UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM *FRAMEWORK* PARA APOIO À CAPTURA DE MOVIMENTOS DAS MÃOS COMO FORMA DE INTERAÇÃO

PAULO EDUARDO CARDOSO ANDRADE

ORIENTADOR: PROF. DR. DELANO MEDEIROS BEDER

São Carlos - SP
Janeiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

CENTRO DE CIÊNCIAS EXATAS E DE TECNOLOGIA

PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

UM *FRAMEWORK* PARA APOIO À CAPTURA DE MOVIMENTOS DAS MÃOS COMO FORMA DE INTERAÇÃO

PAULO EDUARDO CARDOSO ANDRADE

Dissertação apresentada ao Programa de Pós-Graduação em Ciência da Computação da Universidade Federal de São Carlos, como parte dos requisitos para a obtenção do título de Mestre em Ciência da Computação, área de concentração: Engenharia de Software
Orientador: Prof. Dr. Delano Medeiros Beder.

São Carlos - SP
Janeiro/2017

UNIVERSIDADE FEDERAL DE SÃO CARLOS

Centro de Ciências Exatas e de Tecnologia

Programa de Pós-Graduação em Ciência da Computação

Folha de Aprovação

Assinaturas dos membros da comissão examinadora que avaliou e aprovou a defesa de Dissertação de Mestrado do candidato Paulo Eduardo Cardoso Andrade, realizada em 17/02/2017.

Prof. Dr. Delano Medeiros Beder
(UFSCar)

Prof. Dr. Daniel Lucrédio
(UFSCar)

****
Prof. Dr. Marcelo Morandini
(USP)

Certifico que a sessão de defesa foi realizada com a participação à distância do membro Marcelo Morandini, depois das arguições e deliberações realizadas, o participante à distância está de acordo com o conteúdo do parecer da comissão examinadora redigido no relatório de defesa do aluno Paulo Eduardo Cardoso Andrade.

Prof. Dr. Delano Medeiros Beder
Presidente da Comissão Examinadora
(UFSCar)

Dedico este estudo à minha família e amigos, por todos os sacrifícios que fizeram e ajuda que ofereceram para que eu alcançasse os meus objetivos e apoiando minhas decisões, destacando a minha linda que ao meu lado demonstrou em todos os momentos que valia a pena o esforço e que estaria junto comigo para qualquer desafio.

AGRADECIMENTO

Agradeço a todos que acompanharam minha jornada e que de alguma forma direta ou indireta ajudaram a conquista dos objetivos dessa longa caminhada, ao meu orientador e Prof. DR. Delano Medeiros Beder, um agradecimento especial, pois ele enxergou em mim durante a entrevista de alguns minutos, o potencial para desenvolver ao seu lado um trabalho, e durante todo o caminho do mestrado sempre me aconselhando de forma serena e clara, demonstrou verdadeiramente o que é ser um professor, certamente sem ele este trabalho seria impossível.

Largue esse vídeo game, e vá estudar!

Minha Mamuska

RESUMO

Motion capture (Mocap) é um conceito baseado em capturar o movimento de um objeto ou pessoa do mundo real, com a finalidade de reproduzir tais movimentos em personagens, ambientes virtuais ou prover interações com sistemas e aplicativos. Para cada movimento existem várias técnicas capazes de realizar a captura do movimento de forma mais eficiente, pois apresentam diferenças drásticas para capturar o movimento da mão, movimento da cabeça, sendo os pontos chave para a captura de movimentos completamente diferentes de acordo com cada situação. Atualmente existem diversos dispositivos que oferecem diferentes recursos para que os desenvolvedores utilizem em seus aplicativos e sistemas de captura de movimentos, porém, ainda necessitam de um conhecimento específico em captura de movimento, quando comparados a uma interação através de um dispositivo mais tradicional tal como o mouse, sendo este usado como principal meio de interação quando o assunto é interação em aplicações simples, pois o mesmo provém recursos adequados e de fácil desenvolvimento para tais aplicações. Este projeto de mestrado tem como objetivo desenvolver um *framework* para reconhecimento do movimento das mãos para uso em aplicações simples, focados nas ações de pegar e girar, utilizando o dispositivo Kinect v2.0 da Microsoft, uma abordagem de alto nível para desenvolvedores iniciantes, utilizando a captura de movimentos de maneira simples e otimizada, sendo possível comparar sua dificuldade e recursos com o dispositivo mais usado tradicionalmente para este tipo de interação - o mouse -. Foi desenvolvido um estudo de caso utilizando três diferentes abordagens: SDK disponibilizada pela Microsoft; o *framework* proposto - *EasyMoCapHand* - e a interação através do mouse, deste modo exemplificando aspectos relevantes do desenvolvimento de cada abordagem. Posteriormente, um experimento cujo os participantes desenvolvam atividades que, através das mesmas, tenham uma percepção do uso de cada abordagem, contribuindo dessa forma para o uso da captura de movimentos, em aplicações simples que tem seu desenvolvimento sempre atrelado ao uso do mouse como meio de interação, tornando esta forma de interação uma opção para desenvolvedores inexperientes que não utilizariam em sua aplicação a interação através da captura de movimentos.

Palavras-chave: Kinect, Captura de movimentos, Movimento da Mão, *Framework*, Pegar, Girar.

ABSTRACT

Motion capture (Mocap) is a concept based on capturing the motion of an object or person from the real world, in order to reproduce such motion on characters in virtual environment or provide interaction with systems or applications. For each motion, there are several techniques able to capture the motion in a more efficient way, because they present drastic differences to capture the hand motion, head motion, being the key points to capture the motion completely different according to each situation. Currently, there are various devices that offer different sources so that the developers can use on their applications, and systems of motion capture, however they still need specific knowledge in motion capture when compared to an interaction through a more traditional device such as the mouse, being this one used as main way of interaction when it is related to interaction in simple applications, because it provides suitable and easy development resources for such applications. This mater's project aims to develop a framework to recognize the hand motion to be used in simple applications, focused on the actions of catching and turning, using the device Kinect v2.0 of Microsoft and a high-level approach for beginner developers, using the motion capture in a simple and optimized way, being possible to compare their difficulty and the resources to the most used traditionally device for this kind of interaction – the mouse. A case study will be developed using three different approaches: SDK available by Microsoft; the proposed framework – EasyMoCapHand – and the interaction through the mouse; exemplifying, this way, relevant aspects of the development of each approach. And later on, an experiment which participants develop activities that and through them have a perception of the use of each approach, contributing to the use of motion capture in simple applications that have their development always attached to the mouse use as a way of interaction, becoming this way of interaction an option for beginner developers who would not use on their interaction through motion capture.

Keywords: Kinect, Motion Capture, Hand Motion, Framework, To Catch, To Turn.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

3D – *3 dimensions*

CCD – *Charge-Coupled Device*

CGI – *Computer-Generated Imagery*

LED – *Light Emitter Diode*

MOCAP – *Motion capture*

SDK- *Software Development Kit*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

1.1 Contexto

Captura de movimento é o processo de gravar um evento de movimento real e reproduzi-lo em termos matemáticos úteis, buscando pontos chave no espaço. Através da combinação desses pontos, tem-se uma simples representação do movimento capturado (MENACHE, 2011).

Sturman (1994) e Menache (2011) resumem a captura de movimento em uma tecnologia que permite o processo de conversão de um movimento ou uma performance real em uma performance digital para uma análise posterior ou imediata e possível reprodução.

A informação capturada pode ser aplicada em qualquer situação, desde uma simples posição do corpo em um espaço, ou uma situação mais complexa, como deformações de massas musculares (STURMAN, 1994).

A indústria de captura de movimentos vem crescendo de forma acelerada, influenciada pela grande procura por diversos setores e indústrias de ramos como entretenimento, medicina, desempenho esportivo e segurança (FLAM, 2009).

Uma classificação para captura de movimentos apresentada por Moeslund et al. (2006) indica a definição superficial em três grandes grupos: Vigilância (*Surveillance)*, Controle (*Control)* e Análise (*Analysis*).

Vigilância seriam as aplicações focadas em examinar movimentos, seja de uma pessoa comum, objeto, grupos de objetos ou aglomeração de pessoas. Determinar o fluxo de pessoas em uma galeria para descobrir um padrão a fim de

otimizar a localização das lojas é um exemplo. Outro exemplo seria criar um modelo que descreva o comportamento de pessoas em uma prisão. Caso alguma anomalia seja detectada, ações preventivas serão tomadas (FLAM, 2009).

A categoria Análise é geralmente dedicada ao estudo de uma pessoa ou de um conjunto de objetos de forma a ter o movimento capturado, e através dos dados capturados, é realizada uma análise e a partir daí resultados são gerados. Como um exemplo temos o aprimoramento de técnicas de atletas, onde podem ser encontradas falhas e virtudes, que devem ser corrigidos ou melhorados respectivamente.

Já a categoria Controle, se encaixam as aplicações que capturam o movimento de um ator e o transformam em uma sequência de operações ou dados digitais reutilizados. De acordo com Flam (2009), filmes como a trilogia Lord of the Rings (2001 - 2003) e Beowulf (2007) utilizam da captura de movimento para gerar animações de seus personagens. Animações dessa natureza também são encontradas em vários jogos eletrônicos, principalmente em jogos como os de futebol, futebol americano e basquete, que capturam os movimentos dos principais jogadores e os reproduzem nos seus respectivos personagens, tornando os jogos - que apresentem os movimentos capturados - virtuais mais fiéis à realidade tornando maior a imersão dos jogadores no jogo.

De acordo com Sato & Cohen (2010) e Bedoya & Guerrero (2014), captura de movimento geralmente envolve o rastreamento de pontos chave do corpo humano para se obter coordenadas que possam ser transmitidas na forma de dados utilizáveis.

Diversos sistemas de captura são desenvolvidos na atualidade e amplamente utilizados em diversas áreas. Nos últimos anos, técnicas de captura de movimento têm sido desenvolvidas em todo o mundo, incluindo novos sistemas com maiores possibilidades. Com o advento do Kinect (PEDERSOLI et al. , 2012), dispositivo desenvolvido pela Microsoft, a área de captura de movimento tem sido beneficiada, pois se trata de um dispositivo que visa reconhecer o movimento humano e prover interação com jogos e aplicações virtuais.

Nos dias atuais, a técnica de captura de movimento tem se mostrado uma forte opção quanto a forma de interação com jogos eletrônicos, aplicações e seu uso para animações de objetos virtuais por gerar resultados suaves, tornando a animação e seus movimentos mais próximos do mundo real.

1.2 Objetivos da pesquisa

O objetivo deste trabalho consiste em desenvolver um *framework* que possa apoiar o uso da tecnologia de captura de movimentos em aplicações com propostas gerais e simples, para que a interação provida pela captura de movimentos seja um reforço atrativo, oferecendo aos desenvolvedores iniciantes, uma opção de interação fácil e prática comparada com a interação com dispositivos tradicionais como o mouse.

A motivação foi encontrada ao observar jogos eletrônicos educacionais, que não possuem os mesmos recursos que os jogos eletrônicos famosos de alto orçamento disponibilizam. Logo, oferecer uma forma de interação através da captura de movimentos de fácil desenvolvimento pode torná-los mais atrativos, e disponibilizar a desenvolvedores outras possibilidades de interação com nível de dificuldade semelhante a utilização do mouse como dispositivo de interação. Apesar da motivação surgir de jogos educacionais, a utilização em qualquer aplicação que utilize o mouse como meio de interação é viável .

A proposta é desenvolver um *framework* de captura de movimentos que apoie os desenvolvedores que utilizarão como interação o movimento capturado das mãos em diversas aplicações, focado em duas ações: pegar e girar, sendo possível desenvolver uma abordagem de alto nível para desenvolvedores iniciantes, utilizando a captura de movimentos de maneira simples e otimizada, sendo possível comparar sua dificuldade e fazer uma concorrência com o mais tradicional meio de interação para estas aplicações - o mouse -

Para o desenvolvimento do *framework* proposto será utilizado o dispositivo da Microsoft para captura de movimentos (Kinect), bem como sua SDK, utilizada como base para o desenvolvimento.

A validação do *framework* proposto se faz em duas etapas: desenvolver um estudo de caso utilizando três diferentes abordagens (SDK disponibilizada pela Microsoft; o *framework* proposto denominado - *EasyMoCapHand* - e a interação através do mouse); a realização de um experimento com desenvolvedores iniciantes (alunos de um curso técnico em informática) que consiste em um treinamento, desenvolvimento de atividades e percepção através do preenchimento de questionários para cada uma das três abordagens.

Ao final das duas etapas objetiva-se exemplificar aspectos relevantes do desenvolvimento de cada abordagem, sendo possível a validação do proposto por este trabalho.

1.3 Organização do Trabalho

O presente trabalho está organizado em cinco capítulos. O Capítulo 1 apresentou o contexto e motivação do trabalho a ser desenvolvido, os objetivos e metodologias pretendidas.

O Capítulo 2 apresenta conceitos relacionados a Captura de movimentos, que é o tema central do trabalho, abordando tecnologias, dispositivos e áreas de aplicação e trabalho correlatos, afim de contextualizar o conceito.

O Capítulo 3, apresenta o *framework* desenvolvido, apresentando-o e colocando a prova através de um estudo de caso, desenvolvido em três abordagens diferentes de interação: utilizando o mouse, o *framework* proposto e *SDK Microsoft Kinect,* sendo o mesmo, a base do *framework* proposto. Também demonstra as tecnologias e ferramentas utilizadas para o desenvolvimento deste trabalho.

O Capitulo 4 é desenvolvido em torno de um experimento, onde um grupo de participantes - através de um treinamento - realize o desenvolvimento de algumas atividades utilizando as três abordagens expostas no capitulo 3, e ao final através de um questionário demonstrem a percepção do que foi proposto por este trabalho.

Por fim, o capitulo 5 apresenta as considerações finais do trabalho, apresentando as contribuições, limitações e projetos futuros para este trabalho.

1.4 Contribuições do Trabalho

É possível determinar que a principal contribuição do trabalho vem claro no título, que é facilitar o uso da captura de movimentos da mão como interação, e foi atendia, através de um experimento que traz evidências que demonstram que o tempo ao se desenvolver uma aplicação utilizando o *framework* apresentado é

menor que quando utilizado SDK Microsoft Kinect, e que necessita de uma curva de aprendizagem menor, já que o tempo de treinamento e a quantidade de material de apoio é menor. E quando a comparação é com a abordagem através de interação com o mouse, o *framework* se mostrou em nível de igualdade, em tempo de desenvolvimento quanto de dificuldade em aplicações simples.

Capítulo 2

CAPTURA DE MOVIMENTOS

Atualmente, a captura de movimentos é realizada com equipamentos e aplicativos, tecnologias e custos diversos. Essa técnica é bem popular em produções de cinema milionárias ou jogos de computador com um grande orçamento, já que na animação de um personagem, a captura de movimentos consegue resultados muito mais realistas. Também vem sendo utilizado como meio de interação pelas principais industrias de jogos.

2.1 Fundamentação Teórica

A mais simples definição que encontrada para captura de movimentos foi definida por Menache (2011), que é a captura do movimento do mundo real em um determinado evento e o transformar em coordenadas matemáticas úteis, para uma possível reprodução do movimento capturado.

2.1.1 Início da Captura de Movimento

De acordo com Menache (2011) e Kitagawa & Windsor (2008) diversos estudiosos contribuíram para o surgimento da técnica de captura de movimento. Eadweard Muybridge em 1879 realizou o que pode ser considerada uma das primeiras capturas de movimentos ao fotografar um cavalo galopando usando várias

câmeras, que foram acionadas através das patas do animal como visto na figura abaixo.

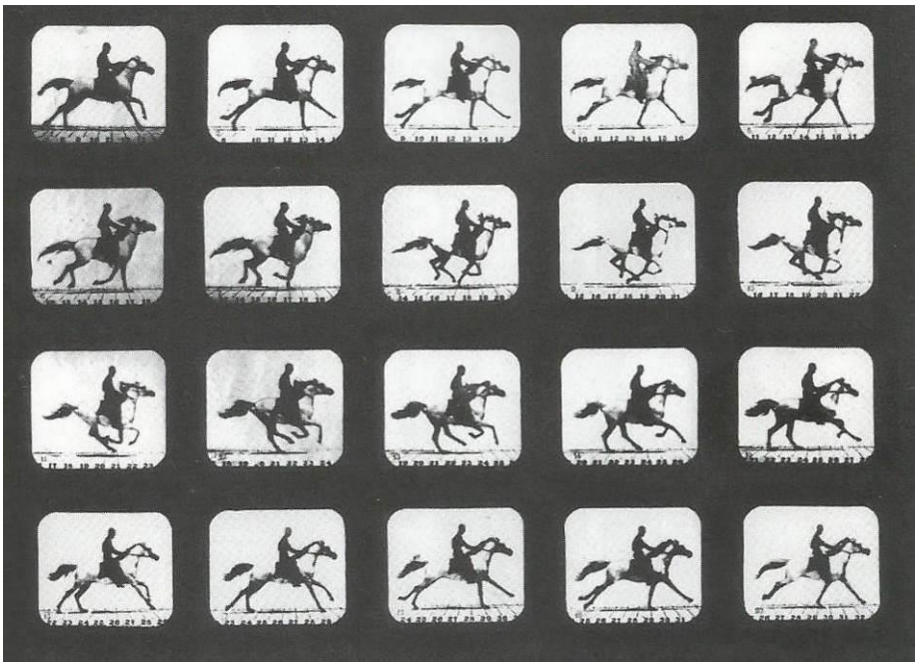

Figura 2.1: Sequencia de fotos obtida por Eadweard Muybridge – (KITAGAWA; WINDSOR, 2008).

Eadweard Muybridge contribuiu também para os primeiros passos do cinema ao inventar o zoopraxiscópio, um dispositivo capaz de exibir uma série de imagens em sequência, dando a impressão de movimento (FLAM,2009). Kitagawa & Windsor (2008) dizem que o zoopraxiscópio é considerado um dos mais antigos dispositivos de captura de movimentos.

Por volta de 1915, Max Fleischer fez sua primeira animação usando um rotoscópio, equipamento responsável por projetar e paralisar cada quadro de um filme. Ele filmou seu irmão movimentando-se e produziu desenhos, quadro a quadro, que combinados tornaram-se uma animação. Mas, devido às dificuldades só foi possível ser concluída um ano após seu início (MENACHE, 2011).

Edgerton criou um instrumento conhecido como estroboscópio em 1931. O estroboscópio é capaz de capturar fotografias nítidas de objetos em movimentos cíclicos de alta velocidade “congelando-os”, piscando luzes na mesma frequência em que o movimento ocorre. Todos esses inventos e criações analógicos impulsionaram o desenvolvimento da captura de movimento em modo bidimensional. Entretanto, somente o surgimento dos computadores e a evolução do cinema

possibilitaram a solidificação da técnica e, mais tarde, a gravação de movimentos tridimensionais na era digital (FLAM, 2009).

2.1.2 Início da Captura de Movimentos digital

Apresentado por Menache (2011) e Kitagawa & Windsor (2008), o desenvolvimento da captura de movimentos digital teve seu início por volta da década de 70, principalmente em aplicações para a área militar e saúde. A indústria de imagens geradas por computador (do inglês *computer-generated imagery* — CGI) somente descobriu as potenciais aplicações da técnica nos anos 80.

De acordo com Flam (2009), o primeiro software de animação tridimensional comercializado foi produzido pela Wavefront Technologies nessa época, onde existia apenas algumas empresas com foco em CGI e produziam comerciais curtos de logos voadores e animados para seus clientes.

Em 1985, os telespectadores foram surpreendidos durante o comercial do Super Bowl (jogo final de futebol americano) ao ver o comercial “Brilliance”. Os criadores desse comercial inventaram sua própria metodologia para animar a personagem. Eles pintaram de preto dezoito juntas de uma atriz que se movia enquanto tiravam fotos de diversos ângulos com várias câmeras. Por não possuírem computadores que fossem capazes de renderizar essa animação, os autores tiveram que pedir emprestado por duas semanas, vários computadores VAX 11, existentes nos Estados Unidos (MENACHE, 2011).

Menache (2011) e Kitagawa & Windsor (2008) destacam que, neste comercial um robô com corpo de uma mulher executava movimentos com uma suavidade muito realista, o que foi um feito notório para época, e por isso é considerado como o primeiro caso de sucesso. Porém, o filme de ficção científica Total Recall (1990), estrelado por Arnold Schwarzenegger e Sharon Stone, por outro lado, foi o primeiro caso de fracasso da técnica. A Metrolight, empresa responsável pelos efeitos especiais do longa, não conseguiu produzir a animação de esqueletos (uma cena em que pessoas passam por uma máquina de raio-x) com dados de captura de movimento, resultando em uma cena ruim.

Continuando na indústria de entretenimento, a área de jogos eletrônicos foi importante para o fortalecimento do uso da captura de movimento digital, citando o jogo de combate FX Fighter, lançado em 1995. Os lutadores são animados em

tempo real com dados obtidos em seções de captura digital de movimento de golpes, como chutes e socos, demonstrando que a captura de movimentos pode ser considerada uma fonte de recursos para diversas áreas.

2.2 Tipos De Sistemas De Captura De Movimentos

Gomide (2006), Kitagawa & Windsor (2008) e Furniss (2015) classificam os sistemas de captura de movimento em três classes principais, dependendo do princípio físico empregado, que são: sistemas ópticos, sistemas magnéticos e sistemas mecânicos, sendo que cada um deles possui pontos fortes e fracos que podem ser explorados.

Geralmente a sequência utilizada para um sistema de captura de movimentos pode ser observada através da Figura 2.2

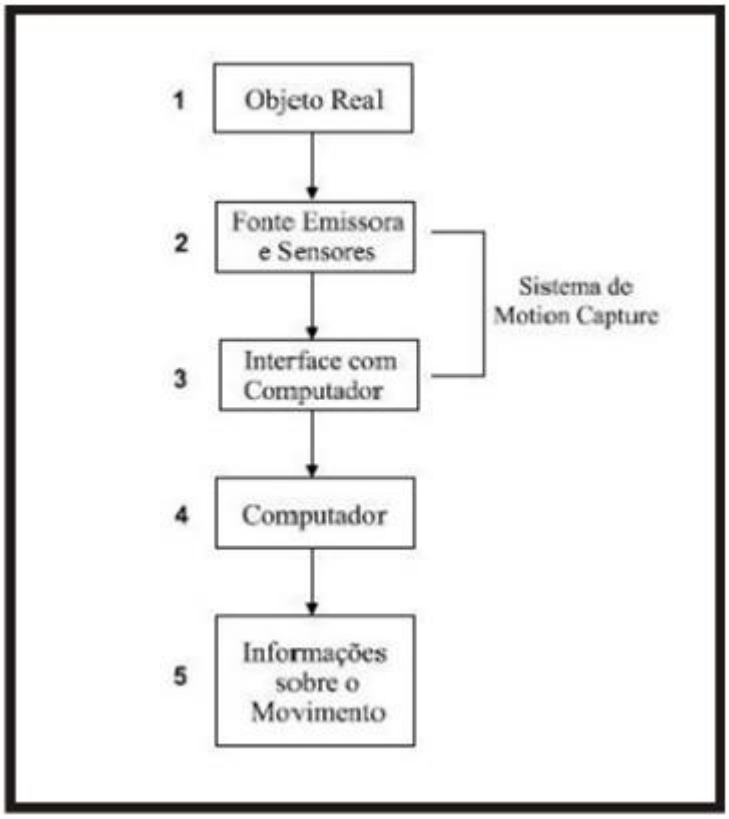

Figura 2.2: Representação de sequência – (SILVA, 1998).

Para acontecer a captura de movimentos, é necessário estabelecer pontos chaves, que são áreas que melhor representam o movimento do objeto a ter seu movimento capturado. Estes pontos devem ser os pontos de articulação ou conexões entre as partes do objeto. Por exemplo, em um ser humano, alguns dos pontos chaves são as articulações, ombros e cotovelos. A localização de cada um desses pontos é identificada por um ou mais sensores, marcadores ou potenciômetros. Com esses pontos bem identificados e definidos, será mais fácil e eficaz a captura de movimentos do objeto alvo.

2.2.1 Sistemas Magnéticos

Em sistemas magnéticos, Silva (1998) afirma que a captura se caracteriza pela velocidade de processamento dos dados capturados. Nesses sistemas, um conjunto de receptores é posicionado nas articulações de um ator. Estes receptores medem a posição e orientação dos pontos chaves em relação a uma antena transmissora, que emite um sinal de pulso. Cada receptor necessita de um cabo que se conecta à antena.

Figura 2.3: Imagem da disposição dos sensores magnéticos de captura – (BODENHEIMER; ROSE, 1997).

Os sistemas magnéticos fornecem dados em tempo real, sendo que o diretor e os atores podem observar os resultados da captura de movimentos, tanto durante

a tomada real ou imediatamente após a captura, com a capacidade ilimitada para ajustar a câmera para uma melhor visualização. Este ciclo de feedback apertado faz a captura de movimento magnético ideal para situações em que a gama de movimento é limitada e a interação direta entre o ator, diretor e personagem/computador é importante. Os sistemas magnéticos são ideais para captura de movimentos mais simples, cuja algumas outras vantagens deste sistema são (DYER; MARTIN; ZULAUF, 1995):

- Tempo de resposta imediato;
- Baixo custo computacional;
- Considerada uma tecnologia de baixo custo;
- Os sensores em momento algum são oclusos;

Mas estes sistemas possuem alguns problemas que delimitam o trabalho do ator e da captura do movimento. As principais desvantagens são:

- Interferência de metais, até mesmo a estrutura do prédio;
- Atores ficam limitados pela quantidade de cabos;
- Área de captura limitada.

Sistemas magnéticos são úteis para detectar objetos escondidos e fornece sistemas de baixo custo em comparação ao sistema de captura de movimento óptico. No entanto, convencionalmente, no sistema magnético de captura, os receptores não estão livres de fiação elétrica (RAAB et al., 1979).

2.2.2 Sistemas Mecânicos

Sistemas mecânicos são compostos de componentes eletrônicos chamados potenciômetros, que é um componente que tem a finalidade de dificultar ou facilitar a passagem da corrente elétrica.

Potenciômetros, quando posicionados nas articulações desejadas, fornecem suas posições e orientações em tempo real. Estes sistemas não são afetados por campos magnéticos ou reflexões indesejadas, problemas que afetam os sistemas magnéticos e ópticos. Por esse motivo, o sistema não necessita de um processo de

calibragem muito longo, o que o torna muito mais fácil de utilizar. As principais vantagens de se utilizar sistemas mecânicos (KITAGAWA; WINDSOR, 2008):

- Liberdade de ação maior do que em sistemas magnéticos;
- Sistema em tempo real;
- Captura de movimentos de vários atores em uma mesma sessão;
- São soluções simples para se capturar movimentos humanos;
- Não recebem interferência de campos magnéticos e de reflexos.

Os sistemas mecânicos ficam limitados ao desempenho do ator, tornando a realidade da cena muito dependente do animador e do tipo da armadura criada. Algumas das principais desvantagens desses sistemas são (MENACHE, 2011):

- Movimentos ficam restritos e às vezes artificiais;
- Alta dependência da habilidade do ator e animador;
- Os sensores são fixos;

A Figura 2 mostra um exemplo de um sistema mecânico de captura.

Figura 2.4: Armadura com potenciômetro para captura do tipo mecânica – (SILVA, 1998).

2.2.3 Sistemas Ópticos

Sistemas ópticos geralmente são usados em áreas voltadas à medicina esportiva, que utiliza estes dados capturados para melhorar a saúde ou, até mesmo, o desempenho de um atleta. Confirmado por Menache (2011), o qual menciona que, a análise esportiva é a maior aplicação da captura de movimento.

Sistemas ópticos de captura utilizam-se de marcadores ópticos, também conhecidos como refletores. Geralmente, são esferas feitas ou cobertas de material reflexivo. Essas esferas são espalhadas em pontos chaves do ator, como juntas, articulações, que são pontos relevantes para a captura do movimento.

Figura 2.5: Imagem da disposição dos marcadores ópticos – (KITAGAWA; WINDSOR, 2008).

De acordo com Silva (1998) e Menache (2011), o funcionamento do sistema óptico é feito por um único computador, que controla a entrada de várias câmeras digitais CCD (*Charge-Coupled Device*) sensíveis à luz. Estas são câmeras especiais que contém um conjunto de LED's (*Light Emitter Diode*) ao redor da lente, que emitem luz infravermelha aos marcadores. A partir deste ponto, os marcadores já estão bem mais visíveis, conseguindo as localizações no espaço dos marcadores para assim, realizar a captura de movimentos. São utilizados algoritmos de processamento de imagens que são rodados a cada *frame* (quadro). Quanto mais câmeras utilizadas em um sistema óptico, mais vezes esse processo será realizado.

Se o sistema for composto de duas câmeras, o processo de identificação dos marcadores seria realizado duas vezes, um para cada câmera, dobrando o processamento computacional.

Os pontos identificados em cada câmera são as coordenadas 2D (bidimensional) da imagem. Essas coordenadas serão tratadas por um algoritmo que irá transformá-las em um ponto 3D (tridimensional). Os pontos tridimensionais serão dispostos em um esqueleto virtual.

O processo de captura será finalizado quando os pontos capturados estiverem cada um em seu respectivo lugar no esqueleto virtual, onde os dados dos marcadores são guardados em arquivos, para serem utilizados ao longo da aplicação do processo de captura de movimento.

Menache (2011), cita que as principais vantagens desse sistema são:

- Os dados ópticos são, geralmente, muito precisos;
- Não há limitação de marcadores;
- Posicionamento de marcadores é flexível;
- Liberdade na área de captura, dependente apenas do campo de visão da câmera
- Captura de movimentos ultrarrápidos
- Liberdade de performance do ator

Como principais desvantagens:

- É um sistema extremamente caro computacionalmente;
- Marcadores podem ser ocultados pelo movimento, podendo causar perda de informações;
- Possível interferência da iluminação do ambiente;
- Precisão variante de acordo com dispositivos e algoritmos

Também é possível através de algoritmos e remoção de fundos, conseguir capturar movimentos de atores e objetos sem a necessidade de marcadores especiais, como por exemplo, fazendo a busca por uma silhueta humana, como mostra as figuras 2.6

Figura 2.6: remoção de fundo separando o movimento do ator – (GOMES; FERNANDES; 2003).

2.2.4 Classificações dos Sistemas de Captura

Menache (2011) classifica sistemas de captura de movimento como: *outside-in*, *inside-out* e sistemas *inside-in*. A classificação é feita através da fonte de captura ou sensores onde os marcadores estão posicionados.

Sistemas *Inside-in:* os sensores e a fonte transmissora estão localizados no próprio corpo do ator. Esses sensores são os mais indicados para a captura de movimentos de objetos pequenos, como os dedos das mãos e olhos. Luvas digitais utilizadas em manipulação 3D se encaixam nessa classificação (SILVA, 1998).

Sistemas *Inside-out:* são sistemas onde os sensores estão conectados ao corpo do ator, e que respondem a sinais emitidos por uma fonte emissora externa. O sistema magnético se encontra nesta categoria. A área de trabalho e a precisão de sistemas *inside-out* estão limitadas devido ao uso de fontes externas, mas em compensação, os dados obtidos fornecem uma boa descrição 3D dos marcadores (MENACHE, 2011).

Sistemas *Outside-In*: a fonte emissora se localiza no corpo do ator, enquanto que, os sensores externos capturam seus sinais. Os sistemas ópticos se encaixam nessa categoria, onde as esferas ou marcadores refletem a luz e as câmeras capturam os sinais. (SILVA, 1998).

2.3 Captura de movimento da mão

A captura de movimentos da mão é algo muito difícil de se realizar, porém ao mesmo tempo, muito valioso. Quando se lida com captura de movimentos de mãos e dedos, por serem áreas pequenas, torna-se uma tarefa muito complicada para colocar marcações (KITAGAWA; WINDSOR, 2008).

Kitagawa & Windsor (2008), propõe que um ótimo lugar para começar é o estudo da anatomia das mãos. Complementa dizendo que é necessário saber como a mão é estruturada, como ela se movimenta e qual o tipo de movimento você deseja capturar.

Observando a Figura 2.7 a quantidade de pontos marcadores que Penelle & Debeir (2014) utilizaram para conseguir identificar o estado que a mão se encontrava. É possível perceber que, a ponta e a base de cada dedo são definidas como pontos chave para a identificação deste estado.

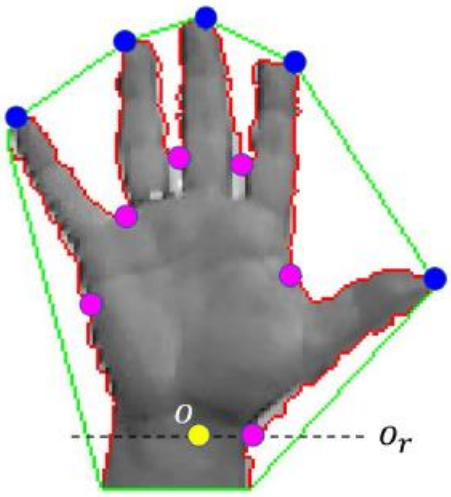

Figura 2.7: Imagens de sistema identificando dedos e espaço da mão – (PENELLE; DEBEIR, 2014).

Considerando ainda que podemos encontrar a mão fechada, que transforma toda a forma de captura de movimentos, principalmente se esta captura utilizar os movimentos dos dedos para algum tipo de aplicativo.

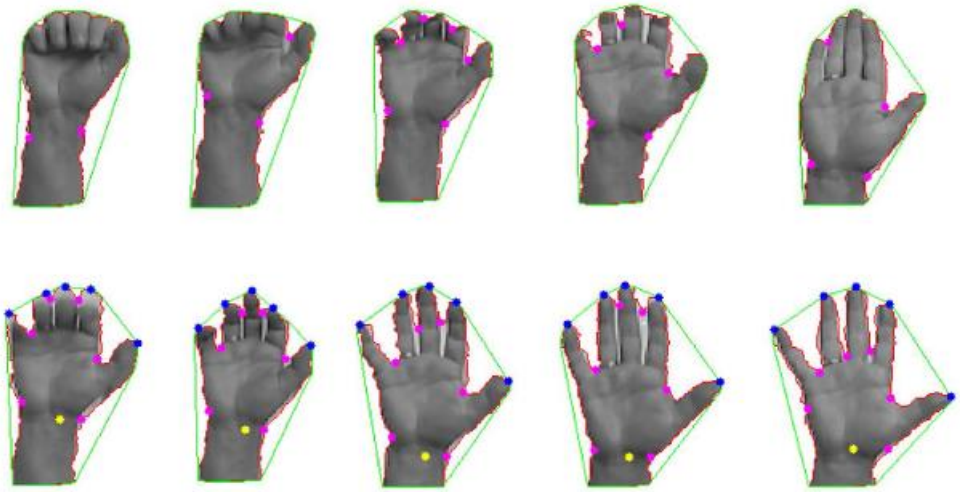

Figura 2.8: Imagens de sistema desde a mão totalmente fechada até totalmente aberta – (PENELLE; DEBEIR, 2014).

Um aplicativo ou sistema que utiliza a captura de movimentos como forma de interação provavelmente precisará capturar os movimentos das mãos. Essa tarefa pode ser muito árdua, principalmente pela quantidade de técnicas e conhecimentos necessários para tal desenvolvimento, ainda mais se tratando de uma captura de movimentos e utilização desses dados em tempo real e também quantidade de pontos chave demonstrados pela figura 2.8.

No mercado já existem alguns dispositivos de baixo custo como por exemplo o sensor da Microsoft Kinect, que entrega boa parte do trabalho realizado, para que o uso da captura de movimentos seja mais fácil e difundido.

2.4 Dispositivos de captura de movimentos

Existem diversos dispositivos e sistemas com custo de milhares dólares, estes utilizados principalmente pelas indústrias de entendimento como cinema e jogos eletrônicos (MENACHE, 2011), até dispositivos considerados como baixo custo, esses três serão mais enfatizados por apresentarem maior utilização entre desenvolvedores: Leap Motion, Kinect, Wii Remote Control.

Porém existem diversas opções. Um outro exemplo seria a pulseira Myo, que é um dispositivo de controle de movimento sem fio, que se comanda através de gestos de antebraço e mão, e permite utilizar e controlar um telefone celular, computador, e outros recursos (MYO, 2015).

2.4.1 Leap Motion

O *Leap Motion* é um computador periférico *USB* lançado em meados de 2013, que utiliza câmeras infravermelho para capturar movimentos de mão, dedo e conseguir tudo isso com precisão (HANTRAKUL; KACZMAREK, 2014).

O *Leap Motion* requer um mínimo de recursos para seu funcionamento (LEAP MOTION, 2015):

- Windows® 7/8 or Mac® OS X 10.7
- Processadores AMD Phenom™ II or Intel® Core™ i3/i5/i7
- 2 GB RAM
- USB 2.0 port

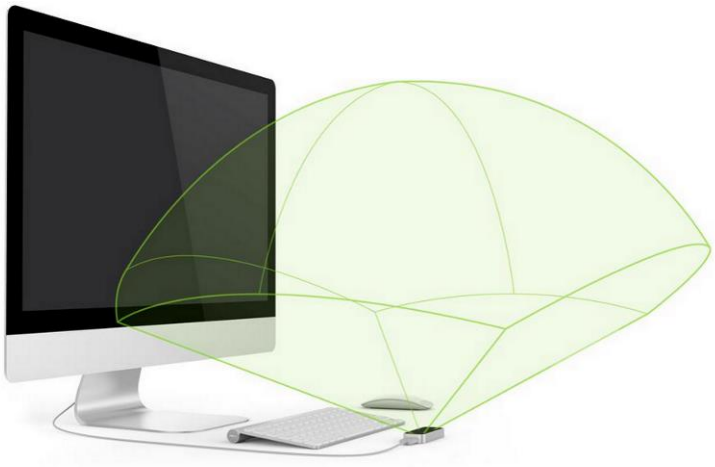

Figura 2.9: Dispositivo Leap Motion e sua área de abrangência – (LEAP MOTION; 2015).

Alguns exemplos de trabalhos propostos com este dispositivo vão desde o reconhecimento de linguagem de sinais (POTTER et al., 2013), ao uso de movimentos gestuais para controle de áudio, que são tendenciais tanto como entretenimento e indústrias de áudio profissional (HANTRAKUL; KACZMAREK, 2014).

2.4.2 Wii remote control

O Wii *remote control* inovou o mercado, pois de uma só vez, inclui várias tecnologias que quase nenhum dispositivo possuía, que foram: 3 eixos de acelerômetro, câmera infravermelha 1024x768 entre outros (MUNOZ ET AL, 2009).

Figura 2.10: Wii *remote control* – (WII; 2015).

Com toda essa inovação na Industria de jogos, logo este dispositivo foi utilizado por desenvolvedores e pesquisadores para diversos fins, como o proposto por Santhanam (2012), que avaliou o uso do Wii *remote* para o uso como um mouse sem fio, padrão para navegação na internet. Porém, seu foco era alcançar as pessoas com paralisia cerebral. Nove participantes com essa deficiência tinham que realizar três tarefas cotidianas de utilização da internet. Os resultados do trabalho apresentam que seis conseguiram melhores tempos quando da utilização do dispositivo. Sendo assim, com uma personalização adequada, pode ser utilizado como uma grande ferramenta para a inclusão digital de pessoas com esse tipo de deficiência.

Seu uso para este projeto foi descartado visto que este dispositivo saiu do mercado e seu foco nunca foi para o desenvolvimento de aplicações.

2.4.3 Kinect

Lançado em novembro de 2010, é um dispositivo com sensor de movimento desenvolvido pela Microsoft para o console de videogame Xbox 360 que visa permitir aos jogadores uma interação com os jogos sem a necessidade de um

joystick, através da captura de movimentos. O Kinect é um dispositivo que vai muito além das tentativas anteriores de criar um controle baseado na captura de movimentos. Utilizando a captura de vídeo, áudio e sensores de profundidade, o Kinect é capaz de detectar movimentos, identificar rostos, reconhecer voz e sons e o discurso de jogadores, o que proporciona aos mesmos a utilização dos seus próprios corpos como controles. Ao contrário dos demais dispositivos do tipo, ele não requer um acessório com acelerômetro ou dispositivo com o qual o jogador precise manter contato físico direto para operar. (KINECT HARDWARES, 2015).

Para desenvolvedores, a Microsoft desenvolveu o Kinect for Windows, onde a intenção é disponibilizar os recursos de que ele dispõe para todos os tipos de aplicações, não ficando restrito o seu uso apenas para o vídeo game XBOX.

Figura 2.11: Imagem do Kinect para windows – (KINECT HARDWARES, 2015).

Com a evolução e chegada da nova geração de vídeo games, o Kinect ganhou uma nova versão para *Xbox One.* E desta vez, a Microsoft criou um adaptador para que o mesmo dispositivo do vídeo game possa ser utilizado para dispositivos diversos, além de uma SDK oficial com diversos recursos.

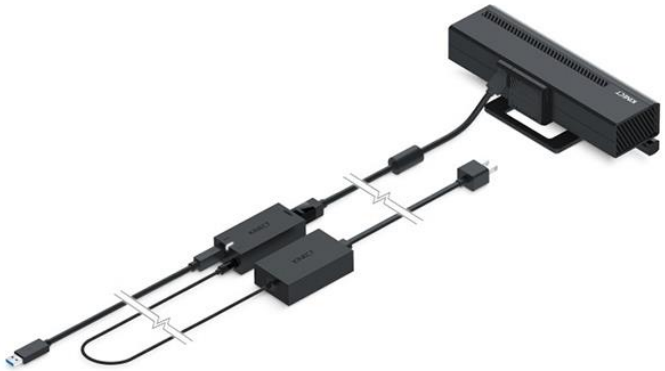

Figura 2.12: Imagem do Kinect v2 e adaptador para uso em computadores – (KINECT, 2015).

2.4.4 Escolha do equipamento

Entre as três opção apresentadas neste trabalho (Wii Remote, Leap Motion e Kinect V2), o *Kinect* foi dispositivo escolhido por prover a interação livre de marcadores, e por estar no mercado de forma mais abrangente, sendo mais fácil de ser adquirido pelo usuário final do aplicativo e desenvolvedor, visto que além de uso em aplicações ele é um periférico de um vídeo gama muito difundido e em uma análise rápida é possível que determinar que o uso de um framework seria muito mais plausível.

O Wii Remote para este projeto foi descartado visto que este dispositivo saiu do mercado e seu foco nunca foi para o desenvolvimento de aplicações.

Apesar do LeaMotion ser desenvolvido para a captura de movimento para mãos seu uso foi descartado pelo presente trabalho produto é raramente encontrado em uso, sendo ele mais utilizados em pesquisas e trabalhos, logo trazendo uma situação de improvável utilização do *framework.*

2.5 Trabalhos relacionados

Há diversos trabalhos publicados que usam o Kinect ou outro dispositivo para captura dos movimentos das mãos. São diversas propostas que abrangem diversas áreas e setores, que incluem desde *frameworks* até aplicações com o uso do movimento capturado através do Kinect e outros dispositivos.

No trabalho apresentado por Pedersoli et al. (2012) é proposto um *framework opensource* que visa uma comunicação mais natural e intuitiva através de gestos. Esse *framework* foi implementado em c++, para o Kinect for Windows v1 que é a versão anterior do dispositivo utilizado neste trabalho, e é mencionado ainda que o *framework* é preparado e tem como objetivo fornecer recursos para desenvolvedores reconhecerem gestões padrões (figura 2.13) para que possa ser usado para a melhoria de comunicação entre máquina e homem através de gestos, ou até mesmo, conseguir enviar mensagens e etc. Semelhante ao proposto por este trabalho, porém Pedersoli et al. (2012) foca o reconhecimento do gesto e oferecer o gesto para o desenvolvedor tomar a decisão do que fazer do mesmo a proposta do

EasyMoCapHand é oferecer a ação, por exemplo pegar o objeto X, de maneira tão simples quanto programa o clicar e arrastar com o mouse.

Figura 2.13: Exemplos de possíveis padrões – (PEDERSOLI ET AL; 2012).

Alexanderson et al. (2012) apresenta método on-line para identificação e rastreamento de marcadores de captura de movimento dos dedos das mãos, como pode ser observado na Figura 2.14, é utilizado uma luva com marcadores para que essa captura seja realizada de forma adequada. Este trabalho apresenta um método de captura que visa melhorar e oferecer uma melhor performance da captura, enquanto o proposto por este trabalho é oferecer um *framework* de fácil utilização que apoie desenvolvedores iniciantes a utilizar captura de movimentos da mão como forma de interação.

Figura 2.14: Luva com marcadores e exemplo de primeiro teste – (ALEXANDERSON et al., 2012).

Nesta proposta é demonstrado que a interação através de captura de movimentos é capaz de proporcionar novas formas de exposição de informações e de interação, como proposto também por Santos et al (2011), que utiliza um Kinect para prover a interação com objetos virtuais através dos seus gestos.

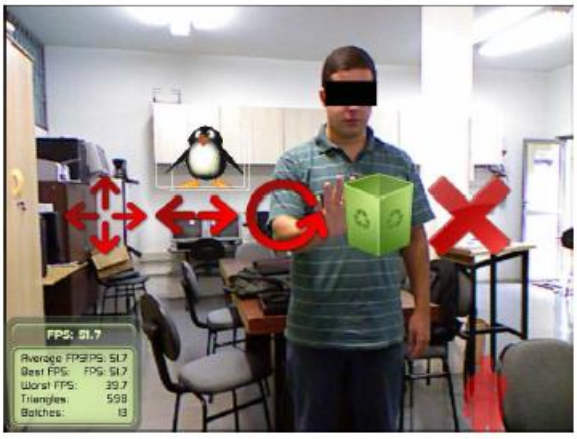

Figura 2.15: Exemplo de utilização do sistema – (SANTOS ET AL; 2011).

O número de possibilidades e aplicações para o reconhecimento de gestos são amplas, tanto quanto setores e funcionalidades, porém a interação se faz através de tempos de espera da mão em cima dos objetos virtuais, com auxílio de realidade aumentada, as ações no objeto são realizadas ficando com a mão parada em cima do objeto por um determinado tempo, tornando diferente do proposto por este trabalho que é interação em tempo real com o objeto sem necessidades de espera, e que ações como girar um objeto dependa apenas se o objeto foi pego ou não, e finalizando Santos oferece um aplicativo para interação e não um *framework*.

Penelle & Debeir (2014), apresentam duas interessantes combinações: primeiro, utiliza dois dispositivos que geralmente são considerados concorrentes (que são o Leap Motion e o Kinect), a proposta é uni-los para conseguir o melhor rastreio possível dos movimentos de mãos e dedos.

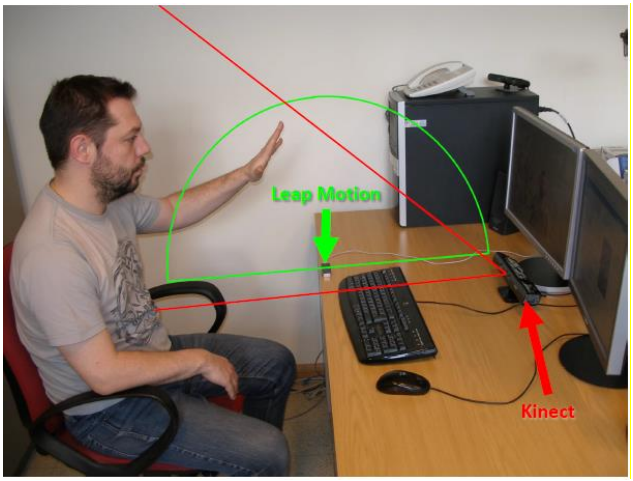

Figura 2.16: Disposição do Kinect e Leap Motion – (PENELLE; DEBEIR; 2014).

A união destes dois sensores é aplicada a uma realidade aumentada que tem como objetivo tratar a dor de membro fantasma em amputados de membros superiores. Sendo assim, o objetivo do Kinect foi adquirir imagens em 3D do paciente em tempo real, já o do Leap Motion, oferecer a captura dos dados mais precisos da mão. Com esses dados capturados, o paciente tem seu membro reconstituído virtualmente, dando-lhe a ilusão de que ele tem os dois braços. Ele tenta utilizar essa virtualização para realizar algumas interações com objetos virtuais, e é relatado que, essa constituição virtual, em alguns casos, permitiu reduzir a intensidade da dor dos pacientes que utilizaram essa ferramenta.

Uma outra utilização de reconhecimento de gestos que pode ser utilizada em níveis comportamentais foi a de Wei (2012), que relata que o comportamento repetitivo de colocar um ou mais dedos na boca, pode interferir na atenção, aprendizado e interações sociais. O uso do Kinect identifica este comportamento de forma mais rápida para que ações possam ser tomadas para inibição deste comportamento, e assim, impedir que a criança tenha seu desenvolvimento normal afetado.

Como há diversas formas de uso, é possível, através do uso do reconhecimento de gestos, fazer jogos interativos, como o jogo pedra-papel-tesoura que foi utilizado como estudo de caso por Ren et al (2011), onde ele propõe um robusto reconhecedor de gestos através do auxílio do dispositivo Kinect. Este trabalho tem como objetivo fornecer um aplicativo que utiliza gestos para interagir com objetos virtuais em um aplicativo de Realidade Aumentada. Também pode ser utilizado para o dia a dia de lugares onde o reconhecimento de gestos seja um atrativo, como por exemplo como proposto por Panger (2012), que visa utilizar os gestos em uma cozinha para navegar por receitas, ativar cronômetros para tempo de cozimentos, alterar músicas, sem tocar em nada, entre outras coisas.

A proposta feita por Zhao et al. (2012) é introduzir uma nova estrutura de aquisições para movimentos de alta fidelidade da mão. A ideia chave é alavancar os dados da posição do marcador através de um sistema de captura de movimento óptico da câmera e dados RGB / Profundidade obtidos através Microsoft Câmera Kinect, em suma, através do Kinect oferecer uma forma de adquirir captura de movimentos com uma grande qualidade.

Um último exemplo que pode ser mostrado é o uso do Kinect para reconhecimento de linguagens gestuais, que seria um auxílio para pessoas com

deficiência na fala conseguirem se comunicar com pessoas que não sabem essa linguagem de sinais (libras), como um tradutor em tempo real. Podemos ver no trabalho proposto por Anjo et al. (2012), que apresenta a identificação de algumas letras do alfabeto da Linguagem Brasileira de Sinais, e também, como Zafrulla et al. (2011) que, baseado em um sistema que utilizava luvas com sensores para identificação dos gestos da linguagem americana de sinais, e adaptou o seu uso para o Microsoft Kinect, que em sua perspectiva, seria de mais fácil utilização, pelo fato de o usuário não precisar vestir nenhum equipamento, sendo necessário apenas a movimentação na frente do sensor.

2.6 Considerações Finais

Neste Capítulo discutiu-se os conceitos relacionados à captura de movimentos, suas características, dispositivos que o utilizam, e sua relevância.

Com esse levantamento bibliográfico, pode-se perceber que a captura de movimentos e reconhecimento de gestos tem uma vasta gama de possibilidades, como pode ser observado no decorrer deste capítulo, sendo possível ir desde aplicações em áreas da saúde como a medicina, ou a níveis comportamentais, chegando até em industriais de entretenimento, como uso em cinema, ou interação em jogos, também propondo novas funcionalidades e formas de interação com aplicativos e dispositivos. O próximo Capítulo apresenta a proposta deste trabalho que visa oferecer a desenvolvedores um apoio quanto ao desenvolvimento de interação utilizando a captura de movimentos, fazendo que a utilização da captura de movimentos seja tão prática como o mais comum meio de interação -mouse-, para aplicações simples.

Capítulo 3

EASYMOCAPHAND

No capítulo anterior foi apresentado o contexto e desenvolvimento da captura de movimentos, surgindo através de orçamentos milionários, sendo acessível somente a grandes setores e indústrias, demonstrando também a evolução para dispositivos de baixo custo, tornando a tecnologia acessível para desenvolvedores de todo o mundo. De acordo com Pedersoli et al. (2012) o Kinect da Microsoft é um bom exemplo de dispositivo de baixo custo. O valor está estimado em U$ 99 dólares, sendo necessário o uso de um adaptador com valor estimado em U$ 40 dólares. Além dos dispositivos evoluírem, houve avanços relacionados ao desenvolvimento de aplicações de captura de movimentos. A programação que outrora era complexa e de baixo nível, tornou-se de alto nível e orientada a objetos.

Uma pergunta surge: se o desenvolvimento está facilitado qual o motivo para o framework proposto? A resposta é simples, enquanto a comparação se der apenas ao desenvolvimento utilizando, a captura de movimentos antiga de grandes empresas com grande orçamentos e disponibilidade de recursos em grande abundância, versus a atual que conta com dispositivos de baixo custo, que chegam a entreter e oferecer recursos a diversas classes de pessoas no dia a dia, não seria necessário o desenvolvimento de um *framework*. Porém, se a comparação for entre a utilização da captura de movimentos com outras formas de interação mais tradicionais, ainda há uma grande diferença.

Um bom exemplo seria utilizar a IDE e *engine Unity* para desenvolver uma aplicação de pegar e arrastar um objeto qualquer, a interação mais tradicional para esse tipo de aplicação seria através do mouse, sendo necessário poucas linhas de código para alcançar o objetivo. O desenvolvedor não teria a necessidade de saber

sobre como a *Unity* extrai as coordenadas do mouse, o conhecimento necessário seria apenas de alguns métodos para desenvolver a aplicação. Em contraponto, utilizando a mesma *engine* através do *plugin* que a Microsoft disponibiliza para o uso do Kinect, o desenvolvedor precisará de algum conhecimento teórico sobre a captura de movimentos, noções sobre orientações e eixos, e a quantidade de código será bem maior, ou seja, o *framework* proposto visa oferecer a captura de movimentos como forma de interação simples, semelhante quanto ao desenvolvimento utilizando-se da interação mais tradicional que se dá através do mouse.

Para demonstrar de forma mais objetiva, este capitulo apresentará na seção 3.1 o *framework* proposto, bem como descrições de métodos, classes e tecnologia utilizadas. Na seção 3.2 será apresentado um estudo de caso desenvolvido utilizando três abordagens: uso do *framework* proposto, SDK Microsoft e interação através do mouse para demonstrar as diferenças entre as abordagens. E por fim, na seção 3.3 considerações finais.

3.1 O *framework – EasyMoCapHand*

O objetivo para esse *framework*, é oferecer um conjunto de métodos que se aproxime em dificuldade e aplicabilidade a interação através da captura de movimentos da mão com a interação através do *mouse*, tornado mais fácil o desenvolvimento em comparação a base oferecida pela *SDK Microsoft Kinect.*

3.1.1.1 Tecnologias, Ferramentas e Dispositivos

Para o desenvolvimento do *framework* e estudos de casos foram necessárias as seguintes tecnologias e dispositivos:

Unity - é um motor de jogo 3D proprietário e uma IDE criado pela Unity Technologies, e que algumas características são importantes a ser ressaltadas: todos os scripts Unity derivam da classe *MonoBehavior*, que difere de um loop de programação normal cujo o código fica se repetindo até que uma condição seja atendida, essa classe base, faz com que a Unity passe o controle para o script somente caso algum evento previamente declarado seja chamado. Ao final do

evento ser completado o Unity retoma controle do Objeto novamente, como pode ser observado através da figura 3.1 o ciclo de vida de classe *MonoBehavior*.

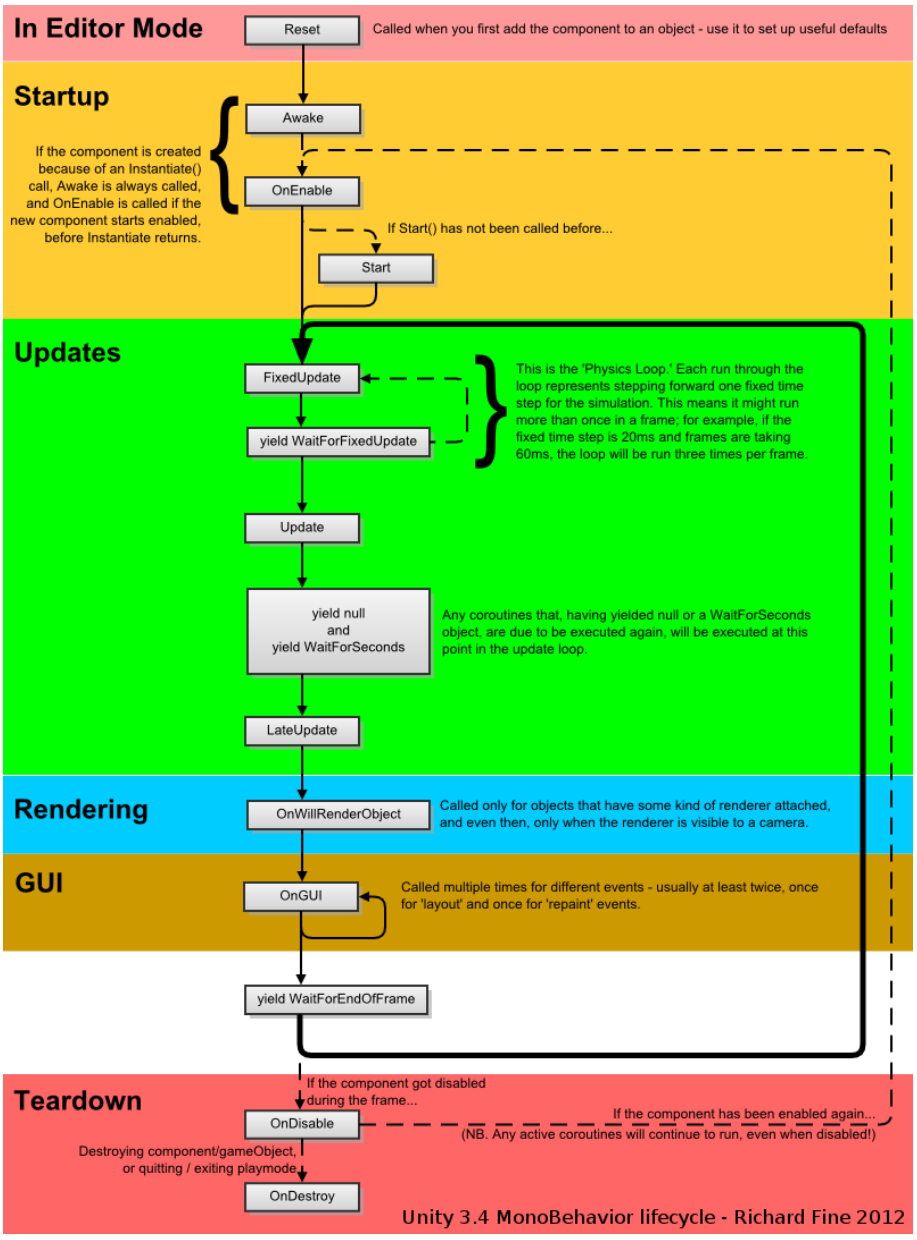

Figura 3.1: Ciclo de vida da classe *MonoBehavior*. (FINE, 2012).

Quando é criado um novo script de comportamento, o *Unity* cria alguns métodos de forma automática, métodos esses que tem funcionalidades específicas e importantes para o desenvolvimento de qualquer aplicação e ainda oferece alguns outros que são essenciais para o desenvolvimento, que são eles:

- *Start* – primeiro método invocado pela aplicação, usado para inicialização de variáveis, instância de objetos e inicialização de sensores.

- *Update* – este método é invocado a cada frame da execução da aplicação, então toda informação ou ação que deva ser realizada e atualizada constantemente em cada frame da aplicação deve ser alocado neste método
- *OnTriggerStay2D* - método fornece todos recursos de um objeto que esta colidindo com o objeto que contém esse método, como por exemplo localização do objeto.
- *OnApplicationQuit* - esse método é invocado somente quando a aplicação se encerra, um exemplo de uso é desligar os sensores que estavam sendo utilizados na aplicação.

Visual Studio 2015 - é um ambiente de desenvolvimento proprietário, desenvolvido pela Microsoft que permite criar aplicativos para Web, Windows, Mac e Linux. A ferramenta é fortemente voltada para desenvolvedores que utilizam a linguagem de programação C#.

SDK Microsoft Kinect v2.0 - permite que desenvolvedores criem aplicações que suportem gestos e até mesmo reconhecimento de voz, utilizando o sensor Kinect v2.0 originalmente desenvolvido para o console *XBOX One*, e adaptado para computadores com o Windows 8 ou superior, oferecendo driver e suporte a todos os recursos do sensor.

Pluggin Unity Kinect for Windows – desenvolvido para ser possível a utilização do sensor Kinect v2.0 na *engine Unity*.

Dispositivo Kinect v2.0 – sensor de captura de movimentos e reconhecimento de voz desenvolvido pela Microsoft, originalmente para o console *Xbox One*.

Adaptador Kinect para computador—para a utilização do Sensor Kinect v2.0 em computadores se fazer necessário o uso de um adaptador desenvolvido pela própria *Microsoft.*

3.1.1.2 Classe e Métodos

O *framework* é denominado *EasyMocapHand* uma abreviação para Easy Motion Capture Hand ou captura fácil de movimentos da mão. Ele é formado pelo conjunto de três classes: HandAction, MotionCaptureResources e HandResources (figura 3.2). Todas três são compostas por recursos da *Unity* e *SDK Microsoft Kinect* e *Pluggin Kinect for Windows - Unity.*

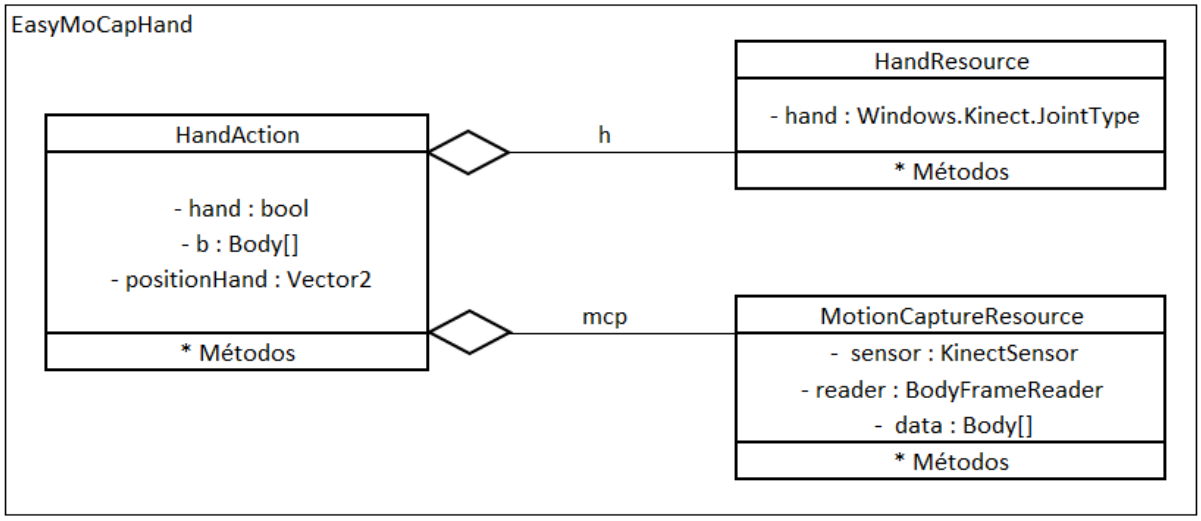

Figura 3.2: Diagrama de Classe Simplificado

Para melhor análise da Classe *HandAction* (figura 3.3), vale ressaltar que esta será a única classe que o desenvolver precisa ter conhecimento e entender seus métodos para ser possível desenvolver aplicações com a proposta de interação através da captura dos movimentos das mãos.

HandAction
- hand : bool - b : Body[] - positionHand : Vector2
+ Start (bool) : void + Update ( ) : void + getPosition( ) : Vector2 + getPosition(ref Transform) : void + getPositionX( ) : float + getPositionY( ) : float + openHand ( ) : bool + turn (Quaternion) : Quaternion + turnObject (ref Collider2D) : void + moveObject ((ref Collider2D) : void + gotObject ( ) : bool + gotObject (bool, bool, Collider2D) : void + quit ( ) : void

Figura 3.3: Representação da classe *HandAction.*

Todos os atributos dessa classe são privados, por não haver a necessidade de acesso pelo desenvolvedor, são eles:

- *hand* – atributo do tipo booleano que controla qual mão será utilizada para captura de movimentos, verdadeiro para mão direita e falso para mão esquerda.
- *b* – Um vetor objeto do tipo *Body* que é a fonte de todas as informações e coordenadas da captura de movimentos *;* ele receberá do objeto da classe *MotionCaptureResource* os corpos que tiveram seus movimentos capturados.
- *positionHand* – um objeto do tipo *Vector2* classe da *engine Unity*, que fornece duas coordenadas (x e y) e métodos para acessa-los e modificá-los, que armazenará as coordenadas x e y da mão escolhida.
- *mcp* – objeto do tipo *MotionCaptureResource*, responsável por fornecer acesso aos recursos de captura de movimentos
- *h* - Objeto da classe *HandResource* responsável por armazenar as informações sobre a mão que tem seus movimentos capturados

Métodos:

- *Start ()* - Método utilizado para inicialização de objetos e variáveis necessários para o desenvolvimento de aplicações que utilizará a captura de movimentos. Este método tem como parâmetro um valor booleano, sendo utilizado para a escolha da mão a ter seu movimento capturado, verdadeiro para mão direita e falso para mão esquerda. Este método deve ser invocado no método homônimo dos scripts de comportamento da *Unity Start*.
- *Update ()* – Tem a função de atualizar os dados da captura de movimentos da mão previamente selecionada. Deve ser invocado no método homônimo dos scripts de comportamento da *Unity Update*, para que a captura de movimentos tenha seus dados atualizados a cada frame.
- *getPosition()* – Retorna um objeto *Vector2,* que armazena as coordenadas x e y da mão escolhida para ter o movimento capturado.
- *getPosition()* – uma sobrecarga que pede como parâmetro posição do objeto que será utilizado para a aplicação do movimento capturado da mão

- *getPositionX( )* – opção oferecida para obter apenas a opção da coordenada x da mão.
- *getPositionY( )* – opção oferecida para obter apenas a opção da coordenada x da mão.
- *openHand ( )* - retorna verdadeiro se a mão estiver aberta.
- *turn ( )* – método recebe como parâmetro um *Quaternion* classe que é baseada em números complexos usado pela Unity, para representar todas as rotações, logo responsável por guardar as orientações para a rotação do objeto e retorna um novo *Quaternion* com a rotação imposta pelo movimento da mão para ser alocado no objeto.
- *turnObject ( )* – recebe a referência do objeto que o cursor colide, e faz que o mesmo seja girado de acordo com o movimento capturado da mão.
- *moveObject ( )* - recebe a referência do objeto que o cursor colide, e faz que o mesmo seja movimentado a partir das coordenadas da mão que teve seus movimentos capturados
- *gotObject ( )* – retorna *true* se a mão estiver fechada, este método deve ser utilizado dentro do método da Unity *OnTriggerStay2D*, pois esse método só é chamando enquanto o objeto estiver colidindo com o cursor.
- *gotObject ( )* – método que recebe dois parâmetros booleanos e uma referência para objeto que o cursor esteja colidindo, se o primeiro parâmetro for passado como verdadeiro, o objeto colidido receberá as mesmas coordenadas da mão, ou seja, será movimentado em x e y, o segundo parâmetro, se verdadeiro, o objeto será girado conforme a orientação da mão com o movimento capturado, caso sejam falsos não realizarão as ações explicadas.
- *quit ( )* – Responsável por finalizar sensor e objetos criados para o uso da captura de movimentos, deve ser alocado no método homônimo da *Unity*.

A próxima classe será a *HandResource* (Figura 3.4). Foi criada para aglomerar os métodos e atributos da mão. A princípio, desenvolvedores não

precisam ter conhecimento dos detalhes de implementação dessa classe, pois a mesma foi desenvolvida apenas para oferecer recursos para a classe *HandAction*.

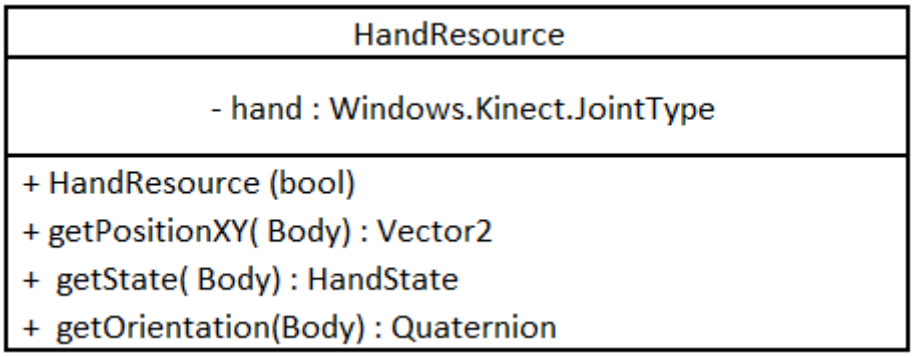

Figura 3.4: Representação da Classe HandResource.

Atributos:

- *hand* – atributo da classe *JointType*, do *pluggin Kinect* for *Windows*, há essa necessidade para manter a *Unity* ciente que este objeto não é do seu tipo *JointType,* classe que identifica o tipo do ponto chave, por exemplo a mão esquerda ou direita, o cotovelo.

Métodos:

- *HandResource* – O construtor da classe que identifica através de seu parâmetro booleano qual mão será utilizada para captura de movimentos, verdadeiro para direita e falso para esquerda
- *getPositionXY* – recebe como parâmetro um objeto *Body*, extrai as coordenadas x e y da mão e as retorna em um objeto *Vector2*
- *getState* – retorna o estado em que a mão se encontra (*Open, Closed, Lasso, Not Tracked, Unknown*.)
- *getOrientation()* – a partir de seu parâmetro *Body,* extrai as coordenadas para orientação através de um retorno *Quaternion*

A última classe é a *MotionCaptureResource* (Figura 3.5). Foi criada para aglomerar os métodos e atributos referentes aos recursos de captura de movimentos. A princípio, desenvolvedores não precisam ter conhecimento dos detalhes de implementação dessa classe, pois a mesma foi desenvolvida apenas para oferecer recursos para a classe *HandAction*.

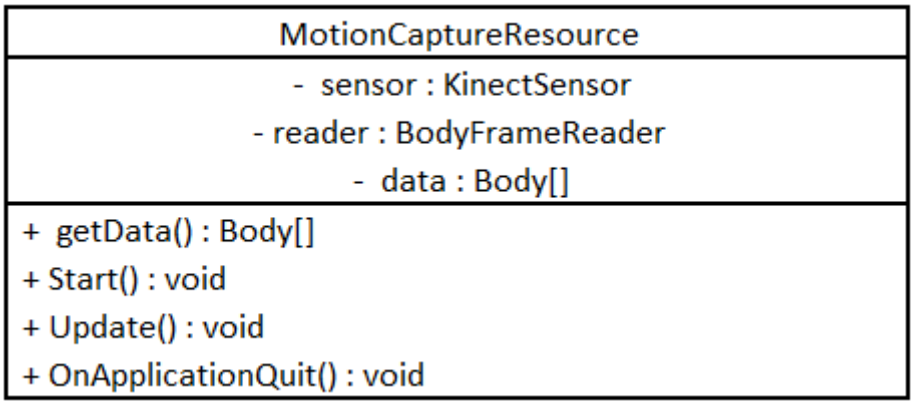

Figura 3.5: Representação da classe *MotionCaptureResources*.

Atributos:

- *sensor* - objeto do tipo *KinectSensor (*classe reponsável pelo dispositivo Kinect*)*, ele armazena o sensor utilizado e fornece os recursos necessários para acessa-lo.
- *reader* - objeto do tipo *BodyFrameReader* responsável por armazenar as informações da captura por frame.
- *data* – vetor do tipo *Body[]* responsável por armazenar todos os corpos capturados pelo frame, uma posição para cada corpo no máximo de 6.

Métodos:

- *getData()* – método *get* convencional do atributo data.
- *Start()* – responsável por inicializar os atributos sensor e reader, bem como testa-los pela possibilidade de estarem com valores nulos.
- *Update()* – Inicializa o vetor *data* com a quantidade de corpos encontrados no frame, e atualizar os dados da captura dos mesmos.
- *OnApplicationQuit()* – fechar sensores e dispor de objetos instanciados pela aplicação.

Com as três classes descritas, a explanação sobre o *framework EasyMoCapHand* está finalizado, pode ser obtido através do link:

https://github.com/pandrade88/EasyMoCapHand.git

3.2 Estudo de caso

Para um melhor vislumbre e exemplificação das facilidades do *framework* proposto, um estudo de caso se faz necessário. A proposta deste estudo de caso se baseia em um jogo conhecido de montagem de palavras através de blocos com letras do alfabeto. A partir deste caso, é proposto o desenvolvimento deste jogo com três diferentes abordagens de interação, sendo elas:

1. Mouse.
2. Captura de movimentos das mãos utilizando a SDK Microsoft Kinect
3. Captura de movimentos das mãos utilizando o *EasyMoCapHand*.

Vale ressaltar que o mouse utilizado a ser comparado, trata-se do dispositivo convencional e não de mouse 3D, que o foco são na utilização de e manuseio de objetos 3D (OLHAR DIGITAL, 2016), já que o foco é a utilização de ambiente 2D neste primeiro momento do *framework*.

3.2.1 O jogo

O jogo foi definido através de um fundo verde semelhante a lousa de giz usualmente encontrada em escolas, um cursor no formato de mão, e cubos coloridos com letras no meio dos mesmos, como pode ser observado na figura 3.6.

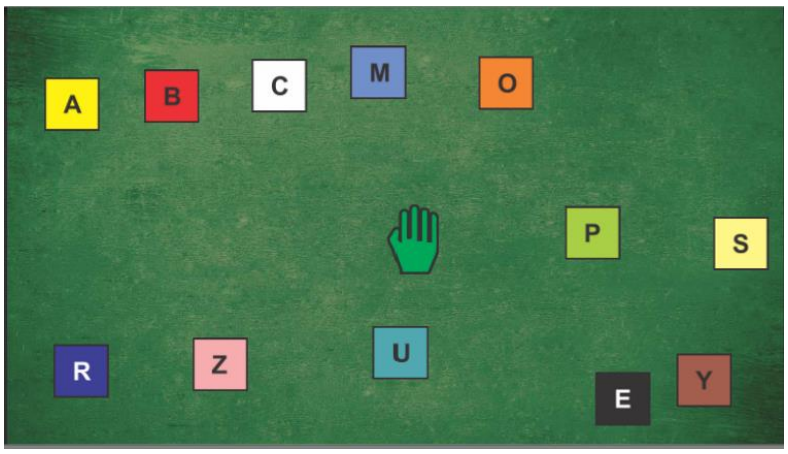

Figura 3.6: Representação da tela do jogo.

Será adicionado um recurso de mudança de formato e cor da mão quando ela estiver aberta ou fechada, mão aberta (verde), mão fechada (azul) como visualizado através da Figura 3.7.

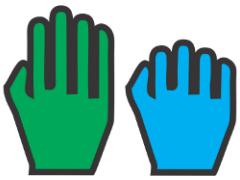

Figura 3.7: Representação da Ação Aberta ou fechada do cursor

Os objetivos deste jogo são simples: ser possível pegar e arrastar os objetos pela cena através do cursor, no intuito de montar palavras com as letras propostas.

3.2.2 Primeira Etapa

Esta etapa demonstrará o que é necessário para desenvolver o jogo nas três abordagens (Mouse, SDK Microsoft e EasyMocapHand) para futuras comparações, que consiste em movimentar o cursor (figura 3.7), aplicando as ações de pegar, arrastar e soltar. Para melhor identificar a ação, o cursor muda de imagem quando houver alterações de seu estado “mão aberta” e” mão fechada”.

3.2.2.1 Abordagem Mouse

Nesta abordagem será utilizado o mouse para a interação do jogo. A posição x e y do mouse servirá para movimentar o cursor pela tela do jogo, já o click será responsável pela ação de pegar. A Game Engine Unity, por padrão, já possui recursos para identificação de eventos como click, e dados sobre a posição do mouse e, sendo assim, nenhuma configuração ou importação de recursos adicional será necessário.

No desenvolvimento do script é necessário a declaração um objeto da classe *Vector3*, como pode ser observado na linha 13 da figura 3.8, mesmo que este trabalho ser focado em ambientes 2D, a *Game Engine* da *Unity*, nos oferece a posição do mouse através deste objeto com três dimensões.

```
13      Vector3 mousePos;
        0 references
14      void Update () {
15          mousePos = Camera.main.ScreenToWorldPoint(Input.mousePosition);
16          mousePos.z = 0f;
17          transform.position = mousePos;
18      }
```

Figura 3.8: Exemplo de código obtendo coordenadas do mouse.

Dentro do método *Update()* da *engine* Unity o objeto *mousePos* recebe o valor de um Objeto do tipo *Vector3*, através de um método que mapeia a posição do mouse na tela, transformando o *Vector3* em coordenadas x, y e z do mouse. Esse método recebe como parâmetro a posição do mouse em coordenadas de pixel, por esse motivo é necessário a utilização do método para conversão em coordenadas em plano cartesiano.

Na linha 16 da figura 3.8, a coordenada z é atualizada para o valor 0, logo abaixo o objeto que contém esse script terá sua posição transformada nas coordenadas oferecidas pelo objeto *mousePos*.

O próximo passo é controlar o click do mouse. Para isso, dois métodos são importantes, *OnMouseDown()* e *OnMouseUp(), onde* ambos os métodos da *Game Engine Unity* são ativadas quando o botão principal do mouse estiver clicado ou não respectivamente. Dentro destes métodos, na linha 29 e 34 da figura 3.9 é trocada a imagem do cursor de aberta e fechada, ou seja, quando o botão esquerdo do mouse estiver clicado a imagem será de mão fechada (azul) e quando não estiver clicado, será de mão aberta (verde). A variável click do tipo *bool* criada servirá como controladora para identificar se o botão do mouse se encontra clicado (*true*) ou não (*false*).

```
25      bool click = false;
        0 references
26      void OnMouseDown()
27      {
28          click = true;
29          gameObject.GetComponent<SpriteRenderer>().sprite = maoFechada;
30      }
        0 references
31      void OnMouseUp()
32      {
33          click = false;
34          gameObject.GetComponent<SpriteRenderer>().sprite = maoAberta;
35
36      }
37  }
```

Figura 3.9: Exemplo de código dos métodos *OnMouseDown()* e *OnMouseUp().*

Utilizando um condicional simples é verificado se o botão está clicado, dentro do método *OnTriggerStay2D,* como demonstra a figura 3.10, que tem como parâmetro o objeto que está colidindo com o cursor (mão), que se a variável click for verdadeira, o mouse está com o botão clicado ou seja, o cursor “pegará” o objeto e irá arrasta-lo até o momento e que o botão esquerdo do mouse deixar de ser clicado.

```
0 references
19  void OnTriggerStay2D(Collider2D colisor)
20  {
21      if (click)
22          colisor.transform.position = mousePos;
23  }
```

Figura 3.10: Exemplo de codificação do Método *OnTriggerStay2D.*

3.2.2.2 Abordagem *SDK Microsoft Kinect*

Nesta abordagem será utilizada a SDK Microsoft Kinect, e o plug-in para o uso na *game engine Unity*. Após as devidas configurações, se faz necessário o uso do *namespace* Windows.Kinect, para facilitar o uso de objetos da SDK no código a ser desenvolvido no Script. O cursor deverá se mover a partir do movimento capturado do ator através do dispositivo *Kinect*, e a ação de pegar e soltar será através do movimento capturado de abrir e fechar da mão do ator.

Para iniciar o desenvolvimento dessa abordagem é necessário a instância de três classes, a primeira responsável pelo sensor físico (*KinectSensor*). Ela será responsável por ativar e desativar o sensor. A segunda (*BodyFrameReader*), é responsável por guardar em cada frame as informações dos corpos dos atores, e por último, um vetor de objetos (*Body*), que guardará os dados de captura dos corpos encontrados em cada frame. A instância de cada objeto pode ser visualizada na figura 3.11.

```
7   private KinectSensor _Sensor;
8   private BodyFrameReader _Reader;
9   private Body[] _Data = null;
```

Figura 3.11: Exemplo de objetos da *SKD Microsoft Kinect.*

Dentro do método *Start* da Unity, é necessário selecionar o Sensor Kinect. Neste trabalho não serão usados múltiplos dispositivos, então, basta selecionar o

sensor padrão (KinectSensor.*GetDefault()*). Uma condicional simples para verificar se o sensor existe, ou seja, diferente de nulo, é aberto o leitor de frames e alocado no objeto adequado (*Reader*). Por fim, uma condicional para verificar se o sensor não está ativo e, se não estiver, o mesmo será ativado. Toda essa descrição pode ser melhor visualizada e entendida na figura 3.12, da linha 25 a 35.

```
25          _Sensor = KinectSensor.GetDefault();
26
27          if (_Sensor != null)
28          {
29              _Reader = _Sensor.BodyFrameSource.OpenReader();
30
31              if (!_Sensor.IsOpen)
32              {
33                  _Sensor.Open();
34              }
35          }
```

Figura 3.12: Trecho de codificação referente a *SDK Microsoft Kinect.*

Dentro do método *Update* da *Unity,* é verificado o objeto (*_Reader*), que guarda os valores dos corpos de cada frame não nulo. Não sendo nulo, é criada uma variável para receber desse mesmo objeto o último frame capturado através do método *AcquireLatestFrame(),* então, é verificado se a variável não é nula, para ter a certeza que um frame foi adquirido. O próximo passo é inicializar o vetor do objeto tipo *Body* com a quantidade de corpos localizados pelo sensor, como pode ser observado na linha 53 da figura 3.13. O método *GetAndRefreshBodyData*(*_Data*), atualiza os dados da captura de movimentos dos corpos para logo em seguida esse frame ser descartado e a variável voltar a ser nula, para que esse procedimento aconteça uma vez a cada frame.

```
43          if (_Reader != null)
44          {
45              var frame = _Reader.AcquireLatestFrame();
46              if (frame != null)
47              {
48                  if (_Data == null)
49                  {
50                      _Data = new Body[_Sensor.BodyFrameSource.BodyCount];
51                  }
52
53                  frame.GetAndRefreshBodyData(_Data);
54
55                  frame.Dispose();
56                  frame = null;
57              }
58          }
```

Figura 3.13: Exemplo de codificação de leitura e obtenção de códigos a cada frame.

Continuando a codificação do método *Update*, logo após as ações anteriores, iniciará a busca por um corpo que seja possível travar suas coordenadas e utiliza-las. Primeiro passo, verificar se o vetor de corpos (*_Data*), não é nulo, com o auxílio de uma estrutura de repetição, percorrer esse vetor de objetos em busca de um corpo que possa ter suas coordenadas utilizadas, o que é demonstrado através da figura 3.14

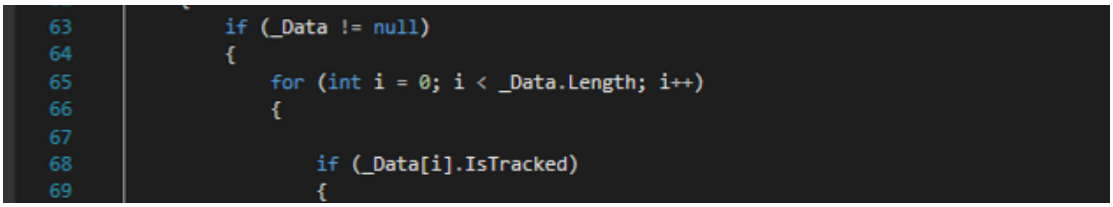

Figura 3.14: Exemplo de estrutura de repetição que percorre o vetor de corpos.

Neste momento, os dados do corpo estão à disposição. Já se torna possível movimentar o cursor a partir dos dados capturados da mão direita como proposto pelo estudo de caso, uma nova instância de um *Vector2* recebe como parâmetro dois valores reais. Esses dois valores são extraídos das juntas (pontos chave) do corpo a qual o movimento foi rastreado. Perceba que as posições x e y são multiplicadas por 10. Esse procedimento é realizado pelos valores das coordenadas disponibilizadas pelo *SDK Microsoft Kinect, que* é proporcionalmente 10 vezes menor que as coordenadas usadas pelo *Unity.* É notório que o objeto que fornece os dados tem um vetor de pontos chave, que neste caso foi escolhido como ponto chave a mão direita (*JointType.HandRight)*. Para oferecer essa posição note que é passado como ponto chave o valor *Windows.Kinect.JointType.HandRight*, por causa que a *Engine* da *Unity*. Também oferece essa classe *JointType*, então, para não haver confusões pela *Unity*, é necessária essa especificação, como pode ser observado na figura 3.15.

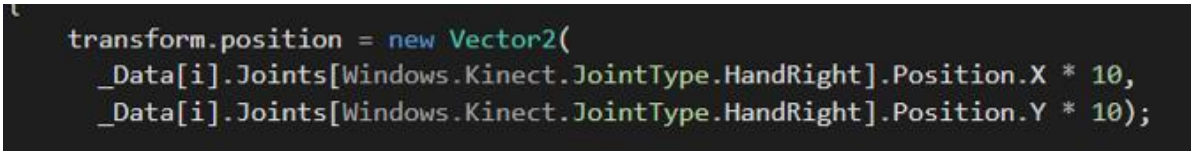

Figura 3.15: Exemplo de codificação específica de tipo de junta pelo.

Para ser possível a realização e a alteração de imagens de mão aberta para mão fechada, como proposto neste estudo de caso, verificou-se através do uso de um condicional, se a mão direita se encontra em três possíveis estados: fechada, dedos indicador e médio esticados (*lasso*) e desconhecido. Todos estes estados serão considerados como mão fechada, ou seja, ela estará apta a pegar o objeto (figura 3.16).

```
84                  if (_Data[i].HandRightState == HandState.Closed ||
85                      _Data[i].HandRightState == HandState.Lasso  ||
86                      _Data[i].HandRightState == HandState.Unknown)
87                  {
88                      gameObject.GetComponent<SpriteRenderer>().sprite = maoFechada;
89                      op = _Data[i].HandRightState;
90                  }
91                  else
92                  {
93                      gameObject.GetComponent<SpriteRenderer>().sprite = maoAberta;
94                      op = _Data[i].HandRightState;
95                  }
```

Figura 3.16: Exemplo de codificação de estados da mão.

Na Figura 3.16 é possível perceber o uso de um objeto de nome OP, que recebe o estado que o ponto chave se encontra naquele momento. Esse objeto que foi criado é da classe *HandState*, sendo inicializado com o estado aberto em primeiro momento como pode ser demonstrado na figura 3.17.

```
HandState op = HandState.Open;
```

Figura 3.17: Exemplo de declaração de objeto op e instância do mesmo.

Dentro do método da *Unity OnTriggerStay2D,* recebe como parâmetro o objeto que colidiu com o cursor. É necessário verificar se essa colisão está ocorrendo com um dos estados que serão considerados como: mão fechada, sendo verdadeiro um dos estados, o objeto colidido receberá, a mesma posição que o cursor. Logo, o objeto será arrastado pela tela do jogo. Toda essa codificação pode ser visualizada na figura 3.18.

```
if (op == HandState.Closed || op == HandState.Lasso || op == HandState.Unknown)
{
    colisor.gameObject.transform.position =
    new Vector2(this.transform.position.x, this.transform.position.y);
}
```

Figura 3.18: Verificação de estado e ação de arrastar objeto.

Quando for encerrada a aplicação é necessário fechar o sensor e se dispor das instâncias criadas para a realização de todo o processo de captura de movimentos, a codificação pode ser observada através da figura 3.19.

```
115         if (_Reader != null)
116         {
117             _Reader.Dispose();
118             _Reader = null;
119         }
120
121         if (_Sensor != null)
122         {
123             if (_Sensor.IsOpen)
124             {
125                 _Sensor.Close();
126             }
127
128             _Sensor = null;
129         }
```

Figura 3.19: Exemplo de codificação para encerramento da aplicação.

3.2.2.3 Abordagem EasyMocapHand

Nesta abordagem será utilizado o *framework EasyMoCapHand,* proposta deste trabalho. A instalação da *SDK Microsoft Kinect* e o *plug-in para* o uso na *game engine Unity,* são necessárias, pois, o *framework* foi desenvolvido utilizando como base estes recursos, sendo assim, necessário a importação das três classes que juntas compõem o *framework EasyMoCapHand.*

Para iniciar o desenvolvimento dessa abordagem é necessário o uso do *namespace EasyMoCapHand*. Com as devidas configurações realizadas, é necessária a instância de apenas um objeto (Figura 3.20) da classe *HandAction*, esse sendo o único objeto necessário para ser utilizado do *framework* proposto no desenvolvimento do estudo de caso.

```
public HandAction hand = new HandAction();
```

Figura 3.20: Declaração e instanciação de objeto da classe *HandAction*.

Dentro do método *Start()* da *Unity*, é necessário que seja chamado o método *Start,* da classe *HandAction* através da instância criada anteriormente, passando como parâmetro verdadeiro para ser capturado o movimento da mão direita como proposto no estudo de caso (Figura 3.21).

```
7    void Start () {
8      hand.Start(true);
9    }
```

Figura 3.21: Utilização do método *Start* do *framework* proposto.

Dentro do método *Update* da *Unity*, o método homônimo também da classe *HandAction,* deve ser invocado para atualização frame a frame dos dados capturados. Para a realização do movimento do cursor, basta atribuir dentro da posição do objeto, o retorno do método *getPosition* da intância da Classe *HandAction,* podendo ser observado na Figura 3.22. O cursor já terá o mesmo movimento da mão direita executado pelo ator.

```
void Update () {
    hand.Update();
    transform.position = hand.getPosition();
}
```

Figura 3.22: Exemplo de codificação do método *Update* do *framework EasyMoCapHand.*

Por fim, a ação de pegar e arrastar é realizada no método *OnTriggerStay2D* da *Unity*, que recebe de parâmetro um objeto que está colidindo com o cursor. Há duas formas para desenvolver tal ação.

Primeira, através de uma condicional simples que receba um verdadeiro (pegou o objeto) ou falso (não pegou o objeto) de um método da classe *HandAction*, posteriormente aplicando o movimento no objeto colidido através do método *moveObject,* como pode ser visto na Figura 3.23.

```
if (hand.gotObject())
{
    hand.moveObject(ref colisor);
}
```

Figura 3.23: Utilização dos métodos *gotObject* e *moveObject* da classe *HandAction.*

Segunda, utilizar a sobrecarga do método *gotObject* da classe *HandAction*, que recebe dois parâmetros booleanos e uma referência para objetos que está colidindo com o cursor, que para sua utilização basta passar verdadeiro no primeiro parâmetro, falso no segundo, como exemplificado na figura 3.24.

```
hand.gotObject(true, false, ref colisor);
```

Figura 3.24: Exemplo de utilização da sobrecarga do método *gotObject*.

Para alternar a imagem do cursor entre “mão aberta” e “mão fechada, basta um condicional simples que receberá a resposta do método *openHand* da classe *HandAction.* Sendo verdadeiro, a mão está aberta e falso, a mão está fechada, como no exemplo da figura 3.25.

```
if (hand.openHand())
    gameObject.GetComponent<SpriteRenderer>().sprite = maoAberta;
else
    gameObject.GetComponent<SpriteRenderer>().sprite = maoFechada;
```

Figura 3.25: Exemplo de uso do método *openHand* do *framework* proposto.

Para finalização, no método *OnApplicationQuit()* da *Unity*, chamar o método *quit* da classe *HandAction,* finalizando o uso do sensor e objetos instanciados exemplificado na figura 3.26.

```
void OnApplicationQuit()
{
    hand.quit();
}
```

Figura 3.26: Exemplo de codificação do método *quit* do *framework EasyMoCapHand.*

3.2.2.4 Análise e comparações entre abordagens

Realizada a explanação do desenvolvimento do jogo proposto como estudo de caso, cuja foi demonstrada as peculiaridades de cada abordagem no que se trata de movimentar o cursor, pegar e arrastar o objeto da cena, e demonstrando, quais foram os objetivos propostos pelo *framework EasyMoCapHand* nesta primeira etapa é possível realizar algumas comparações.

De modo geral sem considerar por um momento a complexidade, e focar apenas na quantidade de código a ser desenvolvido por cada abordagem, a diferença entre a SDK da Microsoft comparado ao *framework* proposto e interação através do mouse é relevante. Excluindo chamadas de métodos da *engine Unity*, e considerando códigos efetivamente para prover interação, a SDK utiliza de quarenta linhas de programação efetivas, enquanto o *framework* proposto utiliza sete linhas e por fim a interação através do mouse utiliza seis linhas, o que é uma grande diferença, ou seja o desenvolvimento com o *framework* proposto é de quase um sexto comparado com a SDK da Microsoft e equivalente a interação através do mouse sendo apenas uma linha maior.

A verbosidade da SDK também reflete a complexidade do desenvolvimento, que logo inicialmente é necessário a criação de três objetos, quatro se considerarmos o objeto auxilio para o estado da mão e um desses objetos sendo um vetor, já para as outras duas abordagens é necessário a instância de apenas um objeto.

No desenvolvimento através da interação do *framework* proposto utilizar apenas uma instância de objeto, o desenvolvedor precisa apenas utilizar sete chamadas de métodos, cujo o desenvolvimento através do mouse são necessárias três chamadas de métodos, alteração de valor em atributos e uma estrutura de decisão, por fim a utilização da SDK fica em torno de doze chamadas de métodos, doze estrutura de decisão, uma estrutura de repetição, sem considerar a quantidade de atributos que o desenvolvedor deve ter conhecimento e precisa fazer uso, como por exemplo os pontos chaves da mão a ter seu movimento capturado (*_Data[i].Joints[Windows.Kinect.JointType.HandRight].Position.X*).

Por fim o *framework* proposto *EasyMoCapHand,* abstrair o conceito da captura de movimentos deixando a programação intuitiva comparada com a SDK, já que o desenvolvedor não precisa ter conhecimento sobre pontos chaves,

coordenadas a serem utilizadas, proporção da coordenada extraída do mundo real a ser aplicada na cena do estudo de caso, tornando desenvolvimento o *EasyMoCapHand* , similar ao desenvolvimento com interação através do mouse.

3.2.3 Segunda Etapa

Nesta etapa será utilizado apenas duas abordagens: SDK Microsoft Kinect e EasyMoCapHand. O foco desta etapa é a ação de rotação, tanto do cursor quanto a do objeto a ser colidido com o cursor, cuja a interação através do mouse convencional não oferecer essas ações de forma direta, descartando essa abordagem. Um exemplo para o uso dessa ação é: se os blocos do jogo viessem girados ou de ponta cabeça como pode ser observado através da Figura 3.27 os blocos “Z” e “O”.

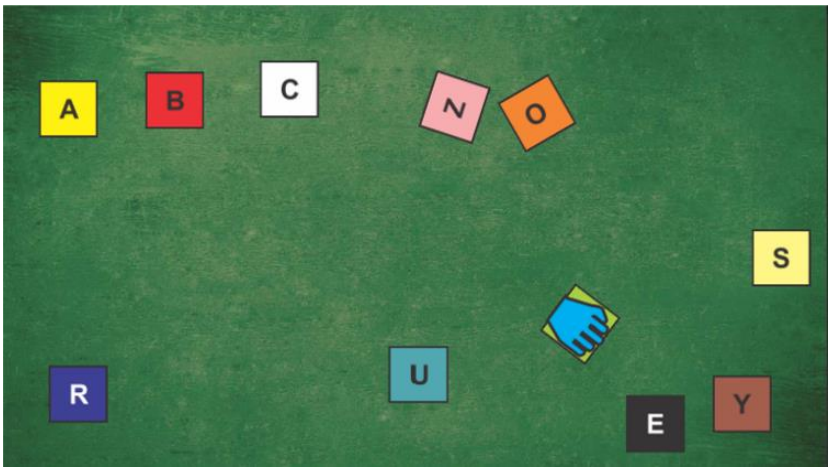

Figura 3.27: Blocos “Z” e “O” girados.

3.2.3.1 Abordagem SDK Microsoft Kinect

Para auxiliar na execução da rotação do cursor e posteriormente do objeto pego, serão necessárias alterações dentro do método *Update* da *Unity*. Porém, antes serão criadas quatro variáveis (ox, oy, oz, ow) reais do tipo *float*, para armazenar as orientações dos pontos chave. Estas orientações virão do corpo, cujos dados estão sendo utilizados (Figura 3.28).

```
if (_Data[i].IsTracked)
{
    transform.position = new Vector2(
      _Data[i].Joints[Windows.Kinect.JointType.HandRight].Position.X * 10,
      _Data[i].Joints[Windows.Kinect.JointType.HandRight].Position.Y * 10);

    ox = _Data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.X;
    oy = _Data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.Y;
    oz = _Data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.Z;
    ow = _Data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.W;
```

Figura 3.28: Obtenção de coordenadas de orientação x, y, z, e w.

Para promover a rotação do cursor é necessário utilizar uma função da classe *Quaternion* chamada *Slerp()*, que basicamente pega a orientação atual do objeto e transforma para uma nova orientação, em determinado intervalo de tempo, como exemplificado na Figura 3.29.

```
transform.localRotation = Quaternion.Slerp(
    transform.localRotation, new Quaternion(oz, oz, oz * 360, ow * 360), 0.08f);
```

Figura 3.29: Exemplo de chamada de método *Slerp.*

O uso de *Quaternion*, que de acordo com Unity Doc (2016), são baseados em números complexos e não são fáceis de compreender de forma intuitiva, usado pela *Unity* para representar todas as rotações, tendo como principais variáveis w, x, y, z.

Para a unidade do *Quaternion* oferecido pelo Kinect ser proporcional na Unity e realizar a rotação de maneira adequada em 2D, o eixo z e w deve ser multiplicado por 360 (DEVELOP NETWORK, 2016).

Por último, deve ser inserido no método *OnTriggerStay2D*, para quando o objeto estiver sendo “segurado” o mesmo também seja aplicada a rotação do cursor, como demonstrado na figura 3.30.

```
if (op == HandState.Closed || op == HandState.Lasso || op == HandState.Unknown)
{
    colisor.gameObject.transform.position =
    new Vector2(this.transform.position.x, this.transform.position.y);
    colisor.transform.localRotation =
    Quaternion.Slerp(this.transform.localRotation,
    new Quaternion(ox, oy, oz * 360, ow * 360), 0.08f);
}
```

Figura 3.30: Exemplo de codificação para rotação do objeto da cena do jogo

3.2.3.2 Abordagem *EasyMoCapHand*

Para ser aplicada a rotação tanto do cursor quanto do objeto, é necessário adicionar um método chamado *turn()* da classe *HandAction* dentro do método *Update* da *Unity*, como demonstrado na Figura 3.31.

```
void Update () {
    hand.Update();
    transform.position = hand.getPosition();
    transform.localRotation = hand.turn(transform.localRotation);
}
```

Figura 3.31: Utilização do método *turn* da classe *HandAction.*

Para finalizar e ser possível exercer a rotação no objeto que esteja colidindo com o cursor será necessário realizar alterações no método OnTriggerStay2D() da *Unity*, podendo ser realizados de duas maneiras, dependendo da escolha na primeira etapa.

A primeira será adicionar o método *turnObject*, recebendo como parâmetro o objeto que esteja colidindo (figura 3.32).

```
if (hand.gotObject())
{
    hand.moveObject(ref colisor);
    hand.turnObject(ref colisor);
}
```

Figura 3.32: Exemplo de uso do método *turn* da classe *HandAction.*

Sendo a segunda uma mudança no segundo parâmetro do método *gotObject*, de *false* para *true*, ficando exatamente como a figura 3.33 demonstra.

```
hand.gotObject(true, true, ref colisor);
```

Figura 3.33: Exemplo de uso do método *gotObject* da classe *HandAction*.

3.2.3.3 Análise e comparações entre abordagens

Ao final desta etapa é possível observar que com apenas uma alteração de parâmetro é possível utilizar a ação de girar quando o desenvolvimento é realizado

através do *framework* proposto *EasyMoCapHand*, agora quanto a utilização da SDK se mostrou mais complexa, por principalmente ter necessidade de se fazer uso de *Quaternion* que na própria documentação da *engine Unity (*UNITY DOC, 2016), diz que são baseados em números complexos e não são fáceis de entender intuitivamente

Acredita-se que ao fim dessa etapa, existem evidências que a abstração alcançada pelo *framework* proposto é relevante e faz com que o desenvolvedor utilize a captura de movimentos como forma interação de maneira tão simples e prática quanto a utilização do mouse convencional como meio de interação, mesmo oferecendo mais recursos como a ação de girar o objeto.

3.3 Considerações Finais

Neste capítulo são abordados aspectos que caracterizam este trabalho, uma visão geral do *framework* bem como suas classes, atributos e métodos desenvolvidos para que através de um estudo de caso, seja possível verificar se com o *framework* proposto, utilizando a captura de movimentos da mão como forma de interação, seja mais fácil do que sendo desenvolvido com a SDK Microsoft Kinect, assemelhando-se ao uso de interação através do mouse.

O estudo de caso baseou-se em desenvolver um jogo com três abordagens, utilizando a SDK Microsoft Kinect, o *framework* proposto e o mouse. Com isso foi demonstrado e explicado todo o processo para o desenvolvimento do jogo com as três abordagens.

Ao final é notável que a abordagem com o mouse não era possível de forma direta a aplicação da ação de girar objetos em cena, definindo que a captura de movimento, traz novos recursos para as aplicações. Já nas abordagens que utiliza captura de movimentos como base, é notório que o grau de dificuldade da SDK é maior que do *framework* proposto, visto que, a quantidade de código é mais do que três vezes maior, a quantidade de objetos, métodos, conceitos necessários é realmente muito mais complexo, comparados com a utilização do *framework* proposto. E mesmo que a interação com o mouse não tenha o recurso de girar, realizando a comparação é possível perceber que a quantidade de códigos e objetos

utilizados tanto na interação com mouse quanto no *EasyMoCapHand*, são extremante semelhantes, e deixando-os com níveis de dificuldades também semelhantes.

Com esses parâmetros expostos, o *framework* traz novos recursos quando comparado com o mouse, e maior facilidade de desenvolvimento quando comparado com a SDK

Este estudo de caso demonstra claramente que o *FrameWork* proposto fornece a interação através de captura de movimentos com a dificuldade de programação de uma interação tradicional realizada através do mouse.

Apesar de o estudo de caso indicar que o objetivo do trabalho foi alcançado e que o *framework* faz o que promete, no próximo capítulo tem como objetivo, apresentar a avaliação deste trabalho, através de um experimento utilizando o *framework,* SKD Microsoft e mouse com um público de programadores inexperientes*,* para desenvolvimento de um jogo como mais uma forma de validação do *framework* proposto, coleta e análise dos dados obtidos.

Capítulo 4

AVALIAÇÃO

No capítulo anterior foi apresentado o *framework EasyMocapHand* e desenvolvido um estudo de caso que demonstrasse e comparasse o uso do mesmo, ao uso da SDK Microsoft Kinect e também com a interação através do mouse. Porém, para uma melhor avaliação, é de grande interesse que seja realizado um experimento para que o *framework* seja melhor avaliado, e que possa ter mais consistência quanto a afirmação de facilitar o uso da captura de movimentos da mão, objetivo deste trabalho.

De acordo com Yin (2015), O planejamento do experimento é caracterizado a partir da análise do problema a ser solucionado. É possível definir em questões os problemas a serem solucionados pelo *framework* proposto.

1. EasyMoCapHand pode facilitar o desenvolvimento de aplicações com o recurso de captura de movimentos das mãos?
2. É possível desenvolver a interação através da captura de movimentos das mãos em uma dificuldade que se assemelhe à interação através do mouse?

4.1 Experimento

Elaborado um experimento que teve como objetivo principal fazer uma comparação do uso da SDK Kinect, uso do mouse e *EasyMocapHand* como meios de interação utilizando o ambiente oferecido pela *engine Unity*. Buscando a

validação do *framework* proposto, para isto, este experimento foi dividido em três partes:

- Treinamento
- Desenvolvimento
- Questionários de Avaliação e Percepção

Foi solicitado junto ao Diretor de uma Escola Técnica que possui o curso de Informática a autorização para divulgar o experimento para os alunos de informática e junto ao Coordenador do Curso de informática, essa divulgação foi realizada através de uma explanação oral (do que era o experimento, para qual fim, como e quando seria realizado). Válido ressaltar que não houve a divulgação, para alunos do primeiro módulo do curso, cujo os mesmos não tinham disciplinas com conteúdo especificados como requisitos mínimos necessários.

Foi definido alguns requisitos mínimos necessários para a participação do experimento que são basicamente: ser aluno de um curso Técnico em Informática ou similar e estar cursando ou ter cursado disciplinas de Desenvolvimento de Software I e Programação de Computadores ou similares, que tenham como diretrizes programação orientada a Objeto, ou seja, participantes que estão iniciando em uma carreira da área de tecnologia da informação.

O experimento contou com a inscrição de doze alunos do curso Técnico de Informática. Porém, foram sorteados apenas oito participantes, quatro para cada grupo, devido a limitação de dispositivos *Kinect* para a realização das atividades. Cada grupo foi formado por dois alunos de terceiro módulo e dois alunos de segundo módulo, tornando assim os grupos o mais homogêneo possível.

4.1.1 Treinamento

O treinamento foi definitivamente expositivo, caracterizado por uma explanação verbal, com auxilio visual de slides, contendo explicações de conceitos, exemplos para a preparação do participante para a realização das atividades que serão propostas, e não um treinamento total da SDK, EasyMocapHand ou mesmo todos os recursos oferecidos para interação através do mouse.

O objetivo era capacitar os participantes para os mesmos conseguirem realizar todas as atividades, através de introdução de conteúdo novo, sistematização de conhecimentos e aprofundamento de conhecimentos (OLIVEIRA, 2016).

O treinamento teve duração total de 32 minutos e 24 segundos, sendo o mesmo dividido em 3 tópicos, como pode ser visualizado na Tabela 4.1.

Tabela 4.1 – Divisão de tempo do treinamento

Tópico	Tempo
Mouse	6 minutos e 20 segundos
SDK MicrosoftKinect	21 minutos e 24 segundos
EasyMoCapHand	4 minutos e 40 segundos

Todas os três tópicos necessitaram de esclarecimentos de dúvidas dos participantes relacionadas aos códigos ou conceitos que lhe foram apresentados. Contudo, é preciso ressaltar que no treinamento, o que gerou o maior número de dúvidas foi referente a SDK Microsoft Kinect, principalmente relacionados a área de rotação da mão, sendo possível observar que a parte desta abordagem ocupou 66% do tempo total de treinamento exemplificado pelo gráfico da figura 4.1.

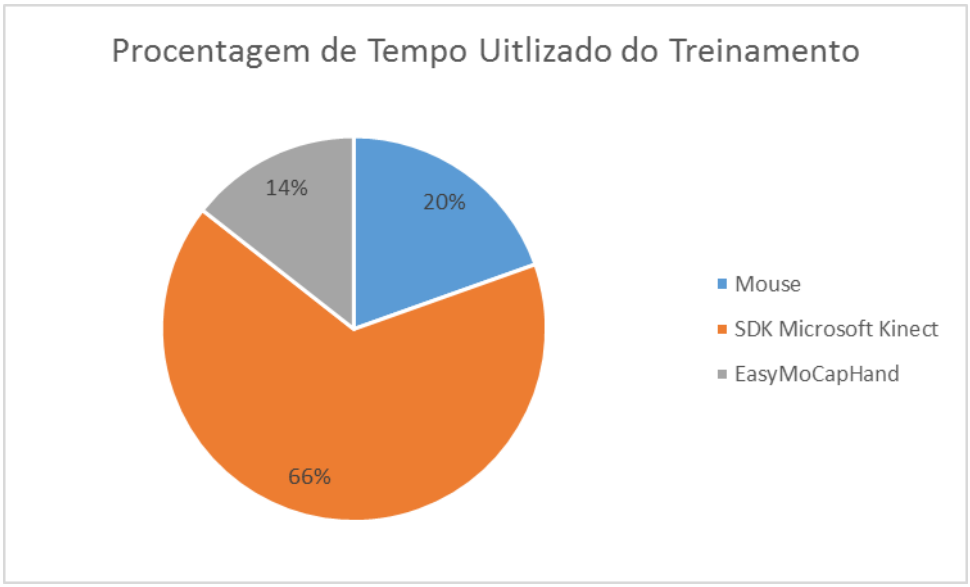

Figura 4.1: Porcentagem do tempo utilizado do treinamento

A necessidade do tempo exato do treinamento e quantidade de slides (tabela 4.2) se fez necessário, pois a diferença do tempo e tamanho dos Treinamentos do *framework* proposto *EasyMoCapHand* foi expressivamente menor quando comparado com a SDK Microsoft Kinect, e muito semelhante quando comparado a interação por meio do mouse, que pode demonstrar indícios mesmo que ainda

superficiais, de uma maior proximidade em escala de dificuldade do *framework* proposto para com a forma de interação através do mouse.

Tabela 4.2 – Quantidade de slides

Tópico	Quantidade Slides
Mouse	8
SDK MicrosoftKinect	24
EasyMoCapHand	7

Ao final do treinamento foi entregue para cada participante, material de apoio (Anexos F, G e H). Para ser utilizado como auxilio durante o desenvolvimento das atividades propostas. Pode ser observado através da tabela 4.3 que o material de apoio necessário para a *SDK Microsoft Kinect* tem o dobro da quantidade de páginas do que as outras duas abordagens.

Tabela 4.3 – Quantidade de páginas treinamento

Partes	Quantidade de páginas
Mouse	2
SDK MicrosoftKinect	4
EasyMoCapHand	2

Em uma primeira análise dos treinamentos já e possível observar indícios que a curva de aprendizagem para esse *framework*, projeta-se menor quando comparada a *SDK Microsoft Kinect.*

4.1.2 Desenvolvimento

Para o desenvolvimento foram utilizados dois equipamentos, um computador (processador Intel I5, 12GB de memória RAM) e um notebook (processador I5, 16 GB de RAM e placa de vídeo de 4GB), estes equipamentos foram utilizados por serem os melhores equipamentos disponíveis, para que a qualidade do equipamento não interferir no desempenho dos estudos de casos, ambos equipamentos foram previamente configurados e testados.

Foram instalados em ambos equipamentos todos recursos necessários para o funcionamento como visto no capitulo 3. Cada equipamento tinha a sua disposição um Kinect v2 com adaptador pronto para o uso.

Todas as atividades propostas foram separadas em pastas, das quais, continham os projetos com recursos referentes a cada exercício devidamente configurados e organizados para o proposto. Cada qual contendo um script para cada abordagem a ser desenvolvida, dentro das subpastas *Assets/Script FrameWorkScript.cs* para a abordagem *EasyMocapHand, MouseScript.cs* para interação com o mouse e *SDKScript.cs* para o uso da *SDK Microsoft Kinect*, como pode ser observado na 4.2.

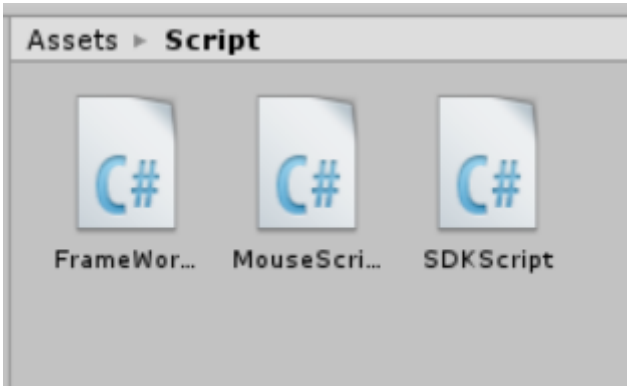

Figura 4.2: Scripts para cada abordagem.

Cada grupo deveria apenas desenvolver os scripts e testa-los afim de verificar o sucesso ou não de sua codificação.

Para todas as atividades, o cursor será representado por apenas uma sendo a mão aberta de cor verde (Figura 4.3), pelo motivo de o experimento não ter a necessidade da animação enquanto a ação pegar não for realizada. Importante ressaltar que todas as atividades serão recomeçadas do início, sem a possibilidade de copiar e colar códigos das atividades antecessoras.

Figura 4.3: Imagem utilizada como cursor nos experimentos.

4.1.2.1 Primeira Atividade

Esta atividade tem como objetivo que cada grupo consiga movimentar o cursor (Figura 4.3) pela cena do jogo, utilizando as três abordagens propostas.

Ambos os grupos conseguiram desenvolver esta atividade, com as três opções de abordagens propostas, o tempo gasto por cada grupo pode ser visualizado através da tabela 4.4.

Tabela 4.4 – Tempo gasto em minutos para realizar a atividade 1 por cada grupo.

Abordagem	Grupo 1	Grupo 2
SDK Microsoft	27 min.	46 min.
Interação Mouse	27 min.	5 min.
EasyMocap Hand	8 min.	3 min.

Pode ser observado através da tabela 4.4, que os tempos de desenvolvimento entre os grupos foram diferentes, porém, é valido ressaltar que o grupo 1 teve problemas com a *Unity* enquanto desenvolvia o movimento do cursor através da interação utilizando o mouse, sendo relatado pelos os participantes que após o problema resolvido, seria necessário um tempo de desenvolvimento muito inferior para o término com êxito das atividades. O tempo total para o desenvolvimento nesta atividade pode ser observado na tabela 4.5.

Tabela 4.5 – Tempo total gasto em minutos para realizar a atividade utilizando todas as abordagens.

Abordagem	Grupo 1	Grupo 2
Total	62 min	54 min

É possível analisar através do gráfico da figura 4.4 que o grupo 1 ocupou apenas 13% do tempo total desenvolvendo a atividade com o *framework* proposto e 44% com a *SDK Microsoft Kinect*, demonstrando mais indícios que o uso do *framework* cumpre o seu objetivo.

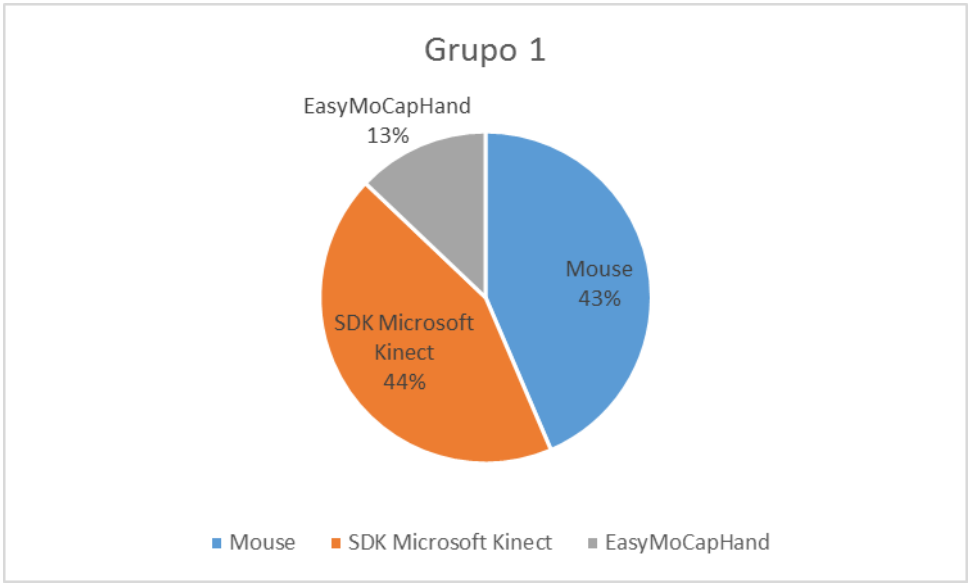

Figura 4.4: Gráfico de porcentagem do tempo utilizado do treinamento por cada abordagem – Grupo 1.

É possível analisar através do gráfico na figura 4.5 que o grupo 2 ocupou ainda menos tempo, ocupando apenas 6% do tempo total gasto nesta atividade para o desenvolvimento com o *framework* proposto. Este grupo demonstrou uma maior dificuldade no desenvolvimento utilizando a SDK, como demonstrado no gráfico, o grupo ocupou 85% do tempo, demonstrando que há uma grande diferença de tempo entre as duas abordagens.

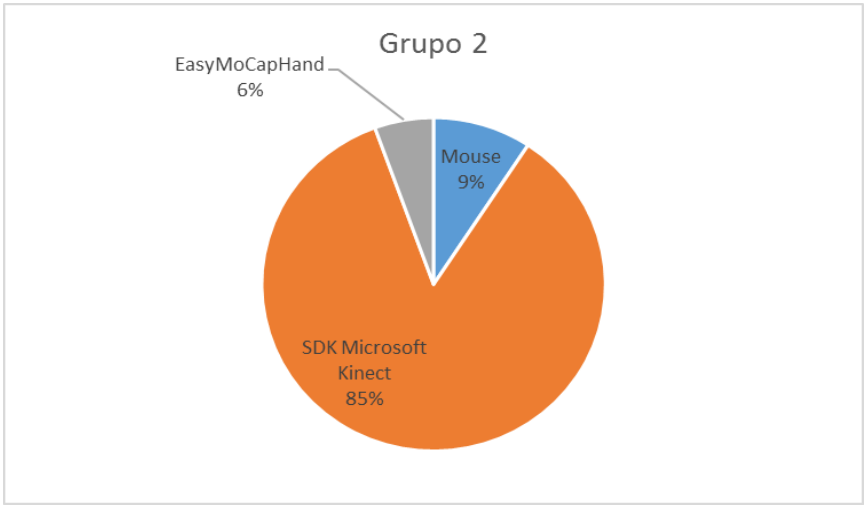

Figura 4.5: Gráfico de porcentagem do tempo utilizado do treinamento por cada abordagem – Grupo 2.

A primeira Atividade no geral demonstrou que a SDK Microsoft Kinect requer um tempo muito elevado para seu desenvolvimento comparado com o *framework* proposto por este trabalho e que em ambos os grupos, o desenvolvimento com o *framework* requer um tempo menor. Este tempo poderia ser considerado similar ao tempo gasto pela abordagem utilizando o mouse como forma de interação

4.1.2.2 Segunda Atividade

Esta atividade tem como objetivo, pegar um objeto/bloco com a letra A (Figura 4.6), arrasta-lo de um canto para o outro da cena.

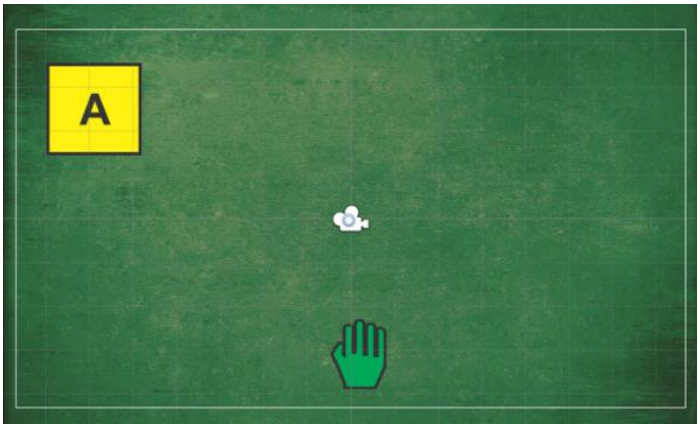

Figura 4.6: Cena em que o Bloco A deve ser arrastado.

A ação de pegar com a SDK e o *EasyMocapHand* será considerado através da mão estar fechada, ou seja, caso o cursor passe por cima do objeto e a mão do ator estiver aberta não acontecerá nenhuma ação, somente e quando a mão do ator estiver fechada o evento de pegar será válido. Para a interação através do mouse a ação de pegar se dará através do click. Todas as abordagens deverão arrastar o objeto tendo como posição inicial a do canto superior esquerdo como visto na Figura 4.6, e para posição final, o canto superior direito, como representado na Figura 4.7.

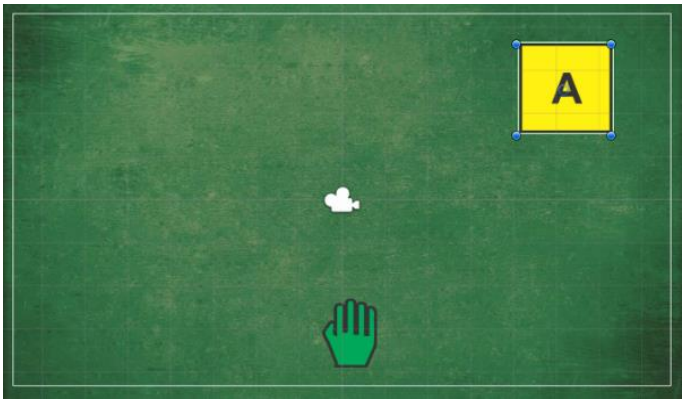

Figura 4.7: Cena em que o Bloco A deve ser solto.

Como pode ser observado na tabela 4.6, os tempos entre os grupos demonstraram certas semelhanças e diferenças relevantes. É notável perceber, que o grupo 1 teve uma maior dificuldade ao desenvolver a atividade com o uso do *framework* proposto por este trabalho, comparado com o tempo gasto pelo grupo 2, ocupando um tempo três vezes maior que o tempo do grupo 2. Em contrapartida, o grupo 2 ocupou quase três vezes mais tempo que o grupo 1 na abordagem através do mouse.

Tabela 4.6 – Tempo gasto em minutos para realizar a atividade 2 por cada grupo.

Método	Grupo 1	Grupo 2
SDK Microsoft	23	30
Interação Mouse	4	11
EasyMocap Hand	9	3

Ambos os grupos ocuparam um tempo muito maior para a abordagem com a *SDK Microsoft Kinect* que a soma das outras duas abordagens. Pode-se analisar através do gráfico da figura 4.8, que o grupo 1 através da abordagem com o *framework*, gastou um tempo menor comparado a SDK e um maior tempo comparado ao uso do Mouse.

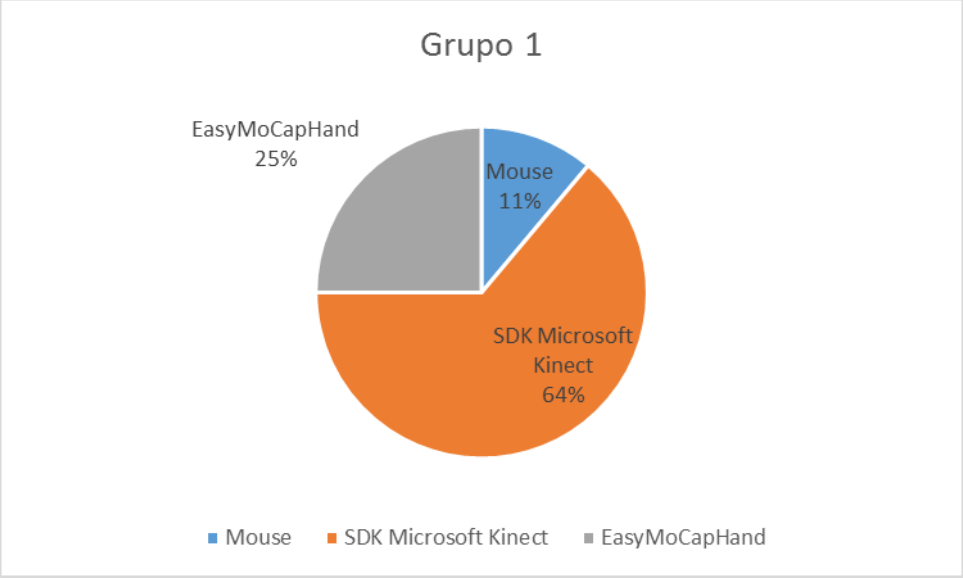

Figura 4.8: Porcentagem do tempo gasto em cada abordagem – Grupo 1.

Através do gráfico da figura 4.9, podemos observar que, diferente do grupo 1, o grupo 2 continua com o *framework* sendo a abordagem que gastou menos tempo total para a conclusão da atividade.

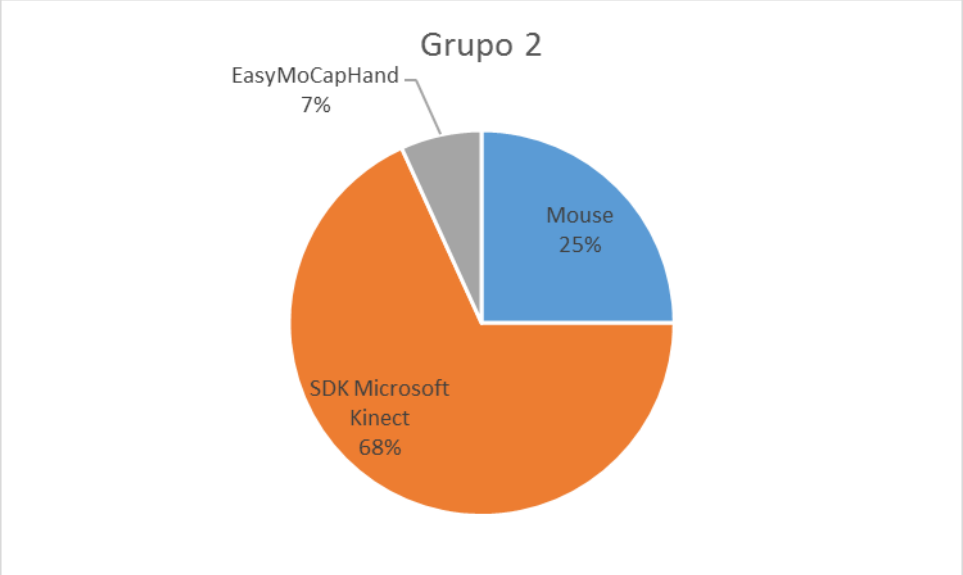

Figura 4.9: Porcentagem do tempo gasto em cada abordagem – Grupo 2.

De modo geral, a segunda atividade demonstrou que depois de realizarem a primeira atividade e se familiarizarem com a SDK, ambos os grupos conseguiram diminuir o tempo para o desenvolvimento, mas, ainda em ambos os casos, o *framework* ocupou um menor tempo em relação a SDK. Porém, apenas o grupo 2 utilizou um tempo menor para o desenvolvimento comparado ao uso do mouse.

4.1.2.3 Terceira Atividade

Nesta terceira atividade como na segunda etapa do estudo de caso relatado no capítulo 3 deste trabalho, será utilizado apenas duas abordagens: *SDK Microsoft Kinect* e *EasyMoCapHand*, cujo o foco desta etapa é ação de rotação tanto do cursor quanto a do objeto a ser colidido com o cursor. Logo, a interação através do mouse não será desenvolvida nesta atividade.

Os grupos deverão deslocar o objeto com o cursor do local onde ele se encontra inicialmente na cena do jogo, para o canto superior direito, com o objeto com rotação de 90 graus a direita, como observado através da figura 4.10.

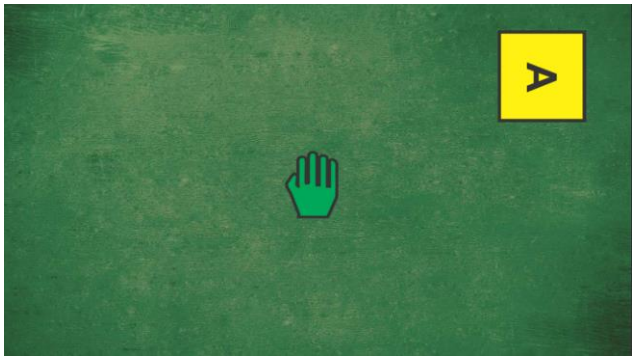

Figura 4.10: Exemplo de cena com o objeto em 90 graus de rotação a direita.

A Tabela 4.7, demonstra que o grupo 2 com a evolução de uma atividade para outra, conseguiu assimilar a *SDK Microsoft Kinect*, pela evidente continuidade em diminuir o tempo do primeiro para esta atividade. O grupo 1 por sua vez, não conseguiu realizar a atividade, por não conseguir identificar erros nos códigos, erros que tinham relação com a rotação e o uso da classe *Quaternion*.

Tabela 4.7 – Tempo gasto em minutos para realizar a terceira atividade por cada grupo.

Método	Grupo 1	Grupo 2
SDK Microsoft	54*	18
EasyMocap Hand	6	4
*Grupo 1 – não conseguiu completar aatividade.		

É possível perceber através da figura 4.11 que apesar do grupo 2 ter evoluído na utilização da SDK, ainda o seu desenvolvimento ocupou 82% do tempo gasto para resolução da atividade.

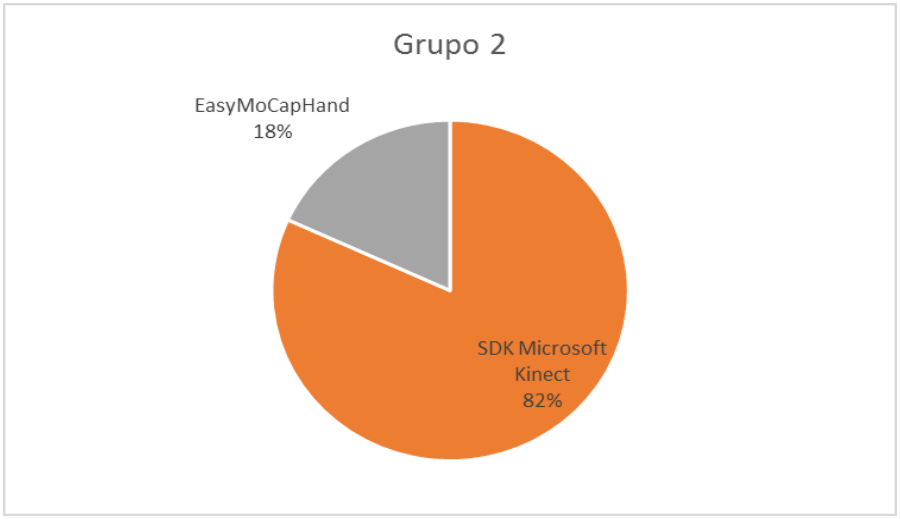

Figura 4.11: Porcentagem do tempo utilizado no desenvolvimento de cada abordagem – Grupo 2.

O fato do grupo 1 não ter resolvido a atividade proposta, mesmo depois de 54 minutos, e ter conseguido resolver a mesma atividade com um tempo de 6 minutos, demonstra que o *framework* facilitou ao menos a este grupo, com o uso de captura de movimentos.

4.1.3 Coleta de Dados

Os questionários foram desenvolvidos utilizando a *Likert Scale* (LIKERT, 1932) (utilizado como escalas: Não Concordo Totalmente, Não Concordo Parcialmente, Indiferente, Concordo Parcialmente e Concordo Totalmente, bem como uma adaptação para com valores referenciados, como: Nenhum, básico, Regular, Intermediário e Avançado.

Para a coleta de dados foram utilizados dois questionários, uma para a coleta de informações de grupo e outro para informação individuais. O questionário de grupo tinha como foco o desenvolvimento e a necessidade dos materiais e treinamento para a execução de cada atividade. Já o questionário individual, tinha a finalidade de oferecer a percepção de cada participante sobre alguns aspectos de cada abordagem e uma auto avalição dos conhecimentos envolvidos no experimento.

4.1.3.1 Questionário em Grupo

Com os dados tabulados de cada atividade, podemos analisar a percepção de cada grupo sobre o material e treinamento aplicado.

A tabela 4.8 mostra a percepção do grupo 1 em relação ao desenvolvimento com o material e treinamento aplicado, demonstrando que em todas as atividades e abordagens, o material foi essencial para o desenvolvimento.

Quanto ao treinamento, quando indicado como único recurso e se o mesmo seria suficiente para o desenvolvimento da atividade, o grupo 1 respondeu que não concordam com esta afirmação para a abordagem SDK Microsoft Kinect, em contrapartida o *framework* proposto saiu com a mesma percepção da interação através do Mouse: concordo parcialmente. O treinamento foi apontado como não sendo essencial para o *framework* demonstrando que apenas uma leitura em seu

material de apoio seria o suficiente, enquanto as outras abordagens precisariam deste recurso.

Tabela 4.8 – Respostas Questionário Grupo 1

Treinamento - Foi Essencial para o desenvolvimento			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Concordo parcialmente	Concordo parcialmente	Concordo parcialmente
Mouse	Concordo parcialmente	Concordo parcialmente	----------
Easy Mocap Hand	Não Concordo Totalmente	Não Concordo Totalmente	Concordo parcialmente
Material de apoio - Foi Essencial para o desenvolvimento			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Concordo totalmente	Concordo totalmente	Concordo totalmente
Mouse	Concordo totalmente	Concordo totalmente	----------
Easy Mocap Hand	Concordo totalmente	Concordo totalmente	Concordo totalmente
Apenas com o treinamento seria possível desenvolver a Atividade			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Não Concordo Totalmente	Não Concordo Totalmente	Não Concordo Totalmente
Mouse	Concordo parcialmente	Concordo parcialmente	----------
Easy Mocap Hand	Concordo parcialmente	Concordo parcialmente	Concordo parcialmente
Apenas com o material de apoio seria possível desenvolver a Atividade			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Concordo totalmente	Concordo totalmente	Concordo totalmente
Mouse	Concordo totalmente	Concordo totalmente	----------
Easy Mocap Hand	Concordo totalmente	Concordo totalmente	Concordo totalmente

O grupo 2, respondendo as mesmas percepções para SDK Microsoft Kinect quanto para o *framework* proposto, mostrando que o treinamento e material de apoio surtiram o mesmo efeito para ambas abordagens, apenas a abordagem de interação através do mouse mostrou alguns pontos diferenciais. (Tabela 4.9)

Tabela 4.9 – Respostas Questionário Grupo 2

Treinamento - Foi Essencial para o desenvolvimento			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Não concordo parcialmente	Não concordo parcialmente	Não Concordo Totalmente
Mouse	Indiferente	Indiferente	----------
Easy Mocap Hand	Não concordo parcialmente	Não concordo parcialmente	Não Concordo Totalmente
Material de apoio - Foi Essencial para o desenvolvimento			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Concordo totalmente	Concordo totalmente	Concordo totalmente
Mouse	Concordo parcialmente	Concordo parcialmente	----------
Easy Mocap Hand	Concordo totalmente	Concordo totalmente	Concordo totalmente
Apenas com o treinamento seria possível desenvolver a Atividade			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Não Concordo Totalmente	Não Concordo Totalmente	Não Concordo Totalmente
Mouse	Indiferente	Indiferente	----------
Easy Mocap Hand	Não Concordo Parcialmente	Não Concordo Totalmente	Não Concordo Totalmente
Apenas com o material de apoio seria possível desenvolver a Atividade			
Abordagem	Atividade 1	Atividade 2	Atividade 3
SDK Microsoft Kinect	Concordo totalmente	Concordo totalmente	Concordo parcialmente
Mouse	Concordo totalmente	Concordo totalmente	----------
Easy Mocap Hand	Concordo totalmente	Concordo totalmente	Concordo parcialmente

4.1.3.2 Questionário Individual

Observando os dois gráficos da figura 4.12 é possível visualizar que em ambos os grupos a auto avaliação se mostrou homogênea em relação ao conhecimento sobre o paradigma de programação Orientada a Objetos e Linguagem de programação c#, ficando em sua maioria entre regular e intermediário.

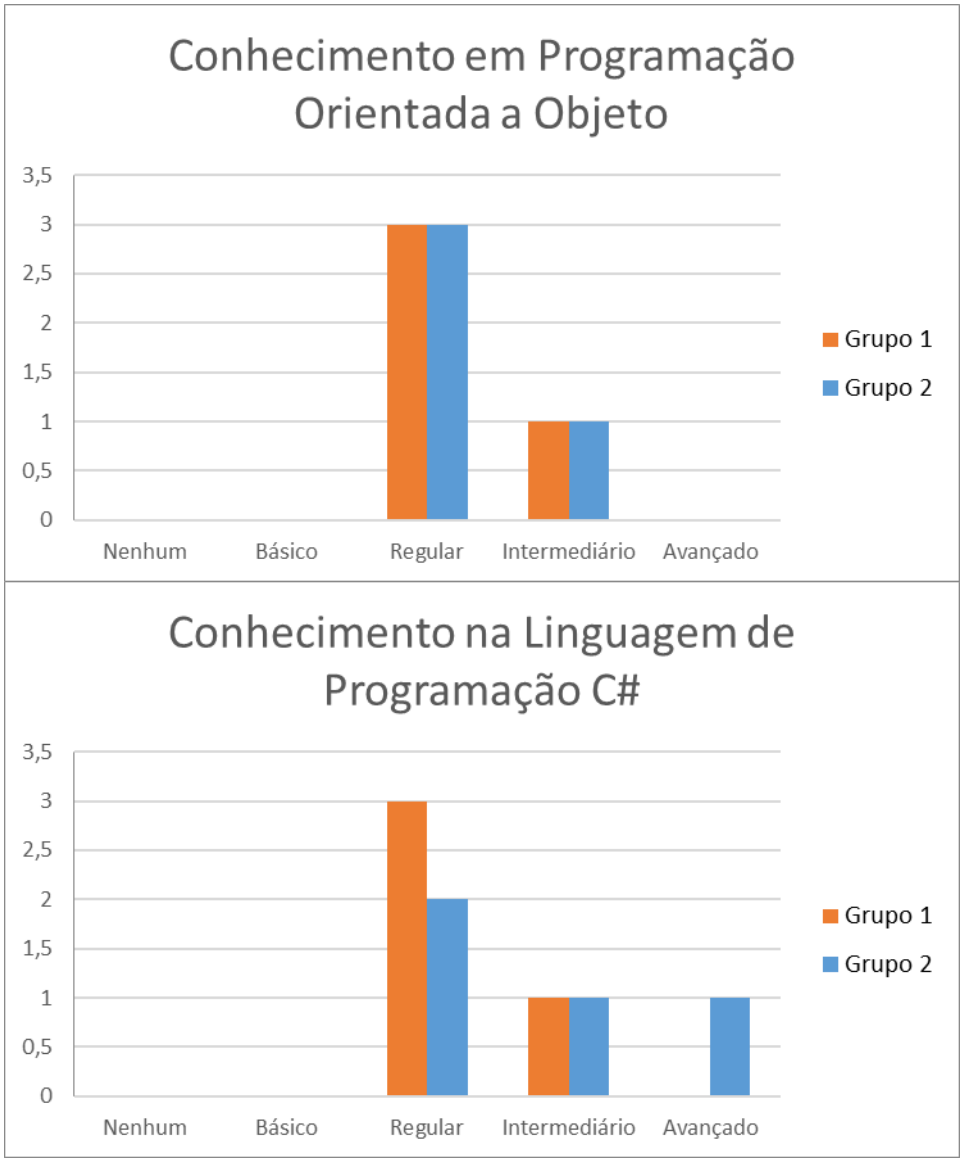

Figura 4.12: Gráfico de levantamento de conhecimento sobre C# e Programação Orientada a Objeto.

Ao analisar os gráficos na figura 4.13, em ambos os grupos os conhecimentos em *Unity* e captura de movimentos, em uma auto avaliação, é considerada no geral básica, pois cada um dos participantes mantiveram suas respostas entre nenhum a regular.

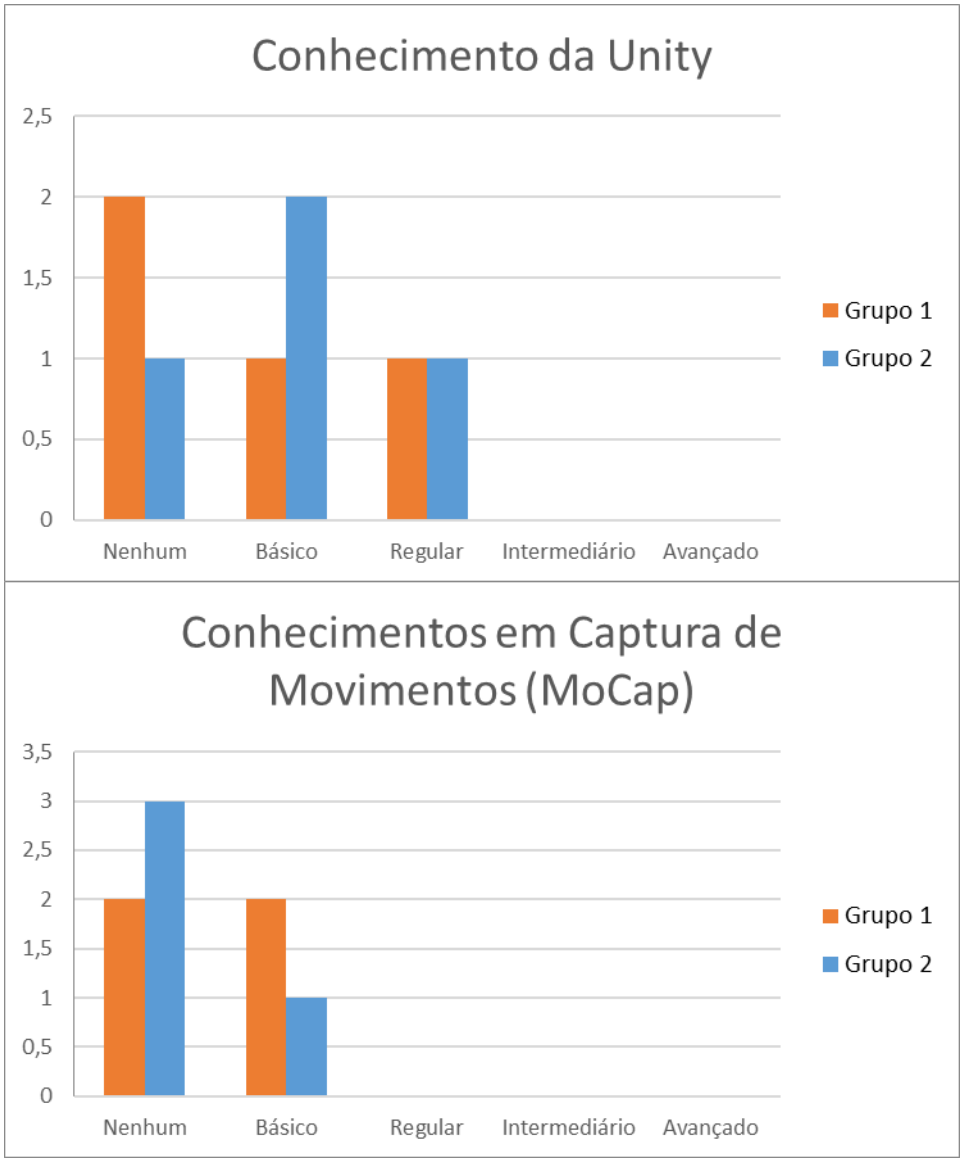

Figura 4.13: Gráfico de levantamento de conhecimento sobre *Unity* e Captura de Movimentos.

Para começar a análise da percepção sobre cada abordagem, as respostas para as perguntas 9 referente a *SDK Microsoft Kinect* e 19 referente o *framework* proposto *EasyMoCapHand* do questionário individual, que se trata de: “É necessário que o nível de conhecimento em captura de movimentos seja avançado para um bom desempenho no desenvolvimento? ”. Na figura 4.14, o gráfico demonstra através da percepção dos participantes a questão, é que se somado as escalas Concordo Parcialmente e Totalmente, alcança um índice de 75% dos participantes, dizendo ser necessário um conhecimento avançado sobre captura de movimentos quando utilizado a *SDK Microsoft Kinect*.

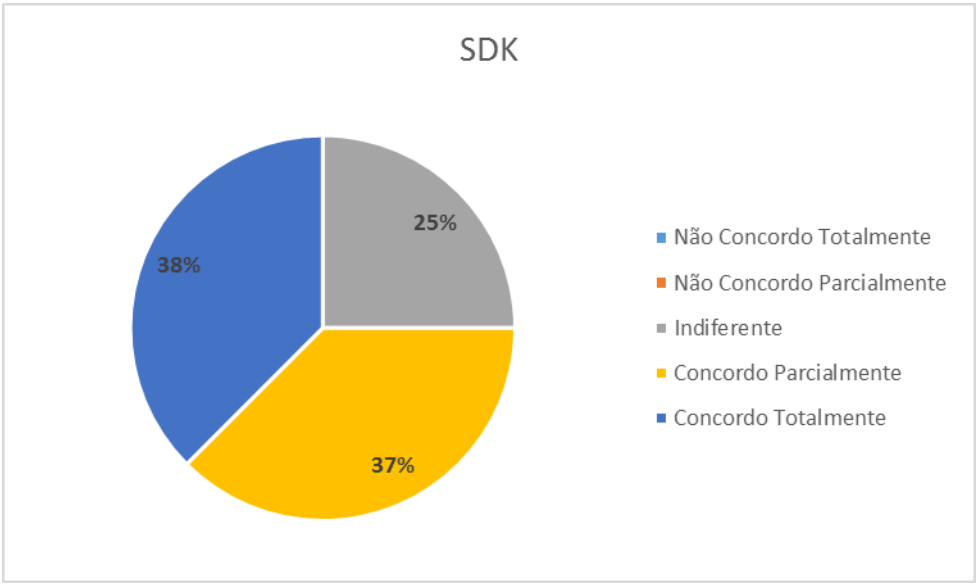

Figura 4.14: Gráficos com porcentagens referentes a questão 9 do questionário Individual.

A partir da mesma análise feita para o *framework EasyMocapHand,* o gráfico da figura 4.14, demonstra basicamente o oposto, sendo a percepção dos participantes que o *framework* proposto não necessita de conhecimentos avançados de captura de movimentos para o desenvolvimento das atividades

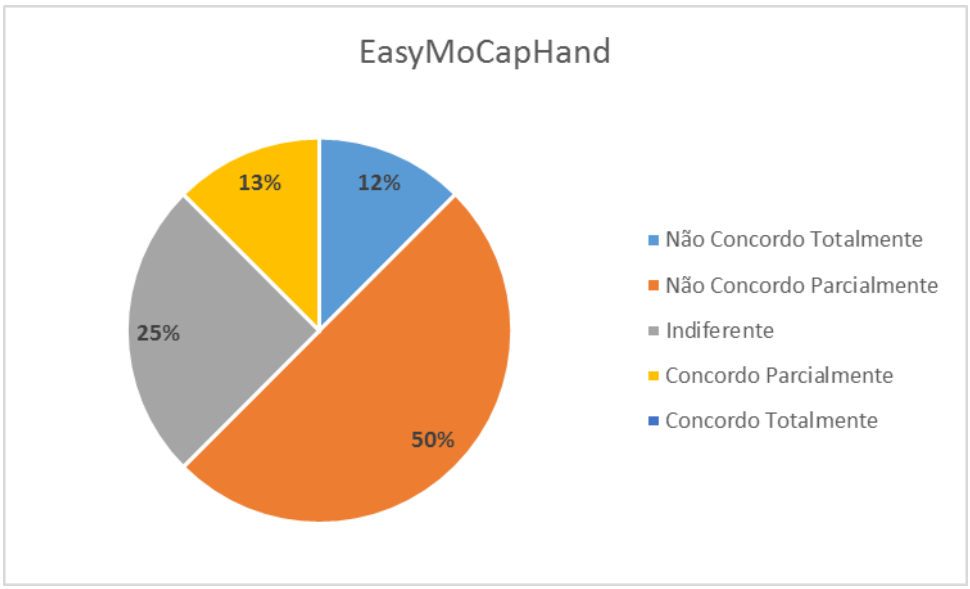

Figura 4.15: Gráficos com porcentagens referentes a questão 19 do questionário Individual.

Outro aspecto pode ser observado em um outro ponto do questionário realizado sobre todas as abordagens (pergunta 10, 15 e 20): “De forma Geral a utilização e desenvolvimento pode ser considerado de nível avançado? ”. A figura 4.16 traz três gráficos, que demonstram o *framework* quanto a sua utilização, de forma geral se assemelha muito com o nível da abordagem utilizando mouse, sendo

quase idênticas as respostas dos participantes, e também demonstra o distanciamento do nível avançado que a percepção dos participantes indicam, como necessário para o uso da *SDK Microsoft Kinect*.

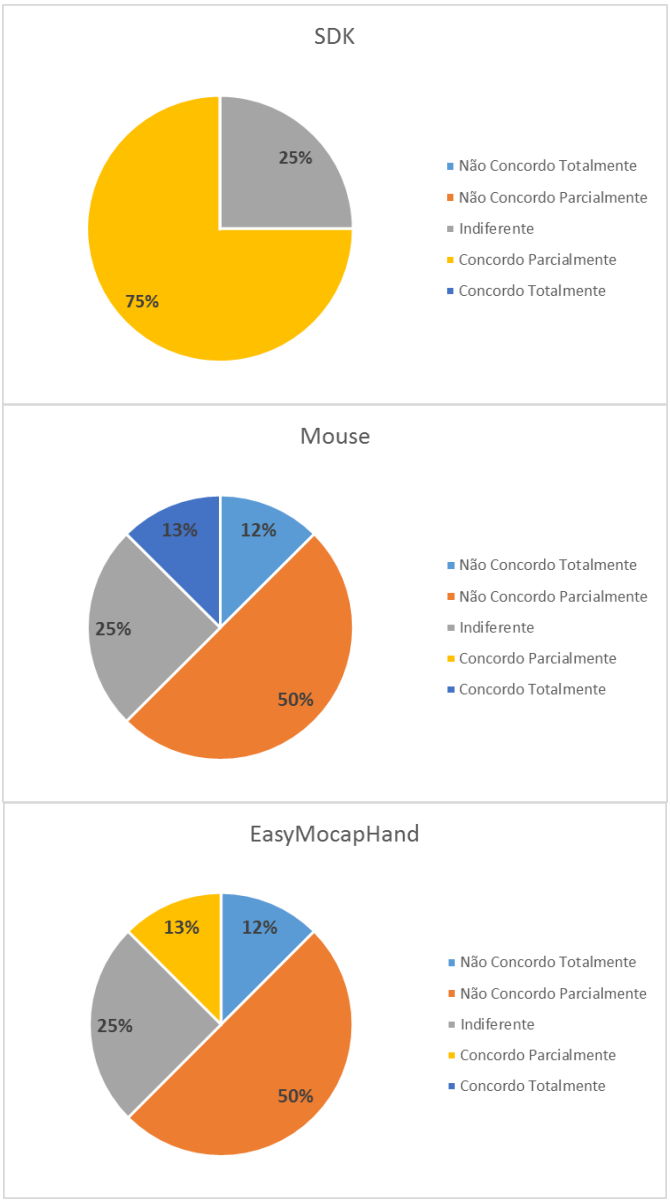

Figura 4.16: Gráfico sobre respostas das questões 10, 15 e 20 do Questionário Individual

A pergunta 21 traz apenas a percepção sobre o *framework* proposto, questionando: “a utilização deste *framework* facilita a utilização da captura de movimentos? ”, através da figura 4.17 o gráfico demonstra que a percepção dos participantes é positiva.

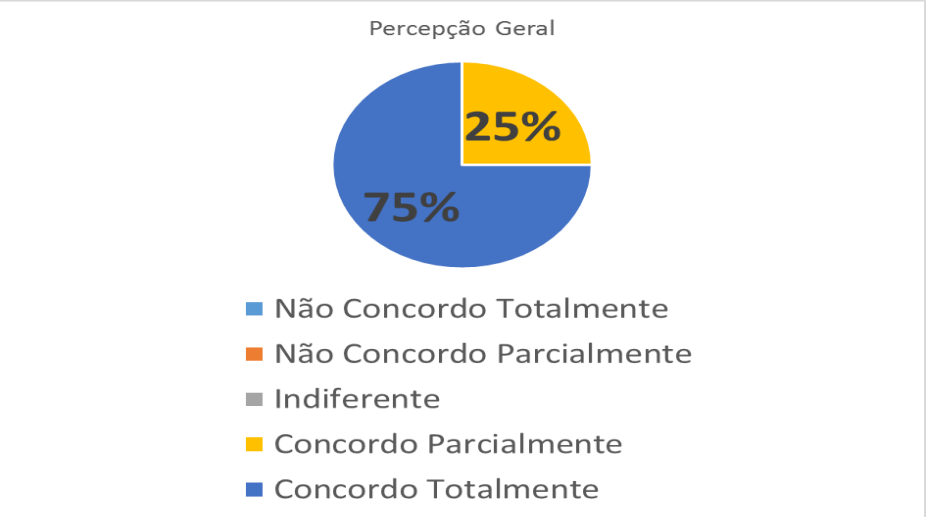

Figura 4.17: Gráfico de percepção sobre a facilidade do uso do *EasyMoCapHand*.

Uma pergunta de resposta opcional foi adicionada ao questionário: “A partir deste experimento, por favor, indique o que considerou pontos fortes e fracos do *framework* proposto”. Sendo citado alguns como:

Pontos Fortes:

- “Simplifica a captura de movimentos na programação”
- “Códigos mais simples”
- “Agiliza o processo de Desenvolvimento”
- “Grande Facilidade em utiliza-lo”

Pontos Fracos

- “Passagens de parâmetro um pouco confusa”.
- “Provável perda de Desempenho”.

Os pontos fortes atingem o objetivo deste trabalho, oferecendo um *framework* que facilite o uso de interação através de captura de movimentos. Em relação aos pontos fracos, é justo ressaltar que a percepção dos participantes é real, que sim, perde desempenho por haver um *framework*, ou seja, há uma camada a mais, porém a intenção do *framework* é utiliza-lo para aplicações simples sem a necessidade de precisão e desempenho elevados, então, essa perda de desempenho possa ser justificada. Sobre a passagem de parâmetros, após uma análise poderia ser atendida em trabalhos futuros.

4.2 Considerações Finais

De forma geral, como pode ser visualizado ao longo deste capitulo, o desenvolvimento das atividades de ambos os grupos tem em média 71% do tempo ocupado quando utilizada abordagem *SDK Microsoft Kinect*, enquanto o *framework* proposto ocupou 13% do tempo total. Isso apresenta evidências que o *framework* proposto requer menos tempo de treinamento e desenvolvimento, ainda contando com um material de apoio mais enxuto como demonstrado no decorrer de todo o experimento. O *framework* comparado a abordagem que utiliza a interação através do mouse, mostrou também resultados significativos, já que nas atividades em que foram aplicadas esta abordagem e treinamento ele ficou com média de 21% de tempo gasto, ou seja, mesmo essa abordagem ao final do experimento, ocupou em média mais tempo do que o *framework EasyMoCapHand.*

Ao final de acordo com o gráfico da figura 4.17, indica que 75% dos participantes concordam totalmente que o *framework* facilita o desenvolvimento e os outros 25%, concordando parcialmente, e ainda quando solicitado uma percepção de modo geral foi respondido que o *framework* proposto se assemelha em níveis de dificuldade de desenvolvimento a abordagem que utiliza o mouse como meio de interação. Demonstra que o uso da *SDK Microsoft Kinect* neste experimento é de nível avançado, enquanto o do *framework* proposto *EasyMoCapHand* se encontra em nível oposto, o que deixa claro as respostas para as duas questões levantadas no início deste capítulo: *EasyMoCapHand* pode facilitar o desenvolvimento de aplicações com o recurso de captura de movimentos das mãos?; É possível desenvolver a interação através da captura de movimentos das mãos em uma dificuldade que se assemelhe com a interação através do mouse?, são definitivamente respondidas positivamente.

Pelo experimento nota-se que a abstração que o *framework* fornece claramente uma maior facilidade que a SDK, através de evidências obtidas através do tempo desenvolvimento, treinamento e opinião dos participantes do experimento, fornecendo um desenvolvimento de interação através de captura de movimento, tão simples e prática, quando comparado a interação mais convencional através da utilização do mouse. Evidenciando que a interação via captura de movimentos, pode

ser utilizada por desenvolvedores iniciantes, oferecendo assim mais opções a este grupo de pessoas.

Ao final pode ser notado que as perguntas idealizadas foram respondidas ambas positivamente durante este capítulo, demonstrando que o *framework* pode facilitar o desenvolvimento de aplicações com o recurso de captura de movimentos, e a interação e dificuldade encontrada é semelhante ao desenvolvimento quanto ao uso de interação através do mouse.

Capítulo 5

CONCLUSÃO

Foi apresentado no decorrer do trabalho, um *framework* que tem o objetivo de facilitar o uso de captura de movimento da mão em aplicações simples, sendo utilizado como meta fornecer um *framework* que disponibilize captura de movimentos da mão com a mesma ou similar dificuldade que a abordagem através da interação com o mouse possui.

Utilizando como base a *SDK da Microsoft Kinect,* foi possível oferecer tal *framework*. Na Seção 5.1, será possível verificar as contribuições obtidas nas atividades realizadas e apresentar limitações identificadas no trabalho através da Seção 5.2. Algumas sugestões de trabalhos futuros para a continuidade desta dissertação foram descritas na Seção 5.3.

5.1 Contribuições

Através do estudo de caso e experimentos, é possível determinar que o principal objetivo do trabalho vem claro no título, que é facilitar o uso da captura de movimentos da mão como interação, e sim, foi atendida no decorrer deste trabalho.

No capítulo 3, foi apresentado evidências através dos exemplos de codificação que a quantidade de linhas programadas, objetos instanciados e chamadas de métodos é mais fácil codificar o *framework EasyMoCapHand,* quando comparado a SDK Microsoft Kinect, e que sua codificação fica muito semelhante em níveis de dificuldade com abordagem que utiliza a interação através do mouse.

Através de um experimento demonstrado no capitulo 4, as evidências sugerem que o tempo ao se desenvolver uma aplicação utilizando o *framework EasyMoCapHand* é menor que quanto utilizado SDK Microsoft Kinect, e que necessita de uma curva de aprendizagem menor, já que o tempo de treinamento e a quantidade de material de apoio é menor. E quando a comparação é com a abordagem através de interação com o mouse, o *framework* se mostrou em nível de igualdade, em tempo de desenvolvimento quanto de dificuldade.

Finalizando através de questionário em forma de entrevista, foi possível conversar com dois professores que lecionam disciplinas voltadas a desenvolvimento orientado a objetos. Antes da entrevista foi realizado o treinamento no mesmo formato ao realizado com os participantes do experimento, e foram três perguntas para que cada professor transcrevesse em um documento sobre a percepção deles do *framework*.

1) Você acredita que o *framework* proposto tornará acessível o desenvolvimento de aplicações através dos movimentos capturados pelas mãos?

   Professor A: “Sim, em nosso curso técnico os alunos começam a trabalhar com desenvolvimento orientado a objetos apenas no segundo de três módulos semestrais, com o tempo curto e por nosso público ser de escolaridade de nível médio, esse *framework* faz ser possível aplicar a captura de movimentos em projetos de forma bem tranquilo, o que acredito que não aconteça utilizando a SDK”

   Professor B: “Sim, sem maior dificuldade, pois os conceitos para o desenvolvimento são iniciais, tanto em relação a linguagem e ferramentas. ”

2) Sem o treinamento e material de apoio aos alunos chegariam a desenvolver estas atividades com a SDK Microsoft Kinect?

   Professor A: “Não acredito, apenas pode acontecer caso apareça um aluno diferenciado que realmente me surpreenda. ”

   Professor B: “Com certeza não, a maior parte os alunos sentem dificuldades de desenvolver simples atividades com conceitos de orientação a objetos”.

3) Comparado ao mouse, o *framework* proposto é possível afirmar que os desenvolvimentos sejam similarmente fáceis.

Professor A: “Sim, porém ainda acredito que o *framework* gere um pouco mais de dificuldade, não tanto no desenvolvimento, mas conectar o dispositivo, testar. ”

Professor B: “Acredito ainda que a sua proposta, seja um pouco mais difícil, mas nada comparado a SDK.”.

Apoiando por fim todas contribuições alcançadas por este trabalho.

5.2 Limitações

Foram identificadas algumas principais limitações deste trabalho podendo ser listados em:

- Quantidade de dispositivos – para este estudo os dispositivos *Kinect* disponíveis foram apenas 2, por este motivo o experimento ficou limitado a apenas dois grupos.
- Ações Limitadas da Captura – para o estudo foram escolhidas apenas duas ações para serem parte do *framework*, que foram: pegar e girar. Para ser possível limitar os estudos e ser possível aplicar experimentos e descobrir se é viável o desenvolvimento deste *framework*, já que no mercado já são oferecidas diversas opções como até mesmo a base do *framework* proposto a *SDK Microsoft Kinect*, que já oferece recursos para a captura.
- Ambiente 2D – do mesmo jeito que as ações para este trabalho, a escolha do ambiente 2D foi utilizado para restringir o estudo e conseguir deixar de forma mais clara se o *framework* proposto atinge o objetivo de facilitar o uso da captura de movimentos.
- Disponibilidade apenas para Unity – Como a base deste *framework* é a *SDK da Microsoft* e a possibilidade de uso em trabalhos futuros em jogos ou aplicativos educacionais do que já utilizam a *Unity* e por esta *engine* ser a mais utilizada pela comunidade de desenvolvimento de jogos, oferecer recurso web, provou ser a melhor escolha.

- Desempenho inferior: por haver uma nova camada, o desempenho diminuir é inevitável, porém nada que atrapalhe e comprometa o foco, que serão aplicações mais simples.

5.3 Trabalhos Futuros

Para trabalhos futuros, algumas propostas já estão em estudo, são elas:

- Uso do *framework* no projeto LabTeca(OTSUKA et al., 2015) do laboratório LOA (Laboratório de Objetos de Aprendizagem): projeto de um *game* onde os jogadores poderão misturar sais e soluções químicas para alcançar novas soluções e para isso usar a captura de movimentos como o objetivo de deixar o movimento de pegar as vidrarias por exemplo mais intuitivo e chamativo através da captura de movimentos. Jogo disponível através do link: <http://www.loa.sead.ufscar.br/labteca.php>.
- Generalizar: desenvolver uma versão do *framework* sem a utilização dos plug-ins da *Unity*, tornando assim compatível com todas as opções conforme a sua base a *SDK Microsoft Kinect* dispõe*.*
- Novos experimentos: realizar novos experimentos com públicos com níveis de graduação e pós graduação, bem como em diversas universidades, buscando parcerias e tornando a validação mais ampla.

Também há opções vislumbradas que podem ser de grande valia para a continuação do *framework EasyMoCapHand*, que são:

- Implementação de novos recursos e ações: um exemplo seria empurrar, e mais opções como escorregar (cujo a mão pode não ter sido fechada o bastante e o objeto escorregado).
- Implementação do terceiro eixo, proporcionando a captura em 3D para alcançar principalmente jogos e aplicações que fazem o uso desta tecnologia.
- Melhoria de desempenho do movimento principalmente na ação girar.

- Melhor clareza quanto a passagens de parâmetros, deixando o uso do *framework EasyMoCapHand* mais intuitivo e compreensivo.

REFERÊNCIAS

ALEXANDERSON, S.; O'SULLIVAN, Carol.; BESKOW, J. Robust Online Motion Capture Labeling of Finger Markers. In: Proceeding MIG '16 Proceedings of the 9th International Conference on Motion in Games – MIG, California, USA, 2016. p. 7-13.

ANJO, M, S.; PIZZOLATO, E. B.; FEUERSTACK, S. A Real-Time System to Recognize Static Gestures of Brazilian Sign Language (Libras) alphabet using Kinect. In: Proceeding IHC '12 Proceedings of the 11th Brazilian Symposium on Human Factors in Computing Systems. Cuibá, MT, Brazil. 2012. p. 259-268.

BEDOYA, N. N.; GUERRERO, C. D. Q. Process Optimization based on Motion Capture for Videogames Assets. In: ACM SIGGRAPH Conferecen on Motion in Games – MIG, Los Angeles, USA, 2014. Proceedings of the Seventh International Conference on Motion in Games. ACM 2014. p. 186-186

BODENHEIMER, B; ROSE, C. 1997. The process of motion capture: dealing with the data. Disponível em: <http://www.vuse.vanderbilt.edu/~bobbyb/pubs/dealingdata97.pdf>. Acesso em: Maio, 2015.

C#. C# Language Specification 5.0. Disponível em: <https://www.microsoft.com/en-us/download/details.aspx?id=7029>. Acessado em: Junho de 2015.

DEVELOP NETWORK. Tracking Users with Kinect Skeletal Tracking. Disponível em: <https://msdn.microsoft.com/en-us/library/jj131025.aspx>. Acesso em: 28 de setembro de 2016

DYER, S; MARTIN, J; ZULAUF, J. Motion Capture White Paper. 1995. Disponível em: <ftp://ftp.sgi.com/sgi/A%7CW/jam/mocap/MoCapWP_v2.0.html>. Acesso em: Abril, 2015.

FINE, R. Unity 3D Mono Behaviour lifecycle. 2011. Disponível em: <http://www.richardfine.co.uk/2012/10/unity3d-monobehaviour-lifecycle/>. Acesso em: Dezembro, 2016.

FLAM, D, L. Openmocap: uma aplicação de código livre para a captura óptica de movimento (Mestrado em Ciência da Computação). Instituto de Ciências Exatas - Universidade Federal de Minas Gerais. 2009.

FRANCESE, R; PASSERO, I; TORTORA, G. Wiimote and Kinect: Gestural User Interfaces add a Natural third dimension to HCI. In: Proceedings of the International Working Conference on Advanced Visual Interfaces – AVI'12, New York, USA, 2012. p. 116-123.

FURNISS, M. Motion Capture. 2015. Disponível em: <http://web.mit.edu/comm-forum/papers/furniss.html#fn32>. Acesso em: Abril, 2015.

GOMES, P. C. R; FERNANDES, L. A. F. 2003. Sistema óptico de captura do movimento humano 2D, Sem utilização de marcações especiais. III Congresso Brasileiro de Computação – CBComp 2003. Computação Gráfica

GOMIDE, J. V. B. 2006. Captura Digital de Movimento no Cinema de Animação. Dissertação de Mestrado, EBA/UFMG.

HANTRAKUL, L.; KACZMAREK, K.; Implementations of the Leap Motion device in sound synthesis and interactive live performance. In: MOCO '14 Proceedings of the 2014 International Workshop on Movement and Computing. Paris, France. 2014. p.142 - 145

KINECT. Kinect para Xbox One. Disponível em: <http://www.xbox.com/pt-BR/Xbox-One/accessories/kinect-for-xbox-one>. Acessado em: Agosto de 2015.

KINECT SDK. Kinect for Windows SDK 2.0. Disponível em: < https://www.microsoft.com/en-us/download/details.aspx?id=44561>. Acessado em: Agosto de 2015.

KINECT HARDWARES. Kinect hardware requirements and sensor setup. Disponível em: <https://dev.windows.com/en-us/kinect/hardware-setup>. Acessado em: Fevereiro de 2015.

KITAGAWA, M; WINDSOR, B. MoCap for Artists: Workflow and Techniques for Motion Capture. Burlington: Focal Press, 2008.

LEAP MOTION. Leap motion. Disponível em: <https://www.leapmotion.com/product/desktop>. Acessado em: setembro de 2015.

LIKERT,R.Atechniqueforthemeasurementofattitudes.ArchivesofPsychology,v.22,n.140, p. 1–55, 1932. Citado na página 63.

MENACHE, A. Understanding motion capture for computer animation. 2 ed. San Francisco: Morgan Kaufmann, 2011.

MOESLUND, T. B.; HILTON, A; KRÜGER, V. A survey of advances in vision-based human motion capture and analysis. COMPUTER VISION AND IMAGE UNDERSTANDING, Elsevier,2006

MUÑOZ, A. M.; GONZÁLEZ, M. L. G.; GALLARDO, J. C.; PRNG based on new HCI devices entropy sources. Wii Remote study case.In: EATIS '09 Proceedings of the 2009 Euro American Conference on Telematics and Information Systems: New Opportunities to increase Digital Citizenship. Prague, CZ. 2009. n. 5

MYO. Myo. Disponível em: <https://www.myo.com/>. Acessado em: Setembro de 2015.

OLHAR DIGITAL. Conheça o Mouse 3D!. Disponível em: < http://olhardigital.uol.com.br/video/conheca-o-mouse-3d/11144>. Acessado em: Setembro de 2016.

OLIVEIRA, S. W. Folhetim de Aprendizagem: Aula Expositiva. Disponível em: < http://www.nadp.ufla.br/2013/wp-content/uploads/2013/09/Folhetim-27-Aula-Expositiva.pdf>. Acessado em: novmebro de 2016.

OTSUKA, J. L. ; BORDINI, R. A. ; BEDER, D. M. ; CAMARGO, A. E. R. ; MENATO, T. ; BORGES, M. T. M. R. . LABTECA: Experiência Lúdica em um Laboratório 3D de Química. RENOTE. Revista Novas Tecnologias na Educação, v. 13, p. 1, 2015.

PANGER, G. Kinect in the Kitchen: Testing Depth Camera Interactions in Practical Home Environments. Proceeding CHI '12 Extended Abstracts on Human Factors in Computing Systems. Austin, Texas, USA. 2012.p. 1985-1990.

PEDERSOLI, F.; ADAMI, N.; BENINI, S.; LEONARDI, R. XKin - eXtendable Hand Pose and Gesture Recognition Library for Kinect. In: Proceeding MM '12 Proceedings of the 20th ACM international conference on Multimedia. MM'12, New York, USA, 2012. p. 1465-1468.

PENELLE, B.; DEBEIR, O.; Multi-Sensor Data Fusion for Hand Trackingusing Kinect and Leap Motion. In: Proceedings of the 2014 Virtual Reality International Conference. Laval, France, 2014. n. 22.

POTTER, E. L.; ARAULLO, J.; CARTER, L.; The Leap Motion controller: A view on sign language. In: OzCHI '13 Proceedings of the 25th Australian Computer-Human Interaction Conference: Augmentation, Application, Innovation, Collaboration. Adelaide, Australia. 2013. p. 175-178

RAAB, F. R.; BLOOD, E. B.; STEINER, T. O.; Jones, H. Magnetic position and orientation tracking system. IEEE Trans. Aero. Electro., vol. AES-15, pp. 709–718, 1979.

REN, Z; MENG, J.; YUAN, J.; ZHANG, Z. Robust Hand gesture recognition with Kinect senson. In: Proceedings of the 19th ACM international conference on Multimedia. Scottdale, Arizona, USA. 2011. p. 759-760.

SANTHANAM, N. Wii Remote as a Web Navigation Device For People with Cerebral Palsy. In: ASSETS '12 Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility. Boulder, Colorado, USA. 2012. p. 303-304.

SANTOS, S. S.; LAMOUNIER, E. A.; CARDOSO, A. Interação em Ambientes de Realidade Aumentada Utilizando o dipositivo Kinect. In: Proceedings of the 2011 XIII Symposium on Virtual Reality. Uberlandia, Minas Gerais, Brasil. p. 112-121

SATO, H.; COHEN, M. Using Motion Capturefor Real-time Augmented Reality Scenes. In: Human and Computers – HC, Fukushima-ken, JAPAN, 2010.

Proceedings of the 13th International Conference on Humans and Computers. Pages 58-62.

SILVA, F. W. S. V. Um sistema de animação baseado em movimento capturado. 1998. 101 f. Dissertação (Mestrado em Computação Gráfica) – Laboratório de Computação Gráfica COPPE/Sistemas, Universidade Federal do Rio de Janeiro, Rio de Janeiro.

STURMAN, D. J. A Brief History of Motion Capture for Computer Character Animation. In SIGGRAPH, Character Motion Systems, Course notes. 1994.

TANG, J. K. T.; CHAN, J. C. P.; LEUNG, H. Interactive Dancing Game with Real-time Recognition of Continuous Dance Moves from 3D Human Motion Capture. In: The 5th International Conference on Ubiquitous Information Management and Communication – ICUIMC '11, Seoul, Coréia do SUL, 2011.

UNITY. Unity. Disponível em: < https://unity3d.com/pt>. Acessado em: Fevereiro de 2015.

UNITY DOC. Unity Documentation. Disponível em: < https://docs.unity3d.com/ScriptReference/Quaternion.html>. Acessado em: Agosto de 2016

VISUAL STUDIO. Visual Studio. Disponível em: < https://www.visualstudio.com/>. Acessado em: Março de 2015.

WANG, Y., MIN, J., ZHANG, J., LIU, Y., XU, F., DAI, Q., CHAI, J. Video-based Hand Manipulation Capture Through Composite Motion Control. ACM Trans. Graph. 32, 4, Article 43 (July 2013), 13 pages.

WEI, T. Detecting the Hand-Mouthing Behavior of Children with Intellectual Disability Using Kinect Imaging Technology. In: Proceedings of the 14th international ACM SIGACCESS conference on Computers and accessibility. Boulder, Colorado, USA. 2012. p. 295-296

WII. Nintendo Wii. Disponível em: <http://wii.com/>. Acessado em: setembro de 2015.

YIN, R. K. Estudo de Caso-: Planejamento e Métodos. [S.l.]: Bookman editora, 2015.

ZAFRULLA, Z.; BRASHEAR, H.; STARNER, T.; HAMILTON, H.; PRESTI, P.; American Sign Language Recognition with the Kinect. In: Proceedings of the 13th international conference on multimodal interfaces. Alicante, Spain. 2011. p. 279-286.

ZHAO, W.; CHAI, J.; XU, Y. Combining Marker-based Mocap and RGB-D Camera for Acquiring High-fidelity Hand Motion Data. In: Eurographics/ ACM SIGGRAPH Symposium on Computer Animation, Lausanne, Switzerland, 2012. Proceedings of the ACM SIGGRAPH/Eurographics Symposium on Computer Animation, SCA '12. p. 33-42

Apêndice A

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDOS – PARTICIPANTES DO EXPERIMENTO

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

1. Você está sendo convidado para participar da pesquisa “UM FRAMEWORK PARA APOIO À CAPTURA DE MOVIMENTOS DAS MÃOS COMO FORMA DE INTERAÇÃO”.

2. Considera-se propor um *framework* para que estimule e facilite o uso da captura de movimentos da mão para interação, uma vez que esta tecnologia está em ascensão. Espera-se que este *framework*, que até os desenvolvedores iniciantes possam utilizar captura de movimentos como forma de interação com pouco ou nenhum esforço.

a. Você foi selecionado e sua participação não é obrigatória.

b. Os objetivos deste estudo são validar a utilização do *framework*.

c. Sua participação nesta pesquisa consistirá em utilizar o *framework* proposto, *framework* padrão, e interação através de mouse e responder a um questionário.

3. Os participantes são escolhidos de forma aleatória. Requisitos ser aluno de um curso Técnico em Informática ou similar e estar cursando ou ter cursado disciplinas de Desenvolvimento de Software I e Programação de Computadores ou similares, que tenham como diretrizes programação orientação a Objeto.

4. A pesquisa baseia-se no teste do *framework* com desenvolvedores de nível técnico visando medir o ganho de produtividade, praticidade, uso e dificuldade.

5. A sua participação neste estudo envolve riscos como desconforto moral e ético pela privacidade, ou físico por ficar exposto (a) a radiação, em contato com o computador.

a. Esta pesquisa será realizada em um ambiente escolar, na Etec Coronel Raphael Brandão, na qual você está com grande frequência, com o intuito de diminuir os riscos de desconfortos.

b. Os dados que serão publicados desse estudo não te identificarão devido ao uso de siglas e números para referir a você, quando necessário.

c. Há risco de o *framework* não melhorar o desempenho e dificuldade dos desenvolvedores.

6. Não se aplica nenhum risco ao participante por utilizar o *framework* em seus projetos ou de participar nestes estudos.

7. Será ministrado uma aula em formato de treinamento, então será proposto três exercícios visando a utilização do *framework* desenvolvido, *framework* padrão e interação através de mouse. Após será solicitado informações sobre a utilização do *framework* e sugestões.

8. A qualquer momento você pode desistir de participar e retirar seu consentimento. Sua recusa não trará nenhum prejuízo a sua relação com o pesquisador ou instituição.

9. As informações obtidas através dessa pesquisa serão confidenciais, e asseguram o sigilo sobre sua participação. Os dados não serão divulgados de forma a possibilitar sua identificação.

10. Caso necessite de uma cópia deste termo, ele pode ser solicitado junto ao Pesquisador Paulo Eduardo Cardoso Andrade pelo e-mail: paulo.andrade@dc.ufscar.br.

_____

Paulo Eduardo Cardoso Andrade
paulo.andrade@dc.ufscar.br / (55) 17 981482743

Barretos, ____ de _____de 2016.

_____

Sujeito da Pesquisa

Apêndice B

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDOS – PROFESSORES

TERMO DE CONSENTIMENTO LIVRE E ESCLARECIDO

1. Você está sendo convidado para participar de uma entrevista sobre "UM FRAMEWORK PARA APOIO À CAPTURA DE MOVIMENTOS DAS MÃOS COMO FORMA DE INTERAÇÃO".

2. Considera-se propor um *framework* para que estimule e facilite o uso da captura de movimentos da mão para interação, uma vez que esta tecnologia está em ascensão. Espera-se que este *framework*, que até os desenvolvedores iniciantes possam utilizar captura de movimentos como forma de interação com pouco ou nenhum esforço.

a. Você foi selecionado e sua participação não é obrigatória.

b. Os objetivos deste estudo são validar a utilização do *framework*.

c. Sua participação nesta entrevista consistirá em utilizar participar de um treinamento no formato de uma aula expositiva e ao final responder um questionário na forma de entrevista.

3. Os participantes são escolhidos através de requisitos: ser professor dos participantes do experimento nas disciplinas de Desenvolvimento de Software I e Programação de Computadores ou similares, que tenham como diretrizes programação orientação a Objeto.

4. A entrevista em uma percepção de professores, sobre a *framework* testada por seus alunos.

5. A sua participação neste estudo envolve riscos como desconforto moral e ético pela privacidade, ou físico por ficar exposto (a) a radiação, em contato com o computador.

a. Esta pesquisa será realizada em um ambiente escolar, na Etec Coronel Raphael Brandão, na qual você está com grande frequência, com o intuito de diminuir os riscos de desconfortos.

b. Os dados que serão publicados desse estudo não te identificarão devido ao uso de siglas e números para referir a você, quando necessário.

c. Há risco de o *framework* não melhorar o desempenho e dificuldade dos desenvolvedores.

6. Não se aplica nenhum risco ao participante por participar do treinamento e responder o questionário, e poderá utilizar em seus projetos ou de participar nestes estudos.

7. A qualquer momento você pode desistir de participar e retirar seu consentimento. Sua recusa não trará nenhum prejuízo a sua relação com o pesquisador ou instituição.

8. As informações obtidas através dessa entrevista serão confidenciais, e asseguram o sigilo sobre sua participação. Os dados não serão divulgados de forma a possibilitar sua identificação.

9. Caso necessite de uma cópia deste termo, ele pode ser solicitado junto ao Pesquisador Paulo Eduardo Cardoso Andrade pelo e-mail: paulo.andrade@dc.ufscar.br.

_____

Paulo Eduardo Cardoso Andrade
paulo.andrade@dc.ufscar.br / (55) 17 981482743

Barretos, ____ de _____de 2016.

_____

Sujeito da Entrevista

Apêndice C

QUESTIONÁRIO EM GRUPO APLICADO PARA VALIDAR O EXPERIMENTO.

Questionário Grupo

Atividade 1 – Movimentar Cursor

Tempo

SDK Microsoft – Início: _____ : _____ Fim: _____ : _____ [ ] Não Realizou
Mouse – Início: _____ : _____ Fim: _____:_____ [ ] Não Realizou
EasyMocapHand – Início: _____:_____ Fim: _____:_____ [ ] Não Realizou

Legenda Alternativas: 1- Não concordo totalmente; 2 - não concordo parcialmente; 3- Indiferente, 4 - Concordo Parcialmente; 5 - Concordo Totalmente.

Treinamento – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Material de Apoio – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o treinamento seria possível desenvolver a atividade -

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o material seria possível conseguiria desenvolver a atividade

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Atividade 2 – Pegar e Arrastar

Tempo

SDK Microsoft – Início: _____ : _____ Fim: _____ : _____ [ ] Não Realizou
Mouse – Início: _____ : _____ Fim: _____:_____ [ ] Não Realizou
EasyMocapHand – Início: _____:_____ Fim: _____:_____ [ ] Não Realizou

Legenda Alternativas: 1- Não concordo totalmente; 2 - não concordo parcialmente; 3- Indiferente, 4 - Concordo Parcialmente; 5 - Concordo Totalmente.

Treinamento – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Material de Apoio – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o treinamento seria possível desenvolver a atividade

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o material seria possível conseguiria desenvolver a atividade

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
Mouse	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Atividade 3 – Pegar e Girar

Tempo

SDK Microsoft – Início: _____ : _____ Fim: _____ : _____ [ ] Não Realizou

EasyMocapHand – Início: _____:_____ Fim: _____:_____ [ ] Não Realizou

Legenda Alternativas: 1- Não concordo totalmente; 2 - não concordo parcialmente; 3- Indiferente, 4 - Concordo Parcialmente; 5 - Concordo Totalmente.

Treinamento – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Material de Apoio – Foi essencial para o desenvolvimento

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o treinamento seria possível desenvolver a atividade

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apenas com o material seria possível desenvolver a atividade

SDK Microsoft	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5
EasyMocapHand	[ ] 1	[ ] 2	[ ] 3	[ ] 4	[ ] 5

Apêndice D

QUESTIONÁRIO INDIVIDUAL APLICADO PARA VALIDAR O EXPERIMENTO.

Questionário

Nome:
E-mail:
1 – Qual seu Grupo?

[ ] 1
[ ] 2

Levantamento de seu Conhecimento:

Legenda:
1-Nenhum;
2 – Básico;
3 - Regular;;
4-intermediário;
5- Avançado;

2 – Conhecimento em Orientação Objeto?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

3 – Conhecimento em C#?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

4 – Conhecimento em Unity?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

5 – Conhecimento em Captura de Movimentos?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

Legenda Alternativas:
1- Não concordo totalmente;
2 - Não concordo parcialmente;
3- Indiferente;
4 -Concordo Parcialmente;
5 - Concordo Totalmente

Sobre o desenvolvimento com SDK da Microsoft

6 – É necessário que o nível de conhecimento em orientação objeto seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

7 – É necessário que o nível de conhecimento em C# seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

8 – É necessário que o nível de conhecimento em Unity seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

9 – É necessário que o nível de conhecimento em Captura de Movimentos seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

10 –De forma geral utilização e desenvolvimento pode ser considerado de nível avançado?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

Sobre o desenvolvimento com a interação Através do Mouse

11 – É necessário que o nível de conhecimento em orientação objeto seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

12 – É necessário que o nível de conhecimento em C# seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

13 – É necessário que o nível de conhecimento em Unity seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

14 – É necessário que o nível de conhecimento em Captura de Movimentos seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

15 –De forma geral utilização e desenvolvimento pode ser considerado de nível avançado?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

Sobre o desenvolvimento EasyMoCapHand

16 – É necessário que o nível de conhecimento em orientação objeto seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

17 – É necessário que o nível de conhecimento em C# seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

18 – É necessário que o nível de conhecimento em Unity seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

19 – É necessário que o nível de conhecimento em Captura de Movimentos seja avançado para um bom desempenho no desenvolvimento?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

20 –De forma geral utilização e desenvolvimento pode ser considerado de nível avançado?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

21 –A utilização desse framework facilita a utilização da captura de movimentos?
[ ] 1 [ ] 2 [ ] 3 [ ] 4 [ ] 5

A partir deste experimento, por favor, indique o que considerou pontos fortes e fracos do framework

Apêndice E

QUESTIONÁRIO ENTREVISTA PROFESSORES.

Questionário Entrevista Professores

1 Você acredita que o *framework* proposto tornará acessível o desenvolvimento de aplicações através dos movimentos capturados pelas mãos?

R:

2 Sem o treinamento e material de apoio aos alunos chegariam a desenvolver estas atividades com a SDK Microsoft Kinect?

R:

3 Comparado ao mouse, o *framework* proposto é possível afirmar que os desenvolvimentos sejam similarmente fáceis.

R:

Apêndice F

MATERIAL DE APOIO *SKD MICROSOFT KINECT*

Material de Apoio
Desenvolvimento SDK KINECT

Objetos:

Os objetos que devem ser instanciados para você obter todo os recursos do sensor Kinect e dos dados da captura de movimentos são:

- Objeto para referenciar o Kinect

```
private KinectSensor sensor;
```

- Objeto para referenciar a captura por frame dos corpos dos atores

```
private BodyFrameReader reader;
```

- Vetor de objetos body para armazenar os corpos com o movimento capturado no frame

```
private Body[] data = null;
```

Método Start Unity

Para inicializar a captura de movimentos você precisa ativar o sensor

- Objeto recebe o Kinect que está conectado ao computador.

```
Sensor = KinectSensor.GetDefault();
```

- Verificação se foi possível identificar o sensor

```
 if sensor != null)
 {
```

- Abre o leitor para os recursos do corpo do ator

```
     reader= sensor.BodyFrameSource.OpenReader();
```

- Verifica se o sensor está aberto

```
     if (!sensor.IsOpen)
```

```
            {
                sensor.Open();
            }
        }
```

Método Update do unity

Para você sempre atualizar os dados e pontos dos movimentos capturados a cada frame.

- Verifica se o objeto não é nulo, ou seja, conseguiu fazer a leitura do kinect

```
if (reader != null)
        {
```

- Pega o último frame de captura

```
            var frame = reader.AcquireLatestFrame();
```

- Verificação se não é nula.

```
           if (frame != null)
             {
```

- Se o vetor for nulo

```
                if (data == null)
                {
```

- Ele é instanciado com a quantidade de corpos que o BodyCount Informar.

```
                data = new Body[sensor.BodyFrameSource.BodyCount];
}
```

- Colocar as informações mais atuais no Vetor _Data

```
                        frame.GetAndRefreshBodyData(data );
```

- Descarta Frame

```
                                    frame.Dispose();
```

- deixa a var nula.

```
                frame = null;
            }
        }
```

Método void FixedUpdate do unity

- Verificação se o vetor que contém as informações de captura não é nulo

```
if (data != null)
        {
```

- Uma estrutura “for” para percorrer todo o vetor

```
            for (int i = 0; i < data .Length; i++)
            {
```

- Verifica se foi possível capturar os movimentos deste corpo

```
                if (data[i].IsTracked)                {
```

- Com o código abaixo é criado um vetor 2D, utilizando como posições X e Y a posição da junta da mão direita ou esquerda bastando trocar para handRight, O kinetc e o unity trabalho com proporções diferentes de telas então é necessário multiplicar o valor da posição x e y por 10.

```
transform.position = new Vector2(data[i].Joints[Windows.Kinect.JointType.HandRight].Position.X * 10, data[i].Joints[Windows.Kinect.JointType.HandRight].Position.Y * 10);
```

- Para eu conseguir girar tanto o cursor ou objeto é preciso quatro informações sobre a orientação das juntas, para facilitar é adequado criar quatro variáveis float como o exemplo abaixo.

```
ox = data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.X;
oy = data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.Y;
oz = data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.Z;
ow = data[i].JointOrientations[Windows.Kinect.JointType.HandRight].Orientation.W;
```

- Com essas quatro informações é possível girar o objeto, porém é necessário fazer alguns ajustes multiplicando ow e oz por 360 para se adequadar aos padrões unity e ainda utilizar um parâmetro t como o.08f float para dizer qual o tempo para o giro acontecer, ou seja, saio da posição inicial para a final em t.

```
transform.localRotation = Quaternion.Slerp(transform.localRotation, new Quaternion(ox, oy, oz * 360, ow * 360), 0.08f);
```

- Para saber quando a mão está aberta ou fechada, Lasso seria dois dedos abertos e o restante fechado. Foi criada uma variável para auxiliar HandState op.

```
if (data[i].HandRightState == HandState.Closed || data[i].HandRightState == HandState.Lasso)
                        {
                            op = data[i].HandRightState;
                        }
                        else
                        {
                            op = data[i].HandRightState;
                        }
                    }
                }
```

- Finaliza saindo do for já que foi possível realizar a captura

```
i = data.Length;
```

Método void OnTriggerStay2D(Collider2D colisor)

Com ele conseguir interagir o cursor que está sendo controlado pelos movimentos do ator

- Primeiro verifico se a mão está no estado fechado, lasso ou desconhecida se estiver em qualquer um destes considero que o objeto foi pego

```
if (op == HandState.Closed || op == HandState.Lasso || op == HandState.Unknown)
        {
```

- Agora é possível pegar o objeto que está colidindo com o cursor e movimenta-lo pegando as posições do próprio objeto que está sendo controlado pelo ator

```
colisor.gameObject.transform.position = new Vector2(this.transform.position.x, this.transform.position.y);
```

- Se eu desejar gira-lo devo utilizar novamente o método explicado para girar o cursor utilizando os mesmos parâmetros

```
colisor.transform.localRotation = Quaternion.Slerp(this.transform.localRotation, new Quaternion(ox, oy, oz * 360, ow * 360), 0.08f);
}
```

Método void OnApplicationQuit()

Para finalizar e fechar o senso assim que aplicação se encerrar.

- Verifico se é diferente de nulo

```
if (reader!= null)
{
    reader.Dispose();
    reader = null;
}
```

- Fecho sensor.

```
if (sensor!= null)
{
    if (sensor.IsOpen)
    {
        sensor.Close();
    }

    Sensor = null;
}
```

Apêndice G

MATERIAL DE APOIO *EASYMOCAPHAND*

Material de Apoio
Desenvolvimento EasyMoCapHand

HandAction

Objeto deve ser instanciado para você ter todos recursos da captura de movimento dentro do mesmo.

Exemplo: `public HandAction hand = new HandAction();`

Método Start

Método responsável por inicializar o sensor para início da captura.

Exemplo: `hand.Start(true);`
Obs.1: este método deve ficar alocado no método start padrão do unity
Obs.2: o parâmetro de inicialização refere-se a mão direita(true) e esquerda (false).

Método Update

Método responsável por atualizar os dados referentes ao movimento da mão jogador.

Exemplo: `hand.Update();`
Obs.1: este método deve ficar alocado no método update padrão do unity

Método getPosition()

Método responsável retornar um vecto2 como posições (x,y)

Exemplo: `transform.position = hand.getposition();`
Obs.1: mesmo position sendo vector3 e o método getposition retornar vetor2 o unity adequa o eixo z para 0.
Obs.2: getposition tem sobrecarga getposition(`ref Transform obj`) recebendo como parâmetro transform.
Obs.3: este método deve ficar alocado no método update padrão do unity

Método turn

Método responsável girar o objeto a partir de um parâmetro de transform.localRotation do objeto a ser girado, e retornando a nova posição e orientação.

Exemplo: `transform.localRotation = hand.turn(transform.localRotation);`
Obs.1: este método deve ficar alocado no método update padrão do unity

Método quit

Método responsável por encerrar sensor. Deve ser alocado em um método padrão da unity `void OnApplicationQuit()`

Exemplo:

```
void OnApplicationQuit()
    {
        hand.quit();
    }
```

Método gotObject

Método responsável por pegar e girar objeto através de seus parâmetros public void gotObject(bool move, bool turn, ref Collider2D colisor)

Exemplo:

```
hand.gotObject(true, false, ref colisor);
```
Obs.: neste caso você quer que quando pegue o objeto, o mesmo deve apenas se mover em 2d.

```
hand.gotObject(false, true, ref colisor);
```
Obs.: neste caso você quer que quando pegue o objeto, o mesmo deve apenas girar.

```
hand.gotObject(true, true, ref colisor);
```
Obs.: neste caso o objeto irá se mover e girar de acordo com a mão do jogador

OBS.1: esse médo deve ficar no método padrão do unity

```
 void OnTriggerStay2D(Collider2D colisor)
{
hand.gotObject(true, true, ref colisor);
}
```

OBS.2: o objeto que colidiu deve passar como referência para o método já entregar o objeto na posição correta.

Apêndice H

MATERIAL DE APOIO *MOUSE*

Material de Apoio
Desenvolvimento com o mouse

Vector 3

Objeto que tem atributos relacionados a dimensões (x,y,z).

Exemplo:
```
Vector3 vetor;
```

Posição do mouse

Utilizando esta classe estática, e passando como parâmetro a a posição do mouse na tela ela vai te retornar um Vector3 ou seja ela te oferece a posição do mouse em “x,y e x”.

```
Camera.main.ScreenToWorldPoint();
Input.mousePosition
```

Exemplo:
```
vetor = Camera.main.ScreenToWorldPoint(Input.mousePosition);
```

Obs.1: Como estamos trabalhando com jogos em 2D, é aconselhável zerar a posição z do objeto Vector3

Obs.2: Ideal que este código fique no método update do unity.

Exemplo:
```
Vetor.z = 0f;
```

Obs.: f é utilizado para forçar a se coloca um valor float.

Métodos relevantes

Método Unity que informa se o objeto que contém este script está colidindo com algum outro objeto.

```
void OnTriggerStay2D(Collider2D colisor)
{
}
```

Método Unity que informa se o click do mouse está permanentemente ativado.

```
void OnMouseDown()
        {

        }
```

Obs.: Ideal para fazer uma variável bool para controlar o click do mouse se está apertado ou não

Método Unity que informa se o click do mouse está permanentemente desativado.

```
void OnMouseUp()
        {
        }
```

Exemplo:

```
   bool click;
void OnMouseDown()
        {
  click = false;
        }
 void OnMouseUp()
        {
  click = true;

        }
```

Obs.: podendo utiliza-la para movimentar o objeto que colidiu quando o mouse estiver clicado.

Exemplo:

```
void OnTriggerStay2D(Collider2D colisor)
                {
                  if (click)
   colisor.transform.position = vetor
                }
```

Movimentar Objeto.

No Unity o ideal para se movimentar um objeto, é utilizar transform.position, esse método espera receber um objeto Vector3 ou coordenadas separadas em x, y ou z.

Exemplo:

```
                transform.position = vetor;
transform.position.x = vetor.x   ou transform.position.x = 15f;
```