UNIVERSIDADE FEDERAL DO PARÁ
INSTITUTO DE CIÊNCIAS EXATAS E NATURAIS
FACULDADE DE COMPUTAÇÃO
BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ESTHER CARDOSO DA SILVA

ATUALIZAÇÃO DO MÓDULO DE RECONHECIMENTO DE MOVIMENTOS DE UM JOGO TERAPÊUTICO UTILIZANDO A BIBLIOTECA OPENPOSE

BELÉM
2019

ESTHER CARDOSO DA SILVA

ATUALIZAÇÃO DO MÓDULO DE RECONHECIMENTO DE MOVIMENTOS DE UM JOGO TERAPÊUTICO UTILIZANDO A BIBLIOTECA OPENPOSE

Trabalho apresentado ao curso de Ciência da Computação da Universidade Federal do Pará como pré-requisito para obtenção do título de Bacharel em Ciência da Computação.
Orientador: Prof. Dr. Dionne Cavalcante Monteiro.
Co-orientador: Prof. Msc. Antônio Fernando Lavareda Jacob Júnior.

Belém, Julho de 2019

Agradecimentos

Aos meus irmãos, Abner e Fabrício, que me guiaram por todo o caminho até aqui e me apoiaram nos momentos mais difíceis.

Aos meus pais por todo o amor e incentivo.

Aos meus grandes amigos Bruno Yusuke, por todo o apoio e compreensão no decorrer deste ano; e Arthur Takeshi, pelo suporte no desenvolvimento deste trabalho.

Aos meus orientadores, Prof. Dr. Dionne Monteiro e Prof. Msc. Antônio Jacob, pela paciência e empenho durante o desenvolvimento deste projeto.

Aos meus colegas de faculdade que me propiciaram momentos inesquecíveis ao longo do curso.

E a todas as pessoas que direta ou indiretamente contribuíram para a realização da minha pesquisa, eu deixo o meu muito obrigada.

"A serenidade é apenas a casca da árvore da sabedoria,
mas, não obstante, serve para esta perseverar."
(Confúcio)

Resumo

A fisioterapia é um processo composto por tarefas repetitivas e, portanto, resulta em altas taxas de desistência de pacientes. Os jogos para o auxílio da prática de exercício, também conhecidos como *exergames*, oferecem uma alternativa de baixo custo para tornar esse processo mais agradável. Porém, os *exergames* comerciais não possuem as rotinas de exercícios necessários para a reabilitação de pacientes com necessidades específicas. Portanto, o desenvolvimento de jogos sérios com atividades projetadas para esse contexto vêm ganhando grande espaço dentro do meio acadêmico e da área da saúde. *Skyway* é um jogo terapêutico desenvolvido por estudantes de Engenharia da Computação da Universidade Federal do Pará, voltado para o fortalecimento do equilíbrio corporal para pacientes com dificuldades de locomoção. O módulo de reconhecimento de movimentos desse *exergame* foi desenvolvido com ferramentas específicas para o dispositivo *Microsoft Kinect*, o sensor de movimentos do console *Xbox*, que faz uso de dados de cor, advindos de imagens RGB, e profundidade, obtidos por meio de infravermelho, para estimar a posição do jogador. Devido à retirada do dispositivo *Kinect* do mercado pela fabricante em setembro de 2017, tornou-se interessante a substituição do mesmo por uma ferramenta mais moderna. *OpenPose* é uma biblioteca de código aberto e sem custos para aplicações não comerciais a qual utiliza técnicas de segmentação de imagem e aprendizado de máquina para reconhecimento de movimentos e expressões de pessoas em imagens 2D. Essa ferramenta foi escolhida para substituição do *kit* de desenvolvimento do *Kinect* por não apresentar custos e permitir a utilização de diferentes tipos de dispositivos para captura de imagens. Após a substituição das ferramentas de desenvolvimento, foi feita uma comparação entre as funcionalidades presentes nas duas versões do jogo. A ferramenta desenvolvida com a biblioteca *OpenPose* teve êxito em movimentar o personagem no eixo horizontal, porém, devido às limitações de coordenadas tridimensionais presentes no *plugin* utilizado no *Unity*, não foi possível implementar a movimentação do personagem no eixo vertical. Também notou-se, a partir de testes limitados, que o jogo mantém o desempenho da versão original somente quando o processamento da biblioteca é feito em placas gráficas com suporte a CUDA, porém, se mostrou inviável para execução exclusiva nos processadores disponíveis para teste.

Palavras-chave: Jogos sérios, Reabilitação, *Motion Tracking*, Visão Computacional, *Deep Learning*.

Abstract

Physical therapy is a process composed of repetitive tasks and, therefore, results in high rates of treatment quitting by the patients. Fitness games, also called exergames, offer a low-cost alternative to make this process more enjoyable. However, commercial exergames do not have the necessary exercise routines for the rehabilitation of patients with specific needs. Therefore, the development of serious games with activities designed for this context has been gaining great interest in the academic environment and the healthcare field. Skyway is a therapeutic game developed by Computer Engineering students at the Federal University of Pará, aimed at improving body balance for patients with locomotion problems. The motion tracking module of this exergame was developed with tools specific to the Microsoft Kinect device, Xbox console's motion sensor, which uses color data, derived from RGB images, and depth data, obtained by infrared, to estimate the player's position. Due to the Kinect device being pulled out from the market by the manufacturer in September 2017, it became desirable to replace it with a more modern tool. OpenPose is a free, open source library for non-commercial applications that uses image segmentation and machine learning techniques to recognize movements and expressions of people in 2D images. This tool was chosen to replace the Kinect development kit for being free of costs and for allowing the use of different kinds of imaging devices. After replacing the development kit, a comparison was made between the features present in both versions of the game. The game developed with the OpenPose library succeeded in moving the character on the horizontal axis, however, due to the limitations of three-dimensional coordinates in the Unity plugin, it was not possible to implement the character movement on the vertical axis. It has also been noted from limited testing that the game retains the performance of the original version only when library processing is done on CUDA-enabled graphics cards, but has proved impractical for execution only on the processors available for testing.

Keywords: Serious Games, Rehabilitation, Motion Tracking, Computer Vision, Deep Learning.

Lista de ilustrações

Lista de tabelas

Lista de abreviaturas e siglas

CUDA	*Compute Unified Device Architecture* (Arquitetura de Dispositivo de Computação Unificada)
cuDNN	NVIDIA CUDA® *Deep Neural Network*
CPU	*Central Processing Unit* (Unidade Central de Processamento)
DLL	*Dynamic-link library* (Biblioteca de *links* dinâmicos)
GPU	*Graphics Processing Unit* (Unidade de Processamento Gráfico)
IDE	*Integrated Development Environment* (Ambiente de Desenvolvimento Integrado)
JSON	*JavaScript Object Notation* (Notação de Objetos Javascript)
OpenCL	*Open Computing Language* (Linguagem de Computação Aberta)
RGB	*Red, Green and Blue*
VCS	*Video Computer System*

Sumário

1 Introdução

Jogos são um elemento da natureza de todos os seres vivos, podendo ser observados até mesmo em comunidades de animais. São anteriores ao surgimento da cultura e um componente essencial da inteligência humana que faz parte de todas as atividades da sociedade, como religião, esportes, guerras e até mesmo o surgimento da arte. Além disso, podem desempenhar papéis importantes dentro dessas práticas, como servir de experiência para situações reais, gastar energia corporal sobressalente, inspirar a criatividade, entre outros [Huizinga 2001]. Com o surgimento e popularização de dispositivos eletrônicos, também foram projetados meios de entretenimento que tirassem proveito dessas tecnologias.

No início da década de 1940, antes mesmo da popularização da venda de computadores, Raymond Redheffer [Redheffer 1948] desenvolveu o que acredita-se que seja a primeira forma de entretenimento eletrônico interativo: uma máquina que conseguia jogar Nim contra um jogador humano. Nesse jogo, cada jogador retira um objeto de uma pilha, e é derrotado quem retirar o último objeto dela. Apesar de utilizar circuitos simples, a máquina conseguiu uma porcentagem de vitórias de 85% sobre os oponentes durante os seis meses em que ficou em exposição.

Os jogos desenvolvidos em sequência à máquina de Nim tinham como maior objetivo testes e demonstrações de novas tecnologias. Entre estes, o OXO, um "jogo da velha" desenvolvido por Alexander Douglas em 1952 para demonstração uma interface gráfica, que se tornou o primeiro computador com gráficos [Vaughan-Nichols 2009]. O primeiro jogo voltado somente para o entretenimento foi *Tennis For Two*, desenvolvido por William Higginbotham em 1958, considerado o primeiro *videogame* da história [Stony Brook University Libraries 2019], e em 1962, Steve Russell desenvolveu o jogo *Spacewar!*, que foi o primeiro jogo com instalações em múltiplas máquinas [Rutter e Bryce 2006].

Somente em 1971 começaram a surgir jogos comerciais com o desenvolvimento de uma versão do jogo *Spacewar!* por Nolan Bushnell, chamada de *Computer Space*, comercializada pela empresa *Nutting Associates* em uma máquina de fliperama. Posteriormente, em 1972, Bushnell fundou a empresa *Atari*, que impulsionou o mercado dos *videogames*, no ano de 1977, com o lançamento do console doméstico chamado de *Atari 2600 Video Computer System* (VCS) [Longuidice e Barton 2014].

Devido ao grande aumento nas vendas dos jogos durante o período de vendas do *Atari*, juntamente com o console concorrente Odyssey, houve uma massificação no desenvolvimento de jogos, diminuindo cada vez mais a qualidade dos mesmos, desmotivando a compra dos consoles e culminando na crise conhecida como "*Videogame Crash*"em 1983, na qual houve um desinteresse generalizado pela compra de jogos. Em 1985, a

empresa japonesa *Nintendo* teve êxito em reaquecer o mercado ao comercializar jogos como brinquedos, diferentemente de entretenimento eletrônico, como eram conhecidos, além de aplicar restrições ao desenvolvimento de jogos para o console NES (*Nintendo Entertainment System*) [Peitz e Waldfogel 2012].

O ressurgimento do mercado de jogos nesse período trouxe mais empresas para esse ramo, as quais deram origem às *Console Wars*, em que consumidores se dividem em preferências pelas empresas, baseando-se principalmente na potência dos respectivos dispositivos. A *Nintendo*, diferentemente das outras gigantes que focaram o desenvolvimento de consoles no poder de processamento, buscou diferenciar os próprios dispositivos por meio de formas diferentes de interação com o jogo, de onde surgiram console com telas de toque, sensores de movimento, entre outros. Com o aumento do poder de processamento de computadores e *smartphones*, estes também se tornaram populares opções para se usufruir dessa forma de entretenimento.

Atualmente, o mercado de jogos movimenta bilhões de dólares por ano. A maioria das casas americanas possuem algum dispositivo utilizado para jogar. Os jogos eletrônicos fazem parte do cotidiano, principalmente de adultos, independentemente do gênero [Entertainment Software Association 2019].

Diante disso, existem vários estudos voltados à melhora da satisfação do jogador durante o exercício dos jogos eletrônicos. Além de elementos como gráficos e som [Brown e Cairns 2004], um conceito bastante aceito como base para a modelagem de jogos é o de [Csikszentmihalyi 1975], que relaciona a experiência de certa atividade com a dificuldade de progredir em uma operação e a habilidade do jogador em relação à tarefa apresentada. Os estados da experiência podem variar entre "frustrante", "entediante" e, quando há equilíbrio entre dificuldade e habilidade, atinge-se o estado ideal, chamado de "*flow*".

A partir desses princípios, surgiram os chamados "jogos sérios", que não buscam somente entreter os jogadores, mas também incentivar o exercício de atividades repetitivas, com objetivos como inserção de hábitos saudáveis no cotidiano do jogador, treinamentos e terapias. Porém, esses jogos, que são geralmente desenvolvidos pelo meio acadêmico, não contam com tanto capital quanto jogos comerciais o que torna necessário utilizar ferramentas de desenvolvimento de baixo custo, geralmente com recursos mais limitados que softwares comerciais com finalidade semelhante.

O processo de reabilitação é compostos de exercícios para desenvolvimento de movimentos específicos em pacientes de fisioterapia. Porém, essas atividades são bastante repetitivas e resulta em uma taxa alta de evasão [Unnikrishnan, Moawad e Bhavani 2013]. O lançamento das plataformas providas de controles com sensores de movimento *Wii*, *Playstation 3* e *Xbox*, auxiliaram a inserção de *exergames* ao mercado, que são jogos que combinam atividades físicas com elementos de videogame, por meio do rastreio de movimentos do corpo ou reações do usuário tentam envolver os usuários em um sistema

mais agradável e interativo [Schmidt et al. 2018].

Porém, os *exergames* comerciais não possuem as rotinas de exercícios necessários para a reabilitação de pacientes com necessidades específicas, por não terem foco terapêutico. Portanto, o desenvolvimento de jogos sérios específicos para a reabilitação de pacientes vem ganhando grande espaço dentro do meio acadêmico e da área da saúde [Pirovano et al. 2016].

Skyway [Soares et al. 2016] é um jogo terapêutico desenvolvido para auxiliar o progresso de sessões fisioterapêuticas, voltado para o desenvolvimento do equilíbrio corporal e consiste em mover o torso para guiar uma asa delta para dentro de argolas que aparecem no cenário. O jogo foi desenvolvido no motor de jogos *Unity* [Unity 2019], utilizando o *Kinect* [Microsoft 2019], um sensor de movimento e profundidade, para reconhecer poses e movimentar o personagem do jogo de acordo com a inclinação do tronco do paciente. Porém, em outubro de 2017, a empresa *Microsoft* anunciou o término do suporte ao *Kinect*, o que pode trazer inconveniências na manutenção do *software*, como incompatibilidades com sistemas operacionais ou ferramentas de desenvolvimento no futuro.

Portanto, tornou-se desejável modernizar o módulo de reconhecimento de poses do jogo. Após uma análise de ferramentas disponibilizadas na internet, foi escolhida a biblioteca *OpenPose* [*CMU Perceptual Computing Lab* 2019] para a substituição, devido a ser relativamente nova, possuir uma comunidade ativa, não apresentar custos para ferramentas não comerciais, e possibilitar o uso de vários tipos diferentes de sensores para captura de imagens.

1.1 Objetivos

O jogo *Skyway* foi desenvolvido para auxiliar o tratamento de pacientes de fisioterapia com problemas de movimentação do tronco, utilizando o sensor *Kinect* para movimentar o personagem por meio da detecção de movimentos do jogador. A retirada desse sensor do mercado pela fabricante torna inconveniente o uso do mesmo, pois dificulta a obtenção do dispositivo e resulta em uma diminuição de suporte pela comunidade, incompatibilidade com ferramentas futuras, etc. Em vista disso, este trabalho tem como objetivo a substituição do *kit* de desenvolvimento do *Kinect* por uma ferramenta com funções semelhantes e que ainda esteja ativa. Como objetivos específicos podem ser citados:

- Analisar as funcionalidades do jogo;
- Identificar uma ferramenta que forneça as funções necessárias;
- Fazer a substituição da ferramenta de desenvolvimento do *Kinect*;
- Avaliar a viabilidade do software atualizado.

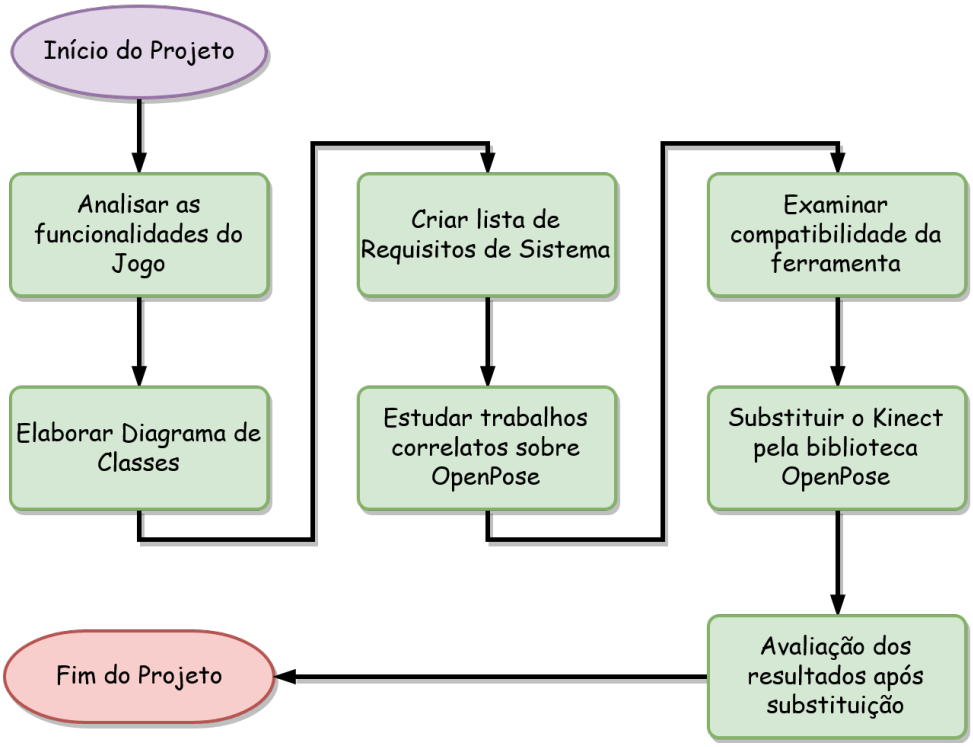
1.2 Metodologia

Para realizar a substituição do módulo de reconhecimento de poses foi necessário fazer o mapeamento das funções do jogo, que, por ter sido desenvolvido por vários programadores em períodos diferentes, sem nenhum tipo de documentação ou padronização de código, tornou-se bastante complexo. Assim, foi construído um diagrama de classes para facilitar o entendimento do fluxo de informações entre as classes do jogo.

Posteriormente, foi feita uma análise de ferramentas para detecção de movimentos que fossem de baixo custo, que possuem funções semelhantes às do *Kinect*, e que estivessem ativas. Para a remodelagem do sistema de *motion tracking* do jogo foi utilizada a biblioteca *OpenPose*.

Para fazer a substituição, foi feito o estudo da ferramenta *OpenPose* e da união da mesma com o motor *Unity*, ambiente de desenvolvimento em que o jogo *Skyway* foi programado. A inserção da biblioteca no jogo foi feita por meio de um plugin [*CMU Perceptual Computing Lab* 2019] ainda experimental criado pelos mesmos desenvolvedores do *OpenPose*, que traduz as funções da biblioteca que foi desenvolvida em C++ para C#, especificamente para uso no *Unity*. Após a inserção da biblioteca no jogo, foi feita uma descrição das funções desempenhadas por cada módulo presente no jogo, para facilitar futuras modificações ao código do jogo. A Figura 1 representa de forma amigável os passos tomados para a realização do projeto.

Figura 1 – Fluxograma da metodologia.



Fonte: Elaborado pelo autor.

1.3 Trabalhos Relacionados

Esta seção apresenta alguns trabalhos nos quais a biblioteca *OpenPose* é utilizada para desenvolver sistemas voltados para o desenvolvimento de jogos sérios ou que possuem funcionalidades com grande potencial para incrementar a área da reabilitação.

Grammatikopoulou et al. 2019 aborda uma aplicação para a reabilitação de pacientes com doença de Parkinson por meio de um jogo sério que auxilia na detecção de níveis de comprometimento e suporte no tratamento da pessoa debilitada. A ferramenta foi desenvolvida utilizando como sensor de captura de imagem e movimento o *Kinect* em conjunto com a biblioteca *OpenPose* para processamento de imagens e detecção das juntas do jogador. O projeto conta com uma abordagem de aprendizagem profunda, analisando as sequências de esqueletos humanos registrados para prever o nível de declínio das habilidades motoras dos pacientes com altas taxas de acurácia.

Com o objetivo de facilitar a utilização de tecnologias por usuários surdos, Grif e Prikhodko 2018 desenvolveu um sistema para reconhecimento de símbolos de linguagens de gestos no sistema de notação de *Hambug*. A biblioteca *OpenPose* foi utilizada para reconhecimento de cinco características na mão detectada: eixo, ou direção do movimento do antebraço; direção da ponta dos dedos e da palma; rotação da mão; trajetória do movimento da mão; e formato, ou estado da mão, pulso e dedos. Esse estudo futuramente poderá ser aplicado à linguagem de sinais, auxiliando na educação de pessoas com problemas de audição.

Qiao, Wang e Li 2017 apresenta um sistema de avaliação de poses em tempo real por meio de um sistema de notas, que se baseia na distância das junta detectada, com pesos atribuídos a cada uma delas. A biblioteca *OpenPose* foi utilizada para reconhecimento de movimento em imagens monoculares. Esta aplicação apresentaria uma forma de ter controle sobre a postura dos pacientes durante as seções de tratamento, tornando possível orientá-lo por meio da interface do próprio jogo, viabilizando sessões remotas.

1.4 Estrutura do Trabalho

Além desta seção de introdução, este trabalho possui outros três capítulos. O capítulo 2 contempla a fundamentação teórica essencial para o desenvolvimento da aplicação; o capítulo 3 explica como o trabalho foi feito, aplicando o que foi descrito na fundamentação teórica, o capítulo 4 expõe conclusões obtidas durante a pesquisa, relata pontos positivos e negativos acerca dos resultados finais, e propõe possíveis melhorias futuras.

2 *Fundamentação Teórica*

Para embasar o processo de adaptação do jogo, foi necessário dedicar esforços para revisar conceitos fundamentais aplicados à jogos sérios e *exergames*, avaliar dispositivos de bibliotecas para captura e processamento de imagens. Neste capítulo serão apresentadas os principais conceitos e ferramentas estudadas, assim como suas contribuições ao processo.

2.1 Jogos Sérios

Jogos sérios, em inglês *serious games*, são desenvolvidos baseados no modelo de um jogo interativo (*software* ou *hardware*) com propósito de adicionar contexto a atividades repetitivas, como ensino, terapias ou treinamentos, não visando apenas o entretenimento. Envolvem também o uso de tecnologias de jogos digitais com o propósito de simular situações do mundo real [Rocha, Bittercourt e Isotani 2015].

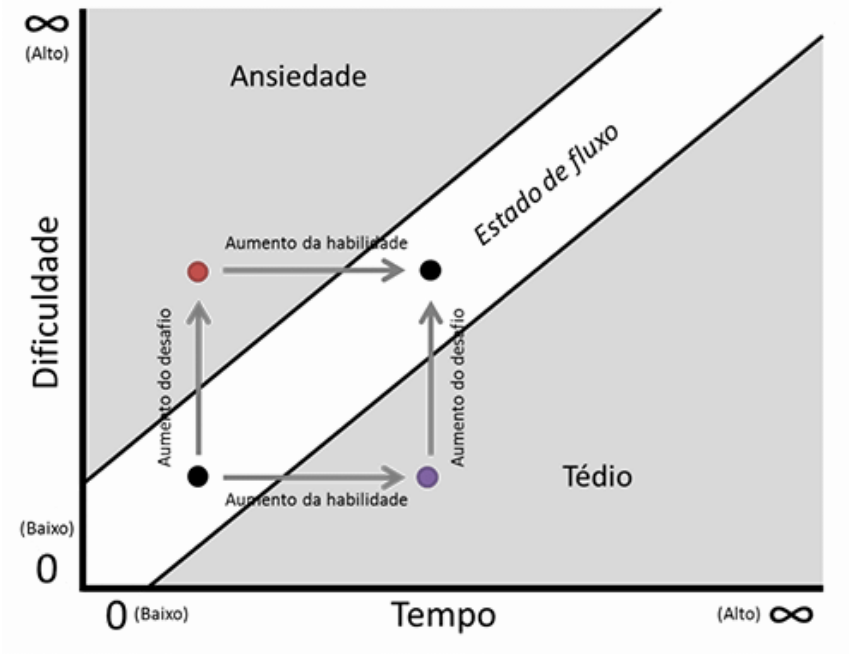
Um conceito bastante aceito para *design* de desafios em jogos é o de [Csikszentmihalyi 1975], que descreve alguns requisitos para que se tenha uma experiência positiva durante o desempenho de alguma atividade. Este conceito é denominado *flow*, e é caracterizado pela concentração das energias de um indivíduo na execução de uma tarefa. Quando em estado de *flow*, os pensamentos, intenções e sentimentos estão alinhados com a tarefa executada, fazendo com que o indivíduo se sinta imerso e perca a percepção de tempo. É também afirmado que a capacidade de atingir o *flow* tem influências diretas na satisfação e execução de atividades.

Csikszentmihalyi hipotetizou que a experiência possui três dimensões: Ansiedade, quando os desafios enfrentados são difíceis demais em relação às habilidades do praticante; tédio, quando a pessoa já domina todas as habilidades necessárias para superar os obstáculos; e *flow*, quando a dificuldade das tarefas não supera as habilidades do indivíduo de superá-las, permitindo que este ganhe conhecimentos que o permitiriam enfrentar atividades com maiores complexidades.

Esses estados variariam de acordo com a função apresentada na Figura 2, onde o conhecimento adquirido durante a execução da tarefa faz com que o indivíduo se sinta entediado e a harmonia pode ser restabelecida por meio de inovação que por sua vez pode causar ansiedade devido a inabilidade do jogador em superar os novos obstáculos. Porém, a partir de testes da hipótese, foram identificados oito estados diferentes. Quando a habilidade era muito baixa e os obstáculos eram muito fáceis, ou quando uma tarefa se repetia muitas vezes, o indivíduo não se sentia comprometido a completar as atividades, o que contraria a hipótese de Csikszentmihalyi. A partir do resultado dos teste, foi proposto

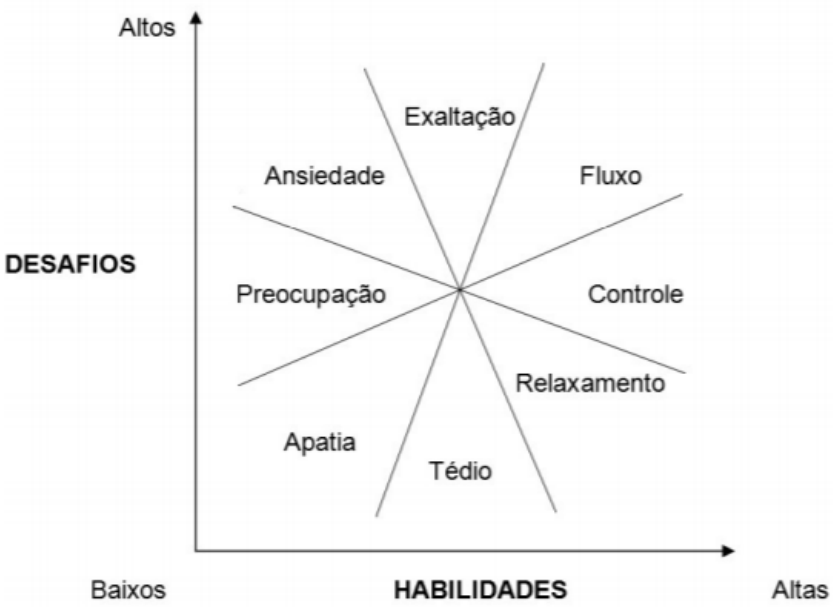
por [Massimini e Carli 1992] o modelo da Figura 3, que apresenta oito dimensões assumidas pela experiência de uma atividade [Cowley et al. 2008].

Figura 2 – Três dimensões da experiências (ansiedade, flow, tédio).



Fonte: Adaptado de Csikszentmihalyi 1975

Figura 3 – Oito dimensões da experiência.



Fonte: Adaptado de Massimini e Carli 1992

Outros fatores relevantes para otimizar a experiência no jogos são o enredo, os gráficos e o áudio. O enredo é importante para a contextualização das atividades que

o jogador vai executar durante o jogo. Já os gráficos e áudio se combinam para criar um ambiente agradável, assim como desafios, onde o jogador precisa prestar atenção na modificação dos elementos em questão [Brown e Cairns 2004].

Porém, há uma barreira financeira no desenvolvimento de jogos sérios, visto que geralmente são para uso não comercial [Fernández-Manjón et al. 2015], o que dificulta a obtenção de recursos para desenvolvimento de atividades para gerem o estado de *flow* ou capturem a atenção do jogador, além de tornar necessária a utilização de *softwares* gratuitos para desenvolvimento, que geralmente são mais limitados que as ferramentas comerciais.

2.1.1 *Jogos Terapêuticos*

A aplicação de métodos clássicos em processos terapêuticos muitas vezes pode ser estressante e enfadonho para o paciente. Com isso em mente, tem se tornado recorrente a inclusão dessas tecnologias, anteriormente utilizadas apenas para o entretenimento, como forma de tornar a reabilitação mais divertida, consequentemente, garantindo um maior comprometimento por parte do paciente [Pirovano et al. 2016].

Conforme [Vaghetti e Botelho 2010] (apud [Bekker e Eggen 2008]; [Berkovsky et al. 2009]) o crescimento na tecnologia da informação e dos jogos sérios, colaboraram para o advento da classe de *games* voltada para a prática de atividade física. O foco é dirigido para a captura de estímulos humanos como dados de entrada, promovendo o lazer em atividades físicas, com a finalidade de ampliar a interatividade e o gasto calórico. Essa classe de jogo é chamada de *exergames*, em resumo, é a junção de *games* com exercícios físicos.

Porém, os *exergames* comerciais, quando utilizados independentemente podem resultar em resultados adversos, como dores nas costas e lesões. Portanto, para o desenvolvimento de um jogo terapêutico, é desejável que haja auxílio de um profissional da área da saúde. [Pirovano et al. 2016].

2.2 Visão Computacional

Visão computacional é ramo da computação que estuda a obtenção de dados através de imagens e vídeos, extraindo suas formas, informações, etc. O objetivo é a formação de descrições sobre um objeto de interesse que será identificado por algoritmos, em seguida, o corpo que foi extraído é submetido a filtros para realçar os resultados e posteriormente realizar alguma operação sobre ele [Rios 2010].

Motion Tracking é o processo de reconhecimento de movimentos e expressões feitas por uma pessoa a partir de sensores, sejam estes câmeras para captura de imagem ou

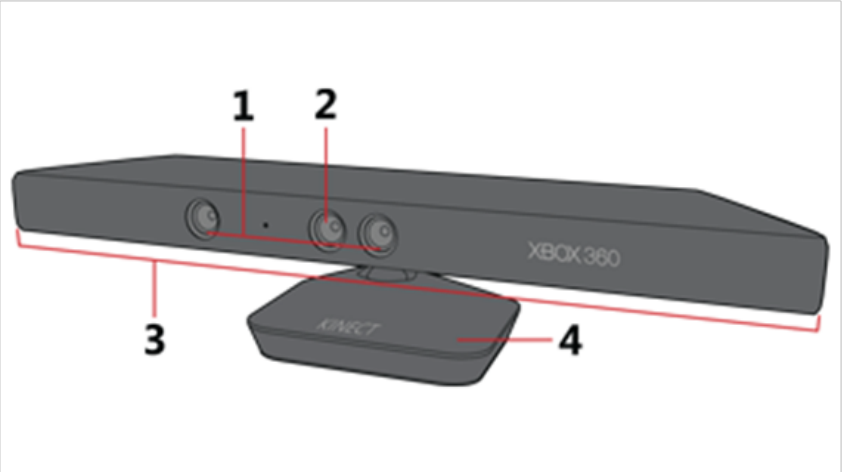
sensores de infravermelho. Para que seja possível que o computador interprete e identifique pessoas, expressões ou objetos nas imagens obtidas, os algoritmos de visão computacional são essenciais. Quando utilizados em conjunto, tornam-se possíveis interações físicas e complexas com sistemas computacionais, que vão além do pressionar de botões [Gao et al. 2018]. Abaixo são listados algumas tecnologias voltadas para a obtenção e processamento de imagens para estimação de poses.

2.2.1 *Microsoft Kinect*

O *Kinect* é um sensor de detecção de cores e profundidade que fornece imagens coloridas e com profundidades sincronizadas. Foi inicialmente usado como um dispositivo de entrada de dados pela *Microsoft* para o console de videogame *Xbox 360*. Com um algoritmo de captura de movimento 3D, ele permite interações entre usuários e um jogo sem a necessidade de manusear um controle [Han et al. 2013].

O *Kinect* é composto de sensores de profundidade tridimensionais que utilizam infravermelho (Figura 4, número 1); câmeras RGB (Red, Blue and Green) (Figura 4, número 2); múltiplos microfones (Figura 4, número 3); e um suporte capaz de se inclinar automaticamente de acordo com o posicionamento dos objetos rastreados (Figura 4, número 4) [Microsoft 2019].

Figura 4 – Componentes do *Kinect*.

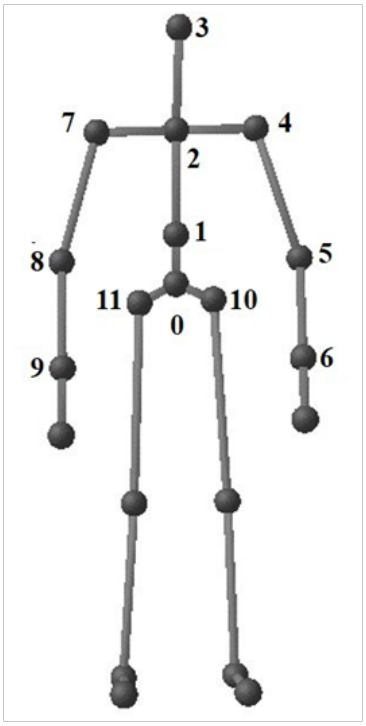


Fonte: Microsoft 2019

As poses do *Kinect* são definidas pelo conjunto de coordenadas tridimensionais de cada uma das juntas identificadas na imagem, que formam o esqueleto mostrado na Figura 5 [Plantard, Shum e Multon 2016]. Esse dispositivo foi desenvolvido para ser posicionado em frente ao jogador e deve ser utilizado em um ambiente sem muitos objetos no escopo do sensor, caso contrário, a acurácia dos dados do esqueleto diminuem. Além disso, já que o reconhecimento do sensor é baseado em imagens de profundidade, a oclusão

parcial ou total afetam muito a capacidade de reconhecimento do *Kinect* [Plantard, Shum e Multon 2017].

Figura 5 – Esqueleto com as juntas captadas pelo *Kinect*.



Fonte: Adaptado de [Plantard, Shum e Multon 2016]

2.2.2 *Azure Kinect* e *HoloLens*

Devido à grande disseminação de dispositivos de realidade virtual no mercado, a *Microsoft* está trabalhando no desenvolvimento da tecnologia de realidade aumentada *HoloLens* [Tuliper 2019], juntamente com um novo sensor de movimento para *Windows* 10, o *Azure Kinect* [*Microsoft* 2019], que foi anunciado no evento *Mobile World Congress* de 2019. O dispositivo apresenta um design mais compacto com funcionalidades semelhantes às do *Kinect* para *Xbox*. A Fabricante anunciou que o novo dispositivo não terá compatibilidade com programas desenvolvidos para as versões anteriores do *Kinect*.

2.2.3 Outros Sensores de Profundidade

VicoVR [*VicoVR* 2019] é um dispositivo que tem como finalidade a utilização de realidade virtual em dispositivos móveis. O sensor também possui uma biblioteca para rastreio de movimentos, que permite a transmissão de dados de movimento e profundidade para *smartphones Android* ou iOS via *bluetooth*. A característica estritamente *wireless* desta tecnologia pode apresentar uma desvantagem por causar *delay* na transmissão dos dados.

A empresa *Orbbec* [*Orbbec* 2019] comercializa dois tipos de sensores: o *Astra*, um dispositivo com funcionalidades semelhantes às do *Kinect* que pode ser utilizado também em *Linux*; e o *Persee*, que possui um sistema operacional integrado e já inclui uma biblioteca de reconhecimento de movimentos, voltado para dispositivos móveis.

O *Stereolabs ZED* [*Stereolabs* 2019] é um sensor que, diferentemente do *Kinect*, utiliza duas câmeras de alta resolução para estimar a percepção de profundidade. Apesar de possuir especificações boas, o dispositivo não possui uma biblioteca de reconhecimento de movimentos própria e não possui microfone.

2.3 *Deep Learning*

Machine learning (em português, aprendizado de máquina) é o ramo da inteligência artificial que tem como objetivo o desenvolvimento de algoritmos de propósito geral para treinar máquinas, a partir do processamento de bases de dados, para execução autônoma de uma determinada atividade [Acharya et al. 2018].

Dentre os algoritmos de *machine learning*, destaca-se a rede neural (em inglês, *Neural Network*), que busca simular o funcionamento do cérebro humano a partir da interação entre estruturas de dados denominadas neurônios. Esse algoritmo pode ser visto como um conjunto de camadas formadas por neurônios ligados uns aos outros por conexões ponderadas. Essas camadas podem ser divididas em três grupos: camada de entrada, que recebe as informações, o conjunto de dados iniciais sobre o qual se deseja operar. A camada oculta, responsável pela avaliação dos dados inseridos e a camada de saída, que armazena o resultado final do processamento. Nesse processo, cada neurônio funciona como uma função de ativação, que é alimentada pela camada anterior e enviada para a camada seguinte, dessa forma quanto maior o número de neurônios, melhores serão os resultados. Esse processo é repetido até a última camada da rede, onde o resultado será avaliado e classificado [Acharya et al. 2018].

O *Deep Learning* (em português, aprendizado profundo) está inserido na área de *machine learning*, e se trata da utilização de redes neurais com um grande número de neurônios e camadas ocultas. O aumento no número de neurônios possibilita a utilização desses algoritmos em processos mais complexos e muitas vezes com fluxo de informações em tempo real, como reconhecimento de imagens de câmeras, reconhecimento de voz ou reconhecimento de expressões faciais. Esses algoritmos são mais propícios a execução em GPU, devido ao grande número de camadas existentes no processo e consequente maior demanda de processamento, se torna inviável a execução em CPU.

2.3.1 *Caffe*

Para o reconhecimento de imagem em tempo real, utilizado pelo *OpenPose*, que possibilita a identificação de pessoas assim como suas articulações e membros, esse faz uso do *framework Caffe* [*Berkeley Vision and Learning Center* (BVLC) 2019], que tem como objetivos principais oferecer as ferramentas necessárias para implementação de redes neurais para *deep learning* de modo modular. Durante o processo de instalação do *OpenPose* em plataformas *Windows*, é instalado automaticamente o *framework Caffe*, no caso dos sistemas *Linux* e *macOS*, deve-se instalar separadamente de acordo com as instruções do site oficial da biblioteca.

2.4 *Unity*

Unity foi o ambiente de desenvolvimento originalmente utilizado para a programação do jogo *Skyway*. Portanto, para a elaboração deste trabalho, foi realizado um estudo das principais funcionalidades presentes na IDE (Ambiente de Desenvolvimento Integrado).

A ferramenta é um *game engine* (motor de jogo) utilizado para a construção de jogos 2D e 3D para múltiplas plataformas. A ferramenta consiste em um conjunto de bibliotecas para facilitar o desenvolvimento de um jogo, que abstraem a camada de *hardware* onde o jogo será executado e o funcionamento de tarefas de baixo nível, como cálculos geométricos e de colisão, tornando o uso da ferramenta mais acessível a usuários que não possuem conhecimentos avançados sobre essas operações. O programa permite a elaboração da aplicação em códigos Java e C# com implementações genéricas que associam as funcionalidades escritas em um *script*.

O *software* permite que sejam criados *assets*, que são amostras de objetos ou componentes previamente criadas pela comunidade ou pelo autor, para auxiliar no desenvolvimento do projeto, poupando tempo para o andamento das atividades e conteúdos adicionais ao jogo. Os jogos criados em *Unity* também são compostos de "cenas", que agrupam e organizam recursos e *scripts* correspondentes, por exemplo, a janelas ou fases individuais do jogo. Cabe ao desenvolvedor alternar entre as cenas corretamente conforme o jogo progride.

A interface principal do *Unity* é composta por várias vistas, que permitem editar características do jogo em alto nível e que podem ser organizadas de acordo com a preferência do desenvolvedor. As principais vistas dessa interface são:

- A vista *Project* (projeto, em português) permite o acesso a todos os recursos contidos na pasta principal do projeto do jogo, como *scripts*, *prefabs* e cenas;
- A vista *Scene* (cena, em português) permite a movimentação de objetos contidos na

cena selecionada na janela do projeto;

- A vista *Game* exibe o jogo em execução;
- A vista *Hierarchy* (hierarquia, em português) exibe os *GameObjects* atrelados à cena selecionada na janela *Project*. Esses objetos podem possuir classes filhas ou dependentes que são posicionadas em uma lista expansível abaixo do mesmo;
- A vista *Inspector* (inspetor, em português) exibe as informações gerais de qualquer elemento do projeto que seja selecionado. Também permite configurar os objetos da cena individualmente a partir da adição de componentes de colisão, imagens, efeitos sonoros, *scripts* de comportamento, entre outros.

O *Unity* também conta com uma classe base chamada *MonoBehaviour*, da qual todos os *scripts* que descrevem comportamentos de objetos são derivados. Essa classe atrela, entre outros atributos, um *GameObject* que é a instância do componente na cena, e um *Transform* que contém a posição do objeto no mundo e possui métodos com funções essenciais para o funcionamento do jogo. As principais funções presentes na classe *MonoBehaviour* estão descritas na Tabela 1.

Tabela 1 – Funções principais da classe *MonoBehaviour*.

Função	**Descrição**
Awake	É chamado quando o jogo está carregando.
FixedUpdate	É chamado a cada frame para atualização de física.
LateUpdate	É chamado a cada frame, caso o objeto esteja desativado.
OnCollisionEnter	É chamado caso o objeto comece a tocar outro objeto com colisão.
OnTriggerEnter	É chamado quando o objeto colide com outro.
OnTriggerExit	É chamado quando o objeto deixa de colidir com o outro.
Start	É chamado ao ínicio do jogo.
Update	É chamado a cada frame do jogo, se o objeto estiver ativo.
GetComponent	Retorna um objeto do tipo especificado pelo desenvolvedor.
Instantiate	Clona o objeto original e retorna a cópia.

Fonte: Adaptado de *Unity* 2019

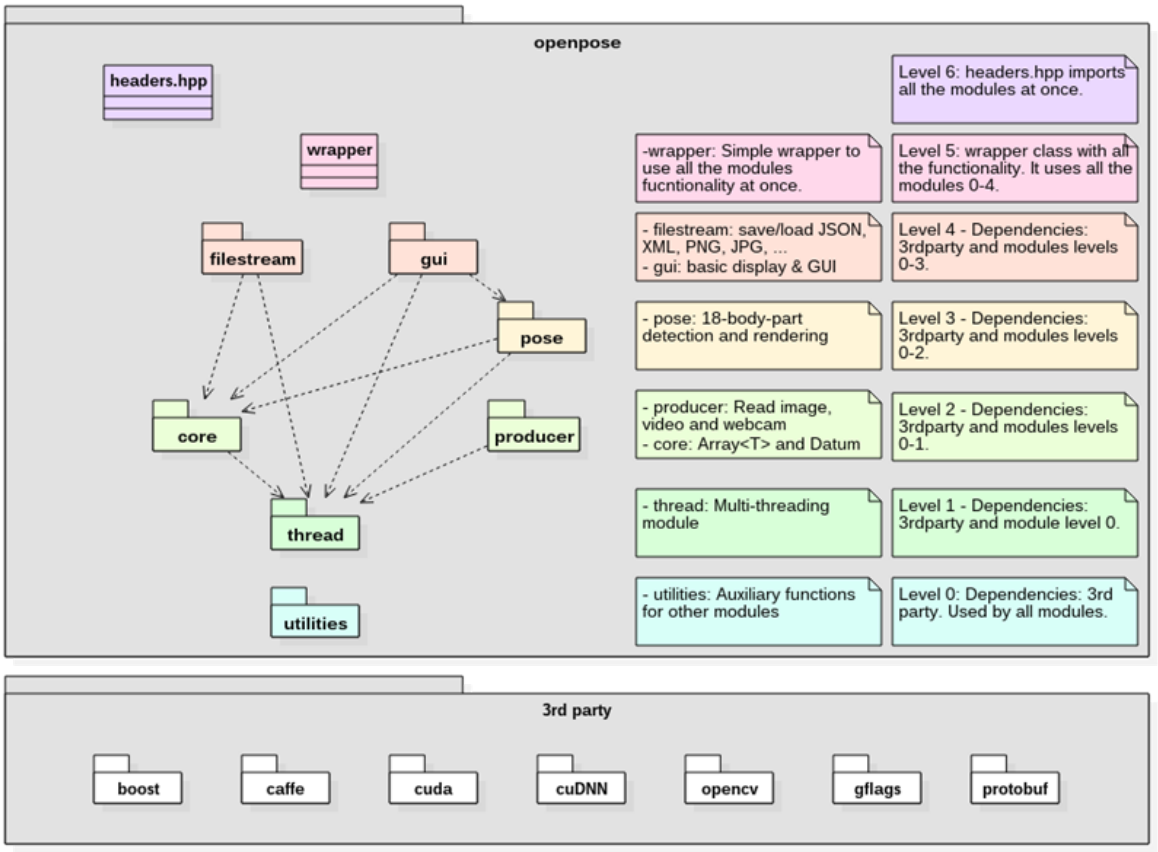
2.5 *OpenPose*

Segundo Nakai et al. 2018, *OpenPose* é uma biblioteca desenvolvida em C++ que realiza o reconhecimento dos membros das pessoas em tempo real, podendo ser utilizado com uma simples câmera web. A ferramenta adota o reconhecimento a partir da análise da imagem em tempo real usando *deep learning*, também fazendo o reconhecimento de partes do corpo. Como resultado, as imagens em movimento geradas pelo *OpenPose*, são marcas que representam articulações e membros que se sobrepõem a figura das pessoas, com precisão de reconhecimento alta, mesmo para múltiplas pessoas e em diferentes ambientes.

A biblioteca é disponibilizada na ferramenta de versionamento *Github* [*GitHub* 2019], por onde é possível contatar os desenvolvedores do projeto e a comunidade que o utiliza para obtenção de soluções para problemas relacionados a utilização da mesma. Para se utilizar a biblioteca, é necessário baixar o código ou clonar o repositório disponível na página principal do projeto, que vem com *scripts* para construção por meio da ferramenta *CMake* [*Kitware* 2019], por onde é possível construir diferentes versões da ferramenta, permitindo a definição do modo de processamento e do suporte (ou não) ao *Unity*. A partir desses *scripts*, são baixados os modelos de pontos de interesse gerado pelo Caffe e uma solução para Visual Studio14 [*Microsoft* 2019], configurada para gerar uma dll (Biblioteca de *links* dinâmicos), formato que permite portar a ferramenta para outras linguagens.

A Figura 6 representa o diagrama de componentes do sistema do *OpenPose*. Os módulos estão divididos em níveis de abstração, onde a camada azul representa encapsulamento de bibliotecas externas, utilizadas por todos os outros módulos (*Caffe*, CUDA, cuDNN, OpenCV, etc.); a camada verde mais escura é de *multithreading*; a camada verde clara representa a coleta de dados externa e o armazenamento destes dados pelo *OpenPose*; a camada amarela representa o módulo de processamento de dados e a renderização das marcações nas imagens, feita por bibliotecas externas; a camada rosa apresenta um encapsulador, que facilita a chamada dos métodos definidos; e a camada roxa representa um arquivo de cabeçalho para importar todos os módulos juntos.

Figura 6 – Componentes do *OpenPose*.

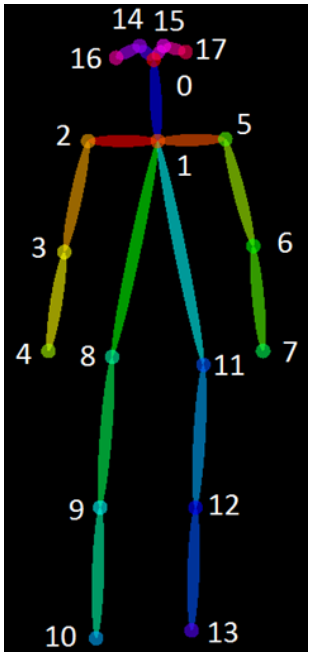


Fonte: *CMU Perceptual Computing Lab* 2019

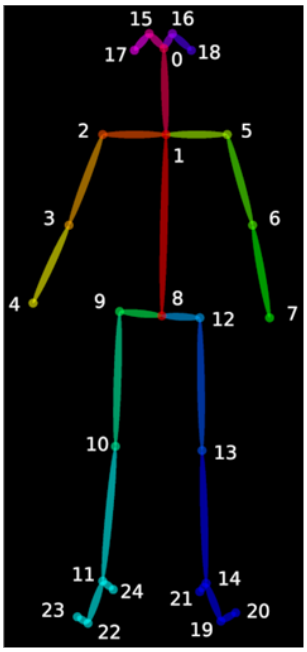
O *OpenPose* possui dois modos de processamento, em CPU e GPU. A versão mais estável da biblioteca é a desenvolvida para dispositivos compatíveis com a plataforma de computação paralela da NVIDIA, CUDA [NVIDIA 2019] nas versões 8 ou 10, juntamente com a biblioteca de primitivas para redes neurais cuDNN [NVIDIA 2019]. Também é recomendado o uso da IDE *Visual Studio* [*Microsoft* 2019] a partir da versão 14 [*CMU Perceptual Computing Lab* 2019].

A biblioteca dispõe de dois modelos diferentes para o corpo humano, um com 18 pontos de interesse (Figura 7a) e o outro com 25 pontos de interesse (Figura 7b), o qual tem o formato mais próximo do esqueleto do *Kinect*. A biblioteca também reconhece detalhadamente pontos de interesse na face (Figura 8a) e nas mãos (Figura 8b). O *OpenPose* pode ser esperado como um modelo eficaz para geração de dados de postura.

Figura 7 – Esqueletos gerados pelo *OpenPose*.

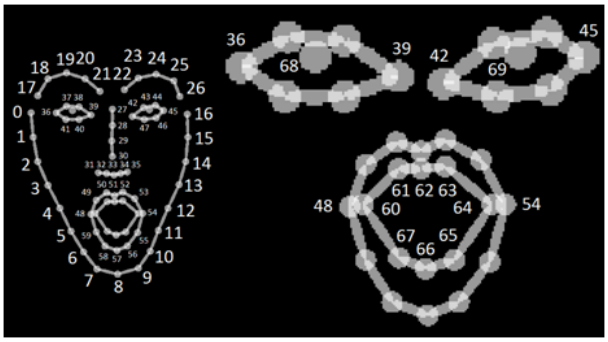


(a) Modelo *Pose*-18

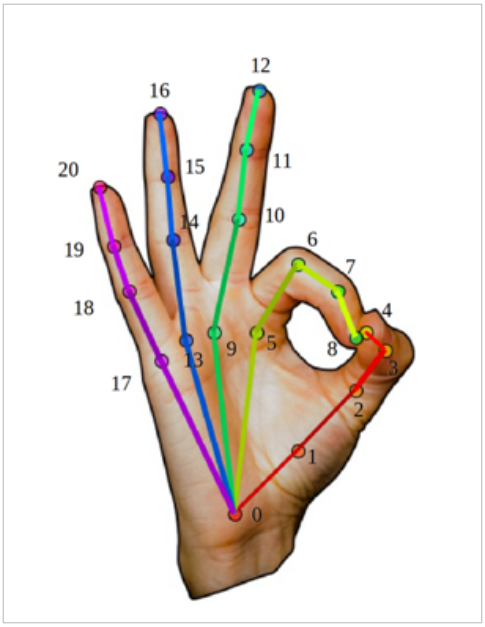


(b) Modelo *Pose*-25

Figura 8 – Modelos de rosto e mão do *OpenPose*.



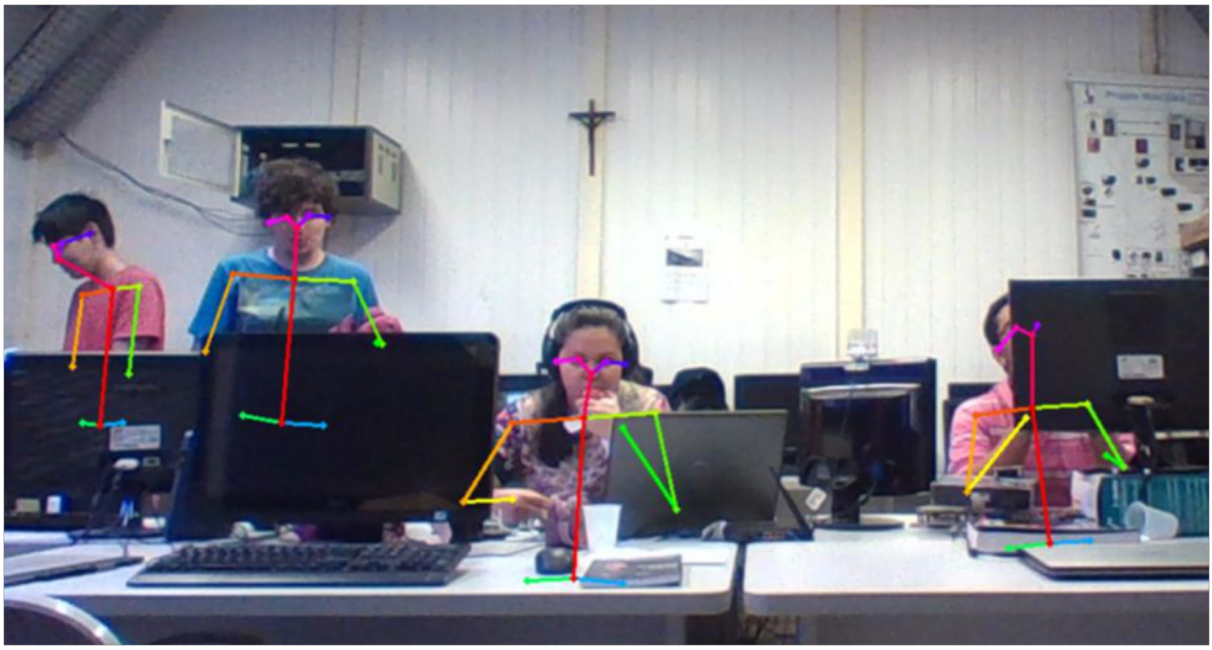
(a) Pontos de interesse da face humana.



(b) Pontos de interesse da mão humana.

A solução para construção do *OpenPose* vem com uma série de testes que podem ser executados e tomados como base para o desenvolvimento de uma aplicação que utilize as funcionalidades. A Figura 9 demonstra o funcionamento do caso de teste *OpenPoseDemo* disponível para construção na ferramenta.

Figura 9 – Testes da biblioteca.



Fonte: Elaborado pelo autor.

A partir dos testes com as demonstrações disponíveis para o *OpenPose*, chegou-se a conclusão de que seria interessante utilizá-lo para reconhecimento de movimentos no jogo *Skyway*, porque, além de ser uma biblioteca livre e de código aberto, o que torna possível a adaptação das funções às necessidades do projeto, esta apresentou bons resultados na identificação de pontos de interesse em imagens com oclusão parcial e também permite que sejam utilizadas imagens de diversos tipos de sensores para captura de imagens, aumentando a acessibilidade da ferramenta. O próximo capítulo descreve a adaptação da biblioteca para o módulo de reconhecimento do jogo.

3 Metodologia

Para reconstruir o módulo de reconhecimento de movimentos do jogo sem causar alterações significativas no funcionamento do mesmo, foi necessário fazer um estudo aprofundado das funções desempenhadas pelas classes e a relação entre elas no projeto. Também foi realizado um estudo sobre a utilização das ferramentas, *OpenPose* e *Unity*, em conjunto. E, por fim, comparou-se o desempenho do software atual com o anterior, desenvolvido com o *kit* de desenvolvimento do *Kinect*.

3.1 *Skyway*

Skyway é um *exergame* terapêutico desenvolvido por estudantes de Engenharia da Computação da Universidade Federal do Pará, com a ajuda de especialistas da área da fisioterapia, para tornar o sessões fisioterapêuticas mais agradáveis, com o foco em desenvolver os movimentos do tronco de pacientes com problemas de locomoção.

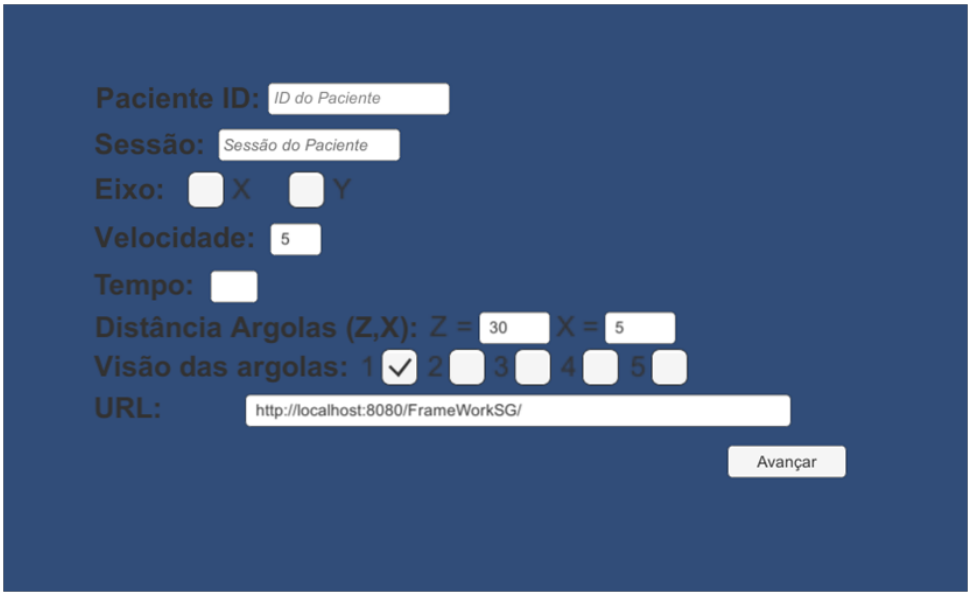
A interface do software foi desenvolvida no ambiente de desenvolvimento do *Unity*, sendo composto por duas cenas: a janela do fisioterapeuta e o jogo interativo. O jogo também conta com um módulo de captura de dados que estrutura em formato JSON e envia os dados de cada sessão, como erros e acertos das argolas, posições assumidas pelo jogador durante a sessão, etc., para uma ferramenta ainda em desenvolvimento (Figura 10) que tem como objetivo exibir os dados em forma de gráficos para que o fisioterapeuta tenha a possibilidade de analisar o progresso do paciente no tratamento.

Ao início do jogo, é chamada uma tela (Figura 11) na qual o usuário pode definir o link para o módulo de captura de dados e também definir as preferências em relação ao tempo de duração e dificuldade da sessão. O jogo só é iniciado após o preenchimento do formulário. O objetivo do jogador é movimentar o personagem, que voa em uma asa delta, para dentro de anéis que aparecem no cenário (Figura 12).

Figura 10 – *Framework* de captura de dados de jogos.

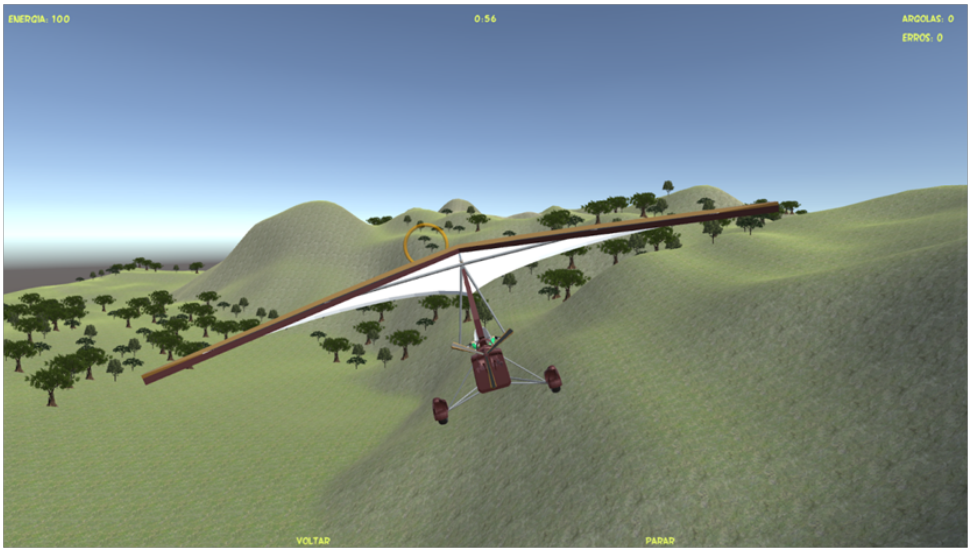
Fonte: Elaborado pelo autor.

Figura 11 – Tela inicial do jogo.



Fonte: Elaborado pelo autor.

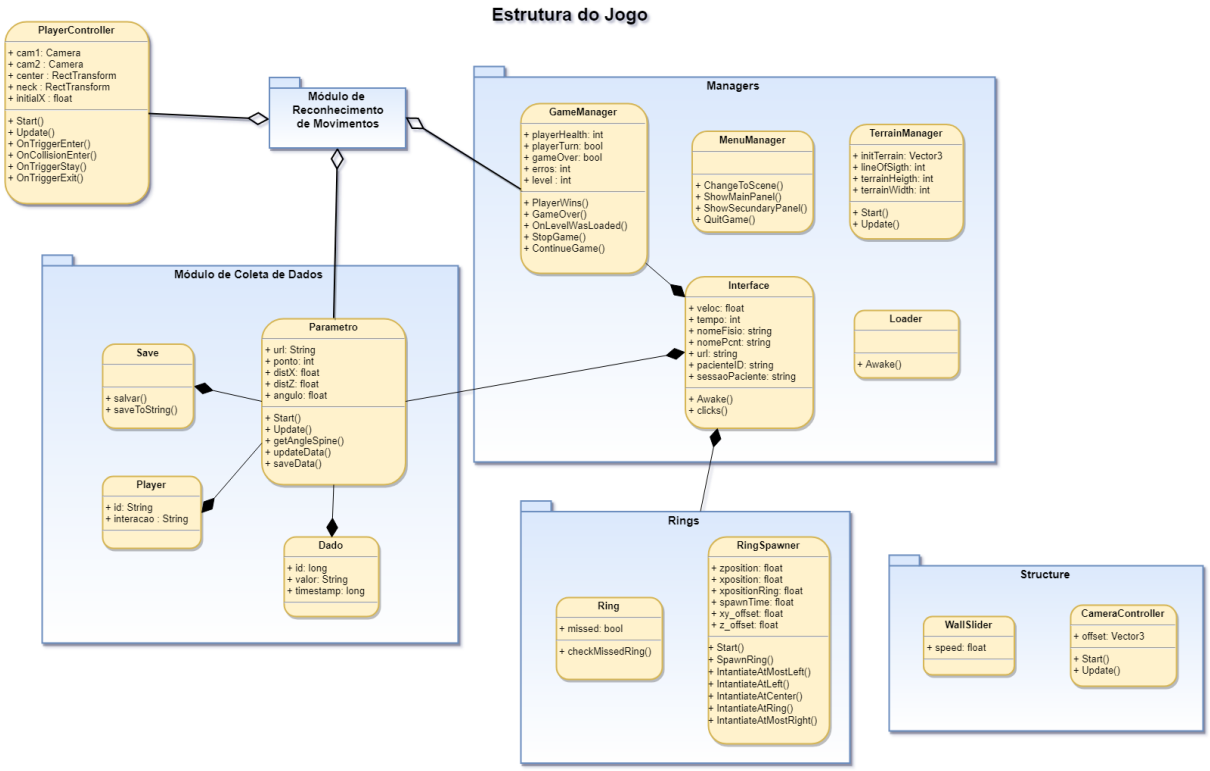
Figura 12 – Jogo em execução.



Fonte: Elaborado pelo autor.

Os módulos do *Skyway* foram desenvolvidos individualmente por vários programadores, sem que houvesse tempo de treinamento e sem uma documentação ou padronização definidas para o projeto, portanto, o código do jogo se tornou bastante complexo. Logo, foi necessário fazer uma interpretação dos componentes do jogo para que pudesse ser feita a substituição de tecnologias sem que houvesse grandes mudanças no funcionamento do mesmo. A Figura 13 mostra uma interpretação da estrutura de classes do jogo, separadas em pacotes referentes à organização de diretórios do projeto do Unity. A função de cada classe é descrita a seguir.

Figura 13 – Diagrama de classes do *Skyway*.



Fonte: Elaborado pelo autor.

3.1.1 Gerentes

A classe *Interface* recebe e gerencia os parâmetros definidos na cena inicial do jogo. Os quais são: identificador do paciente; número da sessão; o eixo de movimento do personagem (atualmente, o personagem só pode se movimentar no eixo X); a duração da sessão; as distâncias de instanciação das argolas nos respectivos eixos e o link do *framework* para captura de dados do jogo. Os dados definidos na primeira cena são mantidos após a chamada da cena principal do jogo.

A classe *GameManager* gerencia a sessão como um todo. Essa classe recebe uma instância de Interface para decrementar o tempo da sessão e também lida com eventos como iniciar o jogo, desenhar os ícones de acerto (Figura 14a) quando o personagem entra em contato com a argola, e de erro (Figura 14b) e definir a vitória ou derrota do jogador. Além disso, esse gerente verifica se o módulo de reconhecimento de poses está funcionando, caso contrário, o jogo é pausado até que o jogador seja detectado.

A classe *TerrainManager* instancia um terreno ao início do jogo e, a cada frame verifica se o personagem chegou ao final do mesmo. Caso positivo, o terreno atual é destruído e um novo é instanciado.

A classe *MenuManager* gerencia os textos exibidos na tela durante a execução

do jogo. Esta recebe uma instância do textitGameObject Player, que é utilizada para atualizar a posição dos textos na tela, de acordo com a posição do objeto no cenário.

A classe *Loader* checa se o objeto do *GameManager* está instanciado na cena, caso contrário é criada uma nova instância desse objeto. Essas funções são atreladas ao *GameObject* da câmera para que se mantenham os eventos visuais gerenciados pelo *GameManager*.

Figura 14 – Ícones de acerto e erro.

(a) Ícone de acerto.

(b) Ícone de erro.

3.1.2 Controles

A classe *PlayerController* é responsável principalmente pelo controle e colisão do personagem, mas também define o controle de mudança de câmera de 3ª pessoa para 1ª pessoa. Os atributos cam1 (3ª pessoa) e cam2 (1ª pessoa) recebem instâncias de câmeras e *center* (posição central do quadril) e *neck* (posição do pescoço) recebem uma instância de um objeto com as posições processadas pela biblioteca de detecção de movimentos.

Ao início do jogo, é definida a câmera padrão em 3ª pessoa e inicia as variáveis de posição do personagem, também recebendo uma instância das "Preferências" definidas na janela inicial, que define a velocidade de vôo do personagem, que aumenta juntamente com a dificuldade escolhida na interface inicial.

A cada *frame* do jogo, verifica-se se houve mudança na instância de câmera, que pode ser feita apertando a tecla "C". Também é calculada a diferença da posição dos transformadores dos atributos *neck* e *center*, que definem a rotação do modelo do personagem e movimentam o terreno do jogo de acordo com a posição do tronco do jogador obtida pelo processamento das imagens capturadas pelo sensor ou câmera utilizados.

Essa classe também controla a colisão do personagem com as paredes e as argolas. Quando o tempo de colisão com a parede excede sessenta segundos, é mostrada uma mensagem para que o jogador volte à posição normal. E quando há colisão com a argola, o jogador ganha mais vinte pontos de energia.

3.1.3 Argolas

A classe *Ring* controla as interações do personagem com as argolas. Caso o jogador passe por dentro do anel, este ganha um ponto; caso contrário, perde vinte pontos de energia.

A classe *RingSpawner* é responsável por desenhar as argolas na cena. Essa classe possui uma matriz para cada uma das cinco dificuldades disponíveis para a sessão. Essas matrizes armazenam números que servem como semente para o método *random*, que gera um número aleatório de zero a quatro que define a posição da próxima argolas a ser desenhada no cenário, onde zero representa 'bem à esquerda', um representa 'à esquerda', dois representa 'no centro', três representa 'à direita' e quatro 'bem à direita'. O anel é desenhado a uma distância pré-definida do centro do cenário.

3.1.4 Classes Estruturais

A classe *WallSlider* recebe uma instância do *GameObject Player*, utilizada para atualizar a posição das paredes do jogo de acordo com a posição do jogador no cenário.

A classe *CameraController* recebe uma instância do *GameObject Player* e atualiza a posição das câmeras de acordo com a posição do objeto no espaço do jogo.

3.2 Módulo de Captura de Dados

O módulo de captura de dados é gerenciado pela classe Parametro, a qual recebe as informações do paciente de *Player* e um vetor de Dado, que é atualizado com a quantidade de argolas alcançadas durante a sessão, os valores dos ângulos assumidos pelo tronco do jogador com o tempo em que aconteceu e a distância de instanciação das argolas nos eixos X e Y durante a sessão. A classe Parametro também recebe uma instância das mesmas juntas passadas ao *PlayerController* para ter o controle da angulação atingida pelo tronco do paciente durante a sessão.

3.3 Módulo de Reconhecimento de Poses

O módulo de reconhecimento de poses tem como função principal movimentar o personagem do jogo a partir da estimativa de movimentos do tronco do paciente. A Tabela 2 numera e descreve as funcionalidades desempenhadas por esse módulo, assim como a prioridade de desenvolvimento de cada uma delas.

Tabela 2 – Tabela de requisitos do módulo de reconhecimento de poses do Skyway.

Nº	**Descrição**	**Prioridade**
1	O jogador deve ter a possibilidade de movimentar o personagem no eixo horizontal por meio da movimentação lateral do tronco.	Alta
2	O jogador deve ter a possibilidade de movimentar o personagem no eixo vertical por meio da inclinação do tronco para frente e para trás.	Alta
3	O jogo deve pausar a execução e alertar o jogador, caso este não seja identificado pelo sensor de captura de imagens. Quando o jogador for identificado pelo sensor, o jogo deve retomar a execução.	Média

Fonte: Elaborado pelo autor.

O jogo originalmente foi desenvolvido com o *asset* do *Kinect* nativo para *Unity*, que consiste no encapsulamento das funções do *Kinect* em uma biblioteca em C# e conta com uma série de demonstrações de uso. A Tabela 3 mostra as principais funções presentes nesse *kit* de desenvolvimento.

Tabela 3 – Funcionalidades do asset do *Kinect* para *Unity*.

Função	**Descrição**
KinectManager	Componente mais básico da ferramenta de desenvolvimento. Controla os sensores e consulta os dados de imagem e profundidade.
AvatarController	Transfere os dados de movimentação do usuário do Kinect para o modelo em que está contido.
BackgroundColorImage	Exibe a imagem obtida pela câmera RGB.
BackgroundDepthImage	Exibe a imagem obtida pela câmera de profundidade.
KinectGestures	Detecta gestos realizados por usuários rastreados.
InteractionInputModule	Projeta um cursor na tela a partir da movimentação da mão do usuário.
InteractionManager	Rastreia movimentos da mão do usuário.

Fonte: Adaptado de RF *Solutions* 2019

A ferramenta de desenvolvimento também possui exemplos de execução. O módulo de reconhecimento de poses do *Skyway* foi desenvolvido baseado na demonstração *KinectRecorderDemo*, uma aplicação simples que consiste em guardar dados do corpo do jogador, controlada por meio de voz e comandos de teclado. Esta demo utiliza o *prefab Cubeman* (Figura 15), que possui objetos filhos referentes às juntas reconhecidas pelo *Kinect* e é controlado pelo *script CubemanController*, que recebe atribui dados de localização das respectivas juntas do jogador.

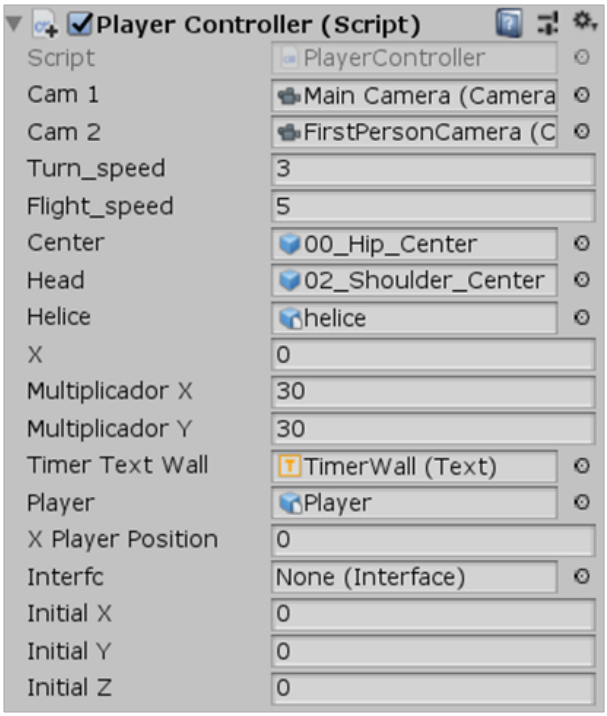
Figura 15 – Hierarquia do *Cubeman*.

Fonte: Elaborado pelo autor.

O script *PlayerController* receber as instâncias dos *GameObjects*, como mostra a Figura 16, "*00_Hip_Center*" (centro do quadril) e "*02_Shoulder_Center*" (centro dos ombros) nas variáveis públicas "*Center*" e "*Head*", que são utilizadas para fazer o cálculo da angulação do tronco do jogador.

A classe Parametro do módulo de coleta de dados instancia uma cópia do *GameObject Cuberman* por meio do método *GetComponent* para ter acesso às posições dos *GameObjects* das juntas, que são utilizados para enviar à ferramenta de exibição de dados a angulação do tronco do paciente.

Figura 16 – *PlayerController* configurado com os dados do *Cubeman*.



Fonte: Elaborado pelo autor.

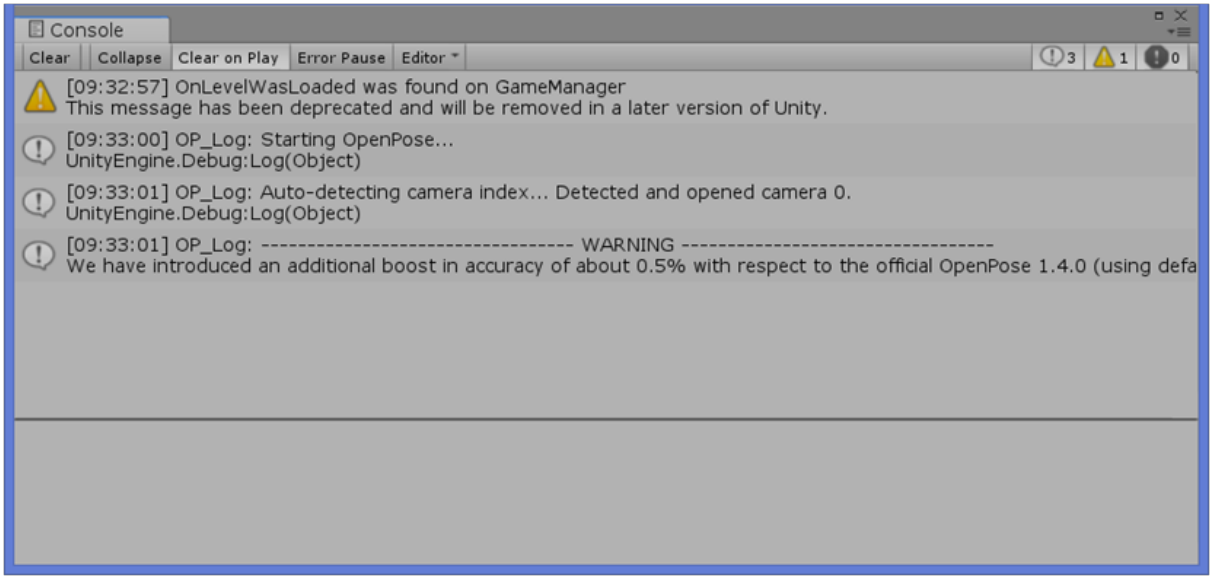
3.3.1 Reconstrução do Módulo de Reconhecimento de Poses

Para modernizar o módulo de reconhecimento de poses do *Skyway*, não foram necessárias mudanças na organização apresentada na Figura 13. Para isso, foi utilizado um *plugin* criado pelos mesmos desenvolvedores da biblioteca *OpenPose* com a finalidade de traduzir as funções da mesma para o *Unity*. Esse *plugin* desempenha as funções de [*CMU Perceptual Computing Lab* 2019]:

- Importação e controle da biblioteca *OpenPose*;
- Log de *debug* do *OpenPose* no *Unity* (Figura 17);
- Encapsular as funções do *OpenPose* para usuários de *Unity*:
  - Inicializar e finalizar a biblioteca;
  - Configurações da biblioteca (dimensões das redes neurais, entrada e saída, interface do usuário);
- Recebimento de dados do *OpenPose*:
  - Pontos de interesse do corpo, das mãos e rostos humanos;
  - Informações de *frame*.

- Exemplo de utilização simples.

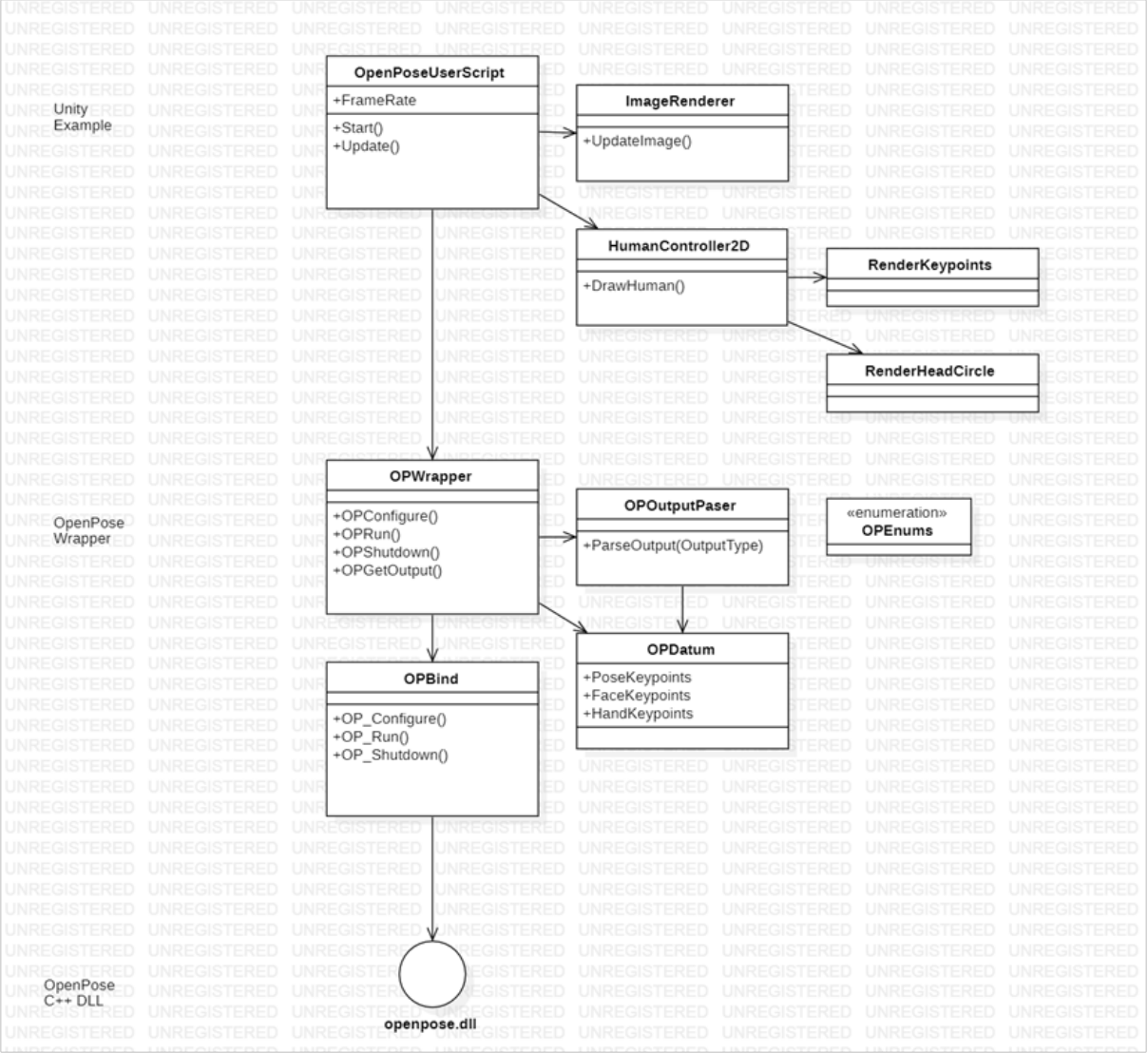
Figura 17 – Log de execução do *OpenPose* para *Unity*.



Fonte: Elaborado pelo autor.

O *plugin* do *OpenPose* é dividido como mostra o diagrama da Figura 18. O projeto baixado do *Github* provém de um arquivo executável que baixa automaticamente a versão mais recente do *OpenPose* específica para placas gráficas compatíveis com CUDA, mas o usuário pode substituir a versão da ferramenta de acordo com a necessidade. A classe *OPBind* encapsula as funções da biblioteca de C++ em métodos de baixo nível programados em C#. Essas funções são utilizadas nos métodos de mais alto nível da classe *OPWrapper*, que também recebe uma instância da classe *OPParser*, que analisa os dados de saída, e da classe *OPDatum*, que contém as unidades de dados da biblioteca. A classe *OPEnums*, contém estados definidos para variáveis de controle como o estado de funcionamento da ferramenta, tipos de *output*, tipo de dispositivo de entrada, etc.

Figura 18 – Diagrama de Classes do *Plugin* do *OpenPose* para *Unity*.



Fonte: *CMU Perceptual Computing Lab* 2019

A demonstração do *plugin* consiste em um pequeno canvas, onde são desenhados esqueletos sobre as imagens das pessoas identificadas no campo de visão do sensor utilizado (Figura 19). Ao lado direito da tela, existe um menu, que pode ser ocultado, que permite a alteração das opções de processamento da imagem controlados pelo *OpenPoseUserScript*, são essas: desenhar ou não a imagem capturada pelo sensor, número máximo de pessoas a serem detectadas pela ferramenta, limiar de renderização (variável que define a qualidade da imagem dos membros detectados a serem renderizados. para valores maiores que 0,5, são renderizados somente partes bem nítidas e menores que 0,1, são renderizados pontos oclusos, o que pode aumentar o número de falsos-positivos) e resolução da rede neural.

Figura 19 – Demo do *OpenPose* para *Unity*.

Fonte: [*CMU Perceptual Computing Lab* 2019]

Para identificar a posição das pessoas na imagem, essa demonstração do *OpenPose* utiliza o *prefab Human2D_Body25* cujos objetos filhos representam as juntas do modelo *Pose*-25 (Figura 7b). Esses objetos são atrelados ao *GameObject OpenPose*, juntamente com o *OpenPoseUserScript*, que configura as informações a serem passadas para os objetos que representam cada junta.

A reconstrução do módulo de reconhecimento de poses do jogo *Skyway* foi feita por meio da substituição dos objetos filhos do *GameObject Cubeman*, que possui as informações oriundas do processamento de imagem *Kinect* pelos objetos subordinados ao *Human2D_Body25*, que contém as informações processadas pela biblioteca *OpenPose*, na atribuição das variáveis *center* e *head* do *PlayerController*. Para isso, foi necessário adicionar um *GameObject* vazio ao projeto para incluir o script de configuração da biblioteca, *OpenPoseUserScript*. Também foram criadas duas variáveis públicas na classe Parametro do tipo *RectTransform* para receber as instâncias dos GameObjects representados pelas juntas.

3.3.2 Resultados

Após a substituição, obteve-se êxito na movimentação do personagem no eixo horizontal porém, apesar de a biblioteca ter suporte à estimativa da coordenada tridimensional por meio da imagem de mútiplas câmeras, essas funções não foram traduzidas pelo plugin

do *OpenPose* para o Unity, logo, não foi possível a implementação do requisito número 2 da Tabela 2.

O jogo também foi executado em máquinas disponíveis no momento do desenvolvimento para obtenção de comparação empírica de desempenho, visando avaliar a viabilidade do software atualizado para utilização com as formas disponíveis de processamento da bilioteca *OpenPose*.

Primeiramente, a nova ferramenta foi testada com as configurações padrões definidas na demonstração disponível no *plugin* para o *Unity* em uma máquina com uma placa gráfica NVIDIA GeForce GTX 1050 TI, com CUDA 8, e um processador AMD Ryzen 7 1700 3.0GHz. Nessas condições, o jogo não apresentou mudanças significativas no desempenho e na precisão do reconhecimento de movimentos.

Posteriormente, o jogo foi executado com a versão de processamento em CPU da biblioteca, na mesma máquina utilizada nos testes citados anteriormente e em uma máquina com um processador i7 5500 2.40GHz. Sob essas condições, o jogo congelou a execução em ambas as máquinas quando o processamento de imagens foi iniciado.

4 Conclusão e Trabalhos Futuros

Skyway é um *exergame* terapêutico desenvolvido com o objetivo de auxiliar a prática de exercícios referentes ao tratamento de pacientes com dificuldades na movimentação do tronco. O jogo foi desenvolvido no ambiente de desenvolvimento do *Unity*, utilizando o *kit* de desenvolvimento do dispositivo *Kinect* para obtenção e processamento dos dados de movimentação do jogador, utilizado para mover o personagem no jogo.

Porém, com a suspensão do suporte ao *Kinect* pela fabricante, o *Skyway* se tornou propenso a inconveniências relacionadas à manutenção do software. Portanto, este trabalho teve como objetivo substituir as ferramentas de desenvolvimento do *Kinect* por outras que desempenhassem um papel semelhante na detecção de movimentos do jogador.

O *OpenPose* é uma biblioteca de código aberto que utiliza visão computacional combinada com *deep learning* para reconhecimento de movimentos em imagens 2D. Além de estar em constante desenvolvimento atualmente, essa ferramenta torna possível a utilização de vários tipos de dispositivos para obtenção de imagens e, desse modo, foi escolhida para integrar o novo módulo de *motion tracking* do *Skyway*.

Para fazer essa substituição, foi necessário construir uma interpretação do código do jogo, que foi disposta em um diagrama de classes, e de um *plugin* que encapsula as funções do *OpenPose* especificamente para o *Unity*. Também foi desenvolvida uma lista de requisitos do sistema, contendo as funções do módulo de reconhecimento de movimentos do jogo, para fins de comparação com a versão reestruturada.

Foram atendidos dois dos três requisitos principais do módulo de reconhecimento de movimentos, tornando possível a movimentação do personagem no eixo horizontal e o gerenciamento do sensor pelo jogo, porém, devido à limitações do *plugin* utilizado para tradução das funções da biblioteca ao *Unity*, não foi possível implementar a movimentação do personagem no eixo vertical.

O jogo remodulado foi executado em máquinas disponíveis no momento de desenvolvimento do projeto para se obter uma avaliação empírica de desempenho. Não foram notadas grandes diferenças no desempenho quando executado em computadores com placas gráficas com suporte a CUDA, porém, apresentou um desempenho extremamente lento quando executado no modo de processamento em CPU, inviabilizando a utilização desse modo com as configurações utilizadas neste projeto.

4.1 Trabalhos Futuros

Como trabalhos futuros, é proposto que se utilize uma versão otimizada da biblioteca *OpenPose* para processamento em CPU, o que tornaria o jogo mais acessível em situações com poucos recursos computacionais. Também propõe-se a implementação do sistema de profundidade para o *plugin* do *Unity*, para viabilizar o desenvolvimento de exercício para movimento no eixo vertical do tronco dos pacientes.

Referências

ACHARYA, U. R. et al. Deep convolutional neural network for the automated detection and diagnosis of seizure using eeg signals. *Computers in Biology and Medicine*, v. 100, n. 1, p. 270–278, 2018. Citado na página 22.

BEKKER, T.; EGGEN, B. Designing for children's physical play. In: EXTENDED ABSTRACTS ON HUMAN FACTORS IN COMPUTING SYSTEMS, 2008, Florência. [S.l.], 2008. p. 2871–2876. Citado na página 19.

BERKOVSKY, S. et al. Design games to motivate physical activity. In: PROCEEDINGS OF THE 4TH INTERNATIONAL CONFERENCE ON PERSUASIVE TECHNOLOGY, 2009, Claremont. [S.l.], 2009. p. 26–29. Citado na página 19.

BROWN, E.; CAIRNS, P. A grounded investigation of game immersion. In: *CHI '04 Extended Abstracts on Human Factors in Computing Systems*. Nova Iorque: ACM, 2004. (CHI EA '04), p. 1297–1300. Citado 2 vezes nas páginas 13 e 19.

COWLEY, B. et al. Toward an understanding of flow in video games. *Computers in Entertainment*, Association for Computing Machinery, v. 6, p. 1–27, 2008. Citado na página 18.

CSIKSZENTMIHALYI, M. *Beyond Boredom and Anxiety*. São Francisco: Jossey-Bass Publishers, 1975. (The Jossey-Bass behavioral science series). Citado 3 vezes nas páginas 13, 17 e 18.

ENTERTAINMENT SOFTWARE ASSOCIATION. *2018 ESSENTIAL FACTS ABOUT THE COMPUTER AND VIDEO GAME INDUSTRY*. 2019. Disponível em: <https://www.theesa.com/wp-content/uploads/2019/03/ESA_EssentialFacts_2018.pdf>. Acesso em: 6 jul. 2019. Citado na página 13.

FERNáNDEZ-MANJóN, B. et al. Challenges of serious games. *EAI Endorsed Transactions on Game-Based Learning*, European Alliance for Innovation(EAI), v. 2, n. 6, p. 55–65, 2015. Citado na página 19.

GAO, P. et al. Large margin structured convolution operator for thermal infrared object tracking. In: 2018 24TH INTERNATIONAL CONFERENCE ON PATTERN RECOGNITION (ICPR), 2018, Beijing. [S.l.], 2018. p. 2380–2385. Citado na página 20.

GRAMMATIKOPOULOU, A. et al. Motion analysis of parkinson diseased patients using a video game approach. In: . [S.l.: s.n.], 2019. Citado na página 16.

Grif, M. G.; Prikhodko, A. L. Approach to the sign language gesture recognition framework based on hamnosys analysis. In: *2018 XIV International Scientific-Technical Conference on Actual Problems of Electronics Instrument Engineering (APEIE)*. [S.l.: s.n.], 2018. p. 426–429. Citado na página 16.

HAN, J. et al. Enhanced computer vision with microsoft kinect sensor: A review. *IEEE Transactions on Cybernetics*, IEEE, v. 43, n. 5, p. 1318–1334, 2013. Citado na página 20.

HUIZINGA, J. *Homo Ludens*. Tradução João Paulo Monteiro. 5. ed. São Paulo: Perspectiva, 2001. 256 p. Citado na página 12.

LONGUIDICE, B.; BARTON, M. *Vintage Game Consoles: An Inside Look at Apple, Atari, Commodore, Nintendo, and the Greatest Gaming Platforms of All Time*. 1. ed. [S.l.]: Routledge, 2014. 368 p. Citado na página 12.

MASSIMINI, F.; CARLI, M. The systematic assessment of flow in daily experience. In: _____. *Optimal Experience: Psychological Studies of Flow in Consciousness*. 1. ed. Nova Iorque: University of Cambrige, 1992. cap. 4, p. 266–288. Citado na página 18.

MICROSOFT. *Kinect para Windows*. 2019. Disponível em: <https://developer.microsoft.com/pt-br/windows/kinect>. Acesso em: 6 jul. 2019. Citado na página 14.

MICROSOFT. *Kinect sensor components*. 2019. Disponível em: <https://support.xbox.com/en-TT/xbox-360/kinect/kinect-sensor-components>. Acesso em: 6 jul. 2019. Citado na página 20.

NAKAI, M. et al. Prediction of basketball free throw shooting by openpose. In: PROCEEDINGS OF THE 32ND ANNUAL CONFERENCE OF THE JAPANESE SOCIETY FOR ARTIFICIAL INTELLIGENCE, 2018, Tóquio. [S.l.], 2018. p. 3Pin139. Citado na página 24.

NVIDIA. *CUDA Toolkit*. 2019. Disponível em: <https://developer.nvidia.com/cuda-toolkit>. Acesso em: 6 jul. 2019. Citado na página 26.

NVIDIA. *NVIDIA cuDNN*. 2019. Disponível em: <https://developer.nvidia.com/cudnn>. Acesso em: 6 jul. 2019. Citado na página 26.

PEITZ, M.; WALDFOGEL, J. *The Oxford Handbook of the Digital Economy*. 1. ed. S.l.: Oxford University Press, 2012. 614 p. Citado na página 13.

PIROVANO, M. et al. Exergaming and rehabilitation: A methodology for the design of effective and safe therapeutic exergames. *Entertainment Computing*, IEEE Computer Society Press, v. 14, p. 55–65, 2016. Citado 2 vezes nas páginas 14 e 19.

PLANTARD, P.; SHUM, H. P. H.; MULTON, F. Ergonomics measurements using kinect with a pose correction framework. In: THE 4TH INTERNATIONAL DIGITAL HUMAN MODELING SYMPOSIUM (DHM2016), 2016, Montreal. [S.l.], 2016. p. 15–17. Citado 2 vezes nas páginas 20 e 21.

PLANTARD, P.; SHUM, H. P. H.; MULTON, F. Filtered pose graph for efficient kinect pose reconstruction. *Multimedia Tools and Applications*, Springer Verlag, v. 76, n. 3, p. 4291–4312, 2017. Citado na página 21.

Qiao, S.; Wang, Y.; Li, J. Real-time human gesture grading based on openpose. In: *2017 10th International Congress on Image and Signal Processing, BioMedical Engineering and Informatics (CISP-BMEI)*. [S.l.: s.n.], 2017. p. 1–6. Citado na página 16.

REDHEFFER, R. A machine for playing the nim. *The American Mathematical Monthly*, v. 55, n. 6, p. 343–349, 1948. Citado na página 12.

RF Solutions. *K2-docs*. 2019. Disponível em: <https://ratemt.com/k2docs/>. Acesso em: 6 jul. 2019. Citado na página 35.

RIOS, L. R. S. Visão computacional. *Departamento de Ciência da computação – Universidade Federal da Bahia (UFBA)*, Salvador, 2010. Citado na página 19.

ROCHA, R. V. d.; BITTERCOURT, I. I.; ISOTANI, S. Análise, projeto, desenvolvimento e avaliação de jogos sérios e afins: Uma revisão de desafios e oportunidades. In: BRAZILIAN SYMPOSIUM ON COMPUTERS IN EDUCATION (SIMPóSIO BRASILEIRO DE INFORMáTICA NA EDUCAçãO-SBIE), 2015, Maceió. [S.l.], 2015. p. 692. Citado na página 17.

RUTTER, J.; BRYCE, J. *Understanding Digital Games*. 1. ed. [S.l.]: SAGE Publications Ltd., 2006. 272 p. Citado na página 12.

SCHMIDT, S. et al. Impact of virtual environments on motivation and engagement during exergames. *2018 Tenth International Conference on Quality of Multimedia Experience (QoMEX)*, IEEE, p. 1–6, 2018. Citado na página 14.

SOARES, T. P. et al. Sky way: Jogo para reabilitação de equilíbrio corporal. In: ESCOLA REGIONAL DE INFORMáTICA NORTE 2016 (ERIN), 2016, Belém. [S.l.]: Sociedade Brasileira de Computação (SBC), 2016. p. 6. Citado na página 14.

STONY BROOK UNIVERSITY LIBRARIES. *Tennis for Two*. 2019. Disponível em: <https://sexedi.ga/09174746.pdf>. Acesso em: 6 jul. 2019. Citado na página 12.

Berkeley Vision and Learning Center (BVLC). *Caffe*. 2019. Disponível em: <https://caffe.berkeleyvision.org>. Acesso em: 6 jul. 2019. Citado na página 23.

CMU Perceptual Computing Lab. *OpenPose*. 2019. Disponível em: <https://github.com/CMU-Perceptual-Computing-Lab/openpose>. Acesso em: 6 jul. 2016. Citado 3 vezes nas páginas 14, 26 e 40.

CMU Perceptual Computing Lab. *OpenPose Unity Plugin*. 2019. Disponível em: <https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin>. Acesso em: 6 jul. 2016. Citado 3 vezes nas páginas 15, 37 e 39.

GitHub. *GitHub*. 2019. Disponível em: <https://github.com>. Acesso em: 6 jul. 2019. Citado na página 25.

Kitware. *CMake*. 2019. Disponível em: <https://cmake.org>. Acesso em: 6 jul. 2019. Citado na página 25.

Microsoft. *Azure Kinect DK*. 2019. Disponível em: <https://azure.microsoft.com/pt-br/services/kinect-dk/>. Acesso em: 6 jul. 2019. Citado na página 21.

Microsoft. *Visual Studio*. 2019. Disponível em: <https://visualstudio.microsoft.com/pt-br/>. Acesso em: 6 jul. 2019. Citado 2 vezes nas páginas 25 e 26.

Orbbec. *Orbbec3D*. 2019. Disponível em: <https://orbbec3d.com>. Acesso em: 6 jul. 2019. Citado na página 22.

Stereolabs. *StereoLabsZED*. 2019. Disponível em: <https://www.stereolabs.com/>. Acesso em: 6 jul. 2019. Citado na página 22.

Unity. *MonoBehaviour*. 2019. Disponível em: <https://docs.unity3d.com/ScriptReference/MonoBehaviour.html>. Acesso em: 6 jul. 2019. Citado na página 24.

VicoVR. *VicoVR*. 2019. Disponível em: <https://vicovr.com>. Acesso em: 6 jul. 2019. Citado na página 21.

TULIPER, A. *Introduction to the Hololens*. 2019. Disponível em: <https://msdn.microsoft.com/en-us/magazine/mt788624.aspx>. Acesso em: 6 jul. 2019. Citado na página 21.

UNITY. *Unity*. 2019. Disponível em: <https://unity.com/>. Acesso em: 6 jul. 2019. Citado na página 14.

UNNIKRISHNAN, R.; MOAWAD, K.; BHAVANI, R. R. physiotherapy toolkit using video games and motion tracking technologies. *2013 IEEE Global Humanitarian Technology Conference: South Asia Satellite (GHTC-SAS)*, IEEE, v. 42, n. 6, p. 90–95, 2013. Citado na página 13.

VAGHETTI, C. A. O.; BOTELHO, S. S. d. C. Ambientes virtuais de aprendizagem na educação física: Uma revisão sobre a utilização de exergames. *Ciências Cognição*, v. 15, n. 1, p. 78–88, 2010. Citado na página 19.

VAUGHAN-NICHOLS, S. J. Game-console makers battle over motion-sensitive controllers. *Computer*, IEEE Computer Society Press, Los Alamitos, v. 42, n. 6, p. 13–15, 2009. Citado na página 12.

Apêndices

APÊNDICE A – Código Fonte do *Skyway*

Neste Apêndice encontram-se os códigos fonte contidos nos *scripts* do jogo *Skyway*, cujas funções são descritas no Capítulo 3 deste trabalho. As listagens apresentam, respectivamente, o código de: *Interface.cs*, *GameManager.cs*, *TerrainManager.cs*, *MenuManager.cs*, *Loader.cs*, *PlayerController.cs*, *Ring.cs*, *RingSpawner.cs*, *WallSlider.cs*, *CameraController.cs*, Parametro.cs, *Player.cs*, Dado.cs e *Save.cs*.

Interface.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using UnityEngine.UI;
4
5  public class Interface : MonoBehaviour {
6
7      public static Interface instance = null;
8      public bool xToggle;
9    public bool yToggle;
10   public float veloc;
11   public float distz;
12     public float distx;
13   public int tempo;
14   public int visArg;
15   public string nomeFisio;
16   public string nomePcnt;
17     public string url;
18     public string pacienteID;
19     public string sessaoPaciente;
20     public static int dif;
21
22     public Dropdown myDropDown;
23
24     void Awake()
25     {
26         if (instance == null)
27             instance = this;
28         else if (instance != this)
29             Destroy(gameObject);
```

```
30      }
31
32    public void clicks ()
33    {
34    //  nomeFisio = GameObject.Find ("Text_Fisioterapeuta").
          GetComponent<Text> ().text;
35    //  nomePcnt = GameObject.Find ("Text_Paciente").GetComponent<
          Text> ().text;
36          pacienteID = GameObject.Find("Text_PacienteID").
              GetComponent<Text>().text;
37          sessaoPaciente = GameObject.Find("Text_Sessao").
              GetComponent<Text>().text;
38          Debug.Log(sessaoPaciente);
39      url = GameObject.Find ("Text_URL").GetComponent<Text> ().text;
40      xToggle = GameObject.Find ("X").GetComponent<Toggle> ().isOn;
41      yToggle = GameObject.Find ("Y").GetComponent<Toggle> ().isOn;
42          //Debug.Log(xToggle + "----" + yToggle);
43          veloc = float.Parse(GameObject.Find ("Text_Velocidade").
              GetComponent<Text> ().text);
44      tempo = int.Parse(GameObject.Find ("Text_Tempo").GetComponent<
            Text> ().text);
45      distz = float.Parse(GameObject.Find ("Text_DistanciaZ").
            GetComponent<Text> ().text);
46          distx = float.Parse(GameObject.Find("Text_DistanciaX").
              GetComponent<Text>().text);
47          //url = GameObject.Find("Text_URL").GetComponent<Text> ().
              text;
48
49      visArg = getNum ();
50          dif = getDifNum();
51          DontDestroyOnLoad(transform.gameObject);
52
53      Application.LoadLevel ("Game");
54      }
55
56    private int getNum ()
57    {
58      ToggleGroup group = GameObject.Find ("Visao_argolas").
            GetComponent<ToggleGroup> ();
59      string active = null;
60      int actvNum;
61
```

```
62      foreach (var item in group.ActiveToggles())
63      {
64        active = item.name;
65        break;
66      }
67
68      actvNum = int.Parse(GameObject.Find (active).
           GetComponentInChildren<Text>().text);
69
70      return actvNum;
71    }
72
73      private int getDifNum()
74      {
75          ToggleGroup group = GameObject.Find("Dificuldade").
              GetComponent<ToggleGroup>();
76          string active = null;
77          int actvNum;
78
79          foreach (var item in group.ActiveToggles())
80          {
81              active = item.name;
82              break;
83          }
84
85          actvNum = int.Parse(GameObject.Find(active).
              GetComponentInChildren<Text>().text);
86          Debug.Log(actvNum);
87          return actvNum;
88      }
89  }
```

GameManager.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using UnityEngine.UI;
4  using UnityEngine.SceneManagement;
5  using OpenPose;
6
7  public class GameManager : MonoBehaviour {
```

```
8
9   public static GameManager instance = null;
10  public GameObject controlPanel;
11  public Text levelText;
12  public Text timerText;
13  public Text argolasText;
14    public Text errosText;
15  public Text healthText;
16  public bool y_axis = false;
17  public Interface interfc;
18
19    public RawImage imagemAcerto;
20    public RawImage imagemErro;
21
22    [HideInInspector] public bool playersTurn = true;
23  [HideInInspector] public bool spawnRings = true;
24  [HideInInspector] public bool gameOver = false;
25  [HideInInspector] public int argolas = 0;
26    [HideInInspector] public int erros = 0;
27  [HideInInspector] public int playerHealth = 1000;
28  [HideInInspector] public static int level = 1;
29
30  public float minutes = 3;
31  private float seconds = 59;
32  private float startMinutes;
33
34    // Output control
35    private OPDatum datum;
36
37    // Awake is always called before any Start functions
38    void Awake ()
39  {
40    playersTurn = false;
41    startMinutes = minutes;
42
43    if (instance == null)
44      instance = this;
45    else if (instance != this)
46      Destroy(gameObject);
47
48    // Sets this to not be destroyed when reloading scene
49    DontDestroyOnLoad(gameObject);
```

```
50
51      if (Interface.instance.yToggle) {
52              Destroy (GameObject.Find ("LeftWall"));
53        Destroy (GameObject.Find ("RightWall"));
54              //Debug.Log("Vertical was chosen");
55      } else {
56              Destroy (GameObject.Find ("TopWall"));
57        Destroy (GameObject.Find ("BottomWall"));
58              //Debug.Log("Horizontal was chosen");
59          }
60      interfc = GameObject.Find ("Preferencias").GetComponent<
           Interface> ();
61      minutes = interfc.tempo;
62      minutes--;
63
64          //imagens ós aparecem ao entrarem em contato com a argola
65          imagemAcerto.enabled = false;
66          imagemErro.enabled = false;
67      InitGame();
68    }
69
70    void InitGame ()
71    {
72          levelText.text = "íNvel:␣" + level;
73          StartCoroutine("CountDown");
74
75      }
76
77    void Update ()
78    {
79      if (playersTurn) {
80        controlPanel.SetActive (true);
81
82        if(playerHealth == 0)
83          GameOver();
84
85        if (seconds <= 0) {
86          seconds = 59;
87          if (minutes >= 1) {
88            minutes--;
89          } else {
90            minutes = 0;
```

```
91          seconds = 0;
92          //timerText.text = minutes.ToString("f0") + ":0" +
                seconds.ToString("f0");
93          PlayerWins ();
94        }
95      } else {
96        seconds -= Time.deltaTime;
97      }
98
99      if (!gameOver) {
100       SetArgolasText ();
101               SetErrosText();
102               healthText.text = "Energia: " + playerHealth.
                    ToString ();
103       if (Mathf.Round (seconds) <= 9)
104         timerText.text = minutes.ToString ("f0") + ":0" + seconds
                .ToString ("f0");
105       else
106         timerText.text = minutes.ToString ("f0") + ":" + seconds.
                ToString ("f0");
107     }
108   }
109
110   }
111
112 void GameOver()
113 {
114   playersTurn = false;
115   spawnRings = false;
116   gameOver = true;
117   timerText.text = "";
118   levelText.text = "Fim de Jogo!";
119   argolasText.text = "";
120   healthText.text = "";
121 }
122
123 void PlayerWins ()
124 {
125   playersTurn = false;
126   spawnRings = false;
127   gameOver = true;
128   timerText.text = "";
```

```
129     levelText.text = "éParabns,␣êvoc␣venceu!\{\}nArgolas:␣" +
            argolas;
130     argolasText.text = "";
131     healthText.text = "";
132   }
133
134   void OnLevelWasLoaded(int index)
135   {
136     level++;
137     InitGame();
138     }
139
140   void SetArgolasText()
141   {
142     argolasText.text = "Argolas:␣" + argolas.ToString ();
143   }
144     //define as argolas erradas
145     void SetErrosText()
146     {
147         errosText.text = "Erros:␣" + erros.ToString();
148     }
149
150
151     IEnumerator CountDown()
152   {
153         if (OpenPose.OPWrapper.state == OPState.Running)
154         {
155               System.Func<bool> playerDetected = new System.Func<
                      bool>(peopleDetected);
156               yield return new WaitUntil(playerDetected);
157         }
158
159         yield return new WaitForSeconds(1);
160
161         levelText.text = "3";
162         yield return new WaitForSeconds(1);
163         levelText.text = "2";
164         yield return new WaitForSeconds(1);
165         levelText.text = "1";
166         yield return new WaitForSeconds(1);
167         levelText.text = "áV!";
168         yield return new WaitForSeconds(1);
```

```
169         levelText.text = "";
170 
171         playersTurn = true;
172     }
173 
174     private bool peopleDetected()
175     {
176         if (datum.poseKeypoints == null || datum.poseKeypoints.
               Empty())
177             return false;
178 
179         return true;
180     }
181 
182   public void GoBack()
183   {
184     playersTurn = false;
185     spawnRings = false;
186     argolas = 0;
187     minutes = startMinutes;
188     seconds = 59;
189     controlPanel.SetActive(false);
190 
191 //    SceneManager.UnloadScene("Game");
192     SceneManager.LoadScene ("StartMenu");
193   }
194 
195   public void StopGame()
196   {
197     playersTurn = false;
198     spawnRings = false;
199 
200     //desabilitar botao de parar
201     //habilitar botao de continuar
202   }
203 
204   public void ContinueGame()
205   {
206     playersTurn = true;
207     spawnRings = true;
208 
209     //desabilitar botao de continuar
```

```
210     //habilitar botao de parar
211   }
212
213   public void RestartGame()
214   {
215     playersTurn = false;
216     spawnRings = false;
217     argolas = 0;
218     minutes = startMinutes;
219     seconds = 59;
220     controlPanel.SetActive(false);
221
222     //voltar com player e camera para o inicio
223     InitGame();
224   }
225 }
```

TerrainManager.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class TerrainManager : MonoBehaviour{
5    public GameObject PlayerObject;
6    private Vector3 initTerrain = new Vector3(0,-100,0);
7    private GameObject _terrain;
8    private GameObject oldTerrain = null;
9    private GameObject newTerrain = null;
10   private int lineOfSigth = 1000;
11   private int terrainHeigth = 450; //informacoes setadas nos
       terrenos
12   private int heigthCorrection = -100;
13   private int terrainWidth = 1500;
14   private int numLoop = 0; //quantos loops na mesma fase ja foram
       feitos, usado para calcular o deslocamento do novo terreno em
       relacao a origem
15   private int terrainLength =1500*3; // *3 porque estou buscando os
        prefabs que sao a çcombinaao de 3 terrenos de 1500
16   // talvez ajustar a velocidade seja uma boa os ter
17
18   void Start(){
```

```
19      _terrain = Resources.Load("Nivel"+GameManager.level+"/Nivel"+
            GameManager.level) as GameObject;
20      oldTerrain = (GameObject)GameObject.Instantiate(_terrain,
            initTerrain, Quaternion.identity); //instancia o terreno
21      numLoop++;
22    }
23
24    void Update(){
25      Vector3 playerPosition = new Vector3(PlayerObject.transform.
            position.x, PlayerObject.transform.position.y, PlayerObject.
            transform.position.z); //Posicao da asa delta
26      // if(GameManager.gametype == porLvl){ //verificar o tipo de
            sessao, progressiva ou por lvl
27      //
28      //Debug.Log(terrainLength*numLoop-lineOfSigth);
29      if((playerPosition.z > (terrainLength*numLoop-lineOfSigth)) &&
            (newTerrain == null)){
30        newTerrain = (GameObject)GameObject.Instantiate(_terrain, new
              Vector3(0,heigthCorrection,terrainLength*numLoop),
            Quaternion.identity); //instancia o novo terreno
31        numLoop++;
32      }
33
34      if(playerPosition.z > (terrainLength*(numLoop-1)) && newTerrain
            != null){ // destroi o terreno anterior
35        Destroy(oldTerrain);
36        // Debug.Log("Destruction Time!!!");
37        oldTerrain = newTerrain;
38        newTerrain = null;
39      }
40      //}
41      //if(GameManager.gametype == progressivo){ //verificar o tipo
            de sessao, progressiva ou por lvl
42      //  corrigir o caso do lvl 5 nao ter um proximo nivel
43      //  Lembrar que para conectar os terrenos de nivel diferentes
            niveis e nesse sario utilizar um terreno de link
44      //  boa sorte o/
45      //  if((playerPosition.z > (terrainLength*numLoop-lineOfSigth))
            && (newTerrain == null) && (linkTerrain == null)){
46      //    linkTerrain = (GameObject)GameObject.Instantiate(
            Resources.Load("Links/"+GameManager.level+"-"+GameManager.
            level+1), new Vector3(0,-100,terrainLength*numLoop),
```

```
        Quaternion.identity);
47      //    //numLoop++;
48      //  }
49      //}
50    }
51  }
```

MenuManager.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using UnityEngine.SceneManagement;
4
5  public class MenuManager : MonoBehaviour {
6
7    public GameObject player, mainPanel;
8
9    // Update is called once per frame
10   void Update ()
11   {
12     player.transform.Rotate(0.0f,20 * Time.deltaTime, 0.0f);
13   }
14
15   public void ChangeToScene (string scene)
16   {
17     SceneManager.LoadScene (scene);
18     //Application.LoadLevel (scene);
19   }
20
21   public void ShowMainPanel(GameObject panel)
22   {
23     player.SetActive (true);
24     mainPanel.SetActive (true);
25     panel.SetActive (false);
26   }
27
28   public void ShowSecondaryPanel (GameObject panel)
29   {
30     player.SetActive (false);
31     mainPanel.SetActive (false);
32     panel.SetActive (true);
```

```
33    }
34
35    public void QuitGame()
36    {
37      Application.Quit();
38    }
39  }
```

Loader.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Loader : MonoBehaviour {
5
6    public GameObject gameManager;
7
8
9    void Awake ()
10   {
11     if (GameManager.instance == null)
12       Instantiate(gameManager);
13   }
14 }
```

PlayerController.cs

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using System.Collections;
4  using System.Collections.Generic;
5  using OpenPose;
6
7  public class PlayerController : MonoBehaviour {
8      //camera
9      public Camera cam1;
10     public Camera cam2;
11
12     //public Text countText;
13     public float turn_speed = 3;
```

```
14  public float flight_speed = 5;
15  public RectTransform center;
16  public RectTransform head;
17  public GameObject helice;
18    //public GameObject parede = GameObject.Find("Parede");
19    public float x;
20  public float multiplicadorX;
21  public float multiplicadorY;
22  float moveHorizontal;
23  float moveVertical;
24
25    //---------MODIFICACOES----------//
26    public Text timerTextWall;
27    private float startTime;
28    private bool count = false;
29    //---------MODIFICACOES----------//
30
31    //-------------çõModificaes finais--------------//
32    public GameObject player;
33    public float xPlayerPosition;
34    //-------------çõModificaes finais--------------//
35
36    private Rigidbody rb;
37  bool knt;
38    public Interface interfc;
39  Parametro parametro;
40  public float initialX;
41  public float initialY;
42  public float initialZ;
43  public Save save;
44
45    // Use this for initialization
46    void Start ()
47  {
48        //-----camera------//
49        cam1.enabled = true;
50        cam2.enabled = false;
51        //-----------------//
52    initialX = gameObject.transform.position.x;
53    initialY = gameObject.transform.position.y;
54    initialZ = gameObject.transform.position.z;
55        interfc = GameObject.Find("Preferencias").GetComponent<
```

```
            Interface>();
56          flight_speed = interfc.veloc;
57   //       turn_speed = flight_speed - 2;
58          rb = GetComponent<Rigidbody> ();
59
60     parametro = GameObject.Find ("DataCollector").GetComponent<
          Parametro> ();
61   }
62
63   // Update is called once per frame
64   void Update ()
65   {
66
67          //Camera
68          if(Input.GetKeyDown(KeyCode.C))
69          {
70              cam1.enabled = !cam1.enabled;
71              cam2.enabled = !cam2.enabled;
72          }
73
74          //----------MODIFICACOES------------//
75          if (count == false)
76          {
77              startTime = Time.time;
78          }
79          //----------MODIFICACOES------------//
80
81          if (!GameManager.instance.playersTurn) return;
82
83     helice.transform.Rotate(0.0f, -flight_speed * 1000 * Time.
          deltaTime, 0.0f);
84
85     Vector3 rotation;
86          if (OpenPose.OPWrapper.state == OPState.Running)
87          {
88              Debug.Log("Funcionando!!");
89              //if (Interface.instance.yToggle)
90              //{
91
92              //    float y = center.transform.position.z - head.
                    transform.position.z; //çdiferena de profundidade
93              //    float movmenty = y * multiplicadorY;
```

```
94
95              //    if (movmenty > turn_speed)
96              //    {
97              //        moveVertical = -(turn_speed);
98              //    }
99              //    else if (movmenty < turn_speed * -1)
100             //    {
101             //        moveVertical = -(turn_speed * -1);
102             //    }
103             //    else
104             //    {
105             //        moveVertical = movmenty;
106             //    }
107             //    moveVertical = -movmenty;
108             //}
109             //else
110             //{
111                 x = head.transform.position.x - center.transform.
                        position.x;
112                 //Debug.Log("X = " + x);
113                 float movmentx = x * multiplicadorX;
114                 if (movmentx > turn_speed)
115                 {
116                     moveHorizontal = turn_speed;
117                 }
118                 else if (movmentx < turn_speed * -1)
119                 {
120                     moveHorizontal = turn_speed * -1;
121                 }
122                 else
123                 {
124                     moveHorizontal = movmentx;
125                 }
126             //}
127         }
128         else {
129             //Debug.Log("test");
130       moveHorizontal = turn_speed * Input.GetAxis ("Horizontal");
131       //moveVertical = -turn_speed * Input.GetAxis ("Vertical");
132     }
133     float rotateVertical = -2 * moveVertical;
134
```

```
135     //if (Interface.instance.yToggle) {
136   //          rotation = new Vector3 (5 * rotateVertical,
        moveVertical, 1.0f);
137     //  moveHorizontal = 0f;
138   //          //Debug.Log(rotation + "-" + moveVertical);
139     //}
140     //else {
141       rotation = new Vector3 (1.0f, moveHorizontal, 5 *
          moveHorizontal);
142       moveVertical = 0f;
143             //Debug.Log(rotation + "-" + moveHorizontal);
144
145     //}
146
147     Vector3 movement = new Vector3 (moveHorizontal, moveVertical,
        0.0f) * Time.deltaTime;
148
149     transform.Translate (flight_speed * Vector3.forward * Time.
        deltaTime, Space.World);
150     transform.Translate(movement, Space.World);
151
152     transform.localEulerAngles = rotation;                    //
        çAplicaao da çrotaao sobre o modelo
153
154         //Debug.Log(this.gameObject.transform.position.z - pos);
155         //Debug.Log(GameManager.instance.erros);
156
157         //-------------çõModificaes finais--------------//
158         //xPlayerPosition = player.transform.position.x;
159         //print(xPlayerPosition);
160
161         //if(xPlayerPosition > -2.5f && xPlayerPosition < 2.5f)
162         //{
163         //    print(xPlayerPosition + "Faixa 2");
164         //}
165         //else if (xPlayerPosition > -7.5 && xPlayerPosition < -2.5
              f)
166         //{
167         //    print(xPlayerPosition + "Faixa 1");
168         //}
169         //else if (xPlayerPosition > 2.5f && xPlayerPosition < 7.5f
              )
```

```
170         //{
171         //    print(xPlayerPosition + "Faixa 3");
172         //}
173
174         //-------------çõModificaes finais--------------//
175     }
176
177     void OnTriggerEnter (Collider other)
178   {
179     if(other.gameObject.CompareTag("Argola"))
180     {
181       GameManager.instance.argolas++;
182             GameManager.instance.imagemAcerto.enabled = true;
183             StartCoroutine("contagemImagem");
184       if (GameManager.instance.playerHealth < 100) {
185         GameManager.instance.playerHealth += 20;
186       }
187       Destroy (other.gameObject);
188             Destroy(GameObject.FindWithTag("Parede"));
189             parametro.ponto = 1;
190     }
191
192         if(other.gameObject.CompareTag("Parede"))
193         {
194             GameManager.instance.erros += 1;
195             GameManager.instance.imagemErro.enabled = true;
196             Destroy(other.gameObject);
197             StartCoroutine("contagemImagem");
198         }
199   }
200
201   void OnCollisionEnter(Collision collision)
202   {
203     if (collision.gameObject.CompareTag ("Wall"))
204     {
205       rb.freezeRotation = true;
206     }
207   }
208
209     //-------------MODIFICACOES------------//
210     void OnTriggerStay(Collider other)
211     {
```

```
212         if (other.gameObject.CompareTag("Wall") && (moveHorizontal
               == 3 || moveHorizontal == -3 || moveVertical == 3 ||
               moveVertical == -3))
213         {
214             //Starts counting when player hits any wall
215             count = true;
216             float t = Time.time - startTime;
217 
218             //string minutes = ((int)t / 60).ToString();
219             string seconds = (t % 60).ToString("f0");
220             //Debug.Log("Teste");
221             //StartCoroutine("CountDown");
222             //timerTextWall.text = "Volte à sua çãposio normal\n" +
                    seconds;
223             //sets time limit to be touching the wall
224             if (t % 60 >= 5)
225             {
226                 timerTextWall.text = "Volte␣à␣sua␣çãposio␣normal!";
227             }
228         }
229         else if (other.gameObject.CompareTag("Wall") &&
               moveHorizontal == 0)
230         {
231             count = false;
232             timerTextWall.text = "";
233         }
234     }
235 
236     //resets timer to zero when leaving the wall
237     void OnTriggerExit(Collider other)
238     {
239 
240         if (other.gameObject.CompareTag("Wall"))
241         {
242             count = false;
243             timerTextWall.text = "";
244         }
245     }
246 
247     IEnumerator contagemImagem()
248     {
249         yield return new WaitForSeconds(1);
```

```
250         GameManager.instance.imagemAcerto.enabled = false;
251         GameManager.instance.imagemErro.enabled = false;
252     }
253 }
```

Ring.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4
5  public class Ring : MonoBehaviour {
6
7    private GameObject player;
8    private bool missed = false;
9    public Interface interfc;
10   public float vision;
11   Parametro parametro;
12   Save save;
13
14   void Start ()
15   {
16     player = GameObject.Find ("Player");
17     interfc = GameObject.Find("Preferencias").GetComponent<
         Interface>();
18          gameObject.GetComponent<Renderer>().enabled = false;
19     vision = interfc.distz * interfc.visArg;
20     parametro = GameObject.Find ("DataCollector").GetComponent<
         Parametro> ();
21     InvokeRepeating ("updateData", 0.1f, 0.1f);
22
23   }
24   // Update is called once per frame
25   void Update ()
26   {
27     if (this.gameObject.transform.position.z - player.transform.
         position.z < vision) {
28       gameObject.GetComponent<Renderer> ().enabled = true;
29     }
30
31     CheckMissedRing ();
```

```
32      if (this.gameObject.transform.position.z < Camera.main.
           transform.position.z - 10) {
33        //save.salvar (parametro.jogador, parametro.parametros,
             parametro.url);
34        //parametro.parametros = new List<Dado> ();
35        Destroy (this.gameObject);
36      }
37
38    }
39    void CheckMissedRing ()
40    {
41      if (this.transform.position.z - player.transform.position.z < 2
           && !missed) {
42
43        GameManager.instance.playerHealth -= 20;
44        missed = true;
45      }
46          //Debug.Log(this.transform.position.z - player.transform.
               position.z + "   " + missed);
47      }
48    void updateData(){
49      if (this.gameObject.transform.position.z - player.transform.
           position.z <= interfc.distz && this.gameObject.transform.
           position.z - player.transform.position.z > 0) {
50        parametro.distX = gameObject.transform.position.x - player.
             transform.position.x;
51        parametro.distZ = gameObject.transform.position.z - player.
             transform.position.z;
52        parametro.ponto = 0;
53      } else {
54        if (gameObject.transform.position.z < Camera.main.transform.
             position.z+(player.transform.position.z - Camera.main.
             transform.position.z)) {
55          parametro.ponto = -1;
56          CancelInvoke ();
57        }
58      }
59    }
60  }
```

RingSpawner.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class RingSpawner : MonoBehaviour
5  {
6
7      public GameObject player;
8      public GameObject argola;
9      //public GameObject lastArgola;
10     public float zPosition;
11     public float xPosition;
12     public float xPositionRing;
13
14     public float spawnTime = 3f;
15     public float xy_offset = 5;
16     public float z_offset = 50;
17     public Interface interfc;
18     public float playerXPosition;
19     public float[,] matriz = new float[5, 5];
20     public float[,] matriz1 = new float[5, 5];
21     public float[,] matriz2 = new float[5, 5];
22     public float[,] matriz3 = new float[5, 5];
23     public float[,] matriz4 = new float[5, 5];
24     public float[,] matriz5 = new float[5, 5];
25     public float somaPeso;
26     public float escolhido;
27     public float posicaoEscolhida = -1;
28     public int c;
29
30     void Start()
31     {
32         interfc = GameObject.Find("Preferencias").GetComponent<
               Interface>();
33         player = GameObject.Find("Player");
34         z_offset = interfc.distz;
35         xy_offset = interfc.distx;
36         InvokeRepeating("SpawnRing", spawnTime, spawnTime);
37         //lastArgola.transform.position = player.transform.position
               ;
38         zPosition = player.transform.position.z;
```

```
39
40          matriz1[0, 0] = 5;
41          matriz1[1, 0] = 4;
42          matriz1[2, 0] = 3;
43          matriz1[3, 0] = 2;
44          matriz1[4, 0] = 1;
45          matriz1[0, 1] = 4;
46          matriz1[1, 1] = 5;
47          matriz1[2, 1] = 4;
48          matriz1[3, 1] = 3;
49          matriz1[4, 1] = 2;
50          matriz1[0, 2] = 3;
51          matriz1[1, 2] = 4;
52          matriz1[2, 2] = 5;
53          matriz1[3, 2] = 4;
54          matriz1[4, 2] = 3;
55          matriz1[0, 3] = 2;
56          matriz1[1, 3] = 3;
57          matriz1[2, 3] = 4;
58          matriz1[3, 3] = 5;
59          matriz1[4, 3] = 4;
60          matriz1[0, 4] = 1;
61          matriz1[1, 4] = 2;
62          matriz1[2, 4] = 3;
63          matriz1[3, 4] = 4;
64          matriz1[4, 4] = 5;
65
66          matriz2[0, 0] = 4;
67          matriz2[1, 0] = 5;
68          matriz2[2, 0] = 4;
69          matriz2[3, 0] = 3;
70          matriz2[4, 0] = 2;
71          matriz2[0, 1] = 5;
72          matriz2[1, 1] = 4;
73          matriz2[2, 1] = 5;
74          matriz2[3, 1] = 3;
75          matriz2[4, 1] = 2;
76          matriz2[0, 2] = 3;
77          matriz2[1, 2] = 5;
78          matriz2[2, 2] = 1;
79          matriz2[3, 2] = 4;
80          matriz2[4, 2] = 3;
```

```
81          matriz2[0, 3] = 2;
82          matriz2[1, 3] = 3;
83          matriz2[2, 3] = 5;
84          matriz2[3, 3] = 4;
85          matriz2[4, 3] = 5;
86          matriz2[0, 4] = 1;
87          matriz2[1, 4] = 2;
88          matriz2[2, 4] = 3;
89          matriz2[3, 4] = 5;
90          matriz2[4, 4] = 4;
91
92          matriz3[0, 0] = 3;
93          matriz3[1, 0] = 4;
94          matriz3[2, 0] = 5;
95          matriz3[3, 0] = 2;
96          matriz3[4, 0] = 1;
97          matriz3[0, 1] = 4;
98          matriz3[1, 1] = 3;
99          matriz3[2, 1] = 4;
100         matriz3[3, 1] = 5;
101         matriz3[4, 1] = 4;
102         matriz3[0, 2] = 5;
103         matriz3[1, 2] = 4;
104         matriz3[2, 2] = 3;
105         matriz3[3, 2] = 4;
106         matriz3[4, 2] = 5;
107         matriz3[0, 3] = 3;
108         matriz3[1, 3] = 5;
109         matriz3[2, 3] = 4;
110         matriz3[3, 3] = 3;
111         matriz3[4, 3] = 4;
112         matriz3[0, 4] = 1;
113         matriz3[1, 4] = 2;
114         matriz3[2, 4] = 5;
115         matriz3[3, 4] = 4;
116         matriz3[4, 4] = 3;
117
118         matriz4[0, 0] = 2;
119         matriz4[1, 0] = 3;
120         matriz4[2, 0] = 4;
121         matriz4[3, 0] = 5;
122         matriz4[4, 0] = 4;
```

```
123         matriz4[0, 1] = 3;
124         matriz4[1, 1] = 2;
125         matriz4[2, 1] = 3;
126         matriz4[3, 1] = 4;
127         matriz4[4, 1] = 5;
128         matriz4[0, 2] = 4;
129         matriz4[1, 2] = 3;
130         matriz4[2, 2] = 2;
131         matriz4[3, 2] = 3;
132         matriz4[4, 2] = 4;
133         matriz4[0, 3] = 5;
134         matriz4[1, 3] = 4;
135         matriz4[2, 3] = 3;
136         matriz4[3, 3] = 2;
137         matriz4[4, 3] = 3;
138         matriz4[0, 4] = 4;
139         matriz4[1, 4] = 5;
140         matriz4[2, 4] = 4;
141         matriz4[3, 4] = 3;
142         matriz4[4, 4] = 2;
143
144         matriz5[0, 0] = 1;
145         matriz5[1, 0] = 2;
146         matriz5[2, 0] = 3;
147         matriz5[3, 0] = 4;
148         matriz5[4, 0] = 5;
149         matriz5[0, 1] = 2;
150         matriz5[1, 1] = 1;
151         matriz5[2, 1] = 2;
152         matriz5[3, 1] = 3;
153         matriz5[4, 1] = 4;
154         matriz5[0, 2] = 3;
155         matriz5[1, 2] = 2;
156         matriz5[2, 2] = 1;
157         matriz5[3, 2] = 2;
158         matriz5[4, 2] = 3;
159         matriz5[0, 3] = 4;
160         matriz5[1, 3] = 3;
161         matriz5[2, 3] = 2;
162         matriz5[3, 3] = 1;
163         matriz5[4, 3] = 2;
164         matriz5[0, 4] = 5;
```

```
165         matriz5[1, 4] = 4;
166         matriz5[2, 4] = 3;
167         matriz5[3, 4] = 2;
168         matriz5[4, 4] = 1;
169     }
170 
171     void SpawnRing()
172     {
173         if (GameManager.instance.spawnRings)
174         {
175             //    int pos = Random.Range (1, 4);
176 
177             //    if (GameManager.instance.y_axis) {
178             //        if (pos == 1)
179             //            InstantiateAtTop (z_offset);
180             //        else if (pos == 2)
181             //            InstantiateAtCenter (z_offset);
182             //        else
183             //            InstantiateAtBottom (z_offset);
184             //    } else {
185             //        if (pos == 1)
186             //            InstantiateAtLeft (z_offset);
187             //        else if (pos == 2)
188             //            InstantiateAtCenter (z_offset);
189             //        else
190             //            InstantiateAtRight (z_offset);
191             //    }
192 
193 
194             //     for (int i = 0; i < 3; i++) {
195             //    for (int j = 0; j < 3; j++) {
196             //        //print(matriz[i, j]);
197             //    }
198             //}
199 
200             if (Interface.dif == 1)
201             {
202                 print("Muito facil");
203                 matriz = matriz1;
204             }
205             else if (Interface.dif == 2)
206             {
```

```
207                 print("Facil");
208                 matriz = matriz2;
209             }
210             else if (Interface.dif == 3)
211             {
212                 print("Medio");
213                 matriz = matriz3;
214             }
215             else if (Interface.dif == 4)
216             {
217                 print("Dificil");
218                 matriz = matriz4;
219             }
220             else if (Interface.dif == 5)
221             {
222                 print("Muito␣dificil");
223                 matriz = matriz5;
224             }
225             somaPeso = 0;
226             posicaoEscolhida = -1;
227             playerXPosition = player.transform.position.x;
228 
229             print(playerXPosition);
230 
231             if (playerXPosition > -3 && playerXPosition < 3)
232             {
233                 c = 0;
234                 print("faixa3");
235 
236                 for (int i = 0; i < 5; i++)
237                 {
238                     somaPeso += matriz[i, 2];
239                 }
240 
241                 escolhido = Random.Range(0, somaPeso);
242                 print("escolhido:␣" + escolhido);
243 
244                 while (escolhido > 0)
245                 {
246                     escolhido -= matriz[c, 2];
247                     c++;
248                     posicaoEscolhida++;
```

```
249                 }
250
251                 if (posicaoEscolhida == 0)
252                     InstantiateAtMostLeft(z_offset);
253                 else if (posicaoEscolhida == 1)
254                     InstantiateAtLeft(z_offset);
255                 else if (posicaoEscolhida == 2)
256                     InstantiateAtCenter(z_offset);
257                 else if (posicaoEscolhida == 3)
258                     InstantiateAtRight(z_offset);
259                 else if (posicaoEscolhida == 4)
260                     InstantiateAtMostRight(z_offset);
261
262                 print(escolhido + "----" + posicaoEscolhida);
263
264             }
265             else if (playerXPosition <= -3 && playerXPosition >=
                    -8)
266             {
267                 c = 0;
268                 print("faixa2");
269
270                 for (int i = 0; i < 5; i++)
271                 {
272                     somaPeso += matriz[i, 1];
273                 }
274
275                 escolhido = Random.Range(0, somaPeso);
276                 print("escolhido:␣" + escolhido);
277
278                 while (escolhido > 0)
279                 {
280                     escolhido -= matriz[c, 1];
281                     c++;
282                     posicaoEscolhida++;
283                 }
284
285                 if (posicaoEscolhida == 0)
286                     InstantiateAtMostLeft(z_offset);
287                 else if (posicaoEscolhida == 1)
288                     InstantiateAtLeft(z_offset);
289                 else if (posicaoEscolhida == 2)
```

```
290                     InstantiateAtCenter(z_offset);
291                 else if (posicaoEscolhida == 3)
292                     InstantiateAtRight(z_offset);
293                 else if (posicaoEscolhida == 4)
294                     InstantiateAtMostRight(z_offset);
295                 print(escolhido + "----" + posicaoEscolhida);
296             }
297             else if (playerXPosition < -8)
298             {
299                 c = 0;
300                 print("faixa0");
301 
302                 for (int i = 0; i < 5; i++)
303                 {
304                     somaPeso += matriz[i, 0];
305                 }
306 
307                 escolhido = Random.Range(0, somaPeso);
308                 print("escolhido:␣" + escolhido);
309 
310                 while (escolhido > 0)
311                 {
312                     escolhido -= matriz[c, 0];
313                     c++;
314                     posicaoEscolhida++;
315                 }
316                 if (posicaoEscolhida == 0)
317                     InstantiateAtMostLeft(z_offset);
318                 else if (posicaoEscolhida == 1)
319                     InstantiateAtLeft(z_offset);
320                 else if (posicaoEscolhida == 2)
321                     InstantiateAtCenter(z_offset);
322                 else if (posicaoEscolhida == 3)
323                     InstantiateAtRight(z_offset);
324                 else if (posicaoEscolhida == 4)
325                     InstantiateAtMostRight(z_offset);
326                 print(escolhido + "----" + posicaoEscolhida);
327 
328             }
329             else if (playerXPosition > 8)
330             {
331                 c = 0;
```

```
332                 print("faixa5");
333
334                 for (int i = 0; i < 5; i++)
335                 {
336                     somaPeso += matriz[i, 4];
337                 }
338
339                 escolhido = Random.Range(0, somaPeso);
340                 print("escolhido:␣" + escolhido);
341
342                 while (escolhido > 0)
343                 {
344                     escolhido -= matriz[c, 4];
345                     c++;
346                     posicaoEscolhida++;
347                 }
348                 if (posicaoEscolhida == 0)
349                     InstantiateAtMostLeft(z_offset);
350                 else if (posicaoEscolhida == 1)
351                     InstantiateAtLeft(z_offset);
352                 else if (posicaoEscolhida == 2)
353                     InstantiateAtCenter(z_offset);
354                 else if (posicaoEscolhida == 3)
355                     InstantiateAtRight(z_offset);
356                 else if (posicaoEscolhida == 4)
357                     InstantiateAtMostRight(z_offset);
358
359                 print(escolhido + "----" + posicaoEscolhida);
360             }
361             else if (playerXPosition >= 3 && playerXPosition <= 8)
362             {
363                 c = 0;
364                 print("faixa4");
365
366                 for (int i = 0; i < 5; i++)
367                 {
368                     somaPeso += matriz[i, 3];
369                 }
370
371                 escolhido = Random.Range(0, somaPeso);
372                 print("escolhido:␣" + escolhido);
373
```

```
374                 while (escolhido > 0)
375                 {
376                     escolhido -= matriz[c, 3];
377                     c++;
378                     posicaoEscolhida++;
379                 }
380                 if (posicaoEscolhida == 0)
381                     InstantiateAtMostLeft(z_offset);
382                 else if (posicaoEscolhida == 1)
383                     InstantiateAtLeft(z_offset);
384                 else if (posicaoEscolhida == 2)
385                     InstantiateAtCenter(z_offset);
386                 else if (posicaoEscolhida == 3)
387                     InstantiateAtRight(z_offset);
388                 else if (posicaoEscolhida == 4)
389                     InstantiateAtMostRight(z_offset);
390                 print(escolhido + "----" + posicaoEscolhida);
391             }
392         }
393     }
394
395     void Update()
396     {
397
398
399         //---------ROLETA----------//
400
401     }
402
403     // Create ring at center
404     void InstantiateAtCenter(float offset)
405     {
406         //Instantiate(argola, new Vector3 (0.0f, 30.5f, lastArgola.
                transform.position.z + offset), Quaternion.AngleAxis
                (270, Vector3.left));
407
408         Instantiate(argola, new Vector3(0.0f, 30.5f, zPosition +
                offset), Quaternion.AngleAxis(270, Vector3.left));
409         zPosition = zPosition + offset;
410     }
411
412     // Create ring at top
```

```
413     void InstantiateAtTop(float offset)
414     {
415         //Instantiate (argola, new Vector3 (0.0f, 30.5f + xy_offset
                , lastArgola.transform.position.z + offset), Quaternion.
                AngleAxis (270, Vector3.left));
416
417         Instantiate(argola, new Vector3(0.0f, 30.5f + xy_offset,
                zPosition + offset), Quaternion.AngleAxis(270, Vector3.
                left));
418         zPosition = zPosition + offset;
419     }
420
421     // Create ring at bottom
422     void InstantiateAtBottom(float offset)
423     {
424         //Instantiate (argola, new Vector3 (0.0f, 30.5f - xy_offset
                , lastArgola.transform.position.z + offset), Quaternion.
                AngleAxis (270, Vector3.left));
425
426         Instantiate(argola, new Vector3(0.0f, 30.5f - xy_offset,
                zPosition + offset), Quaternion.AngleAxis(270, Vector3.
                left));
427         zPosition = zPosition + offset;
428     }
429
430     // Create ring at left
431     void InstantiateAtLeft(float offset)
432     {
433         //Instantiate (argola, new Vector3 (-xy_offset, 30.5f,
                lastArgola.transform.position.z + offset), Quaternion.
                AngleAxis (270, Vector3.left));
434
435         Instantiate(argola, new Vector3(-xy_offset, 30.5f,
                zPosition + offset), Quaternion.AngleAxis(270, Vector3.
                left));
436         zPosition = zPosition + offset;
437     }
438
439     void InstantiateAtMostLeft(float offset)
440     {
441         //Instantiate (argola, new Vector3 (-xy_offset, 30.5f,
                lastArgola.transform.position.z + offset), Quaternion.
```

```
            AngleAxis (270, Vector3.left));
442
443         Instantiate(argola, new Vector3(-xy_offset * 2, 30.5f,
                zPosition + offset), Quaternion.AngleAxis(270, Vector3.
                left));
444         zPosition = zPosition + offset;
445     }
446
447     void InstantiateAtMostRight(float offset)
448     {
449         //Instantiate (argola, new Vector3 (-xy_offset, 30.5f,
                lastArgola.transform.position.z + offset), Quaternion.
                AngleAxis (270, Vector3.left));
450
451         Instantiate(argola, new Vector3(xy_offset * 2, 30.5f,
                zPosition + offset), Quaternion.AngleAxis(270, Vector3.
                left));
452         zPosition = zPosition + offset;
453     }
454     // Create ring at right
455     void InstantiateAtRight(float offset)
456     {
457         //Instantiate (argola, new Vector3 (xy_offset, 30.5f,
                lastArgola.transform.position.z + offset), Quaternion.
                AngleAxis (270, Vector3.left));
458
459         Instantiate(argola, new Vector3(xy_offset, 30.5f, zPosition
                + offset), Quaternion.AngleAxis(270, Vector3.left));
460         zPosition = zPosition + offset;
461     }
462 }
```

WallSlider.cs

```
1  using UnityEngine;
2  using UnityEngine.UI;
3  using System.Collections;
4
5  public class WallSlider : MonoBehaviour
6  {
7      public GameObject player;
```

```
8
9       private float speed;
10
11      // Use this for initialization
12      void Start()
13      {
14          speed = player.gameObject.GetComponent<PlayerController>().
                flight_speed;
15          //timerTextWall.GetComponent<Renderer>().enabled = false;
16      }
17
18      // Update is called once per frame
19      void LateUpdate()
20      {
21          if (!GameManager.instance.playersTurn) return;
22          transform.Translate(speed * Vector3.forward * Time.
                deltaTime, Space.World);
23      }
24  }
```

CameraController.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class CameraController : MonoBehaviour {
5
6    public GameObject player;
7    private Vector3 offset;
8
9    // Use this for initialization
10   void Start ()
11   {
12     offset = transform.position - player.transform.position;
13   }
14
15   // Update is called once per frame
16   void LateUpdate ()
17   {
18     transform.position = player.transform.position + offset;
19   }
```

```
20 }
```

Parametro.cs

```
1  using UnityEngine;
2  using System.Collections;
3  using System.Collections.Generic;
4  using OpenPose;
5
6  public class Parametro : MonoBehaviour {
7
8      public Interface interfc;
9      public string url;
10     public List<Dado> parametros;
11     private GameObject player;
12   //public float xHead;
13   //public float yHead;
14   //public float xCenter;
15   //public float yCenter;
16   public float distX;
17   public float distZ;
18   public int ponto;
19   public float angulo = 0f;
20     public Player jogador;
21   public Save save;
22
23     //Joints' Controller.
24     public RectTransform neck, midHip;
25
26     // Output control
27     private OPDatum datum;
28
29     //IDList: {"angulo":4,"distZ":5,"distX":6,"ponto":7}
30     // Use this for initialization
31     void Start () {
32     save = new Save ();
33     interfc = GameObject.Find("Preferencias").GetComponent<
          Interface>();
34     jogador = new Player (){ id = interfc.pacienteID, interacao =
          interfc.sessaoPaciente };
35     parametros = new List<Dado> ();
```

```
36          url = interfc.url;
37          player = GameObject.Find("Player");
38          //InvokeRepeating ("updateData", 0.1f, 0.1f);
39          StartCoroutine("starGetData");
40      }
41
42  // Update is called once per frame
43  void Update () {
44      if (GameManager.instance.gameOver && !save.saved) {
45          StopCoroutine ("starGetData");
46          saveData ();
47          //if (!save.success) {
48          //  saveData ();
49          //}
50      }
51      if(Input.GetKeyDown(KeyCode.P)){
52          print ("Save>>>>>>");
53          saveData ();
54      }
55
56      }
57
58  //public void getAngulo(){
59          //    float produtoEscalar = (xCenter * xHead) + (50 *
                yHead);
60          //    float moduloHead = Mathf.Sqrt ((xHead * xHead) + (
                yHead * yHead));
61          //    float moduloCenter = Mathf.Sqrt ((xCenter * xCenter)
                + (50 * 50));
62          //    float cossAngulo = produtoEscalar / ((moduloHead) * (
                moduloCenter));
63          //    float angulo = Mathf.Acos (cossAngulo);
64          //Vector3 head = new Vector3(xHead - xCenter, yHead -
                yCenter);
65          //Vector3 center = new Vector3(xCenter - xCenter, 50f -
                yCenter);
66          //Vector3 test = new Vector3(0, 1);
67          //Vector3 test2 = new Vector3(0, 0);
68          //Debug.Log(Vector3.Angle(test, test2));
69          //this.angulo = Vector3.Angle(center, head);
70      // }
71
```

```
72      float getAngleSpine( Vector3 head, Vector3 center ) {
73          return (Mathf.Atan2(head.x - center.x, head.y - center.y) *
                180f) / Mathf.PI;//EmGraus
74      }
75
76      void updateData()
77      {
78          if (OpenPose.OPWrapper.state == OPState.Running)
79          {
80              if (datum.poseKeypoints != null)
81               {
82
83                  Vector3 head = new Vector3(neck.transform.position.
                        x, neck.transform.position.y);
84                  Vector3 center = new Vector3(midHip.transform.
                        position.x, midHip.transform.position.y);
85
86             //       getAngleSpine(head, center);
87              //      Debug.Log(getAngleSpine(head, center));
88                  //getAngulo();
89                  parametros.Add(new Dado() { id = 4, valor = "" +
                        angulo, timestamp = (long)(System.DateTime.
                        UtcNow.Subtract(new System.DateTime(1970, 1, 1))
                        ).TotalMilliseconds });
90                  parametros.Add(new Dado() { id = 5, valor = "" +
                        distZ, timestamp = (long)(System.DateTime.UtcNow
                        .Subtract(new System.DateTime(1970, 1, 1))).
                        TotalMilliseconds });
91                  parametros.Add(new Dado() { id = 6, valor = "" +
                        distX, timestamp = (long)(System.DateTime.UtcNow
                        .Subtract(new System.DateTime(1970, 1, 1))).
                        TotalMilliseconds });
92                  parametros.Add(new Dado() { id = 7, valor = "" +
                        ponto, timestamp = (long)(System.DateTime.UtcNow
                        .Subtract(new System.DateTime(1970, 1, 1))).
                        TotalMilliseconds });
93                  //print(ponto);
94                  //Debug.Log(angulo);
95              }
96          }
97      }
98
```

```
99    void saveData(){
100     save.salvar (jogador, parametros, url);
101   }
102
103   IEnumerator starGetData(){
104     InvokeRepeating ("updateData",1.0f,0.1f);
105     yield return null;
106   }
107 }
```

Player.cs

```
1  using UnityEngine;
2  using System.Collections;
3
4  public class Player {
5    public string id { get; set; }
6    public string interacao { get; set; }
7  }
```

Dado.cs

```
1  using UnityEngine;
2
3  public class Dado {
4
5    public long id { get; set; }
6    public string valor { get; set; }
7    public long timestamp { get; set; }
8  }
```

Save.cs

```
1  using UnityEngine;
2  using UnityEngine.Networking;
3  using System.Collections;
4  using System.Collections.Generic;
5
6  public class Save {
```

```
7   public bool saved = false;
8   //public string url = "http://localhost:8080/FrameWorkSG/
        Parametro/salvar";
9   //void Start(){
10  //  salvar(teste ());
11  //}
12  //Metodo salvar recebe como parametro uma variavel do tipo
        Dictionary<long,string>,
13  //contendo os parametros para serem enviados.
14  //a variavel Dictionary deve conter como chave, o id do parametro
        e
15  //seu valor deve ser o valor a ser enviado. exemplo no metodo de
        teste ao final desta classe.
16
17  public void salvar(Player player,List<Dado> parametros,string url
        ){
18    string subimitUrl = url + "Parametro/salvar";
19    string json = "jsonData="+this.SaveToString(player, parametros)
          ;
20    WWWForm form = new WWWForm ();
21    form.AddField ("jsonData", this.SaveToString (player,
          parametros));
22    Debug.Log(subimitUrl.ToString()+"");
23    Debug.Log ("" + json);
24    WWW www = new WWW (subimitUrl, form);
25    //WWW www = new WWW (subimitUrl+"?"+json);
26    //yield return www;
27    this.save(www);
28    saved = true;
29    //if (!string.IsNullOrEmpty (www.error)) {
30    //  print (www.error);
31    //  success = false;
32    //}else{
33    //  print ("SUCCESS");
34    //  success = true;
35    //}
36    //WWW www = new WWW (subimitUrl+"?"+json);
37    //StartCoroutine (save (www));
38  }
39
40  // Este metodo e utilizado pelo metodo salvar
41
```

```
42  public string SaveToString(Player player, List<Dado> parametros){
43    JSONObject j = new JSONObject (JSONObject.Type.OBJECT);
44    JSONObject a;
45    JSONObject arr = new JSONObject (JSONObject.Type.ARRAY);
46    j.AddField("playerId", player.id);
47        j.AddField("gameId", 2);
48    j.AddField("gameToken", "11eaf18333a6d665c82bb21ea4e68e5f");
49    //j.AddField("timestamp", (long)(System.DateTime.UtcNow.
          Subtract(new System.DateTime(1970, 1, 1))).TotalMilliseconds
          );
50    foreach(Dado dado in parametros){
51      a = new JSONObject (JSONObject.Type.OBJECT);
52      a.AddField("parametroId",dado.id);
53      a.AddField("valor",dado.valor);
54      a.AddField("timestamp", dado.timestamp);
55      arr.Add (a);
56    }
57    j.AddField ("parametros", arr);
58    string data = j.Print();
59    return data;
60  }
61
62
63  //Este metodo e utilizado pelo metodo salvar
64  IEnumerator save(WWW www) {
65    yield return www;
66
67    if(www.error == null) {
68      Debug.Log("Form upload complete!");
69      Debug.Log(www.error);
70    }
71    else {
72      Debug.Log(www.error);
73    }
74  }
75
76
77  //Metodo de teste---
78
79  //public Dictionary<long,string> teste(){
80  //  Dictionary<long,string> param = new Dictionary<long,string>()
      ;
```

```
81    //  param.Add (1,"777");
82    //  param.Add (2, "testevar01");
83    //  param.Add (3, "testevar02");
84    //  return param;
85    //}
86  }
```