UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA - CT
DEPARTAMENTO DE ELETRÔNICA E COMPUTAÇÃO
CURSO DE CIÊNCIA DA COMPUTAÇÃO - CCC

APLICATIVO PARA DETECÇÃO DE MOVIMENTOS PARA REABILITAÇÃO FISIOTERAPÊUTICA UTILIZANDO MICROSOFT KINECT

TRABALHO FINAL DE GRADUAÇÃO

Luiz Eduardo Viegas Flores

Santa Maria, RS, Brasil
2011

APLICATIVO PARA DETECÇÃO DE MOVIMENTOS PARA REABILITAÇÃO FISIOTERAPÊUTICA UTILIZANDO MICROSOFT KINECT

Por

Luiz Eduardo Viegas Flores

Monografia apresentada ao Curso de Ciência da
Computação do Departamento de Eletrônica e Computação da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação.

Orientador: Prof. Marcos Cordeiro d'Ornellas (UFSM)

Trabalho de Graduação N.333
Santa Maria, RS, Brasil
2011

Universidade Federal de Santa Maria
Centro de Tecnologia
Curso de Ciência da Computação

A Comissão Examinadora, abaixo assinada,
aprova o Trabalho de Graduação

APLICATIVO PARA DETECÇÃO DE MOVIMENTOS PARA REABILITAÇÃO FISIOTERAPÊUTICA UTILIZANDO MICROSOFT KINECT

elaborado por
Luiz Eduardo Viegas Flores

como requisito parcial para obtenção do grau de
Bacharel em Ciência da Computação

COMISSÃO EXAMINADORA:

_____
Prof. Marcos Cordeiro D'Ornellas (UFSM)
(Presidente/Orientador)

_____
Profa. Lisandra Manzoni Fontoura (UFSM)

_____
Prof. Cesar Tadeu Pozzer UFSM)

Santa Maria, 20 de dezembro de 2011.

AGRADECIMENTOS

Gostaria de agradecer ao apoio incondicional dado por meus pais e meu irmão Bruno à conclusão deste trabalho, e principalmente deste curso. Obrigado por tudo, principalmente pelos conselhos, mesmo parecendo que tudo "entrava por um ouvido e saia pelo outro". Vocês definiram quem eu sou.

Aos meus grandes amigos de discussões, baladas, nerd-nights e samba do Orlando, Delbio, Lucas e o saudoso Vivico, que começou a namorar e sumiu! Boas lembranças...das discussões acirradas sobre qual era o melhor curso da UFSM, dos debates políticos e filosóficos sobre os quais, no outro dia, ninguém se lembrava de nada.

Aos meus amigos Mario e Adriel pelo convívio e amizades que transcenderam o tempo e que na dificuldade ou não, estavam sempre prontos para oferecer a mão, ou um braço. Tudo pelos amigos.

Aos meus companheiros de todo dia, os cachorros Girafa, Bola e a novata Nina, pelo companheirismo nas noites de programação, nas noites de stress, nas noites de tristeza, nas noites de alegria. Melhor amigo do homem é uma verdade subestimada.

Aos meus colegas de faculdade, o grande amigo Bernardo pelas risadas, histórias e o reader e blogs, fielmente vistos todos os dias no LaCA. Ao amigo Carioca, que provavelmente deve estar se esquecendo de alguma prova nesse exato momento, ao Fabiano pela identificação imediata de interesses, e discussões. Aos demais colegas do LaCA, que além de me ajudarem diretamente, indiretamente contribuíram para amadurecimento de várias ideias. Principalmente ao grande companheiro Lamarck, que nesses anos de convívio me ensinou muitas lições importantes de vida e manteve minha cabeça no lugar várias e várias vezes e várias vezes.

Em especial, à pessoa que mais me ajudou a perceber o que realmente é importante nessa vida, a dar valor à tudo isso. Sem ela, nada disso teria sentido ou seria possível. Gabriela, obrigado, meu amor.

- ao meu avô Crescêncio

RESUMO

Trabalho de Graduação
Curso de Ciência da Computação
Universidade Federal de Santa Maria

APLICATIVO PARA DETECÇÃO DE MOVIMENTOS PARA REABILITAÇÃO FISIOTERAPÊUTICA UTILIZANDO MICROSOFT KINECT

AUTOR: LUIZ EDUARDO VIEGAS FLORES
ORIENTADOR: PROF. MARCOS CORDEIRO D'ORNELLAS
Local e data da defesa final: Santa Maria, 20 de Dezembro de 2011

Vítimas de lesões e acidentes envolvendo a musculatura do braço necessitam de tratamento fisioterápico por meses ou por anos, passando por repetitivos exercícios fisioterápicos. Diversas alternativas de apoio ao tratamento têm surgido, com a utilização das recentes tecnologias de detecção de movimento como o Wii, Kinect e The Eye. Com o intuito verificar a viabilidade do Microsoft Kinect no desenvolvimento de aplicativos voltados para essa área, foi desenvolvido um protótipo integrado à *engine* Unity 3D. O usuário pode calibrar o programa para dinamicamente gravar diversos tipos de movimentos com o braço. Após esta gravação, são criados *GameObjects* da Unity 3D em posições específicas relativas ao movimento gravado. Para realizar a detecção completa, o usuário deve atingir todos esses objetos sequencialmente. O gesto pode ser detectado em sua totalidade, ou parcialmente. Os dados do movimento realizado são disponibilizados para o usuário a fim de medir o progresso obtido. Espera-se que os resultados desse trabalho sejam utilizados no desenvolvimento de jogos voltados à fisioterapia.

Palavras-chave: Microsoft Kinect, Reabilitação Fisioterapêutica, Manguito-Rotador, Unity3D

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS

API – Application Programming Interface

HUSM – Hospital Universitário de Santa Maria

IR – InfraRed

LaCA – Laboratório de Computação Aplicada

NUI – Natural User Interface

QVGA – Quarter vídeo graphics *array*

RGB - Sistema de cor Red, Green, Blue (vermelho, verde, azul).

SDK – Software Development Kit

TOF – Time of Flight

VGA – Video graphics *array*

XYZ – Eixos x, y e z

SUMÁRIO

1 INTRODUÇÃO

O corpo humano é de uma complexidade difícil de ser reproduzida, mesmo pelo mais poderoso computador. Não é rara a ocorrência de problemas, especialmente na época atual, em que um indivíduo se expõe à possibilidade de contrair diversas lesões cotidianamente, provavelmente geradas por acidentes ou esforços repetitivos. Tais problemas podem ser detectados com um diagnóstico simples para a medicina, mas o tratamento é incrivelmente incômodo para os diagnosticados. Esses pacientes, após a realização dos procedimentos médicos convencionais, frequentemente sofrem com sequelas como perda de parte dos movimentos do membro lesionado, dores crônicas e impossibilidade de executar algumas ações. Essas enfermidades são abrandadas ou curadas por sessões intensivas de reabilitação fisioterapêutica.

A maior parte da terapia envolve a execução de exercícios específicos para cada caso, o que significam muitas horas de atividades repetitivas e exaustivas, que geralmente envolvem o deslocamento do paciente à clínica de reabilitação. Esses motivos são a principal causa da evasão ao tratamento por parte dos pacientes.

Em visitas ao setor de fisioterapia do HUSM, alguns pacientes mencionaram estar passando pela segunda vez pela reabilitação de uma lesão relacionada ao Manguito Rotador. Na primeira vez, após certo tempo de tratamento, as dores aparentemente sumiram. Mesmo contra a recomendação do Fisioterapeuta os pacientes abandonaram o tratamento, somente para depois descobrir que sua condição tinha retornado. Pensando nessas ocorrências, tem-se estudado opções viáveis para diminuir a taxa de evasão aos tratamentos.

Com ascensão de tecnologias com suporte para detecção de movimento e interfaces naturais, como Kinect, Playstation The Eye e WII, originou-se o estudo do uso desses recursos no apoio à reabilitação. A idéia básica é proporcionar diversão, concomitantemente à execução dos exercícios de fisioterapia, com o objetivo de recondicionar o corpo e a mente do paciente através dos movimentos simulados em jogos desenvolvidos para essas plataformas. Adicionalmente, os sistemas capazes de realizar o mapeamento do esqueleto tornam possível a obtenção e disponibilização dos dados do movimento executado.

Recentemente o LaCA recebeu 2 aparelhos Microsoft Kinect, destinados a projetos de pesquisa. Com estes recursos e a parceria firmada com o Curso de Fisioterapia – UFSM foi possível a idealização e execução deste Trabalho Final de Graduação.

1.1 Justificativa de escolha do tema

A principal justificativa para escolha do tema foi a possibilidade de trabalho sobre um tema de computação que trouxesse melhorias na qualidade de vida das pessoas.

Desde que se iniciaram os trabalhos sobre o Microsoft Kinect na UFSM (Através do LaCA), foram realizadas pesquisas sobre principais novidades no ramo de desenvolvimento de software que utilizam as novas tecnologias de detecção de movimento. Foi constatado que a área de pesquisa que envolve busca de tratamentos alternativos de reabilitação fisioterápica está em expansão. As capacidades de detecção do mapa de profundidade 3D e rastreamento de esqueletos a baixo custo do Kinect despertaram interesse na realização de um estudo sobre sua viabilidade como hardware primário de entrada em aplicativos de apoio ao tratamento fisioterápico. Em seguida, foi necessário escolher especificamente algum músculo do corpo para ser abordado no trabalho, pois foi definido no escopo do trabalho que a detecção de movimentos de um grupo de músculos seria suficiente para demonstrar a viabilidade da idéia deste trabalho. Optou-se então pela avaliação dos exercícios de Fisioterapia dos Braços e ombros (Através do Manguito Rotador), para facilitar a replicação dos exercícios e dos testes. Nada impede, entretanto, que os conhecimentos apresentados sejam utilizados na implementação de aplicativos identificação de movimentos de outras partes do corpo.

1.2 Objetivo Geral

O objetivo geral deste trabalho versa sobre a demonstração da viabilidade do desenvolvimento de jogos voltados à reabilitação fisioterapêutica, utilizando o sistema de *Skeleton Tracking* do Microsoft Kinect.

1.3 Objetivos Específicos

- Estudo do Hardware do Kinect, limites e possibilidades;
- Identificação dos requisitos de software e hardware do aplicativo a ser desenvolvido através do estudo do problema fisioterapêutico em questão;
- Estudo de uma solução para a detecção de movimentos específicos dos exercícios de fisioterapia através do sistema de *Skeleton Tracking* do Kinect;
- Implementação de um protótipo integrado com a engine Unity 3D para demonstrar a viabilidade da solução na criação de jogos voltados à reabilitação fisioterapêutica;

2 REVISÃO BIBLIOGRÁFICA E FUNDAMENTAÇÃO

Nessa sessão, são detalhados os princípios teóricos referentes ao grupo de músculos Manguito Rotador, a fim de contextualizar o problema à área de computação e identificar requisitos do protótipo a ser desenvolvido. Esse capítulo versa também sobre o Kinect e sua SDK, avaliando suas capacidades, a fim de comprovar sua viabilidade para resolução do problema em questão. Em seguida são explicados as principais especificações da engine Unity 3D abordadas na implementação do protótipo, e como foi realizada sua integração com a SDK Microsoft Kinect.

2.1 Manguito Rotador

O Manguito Rotador é um conjunto de 4 músculos: subescapular, supra-espinhoso, infra-espinhoso e redondo menor (Figura 2-1). Esses músculos cobrem a cabeça do úmero e são responsáveis pela estabilização, força e mobilização. É também, um dos principais causadores de dores e lesões nos ombros e braços.

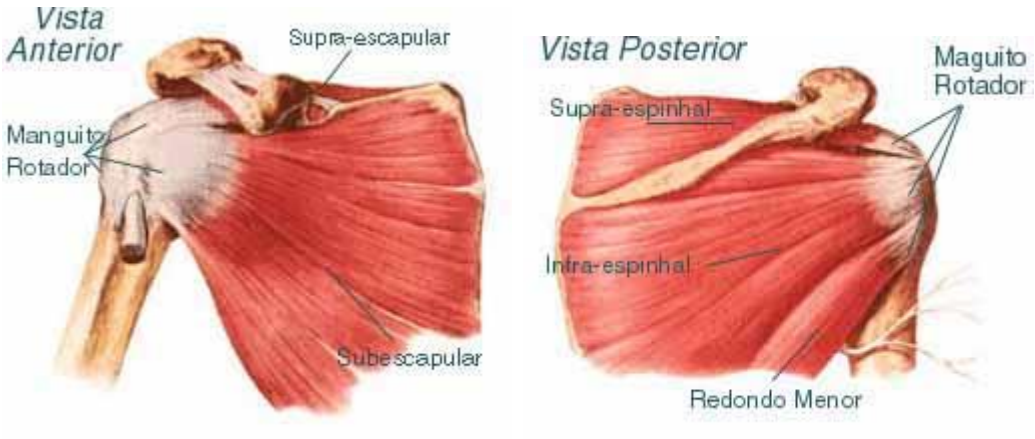

Figura 2-1 – Manguito Rotador (Clinica Deckers, 2009).

Existem diversas formas de lesionar o ombro: Impacto por quedas e acidentes (fraturas, luxação, distensão muscular), movimentos repetitivos (tendinites e bursites) e degeneração (artrose). Porém a dor pode aparecer mesmo sem ter havido nenhuma lesão. Normalmente isso acontece devido a uma mecânica de movimento faltosa ao elevar o braço (MATSUO, Tatiana, 2010) (Figura 2-2).

Figura 2-2 – Paciente com o manguito-rotador lesionado (Rotator Cuff Injury, 2008).

Todo o movimento do braço está diretamente conectado à escápula que é um osso chato que se encontra nas costas. Quando um dos ombros é elevado em direção às orelhas com a mão sobre ele e dedos sobre o osso mais atrás, sente-se que esse osso desliza pra cima e, quando o ombro é distanciado das orelhas ele escorrega para baixo. Esse deslocamento da escápula deve ser feito sem muito esforço, para não causar tensões no trapézio, músculo próximo ao pescoço. No dia a dia, ocorre naturalmente a compressão do ombro, trazendo-os para cada vez mais perto das orelhas, seja ao dirigir ou atender à algum telefone, gerando esse novo hábito de manter os ombros elevados, o que vai influenciar no movimento dos braços. O problema é que na escápula existe uma ponta saliente na parte superior sobre o ombro, quase na articulação, o acrômio. Quando o ombro está elevado, essa ponta vai deslizar à frente podendo pinçar o nervo ou o tendão que passa por baixo dela indo em direção ao braço. Esse pinçamento ou compressão poderá trazer lesões caso o braço seja elevado acima da altura dos ombros com uma sobrecarga, seja por carga muito pesada, ou esforço repetitivo. Esse problema pode até ocasionar ruptura parcial ou total do tendão (MATSUO, Tatiana, 2010).

Quando uma lesão de um paciente é tratada não necessariamente é solucionada a causa, pois, se esta lesão se originou de um movimento faltoso, ela não será corrigida automaticamente. Após o trauma é normal que o corpo crie movimentos compensatórios para se proteger da dor e compensar o trabalho do músculo lesionado, agravando ainda mais a situação. Não é raro surgir a memória da dor, que, mesmo após a recuperação da lesão, faz com que toda a estrutura trave e se contraia, tentando evitá-la. Porém essa contração repentina causa uma dor pior quando o músculo espreme o nervo. Muitos pacientes, aparentemente recuperados da lesão, permanecem com uma dor de intensidade crescente, pois o corpo continua se protegendo. Ao evitar o movimento do membro, o ombro se enfraquece e a dor aumenta, porém ao tentar mexer, o ombro continua a doer, transformando-se no chamado

ciclo da dor. Essa tensão muscular pode levar a tendinites e outras inflamações, pois o músculo não consegue descansar e se recuperar (Clinica Deckers).

Na quebra do ciclo de dor do paciente é importante que sejam introduzidos movimentos de soltura em forma de pêndulo. Com o corpo todo relaxado e, com a mão sobre o ombro dolorido, o paciente vai girando o corpo e deixando o braço balançar. Esse movimento deve ser repetido em várias direções, e posições do ombro diferentes.

Quando a lesão sai do estágio agudo o paciente deve começar a fortalecer e aumentar a amplitude de movimento. É importante lembrar que, fazer a correção do movimento num exercício lento não é tão difícil, o mais complicado é a execução dos movimentos corretos em uma situação cotidiana, quando o paciente tem muitas outras coisas em mente para se preocupar (SCHRONAUER, 2010).

Em casos posteriores à fase aguda da lesão a dor ainda permanece, como no caso dos ombros congelados, é de extrema importância que se elimine a memória da dor, para efetivamente realizar-se a quebra do ciclo. Para isso, é necessário fazer com que o corpo aceite novamente o movimento correto do ombro.

O método ideal indicado por fisioterapeutas é a realização repetitiva de movimentos naturais, pois o cérebro interpreta a necessidade cotidiana da movimentação do ombro com menos esforço do que exercícios pré-estipulados. Além disso, a maneira com que são realizados os exercícios é despercebida pelo paciente, diminuindo a carga de stress e aliviando a dor, pois o cérebro está ocupado em outras atividades (SCHRONAUER, 2010).

Por conseguinte, a utilização de jogos com interface natural é altamente recomendada, em adição aos exercícios convencionais do paciente. O desafio e diversão proporcionados pelos *games* ajudará a quebrar o ciclo da dor no cérebro. Como exemplo, na Figura 2-3 observa-se o resultado da adição de uma interface de entrada no jogo *World of Warcraft* através de um hack do Kinect. Os movimentos utilizados são muito parecidos com o exercício de alongamento de Biceps (Figura 2-4).

Figura 2-3 – Exemplo de Movimento utilizado para controlar um jogo (MIT, 2010).

Figura 2-4 – Exemplo de exercícios para a síndrome do ombro congelado (Clinica Deckers, 2008).

Através do estudo do posicionamento das partes do corpo nos exercícios mostrados na Figura 2-4, foi constatado que um movimento de reabilitação fisioterapêutica do Manguito Rotador envolve controle do ângulo de abertura dos braços, posição dos cotovelos, ombros, pulsos e mãos. Baseando-se nessas premissas foi possível identificar os requisitos para a execução correta de algum exercício específico de fisioterapia.

2.2 Microsoft Kinect

O dispositivo, primariamente projetado para o Videogame Microsoft XBOX, permite que jogadores interajam com o console sem a utilização de controles/joysticks através da detecção tridimensional dos esqueletos dos jogadores, mapeando os pontos de articulação do corpo. Por conseguinte, a interação é realizada através de gestos ou movimentos corporais. Os sensores do Kinect incluem uma câmera RGB, um sensor e um projetor IR (infrared), utilizado em mapeamentos de profundidade e um grupo de microfones de confiabilidade na detecção de vozes, com algoritmos de cancelamento de eco e de ruído. Por fim, o aparelho possui um motor de inclinação, que permite o movimento dos sensores no eixo vertical, para mudanças no ângulo de visão (Microsoft, 2011). Os dados técnicos do Kinect são apresentados na Tabela 2-1.

Tabela 2-1 - Especificações Técnicas do Microsoft Kinect

Ângulo de Visão	43º vertical e 57º horizontal
Ângulo de inclinação do motor mecânico	~28º
Taxa de Atualização de tela	30 FPS
Resolução, câmera de profundidade	QVGA (320x240)
Resolução, câmera RGB	VGA(640x480)
Características dos Microfones	Combinação de 4 microfones com conversores analog-to-digital(ADC) de 24 bits, com cancelamento de eco e supressão de ruído

É útil ressaltar o fato de que para a detecção correta dos movimentos dos jogadores, é necessário um espaço grande, em uma escala relativa à uma sala comum. Portanto, um ambiente ideal de uso do Kinect deve ter espaço lateral de movimentação suficiente para, no mínimo, um jogador. Adicionalmente, o espaço deve ter uma profundidade de no mínimo 4 metros (Figura 2-5). Outra limitação encontrada é a taxa de atualização das câmeras e do sistema de detecção, 30 FPS, considerada insuficiente para aplicações nas quais seja necessário detectar movimentos muito curtos ou muito rápidos.

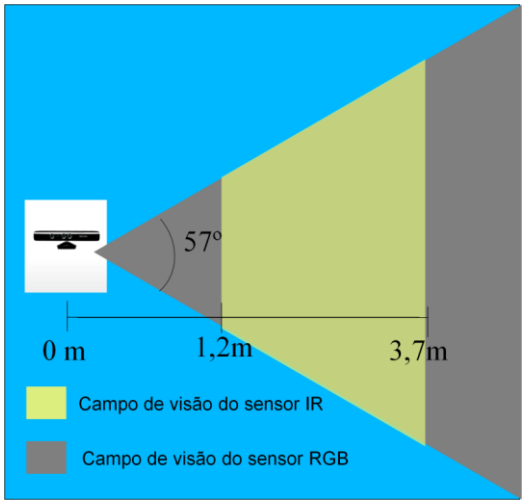

Figura 2-5 – Vista de topo do Campo de visão do Kinect.

O Kinect é o resultado de pesquisa da Microsoft em cima de uma solução tecnológica criada pela empresa Israelense Primesense, especializada em visão computacional e câmeras de mapeamento 3D (KIPPMAN, Alex). A Microsoft, empresa conhecida por ser bem sucedida na compra de projetos promissores, em parceria com a Primesense, anunciou o Projeto Natal. Como resultado, em 2009, na E3, foi anunciado o Microsoft Kinect, que impressionou a comunidade *gamer* pelas suas capacidades. Além disso, o aparelho atraiu olhares da comunidade científica, pois a tecnologia de mapeamento 3D envolvida no conceito do funcionamento do Kinect, até seu anúncio, era extremamente cara.

Desde o seu comentado anúncio, as promessas de revolução de como funcionará a interação com jogos de vídeo-game no futuro foram tratadas com ceticismo e ansiedade pelos críticos. Isso se deve ao fato de que a tecnologia abordada no Kinect não ser necessariamente uma novidade, mas o preço baixo apresentado pela Microsoft deixou dúvidas quanto à funcionalidade do novo aparelho.

Embora a intenção inicial do Kinect fosse servir de acessório para o console de Video-game XBOX, o grande interesse da comunidade científica e acadêmica levou a Microsoft a oferecer suporte para pesquisadores, seja através da SDK, que foi lançada em junho de 2011 ou através de suporte técnico sobre o funcionamento do aparelho, através de participação de membros do projeto natal nas principais discussões do tema.

Neste trabalho de graduação, foi importante o estudo dos limites do hardware do Kinect na avaliação de suas potencialidades e confirmar sua capacidade para comportar adequadamente os softwares de apoio à reabilitação fisioterapêutica. O compreendimento do funcionamento foi obtido através da discussão em fóruns científicos especializados e a

veracidade de suas informações, confirmada pelo líder do projeto, Alex Kippman, que participa desses grupos de discussão (Vetta Labs, 2010).

2.2.1 Funcionamento do Kinect

A visualização 3D tradicional imita a forma com que animais e seres humanos utilizam a visão. Cada um dos olhos vê uma imagem ligeiramente diferente. O cérebro combina essas duas imagens e fornece a percepção 3D. Este processo é chamado de visão estereoscópica. Na visualização estereoscópica 3D são utilizadas duas câmeras para obter essas imagens diferentes, e um algoritmo calcula a posição dos objetos baseada nas suas diferenças. Essa abordagem sofre do seguinte problema: dependendo da posição do objeto é necessário mudar o ângulo entre as câmeras e o seu foco, tornando árdua a tarefa de obter um modelo 3D confiável.

Outra abordagem é baseada em sonares de golfinhos ou morcegos, ou seja, baseada em eco. Esses animais são capazes de emitir um pulso de som e medem o tempo decorrido até o som bater em algum objeto e retornar. Baseando-se nesse tempo é calculada a distância, e com isso o animal pode se localizar no ambiente. As câmeras baseadas no princípio do eco são chamadas de câmeras de tempo de vôo (*time of flight*). Para cada pixel é calculado o tempo de captura, e com esse dado é realizado o calculo da distância até o objeto, resultando na obtenção do mapa de profundidade.

Já a visualização 3D do Kinect é baseada no modelo sensor da Primesense (Primesense, 2010), utilizando o processo denominado Luz Estruturada (S*tructured Light*). Projetando-se uma matriz estruturada de pontos luminosos em uma cena, objetos mais próximos apresentarão os pontos com um tamanho maior, ou distorcidos, caso reflitam em uma superfície irregular, por exemplo. Com essa distorção e tamanho dos pontos é realizado o mapeamento de profundidade (Figura 2-6). Luz estruturada pode tomar forma de diversos padrões, como linhas paralelas, matriz de pontos, ou outras estruturas mais complexas, mas que codifiquem mais informação (Vetta Labs, 2010). São obtidos 30 quadros do mapa de profundidade por segundo. Esta taxa de atualização é considerada baixa para a detecção de movimentos muito sutis ou rápidos.

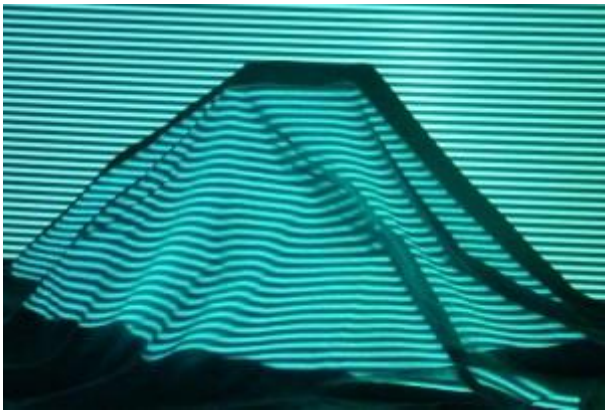

Figura 2-6 – Luz Estruturada em superfície irregular (Primesense, 2009).

O Kinect cria a matriz de pontos através de um projetor de luz infravermelha, invisível ao olho humano, mas detectável por qualquer sensor de câmera digital comum (Figura 2-7). Ao realizar a combinação da disposição da matriz de pontos na cena um algoritmo para cálculo de profundidade baseado nesses pontos gera o mapa de profundidade. Com isso, é possível a construção de uma imagem de profundidade de cena (Figura 2-8), onde cores mais claras representam objetos mais próximos do sensor.

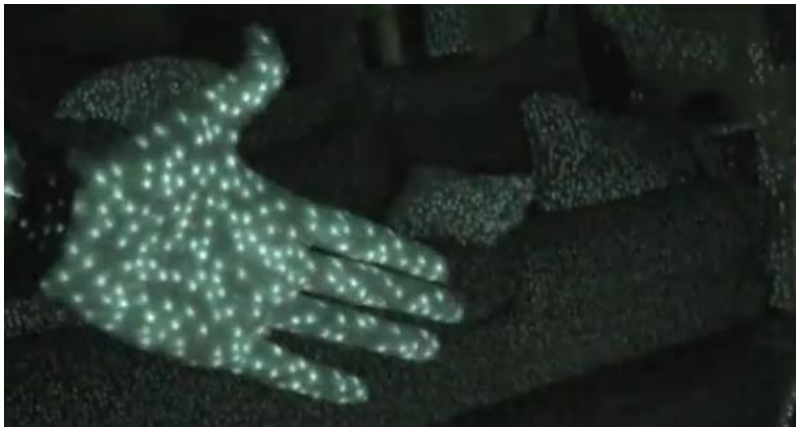

Figura 2-7 – Matriz de pontos gerada pelo projetor IR (Microsoft, 2010).

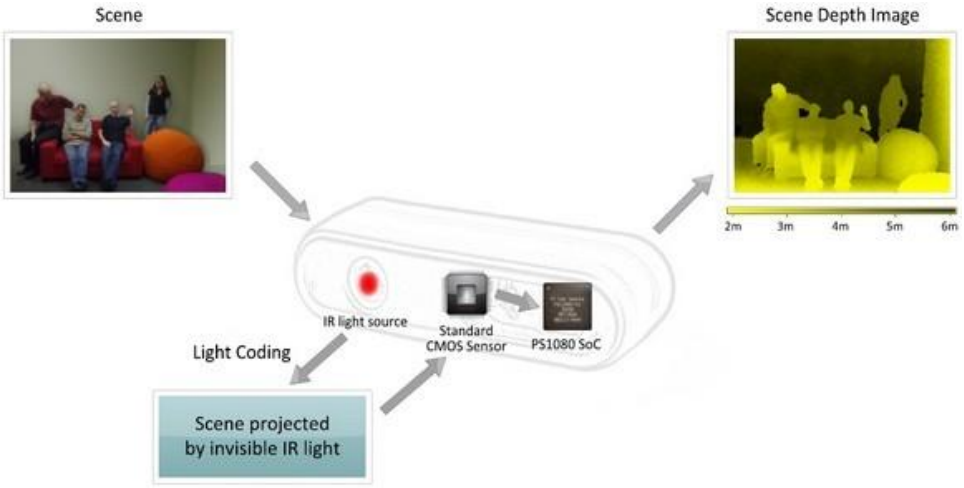

Figura 2-8 – Funcionamento do Mapeamento de profundidade (Primesense, 2009).

A vantagem do uso do conceito de luz estruturada ao invés de câmeras TOF é o custo do hardware, pois nas TOF, para calcularmos a distância de eco é necessário um contador de tempo para cada pixel, o que justifica o custo proibitivo, visto que seria necessário um hardware muito mais complexo.

O mapeamento 3D, no entanto, é apenas metade do problema. A detecção de jogadores no mapa de profundidade (*Depth Map*) foi a parte mais trabalhosa do seu desenvolvimento, pois é de extrema complexidade obter informações confiáveis sobre os esqueletos dos jogadores à partir de tantas possibilidades de estereótipos. Existem pessoas altas, baixas, gordas, magras. Para contornar esse problema, foram utilizados algoritmos de aprendizado guiados por regras arbitrárias. Um exemplo de regra seria dizer que joelhos e cotovelos não traçam um arco maior que 180º. A Microsoft comprou uma grande quantidade de dados de captura provindos de filmes com soluções alternativas (Figura 2-9) e jogos antigos para a inferência de mais regras e alimentar o algoritmo de aprendizado.

Além de duas câmeras e software sofisticado para detecção de posição do corpo em tempo real, a Microsoft também integrou ao Kinect microfones direcionais e um bem calibrado sistema de reconhecimento de voz. Esse sistema foi treinado (usando algoritmo de aprendizado de máquina, como no caso da posição do corpo) com vozes de milhares de pessoas diferentes, permitindo que ele detecte comandos simples de pessoas diferentes sem treinamento prévio para cada uma.

O verdadeiro mérito da Microsoft está justamente no esforço de criação e alimentação do processo de aprendizagem levando em consideração os vários estereótipos passíveis de detecção.

Figura 2-9 – Captura de movimentos através de sensores presos na roupa (Microsoft, 2010).

2.2.2 Histórico do desenvolvimento dos *drivers*

O Kinect, à princípio, não tinha suporte oficial à desenvolvedores, o que levou vários grupos, como o Openkinect.org (projeto libfreenect) a buscar soluções, com drivers desenvolvidos através de *hacks*, para a criação de interfaces com o dispositivo.

O processo de detecção de esqueleto na lib do Openkinect é customizável pelo programador, necessitando de uma rotina de calibração que envolve vários testes com o usuário em algumas posições pré-estabelecidas. Além disso, o projeto tem severos problemas de incompatibilidade com outras bibliotecas ou *engines* 3D, o que requer processos de configuração alternativos para o processo de desenvolvimento de aplicativos. Isso se agrava devido ao fato da documentação da *lib* ser bastante escassa, basicamente constituída através de fóruns de discussão descentralizados. O processo de detecção, no entanto, é configurável, possibilitando ao programador a criação de mais juntas que o software do Kinect reconhece intrinsecamente. Com isso, com a utilização dessa *lib*, a customização de gestos manuais ou bastante específicos fica viável, uma vez que o algoritmo de *Skeleton Tracking* do Kinect não reconhece dedos das mãos. Na Figura 2-10, é apresentado o excelente aplicativo de controle de interface, produzido por pesquisadores do MIT com a *libfreenect*, que reproduz uma interface semelhante a dos computadores do filme Minority Report (Dirigido por Steven Spielberg, 2002).

Figura 2-10 – Aplicativo que simula interface manual (OpenKinect.org, 2010).

Mais tarde a empresa PrimeSense, parceira da Microsoft no desenvolvimento do hardware do Kinect, utilizou OpenNI, que é uma biblioteca de programação de licença livre voltada à *Natural Interfaces*, para lançar uma pacote de programação para o Kinect. Até hoje, mesmo com o advento da SDK Microsofk Kinect, a lib Primesense é bem vista pela comunidade de desenvolvimento e é seguramente, a mais utilizada no desenvolvimento de projetos por programadores da comunidade acadêmica e científica.

O principal atrativo dessa biblioteca é a facilidade de programação, configuração, e integração com outras libs e engines, como ARToolkit (Figura 2-11) e Unity 3D. Algumas de suas atualizações de drivers, entretanto, não têm compatibilidade com versões desenvolvidas anteriormente

Figura 2-11 – Aplicativo de realidade aumentada com ARToolkit e OpenNI (Kinect Kamehameha, 2011).

No início de 2011 a Microsoft anunciou o lançamento de uma SDK para o desenvolvimento, com a promessa de suporte oficial às aplicações desenvolvidas. A SDK contém pacote para C#, C++ e suporte aos drivers de reconhecimento de esqueleto de até 2 jogadores. Embora o pacote ainda esteja em fase de teste, sua documentação detalhada e

suporte oficial foram uns dos atrativos para a escolha do seu uso no desenvolvimento do jogo voltado à reabilitação fisioterapêutica apresentado neste trabalho.

2.2.3 SDK Microsoft Kinect

Com o intuito de encorajar novos desenvolvedores, e alimentar o interesse da comunidade científica no Kinect, a Microsoft disponibilizou a SDK Microsoft Kinect Beta, contendo as principais funções para controle do sensor, obtenção do mapa 3D e detecção de jogadores.

O kit inclui os *drivers* para utilização do aparelho em ambiente Windows 7, as funções para comunicação com a interface do dispositivo, que são demonstradas nesse trabalho, para melhor entendimento da programação nesse ambiente. Até o final de 2011, a Microsoft apresentou pacotes somente para C# e C++.

Os requisitos mínimos de hardware, listados na documentação da SDK (Tabela 2-2) sugerem que praticamente qualquer computador atual consegue executar tranquilamente a maioria dos aplicativos feitos para o Kinect. A necessidade de uma placa de vídeo compatível com DirectX pode ser um fator que limitará seu uso para alguns usuários.

Tabela 2-2 - Requisitos de Hardware SDK Microsoft Kinect

Processador Dual core, 2.66 MHz ou melhor
Placa de vídeo compatível com Windows 7 e suporte para DirectX 9.0c
2 GB de memória RAM
Sensor Microsoft Kinect, edição clássica com conector USB

Adicionalmente, a Microsoft procurou facilitar o acesso ao ambiente de programação à maioria dos programadores acadêmicos, criando aplicativos de teste para serem executados no Windows 7, no ambiente Visual Studio 2010, facilmente acessíveis a qualquer desenvolvedor. Os requisitos para programação na SDK são listados na Tabela 2-3.

Tabela 2-3 - Requisitos para desenvolvimento de software SDK Microsoft Kinect

Visual Studio 2010 Express
Microsoft .NET framework 4.0
Microsoft Speech Platform Server (reconhecimento de microfones), v10.2
Microsoft Speech SDK, v10.2
Pacote de Linguas – Kinect para Windows, v 0.9

As funcionalidades principais do kit incluem as funções de rastreamento do esqueleto de até duas pessoas, simultaneamente, no campo de visão do sensor. Além disso, é disponibilizado o acesso ao fluxo de video de uma câmera de profundidade XYZ, com a profundidade da cena, indicando a distância de objetos até o sensor. Adicionalmente, ficam disponíveis os dados da câmera RGB regular. Para o controle do fluxo do áudio, a SDK oferece controle sobre a entrada de dados do conjunto de 4 microfones, com avançado algoritmo de cancelamento de eco e reconhecimento de padrões de voz, integrado com a plataforma de reconhecimento de voz da Microsoft.

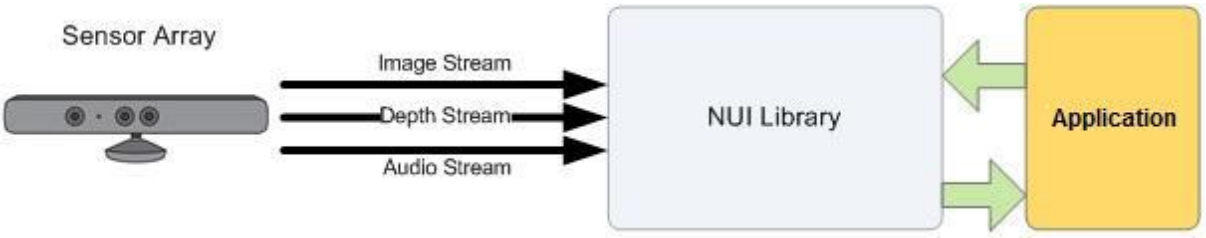

Figura 2-12 – Arquitetura de aplicações desenvolvidas com a SDK Microsoft Kinect (Microsot SDK, 2011).

O diagrama da arquitetura de aplicações para Kinect, ilustrado na Figura 2-12 mostra a importância da lib NUI, que possui as funções de obtenção de stream de imagem, de áudio e de controle do dispositivo, incluindo a possibilidade de uso de vários aparelhos Kinect simultaneamente.

Cada frame é obtido através de um modelo denominado *polling*, onde a aplicação requisita um quadro e especifica um tempo de espera máximo até o próximo quadro. Caso o próximo esteja pronto, pega-o imediatamente, senão, espera-se expirar o tempo definido. É possível, ainda especificar o tempo de espera como 0. Nesse caso, fica disponível a possibilidade da aplicação continuar trabalhando enquanto o novo quadro não fica disponível.

Se uma aplicação requer quadros mais rapidamente do que eles apresentam-se disponíveis, existe a opção de esperar pelo próximo, ou tentar de novo mais tarde.

Existe ainda outro modelo de obtenção de frames é o modelo de eventos. A aplicação passa um gatilho de eventos ao método NuiImageStreamOpen. Quando um novo quadro está pronto, quaisquer threads do processo que estejam esperando são desbloqueadas e podem usar o método NuiImageGetNextFrame, e o evento é reiniciado (Microsoft, 2011).

Para a obtenção do fluxo de dados de imagens desejado, na inicialização da NUI devemos especificar um ou mais parâmetros de identificação de tipo de dados utilizados, que pode ser *Color Data* (Imagens RGB), *Depth Data* (Profundidade da imagem) e *Player Segmentation Data* (dados da identificação de jogadores na cena).

O parâmetro *Color Data* oferece a possibilidade de reproduzir a cena em uma resolução média (com imagens comprimidas e retornadas à 1280x1024) à 30 FPS, e uma certa perda de fidelidade ou resolução alta (sem compressão), 15 FPS, e requer a alocação de buffers maiores pela NUI.

O parâmetro *Depth Data* provém a distância em milímetros até o objeto mais próximo daquela coordenada (x,y) em particular, no campo de visão do sensor de profundidade. Pode ser utilizado na identificação de objetos ou movimentos na cena, incluindo funções que ignoram alguns objetos muito próximos ou muito distantes.

O parâmetro *Segmentation Data* retorna o mapa de segmentação dos jogadores detectados no campo de visão. Ele vem em forma de um simples bitmap no qual os valores dos pixels correspondem ao índice do jogador encontrado. Se um pixel tem valor zero, nenhum jogador foi encontrado. Valores 1 e 2 identificam os jogadores.

Com as informações obtidas pelo fluxo de dados de imagem, é possivel identificar potenciais jogadores através dos dados retornados pelo parâmetro *Segmentation Data*. Baseado nessas informações, o algoritmo de detecção de esqueleto da SDK mapeia as juntas do corpo humano em 20 pontos (Figura 2-13).

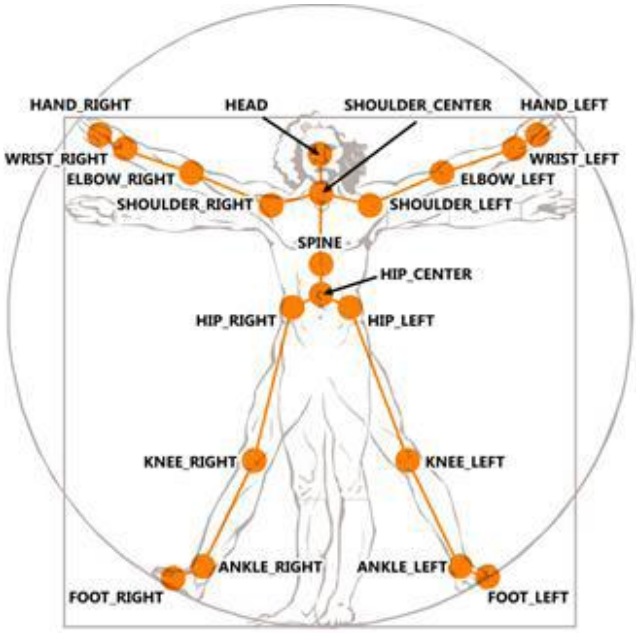

Figura 2-13– Pontos de mapeamento do esqueleto do jogador (Microsoft SDK, 2011).

O Kinect tem capacidade de rastrear ativamente 2 jogadores dentro do campo de visão (Figura 2-14), mas também consegue descobrir mais 4 jogadores, que são passivamente rastreados, ou seja, seu esqueleto contém informações limitadas sobre sua localização. As informações obtidas de um esqueleto detectado ativamente são: posição do jogador e pose.

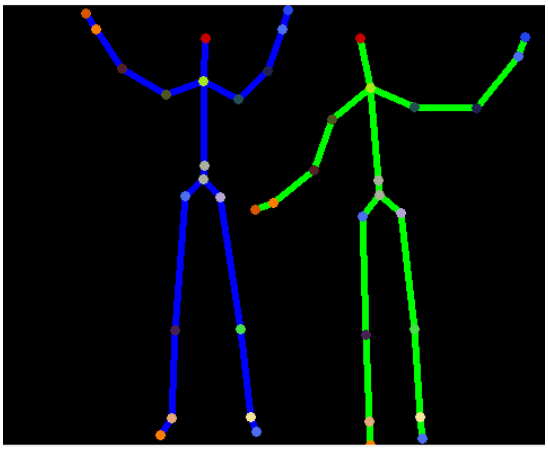

Figura 2-14 – Dois jogadores(esqueletos) ativamente rastreados (Microsoft SDK, 2011).

A função retorna os dados de cada um dos esqueletos rastreados (ativos e passivos) e a pose em um modelo de estruturas de esqueleto. Primeiramente, são diferenciados os esqueletos ativamente e passivamente rastreados por um ID atribuído aos ativos, e permanece desse jeito enquanto o jogador se move pela tela, sem sair do campo de visão do sensor. Logo após, a função retorna um vetor do tipo Vector4 que indica o centro de massa dos esqueletos

(HIP_CENTER). Esse valor é a única referência da posição de esqueletos rastreados passivamente. Para os jogadores ativos, os dados incluem posição de todos os pontos do corpo que formarão as juntas, formando a pose (Microsoft, 2011). Cálculos sobre a distância das juntas e dos movimentos dos membros do corpo, permitem configurar gestos que servirão de gatilhos para uma interface natural, seja em um jogo, ou em um método de entrada de dados para uma aplicação qualquer do computador.

No contexto da reabilitação fisioterapêutica, a obtenção dos dados do esqueleto juntamente com os dados do mapeamento de profundidade, possibilitaram a viabilização desse trabalho. Com o cálculo de angulação de movimento das articulações e juntas fica viável, também, monitorar o progresso do paciente no exercício e trazer informações confiáveis sobre a evolução do exercício ao fisioterapeuta. A dinâmica de movimento dos braços através dos pontos do esqueleto serviu para a criação de movimentos gestuais que serão os métodos de interação com o protótipo.

2.3 Unity 3D

A *engine* gráfica Unity 3D é uma ferramenta proprietária para a criação de jogos 3D ou outro tipo de conteúdo interativo como visualizações arquiteturais ou animações em tempo real. É possível produzir jogos para as mais diversas plataformas, incluindo consoles de vídeo game e browsers de internet. A Unity possibilita a edição visual de objetos de jogo (chamados *GameObjects*) que possibilita a inspeção e alteração fáceis de cada GameObject e suas propriedades. Adicionalmente, a qualquer momento, no modo edição, é possível testar o aplicativo criado, (Figura 2-15) possibilitando a alteração e visualização de variáveis sem a necessidade de geração de um arquivo executável, o que simplifica o processo de depuração.

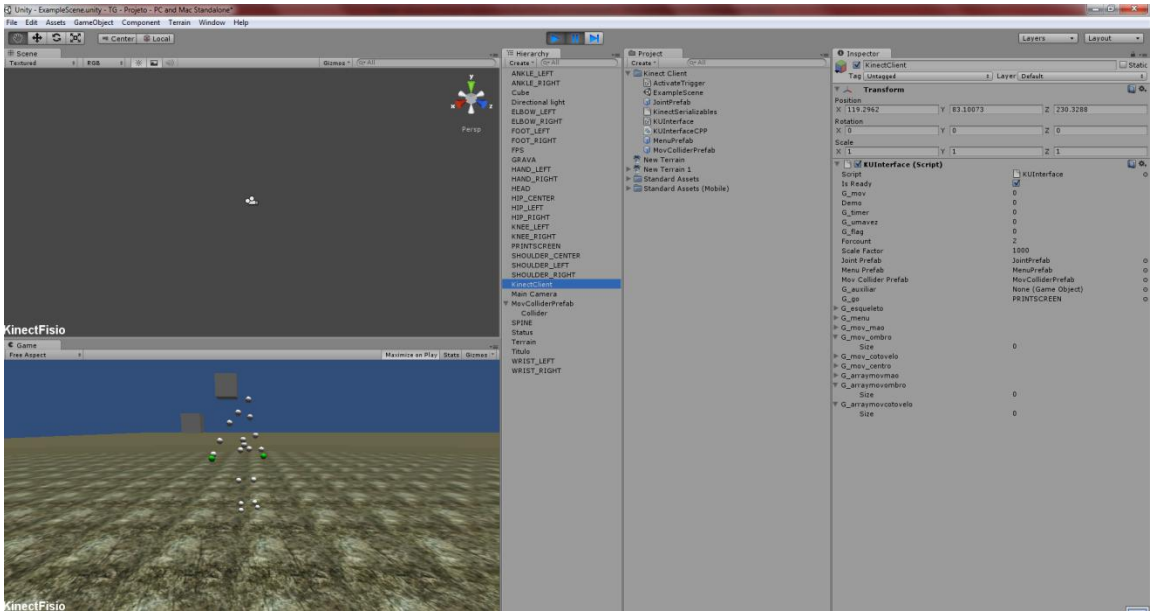

Figura 2-15 – Teste de aplicativo e alteração de variáveis em tempo de execução no ambiente Unity 3D.

A Unity é baseada em orientação a objetos. Por conseguinte os objetos no mundo de jogo podem utilizar-se das características desse paradigma. Um objeto de jogo pode ser uma instância de outro objeto já definido, o que traduz o conceito Herança. Um objeto de jogo não possui capacidade de intervir diretamente nos atributos de outro objeto, o que caracteriza Encapsulamento. Finalmente, um objeto de jogo pode ser de vários tipos, que compartilham algumas mesmas propriedades, o que caracteriza polimorfismo (Unity3D, 2009). A Figura 2-16 demonstra na interface do editor a característica de Herança, uma vez que o novo objeto criado (de nome *Collider*) é do tipo *MovColliderPrefab*, herdando assim as suas propriedades.

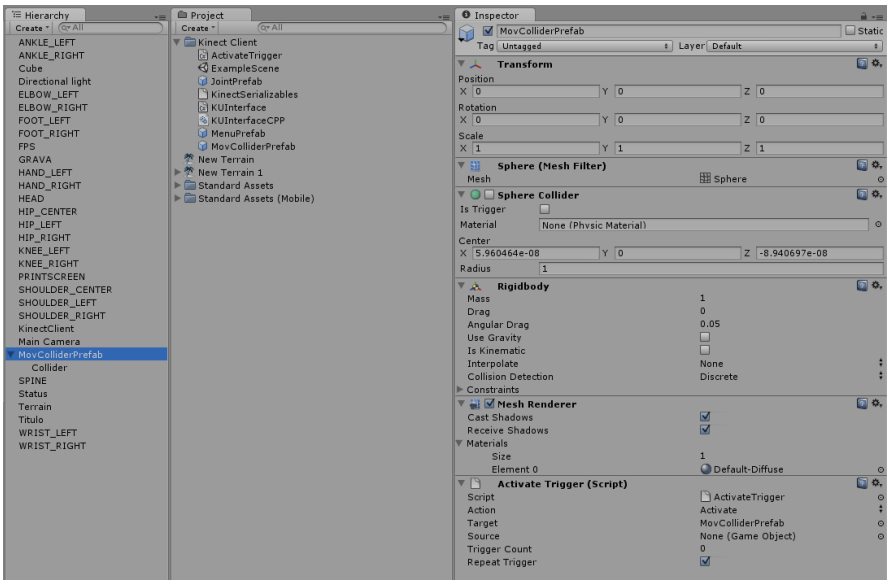

Figura 2-16 – Conceito de Herança representado na Unity3D

Como também observado na Figura 2-16, à direita o visualizador de propriedades, nesse caso, contém os parâmetros *Transform, Sphere, Sphere Collider, RigidBody, Mesh Renderer e Activate Trigger*. Cada um desses atributos representa um componente do objeto, dotando-o de propriedades como interações físicas (gravidade, massa) ou simplesmente sua forma e colisão com outros objetos. Através da configuração dessas propriedades, de modo a habilitar a interação de objetos, é possível desenvolver a jogabilidade planejada pelo designer (Unity3D, 2009) do jogo.

A maioria das propriedades de um objeto requeridas para a criação de um jogo 3D qualquer como a habilidade de um corpo quicar ao alcançar uma superfície ou simplesmente obedecer à gravidade já estão implementadas nas ferramentas da Unity. Aliando isso à prática de inserção scripts em C#, boo(pyton) e JavaScript aos objetos, apresenta ao desenvolvedor todas as ferramentas que necessita para a criação facilitada de objetos complexos, de acordo com a necessidade do jogo.

2.3.1 Componentes da Unity 3D

Cada objeto criado dentro do ambiente Unity pode ter diversas propriedades, que são utilizadas para simular características que se deseja ter nesses objetos. Essas características são apresentadas através da classe *Component*. Essa classe contém todas as referências para qualquer propriedade que esteja ligada a um *GameObject*.

Como a lista de componentes existentes na Unity é relativamente extensa, foi optado pela explicação apenas dos componentes utilizados no desenvolvimento do protótipo para reabilitação fisioterapêutica. As suas características estão listadas na Tabela 2-4.

Tabela 2-4 – Funções dos componentes dos GameObjects (Unity 3D, 2009).

Nome	**Função**
Transform	Manipulação de rotação, posição e escala
RigidBody	Controle de simulação da física do objeto
Camera	Controle da visão do mundo
Collider	Controle de colisão entre objetos
Renderer	Controle das propriedades de visualização e renderização do objeto

O componente *Transform* determina a posição atual, rotação e escala de um objeto na cena. É o componente mais importante para um jogo, uma vez que através da posição dos objetos, é possível relacionar suas interações, criando a jogabilidade. É ilustrado na figura 2-17. É possível alterar as posições do objeto através do editor, ou através de um script que altera o parâmetro *Transform.position.* Neste trabalho, cada junta é reposicionada usando o *Transform* cada vez que um novo quadro do stream de vídeo do Kinect é recebido.

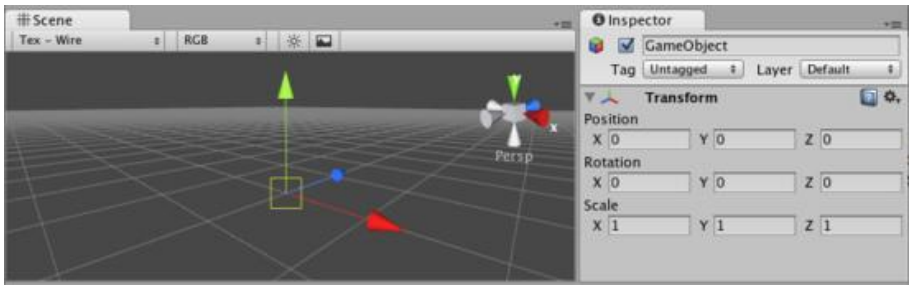

Figura 2-17– Propriedades do componente *Transform*.

O componente *Rigidbody* possibilita o objeto a agir de acordo com os parâmetros da física. Habilita-o a receber forças de objetos externos e se mover de um jeito realístico, de acordo com características como massa, gravidade, resistência ao ar e propriedades de rotação ao ser atingido. Na figura 2-18 um objeto representando um carro possui um *Rigidbody*, significando que esse carro está sujeito à física, à gravidade e interferência de outros objetos que tenham propriedades de colisão.

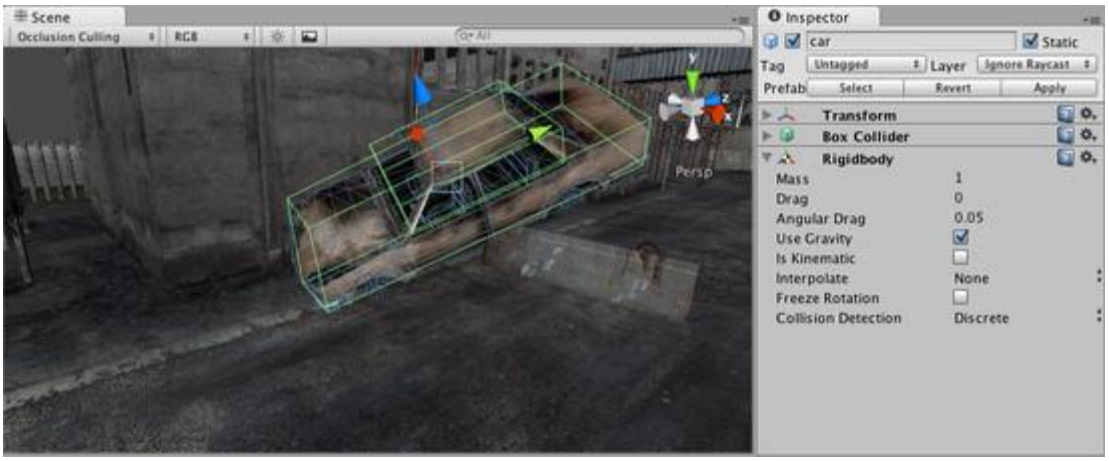

Figura 2-18 – Propriedades do componente RigidBody (Unity 3D, 2010).

O componente *Camera* (Figura 2-19) é utilizado na obtenção da visualização do mundo de jogo, através de câmeras virtuais. Manipulando a disposição dessas câmeras e seu tipo de projeção, é possível criar os mais variados efeitos, dependendo do que o jogo requer.

A posição, rotação e *lookat* (ponto para o qual a câmera está olhando) da câmera são alteradas pelo seu componente *Transform*. É possível visualizar, também, a projeção ortográfica, caso seja útil para o jogo.

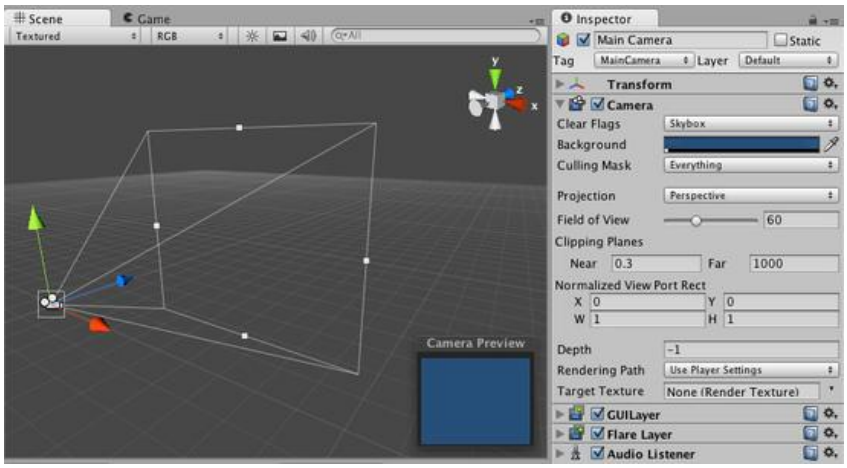

Figura 2-19 – Propriedades do componente Camera.

A disposição da câmera é um elemento importante do design do jogo, e o estudo da sua posição otimizada num mundo específico de jogo é importante para a criação de uma interface satisfatória para o usuário. Por exemplo, para um jogo de quebra-cabeças, pode ser útil manter a câmera apenas no mesmo local. Para um jogo FPS *shooter* (tiro em primeira pessoa) é interessante manter a câmera na altura dos olhos do jogador, e acompanhando seus movimentos.

O componente *Collider* é o responsável pela maioria das interações que acontecem entre os elementos de um jogo. Os *Collider* trabalham juntamente com os Rigidbody para trazer as características da física do mundo real para os objetos. Quando acontece uma colisão entre dois objetos e pelo menos um deles tem a ligação de um Rigidbody, três mensagens de colisão são enviadas a esses objetos envolvidos na colisão: OnCollisionEnter, OnCollisionStay e OnCollisionExit. Através do tratamento dessas mensagens utilizando scripts, torna-se possível a criação de comportamentos customizados para uso no jogo a ser desenvolvido. A Figura 2-20 apresenta as configurações do componente *Collider*.

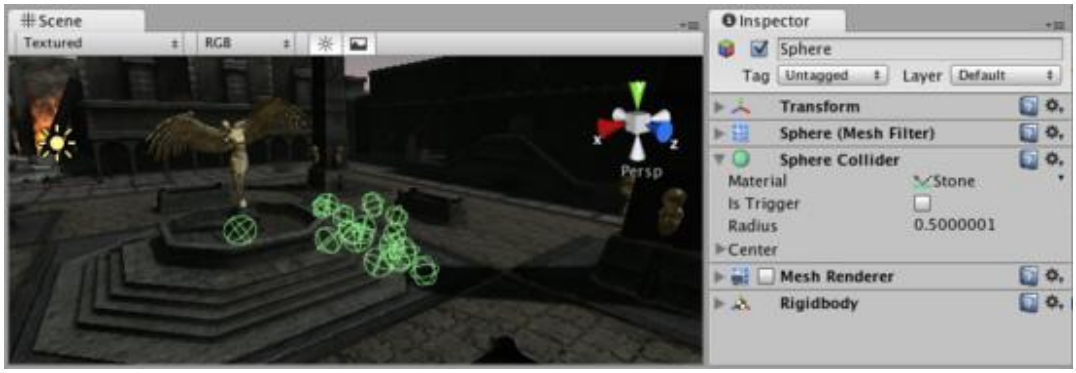

Figura 2-20 – Propriedades do componente Sphere *Collider* (Unity 3D, 2010).

O componente *Renderer* controla as propriedades de renderização do objeto, através de alterações de atributos como cor, texturas, shaders e geração de sombras no cenário. Com essas opções, fica acessível a diferenciação gráfica no jogo, através de texturas, a fim de retratar com mais fidelidade o mundo real.

2.3.2 Unity 3D e Kinect

A integração entre Kinect SDK e Unity foi alcançada através de um plugin criado por Andrew Devine, da University of Central Florida. Com isso, é possível acessar as funções da SDK Microsoft Kinect através de um script C# que é associado à um GameObject da Unity3D. A sintaxe dos métodos é a mesma da SDK, e a rotina para obtenção dos dados do esqueleto do jogador para coleta da informação das juntas foi simplificada, sendo organizada em uma estrutura Enum (array de strings cujas posições podem ser acessadas, também, através de integers), para facilitar a manipulação das posições das juntas. Sem esse plugin, o desenvolvimento desse trabalho necessitaria do uso de um servidor TCP, que buscaria os quadros de informação advindos do Kinect, e os transferiria à um script da Unity que realizaria a leitura através da desserialização, o que teria um desempenho muito inferior e geraria um certo atraso em tempo de execução do protótipo, pois o servidor perde alguns milissegundos com o processo de serialização e desserialização dos fluxos de vídeo fornecidos pelo Kinect.

3 IMPLEMENTAÇÃO DO PROTÓTIPO

Este capítulo versa sobre todo o processo de desenvolvimento do protótipo, passando pela coleta de requisitos, consolidação da ideia para a solução e detalhes sobre a implementação.

3.1 Definição dos requisitos do protótipo

Para definição dos requisitos do aplicativo desenvolvido, foram necessárias algumas visitas à ala de fisioterapia no Hospital Universitário da UFSM, incluindo acompanhamento de sessões de fisioterapia em pacientes com lesões no ombro, ou no braço. Adicionalmente, foi feito um estudo sobre os principais movimentos realizados como parte dos exercícios de reabilitação dessas partes do corpo, a fim de identificar padrões desses movimentos. Com essas informações, foi possível definir os princípios que passaram a reger a maneira de como os exercícios são detectados no protótipo.

Os movimentos específicos envolvidos na reabilitação fisioterapêutica devem respeitar regras, de acordo com o objetivo a ser alcançado nessa prática. Como exemplo, na Figura 3-1 é mostrado o exercício de rotação externa, devendo ser contabilizada não somente a posição do braço, mas o fato de que os cotovelos devem manter-se junto ao corpo.

Figura 3-1 – Execução do exercício de rotação externa (MATSUO, Tatiana, 2010).

Chegou-se à conclusão que para a detecção correta do movimento, era necessário incluir na lógica do programa as informações sobre posição dos cotovelos, mãos, e também ombros em um instante de tempo t de captura da imagem. Além disso, constatou-se que

alguns pacientes do HUSM não obtêm sucesso na conclusão total do movimento, por causa de dor, ou incapacidade causada por sua lesão. Deve ser possível, então, a captura de um movimento incompleto. À partir dessas conclusões, foram elaborados os requisitos do protótipo:

- O sistema deve ser capaz de avaliar os movimentos à partir das mãos, cotovelos e ombros simultaneamente, em um determinado instante de tempo t;
- O sistema deve ser capaz de detectar movimentos incompletos através da medida da amplitude do movimento realizado, comparada com a medida da amplitude do movimento completo;
- O sistema deve disponibilizar o progresso do usuário no movimento para comparação com outras sessões de uso.

3.2 Desenvolvimento do Protótipo

Com base nos requisitos, foram estudadas soluções para a detecção completa ou incompleta do movimento. Chegou-se à conclusão de que seria útil para o projeto utilizar-se do sistema de coordenadas da Unity 3D, uma vez que o personagem que representaria o jogador no mundo de jogo estaria situado em algum ponto do espaço da cena. Calculando as mecânicas de movimento das partes do corpo usando as coordenadas do componente *Transform* de cada junta em um determinado instante de tempo, e verificando a correspondência de suas posições requeridas na execução do movimento é validada a detecção.

3.2.1 Disposição do esqueleto na cena

Primeiramente foi necessário inserir o esqueleto do jogador no mundo de jogo. O primeiro passo tomado, foi criar um componente *Camera* (MainCamera), com coordenadas de visualização mostradas no componente *Transform* contido na Figura 3-2. As coordenadas da câmera foram utilizadas no posicionamento das juntas e demais elementos da interface, de forma a tornar visível o esqueleto ao usuário.

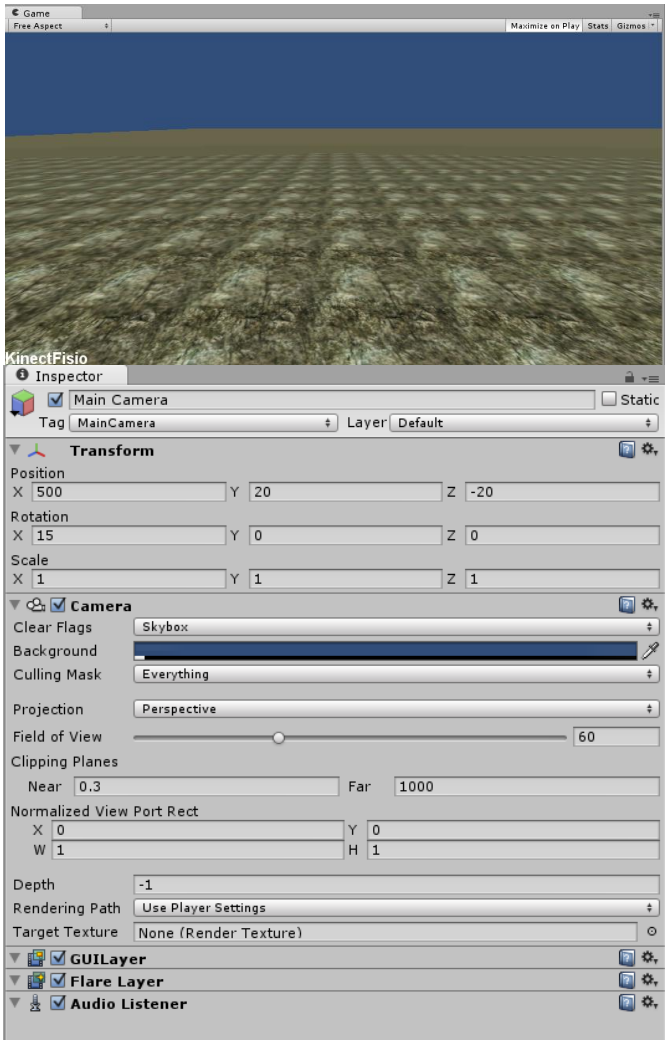

Figura 3-2– Componente *Camera* e o seu campo de visualização.

O tratamento das imagens e detecção do esqueleto foi feito através de um *GameObject* ligado à um componente script (KUIinterface.cs) que comporta as funções utilizadas na interface com o Kinect e processa a validação do movimento.

Para capturar a posição de uma junta no campo de visão do Kinect, foi criada a função GetJointPos(), que recebe como parâmetro o nome, ou número inteiro equivalente à junta requisitada. Esta função trabalha através de uma chamada à função Nui.skeleton.transform, os algoritmos de *Skeleton Tracking* do Kinect realizam a detecção automaticamente e retornam suas posições. A função transforma em milímetros as coordenadas X, Y e Z da junta rastreada em um quadro de vídeo do Kinect e oferece retorno através de um Vector3. Esse vetor equivale à posição do esqueleto do jogador, relativa à posição do aparelho Kinect, que serve como a origem do sistema de coordenadas (ponto 0f,0f,0f). Como o Kinect consegue capturar até 30 quadros de vídeo por segundo, essa função é chamada 30 vezes por segundo para cada

nova atualização. O retorno da função será utilizado para o posicionamento dos *GameObjects* que serão criados e correspondem às partes do corpo.

Em seguida, foram criados os *GameObjects* necessários, cada um equivalendo a uma junta e recebendo a mesma denominação que a SDK as atribui. Esses objetos são criados na posição da origem, para posterior reposicionamento no campo de visão da câmera.

O trecho de código da função que cria os *GameObjects* é mostrado na Figura 3-3.

```
for (auxjunta= KinectWrapper.Joints.HIP_CENTER,
forcount=0;auxjunta<KinectWrapper.Joints.COUNT;forcount++,auxjunta++)
{
        g_go = (GameObject)Instantiate(JointPrefab,junta,Quaternion.identity);
        g_go.name = Enum.GetName(typeof(KinectWrapper.Joints), auxjunta);
        g_esqueleto[forcount]=g_go;
}
```

Figura 3-3– Criação dos GameObjects equivalentes a cada junta do corpo.

Uma variável auxiliar g_go do tipo *GameObject* recebe uma instância do tipo *JointPreFab*, que é um objeto criado para representar cada junta do corpo. O tipo *JointPreFab* possui um componente *Renderer* de formato esférico e um *Collider*, também esférico, que serão utilizados posteriormente na detecção do movimento e interface com programa. Essas instâncias de JointPreFab são guardadas no vetor g_esqueleto[].

Posteriormente, com os resultados da chamada da função GetJointPos() de cada junta, é aplicada uma Transform.position pré-definida em todos os pontos de g_esqueleto[] com as coordenadas do centro do campo de visão da câmera. Essencialmente, isso significa transferir os objetos do *array* g_esqueleto[] para a área visível ao usuário, ou "área filmada" pelo objeto MainCamera.

Com isso, foi efetivamente criado e posicionado o "personagem" no mundo de jogo, que é uma representação da captura dos movimentos do esqueleto do usuário pelo Kinect, a 30 FPS (Figura 3-4).

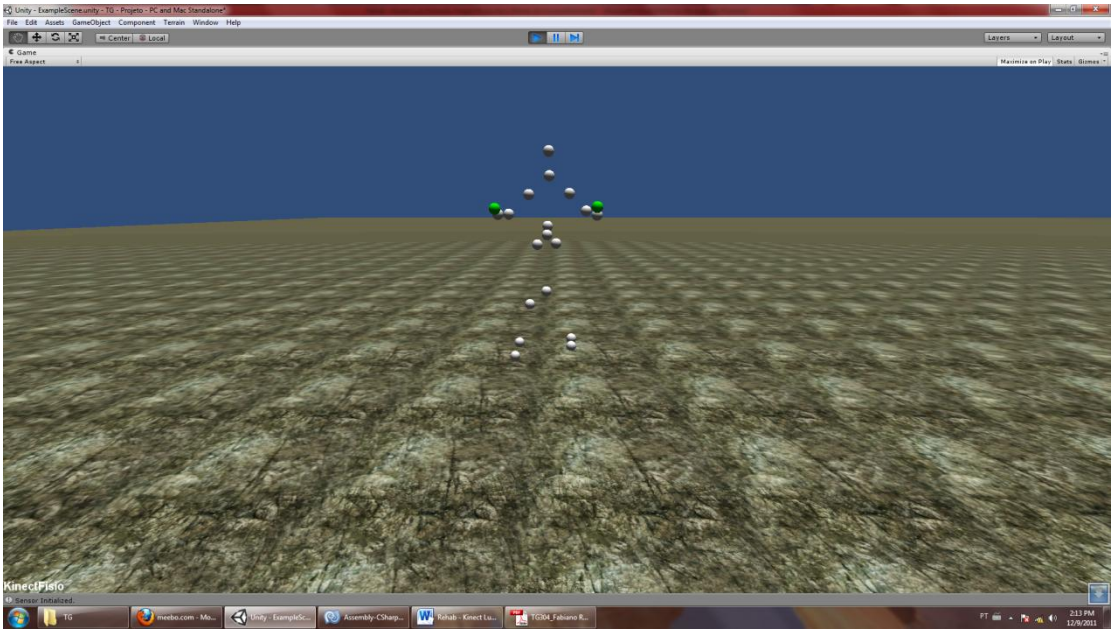

Figura 3-4 – Representação do esqueleto do usuário no cenário criado.

3.2.2 Detecção de movimentos específicos

Partindo do princípio que um exercício específico do braço envolve o posicionamento sequencial das juntas (Cotovelos, ombros e mãos) em pontos pré-definidos, a ideia utilizada para a verificação da validade do movimento envolve a gravação do gesto a ser executado pelo paciente. Trinta posições por segundo são guardadas em 3 *arrays* de coordenadas do tipo Vector3, mov_mao, mov_ombro e mov_cotovelo cada um equivalendo ao posicionamento de uma das juntas ligadas à movimentação do braço (HAND_RIGHT, SHOULDER_RIGHT e ELBOW_RIGHT). Adicionalmente, as coordenadas do centro de massa do esqueleto (HIP_CENTER) também são salvas. Com isso o movimento do exercício é efetivamente gravado e facilmente pode ser acessado, quando for necessário.

Para cada ponto dos *arrays gravados* é criado um objeto do tipo *MovColliderPrefab*, que agirá como um detector de movimento, portanto possuindo componentes *Collider*, inicialmente desativado. Esses objetos são guardados um *array* para cada junta envolvida no exercício. É aplicada uma Transform.position em todos, de modo à posicioná-los para a obtenção da representação de uma junta em um instante do movimento. O posicionamento é relacionado ao posicionamento do centro de massa do usuário (HIP_CENTER), ajustando-se caso o usuário se desloque pelo campo de visão do Kinect. Com isso, o usuário pode mover-se lateralmente, ou posicionar-se em qualquer distancia dentro do campo de visão do aparelho

sem que o processo de detecção de movimentos do exercício seja comprometido. Esse trecho de código é visualizado na Figura 3-5.

```
for (forcount=0;forcount<g_mov;forcount++) //g_mov equivale ao número de quadros de movimento gravados pelo Kinect
{
        junta = Vector3.zero;
        //arrays do tipo MovColliderPrefab, com pontos posicionados no local exato onde o
        //movimento foi capturado
        go_arraymovmao[forcount].transform.position=g_mov_mao[forcount];
        go_arraymovombro[forcount].transform.position=g_mov_mao[forcount];
        go_arraymovcotovelo[forcount].transform.position=g_mov_cotovelo[forcount];
        //Subtrai a posição atual do esqueleto da posição antiga,
        //atualizando assim o posicionamento dos pontos dos arrays de MovColliderPrefab
        junta.z=g_esqueleto[HIP_CENTER].transform.position.z-g_mov_centro[forcount].z;
        junta.x=g_esqueleto[HIP_CENTER].transform.position.x-g_mov_centro[forcount].x;
        junta.y=g_esqueleto[HIP_CENTER].transform.position.y-g_mov_centro[forcount].y;
        go_arraymovmao[forcount].transform.position+=junta;
        go_arraymovcotovelo[forcount].transform.position+=junta;
        go_arraymovombro[forcount].transform.position+=junta;
}
```

Figura 3-5 – Posicionamento do *array* de objetos do tipo *MovColliderPrefab*

As propriedades do tipo *MovColliderPrefab* foram definidas de forma que caso ocorra uma colisão com uma JointPrefab (tipo de objeto equivalente às juntas do corpo), é chamada uma função que passa a permitir colisão com o próximo objeto do *array* de *MovColliderPrefab*. Exemplificando com um movimento que leve em consideração apenas as mãos do jogador, caso uma junta equivalente à HAND_RIGHT (mão direita do jogador) sofra uma colisão com o primeiro elemento do *array* equivalente na figura 3-5 (go_arraymovmao[0]), é chamada a função OnCollisionEnter(), que por sua vez ativa o componente *Collider* do próximo elemento do (g_arraymovmao[1]).

Portanto, para a verificação do movimento, simplesmente o usuário deve mover-se de modo a atingir em sequência, com as juntas do corpo no cenário de jogo, cada um dos objetos contidos em um array de movimento gravado, começando do elemento 0 e levando em conta

todas as juntas envolvidas (no caso do protótipo, mãos, cotovelos e ombros) em um instante. Ocorre a completa detecção quando os últimos pontos de todos os *arrays gravados* enviarem uma mensagem de colisão com as juntas equivalentes simultaneamente.

Com isso, torna-se possível não só a verificação do movimento completo, mas, impondo um limite de tempo para conclusão do gesto, pode-se identificar quantas partes do exercício o usuário não conseguiu satisfazer, possibilitando assim a medição de seu progresso.

Ademais, caso deseje-se gravar movimentos mais complexos, que envolvam mais partes do corpo, basta adicionar a gravação dessas juntas na função de gravação e posicionamento.

A visualização do desempenho do usuário em um movimento foi feita através de um mostrador de porcentagem, marcando de 0 a 100, dependendo de qual o último elemento dos *arrays* de *MovColliderPrefab* que enviou sua mensagem de colisão. As esferas vermelhas da Figura 3-6 representam os objetos desses *arrays* que colidiram com as juntas do braço envolvidas no movimento. Os cubos cinza da Figura 3-6 representam as opções de interface utilizadas para iniciar e parar a gravação do movimento. Embora os movimentos do ombro sejam gravados, e computados quando se realiza a verificação do movimento, o array de MovColliderPrefab relativo ao ombro foi marcado como invisível, para fins de melhoria da visualização da interface.

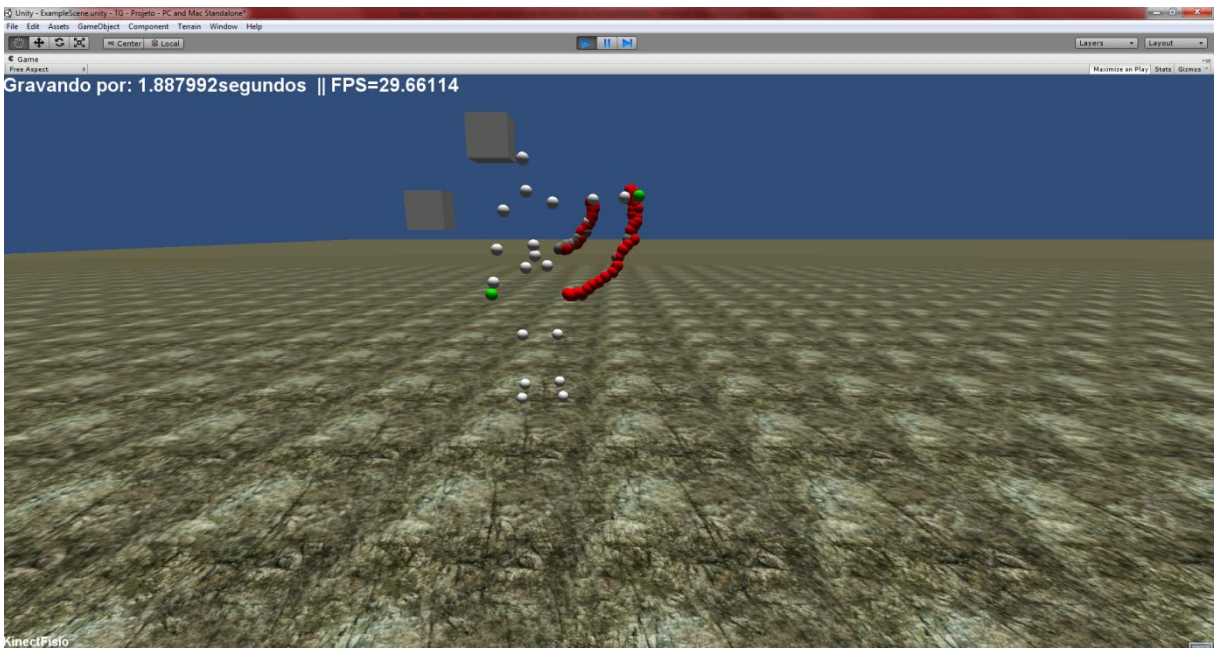

Figura 3-6 – Avaliação do movimento de abdução.

4 CONCLUSÕES E TRABALHOS FUTUROS

Neste trabalho, pode-se entender que o objetivo da computação é servir às outras áreas de conhecimento, seja na resolução de problemas, bem como na simplificação de processos ou melhora da qualidade do trabalho. Nas visitas à ala de fisioterapia do HUSM, e conversações travadas com os envolvidos, pôde-se identificar diversos pontos que poderiam ser melhorados com o auxílio das tecnologias modernas.

Dentre as novidades tecnológicas disponíveis para a realização de pesquisas, foi escolhido o Microsoft Kinect, com sua capacidade de obtenção do esqueleto de usuários à baixo custo, e simplicidade na sua interface de programação através da SDK Microsoft Kinect. O rastreamento de esqueletos abre uma gama de possibilidades na área de entretenimento e área da saúde.

Um dos objetivos alcançados pelo trabalho foi a confirmação da viabilidade do Microsoft Kinect na criação de aplicativos voltados à área de Fisioterapia. Isso foi alcançado com a implementação de um protótipo para detecção de movimentos específicos no ambiente Unity 3D.

O protótipo desenvolvido é capaz de gravar movimentos dos braços, calibrados pelo usuário, podendo ser utilizado para sessões de fisioterapia para auxiliar no tratamento de pacientes através da validação de um movimento executado, em comparação com o movimento gravado. Diferente de tratamentos convencionais, esse sistema proporciona interatividade e diversão ao pacientes, fazendo com que a etapa de reabilitação torne-se menos tortuosa e mais eficaz.

A detecção do movimento depende da inserção das informações do Kinect no ambiente Unity3D. Através da obtenção de um *array* de posições das partes do corpo, chamado *array* de movimento, foi possível criar um conjunto de componentes *Collider*, que serviram de controle para a execução do movimento. O movimento poderia ser medido, então, através de verificação de quantas juntas envolvidas no movimento colidiam sequencialmente com os objetos criados pelo *array* de movimento.

Com essa solução, tornou-se possível não só a detecção de um tipo de exercício pré-definido, mas de qualquer gesto ou movimento que envolva os pontos do esqueleto detectáveis pelo sistema de *Skeleton Tracking* do Kinect. Espera-se que essa ideia seja utilizada em jogos que envolvam detecção de diversos tipos de exercícios simultaneamente e que contribua para a pesquisa na área de reabilitação fisioterapêutica.

Pretende-se dar continuidade para o trabalho através da criação de um pacote para Unity3D e Kinect contendo as classes necessárias para criação e detecção de gestos, criação de menus acionados por movimentos específicos. Com este pacote ficará simples a criação dinâmica de gestos, que poderão ser utilizados como *input* para jogos e aplicativos de controle do computador.

Embora a detecção de movimentos apresentada neste trabalho tenha o intuito de servir a propósitos relacionados à reabilitação fisioterapêutica, a solução encontrada foi base para o surgimento de outras ideias, por exemplo, a facilitação da criação de interfaces orientadas à gestos.

5 REFERÊNCIAS

Clinica Deckers. **Lesão no Mnguito Rotador.** Disponível em: <http://www.clinicadeckers.com.br/lesão-no-manguito-rotador> Acesso em: 22 Out. 2011.

DEVINE, Andrew. **Kinect SDK - Unity 3D Interface Plugin.** Disponível em: <http://www.eecs.ucf.edu/isuelab/unity.php> Acesso em: 17 Out. 2011.

GEURTS, Luc; VANDEN, Vero; HUSSON, Jeele; WYNDEI, Frederik. **Digital Games for Physical Therapy**: Fulfilling the Need for Calibration. ACM (2010) 117-124.

JEREMY, C. **Unity3d and Microsoft Kinec**t. Disponível em: <http://www.seethroughskin.com/blog/?p=1159> Acesso em: 03 Out. 2011

MATSUO, Tatiana. **Dor no Ombro e o Manguito-Rotador**. Disponível em: <http://ctctatipilates.wordpress.com/2008/11/02/dor-no-ombro-e-o-manguito-rotador/>Acesso em: 03 Out. 2011.

MAURO, Alessandro de. **Virtual reality based rehabilitation and game technology.** Disponível em: <http://www.cs.swansea.ac.uk/EICS4Med/images/papers/eics4med%2009.pdf> Acesso em: 06 Out. 2011.

Microsoft. **Microsoft Kinect for Windows.** Disponível em: <http://www.kinectforwindows.org> Acesso em: 15 Set. 2011.

OpenKinect. **OpenKinect project**. Disponível em: < http://openkinect.org/wiki/Main_Page> Acesso em: 29 Out. 2011.

Primesense. **Primesense - Natural Interaction**. Disponível em: <http://primesense.com> Acesso em: 29 Out.2011.

Primesense. **Primesense NITE | Kinect Hacks**. Disponível em: < http://kinect.dashhacks.com/primesense-nite> Acesso em: 29 Out. 2011.

QUEIROS, Murilo; KIPPMAN, Alex. **Um cientista explica o Microsoft Kinect**. Disponível em: <http://blog.vettalabs.com/2010/11/02/um-cientista-explica-o-microsoft-kinect-parte-ii/> Acesso em: 25 Out. de 2011

JointHealing**. Rotator Cuff Tear**. Disponível em: < http://www.jointhealing.com/pages/shoulder/rotatorcuff.html> Acesso em: 16 Out. de 2011.

SCHÖNAUER, Christian; PINTARIC, Thomas; KAUFFMAN Hannes. **Full Body Interaction for Serious Games in Motor Rehabilitation.** ACM (2011) 01-08.

Unity3D. **Unity: Documentation**. Disponível em: < http://unity3d.com/support/documentation/> Acesso em: 18 Out. 2011.

WACHS, Juan Pablo; KÖLSCH, Mathias; STERN, Helman; EDAN, Yael. Vision-based hand-gesture applications. **Communication of the acm,** vol. 54, nº. 2, Fevereiro de 2011. ACM (2011) 60 - 71.

WASHIO, Tomoto. **Kinect Kamehameha.** Disponível em: <http://code.google.com/p/kinect-kamehameha/> Acesso em: 22 Out. 2011.