Universidade Estadual do Norte Fluminense

Centro de Ciências e Tecnologia - CCT

Labotório de Ciências Matemáticas - LCMAT

Desenvolvimento de Jogos para Estimulação de Movimentação Utilizando o Sensor de Movimento

Monografia para obtar o grau acadêmico de:
Bacharel em Ciência da Computação

Apresentado por:

Vinícius Galiaço

Campos dos Goytacazes - RJ, Brasil, Dezembro de 2016.

Vinícius Galiaço

Desenvolvimento de Jogos para Estimulação de Movimentação Utilizando o Sensor de Movimento

Monografia apresentada junto ao Curso de Ciência da Computação, da Universidade Estadual do Norte Fluminense Darcy Ribeiro - Campos / RJ, como requisito para obtenção do título de Bacharel em Ciência da Computação.

Orientador: Prof. Luis Rivera

Universidade Estadual do Norte Fluminense
Darcy Ribeiro

Campos dos Goytacazes, RJ, Brasil, 16 de janeiro de 2017.

"*Dedico este trabalho à minha família e àqueles que me apoiaram, tanto os que ainda estão aqui quanto os que já se foram*"

Agradecimentos

A Deus, que me permitiu chegar até aqui.

À minha família, que sempre me apoiou a seguir o que eu sonhava.

Aos meus professores e educadores, que fizeram seu melhor para que eu aprendesse o melhor possível e conseguisse realizar meus sonhos.

Aos meus amigos, presentes e distantes, que levantavam meu espírito quando as dificuldades me faziam vacilar.

Resumo

Desenvolvimento de um jogo eletrônico que utiliza controle por captura de movimento com o objetivo de ajudar a diminuir o sedentarismo na infância e adolescência. Foi utilizado o sensor de movimentos Kinect para capturar os movimentos do jogador e transformá-los em comandos, utilizados pelo motor de jogos Unity para controlar o personagem do jogo. O jogo utilizado foi um exemplo simples de jogo de plataforma, e o personagem utilizou como base uma máquina de estados finita.

Abstract

Development of an electronic game that utilizes motion capture controls with the goal of helping lower sedentary numbers during childhood and teenagerhood. The motion sensor Kinect was utilized to capture the player's movements and convert them in commands, that were utilized by the game engine Unity to control the game character. The game used was a simple example of a platformer, and the character used as a base a finite state machine.

Sumário

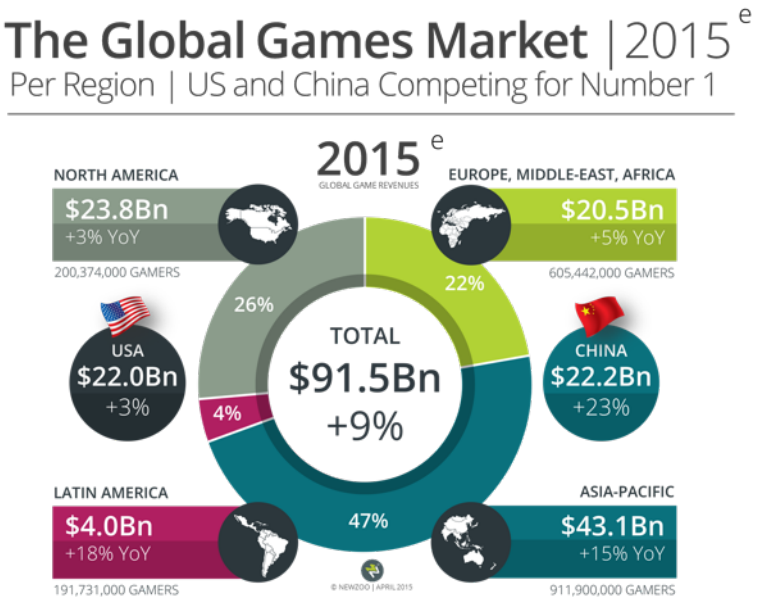
Lista de Figuras

Capítulo 1

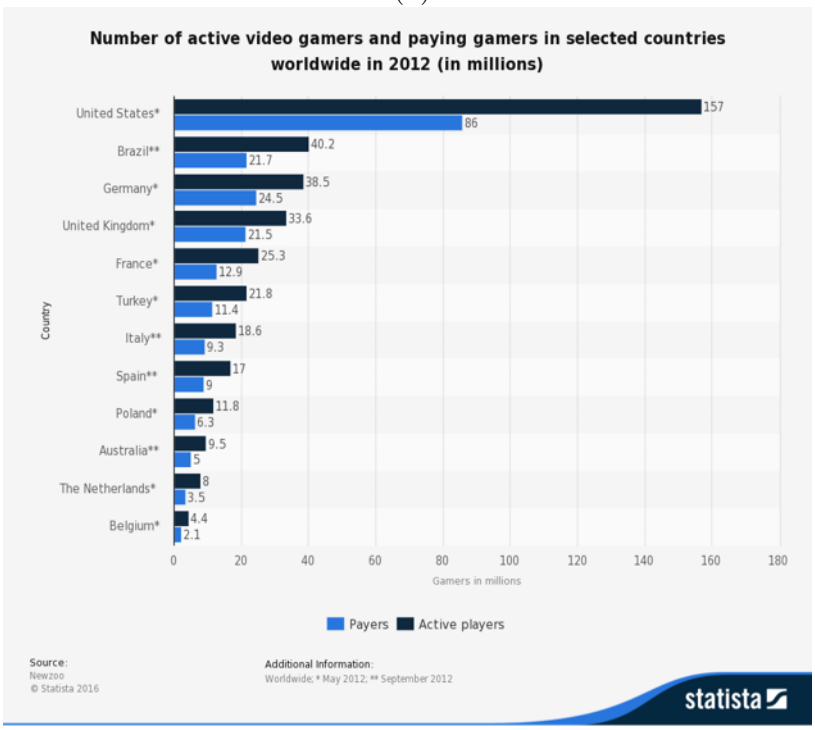
Introdução

A indústria de jogos virtuais, ou entretenimento eletrônico interativo, está em constante expansão desde seu início nos anos de 1970. De fato, segundo pesquisas mesmo no Brasil, em que ela não é desenvolvida em comparação com países como Estados Unidos e Japão, apenas no início de 2015 ela empregava mais de 4 mil pessoas e faturou $R\$$ 900 milhões por ano apenas no país, com sua maior dificuldade sendo encontrar pessoal qualificado [Gubertt2015]. O país foi classificado, em 2015, como o quarto consumidor mundial de games, mas como já dito lhe falta mão de obra qualificada para investir na indústria. Os empregos no setor cresceram 13 vezes mais no país que o mercado em geral, mas possui mais vagas do que candidatos. Segundo Marcelo Hodger, diretor da Escola de Games, "este é um mercado que está empregando muita gente, está crescendo, mas sofre com mão de obra qualificada. Hoje, na indústria criativa, temos um salário médio que é três vezes maior do que o mercado normal. Por quê? Porque falta gente qualificada para entrar nessa indústria" [OGlobo2015].

No mundo, sendo o mercado de games uma das maiores indústrias atuais internacionalmente, superando o faturamento de Hollywood e estimando-se superar o valor de $U\$$ 100 bilhões até 2017 [Ferreira2014], a empresa de videogames está sempre se expandindo, e o mercado de computadores pessoais se expande para acomodar esta indústria. Um dos modos em que esta expansão ocorre é em tentativas de aumentar a imersão do jogador, como, no caso deste trabalho, com interação natural baseada em movimentos e gestos.



(a)



(b)

Figura 1.1: Dados do mercado de jogos: (a) Divisão das vendas do mercado de jogos pelo mundo em 2015 (fonte: NewZoo); (b) Número de jogadores, ativos e pagantes, em 2012 (fonte: Cruz et al, 2012)

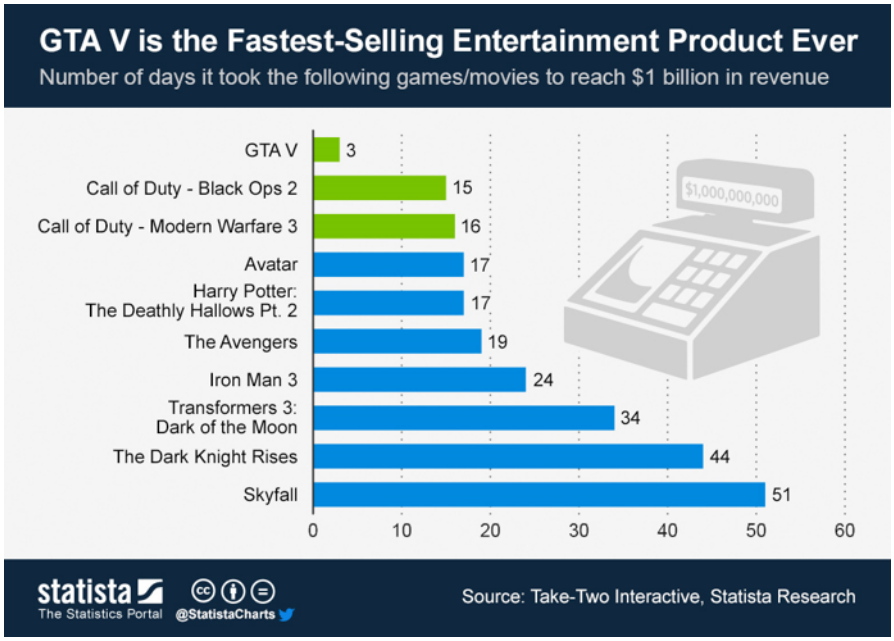


Figura 1.2: Tempo para alcançar um milhão de dólares em lucro. Alguns jogos superaram grandes filmes.

1.1 Problemas gerados por jogos

Sedentarismo é um problema exacerbado pela tecnologia, desde a televisão até as redes sociais.

Almeida e Guedes [Almeida-Guedes2015] afirmam em relação às atividades físicas e estilo de vida impostas pela tecnologia e sociedade: "A partir da Revolução Industrial, com a tecnologia se expandindo de forma avassaladora, observou-se uma mudança muito grande no estilo de vida dos indivíduos. A sociedade que era acostumada a realizar trabalhos que utilizava muito esforço físico, transformou-se em uma sociedade urbana, com uma população estressada, ansiosa e que pratica pouca atividade física. Atualmente, mesmo com a facilidade em obter informações que a tecnologia proporciona, muitas pessoas só começam a praticar atividade física após a recomendação médica."

Um dos grupos mais afetados, no entanto, é a faixa etária que constitui a infância e adolescência, pois são partes formativas que afetam o resto da vida dos indivíduos. Continuam Almeida e Guedes "brincadeiras como correr, saltar, pique-pega, esconde-esconde dentre outras, são cada vez mais difíceis de serem vistas no cotidiano de crianças. Com a expansão da tecnologia, aparelhos eletrônicos como: DVD, compu-

tadores, televisão, celulares e tablets, tornaram as crianças mais inativas e propícias ao sedentarismo. Este fenômeno influência diretamente a maturação cognitiva, afetiva, social e motora das crianças. Vários estudos apontam prevalências consideráveis de doenças relacionadas ao sedentarismo na população. Este fato pode ser atribuído ao atual estilo de vida das pessoas, tais como: utilização demasiada de aparelhos eletrônicos, estudos excessivos, trabalho sem esforço físico, maior oferta e consumo de produtos de alto valor energético. Torna-se ainda mais preocupante em crianças e adolescentes, uma vez que é nesta fase, que o indivíduo adquire vários hábitos que tendem a permanecer na vida adulta."

Não que a tecnologia seja a única culpada, a diminuição de áreas de lazer disponíveis também tem sua contribuição. O avanço da tecnologia, a expansão das cidades e o aumento do número de condomínios prediais, limitaram as áreas destinadas ao lazer e à prática esportiva, e, as crianças perderam o espaço para brincar. Esses fatores contribuíram para o aumento do sedentarismo, diminuindo as opções de brincadeiras, a autonomia das crianças, prejudicando o desenvolvimento motor e das capacidades físicas: resistência, agilidade, velocidade, força, equilíbrio, coordenação, flexibilidade. Diante deste contexto em que a sociedade se encontra, os grandes prejudicados são as crianças e adolescentes em idade escolar, que estão se tornando cada dia mais sedentários. Ao invés de saírem para brincar nos parques, nas ruas ou nas quadras, eles passam o dia em frente à TV, computadores e celulares, desfrutando desse conforto que a tecnologia proporciona [Almeida-Guedes2015].

Apesar de games serem considerados um dos principais fatores do problema de sedentarismo, a jogabilidade por utilização de movimento tem sido considerada uma solução por anos. "No campo dos jogos que possibilitam a movimentação corporal como forma de interação, um dos primeiros exemplos que começou a chamar a atenção da mídia e de pesquisadores foi o Dance Dance Revolution - DDR (tapete de dança). Em 2007, por exemplo, o jornal The New York Times enfatizava em uma de suas matérias o fato de que centenas de escolas em pelo menos dez estados americanos passaram a utilizar o DDR como parte regular de seu currículo de Educação Física." [FincoFraga2011].

De fato, recentes pesquisar apontam que os fatores que levam ao sedentarismo não são diretamente relacionados aos jogos em si. "Nesta pesquisa os objetivos específicos foram analisar os tipos de jogos utilizados pelos adolescentes, e quantificar o tempo de

utilização diária dos jogos, bem como o tempo diário de atividades físicas praticadas. O método utilizado foi à aplicação de questionário aos estudados, caracterizando a pesquisa como estudo exploratório descritivo. Percebeu-se que os jogos eletrônicos não influenciam na ausência de práticas de atividades físicas dos adolescentes. Estes preferem, na maioria, atividades físicas 'na rua' aos jogos eletrônicos. Identificou-se também a 'esteriotipação' de: esporte saudável: na rua; jogar videogame: ruim à saúde, na relação atividade física e sedentarismo." [deFaria2015].

Sendo assim, uma das áreas em que jogos controlados por movimentos físicos tem potencial para serem utilizados é em combate ao Sedentarismo.

1.2 Aplicações dos jogos

Tendo em vista o avanço tecnológico que representam, os produtos da indústria de games se tornaram uma ferramenta muito utilizada, seja para uso medicinal, educacional, militar ou mesmo econômico, e uma variedade de outros campos onde exista a interação humana.

No campo medicinal, aplica-se em tratamento de pacientes que sofreram derrames que poderam recuperar controle motor dos braços até cinco vezes mais rápido utilizando jogos eletrônicos do que com terapia convencional [MichaelHosp2011]. Na melhoria da capacidade visual, como comprovado em um estudo realizado pela universidade de Rochester [Univ-Rochester2007]. Também treinam a habilidade de lidar com várias tarefas ao mesmo tempo [Green-Baveller2003], ao ponto de alguns estudos demonstrarem um aumento de 10 a 20 por cento nas habilidades perceptuais e cognitivas [Freeman2010], e outras aplicações relacionadas.

No campo educacional, os jogos eletrônicos são muito utilizados. Os mais conhecidos são jogos para ensinar matérias escolares, como os clássicos jogos da série "Onde Está Carmen Sandiego"[HMHCO2017]. Mas também existem jogos para ensinar a digitar rápido e corretamente, como a série The Typing of The Dead[Park2001], e até mesmo ensinar outros idiomas menos comuns, como os jogos Slime Forest Adventure[lrnj] e Koe[StrwGam] ensinam japonês, e outros usos.

No campo militar, os jogos de treinamento, como simuladores militares. Além dos já conhecidos simuladores de vôo, o setor militar dos Estados Unidos investiu em

simulações virtuais para treinamento de controle de drones [Robillard2011] e dramatizações virtuais de atos inimigos reais [Jean2009].

1.3 Objetivos

Este trabalho tem como objetivo desenvolver um jogo eletrônico controlado através de detecção de movimentos, fazendo o jogador mover o próprio corpo e se exercitar para cumprir o objetivo do jogo.

Para tal, foi criado um jogo com comandos simples para o jogador, com o objetivo do avatar adquirir uma chave com a qual sair de uma sala onde se encontra preso.

1.4 Metodologia

O jogo em questão será criado utilizando o motor de desenvolvimento de jogos Unity e o controlador Kinect, utilizando o pacote Kinect with MS-SDK como base para a programação.

A programação foi feita usando a linguagem C#, uma vez que é a linguagem comum entre o Unity e o kit de desenvolvimento padrão para Kinect disponibilizado pela Microsoft. Foi baseada em uma máquina de estados, tornando definidos os eventos que afetam a movimentação do avatar e o que ele é capaz de fazer a cada determinado momento.

1.5 Organização do trabalho

O Capítulo 2 abordará conceitos de jogos eletrônicos, quer utilizem controles por detecção de movimentos ou não, assim como trabalhos relacionados. O Capítulo 3 abordará as teorias utilizadas pelo jogo eletrônico gerado para este trabalho e o processo de implementação do jogo em si. No Capítulo 4 são descritos as conclusões e planos para trabalhos futuros.

Capítulo 2

Jogos Interativos Eletrônicos por Controle de Sensores de Movimentos Físicos

O jogo eletrônico, videojogo ou *videogame* é aquele que usa a tecnologia de computador. Ele pode ser jogado em computadores pessoais (incluindo mobiles), em máquinas de fliperama ou em consoles. Um console é um computador pequeno que serve basicamente para jogar videogame-PlayStation, Xbox e Wii são exemplos.

Esse tipo de jogos, na linha de "jogos fisicamente interativos (JFIs)", é uma alternativa para evitar os distúrbios físicos, e mais bem permite o fortalecimento e desenvolvimento físico nos infantes e adolescentes, que também eles podem tirar vantagens de esse tipo de aplicativos para inserir desenvolvimentos mental e psicológico. Nos JFIs, usualmente os movimentos de partes de corpo do usuário são usados para interatuar direta ou indiretamente com o jogo. Os movimentos do corpo são capturados por um dispositivo ou sensor de captura, tipo câmera ou kinect, e com a ajuda de visão computacional são transferidos como controle de movimentos para o ambiente virtual do jogo. A interação pode ser através de um objeto animado que representa ao usuário no ambiente virtual, de forma que esse objeto virtual realiza ações que o usuário realiza na área de visão do sensor.

2.1 Controladores e Sensores de Movimento

Os sensores de movimentos são dispositivos que capturam a sequencia de posturas de movimento que são informações fundamentais para interpretar a intenção do usuário em movimento. A interpretação do movimento gera os comandos de controle para o processador realizar as operações desejadas no jogo. O Kinect é um exemplo de sensor de movimentos.

Captura de movimentos é um dos métodos de controle de jogos mais antigos, havendo tentativas de aplica-lo desde o final da década de 1980, quando a Nintendo lançou o acessório Power Glove para seu sistema 8-bits Nintendo Entertainment System, baseado na tecnologia já existente de luvas virtuais, que utilizava de ultrassom para detectar movimentos. Ele era considerado impreciso pois seus componentes eram de baixa qualidade, com muitas distorções e interferências [Stefani2016], além de terem apenas 4 graus de liberdade[1], as coordenadas e mais roll [Queiroz2010]. Este processo foi utilizado novamente com o Wii Remote, apresentado no Tokyo Game Show em 2005 [Mirab.Casa.2010], desta vez utilizando sensores infravermelhos de melhor qualidade e todos os seis graus de liberdade do objeto navegador. No entanto, isto ainda dependia de um controle cuja posição é mapeada, em vez de um mapeamento do indivíduo.

A utilização de câmeras para mapeamento de imagens e captura de movimentos não é uma novidade, e a Sony tentou utilizar disto para lançar uma linha de jogos baseados nesta tecnologia utilizando-se do acessório EyeToy antes mesmo do lançamento do Wiimote. No entanto, o EyeToy utilizava um sistema de Detecção de Bordas, o que além de causar problemas em iluminações baixas não era uma captura de movimentos real [Oliveira2010].

Mas o Project Natal da PrimeSense[RedmondTelAviv2010], depois adquirido pela Microsoft e transformado no Kinect, utiliza-se de duas câmeras, uma tradicional e uma que captura luz infravermelha, e um projetor de luz infravermelha, e analisa os padrões formados por ela para criar um modelo tridimensional detalhado. Mas para transformar este modelo 3d em um verdadeiro controlador por detecção de mo-

[1]DOF, Degreed of Freedom, modo de se registrar o posicionamento de objetos no espaço. Consistem no posicionamento espacial nas coordenadas de translação x, y e z, e nos três eixos rotacionais, roll, rotação do objeto no eixo frontal, yaw, rotação do objeto no eixo vertical e pitch, rotação do objeto no eixo lateral que resulta na elevação frontal[Clarke11]

vimento, o aparelho necessita ser capaz de analisar este modelo e calcular o que nele é uma pessoa, não importa seu tamanho e forma corporal, e a partir dali calcular a posição de seu esqueleto estrutural e os movimentos que fizer. Para tal, foi criada uma máquina de aprendizado para que o Kinect possa aprender através da análise de milhares de exemplos, incluindo dados de captura de imagem usados em filmes e jogos antigos para inferir regras que serviriam como padrões para a análise do corpo humano, permitindo que o software aprendesse a detectar a posição do corpo de pessoas nos mais variados tamanhos, posições e ações. Este é considerado o maior projeto de aprendizado de máquina da história até então [Queiroz2010].

2.1.1 Sensor

Kinect proporciona um sensor de profundidade, uma câmera RGB, um acelerómetro, um motor de passo e um sistema de microfones multi-array. A Figura 2.1 ilustra os componentes do kinect que serão explicados detalhadamente na sequência.

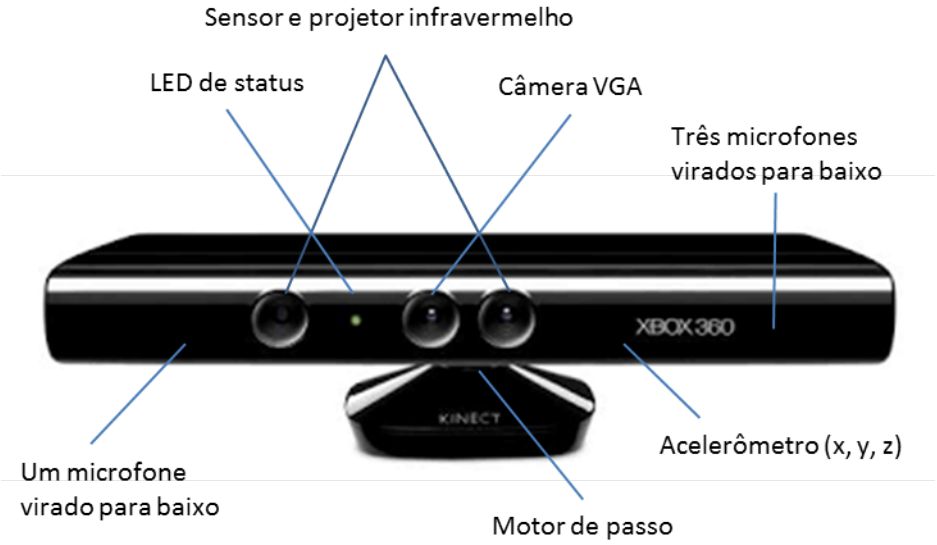


Figura 2.1: Sensor Kinect e componentes.

A.Sensor de profundade. Consiste em emissor de laser infravermelho e uma câmera (sensor) infravermelha. O laser emissor cria um padrão ruidoso conhecido de luz IR estruturado, enquanto a câmera sensor, que opera a 30Hertz e captura imagens de 1200x960 pixels, captura 2048 níveis de sensibilidade. O campo de

visão do sensor é de 58 graus na horizontal, 45 graus na vertical, 70 graus na diagonal, e a faixa operacional é de 0.8mts a 3.5mts. A detecção de profundidade utiliza um método de luz estruturada para medir a profundidade. O padrão conhecido de pontos é projetado a partir do emissor infravermelho. Esses pontos são gravados pela câmera sensor, e depois comparados com o padrão conhecido. Quaisquer perturbações são conhecidas por serem variações na superfície e poder ser detectadas como mais próximas ou mais afastadas. A imagem 2, centro e direita, mostram os resultados capturados por este sensor.

B. Câmera VGA. Câmera VGA opera a 30 Hz e pode capturar imagens de 640x480 pixels RGB de 8 bits por canal. Kinect também tem a opção de mudar para câmera de alta resolução de 10fps em 1280x1024 pixels. A própria câmera possui um conjunto de recursos, incluindo balanceamento automático de branco, preto, estabilização de tremor, saturação de cor, e correção de defeitos. As saídas podem ser emitidas em padrões RG ou GB. A Figura 2.2 (esquerdo) mostra a imagem em RGB capturado por Kinect.

C. Motor, Acelerômetro e Microfones. Kinect tem duas funções complementares importantes que permite acompanhamento dos movimentos dos usuários: um mecanismo para inclinação vertical (para abaixo ou para cima) da cabeça, e um acelerômetro. A inclinação da cabeça é feita por um motor com uma engrenagem para orientar a cabeça para abaixo ou para cima. O acelerômetro para determinar a posição em que está a cabeça. Na verdade, o motor faz com que a câmera se mexa na hora de escanear o corpo do jogador, enquanto que o acelerômetro ajuda o motor a ter mais precisão de movimento quando se mexe.

A matriz de microfone possui quatro cápsulas de microfone e opera com cada canal processando áudio de 16 bits a uma taxa de amostragem de 16 kHz.

2.2 Jogos Interativos Eletrônicos Tradicionais

Jogos interativos dependem, como o próprio nome diz, de sua interação com o usuário, também conhecido como "jogador". Consequentemente, uma parte importante destes é a interface de entrada pela qual os comandos do jogador são transmitidos e convertidos em comandos.

Nos jogos interativos tradicionais a interação do jogador com o aplicativo, neste caso videogame, é realizada através do controle que varia de acordo com a plataforma. Um controle pode ser constituído por um direcional e um único botão, outros podem ter vários botões e mais de um direcional. Muitos jogos também podem usar teclados de computador como controle de interação, mouse ou combinação dos dois simultaneamente. Também podem ser usadas outras maneiras de interação e prover informações ao jogador, como o uso de sons, algo usado em larga escala desde os primórdios. Outros tipos de respostas também comuns de se usar são dispositivos de vibração, reconhecimentos de voz, e o principio de imersão com periféricos sensores de movimentos. Esses últimos são das últimas gerações de jogos.

Há uma certa disputa entre qual seria o primeiro jogo interativo eletrônico a ser considerado, em principais o "Cathode-Ray Tube Entertainment Device" [Blitz2016], uma vez que usa hardware puramente analógico, não roda através de um dispositivo computacional e não chegou a ser manufaturado, ou "Tennis for Two" [Brok.Nat.Lab2015], que apesar de não ser um dos primeiros jogos eletrônicos computacionais criados é considerado o primeiro criado com o objetivo inicial de entretenimento.

Mas não se disputa que um dos primeiros e mais influentes jogos interativos eletrônicos foi Spacewar!, especialmente em questão de controladores [Brok.Nat.Lab1981]. Os primeiros jogos eletrônicos, não importando se você considera o "Cathode-Ray Tube Entertainment Device" ou "Tennis for Two", eram controlado por botões no aparelho. O primeiro controlador externo foi criado para Spacewar!, pois foi projetado para ser utilizado por mais de um jogador mas o computador em que rodava dava vantagem a um deles. Atualmente, os controladores dispõem não apenas de botões, mas também de alavancas sensíveis a pressão e alguns possuem até mesmo

(a) (b) (c)

Figura 2.2: Data fornecida por Kinect: (a) imagem RGB; (b) imagem de profundidade; e (c) imagem infravermelha. (fonte: Cruz et al, 2012).

giroscópios.

2.2.1 Jogos de Entretenimento

Os jogos de entretenimento possuem pelo menos uma estrutura similar à do objeto de conhecimento, o que facilita a aprendizagem deste [Costa2009]. Os jogos de entretenimento são projetados com o objetivo de divertir e prender a atenção, entreter seus jogadores. O inusitado é que, mesmo sem serem projetados com essa intenção, tais jogos alcancem resultados impressionantes no papel de jogos pedagógicos.

Em geral, os videogames funcionam da seguinte forma. Uma tela apresenta uma situação de jogo que pode ser modificada pela pessoa por meio de um controle. Em tempo real, as modificações realizadas, bem como suas consequências, vão sendo mostradas na tela. Os videogames, como um caso de sistemas interativos, possuem uma estrutura composta de partes: 1) usuário jogador; 2) controle que pode ser joystick, mouse, outros dispositivos externos; 3) uma tela, que exibe as imagens do jogo; 4) um console, que contém um processador que executa os softwares; 5) um software interativo, o programa que dita as condições virtuais do jogo.

2.2.2 Jogos Educativos

Os jogos educacionais facilitam e estimulam a aprendizagem através da interação. Incitam à resolução dos problemas propostos permitindo ao utilizador raciocinar e estimular as suas capacidades cognitivas, assim como desenvolver a sua coordenação motora e reflexiva. O jogo por si só não leva a um resultado, ou à apropriação do conhecimento simplesmente pela diversão, portanto a aprendiz precisa ser guiada, para que os objetivos sejam alcançados, assim surgiu o conceito de jogo educativo.

Jogos educativos são aqueles que estimulam e favorecem o aprendizado do aprendiz, através de um processo de socialização que contribui para a formação de sua personalidade. Eles visam estimular o impulso natural do aprendiz a aprender. Para isso, os jogos educativos mobilizam esquemas mentais, estimulam o pensamento, a ordenação de tempo e de espaço, ao mesmo tempo em que abrangem dimensões da personalidade como a afetiva, a social, a motora e a cognitiva. Eles também favorecem a aquisição de condutas cognitivas e desenvolvimento de habilidades como

coordenação, destreza, rapidez, força e concentração[2].

O surgimento do jogo educativo se deu no renascimento, os jogos de todos os tipos que na era medieval eram abominados ressurgem se incorporando novamente no cotidiano das pessoas e também como um material pedagógico no ensino. Atualmente os jogos na educação são classificados de acordo com duas funções. A primeira é a lúdica que fornece prazer e diversão, a segunda é a educativa o jogo pode auxiliar ou promover a aquisição de saberes. É importante que na aplicação de jogos haja a mediação do professor para que a função lúdica não se sobreponha a função educativa, deve haver um equilíbrio entre essas duas funções gerando maior interesse nos alunos.

Costa [Costa2009], em seu livro "O que os jogos de entretenimento têm que os educativos não têm: 7 princípios para projetar jogos educativos eficientes", apresenta as características que um jogo educativo deve ter:

1. Um jogo educativo deve possuir pelo menos uma estrutura similar ou comum à estrutura do objeto de conhecimento
2. Essa estrutura do jogo deve ser perceptível ao jogador enquanto o joga
3. A aprendizagem dessa estrutura deve ser indispensável para que se vença o jogo
4. Em um jogo educativo, tudo deve estar a favor da diversão e do entretenimento
5. A estrutura similar ou comum a do objeto de conhecimento deve estar relacionada ao jogo a que pertence por relações estruturais essenciais
6. No que depender do objeto de conhecimento, um jogo educativo deve ser uma forma original de jogo
7. Um jogo educativo deve ser, pelo menos para o seu público-alvo, melhor como jogo que qualquer uma de suas partes isoladamente ou a simples soma delas

Com esses princípios, demonstrando que jogos de entretenimento, como os videogames por exemplo, são mais efetivos quando utilizados para fins pedagógicos do que os jogos chamados educativos.

[2]pt.wikipedia.org/wiki/Jogo_educativo

2.2.3 Jogos Sérios

Segundo Wikipedia[3], um "jogo sério", do inglês "serious game", é um software ou hardware desenvolvido por meio dos princípios do design de jogo interativo, com o objetivo de transmitir um conteúdo educacional ou de treinamento ao usuário. O termo "sério" (serious) refere-se que o jogo é voltado mais para fins educacionais do que de entretenimento. Eles têm sido amplamente utilizados nas áreas de defesa, educação, exploração científica, serviços de saúde, gestão de emergência, planejamento urbano, engenharia, religião e política, de uma forma imersiva ou interativa que possam ser usufruídas da melhor forma possível.

O conceito de utilizar jogos com propósitos educativos tem a sua origem ainda antes da revolução tecnológica e do uso comum de computadores: O primeiro "serious game" foi o Army Battlezone, um projeto desenvolvido pela empresa Atari nos anos 80. Este jogo foi uma adaptação do jogo Battlezone concebida para treinar militares em situação de batalha [GamingHistory2014]. Ao longo dos anos, e à medida que os computadores para uso pessoal estão sendo desenvolvidos, os "serious games" são concebidos para uma cada vez maior variedade de áreas: educação, treinamento profissional, saúde, publicidade, e políticas públicas. Por exemplo, em área de saúde esta o "insuonline"[4] que é um jogo para médicos na aplicação de insulina.

2.3 Trabalhos Relacionados

A maioria dos trabalhos relacionados ao assunto é sobre os jogos eletrônicos utilizados para estimular exercícios físicos ou para uso em terapias, assim como este. Mas todos aparentam ter sido escritos do ponto de vista de profissionais da Educação Física ou Medicina. Estudos também demonstram um efeito terapêutico de certos jogos, como Tetris, por exemplo, tem um efeito de melhoria mental após efeitos traumáticos segundo uma pesquisa da Universidade de Oxford [Alleyne2009], e alguns jogos têm uma influência positiva na personalidade, pelo menos temporária [TheEconomist2009].

Vaghetti e Botelho [VaghettiBot2010], no artigo "Ambientes virtuais de aprendizagem na educação física: uma revisão sobre a utilização de Exergames", analisam as

[3]https://pt.wikipedia.org/wiki/Serious_game
[4]http://oge.mobi/insuonline/

capacidades de jogos na educação, tanto mental quanto física, que cresceram graças ao fato da captura de movimento para controle de jogos ter se tornado mais avançada e popular na época em que o artigo foi escrito, graças aos avanços tecnológicos que a tornaram mais acessíveis. Nessa área de educação física, um trabalho realizado por Liliana Kffuri [Kffuri2013] propõe os vídeo games com sensor de movimentos seja uma ferramenta didática pedagógica, aliando as novas tecnologias à atual geração de alunos que têm uma grande predisposição ao domínio das tecnologias. Ela considera que os conteúdos de Diretrizes Curriculares da Educação Básica (DCE), incluindo esporte e danças, tenham uma forma diferenciada com esse tipo de ferramentas tecnológicas, já que os sensores de movimento, tipo Kinect, utiliza todo o corpo para jogar.

Talvez, uma das áreas interessantes, similar às disciplinas de prática de educação física, é a terapia médica. Uma forma em que os pacientes pratiquem as terapias de movimentos da parte do corpo com problema sem muito esforço e minimizando dores, como jogando, já que sua concentração vai estar focada no jogo. Trabalhos nessa área são inúmeros. Como abordado por Dobnik [Dobnik2004], do ponto de vista da área médico, em sua maioria também são uma grande fonte de treinamento em coordenação motora e visual, como comprovado em um estudo que resultou em uma melhora considerável em habilidade em laparoscopia em médicos com treinamento em videogames.

No artigo "Uma revisão bibliográfica sobre a utilização do Nintendo Wii como instrumento terapêutico e seus fatores de risco", Souza [Souza2011] teve uma conclusão positiva na utilização do Nintendo Wii em reabilitações motoras apesar das poucas pesquisas realizadas na época. Algo que deve ser levado em consideração é que a maioria dos trabalhos científicos, focados no uso dos jogos em terapias, se concentra no Nintendo Wii, e em sua maioria utilizam também o acessório WiiFit[5]. No artigo "Análise do equilíbrio em pacientes hemiparéticos após o treino com o programa Wii Fit" (assumo que o nome se baseie no programa de fisioterapia utilizando o acessório que foi abordado no artigo), Barcala, Colella, Araujo, Salgado e Oliveira [Barcala2015], em uma comparação entre pacientes sofrendo hemiparesia (paralisia branda de um dos lados do corpo) concluiu que "a fisioterapia associada ao treino de equilíbrio com o Wii Fit apresenta resultados significantes na reabilitação dos indivíduos hemiparéticos, obtendo, assim, mais um recurso terapêutico na fisioterapia",

[5]WiiFit é um acessório para o console Nintendo Wii, constituído de uma placa com detectores de pressão para analisar o equilíbrio da pessoa utilizando-o

uma vez que a fisioterapia convencional e uma terapia convencional alternada com treino de equilíbrio utilizando o WiiFit obtiveram resultados similares, com a terapia utilizando o auxílio do WiiFit obtendo um resultado melhor em relação à oscilação ântero-posterior.

Tavares, Carbonero, Finamore e Kós [Tavares2013], no artigo "Uso do nintendo Wii para reabilitação de crianças com paralisia cerebral: estudo de caso", observam que os resultados sugerem que a intervenção com o console Nintendo Wii é eficaz para incremento da função motora grossa em crianças com comprometimento moderado e equilíbrio em pacientes com comprometimento leve, porém, é necessário um estudo com uma população maior para caracterizar o NW como uma ferramenta complementar de reabilitação. Também Wibelinger et al. [Wibelinger2013], no trabalho "Efeitos da fisioterapia convencional e da wiiterapia na dor e capacidade funcional de mulheres idosas com osteoartrite de joelho", observam uma superioridade da wiiterapia na melhora da rigidez e equilíbrio em mulheres idosas portadoras de osteoartrite em relação à fisioterapia convencional.

Os jogos combinando ambientes virtuais e princípios de imersão que gera Realidade Aumentada tem muitas aplicações. Fernandes Meirelles Araújo [Fernandes2015], no trabalho "Desenvolvimento de um sistema de medições livre de marcadores utilizando sensores de profundidade", descreve "a tecnologia também abre portas para um série de novas funcionalidades em outros projetos como: sistemas de reconhecimentos de gestos em diversas áreas, realidade aumentada sem marcadores, modelagem de objetos 3D facilitada, mapeamento de cenários para navegação em robótica e auxílio de medição através de processamento de imagens." Também, a Realidade aumentada aplicada na terapia médica é parte das pesquisas bem sucedidas. Nesta categoria, Schiavinato, Baldan, Melatto e Lima [Schiavinato2010], no artigo "Influência do Wii Fit no equilíbrio de paciente com disfunção cerebelar: estudo de caso", obtiveram como conclusão de que "a realidade virtual oferece melhora do equilíbrio de pacientes com disfunções cerebelares, assim como maior independência para realização das tarefas diárias", com uma melhora no paciente, sofrendo de Ataxia Cerebelar Precoce, de mais de 20% de acordo com a escala de equilíbrio de Berg, que mede a capacidade de equilíbrio, 10% no índice de Barthel, que avalia as atividades da vida diária, e 25% na escala de Lawton, que avalia as atividades da vida prática.

Com a popularização dos jogos online, e consequentemente dos jogos MMOs (Mas-

sive Multiplayer Online), as possibilidades de utilização dos jogos aumentaram ainda mais. MMOs em geral são utilizados para estudos de teorias de modelos econômicos. Eles possuem uma economia interna, em alguns casos altamente desenvolvidos, incluindo bancos e mercados de ações, permitindo situações que não podem ser simuladas devidamente devido ao fator humano, tornando MMOs um novo campo de estudo revolucionário para economistas [Plumer 2012]. MMOs também permitem aos jogadores ganhar experiência em liderança. Os desafios organizacionais e estratégicos são similares aos de líderes em empresas, mas possuem ainda mais dificuldades, pois os outros membros da equipe são voluntários e o ambiente é fluido [Reeves2008].

Capítulo 3

Desenvolvimento e Implementação

Tendo a ideia inicial do que desejamos, é preciso decidir o que utilizar para torna-la realidade, mesmo que virtual. No caso em questão, é necessário um controlador por sensores de movimentos físicos, um motor de desenvolvimento e um pacote de conversão de dados para que o motor de desenvolvimento possa utilizar os comandos recebidos pelo sensor.

3.1 Motor de Desenvolvimento de Jogos

O motor utilizado para este trabalho foi o Unity3D. Sua interface é gráfica, facilitando o desenvolvimento das partes menos relevantes do trabalho. Possui muitos recursos para uso, grátis ou de baixo custo, e é de fácil aprendizado. Também suporta as linguagens de programação C# e Javascript, o que permite uma maior liberdade em programações adicionais. O Kinect envia os eventos capturados, manipuladas por SKD, ao motor para análise dos movimentos e consequente ativação do avatar.

3.1.1 Sensor de Movimentos

O Kinect foi considerado a melhor opção para o projeto, pois sua capacidade de detecção de movimento apenas pelo aspecto visual está muito à frente de outros dispositivos do tipo.

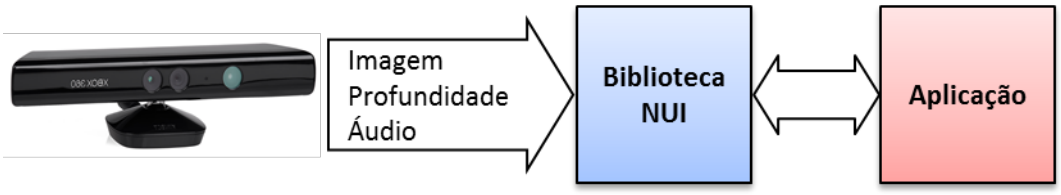


Figura 3.1: Arquitetura de funcionamento de Kinect.

A detecção de movimento pelo aspecto visual é uma grande parte do projeto, pois métodos utilizando sensores físicos, como o Wiimote (controle do console Nintendo Wii), limitariam ou a capacidade de leitura dos comandos ou a liberdade do jogador (no caso mencionado, o Wiimote apenas detecta sua própria posição e dados recebidos de acessórios que possuem suas próprias limitações).

O sistema Real-Time Human Pose Recognition do Kinect foi desenvolvido em base de uma grande quantidade de dados de capturas de movimentos, em cenários da vida real, utilizando um sistema de aprendizado de máquina. Este foi capaz de mapear os dados para modelos que representavam pessoas em fisionomias diferentes. Com estes dados, os pesquisadores conseguiram ensinar o sistema a detectar milhares de tipos de poses humanas, mapeando-as para um esqueleto tridimensional [Shotton2010]. Esse sistema permite ao desenvolvedor de aplicações com detecção de movimento uma grande facilidade na geração de movimentos dos avatares, representações gráficas dos jogadores.

Os dados referentes ao reconhecimento de jogadores e de seus esqueletos são processados diretamente no hardware do Kinect, sem a necessidade de um software específico para trazer esta inovação ao computador ou ao XBOX.

A câmera de vídeo de Kinect captura cenas com resolução VGA a uma velocidade 30 quadros por segundo. O software rastreia quarenta e oito pontos do corpo humano de cada jogador à frente do aparelho. A técnica desenvolvida considerando que todos os simples movimentos do corpo humano fossem entradas para o dispositivo, assim o programa não combina todos os movimentos, mas sim permite o sistema como reagir.

O kinect se comunica com o computador e com a aplicação por meio de drivers e de uma biblioteca de funções, chamada Natural User Interface (NUI) Library, tal como ilustrada na Figura 3.1.

3.1.2 Pacote de Conversão

O Kinect for Windows SDK é um pacote de desenvolvimento de softwares liberado pela própria Microsoft para gerar aplicativos com acesso ao dispositivo Kinect. Este pacote inclui drivers, acesso aos dados do sensor de profundidade e da câmera colorida RGB, funções para movimentação do motor, funções para rastreamento de corpos humanos, documentação e exemplos de aplicações. Este pacote permite aos desenvolvedores criarem seus aplicativos utilizando as linguagens de programação C++, C# ou Visual Basic. O SDK não fornece nenhum algoritmo de visão computacional ou processamento de imagens para serem aplicados nos dados adquiridos pelos sensores do Kinect.

No entanto este pacote é genérico, e lhe faltam as conversões necessárias para que os dados recebidos sejam utilizados pelo motor de desenvolvimento.

Inicialmente, seriam utilizados o pacote desenvolvido pelo time Zigfu, ou o Kinect Wrapper Package. No entanto, eles se mostraram desatualizados com a capacidade do Kinect, e de difícil entendimento.

Consequentemente, o pacote utilizado foi Kinect with MS-SDK, por RF Solutions. Ele é o mais atualizado no momento em que este trabalho está sendo escrito, e as demonstrações que vêm com ele se mostraram de fácil entendimento.

A linguagem de programação em comum entre o Kinect for Windows SDK e o motor de jogo Unity é C#, e assim a linguagem utilizada na programação do trabalho.

3.2 Elementos do Jogo

O jogo criado para este trabalho tem como base um jogo de plataformas no espaço bidimensional. No ambiente de jogo estão, aparte das paredes limitantes e plataformas, um avatar, uma escada, uma chave e uma porta. O cenário é que o avatar, movimentado por movimentos e gestos do corpo do jogador humano, deseja sair do ambiente fechado, um salão de um castelo. A Figura 3.2 ilustra o ambiente do cenário do jogo.

A porta do ambiente esta trancada, para escapar o avatar deve buscar a chave que

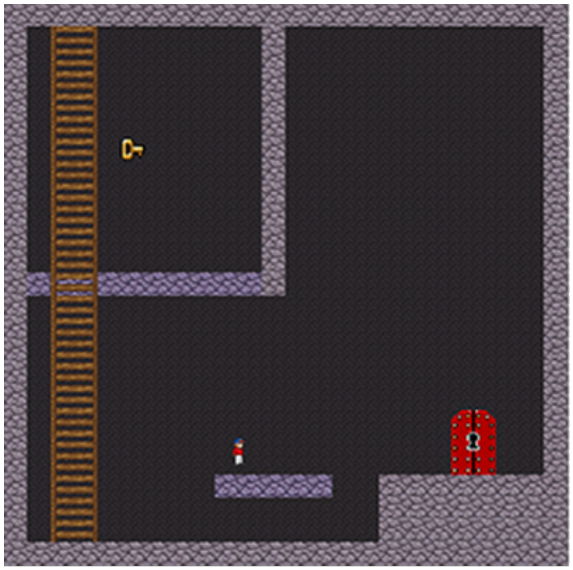


Figura 3.2: Nível inicial do jogo, que requer utilizar todos os comandos mas não muita habilidade.

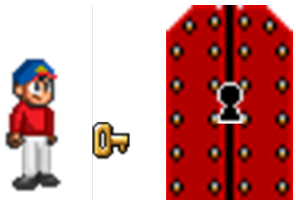


Figura 3.3: Avatar, Chave, Porta trancada.

está em um cômodo superior. Para tal, o avatar deve andar e pular para chagar para a escada, subir pela escada até o cômodo superior, e uma vez pega a chave o avatar não tem outra saída que descer pela escada, aproximar-se da porta e a abrir com a chave (Figura 3.3 ilustra o avatar com a chave na porta do ambiente), e começar outras etapas de escape do castelo até que finalmente tenha conseguido escapar.

O avatar responde aos movimentos do jogador, não os repetindo fielmente, mas os entendendo como comandos. Ele é controlado por uma máquina de estados finita, um comando em certo estado pode ou ser ignorado ou produzir uma resposta diferente da de outro estado.

O ambiente de jogo foi gerado, em sua maioria, internamente no Unity, mas os gráficos e efeitos sonoros foram criados externamente, em blocos, e depois utilizados pelo programa. Os gráficos, com excessão da chave e porta que vieram do jogo Super Mario World da Nintendo, eram originalmente parte do pacote padrão do motor de

desenvolvimento de jogos RPG Maker, desenvolvido pela empresa japonesa Kadokawa e distribuído pela empresa Degica, e editados em um editor de imagens que permite o uso de transparência, nest caso em específico o Adobe Photoshop. Os efeitos sonoros foram utilizados como vieram do pacote do RPG Maker, sem edição.

O avatar foi gerado no mesmo editor de imagens, e foi de desenvolvimento original, mas as imagens do personagem principal do jogo Megaman 7 da Capcom foram utilizadas como base inspiracional para sua criação e portanto se tornaram semelhantes visualmente.

3.3 Movimentação do Avatar

O avatar é um personagem sintético que representa o jogador humano no ambiente virtual. O usuário ao ser representado pelo avatar no jogo sente a sensação do princípio de imersão no ambiente virtual do jogo, e atua como se ele mesmo estivesse realizando todas as ações do avatar dentro do ambiente de jogo.

Os diferentes movimentos do avatar são uma sequencia de posturas definidas como frames, definidos no momento de design do aplicativo. Estas posturas se encontram em uma biblioteca para serem usadas e exibidas na janela do jogo, quando são acionadas pelos respectivos movimentos do corpo capturados pelo Kinect e identificados os tipos de movimento pelo programa de SDK. Nestas condições, qualquer movimento pode ativar o movimento do avatar, até movimentos sem sequências lógicas. Para evitar esse caso, os movimentos permitidos do avatar seguem a estrutura definida por um sistema de estados, tal como mostrada pela Figura 3.4.

O sistema de estados é representado por um grafo de estados conectados por arestas, que representam os eventos do movimento indicados pelo SDK. Os estados, neste caso, são oito: Parado, Andando, Agachando, Pulando, Virando, Escalando, Subindo, Descendo. Os eventos são relacionados com esses estados, como Andar, Agachar, Virar, Pular, Levantar, Subir e Descer gerados pelos comandos do jogador e Tocar Chão e Tocar Escada, gerados pelo posicionamento do avatar na área de jogo. Pela simplicidade no jogo neste trabalho, como um caso exemplo de jogo fisicamente interativo, não existem estados gerados por outros comandos. O jogador/controlador (J), ao produzir um evento com um movimento permitido, faz os comandos que

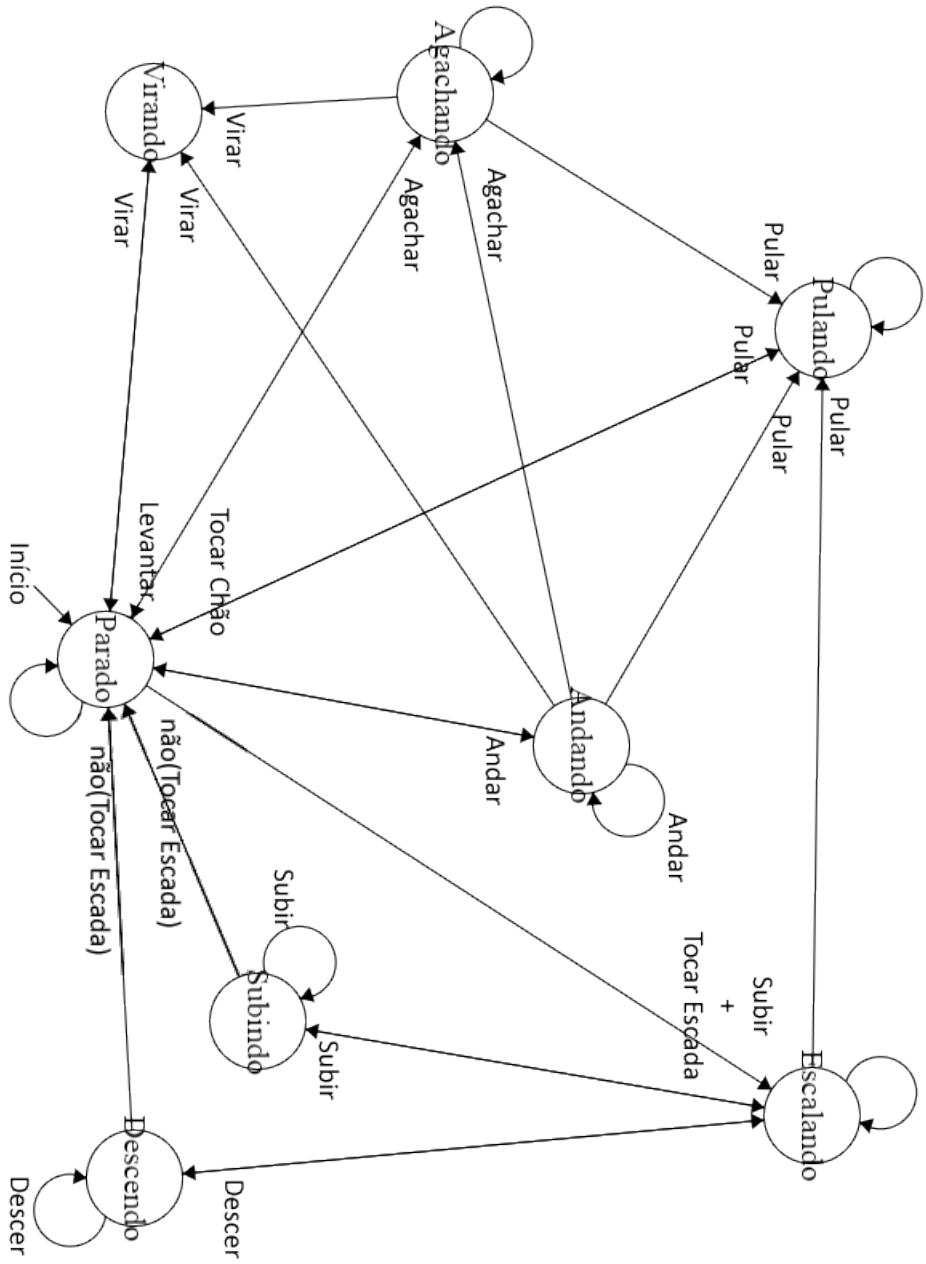


Figura 3.4: Máquina de estados dos movimentos permitidos do avatar no ambiente de jogo.

alteram estes estados.

Os eventos e estados, segundo o sistema de estados da Figura 3.5, são descritos a seguir:

A) Parado (o estado inicial): sem movimentação

- Se repete infinitamente na falta de outros comandos
- Caso J execute um movimento de Andar, troca de estado para Andando
- Caso J execute um movimento de Agachar, troca de estado para Agachando
- Caso J execute um movimento de Pular, troca de estado para Pulando
- Caso J execute um movimento de virar, troca de de estado para Virando
- Caso J execute um movimento de Subir ao mesmo tempo que o avatar realiza o evento Tocar Escada, troca de estado para Escalando
- Caso o avatar deixe de realizar o evento Tocar Chão, troca de estado para Pulando

B) Andando: avatar se move adiante

- Caso J pare de executar movimento de Andar, troca de estado para Parado
- Caso J continue executando o movimento de Andar, o estado se repete
- Caso J execute um movimento de Virar, troca de de estado para Vira
- Caso J execute um movimento de Subir ao mesmo tempo que o avatar realiza o evento Tocar Escada, troca de estado para Escalando
- Caso o avatar deixe de realizar o evento Tocar Chão, troca de estado para Pulando

C) Agachando: avatar se abaixa, permitindo usar o modo de pulo alto

- Se repete infinitamente na falta de outros comandos
- Caso J execute um movimento de Levantar, troca de estado para Parado
- Caso J execute um movimento de Pular, troca de estado para Pulando
- Caso J execute um movimento de Virar, troca de estado para Virando

D) Pulando: caso avatar estiver anteriormente no estado Agachando, sobe verticalmente antes da gravidade simulada o puxar novamente para baixo, caso contrário e o avatar estiver realizando o evento Tocar Chão, se move adiante em um arco, permitindo passar por locais em que não é possível se mover com o estado Andando. Em caso negativo para ambos, apenas ocorre a mudança gráfica até que o avatar mude de estado.

- Caso o avatar realize o evento Tocar Chão, troca de estado para Parado

E) Virando: avatar se inverte no eixo x, e os movimentos dos estados Andando e Pulando passam a mover o avatar na direção oposta

- Troca de estado automaticamente para Parado

F) Escalando: avatar se torna imume aos efeitos da gravidade simulada

- Se repete infinitamente na falta de outros comandos.
- Caso J execute um movimento de Pular, troca de estado para Pulando.
- Caso J execute um movimento de Subir, troca de estado para Subindo.
- Caso J execute um movimento de Descer, troca de estado para Descendo.

G) Sobe: avatar se move para cima

- Troca de estado automaticamente para Escalando
- Caso J continue executando o movimento de Subir, o estado se repete
- Caso o avatar deixe de realizar o evento Tocar Escada, troca de evento para Parado

H) Desce: avatar se move para baixo

- Troca de estado automaticamente para Escalando
- Caso J continue executando o movimento de Descer, o estado se repete
- Caso o avatar deixe de realizar o evento Tocar Escada, troca de evento para Parado

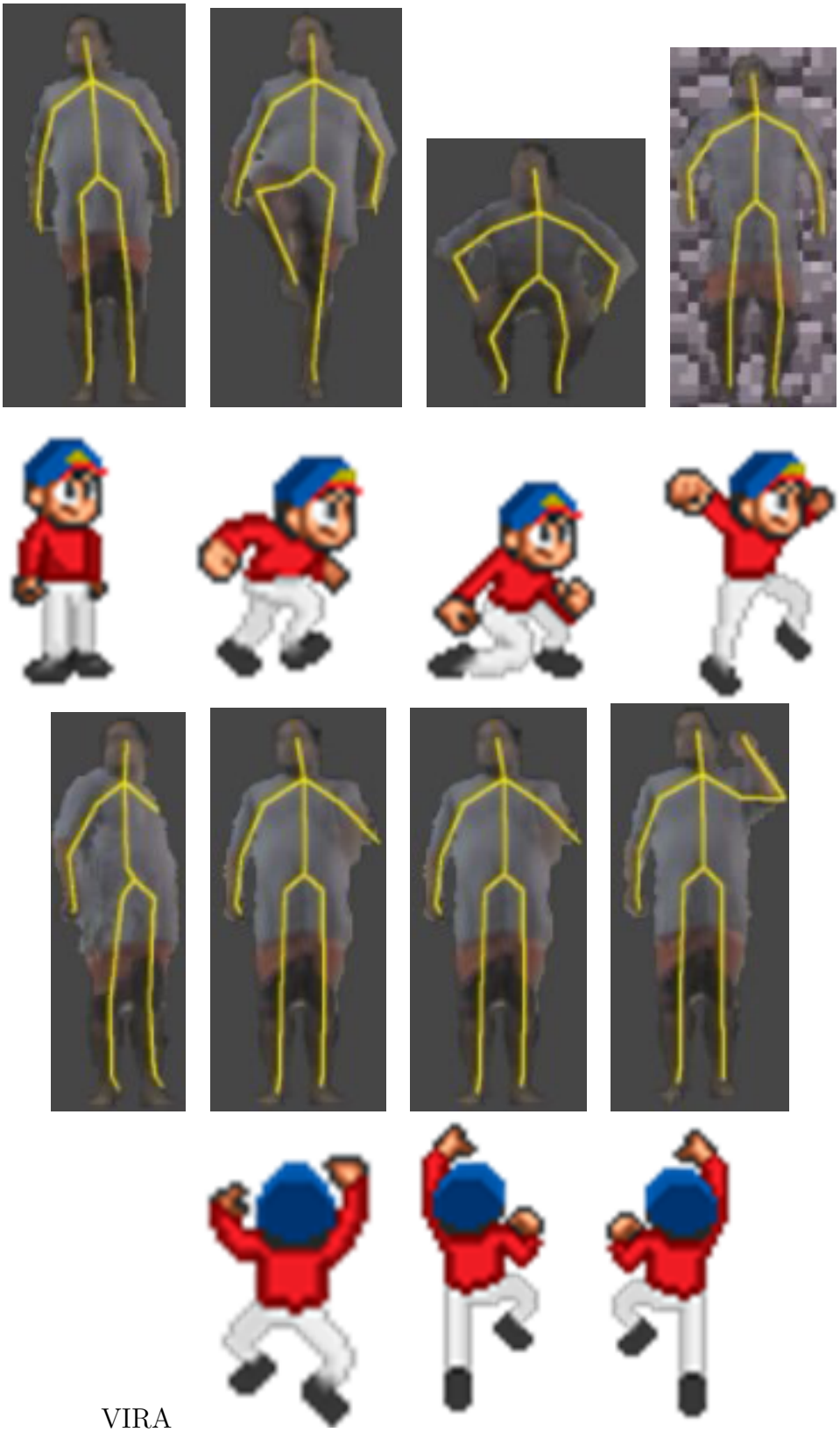


Figura 3.5: Exemplo de movimentos permitidos e as respectivas posturas do avatar.

3.4 Conversão dos Movimentos em Comandos

O Kinect, em geral, calcula um esqueleto através da imagem detectada, e atribui seus pontos de ligamento aos pontos-chave equivalentes do corpo.

No entanto, este método permite um número quase infinito de posições para os pontos de ligamento, o que se torna ainda pior levando em consideração a imprecisão inerente ao ser humano utilizado como controlador.

Portanto, o cálculo de comandos é calculado através não de movimentos precisos, mas de comparações entre a localização de certos pontos de ligação em uma área a certa distância em comparação a outros, seja por um espaço de tempo para calcular falta de movimentos temporária, ou em estágios de localização dentro de um prazo-limite entre localizações para calcular movimentação.

Por exemplo, para selecionar um objeto são possíveis dois comandos distintos: o primeiro, manter um ícone de seleção que segue as coordenadas x e y da mão sobre a área de detecção do objeto por um prazo de tempo, ou calcular a diferença da distância inicial e final da mão, mantendo o ícone sobre a área do objeto. Em ambos os casos, a posição da mão é calculada comparando-a com a posição do respectivo ombro.

Deve se ter em mente que, apesar de ser o comum em projetos utilizando o Kinect, nada lhe obriga realmente a ter o avatar, uma representação gráfica do usuário, copiando fielmente os movimentos detectados. De fato, os sistemas de cópia de movimento dos avatares e de captura de comandos são distintos.

Neste caso em específico são utilizados quatro scripts diferentes para a leitura dos comandos:

- KinectGestures.cs, lê os dados recebidos pelo Kinect e verifica se ativam comandos pré-programados através de etapas (múltiplas verificações para verificar se o posicionamento de certo ponto no corpo do jogador encontra-se dentro de áreas pré-definidas dentro de certa medida de tempo, caso o resultado seja inválido em alguma verificação esta retorna ao início)
- GestureListener.cs, transforma os dados recebidos pelo KinectGestures.cs e os transporta para variáveis exclusivas para cada comando

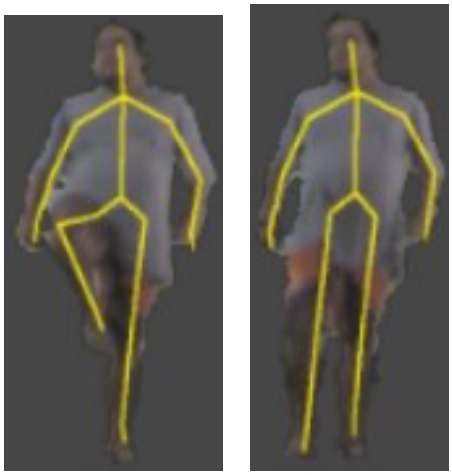


Figura 3.6: Movimento das pernas para Andar.

- ShoulderCheck.cs, exclusivo do movimento de virar, verifica o ângulo entre os ombros
- AvatarMove.cs, utiliza os dados recebidos pelo GestureListener.cs e ShoulderCheck.cs e os utiliza em comandos no jogo em si.

3.4.1 Movimento de Andar

Apesar de apresentar a capacidade para ser utilizado no corpo inteiro, o Kinect é geralmente utilizado apenas para detectar movimentos da parte superior do mesmo. Há exceções, mas elas são, em geral, jogos de dança em que os movimentos são comparados com outros pré-programados, e não transformados em comandos.

Para calcular o movimento de andar, foi comparada a posição vertical do pé com a do oposto. O movimento é considerado iniciado quando um pé está a mais de 5cm acima do outro, e terminado - e assim confirmado - quando o primeiro é abaixado para menos de 4cm acima e se mantendo em uma área de raio 6 cm da posição inicial no plano formado pelos outros dois eixos. 3.6.

Para evitar que o movimento simulado de andar seja substituído por apenas levantar e abaixar a mesma perna repetidamente, o comando apenas é aceito ou em pernas alternadas, ou com a mesma perna após 3 segundos do comando ser aceito pela última vez.

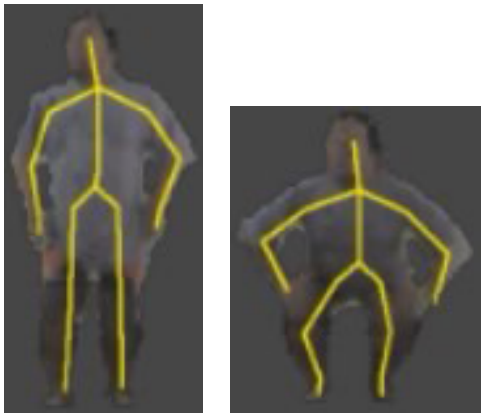


Figura 3.7: Movimento para Agachar.

3.4.2 Movimentos de Agachar e Pular/Levantar

Para detectar o movimento de agachar, apenas é necessário se detectar que o centro da cintura se moveu o bastante (aproximadamente 20cm) para baixo (Figura 3.7 ilustra as duas posturas de Agachar).

Para detectar o movimento de pular, apenas é necessário detectar que a mesma se moveu o bastante para cima (aproximadamente 10cm).

No entanto, isto apresentou problemas. A descida do pulo contar como um agachar caso grande o bastante foi simples de resolver, fazendo o comando de agachar apenas ser aceito caso o avatar não esteja no estado Pulando. Mas como no trabalho foram utilizados pulos diferentes para quando se inicia no estado Agachando e fora dele, como dito na explicação do estado Pulando, o comando Pular é necessário mesmo quando o avatar se encontra do estado Agachando.

Com isto, foi utilizada uma medida de tempo utilizando-se do fato de os comandos de movimento do Kinect serem calculados por etapas. A etapa inicial do pulo é simplesmente a cintura se localizar entre 90cm e 1,3m de altura, enquanto a segunda é a diferença do posicionamento da cintura. Caso o controlador detecte a segunda etapa dentro de um prazo de tempo abaixo de aproximadamente 15 segundos, contará como um Pular quando agachado. Caso a detecte em um prazo de tempo maior, será contado como um comando em separado, Levantar. As duas posturas ilustrada pela Figura 3.8 ilustra o movimento de pular.

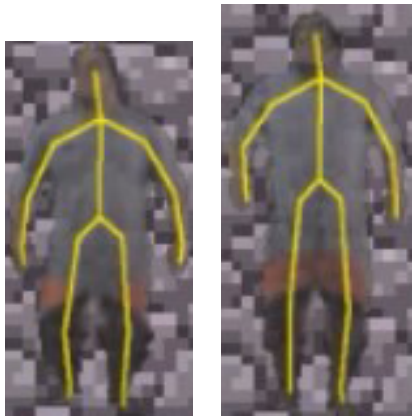


Figura 3.8: Movimento para Pular/Levantar.

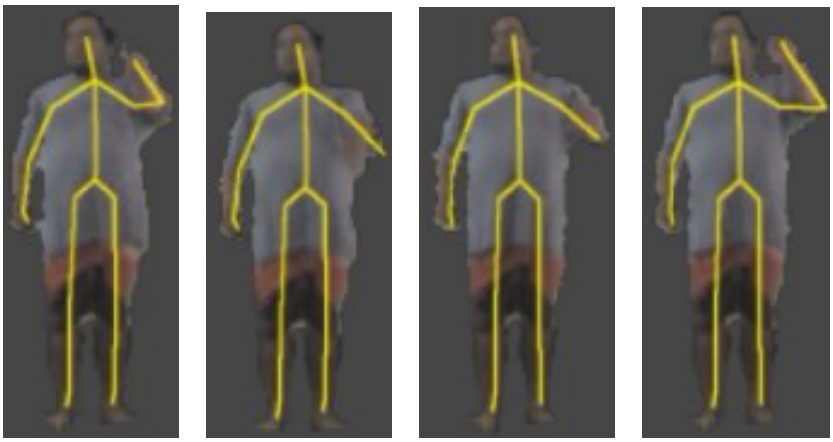


Figura 3.9: Movimento para Subir (dois da esquerda) e para Descer (dois da direita).

3.4.3 Movimentos para Subir e Descer

Para a movimentação em escadas foram utilizados movimentos baseados nos comandos padrão Swipe Up e Swipe Down (movimento com os braços atravessando a área na frente do corpo verticalmente para cima e para baixo, respectivamente) para se conseguir uma movimentação simulando mais fielmente uma pessoa realmente subindo e descendo uma escada com as mãos (Figura 3.9). Para tal, foi utilizado como ponto inicial do movimento uma área próxima do ombro do lado da mesma mão (uma diferença aceitável de até 20cm horizontalmente e 10cm verticalmente) e como ponto final pelo menos 15cm de diferença vertical (não passando de uma diferença posicional de 10cm horizontal) entre a mão e o ombro, seja para baixo para o comando Subir ou para cima para o comando Descer.

Assim como o movimento de andar, os movimentos dos braços devem ser alterna-

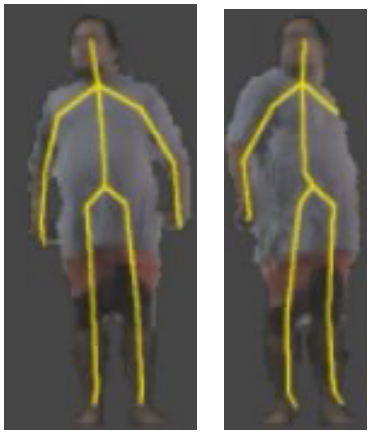


Figura 3.10: Gestos para Virar.

dos ou ter uma pausa de 3 segundos entre eles.

3.4.4 Movimentação dos Ombros para Virar

Para a simular a rotação do personagem, ele se virar foi atribuído a um movimento de ombros do controlador. Isto foi feito utilizando o script ShoulderCheck.cs, que calculan o ângulo entre os ombros e, caso esteja entre 30Â° e o limite de leitura do Kinect de aproximadamente 80Â° para o lado oposto do que o Avatar está, o personagem se vira. Figura 3.10 ilustra a movimento de virar.

Capítulo 4

Conclusões e Trabalhos Futuros

Este trabalho apresentou a implementação de um programa iterativo fisicamente através de movimentos do corpo humano em tempo real, utilizando o dispositivo Kinect para captura de informações do ambiente para que este pudesse ser segmentado. Por ser tratar de um programa fisicamente interativo o usuário jogador utiliza movimentos do corpo, algo importante para evitar sedentarismos, em particular para crianças e adolescentes que ficam tempo todo nos jogos eletrônicos.

A parte mais trabalhosa da criação deste trabalho foi o desenvolvimento dos comandos de detecção de movimento para serem calculados pelo Kinect (todos os comandos utilizados no trabalho com excessão de Agachar e Pular/Levantar foram criados exclusivamente para o mesmo, apesar de utilizarem comandos já existentes como uma base). Eles precisam ser precisos para não serem detectados erroneamente (ou para que não sejam ignorados), mas ao mesmo tempo abrangentes o bastante para poderem ser realizados por movimentos de seres humanos, que sempre possuem um certo grau de imprecisão. Isto sem levar em conta que, por mais avançado que o Kinect seja, não é um aparelho perfeito, uma versão mais avançada e precisa já tendo sido lançada.

Para melhoras futuras, um dos primeiros passos, seria o desenvolvimento de níveis mais desafiadores que demandariam um maior esforço físico e mental do jogador.

Mas outro passo importante seria uma melhoria na programação de detecção destes comandos. Apesar de funcionais, não estão funcionando com perfeição o tempo todo.

Outra melhoria seria a inclusão de novos comandos, o que permitiria mais movimentos para o avatar e a utilização dos mesmos pelos desafios do jogo.

Eventualmente, o jogo poderia ser adaptado para uma versão tridimensional. A maioria dos comandos atuais poderia ser adaptada sem muitos problemas para tal, e novos comandos poderiam ser criados para o que mais for necessário.

Uma adaptação para a utilização da nova versão do Kinect, caso necessário, ou até mesmo suas versões alternativas lançadas posteriormente, também seria uma boa adição.

Apêndice A

Códigos

A.1 Código KinectGestures.cs

Código do arquivo .cs que recebe os dados do sensor Kinect e, através de etapas, os transforma em variáveis. A gesture Jump foi a única alterada, e as gestures Walk, PullUp e PullDown foram criadas para este trabalho. Apenas Jump, Squat, Walk, PullUp e PullDown foram utilizadas.

É a ponte entre os dados do Kinect e dados utilizáveis pelo motor de jogo.

```
using UnityEngine;
using System.Collections;
using System.Collections.Generic;

public class KinectGestures
{

public interface GestureListenerInterface
{
// Invoked when a new user is detected and tracking starts
// Here you can start gesture detection with KinectManager.DetectGesture()
void UserDetected(uint userId, int userIndex);

// Invoked when a user is lost
// Gestures for this user are cleared automatically, but you can free the used resources
void UserLost(uint userId, int userIndex);

// Invoked when a gesture is in progress
void GestureInProgress(uint userId, int userIndex, Gestures gesture, float progress,
                       KinectWrapper.NuiSkeletonPositionIndex joint, Vector3 screenPos);
```

```
// Invoked if a gesture is completed.
// Returns true, if the gesture detection must be restarted, false otherwise
bool GestureCompleted(uint userId, int userIndex, Gestures gesture,
                      KinectWrapper.NuiSkeletonPositionIndex joint, Vector3 screenPos);

// Invoked if a gesture is cancelled.
// Returns true, if the gesture detection must be retarted, false otherwise
bool GestureCancelled(uint userId, int userIndex, Gestures gesture,
                      KinectWrapper.NuiSkeletonPositionIndex joint);
}


public enum Gestures
{
None = 0,
RaiseRightHand,
RaiseLeftHand,
Psi,
Tpose,
Stop,
Wave,
Click,
SwipeLeft,
SwipeRight,
SwipeUp,
SwipeDown,
RightHandCursor,
LeftHandCursor,
ZoomOut,
ZoomIn,
Wheel,
Jump,
Squat,
Push,
Pull,
Walk,
PullUp,
PullDown
}


public struct GestureData
{
public uint userId;
public Gestures gesture;
public int state;
public float timestamp;
public int joint;
public Vector3 jointPos;
public Vector3 screenPos;
public float tagFloat;
public Vector3 tagVector;
public Vector3 tagVector2;
```

```
public float progress;
public bool complete;
public bool cancelled;
public List<Gestures> checkForGestures;
public float startTrackingAtTime;
public int footfirst;
public int handfirst;
public float timeofgesture;
}



// Gesture related constants, variables and functions
private const int leftHandIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.HandLeft;
private const int rightHandIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.HandRight;

private const int leftElbowIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.ElbowLeft;
private const int rightElbowIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.ElbowRight;

private const int leftShoulderIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderLeft;
private const int rightShoulderIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderRight;

private const int hipCenterIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.HipCenter;
private const int shoulderCenterIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderCenter;
private const int leftHipIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.HipLeft;
private const int rightHipIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.HipRight;

private const int leftKneeIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.KneeLeft;
private const int rightKneeIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.KneeRight;

private const int leftFootIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.FootLeft;
private const int rightFootIndex = (int)KinectWrapper.NuiSkeletonPositionIndex.FootRight;



private static void SetGestureJoint(ref GestureData gestureData, float timestamp, int joint, Vector3 jointPos)
{
gestureData.joint = joint;
gestureData.jointPos = jointPos;
gestureData.timestamp = timestamp;
gestureData.state++;
}

private static void SetGestureCancelled(ref GestureData gestureData)
{
gestureData.state = 0;
gestureData.progress = 0f;
gestureData.footfirst = 0;
gestureData.handfirst = 0;
gestureData.cancelled = true;
}

private static void CheckPoseComplete(ref GestureData gestureData, float timestamp, Vector3 jointPos,
```

```
        bool isInPose, float durationToComplete)
{
if(isInPose)
{
float timeLeft = timestamp - gestureData.timestamp;
gestureData.progress = durationToComplete > 0f ? Mathf.Clamp01(timeLeft / durationToComplete) : 1.0f;

if(timeLeft >= durationToComplete)
{
gestureData.timestamp = timestamp;
gestureData.jointPos = jointPos;
gestureData.state++;
gestureData.complete = true;
}
}
else
{
SetGestureCancelled(ref gestureData);
}
}

private static void SetScreenPos(uint userId, ref GestureData gestureData, ref Vector3[] jointsPos,
        ref bool[] jointsTracked)
{
Vector3 handPos = jointsPos[rightHandIndex];
bool calculateCoords = false;

if(gestureData.joint == rightHandIndex)
{
if(jointsTracked[rightHandIndex])
{
calculateCoords = true;
}
}
else if(gestureData.joint == leftHandIndex)
{
if(jointsTracked[leftHandIndex])
{
handPos = jointsPos[leftHandIndex];

calculateCoords = true;
}
}

if(calculateCoords)
{

if(jointsTracked[hipCenterIndex] && jointsTracked[shoulderCenterIndex] &&
jointsTracked[leftShoulderIndex] && jointsTracked[rightShoulderIndex])
{
Vector3 neckToHips = jointsPos[shoulderCenterIndex] - jointsPos[hipCenterIndex];
Vector3 rightToLeft = jointsPos[rightShoulderIndex] - jointsPos[leftShoulderIndex];
```

```
gestureData.tagVector2.x = rightToLeft.x; // * 1.2f;
gestureData.tagVector2.y = neckToHips.y; // * 1.2f;

if(gestureData.joint == rightHandIndex)
{
gestureData.tagVector.x = jointsPos[rightShoulderIndex].x - gestureData.tagVector2.x / 2;
gestureData.tagVector.y = jointsPos[hipCenterIndex].y;
}
else
{
gestureData.tagVector.x = jointsPos[leftShoulderIndex].x - gestureData.tagVector2.x / 2;
gestureData.tagVector.y = jointsPos[hipCenterIndex].y;
}
}

if(gestureData.tagVector2.x != 0 && gestureData.tagVector2.y != 0)
{
Vector3 relHandPos = handPos - gestureData.tagVector;
gestureData.screenPos.x = Mathf.Clamp01(relHandPos.x / gestureData.tagVector2.x);
gestureData.screenPos.y = Mathf.Clamp01(relHandPos.y / gestureData.tagVector2.y);
}

}
}

private static void SetZoomFactor(uint userId, ref GestureData gestureData, float initialZoom,
        ref Vector3[] jointsPos, ref bool[] jointsTracked)
{
Vector3 vectorZooming = jointsPos[rightHandIndex] - jointsPos[leftHandIndex];

if(gestureData.tagFloat == 0f || gestureData.userId != userId)
{
gestureData.tagFloat = 0.5f; // this is 100%
}

float distZooming = vectorZooming.magnitude;
gestureData.screenPos.z = initialZoom + (distZooming / gestureData.tagFloat);
}

// estimate the next state and completeness of the gesture
public static void CheckForGesture(uint userId, ref GestureData gestureData, float timestamp,
        ref Vector3[] jointsPos, ref bool[] jointsTracked)
{
if(gestureData.complete)
return;

float bandSize = (jointsPos[shoulderCenterIndex].y - jointsPos[hipCenterIndex].y);
float gestureTop = jointsPos[shoulderCenterIndex].y + bandSize / 2;
float gestureBottom = jointsPos[shoulderCenterIndex].y - bandSize;
float gestureRight = jointsPos[rightHipIndex].x;
float gestureLeft = jointsPos[leftHipIndex].x;

switch(gestureData.gesture)
```

```
{
// check for RaiseRightHand
case Gestures.RaiseRightHand:
switch(gestureData.state)
{
case 0:  // gesture detection
if(jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
}
break;

case 1:  // gesture complete
bool isInPose = jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.1f;

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                    KinectWrapper.Constants.PoseCompleteDuration);
break;
}
break;

// check for RaiseLeftHand
case Gestures.RaiseLeftHand:
switch(gestureData.state)
{
case 0:  // gesture detection
if(jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
}
break;

case 1:  // gesture complete
bool isInPose = jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.1f;

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                    KinectWrapper.Constants.PoseCompleteDuration);
break;
}
break;

// check for Psi
case Gestures.Psi:
switch(gestureData.state)
{
case 0:  // gesture detection
if(jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
```

```
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.1f &&
jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
}
break;

case 1:  // gesture complete
bool isInPose = jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.1f &&
jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.1f;

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                  KinectWrapper.Constants.PoseCompleteDuration);
break;
}
break;

// check for Tpose
case Gestures.Tpose:
switch(gestureData.state)
{
case 0:  // gesture detection
if(jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
                  jointsTracked[rightShoulderIndex] &&
Mathf.Abs(jointsPos[rightElbowIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f &&  // 0.07f
Mathf.Abs(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f &&  // 0.7f
jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
                  jointsTracked[leftShoulderIndex] &&
Mathf.Abs(jointsPos[leftElbowIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
}
break;

case 1:  // gesture complete
bool isInPose = jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
                  jointsTracked[rightShoulderIndex] &&
Mathf.Abs(jointsPos[rightElbowIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f &&  // 0.7f
Mathf.Abs(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f &&  // 0.7f
jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
                  jointsTracked[leftShoulderIndex] &&
Mathf.Abs(jointsPos[leftElbowIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f;

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                  KinectWrapper.Constants.PoseCompleteDuration);
break;
```

```
}
break;

// check for Stop
case Gestures.Stop:
switch(gestureData.state)
{
case 0:  // gesture detection
if(jointsTracked[rightHandIndex] && jointsTracked[rightHipIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightHipIndex].y) < 0.1f &&
(jointsPos[rightHandIndex].x - jointsPos[rightHipIndex].x) >= 0.4f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
}
else if(jointsTracked[leftHandIndex] && jointsTracked[leftHipIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftHipIndex].y) < 0.1f &&
(jointsPos[leftHandIndex].x - jointsPos[leftHipIndex].x) <= -0.4f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
}
break;

case 1:  // gesture complete
bool isInPose = (gestureData.joint == rightHandIndex) ?
(jointsTracked[rightHandIndex] && jointsTracked[rightHipIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightHipIndex].y) < 0.1f &&
(jointsPos[rightHandIndex].x - jointsPos[rightHipIndex].x) >= 0.4f) :
(jointsTracked[leftHandIndex] && jointsTracked[leftHipIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftHipIndex].y) < 0.1f &&
(jointsPos[leftHandIndex].x - jointsPos[leftHipIndex].x) <= -0.4f);

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                  KinectWrapper.Constants.PoseCompleteDuration);
break;

}
break;

// check for Wave
case Gestures.Wave:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightElbowIndex].y) > 0.1f &&
(jointsPos[rightHandIndex].x - jointsPos[rightElbowIndex].x) > 0.05f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.3f;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftElbowIndex].y) > 0.1f &&
```

```
(jointsPos[leftHandIndex].x - jointsPos[leftElbowIndex].x) < -0.05f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.3f;
}
break;

case 1:  // gesture - phase 2
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightElbowIndex].y) > 0.1f &&
(jointsPos[rightHandIndex].x - jointsPos[rightElbowIndex].x) < -0.05f :
jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftElbowIndex].y) > 0.1f &&
(jointsPos[leftHandIndex].x - jointsPos[leftElbowIndex].x) > 0.05f;

if(isInPose)
{
gestureData.timestamp = timestamp;
gestureData.state++;
gestureData.progress = 0.7f;
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;

case 2:  // gesture phase 3 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightElbowIndex].y) > 0.1f &&
(jointsPos[rightHandIndex].x - jointsPos[rightElbowIndex].x) > 0.05f :
jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftElbowIndex].y) > 0.1f &&
(jointsPos[leftHandIndex].x - jointsPos[leftElbowIndex].x) < -0.05f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
```

```
}
break;
}
break;

// check for Click
case Gestures.Click:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightElbowIndex].y) > -0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.3f;

// set screen position at the start, because this is the most accurate click position
SetScreenPos(userId, ref gestureData, ref jointsPos, ref jointsTracked);
}
else if(jointsTracked[leftHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftElbowIndex].y) > -0.1f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.3f;

// set screen position at the start, because this is the most accurate click position
SetScreenPos(userId, ref gestureData, ref jointsPos, ref jointsTracked);
}
break;

case 1:  // gesture - phase 2
{
// check for stay-in-place
Vector3 distVector = jointsPos[gestureData.joint] - gestureData.jointPos;
bool isInPose = distVector.magnitude < 0.05f;

Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose,
                        KinectWrapper.Constants.ClickStayDuration);
// SetGestureCancelled(gestureData);
}
break;
}
break;

// check for SwipeLeft
case Gestures.SwipeLeft:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[hipCenterIndex] &&
                    jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                    jointsTracked[rightHipIndex] &&
```

```
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].x <= gestureRight && jointsPos[rightHandIndex].x > gestureLeft)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.1f;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = jointsTracked[rightHandIndex] && jointsTracked[hipCenterIndex] &&
                        jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                        jointsTracked[rightHipIndex] &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].x < gestureLeft;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
else if(jointsPos[rightHandIndex].x <= gestureRight)
{
float gestureSize = gestureRight - gestureLeft;
gestureData.progress = gestureSize > 0.01f ? (gestureRight
                            - jointsPos[rightHandIndex].x) / gestureSize : 0f;
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for SwipeRight
case Gestures.SwipeRight:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[leftHandIndex] && jointsTracked[hipCenterIndex] &&
                    jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                    jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[leftHandIndex].x >= gestureLeft && jointsPos[leftHandIndex].x < gestureRight)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.1f;
}
break;
```

```
case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = jointsTracked[leftHandIndex] && jointsTracked[hipCenterIndex] &&
                        jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                        jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[leftHandIndex].x > gestureRight;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
else if(jointsPos[leftHandIndex].x >= gestureLeft)
{
float gestureSize = gestureRight - gestureLeft;
gestureData.progress = gestureSize > 0.01f ? (jointsPos[leftHandIndex].x
                        - gestureLeft) / gestureSize : 0f;
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for SwipeUp
case Gestures.SwipeUp:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) < 0.0f &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) > -0.15f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) < 0.0f &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) > -0.15f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.5f;
}
break;

case 1:  // gesture phase 2 = complete
```

```
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.05f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) <= 0.1f :
jointsTracked[leftHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.05f &&
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) <= 0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for SwipeDown
case Gestures.SwipeDown:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.05f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.05f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.5f;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) < -0.15f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) <= 0.1f :
jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) < -0.15f &&
```

```
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) <= 0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for RightHandCursor
case Gestures.RightHandCursor:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1 (perpetual)
if(jointsTracked[rightHandIndex] && jointsTracked[rightHipIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightHipIndex].y) > -0.1f)
{
gestureData.joint = rightHandIndex;
gestureData.timestamp = timestamp;
//gestureData.jointPos = jointsPos[rightHandIndex];
SetScreenPos(userId, ref gestureData, ref jointsPos, ref jointsTracked);
gestureData.progress = 0.7f;
}
else
{
// cancel the gesture
//SetGestureCancelled(ref gestureData);
gestureData.progress = 0f;
}
break;

}
break;

// check for LeftHandCursor
case Gestures.LeftHandCursor:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1 (perpetual)
if(jointsTracked[leftHandIndex] && jointsTracked[leftHipIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftHipIndex].y) > -0.1f)
{
gestureData.joint = leftHandIndex;
gestureData.timestamp = timestamp;
//gestureData.jointPos = jointsPos[leftHandIndex];
```

```
SetScreenPos(userId, ref gestureData, ref jointsPos, ref jointsTracked);
gestureData.progress = 0.7f;
}
else
{
// cancel the gesture
//SetGestureCancelled(ref gestureData);
gestureData.progress = 0f;
}
break;

}
break;

// check for ZoomOut
case Gestures.ZoomOut:
Vector3 vectorZoomOut = (Vector3)jointsPos[rightHandIndex] - jointsPos[leftHandIndex];
float distZoomOut = vectorZoomOut.magnitude;

switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] && jointsTracked[hipCenterIndex] &&
                    jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                    jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distZoomOut < 0.3f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.tagVector = Vector3.right;
gestureData.tagFloat = 0f;
gestureData.progress = 0.3f;
}
break;

case 1:  // gesture phase 2 = zooming
if((timestamp - gestureData.timestamp) < 1.5f)
{
float angleZoomOut = Vector3.Angle(gestureData.tagVector, vectorZoomOut) *
                        Mathf.Sign(vectorZoomOut.y - gestureData.tagVector.y);
bool isInPose = jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] &&
                        jointsTracked[hipCenterIndex] && jointsTracked[shoulderCenterIndex] &&
                        jointsTracked[leftHipIndex] && jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distZoomOut < 1.5f && Mathf.Abs(angleZoomOut) < 20f;

if(isInPose)
{
SetZoomFactor(userId, ref gestureData, 1.0f, ref jointsPos, ref jointsTracked);
gestureData.timestamp = timestamp;
gestureData.progress = 0.7f;
```

```
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;

}
break;

// check for ZoomIn
case Gestures.ZoomIn:
Vector3 vectorZoomIn = (Vector3)jointsPos[rightHandIndex] - jointsPos[leftHandIndex];
float distZoomIn = vectorZoomIn.magnitude;

switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] && jointsTracked[hipCenterIndex] &&
                    jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                    jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distZoomIn >= 0.7f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.tagVector = Vector3.right;
gestureData.tagFloat = distZoomIn;
gestureData.progress = 0.3f;
}
break;

case 1:  // gesture phase 2 = zooming
if((timestamp - gestureData.timestamp) < 1.5f)
{
float angleZoomIn = Vector3.Angle(gestureData.tagVector, vectorZoomIn)
                        * Mathf.Sign(vectorZoomIn.y - gestureData.tagVector.y);
bool isInPose = jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] &&
                        jointsTracked[hipCenterIndex] && jointsTracked[shoulderCenterIndex] &&
                        jointsTracked[leftHipIndex] && jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distZoomIn >= 0.2f && Mathf.Abs(angleZoomIn) < 20f;

if(isInPose)
{
SetZoomFactor(userId, ref gestureData, 0.0f, ref jointsPos, ref jointsTracked);
gestureData.timestamp = timestamp;
gestureData.progress = 0.7f;
}
}
```

```
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;

}
break;

// check for Wheel
case Gestures.Wheel:
Vector3 vectorWheel = (Vector3)jointsPos[rightHandIndex] - jointsPos[leftHandIndex];
float distWheel = vectorWheel.magnitude;

switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] && jointsTracked[hipCenterIndex] &&
                    jointsTracked[shoulderCenterIndex] && jointsTracked[leftHipIndex] &&
                    jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distWheel >= 0.3f && distWheel < 0.7f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.tagVector = Vector3.right;
gestureData.tagFloat = distWheel;
gestureData.progress = 0.3f;
}
break;

case 1:  // gesture phase 2 = turning wheel
if((timestamp - gestureData.timestamp) < 1.5f)
{
float angle = Vector3.Angle(gestureData.tagVector, vectorWheel)
                        * Mathf.Sign(vectorWheel.y - gestureData.tagVector.y);
bool isInPose = jointsTracked[leftHandIndex] && jointsTracked[rightHandIndex] &&
                        jointsTracked[hipCenterIndex] && jointsTracked[shoulderCenterIndex] &&
                        jointsTracked[leftHipIndex] && jointsTracked[rightHipIndex] &&
jointsPos[leftHandIndex].y >= gestureBottom && jointsPos[leftHandIndex].y <= gestureTop &&
jointsPos[rightHandIndex].y >= gestureBottom && jointsPos[rightHandIndex].y <= gestureTop &&
distWheel >= 0.3f && distWheel < 0.7f &&
Mathf.Abs(distWheel - gestureData.tagFloat) < 0.1f;

if(isInPose)
{
//SetWheelRotation(userId, ref gestureData, gestureData.tagVector, vectorWheel);
gestureData.screenPos.z = angle;  // wheel angle
gestureData.timestamp = timestamp;
gestureData.tagFloat = distWheel;
gestureData.progress = 0.7f;
}
```

```
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;

}
break;

// check for Jump
case Gestures.Jump:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[hipCenterIndex] &&
(jointsPos[hipCenterIndex].y > 0.9f) && (jointsPos[hipCenterIndex].y < 1.3f))
{
SetGestureJoint(ref gestureData, timestamp, hipCenterIndex, jointsPos[hipCenterIndex]);
gestureData.progress = 0.5f;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 4.5f)
{
gestureData.timeofgesture = (timestamp - gestureData.timestamp);
bool isInPose = jointsTracked[hipCenterIndex] &&
(jointsPos[hipCenterIndex].y - gestureData.jointPos.y) > 0.10f &&
Mathf.Abs(jointsPos[hipCenterIndex].x - gestureData.jointPos.x) < 0.2f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
gestureData.screenPos.z = gestureData.timeofgesture;
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;


// check for Squat
case Gestures.Squat:
switch(gestureData.state)
{
```

```
case 0:  // gesture detection - phase 1
if(jointsTracked[hipCenterIndex] &&
   (jointsPos[hipCenterIndex].y <= 0.9f))
{
SetGestureJoint(ref gestureData, timestamp, hipCenterIndex, jointsPos[hipCenterIndex]);
gestureData.progress = 0.5f;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = jointsTracked[hipCenterIndex] &&
(jointsPos[hipCenterIndex].y - gestureData.jointPos.y) < -0.2f &&
Mathf.Abs(jointsPos[hipCenterIndex].x - gestureData.jointPos.x) < 0.2f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for Push
case Gestures.Push:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
                   jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[rightHandIndex].x - jointsPos[rightShoulderIndex].x) < 0.2f &&
(jointsPos[rightHandIndex].z - jointsPos[leftElbowIndex].z) < -0.2f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
                   jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].x - jointsPos[leftShoulderIndex].x) < 0.2f &&
(jointsPos[leftHandIndex].z - jointsPos[rightElbowIndex].z) < -0.2f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.5f;
```

```
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
                        jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) < 0.2f &&
(jointsPos[rightHandIndex].z - gestureData.jointPos.z) < -0.1f :
jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
                        jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) < 0.2f &&
(jointsPos[leftHandIndex].z - gestureData.jointPos.z) < -0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for Pull
case Gestures.Pull:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
if(jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
                    jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[rightHandIndex].x - jointsPos[rightShoulderIndex].x) < 0.2f &&
(jointsPos[rightHandIndex].z - jointsPos[leftElbowIndex].z) < -0.3f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
                    jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].x - jointsPos[leftShoulderIndex].x) < 0.2f &&
(jointsPos[leftHandIndex].z - jointsPos[rightElbowIndex].z) < -0.3f)
{
```

```
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.5f;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[leftElbowIndex] &&
                        jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[leftElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) < 0.2f &&
(jointsPos[rightHandIndex].z - gestureData.jointPos.z) > 0.1f :
jointsTracked[leftHandIndex] && jointsTracked[rightElbowIndex] &&
                        jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[rightElbowIndex].y) > -0.1f &&
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) < 0.2f &&
(jointsPos[leftHandIndex].z - gestureData.jointPos.z) > 0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for Walk
case Gestures.Walk:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
gestureData.footfirst = 0;
if(jointsTracked[rightFootIndex] && jointsTracked[leftFootIndex] &&
   Mathf.Abs(jointsPos[rightFootIndex].y - jointsPos[leftFootIndex].y) > 0.05f)
{
if ((jointsPos[rightFootIndex].y - jointsPos[leftFootIndex].y) > 0)
{
SetGestureJoint(ref gestureData, timestamp, rightFootIndex, jointsPos[rightFootIndex]);
gestureData.progress = 0.5f;
gestureData.footfirst = 1;
}
else
{
SetGestureJoint(ref gestureData, timestamp, leftFootIndex, jointsPos[leftFootIndex]);
```

```
gestureData.progress = 0.5f;
gestureData.footfirst = 2;
}
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose =
jointsTracked[rightFootIndex] && jointsTracked[leftFootIndex] &&
Mathf.Abs(jointsPos[rightFootIndex].y - jointsPos[leftFootIndex].y) < 0.04f &&
(gestureData.footfirst == 1 ?
 Mathf.Abs(jointsPos[rightFootIndex].x - gestureData.jointPos.x) < 0.06f &&
 Mathf.Abs(jointsPos[rightFootIndex].z - gestureData.jointPos.z) < 0.06f :
 Mathf.Abs(jointsPos[leftFootIndex].x - gestureData.jointPos.x) < 0.06f)&&
 Mathf.Abs(jointsPos[rightFootIndex].z - gestureData.jointPos.z) < 0.06f ;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
gestureData.screenPos.y = gestureData.footfirst;
gestureData.footfirst = 0;
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for PullUp
case Gestures.PullUp:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
gestureData.handfirst = 0;
if(jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
Mathf.Abs(jointsPos[rightHandIndex].x - jointsPos[rightShoulderIndex].x) < 0.2f &&
Mathf.Abs(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
gestureData.handfirst = 1;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
Mathf.Abs(jointsPos[leftHandIndex].x - jointsPos[leftShoulderIndex].x) < 0.2f &&
Mathf.Abs(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f)
{
```

```
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
gestureData.progress = 0.5f;
gestureData.handfirst = 2;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) < -0.15f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) <= 0.1f :
jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) < -0.15f &&
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) <= 0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
gestureData.screenPos.y = gestureData.handfirst;
gestureData.handfirst = 0;
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;

// check for PullDown
case Gestures.PullDown:
switch(gestureData.state)
{
case 0:  // gesture detection - phase 1
gestureData.handfirst = 0;
if(jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
Mathf.Abs(jointsPos[rightHandIndex].x - jointsPos[rightShoulderIndex].x) < 0.2f &&
Mathf.Abs(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) < 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, rightHandIndex, jointsPos[rightHandIndex]);
gestureData.progress = 0.5f;
gestureData.handfirst = 1;
}
else if(jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
Mathf.Abs(jointsPos[leftHandIndex].x - jointsPos[leftShoulderIndex].x) < 0.2f &&
Mathf.Abs(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) < 0.1f)
{
SetGestureJoint(ref gestureData, timestamp, leftHandIndex, jointsPos[leftHandIndex]);
```

```
gestureData.progress = 0.5f;
gestureData.handfirst = 2;
}
break;

case 1:  // gesture phase 2 = complete
if((timestamp - gestureData.timestamp) < 1.5f)
{
bool isInPose = gestureData.joint == rightHandIndex ?
jointsTracked[rightHandIndex] && jointsTracked[rightShoulderIndex] &&
(jointsPos[rightHandIndex].y - jointsPos[rightShoulderIndex].y) > 0.15f &&
Mathf.Abs(jointsPos[rightHandIndex].x - gestureData.jointPos.x) <= 0.1f :
jointsTracked[leftHandIndex] && jointsTracked[leftShoulderIndex] &&
(jointsPos[leftHandIndex].y - jointsPos[leftShoulderIndex].y) > 0.15f &&
Mathf.Abs(jointsPos[leftHandIndex].x - gestureData.jointPos.x) <= 0.1f;

if(isInPose)
{
Vector3 jointPos = jointsPos[gestureData.joint];
gestureData.screenPos.y = gestureData.handfirst;
gestureData.handfirst = 0;
CheckPoseComplete(ref gestureData, timestamp, jointPos, isInPose, 0f);
}
}
else
{
// cancel the gesture
SetGestureCancelled(ref gestureData);
}
break;
}
break;


}
}

}
```

A.2 Código GestureListener.cs

Código do arquivo .cs que transforma os dados recebidos pelo KinectGestures.cs em booleanos para demonstrar que o comando foi acionado e outras variáveis caso sejam necessárias.

Ele é utilizado pois, enquanto KinectGesture é genérico, enviando os dados sempre nos mesmos formatos e variáveis, GestureListener pode converter os dados recebidos

por este em variáveis mais especializadas.

```
using UnityEngine;
using System.Collections;
using System;

public class GestureListener : MonoBehaviour, KinectGestures.GestureListenerInterface
{
// GUI Text to display the gesture messages.
public GUIText GestureInfo;

// private bool to track if progress message has been displayed
private bool progressDisplayed;

private bool jump;
private bool squat;
private bool walk;
private bool pullup;
private bool pulldown;

public float footfirst;
public float handfirst;
public float timeofgesture;

public bool IsJump()
{
if(jump)
{
jump = false;
return true;
}

return false;
}
public bool IsSquat()
{
if(squat)
{
squat = false;
return true;
}

return false;
}
public bool IsWalk()
{
if(walk)
{
walk = false;
return true;
}
```

```
return false;
}
public bool IsPullUp()
{
if(pullup)
{
pullup = false;
return true;
}

return false;
}
public bool IsPullDown()
{
if(pulldown)
{
pulldown = false;
return true;
}

return false;
}


public void UserDetected(uint userId, int userIndex)
{
// as an example - detect these user specific gestures
KinectManager manager = KinectManager.Instance;

manager.DetectGesture(userId, KinectGestures.Gestures.Jump);
manager.DetectGesture(userId, KinectGestures.Gestures.Squat);

manager.DetectGesture(userId, KinectGestures.Gestures.Walk);

manager.DetectGesture(userId, KinectGestures.Gestures.PullUp);
manager.DetectGesture(userId, KinectGestures.Gestures.PullDown);

if(GestureInfo != null)
{
GestureInfo.GetComponent<GUIText>().text = string.Empty;
}
}

public void UserLost(uint userId, int userIndex)
{
if(GestureInfo != null)
{
GestureInfo.GetComponent<GUIText>().text = string.Empty;
}
}

public void GestureInProgress(uint userId, int userIndex, KinectGestures.Gestures gesture,
               float progress, KinectWrapper.NuiSkeletonPositionIndex joint, Vector3 screenPos)
```

```
{
}

public bool GestureCompleted (uint userId, int userIndex, KinectGestures.Gestures gesture,
            KinectWrapper.NuiSkeletonPositionIndex joint, Vector3 screenPos)
{
string sGestureText = gesture + " detected";
if(GestureInfo != null)
GestureInfo.GetComponent<GUIText>().text = sGestureText;

if (gesture == KinectGestures.Gestures.Jump) {
timeofgesture = screenPos.z;
jump = true;
} else if (gesture == KinectGestures.Gestures.Squat)
squat = true;
else if (gesture == KinectGestures.Gestures.Walk)
{
footfirst = screenPos.y;
walk = true;
}
else if (gesture == KinectGestures.Gestures.PullUp)
{
handfirst = screenPos.y;
pullup = true;
}
else if (gesture == KinectGestures.Gestures.PullDown)
{
handfirst = screenPos.y;
pulldown = true;
}

return true;
}

public bool GestureCancelled (uint userId, int userIndex, KinectGestures.Gestures gesture,
            KinectWrapper.NuiSkeletonPositionIndex joint)
{
return true;
}

}
```

A.3 Código ShoulderCheck.cs

Código do arquivo .cs que recebe os dados do sensor Kinect e, através deles, calcula o ângulo dos ombros e se estão nos ângulos desejados.

É um cálculo constante, e não uma verificação de posicionamentos como os vistos

pelo KinectGestures.

```
using UnityEngine;
using System.Collections;

public class ShoulderCheck : MonoBehaviour
{

Quaternion angleplayer;
public bool turn = false;
public float turnside = 0;


void Update ()
{
KinectManager manager = KinectManager.Instance;

if(manager && manager.IsInitialized())
{
if(manager.IsUserDetected())
{
uint userId = manager.GetPlayer1ID();

if(manager.IsJointTracked(userId, (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderLeft) &&
                    manager.IsJointTracked(userId, (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderRight))
{
Vector3 posLeftShoulder = manager.GetJointPosition(userId,
                        (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderLeft);
Vector3 posRightShoulder = manager.GetJointPosition(userId,
                        (int)KinectWrapper.NuiSkeletonPositionIndex.ShoulderRight);

posLeftShoulder.z = -posLeftShoulder.z;
posRightShoulder.z = -posRightShoulder.z;

Vector3 dirLeftRight = posRightShoulder - posLeftShoulder;
dirLeftRight -= Vector3.Project(dirLeftRight, Vector3.up);

Quaternion rotationShoulders = Quaternion.FromToRotation(Vector3.right, dirLeftRight);

angleplayer = rotationShoulders;

float playery = angleplayer.eulerAngles.y;
if (playery > 180) playery = -360 + playery;
if ((playery > 30) && (playery < 330)) {
if (playery < 180) {
turn = true;
turnside = 1;
}
else {
turn = true;
turnside = 2;
}
```

```
}
else if ((playery < -30) && (playery > -330)) {
if (playery > -180) {
turn = true;
turnside = 2;
}
else {
turn = true;
turnside = 1;
}
}
else turn = false;

}
}
}
}

}
```

A.4 Código AvatarMove.cs

Código do arquivo .cs que recebe os dados do GestureListener e ShoulderCheck e os utiliza para afetar o aplicativo diretamente.

É onde os dados recebidos pelo KinectGestures e traduzidos pelo GestureListener, e a verificação constante feita pelo ShoulderCheck, são transformados em comandos para o jogo.

```
using System;
using UnityEngine;

public class AvatarMove : MonoBehaviour
{
[SerializeField] private float m_WalkSpeed = 10f;
        // The fastest the player can travel in the x axis.
[SerializeField] private float m_ClimbSpeed = 10f;
        // The fastest the player can travel in the y axis.
[SerializeField] private float m_JumpForce_h = 200f;
        // Amount of force added vertically when the player jumps.
[SerializeField] private float m_JumpForce_v = 300f;
        // Amount of force added horizontally when the player jumps.
[SerializeField] private LayerMask m_WhatIsGround;
        // A mask determining what is ground to the character

private Transform m_GroundCheck;
        // A position marking where to check if the player is grounded.
```

```
const float k_GroundedRadius = .01f;
        // Radius of the overlap circle to determine if grounded
public bool m_Grounded;
        // Whether or not the player is grounded.
private Transform m_CeilingCheck;
        // A position marking where to check for ceilings
const float k_CeilingRadius = .01f;
        // Radius of the overlap circle to determine if the player can stand up
private Animator m_Anim;
        // Reference to the player's animator component.
private Rigidbody2D m_Rigidbody2D;
private bool m_FacingRight = true;  // For determining which way the player is currently facing.
private float originalGravityScale;

private AudioSource audioSource;
public AudioClip sound_hJump;
public AudioClip sound_vJump;


private bool m_Jump;
private bool m_vJump;

private SimpleGestureListener gl;
[SerializeField] private GameObject gestureinfo;

private bool squatting = false;
public float jumptime = 0.25f;

private float footfirst = 0;
private float timeforstep;

private float handfirst = 0;
private float timeforclimb;

private ShoulderCheck sc;

private bool m_FaceRight = true;
private float step = 0;

public bool onLadder = false;
public bool onTopOfLadder = false;
public bool onBottomOfLadder = false;
public bool climbing = false;
private float climb = 0;
public float climbdirection = 0;


private float move_h;
private float move_v;
public bool jumping;

public void Awake()
{
```

```
m_GroundCheck = transform.Find("GroundCheck");
m_CeilingCheck = transform.Find("CeilingCheck");
m_Anim = GetComponent<Animator>();
m_Rigidbody2D = GetComponent<Rigidbody2D>();
originalGravityScale = m_Rigidbody2D.gravityScale;
audioSource = GetComponent<AudioSource> ();

gl = gestureinfo.GetComponent<SimpleGestureListener>();
sc = GetComponent<ShoulderCheck> ();
timeforstep = Time.time;
timeforclimb = Time.time;
}


public void Update()
{
MoveWalk ();
MoveClimb();

// dont run Update() if there is no user
KinectManager kinectManager = KinectManager.Instance;
if (!kinectManager || !kinectManager.IsInitialized () || !kinectManager.IsUserDetected ())
return;

if (!onLadder) {
climbing = false;
}

if (squatting) {
if (gl.IsJump ()) {
squatting = false;
if (gl.timeofgesture < jumptime) {
m_vJump = true;
m_Jump = true;
}

}
} else if (climbing) {
if (gl.IsJump ()) {
if (gl.timeofgesture < jumptime) {
climbing = false;
move_v = 0;
}
}
else if (!onTopOfLadder && gl.IsPullUp ()) {
if (Time.time > (timeforclimb + 3f)) {
handfirst = 0;
}

if ((gl.handfirst == 1) && ((handfirst == 0) || (handfirst == 2))) {
handfirst = 1;
climb += 1;
climbdirection = 1;
```

```
}

if ((gl.handfirst == 2) && ((handfirst == 0) || (handfirst == 1))) {
handfirst = 2;
climb += 1;
climbdirection = 1;
}
timeforclimb = Time.time;
}
else if (!onBottomOfLadder && gl.IsPullDown ()) {
if (Time.time > (timeforclimb + 3f)) {
handfirst = 0;
}

if ((gl.handfirst == 1) && ((handfirst == 0) || (handfirst == 2))) {
handfirst = 1;
climb -= 1;
climbdirection = -1;
}

if ((gl.handfirst == 2) && ((handfirst == 0) || (handfirst == 1))) {
handfirst = 2;
climb -= 1;
climbdirection = -1;
}
timeforclimb = Time.time;
}
}
else {
if (gl.IsSquat () && !climbing) {
squatting = true;
} else if (gl.IsJump ()) {
if (gl.timeofgesture < jumptime)
m_Jump = true;
} else if (gl.IsWalk ()) {
if (Time.time > (timeforstep + 3.0f)) {
footfirst = 0;
}

if ((gl.footfirst == 1) && ((footfirst == 0) || (footfirst == 2))) {
footfirst = 1;
step += 1;
}

if ((gl.footfirst == 2) && ((footfirst == 0) || (footfirst == 1))) {
footfirst = 2;
step += 1;
}
timeforstep = Time.time;
} else if (onLadder && gl.IsPullUp()) {
if (gl.handfirst == 1)
handfirst = 1;
if (gl.handfirst == 2)
```

```
handfirst = 2;
timeforclimb = Time.time;
climbing = true;
m_Rigidbody2D.velocity = Vector2.zero;
} else if (sc.turn) {
if (sc.turnside == 1 && !m_FaceRight)
m_FaceRight = true;
else if (sc.turnside == 2 && m_FaceRight)
m_FaceRight = false;
}
}

// Read the inputs.
bool crouch = squatting;
float h = (m_FaceRight ? step : -step);
// Pass all parameters to the character control script.
Move(m_FaceRight, h, crouch, climbing, climb, m_Jump, m_vJump);
// Reset all reusable parameters.
step = 0;
climb = 0;
m_Jump = false;
m_vJump = false;
}


public void FixedUpdate()
{
m_Grounded = false;

// The player is grounded if a circle cast to the groundcheck position hits
        // anything designated as ground
// This can be done using layers instead
        // but Sample Assets will not overwrite your project settings.
Collider2D[] colliders = Physics2D.OverlapCircleAll(m_GroundCheck.position, k_GroundedRadius,
            m_WhatIsGround);
for (int i = 0; i < colliders.Length; i++)
{
if (colliders[i].gameObject != gameObject)
m_Grounded = true;
}
m_Anim.SetBool("Ground", m_Grounded);

// Set the vertical animation
if (!climbing) m_Anim.SetFloat("vSpeed", m_Rigidbody2D.velocity.y);
else if (Mathf.Abs(move_v) > 0.01) m_Anim.SetFloat("vSpeed", 1);
else m_Anim.SetFloat("vSpeed", 0);

// Set the horizontal animation
if (Mathf.Abs(move_h) > 0.01) m_Anim.SetFloat("Speed", 1);
else m_Anim.SetFloat("Speed", Mathf.Abs(m_Rigidbody2D.velocity.x));
}

public void Move(bool faceRight, float step, bool crouch, bool climbing, float climb,
```

```
        bool jump, bool vJump)
{
// If crouching, check to see if the character can stand up
if (!crouch && m_Anim.GetBool("Crouch"))
{
// If the character has a ceiling preventing them from standing up, keep them crouching
if (Physics2D.OverlapCircle(m_CeilingCheck.position, k_CeilingRadius, m_WhatIsGround))
{
crouch = true;
}
}

// Set whether or not the character is crouching in the animator
m_Anim.SetBool("Crouch", crouch);

//If the player is climbing
if (climbing) {
m_Anim.SetBool("Climb", true);
Physics2D.IgnoreLayerCollision (8, 9);
Physics2D.IgnoreLayerCollision (8, 10);
m_Rigidbody2D.gravityScale = 0;

move_v += climb;
//m_Rigidbody2D.AddForce (new Vector2 (0, (climb * m_ClimbSpeed)));
} else {
m_Anim.SetBool("Climb", false);
Physics2D.IgnoreLayerCollision (8, 9, false);
Physics2D.IgnoreLayerCollision (8, 10, false);
m_Rigidbody2D.gravityScale = originalGravityScale;

move_v = 0;
}

// If the player should jump...
if (!climbing && m_Grounded && jump && m_Anim.GetBool ("Ground")) {
// Add a vertical force to the player.
m_Grounded = false;
m_Anim.SetBool ("Ground", false);
if (vJump) {
m_Rigidbody2D.AddForce (new Vector2 (0f, (m_JumpForce_v * 1.7f)));
audioSource.PlayOneShot (sound_vJump);
jumping = true;
} else {
m_Rigidbody2D.AddForce (new Vector2 ((m_FacingRight ? m_JumpForce_h :
                                      -m_JumpForce_h), m_JumpForce_v));
audioSource.PlayOneShot (sound_hJump);
jumping = true;
}
}
//only control the player if grounded
else if (m_Grounded) {
// If the input is moving the player right and the player is facing left...
if (faceRight && !m_FacingRight) {
```

```
// ... flip the player.
Flip ();
step = 0;
}
// Otherwise if the input is moving the player left and the player is facing right...
else if (!faceRight && m_FacingRight) {
// ... flip the player.
Flip ();
step = 0;
}
// Reduce the speed if crouching
step = (crouch ? 0 : step);

// Move the character
move_h += step;
//m_Rigidbody2D.AddForce (new Vector2 ((step * m_WalkSpeed), 0f));

}
}

private void Flip()
{
// Switch the way the player is labelled as facing.
m_FacingRight = !m_FacingRight;

// Multiply the player's x local scale by -1.
Vector3 theScale = transform.localScale;
theScale.x *= -1;
transform.localScale = theScale;

move_h = 0;
}

private void MoveWalk()
{
float move_step = move_h * Time.smoothDeltaTime * m_WalkSpeed;

if (move_step != 0 && !jumping) {
m_Rigidbody2D.MovePosition (new Vector2 (m_Rigidbody2D.position.x + move_step,
                                         m_Rigidbody2D.position.y));
move_h -= move_step;
if (m_FacingRight && move_h < 0)
move_h = 0;
else if (!m_FacingRight && move_h > 0)
move_h = 0;
}
if (Mathf.Abs (move_h) < 0.01)
move_h = 0;

if (jumping && m_Grounded)
jumping = false;
}
```

```
private void MoveClimb()
{
float move_climb = move_v * Time.smoothDeltaTime * m_ClimbSpeed;

if (climbing && move_climb != 0) {
m_Rigidbody2D.MovePosition (new Vector2 (m_Rigidbody2D.position.x,
                                         m_Rigidbody2D.position.y + move_climb));
move_v -= move_climb;
if (climbdirection == 1 && move_v < 0)
move_v = 0;
else if (climbdirection == -1 && move_v > 0)
move_v = 0;
}
if (!climbing && move_climb != 0) {
move_v = 0;
}
if (Mathf.Abs (move_v) < 0.01)
move_v = 0;

}
}
```

Referências Bibliográficas

[Alleyne2009] Alleyne, R. "Playing the video game 'Tetris' could reduce trauma", claim Oxford University, 6 de Janeiro de 2009, The Telegraph, http://www.telegraph.co.uk/news/science/science-news/4142908/Playing-the-video-game-Tetris-could-reduce-trauma-claim-Oxford-University.html

[Almeida-Guedes2015] Almeida, A. J.; Guedes, N. P. "A influência da tecnologia para o sedentarismo de estudantes no ensino fundamental", 2015. UniCEUB Centro Universitário de Brasília, http://repositorio.uniceub.br/handle/235/7546

[Gubertt2015] Gubertt, B. "Indústria de games cresce no Brasil e anda na contramão da economia". OGlobo, Brasil, 02 de abril de 2015, http://g1.globo.com/hora1/noticia/2015/04/industria-de-games-cresce-no-brasil-e-anda-na-contramao-da-economia.html

[OGlobo2015] O Globo. "Brasil é o 4º consumidor de games, mas mercado carece de mão de obra". OGlobo, Brasil, 9 de novembro de 2015, http://g1.globo.com/rio-de-janeiro/noticia/2015/11/brasil-e-o-4-consumidor-de-games-mas-mercado-carece-de-mao-de-obra.html

[Ferreira2014] Ferreira, M. "Indústria de games supera o faturamento de Hollywood". Jovens Jornalistas, Universidade Federal de Goiás, 2014. http://webnoticias.fic.ufg.br/n/68881-industria-de-games-supera-o-faturamento-de-hollywood

[FincoFraga2011] Finco, M. D.; Fraga, A. B. "Rompendo fronteiras na Educação Física através dos videogames com interação cor-

poral", SciELO Scientific Electronic Library Online, 2011. http://www.scielo.br/pdf/motriz/v18n3/a14v18n3

[deFaria2015] de Faria, E. G. "Sedentarismo na adolescência VS jogos eletrônicos", Unesc, 2015, http://200.18.15.27/handle/1/3074

[MichaelHosp2011] Michael's Hospital. "Video games effective treatment for stroke patients: study". Medical X Press, 2011. http://medicalxpress.com/news/2011-04-video-games-effective-treatment-patients.html

[Univ-Rochester2007] University of Rochester. "Action Video Games Sharpen Vision 20 Percent", 2007. http://www.rochester.edu/news/show.php?id=2764

[Green-Baveller2003] Green, C. S.; Baveller, D. "Action video game modifies visual", The Sackler Institutes for Developmental Psychobiology, 2003. www.sacklerinstitute.org/cornell/summer_institute/ARCHIVE/2003/Bavelier.pdf

[Freeman2010] Freeman, B. "Researchers Examine Video Gaming's Benefits", U. S. Department of Defense, 2010. http://archive.defense.gov/news/newsarticle.aspx?id=57695

[HMHCO2017] Houghton Mifflin Harcourt. "The History of Carmen Sandiego", Houghton Mifflin Harcourt, 2017. http://www.hmhco.com/at-home/featured-shops/the-learning-company/carmen-sandiego/history

[Park2001] Andrew Park. "The Typing of the Dead Review", GameSpot, 9 de Outubro de 2001. http://www.gamespot.com/reviews/the-typing-of-the-dead-review/1900-2816957/

[lrnj] Darrell Johnson. "LeaRN Japanese RPG: Slime Forest Adventure", LRNJ, 24 de Novembro de 2014. https://lrnj.com/

[StrwGam] Strawberry Games. "STRAWBERRY GAMES A Game Studio Run By Fruit", Strawberry Games, 8 de Janeiro de 2017. https://strawberry-games.com/

[Robillard2011] Robillard, T. K. "Picatinny advances EOD training with video game technology". U. S. Army, 2011.

www.army.mil/article/53259/Picatinny_advances_EOD_training_with_video_game_technology/

[Jean2009] Jean, G. V. "To Train Troops, Army Creates Digital Reenactments of Roadside Bomb Attacks". National Defense Magazine, 2009. www.nationaldefensemagazine.org/archive/2009/December/Pages/ To-TrainTroops,ArmyCreatesDigitalReenactmentsofRoadsideBombAttacks.aspx

[Clarke11] Clarke, J. "Six-Degrees of Freedom". John Clarke, 18 de Dezembro de 2011. http://johnclarkeonline.com/2011/12/18/six-degrees-of-freedom/

[Stefani2016] Stefanini, E. "Realidade Virtual: Entenda seus Diversos Formatos — Luvas". RealidadeVirtualBR, 2016. www.realidadevirtualbr.com.br/realidade-virtual-entenda-seus-diversos-formatos-luvas/

[Queiroz2010] Queiroz, M. "Um cientista explica o Microsoft Kinect". Webholic, 2010. http://webholic.com.br/um-cientista-explica-o-microsoft-kinect/

[Mirab.Casa.2010] Mirabella, F.; Casamassina, M. TGS 2005: Hands-on the revolution controller. http://www.ign.com/articles/2005/09/16/tgs-2005-hands-on-the-revolution-controller

[Oliveira2010] Oliveira, S. "A captura de movimento nos jogos de vídeo game". NintendoBlast, 2010. www.nintendoblast.com.br/2010/03/captura-de-movimento-nos-video-games.html

[RedmondTelAviv2010] Redmond, Wash., Tel Aviv, Israel "PrimeSense Supplies 3-D-Sensing Technology to "Project Natal" for Xbox 360". NintendoBlast, 31 de Março de 2010. https://news.microsoft.com/2010/03/31/primesense-supplies-3-d-sensing-technology-to-project-natal-for-xbox-360/

[Blitz2016] Blitz, M. "The Unlikely Story of the First Video Game". Popular Mechanics, 2016. www.popularmechanics.com/culture/gaming/a20129/the-very-first-video-game/

[Brok.Nat.Lab2015] Brookhaven National Laboratory. "Video Games, Did They Begin at Brookhaven?", DOE R&D Accomplishments, 2015. www.osti.gov/accomplishments/videogame.html

[Brok.Nat.Lab1981] Graetz, J. "The origin of Spacewar", Creative Computing, 1981. www.wheels.org/spacewar/creative/SpacewarOrigin.html

[Costa2009] Costa, L. D. "Advergames e jogos educativos: mesmos princípios projetuais?",Portal SBGames, 2009, www.sbgames.org/papers/sbgames09/artanddesign/60430.pdf

[GamingHistory2014] Gaming History. "Battlezone The Arcade PCB by Atari, Inc.", Gaming History, 19 de Dezembro de 2014. https://www.arcade-history.com/?page=detail&id=210

[TheEconomist2009] The Economist. "Good Game?", The Economist, 2009. www.economist.com/node/13726738

[VaghettiBot2010] Vaghetti, C.A; Botelho, S.S. "Ambientes virtuais de aprendizagem na educação física: uma revisão sobre a utlização de exergames", Ciências & Cognição, 2010, V.15, n. 1.

[Kffuri2013] Kffuri, L. "Que tal utilizar o console XBOX na praxis da educação física?", Os Desafios da escola pública paranaense na perspectiva do professor Candernos PDE, 2013, Vol II.

[Dobnik2004] Dobnik, V. "Surgeons may err less by playing videogames", NBC News, 2004. www.nbcnews.com/id/4685909/#.WDYk8S0rK00

[Souza2011] Souza, F. H. "Uma revisão bibliográfica sobre a utilização do Nintendo Wii como instrumento terapêutico e seus fatores de risco", Portal de Periódicos da Universidade Estadual de Maringá, 2011. http://ojs.uem.br/ojs/index.php/EspacoAcademico/article/view/13045/7605

[Barcala2015] Barcala, L., Colella, F., Araujo, M. C., Inoue Salgado, A. S., Santos Oliveira, C. Análise do equilíbrio em pacientes hemiparéticos após o treino, SciELO Scientific Electronic Library, 2011, Online: http://www.scielo.br/pdf/fm/v24n2/a15v24n2

[Tavares2013] Tavares, CN; Carbonero, FC; Finamore, PS; Kós, RS. "Uso de nintendo Wii para reabilitação de crianças com paralisia cerebral: estudo de caso", Rev. Neurociência, 21(2), pag 286-293.

[Wibelinger2013] Wibelinger, L. M.; Secchi Batista, J.; Vidmar, M. F.; Kayser, B.; Pasqualotti, A.; Schneider, R. H. "Efeitos da fisioterapia convencional e da wiiterapia na dor e capacidade funcional de mulheres idosas com osteoartrite de joelho", Revista Dor, vol 14, no 3, 2013.

[Fernandes2015] Fernandes Meirelles Araújo, É. M. Desenvolvimento de um Sistema de Medições Livre de Marcadores Utilizando Sensores de Profundidade, Dissertação de Mestrado, UENF, 2015. https://drive.google.com/file/d/0B8YqovDlquBgazg4U1F4RVhCSkk/view

[Schiavinato2010] Schiavinato, A. M.; Baldan, C.; Melatto, L.; Lima, L. S. "Influência do Wii Fit no equilíbrio de paciente com disfunção cerebelar: estudo de caso", UNIP Universidade Paulista. http://200.196.224.129/comunicacao/publicacoes/ics/edicoes/2010/01_jan-mar/V28_n1_2010_p50-52.pdf

[Plumer 2012] Plumer, B. "The economics of video games". The Washington Post, 2012,www.washingtonpost.com/news/wonk/wp/2012/09/28/the-economics-of-video-games/

[Reeves2008] Reeves, B.; Malone, T. W.; O'Driscoll, T. "Leadership's Online Labs", Harvard Business Review, 2008. https://hbr.org/2008/05/leaderships-online-labs

[Shotton2010] Shotton, J.; Fitzgibbon, A.; Cook, M.; Sharp, S.; Finocchio, M.; Moore, R.; Kipman, A.; Blake, A. "Real-Time Human Pose Recognition in Parts from Single Depth Images". Microsoft Research Cambridge & Xbox Incubation, Cambridge, 2010. http://www.cse.chalmers.se/edu/year/2010/course/TDA361/Advanced Computer Graphics/BodyPartRecognition.pdf