais, por isso este trabalho utiliza um carregador criado por Andrew Davison em seu livro Pro Java 6 3D Game Development (DAVISON, 2013). Essa ferramenta permite carregar modelos tridimensionais e texturas salvos no formato 3DS.

Em suma, como explicado neste subitem, este artigo adota as seguintes bibliotecas Java: NyARToolKIT (Realidade Aumentada), QuickTime (captura da câmera) e JOGL (visualização gráfica). Adicionalmente, adota-se o carregador de modelos de Andrew Davison (2013).

3.4. Realização dos Testes

Foram feitos quatro testes para verificar como funcionam as bibliotecas escolhidas e que tipo de resultados elas podem oferecer.

Esta seção apresenta o objetivo, a descrição detalhada e o resultado obtido em cada teste.

O primeiro teste consiste em verificar como é feita a captura de imagem da câmera, como detectar a presença de um marcador e como posicionar um objeto 3D simples em cima desse marcador.

Este primeiro teste pode parecer muito simples, mas com ele já é possível identificar o funcionamento básico da NyARToolKit e identificar que por ser fortemente integrada as outras bibliotecas, ela torna o desenvolvimento muito mais rápido. Por exemplo, através deste primeiro teste, sabe-se que é a classe NyARQtCamera (especialização da NyARSensor, que gerencia o processamento e captura das imagens da câmera de uma forma mais geral), que oferece ao desenvolvedor a manipulação de imagens QuickTime. Para criar um objeto NyARQtCamera é preciso ter um objeto da classe QtCameraCapture, que é a classe que realmente captura as imagens e as deixa prontas para serem acessadas pelo usuário através do método getSourceImage() da classe NyARQtCamera.

Outra classe muito importante nesse primeiro teste é a NyARGlMarkerSystem, que é responsável por gerenciar todos os marcadores e verificar se estão ou não sendo capturados. Ela é uma especialização da NyARMarkerSystem, mas já faz todos os tratamentos de criação, posição e rotação dos marcadores para ficar da forma que a JOGL entenda. Para usá-la basta criar um objeto NyARGlMarkerSystem, adicionar os marcadores através do método addARMarker(), atualizar a cada momento a imagem que ele recebe da câmera, passando um objeto da classe NyARQtCamera como parâmetro no método update() e verificar se existe ou não um marcador sendo capturado com o método isExistMaker(), que tem como único parâmetro uma referência a um marcador específico e retorna um valor verdadeiro caso esse marcador esteja sendo capturado.

A classe que termina a sequência desse primeiro teste é a NyARGlMarkerSystemRender, responsável por desenhar em cima da imagem que a câmera está capturando, adicionando nessa imagem os objetos associados aos marcadores. Basta usar o método loadMarkerMatrix() para carregar a posição do marcador e chamar os métodos que desenham o objeto que o representa. Como resultado, esse objeto já estará na posição correta. No caso desse teste, o objeto associado ao marcador é um cubo colorido.

O resultado desse primeiro teste foi considerado bem sucedido, o objeto apareceu em cima do marcador, respondendo muito bem aos movimentos realizados. Na Ilustração 5 é

possível ver qual foi o marcador utilizado e como o objeto aparece para o usuário durante a execução do programa de teste.

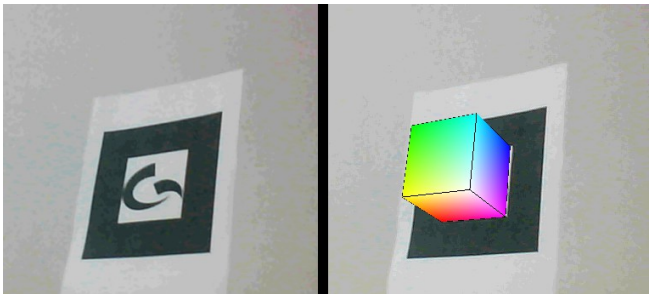


Ilustração 5: Marcador sem e com seu objeto associado

Um segundo teste realizado consiste em comprovar se realmente seria possível utilizar dois marcadores diferentes ao mesmo tempo.

O funcionamento básico desse teste é igual ao do teste anterior, com a única diferença de que dois marcadores foram adicionados no NyARGlMarkerSystem usando o método addARMarker() para cada um dos marcadores, e duas checagens do método isExistMaker() foram chamadas, uma para cada marcador.

Esse segundo teste também foi bem sucedido, sendo possível observar que com dois marcadores o sistema se comportou da mesma forma que o anterior com apenas um marcador. A Ilustração 6 mostra como cada um dos marcadores ficou durante a execução do programa.

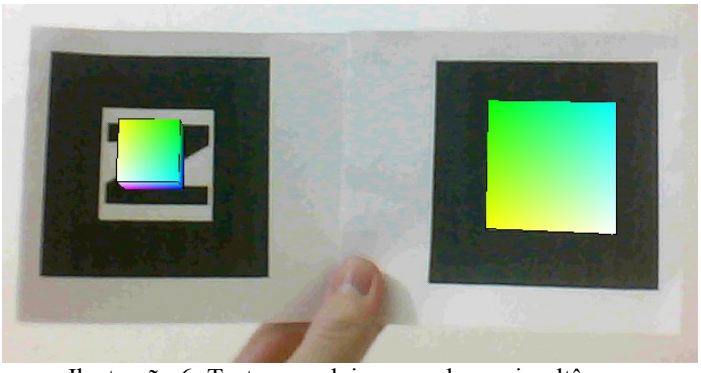


Ilustração 6: Teste com dois marcadores simultâneos

O terceiro teste realizado consiste em carregar um arquivo que descreve um objeto modelado e texturizado usando ferramentas para criação de objetos tridimensionais. Esse foi o teste usado para validar o carregador de objetos desenvolvido por Davison (2013).

Esse teste teve como base o código criado para o primeiro teste, com a adição da criação de um objeto Java da classe OBJModel. Esse objeto representa um modelo 3D que é posicionado em cima do marcador pelo método loadMarkerMatrix() da classe NyARGlMarkerSystemRender e desenhado com o seu próprio método draw(). A Ilustração 7 mostra o resultado desse teste.

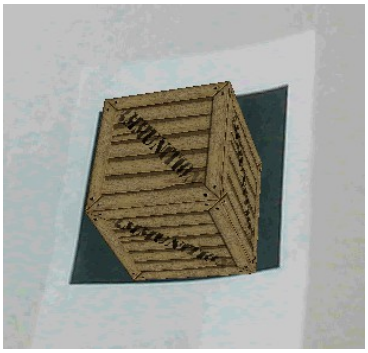


Ilustração 7: Objeto 3D
carregado de um arquivo .3DS

O quarto e último teste consiste em verificar como o sistema responderia a vários marcadores entrando e saindo da área de captura da tela. O código gerado nesse teste está no Apêndice A com o título Models.java.

Esse teste usou como base o teste anterior, mas com vários marcadores, cada um dos quais associado a um respectivo arquivo 3DS de modelo 3D com textura. Na Ilustração 8 é possível ver ao mesmo tempo na tela, cinco marcadores associados aos seus respectivos objetos 3D.

Ilustração 8: Cinco marcadores com seus modelos relacionados.

3. Conclusão

Este artigo tem como objetivo validar o uso de Realidade Aumentada no projeto Univerciência que vem sendo desenvolvido pelo GeNESS. Logo, tudo o que foi aprendido e desenvolvido nesse artigo contribuirá diretamente na geração de funcionalidades, onde os resultados aqui gerados serão reproduzidos.

Inicialmente, foi possível visualizar que os fundamentos teóricos que embasam a Realidade Aumentada já vêm sendo estudados há algum tempo e possuem diversas pesquisas que comprovam as vantagens de utilização desta tecnologia para diversas áreas de aplicação.

Posteriormente, foram realizados os testes que servem para validar o uso de RA, pondo em prática os conceitos correlatos levantados no capítulo anterior. Inicialmente, é demonstrado o levantamento das ferramentas para se desenvolverem aplicações de Realidade Aumentada em Java. Foi possível verificar que faltam soluções atualizadas e de fácil compreensão, ou seja, a maior parte das ferramentas estava desatualizada ou já havia sido abandonada há vários anos e a ferramenta mais adequada encontrada, e que foi usada na elaboração desse trabalho, possuía a maior parte da sua documentação em japonês. Apesar dessa barreira linguística, não foi difícil utilizá-la, pois ela já vinha completa e seus criadores tiveram o cuidado de criar dependências a mais de uma biblioteca. Assim, quando uma das opções era inadequada, existia outra opção pronta para ser usada. Essa biblioteca trata Realidade Aumentada através do uso de marcadores, o que cria mais uma dependência, além da já existente ao computador e a *webcam*, mas que é facilmente resolvida se existir um arquivo digital desses marcadores, permitindo ao usuário imprimir livremente quantos marcadores necessitar.

A partir dos resultados deste trabalho, considera-se que essa biblioteca (NyARToolKit) se mostrou eficiente e confirmou que é possível desenvolver aplicações de Realidade Aumentada para computador na linguagem Java, pois, ela mostrou um desempenho adequado em todos os testes desenvolvidos.

Referências

EXAME. "**15 Campanhas Inteligentes com Realidade Aumentada. Fevereiro**", 2013. Disponível em: <http://exame.abril.com.br/marketing/noticias/16-usos-inteligentes-de-realidade-aumentada-em-campanhas> Acessado em: Maio, 2013

SANTIN, M. "**Desenvolvimento de técnicas de interação para aplicações de realidade aumentada com ARToolKit**", I Workshop de realidade aumentada, 2004.

KIRNER, C., ZORZAL, E. R. "**Aplicações Educacionais em Ambientes Colaborativos com Realidade Aumentada**" Disponível em: <http://www.realidadeaumentada.com.br/artigos/13164.pdf>. Acessado em: Maio, 2013

AZUMA, R. et al. "**Recent Advances in Augmented Reality**". IEEE Computer Graphics and Applications 21, 6 (Nov/Dec 2001), p. 34-47.

ZORZAL, E. R., BUCCIOLI, A. A. B. and KIRNER, C. "**Desenvolvimento de Jogos em Ambiente de Realidade Aumentada**". SBGAMES – Simpósio Brasileiro de Jogos para Computador e Entretenimento Digital, WJogos, USP, São Paulo, SP. 2005.

MILGRAM, P. et al. "**Augmented Reality: A class of displays on the reality-virtuality continuum**", SPIE, V.2351, p. 282-292, 1994.

NYARTOOLKIT. **NyARToolKitCPP/2.4.0.** Disponível em: <http://nyatla.jp/nyartoolkit/wp/?p=448> Novembro, 2009. Acessado em: Abril, 2013.

SANTIN, M. "**Desenvolvimento de técnicas de interação para aplicações de realidade aumentada com ARToolKit**", I Workshop de realidade aumentada, 2004.

SOCIALCOMPARE. **Augmented Reality SDK Comparison** <http://socialcompare.com/en/comparison/augmented-reality-sdks> Maio, 2013. Acessado em: Maio, 2013.

JARTOOLKIT. **JARToolKit.** Disponível em: <http://sourceforge.net/projects/jartoolkit/> Acessado em: Maio, 2013a.

JARTOOLKIT. **ReadMe.txt**. Disponível em: <http://sourceforge.net/projects/jartoolkit/files/latest/download?source=navbar> Acessado em: Maio, 2013b.

NYATLA.. **NyARToolKit**. Disponível em: <http://nyatla.jp/nyartoolkit/wp/>. Acessado em: Maio. 2013

ARMONKEYKIT. **ARMonkeyKit.** Disponível em: <http://armonkeykit.wordpress.com/>. Acessado em: Maio, 2013a.

ARMONKEYKIT. **Porting ARMonkeyKit to JME3 some questions.** Disponível em: <http://jmonkeyengine.org/forum/topic/porting-armonkeykit-to-jme3-some-questions/> Acessado em: Maio, 2013b.

DAVISON, Andrew. **Chapter 17. Picking the models.** Disponível em: <http://fivedots.coe.psu.ac.th/~ad/jg2/ch17/index.html> Acessado em: Maio, 2013.