UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

MAPA VIRTUAL 3D DA FURB NA PLATAFORMA ANDROID

MARCUS VINÍCIUS PITZ

BLUMENAU
2015

2015/1-23

MARCUS VINÍCIUS PITZ

MAPA VIRTUAL 3D DA FURB NA PLATAFORMA ANDROID

Trabalho de Conclusão de Curso apresentado ao curso de graduação em Ciência da Computação do Centro de Ciências Exatas e Naturais da Universidade Regional de Blumenau como requisito parcial para a obtenção do grau de Bacharel em Ciência da Computação.

Prof. Dalton Solano dos Reis, Mestre - Orientador

BLUMENAU
2015

2015/1-23

MAPA VIRTUAL 3D DA FURB NA PLATAFORMA ANDROID

Por

MARCUS VINÍCIUS PITZ

Trabalho de Conclusão de Curso aprovado para obtenção dos créditos na disciplina de Trabalho de Conclusão de Curso II pela banca examinadora formada por:

Presidente:	Prof. Dalton Solano dos Reis, Mestre – Orientador, FURB
Membro:	Prof. Aurélio Faustino Hoppe, Mestre – FURB
Membro:	Prof. Antonio Carlos Tavares, Mestre – FURB

Blumenau, 02 de julho de 2015

Dedico este trabalho à minha família, amigos e meu orientador, que sempre me motivaram para a conclusão do mesmo.

AGRADECIMENTOS

À minha família, pela motivação e incentivo. Agradeço em especial meus pais, Carlito Antônio Pitz e Jessoni Schmitt Pitz, que sempre me ensinaram a importância dos estudos.

Ao meu irmão Matheus Henrique Pitz, que me auxiliou na conclusão deste trabalho, através do compartilhamento de seu conhecimento e experiência.

Ao meu orientador, Dalton Solano dos Reis, pela orientação e por ter acreditado na conclusão deste trabalho.

O lucro do nosso estudo é que tornamo-nos melhores e mais sábios.

Michel de Montaigne

RESUMO

Este trabalho apresenta a construção de um aplicativo para a plataforma Android que apresenta um mapa virtual e tridimensional, que através de recursos de geolocalização e georreferenciamento, permite apresentar no mapa a posição do usuário representada por um *avatar* e locais com informações relevantes. É composto pelo aplicativo para a plataforma Android, o mapa tridimensional da região dos blocos R, S e T da Fundação Universidade Regional de Blumenau (FURB) e um *avatar* que representa o usuário no mapa virtual. Utilizou-se o *framework* Unity 3D para criação do aplicativo, o software Blender para modelagem do mapa e do *avatar*, o sensor GPS para obtenção das coordenadas geográficas do usuário e o sensor bússola para obtenção da orientação do usuário. Durante o desenvolvimento do trabalho são apresentados conceitos e técnicas de geolocalização e georreferenciamento, como o uso do filtro de Kalman para correção de valores de coordenadas e a fórmula de Haversine para o cálculo de distanciamento entre coordenadas. Como resultado foi desenvolvido um aplicativo que permite ao usuário se localizar no mapa tridimensional através do seu *avatar* e visualizar pontos de interesse disponíveis no mapa, ambos baseados em sua posição. Os principais pontos negativos são decorrentes de limitações do sensor GPS como a variação nos valores das coordenadas, imprecisão de acordo com ambiente e o uso de conexão com a internet para o seu funcionamento, A-GPS (*Assisted* GPS). Os principais pontos positivos são o uso de mapas tridimensionais para imersão do usuário no ambiente virtual, exibição de pontos de interesse informativos e por fim o auxílio na locomoção no ambiente da FURB desenvolvido.

Palavras-chave: Mapa tridimensional. Georreferenciamento. Mapa Android. Unity 3D. Geolocalização. Filtro de Kalman. Fórmula de Haversine.

ABSTRACT

This work presents the construction of an application for the Android platform, which features a virtual three-dimensional map, which through geolocation and geo-referencing capabilities, allows display on the map the user's position represented by an avatar and sites with relevant information. It consists of an application for the Android platform, three-dimensional map of the area of the blocks R, S and T of Foundation Regional University of Blumenau (FURB) and an avatar that represents the user in virtual map. We used the Unity 3D framework for application creation, Blender software for modeling the map and Avatar, the GPS sensor to obtain the user's geographical coordinates and compass sensor to obtain the user guidance. During the development are presents concepts and techniques of geolocation and georeferencing, such as using Kalman filter to correct coordinate values and the Haversine formula for distance between coordinates. As a result it developed an application that allows the user to locate the three-dimensional map through their avatar and view points of interest available on the map, both based on their position. The main drawbacks are due to GPS sensor limitations as the change in coordinate values, according to environment vagueness and use the connection to the internet for their operation, A-GPS (Assisted GPS). The main strengths are the use of three-dimensional maps for user immersion in the virtual environment, display informative points of interest and finally the aid in locomotion in FURB the environment developed.

Key-words: Three-dimensional map. Georeferencing. Android map. Unity 3D. Geolocation. Kalman filter. Haversine formula.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

A-GPS – *Assisted Global Positioning System*

ECMA – *European Computer Manufacturers Association*

FPS – *Frames Per Second*

GPS – *Global Positioning System*

IDE – *Integrated Development Environment*

RF – Requisito Funcional

RNF – Requisito Não Funcional

UML – *Unified Modeling Language*

UTM – *Universal Transverse Mercator*

XML – *Extensible Markup Language*

SUMÁRIO

1 INTRODUÇÃO

Mapas sempre foram utilizados pelos seres humanos para se localizarem em algum espaço, ou definir caminhos de como chegar a um determinado destino. Com o avanço da tecnologia, as formas de criar e conceber um mapa vêm se inovando cada vez mais. Atualmente novos mapas, que antes eram impossíveis de serem concebidos em papel, com recursos como animação, interatividade, hipertextualidade, multimídia, agora são uma realidade (RAMOS, 2003, p. 15). Um bom exemplo deste avanço é o software *Google Maps* e suas extensões, que permite utilizar mapas interativos. Com ele você tem diversas informações uteis como, melhores maneiras de chegar a um destino, locais de referência, locais favoritos, e possibilidade de utilização do *Global Positioning System* (GPS) para navegação, permitindo geolocalizar o usuário no mapa. E todos esses recursos são disponibilizados em uma plataforma gratuita (GOOGLE, 2015).

Muitos mapas são utilizados em estabelecimentos comerciais para permitir o usuário se localizar. Um destes exemplos é a Universidade Regional de Blumenau (FURB), que atualmente disponibiliza um mapa de localização de seus câmpus e serviços (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014c) e um projeto em desenvolvimento, o FURB-Mobile (GRUPO DE PESQUISA EM COMPUTAÇÃO GRÁFICA, 2014), que procuram auxiliar os usuários a se locomoverem dentro da Universidade.

A FURB possui também uma grande integração com a comunidade da região, através de programas contínuos e projetos de extensão. Anualmente, a Universidade realiza 350 atividades de extensão, beneficiando mais de 6 mil estudantes e prestando 90 mil atendimentos à comunidade de Blumenau e região (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014a). Uma dessas integrações com a comunidade é o evento Interação FURB. O Interação FURB é um programa que proporciona aos estudantes de ensino médio a oportunidade de conhecerem toda a estrutura da Universidade, participando de oficinas e atividades, no intuito de auxiliá-los na escolha profissional (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014b). O espaço físico da FURB é extenso e esparso, possuindo uma estrutura separada por câmpus localizados em regiões distintas, onde cada câmpus possui seus blocos e por fim as salas (UNIVERSIDADE REGIONAL DE BLUMENAU, 2014c). Alunos, visitantes e participantes de eventos acabam encontrando dificuldades para se locomover dentro do espaço da FURB, muitas vezes necessitando de auxílio para encontrar o seu local de destino, mesmo para quem utiliza informações de mapas disponíveis no site da FURB. Para

auxiliar estas pessoas a se locomoverem, é necessário um investimento em profissionais, placas sinalizadoras e materiais institucionais (KESTRING, 2014).

Decorrente das necessidades anteriormente apontadas foi realizado o desenvolvimento de um mapa 3D dos blocos R, S e T do câmpus um da FURB, para permitir geolocalizar um usuário usando as informações dos periféricos GPS e bússola. Desta forma o usuário poderia visualizar os espaços da FURB e caminhar por eles quando não estiver presente na FURB, ou então quando estiver presente, utilizar sua localização física dentro do mapa virtual (ambos representados por um *avatar*). Também se observou a necessidade de disponibilizar informações georreferenciadas acerca da FURB, neste caso, localização de salas de aula, blocos e serviços como cantinas e refeitórios.

1.1 OBJETIVOS

O objetivo deste trabalho é criar um aplicativo para a plataforma móvel Android que permita o usuário se localizar em um mapa e obter informações de acordo com sua localização física através de pontos de interesse.

Os objetivos específicos do trabalho são:

a) construir um mapa tridimensional dos blocos R, S e T da FURB;
b) utilizar os sensores GPS e bússola para geolocalizar o usuário dentro do mapa através de um *avatar* que o representará virtualmente.

1.2 ESTRUTURA

Este trabalho está dividido em quatro capítulos. O primeiro capítulo apresenta a introdução do trabalho e seus objetivos gerais e específicos. O capítulo dois contém a fundamentação teórica, necessária para entender melhor o emprego das tecnologias e técnicas utilizadas neste trabalho. O capítulo três apresenta as etapas de desenvolvimento do trabalho, apresentando a especificação através de diagrama de caso de uso, diagramas de classe, diagrama de atividade e diagrama de arquitetura. Ainda no capítulo três são apresentadas as técnicas e ferramentas utilizadas para o desenvolvimento do trabalho, apresentados exemplos, trechos de códigos e explicações textuais. Neste capítulo também é apresentada a operacionalidade do aplicativo desenvolvido, demonstrando como utilizá-lo em nível de usuário e por fim são apresentados os resultados e discussões a respeito do desenvolvimento, e uma breve comparação entre este trabalho e os trabalhos correlatos apresentados no capítulo dois. Por último, o capítulo quatro apresenta a conclusão do trabalho e sugestões para extensão do trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

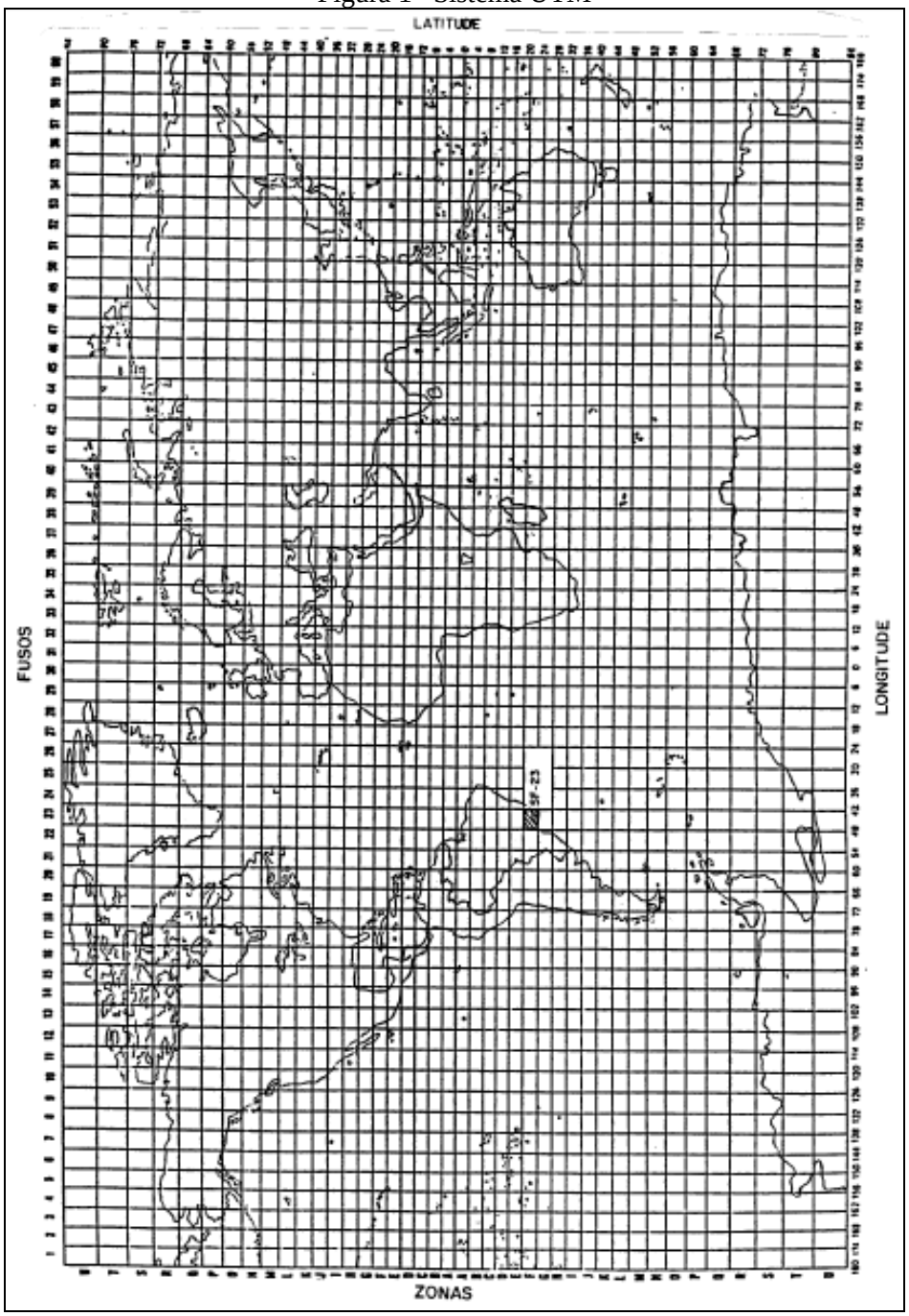
A seção 2.1 apresenta conceitos básicos de cartografia, coordenadas *Universal Transverse Mercator (*UTM), coordenadas planas, latitude e longitude, e os elementos do processo de geolocalização. A seção 2.2 apresenta o filtro de Kalman, bem como seus conceitos e aplicações. A seção 2.3 apresenta a fórmula de Haversine, seu conceito, definição e aplicação. A seção 2.4 apresenta o *framework* Unity 3D, suas características e aplicações. Por fim, a seção 2.5 apresenta os trabalhos correlatos ao trabalho desenvolvido.

2.1 GEOLOCALIZAÇÃO (CARTOGRAFIA, COORDENADAS UTM, LATITUDE E LONGITUDE)

De acordo com IBGE (1998, p. 9), a Cartografia apresenta-se como o conjunto de estudos e operações científicas, técnicas e artísticas que, tendo por base os resultados de observações diretas ou da análise de documentação, se voltam para a elaboração de mapas, cartas e outras formas de expressão ou representação de objetos, elementos, fenômenos e ambientes físicos e socioeconômicos, bem como a sua utilização. O processo cartográfico, partindo da coleta de dados, envolve estudo, análise, composição e representação de observações, de fatos, fenômenos e dados pertinentes a diversos campos científicos associados à superfície terrestre.

O UTM é um sistema de coordenadas baseado no plano cartesiano (eixos x, y) e usa o metro (m) como unidade para medir distâncias e determinar a posição de um objeto. O sistema UTM não acompanha a curvatura da Terra e por isso seus pares de coordenadas também são chamados de coordenadas planas (GEOREFERENCE, 2014). O sistema UTM é representado pela Figura 1.

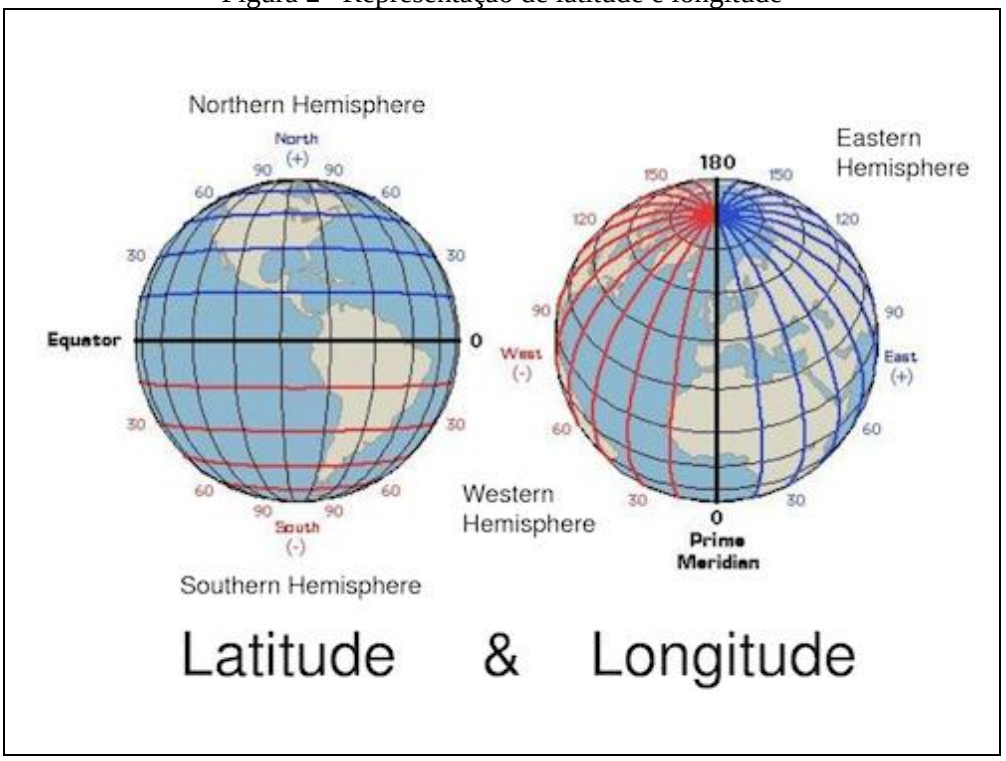
Figura 1– Sistema UTM



Fonte: Eastman (2014).

Latitude e longitude são descrições da localização, ou coordenadas geográficas, de um determinado lugar na Terra. A latitude é a distância de um ponto até a linha do Equador, medida ao longo do meridiano de Greenwich, esta distância mede-se em graus podendo variar entre 0º (no equador) e 90º para Norte ou para Sul. Por outro lado, a longitude é medida ao longo do Equador, e representa a distância entre um ponto e o Meridiano de Greenwich. Também é medida em graus, podendo ir de 0º a 180º para Leste ou para Oeste (SIGNIFICADOS, 2014). A representação de latitude e longitude pode ser observada na Figura 2.

Figura 2– Representação de latitude e longitude



Fonte: Eastman (2014).

Conforme pode ser observado na Figura 2, a latitude é à medida que representa a distância até a linha do equador paralela aos polos, podendo variar de 0 a 90 graus. Já a longitude, é a distância ao longo da linha do equador, podendo variar de 0 a 180 graus. Neste caso é como se a terra estivesse com uma grade sobrepondo a sua área, podendo assim dizer que existem duas coordenadas para se localizar (EASTMAN, 2014). Os termos em inglês citados na Figura 2, *Northern Hemisphere*, *Southern Hemisphere*, *Equator*, *South*, *West*, *Western Hemisphere*, *Eastern Hemisphere* e *Prime Meridian*, são respectivamente, hemisfério norte, hemisfério sul, linha do equador, sul, ocidente, hemisfério ocidental, hemisfério oriental e meridiano de Greenwich.

Aires e Hahn (2014, p. 6) afirmam que a geolocalização define a posição de algo, pessoa, lugar ou coisa no planeta por dois componentes, a latitude e longitude, obtidas através de equipamentos eletrônicos como o GPS. Uma vez que o ponto foi localizado, esta informação pode ser reutilizada para se obterem mais informações sobre a área ao seu redor e também informações sobre outros pontos próximos como empresas, engarrafamentos ou até mesmo outras pessoas. Segundo Oliveira (2011) uma das primeiras formas de orientação usadas pelos viajantes foi a observação de estrelas. Em função de suas posições no céu, eles podiam identificar a latitude em que estavam. A longitude podia ser determinada em função da hora em que as estrelas passavam pelo ponto mais alto do céu, chamado de zênite.

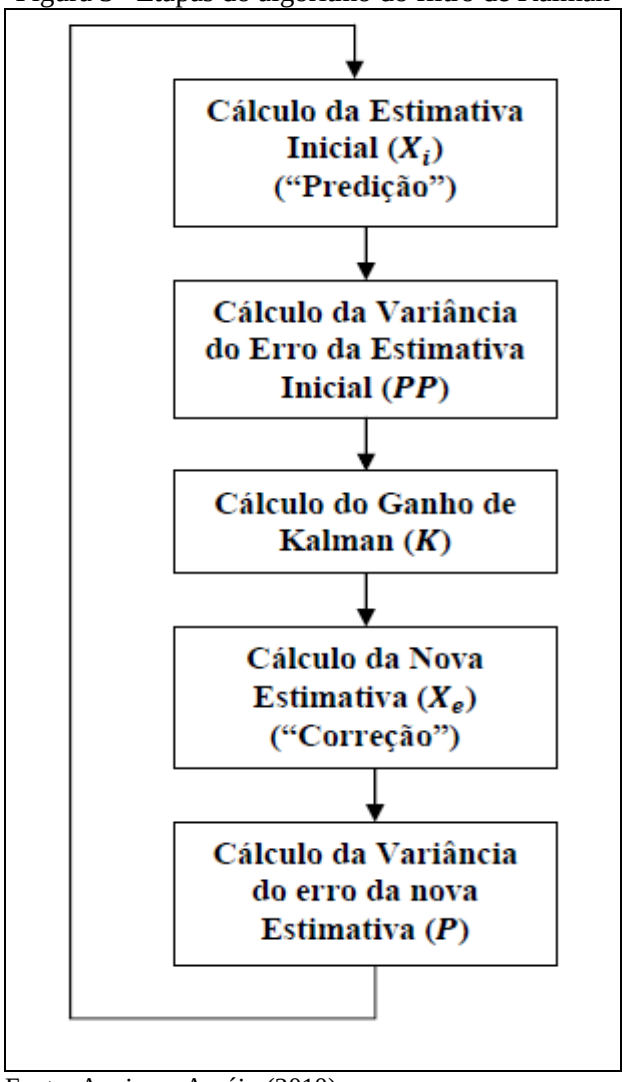
2.2 FILTRO DE KALMAN

Segundo Aquino e Araújo (2010, p. 1), o filtro de Kalman é um algoritmo recursivo simples e eficiente, capaz de estimar as variáveis de estado de sistemas lineares a partir de medidas ruidosas dos sensores que compõem o sistema. Por esta razão, o filtro de Kalman é bastante utilizado em sistemas de navegação e rastreamento de veículos. O objetivo do filtro de Kalman em um sistema dinâmico é estimar, a cada instante e de forma ótima, a saída do sistema. Para isso é necessário conhecer a variável de estado (`Xi`) e saber que a variável de saída (`Yi`) está sendo contaminada por um ruído (`Ui`) e por uma perturbação (`Wi`).

2.2.1 Etapas do filtro

A seguir serão demonstradas as etapas do algoritmo do filtro de Kalman, conforme Figura 3.

Figura 3– Etapas do algoritmo do filtro de Kalman



Fonte: Aquino e Araújo (2010).

Conforme afirmado por Aquino e Araújo (2010, p. 2-3), o algoritmo do filtro de Kalman é separado em cinco etapas conforme definido na Figura 3 e explicado abaixo.

O cálculo da Estimativa Inicial, que é a primeira etapa para a aplicação do filtro, é realizado através da multiplicação da estimativa anterior (`Xe`) pela constante do sistema linear (`a`). A primeira Estimativa Inicial, onde não se sabe qual é a estimativa anterior, é definida de forma aleatória e por isso não é possível definir a variância de seu erro. A Estimativa Inicial é calculada conforme definido na equação da Figura 4.

Figura 4– Equação da estimativa inicial

$$X_i = a \cdot X_e(0)$$

Fonte: adaptado de Aquino e Araújo (2010, p. 2).

Na etapa Cálculo da Variância do Erro da Estimativa Inicial, é calculada a Variância do Erro da Estimativa Inicial que é encontrada através da multiplicação dos valores da variância do erro da estimativa anterior (`P`), da variância da perturbação do sistema (`W`) e novamente da constante do sistema linear (`a`). Como foi dito anteriormente, não é possível definir a variância do erro para a primeira estimativa inicial e por isso o valor dessa variância é definido de forma aleatória e com valor geralmente elevado. A variância da estimativa inicial é calculada conforme equação da Figura 5.

Figura 5– Equação da variância inicial

$$PP = P \cdot a^2 \cdot W$$

Fonte: adaptado de Aquino e Araújo (2010, p. 3).

Para as etapas Cálculo do Ganho de Kalman, Cálculo da Nova Estimativa e Cálculo da Variância do Erro da Nova Estimativa será inserido o ruído na medição de `x(k)`, que é exatamente o ruído na medida dos sensores que compõe o sistema, obtendo assim um novo sistema linear com uma nova equação. A medida é `y(k)`, a constante do sistema é `M` e o ruído é `u(k)` de variância `U`. A equação do novo sistema será semelhante à apresentada pela Figura 6.

Figura 6– Equação com inserção do ruído

$$y(1) = M \cdot x(1) + u(1)$$

Fonte: adaptado de Aquino e Araújo (2010, p. 3).

Na etapa Cálculo do Ganho de Kalman é utilizada a constante do novo sistema (`M`), a variância calculada na segunda etapa (`PP`) e a variância do erro da medida do sensor (`U`) para calcular o ganho de Kalman. O ganho de Kalman será utilizado para encontrar a nova

estimativa (`Xe(1)`) e a variância do erro da nova estimativa (`P`) que são as duas últimas etapas. A Figura 7 representa a equação utilizada para calcular o ganho de Kalman.

Figura 7– Equação de ganho

$$K = \frac{M \cdot PP}{PP \cdot M^2 + U}$$

Fonte: adaptado de Aquino e Araújo (2010, p. 3).

Na etapa Cálculo da Nova Estimativa será realizado uma nova estimativa (`Xe(1)`) que será a saída do filtro, ou seja, a estimativa que se deseja encontrar. O cálculo dessa nova estimativa é feito através dos valores da estimativa inicial (`Xi`), do ganho de Kalman (`K`), da medida ruidosa (`y(1)`), que é o valor de entrada do filtro e para a aplicação serão as coordenadas geográficas (latitude e longitude), e da constante do novo sistema (`M`). A Figura 8 apresenta a equação utilizada para calcular a nova estimativa.

Figura 8– Equação de nova estimativa

$$X_e(1) = X_p + K\{y(1) - M \cdot x_p\}$$

Fonte: adaptado de Aquino e Araújo (2010, p.3).

Na última etapa Cálculo da Variância do Erro da Nova Estimativa, será calculada uma nova variância, agora para o erro da nova estimativa. Para esse cálculo será necessário o valor da variância do erro da estimativa inicial (`PP`), da variância do erro da medida do sensor (`U`), do ganho de Kalman (`K`) e do valor da constante do novo sistema (`M`). A nova variância pode ser obtida através da equação representada pela Figura 9.

Figura 9– Equação de nova variância

$$P(1) = PP \cdot (1 - K \cdot M)^2 + (U \cdot K^2)$$

Fonte: Aquino e Araújo (2010).

Ainda mencionado por Aquino e Araújo (2010, p. 1-3), o valor da variância do erro da nova estimativa será utilizado para o cálculo da próxima estimativa inicial, reiniciando assim a sequência do filtro de Kalman.

2.2.2 Valores obtidos

Ainda de acordo com Aquino e Araújo (2010, p. 4), se aplicado o filtro de Kalman em um sistema de coordenadas `X`, `Y` com erro `U` os resultados são positivos e demonstrados na Figura 10.

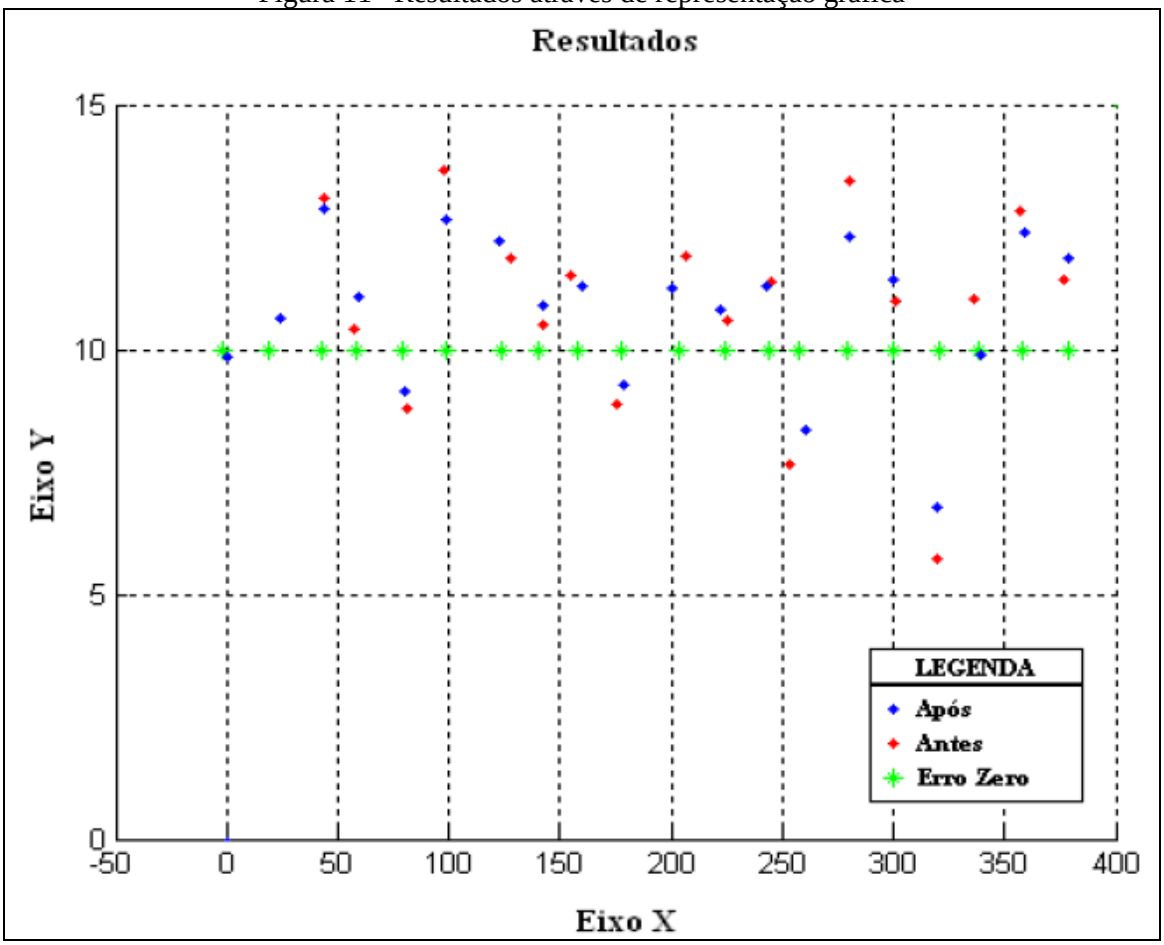
Figura 10– Resultados para um sistema de coordenadas X, Y

Id	X (Antes)	X (Fitrado)	X (Erro Zero)	Y (Antes)	Y (Fitrado)	Y (Erro Zero)
1	0,2256	0,2256	-2,194	9,84	9,84	10,0
2	23,7659	23,765	18,4605	10,6243	10,6241	10,0
3	43,5079	44,1408	42,9952	13,0707	12,8858	10,0
4	57,3707	59,8943	57,9669	10,4021	11,0757	10,0
5	80,6458	79,9924	79,0768	8,788	9,1597	10,0
6	97,7779	98,568	98,8682	13,6542	12,6656	10,0
7	127,7865	122,4234	123,6219	11,8452	12,2196	10,0
8	142,6559	142,6837	140,1809	10,516	10,8898	10,0
9	154,8378	159,8916	157,5827	11,5018	11,3053	10,0
10	175,9876	178,3693	177,4855	8,8639	9,2943	10,0
11	206,8795	200,7609	203,8811	11,9149	11,2618	10,0
12	225,2001	222,0305	224,3323	10,6004	10,8077	10,0
13	244,535	242,8034	243,6561	11,3975	11,2944	10,0
14	253,3739	260,5241	257,1916	7,6581	8,3514	10,0
15	280,032	280,3451	279,5225	13,4339	12,2813	10,0
16	301,0883	300,4454	299,5552	10,9714	11,4441	10,0
17	319,2637	320,1346	320,6876	5,7132	6,774	10,0
18	336,4783	339,3123	338,7202	11,0508	9,881	10,0
19	357,3989	358,8145	357,8139	12,8495	12,4065	10,0
20	376,644	378,2622	379,0009	11,4278	11,875	10,0

Fonte: adaptado de Aquino e Araújo (2010, p. 4).

Na Figura 11 é possível observar os valores do resultado obtido através de uma representação gráfica.

Figura 11– Resultados através de representação gráfica



Fonte: Aquino e Araújo (2010, p. 4).

Por fim, conforme Aquino e Araújo (2010, p. 4), como forma de comprovar a redução do erro após a aplicação do filtro, foi calculado o erro quadrático médio antes e após a aplicação e apresentado na Figura 12. O erro quadrático médio é representado pela soma das diferenças entre o valor estimado e o valor real dos dados, em relação ao número de termos utilizados, quanto menor o resultado deste erro menor é a variação entre o valor estimado e o valor real e maior a precisão (SEUBERT, 2015).

Figura 12– Resultados através de representação gráfica

Eixo	Antes	Depois
X	5,6	4,215
Y	4,2943	2,9644

Fonte: adaptado de Aquino e Araújo (2010, p. 4).

2.3 FÓRMULA DE HAVERSINE

Conforme Carvalho (2013, p. 39), a fórmula de Haversine é uma importante equação usada em navegação, fornecendo distâncias entre dois pontos de uma esfera a partir de suas latitudes e longitudes. É um caso especial de fórmula mais geral da trigonometria esférica, a lei dos Haversines, relacionando os lados a ângulos de uma esfera "triangular". Segundo

Ribeiro Júnior et al. (2013), essa função é bastante usada em sistemas de navegação e é capaz de fornecer a distância entre dois pontos de uma esfera utilizando coordenadas geográficas (latitude e longitude), realizando uma aproximação da terra como uma esfera perfeita e gerando um erro médio de 0,3% nos resultados do cálculo. A função Haversine é demonstrada na Figura 13.

Figura 13– Equação de Haversine

$$haversine(\theta) \equiv \sin^2\left(\frac{\theta}{2}\right)$$

Fonte: adaptado de Ribeiro Júnior et al. (2013, p. 7).

Considerando dois pontos de uma esfera de raio `R`, com latitude e longitudes (`l1`, `Δ1`), (`l2`, `Δ2`), respectivamente, a distância `d` é definida pela equação demonstrada na Figura 14.

Figura 14– Equação de distância entre duas coordenadas

$$d = 2R \arcsin\left(\sqrt{\sin^2\left(\frac{l_2 - l_1}{2}\right) + \cos(l_1)\cos(l_2)\sin^2\left(\frac{\Delta_2 - \Delta_1}{2}\right)}\right)$$

Fonte: adaptado de Ribeiro Júnior et al. (2013, p. 7).

Os valores de latitude e longitude utilizados na equação devem possuir suas medidas em ângulos radianos. A medida de distância retornada está totalmente relacionada a medida do raio utilizado. Para o cálculo de distância de coordenadas geográficas, é utilizada uma aproximação do raio médio terrestre de 6371 quilômetros.

2.4 UNITY

A Unity é um ecossistema de desenvolvimento de jogos, com mecanismo de renderização totalmente integrado com um conjunto completo de ferramentas e fluxos de trabalho, para criar conteúdos interativos em 3D e 2D de fácil publicação em multiplataforma. O Unity permite a construção em diversas plataformas como, IOS, Android, Windows Phone 8, BlackBerry 10, Mac, Linux, Web Player, PS3, PS4, PSVita, Xbox 360, Xbox One e Wii U (UNITY TECHNOLOGIES, 2014b).

2.4.1 Interface gráfica

O editor gráfico da Unity permite efetuar a construção de cenas em tempo real. O editor traz diferentes modos de exibição, ou janelas, contendo as ferramentas e fluxos de trabalho para a criação de jogos. É possível visualizar todos os objetos disponíveis em uma cena e suas respectivas propriedades, além de criar hierarquia entre os objetos (UNITY TECHNOLOGIES, 2014a). A interface gráfica é apresentada na Figura 15.

Figura 15– Interface gráfica da Unity e suas separações

Fonte: adaptado de Passos et al. (2009, p. 5).

2.4.1.1 Scene

A `Scene` é a principal janela da interface gráfica da Unity e a principal forma de manipulação de objetos nas cenas, possibilitando a orientação (rotação) e posicionamento (translação) desses objetos com um *feedback* imediato da alteração exibida em tela. As manipulações nesta janela são semelhantes a softwares como o Blender, onde é possível arrastar objetos, manipular câmera, cenários e todos os elementos que compõem a cena. É possível configurar atalhos para facilitar as manipulações na cena, tais como, *pan*, translação, rotação e escala (PASSOS et al, 2009, p. 6).

2.4.1.2 Game

A janela `Game` é responsável por uma pré-visualização da aplicação na forma como será exibida quando finalizada. Nesta janela é possível observar o comportamento da aplicação quando executada, podendo observar valores de desempenho, e outros valores relativos ao desenvolvimento, como tempo de processamento, número de quadros desenhados, memória utilizada. Opções de depuração também estão disponíveis nesta janela,

possibilitando efetuar pausas no decorrer da execução e analisar valores em *logs* e valores de objetos em cena (PASSOS et al, 2009, p. 7).

2.4.1.3 Hierarchy

A janela `Hierarchy` exibe todos os elementos da cena, suas organizações e hierarquias de composição de vários objetos, formando um grafo de cena (PASSOS et al, 2009, p. 6). Nesta janela é possível adicionar novos objetos para a cena, efetuar pesquisas por objetos já existentes, duplicar, copiar, renomear ou remover objetos.

Ainda segundo Passos et al. (2009, p. 6-7) ao selecionar um objeto na janela `Scene` este também será selecionado na janela `Hierarchy`, sendo que o contrário também ocorre, tornando assim o objeto selecionado na janela `Hierarchy` como o objeto atualmente selecionado na cena. As propriedades deste objeto selecionado estarão disponíveis na janela `Inspector`. A Figura 16 demonstra a disposição de dois objetos na janela `Hierarchy`.

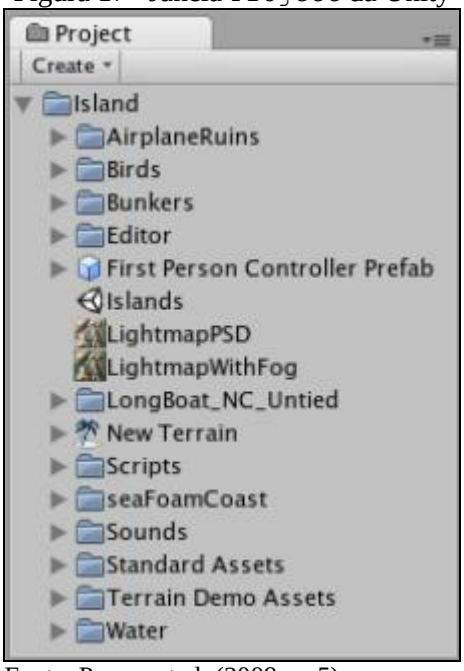
Figura 16– Janela de hierarquia da Unity

Fonte: Passos et al. (2009, p. 6).

2.4.1.4 Project

A janela `Project` é responsável pela organização e manipulação de todos os arquivos que compõem um projeto da Unity, como áudios, imagens, modelos, texturas, *scripts*, entre outros. Estes arquivos presentes dentro do projeto são chamados de *Assets* ou na tradução literal “ativos”. Através desta janela são exportados ou importados novos *Assets* para o projeto, permitindo também a criação de novas pastas, efetuar pesquisas, copiar, renomear ou remover um *Asset* (PASSOS et al, 2009, p. 5). A Figura 17 demonstra a janela `Project` e alguns *Assets* presentes.

Figura 17– Janela `Project` da Unity



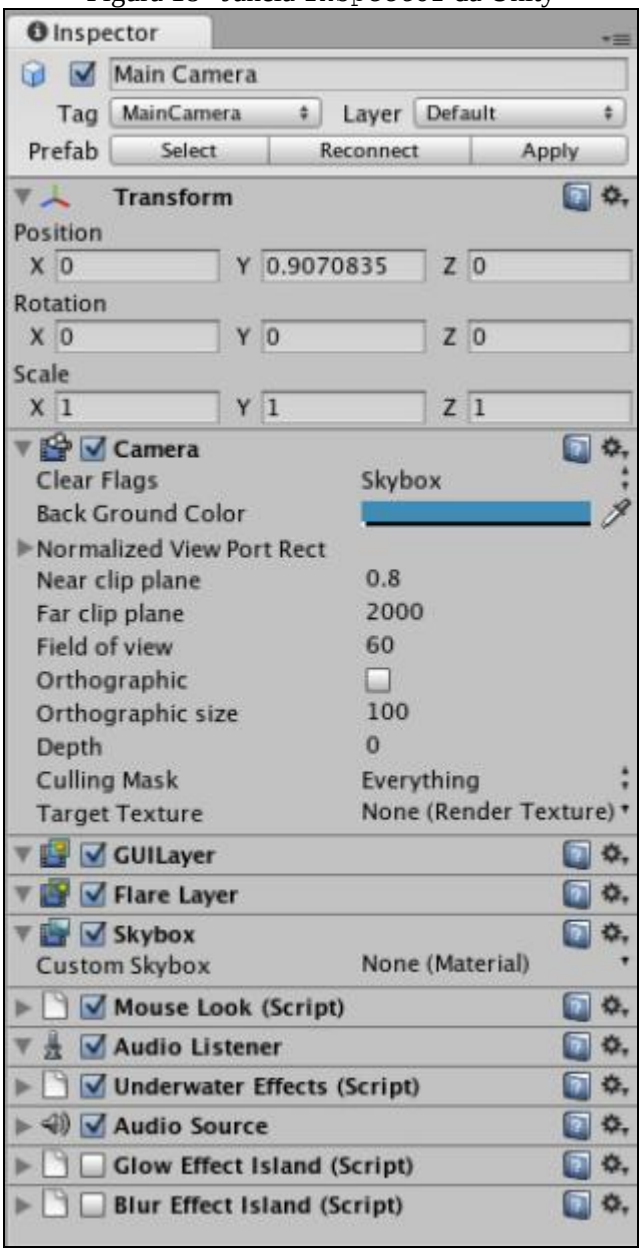
Fonte: Passos et al. (2009, p. 5).

Os arquivos presentes na janela `Project` não tem uma ligação direta com a cena atual, sendo que um *Asset* pode estar disponível em mais de uma cena, e qualquer alteração neste *Asset* afeta todas as cenas que o utilizam. Por este motivo a janela `Project` permite a localização das dependências de um *Asset*, selecionando todos os *Assets* que dependem deste e ainda possibilita verificar se este *Asset* é utilizado na cena atual.

2.4.1.5 Inspector

A janela `Inspector` permite a visualização e manipulação de componentes ligados a objetos em cena, alterando valores ou inserindo novos componentes. Nesta janela é possível alterar os atributos públicos de cada componente, estes atributos representam os parâmetros para execução do componente (PASSOS et al, 2009, p. 8). A seção 2.4.2 descreve com mais detalhes as informações manipuladas na janela `Inspector`. A Figura 18 demonstra a janela `Inspector` e a visualização de alguns componentes do objeto `Main Camera` e seus atributos.

Figura 18– Janela `Inspector` da Unity



Fonte: Passos et al. (2009, p. 8).

2.4.2 Game Object

O principal elemento da Unity é o *Game Object*. Este componente serve de base para todos os outros componentes, como caixas de colisão, componentes de física, entre outros. O *Game Object* funciona como um "repositório" de funcionalidades, onde cada componente adicionado representa uma nova funcionalidade ou comportamento. Todo objeto na cena é representado por uma instância de *Game Object* desde um simples objeto de câmera até um modelo mais complexo. Todo *Game Object* inicialmente possui a mesma característica, uma transformação na qual indica sua escala, rotação e translação em relação a cena onde está inserido (CARVALHO, 2012, p. 16).

O *Game Object* pode ser armazenado e utilizado em várias cenas diversas vezes, neste caso ele é chamado de *Prefab*. Quando um *Game Object* é adicionado em uma cena através de um *Prefab*, ele herda todos os componentes e atributos deste *Prefab*, permitindo alterar alguma propriedade ou componente deste. Toda alteração realizada no *Prefab* é aplicada aos *Game Objects* em cena que foram gerados a partir do *Prefab* alterado. Todas as manipulações dos componentes de um *Game Object* devem ser feitas através da janela `Inspector` (PASSOS et al, 2009, p. 8-10).

2.4.3 Assets

Assets são todos os arquivos presentes em um projeto da Unity, como áudio, texturas, imagens, modelos, scripts, entre outros. Eles são importados para dentro do projeto da Unity através do arrasto do mouse ou através de opções de importação (CARVALHO, 2012, p. 16). Esses diferentes tipos de artefatos são desenvolvidos em ferramentas externas, especializados na construção de cada um dos tipos de *Assets*. Após sua criação ou edição, *Assets* precisam de alguma forma serem importados para dentro de editor de cenas da Unity (PASSOS et al, 2009, p. 10).

Segundo Passos et al. (2009, p. 10) a Unity permite alterações feitas em um *Asset* em tempo real. A Unity verifica as modificações realizadas no *Asset* e atualiza automaticamente dentro do projeto. Com esta funcionalidade não é necessário realizar novamente a importação deste *Asset* quando o mesmo sofrer alguma alteração.

2.4.4 Prefabs

Prefabs são objetos armazenados que podem ser utilizados inúmeras vezes em qualquer cena do projeto (CARVALHO, 2012, p. 15). Ao se adicionar um *Prefab* a uma cena, está sendo criada uma instância do mesmo representado por um *Game Object*. Independente de quantas instâncias existam no projeto, qualquer mudança feita ao *Prefab* original será aplicada a todas as cópias existentes nas cenas (PASSOS et al, 2009, p. 26).

Ainda segundo Passos et al. (2009, p. 26) um *Prefab* é um tipo de *Asset* acessado através da janela `Project`. Para que ele exista na cena basta arrastá-lo para dentro da janela `Hierarchy`, ou ainda instanciá-lo em tempo de execução através de funções específicas dentro dos *scripts*. O contrário também é possível, arrastando um objeto da janela `Hierarchy` para a janela `Project` ele irá se transformar em um novo *Asset* do tipo *Prefab*.

2.4.5 Scripts

O sistema de *scripting* da Unity permite a construção de *scripts* em três linguagens: *C Sharp*, *JavaScript* e *Boo*. Ambos os scripts rodam em uma versão modificada da biblioteca *Mono* (CARVALHO, 2012, p. 17). Mono é uma implementação multiplataforma de código aberto do *.NET Framework* da empresa Microsoft, com base nos padrões da linguagem C Sharp (ECMA, *European Computer Manufacturers Association*) (MONO, 2014). Os *scripts* na Unity são acoplados aos objetos como sendo componentes de um *Game Object* (PASSOS et al, 2009, p. 22).

Conforme Passos et al. (2009, p. 22-24) a composição básica de todo o *script* que é utilizado na Unity como um componente de um *Game Object*, é herdar suas características a partir da classe `MonoBehaviour`, esta por sua vez possui dois métodos padrões que necessitam serem implementados. O método `Start` que é chamado durante a criação da cena e o método `Update` que é chamado a cada execução de um *frame*. Todas as variáveis de visibilidade pública existentes nos *scripts* serão atributos do componente que o representa.

2.4.6 PlayerPrefs

A estrutura *PlayerPrefs (*preferências de jogadores) permite salvar e acessar dados entre as sessões dos jogos (UNITY TECHNOLOGIES, 2014c). Ainda segundo Unity Technologies (2014c) os dados são armazenados e consultados utilizando chaves de acesso, onde cada valor possui a sua chave. São suportados números inteiros, caracteres e números com casas decimais.

2.5 TRABALHOS CORRELATOS

Esta seção tem por finalidade apresentar quatro trabalhos relacionados e suas principais características. Dentro os trabalhos, os selecionados foram o “jogo sério *3D* de apoio a aprendizagem sensível ao contexto” de Lima, Mendes Neto e Rodrigues (2012), o aplicativo “Harvard móbile” de Harvard (2014), o trabalho de conclusão de curso “sistema móvel na plataforma Android para compartilhamento de geolocalização usando mapas e notificações” de Kestring (2014) e por fim o trabalho de conclusão de curso “combinando Kinect e *smartphones* para rastreamento do movimento do pulso” de Baião e Lima (2014). Ao final desta seção será apresentado um breve comparativo entre os trabalhos correlatos apresentados.

2.5.1 Jogo 3D de apoio a aprendizagem sensível ao contexto

O jogo sério é capaz de perceber a localização do usuário e identificar em qual ambiente o usuário se encontra, para, posteriormente recomendar desafios relacionados aquela localidade ou área de estudo (LIMA; MENDES NETO; RODRIGUES, 2012). Ainda segundo Lima, Mendes Neto e Rodrigues (2012), o usuário aprenderá diversos conceitos de disciplinas, podendo interagir com *avatares* presentes no ambiente virtual, sendo que o ambiente virtual também conta com desafios que utilizam a realidade aumentada para que o jogador aprenda melhor os conceitos em sala de aula, através de aplicações do mundo real. Segundo Lima, Mendes Neto e Rodrigues (2012) com o intuito de facilitar esse processo de ensino-aprendizagem, foi criado, no cenário de uma Universidade Virtual, um jogo sensível ao contexto dos estudantes com desafios educativos em 2D e 3D para diversos dispositivos computacionais, através da plataforma web, precisando apenas de um navegador de internet para ser executado.

Dentro do ambiente do jogo o usuário se cadastra e obtém uma conta com usuário e senha, podendo optar pelas opções de perfil, amigos, Universidade virtual, *chat*, notícias e geolocalização (LIMA; MENDES NETO; RODRIGUES, 2012). O jogo está disponível para a plataforma web.

Conforme o autor (LIMA; MENDES NETO; RODRIGUES, 2012), na opção perfil o usuário visualiza os seus dados, tais como nome, sobrenome, instituição de ensino, cidade, país, área de afinidade (informática, medicina, biologia, etc.), entre outros dados. Na opção amigos, o usuário é informado sobre sua lista de amizades, além de visualizar todos os usuários cadastrados no sistema, cuja afinidade seja semelhante. Com esta lista de amigos ele pode acionar a opção *chat* e abrir uma nova conversa. Na opção Universidade virtual, o usuário pode interagir com os *avatares* presentes no ambiente ou então solucionar desafios propostos nos laboratórios espalhados pelo jogo. Na opção notícias, é possível ler as últimas notícias cadastradas pelos professores ou administradores do jogo, podendo ser novos desafios e localizações. Por fim na opção geolocalização, o usuário através de um dispositivo móvel, envia sua localização ao servidor e este determina o local onde ele está no mapa e também efetua algumas recomendações baseadas na localização do usuário.

Na Figura 19 é demonstrada uma imagem do jogo e suas opções, com a opção geolocalização selecionada e demonstrando a posição do usuário em relação ao mapa. Com base na localização do usuário o jogo propôs desafios cadastrados para aquele local, por exemplo: o usuário se encontra no laboratório de química, ao efetuar a opção *check in*, será

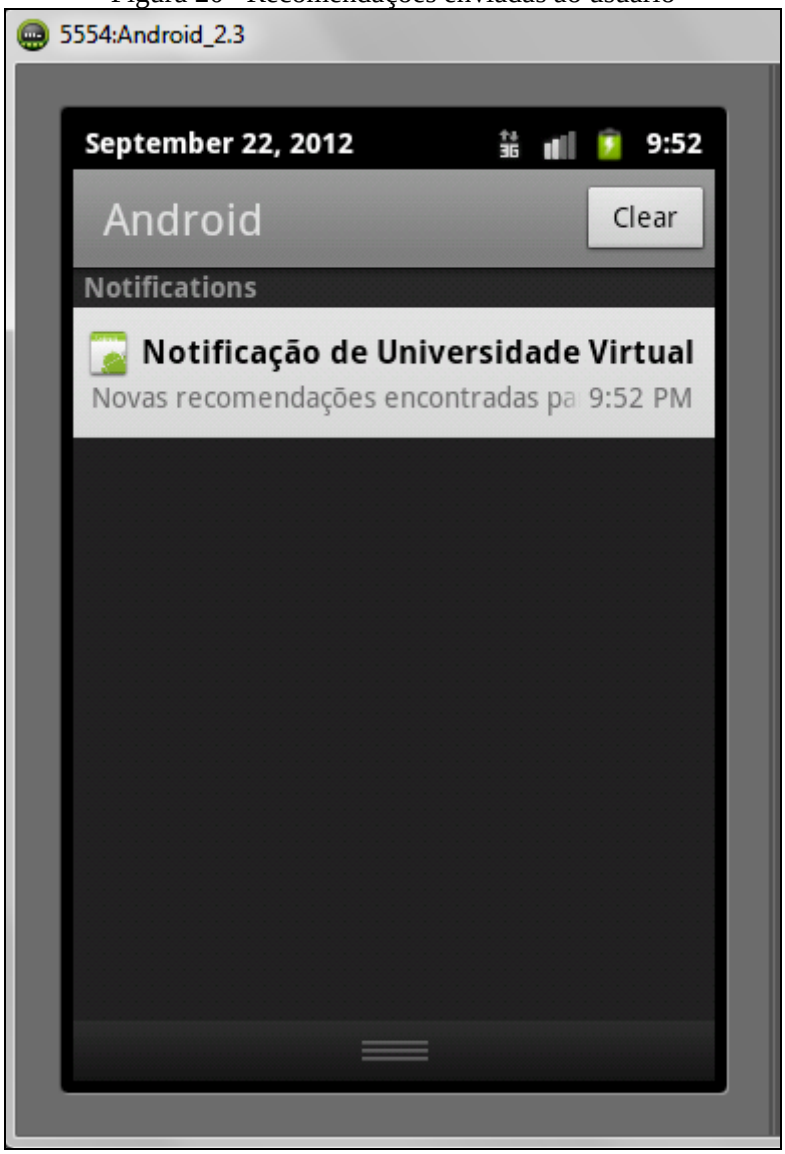
avalia a sua posição e os desafios cadastrados para o laboratório de química (LIMA; MENDES NETO; RODRIGUES, 2012).

Figura 19– Imagem do jogo

Fonte: Lima, Mendes Neto e Rodrigues (2012).

A Figura 20 demonstra o sistema de notificações do jogo. Segundo Lima, Mendes Neto e Rodrigues (2012) o mecanismo de recomendação considera as informações de localidade cadastradas, tais como nome do local, longitude e latitude, como também informações relacionadas à área de interesse do estudante e horário preferido de estudo. Caso as informações comparadas estejam de acordo com o perfil do estudante, desafios e assuntos de interesse são enviados para o mesmo, através de uma aplicação que é executada em segundo plano no dispositivo móvel. Para alertá-lo de novas recomendações, o ambiente emite um som. Algumas recomendações podem utilizar realidade aumentada para facilitar a aprendizagem, neste caso, o sistema irá exibir desafios ou conteúdos relacionados ao objeto demonstrado.

Figura 20– Recomendações enviadas ao usuário



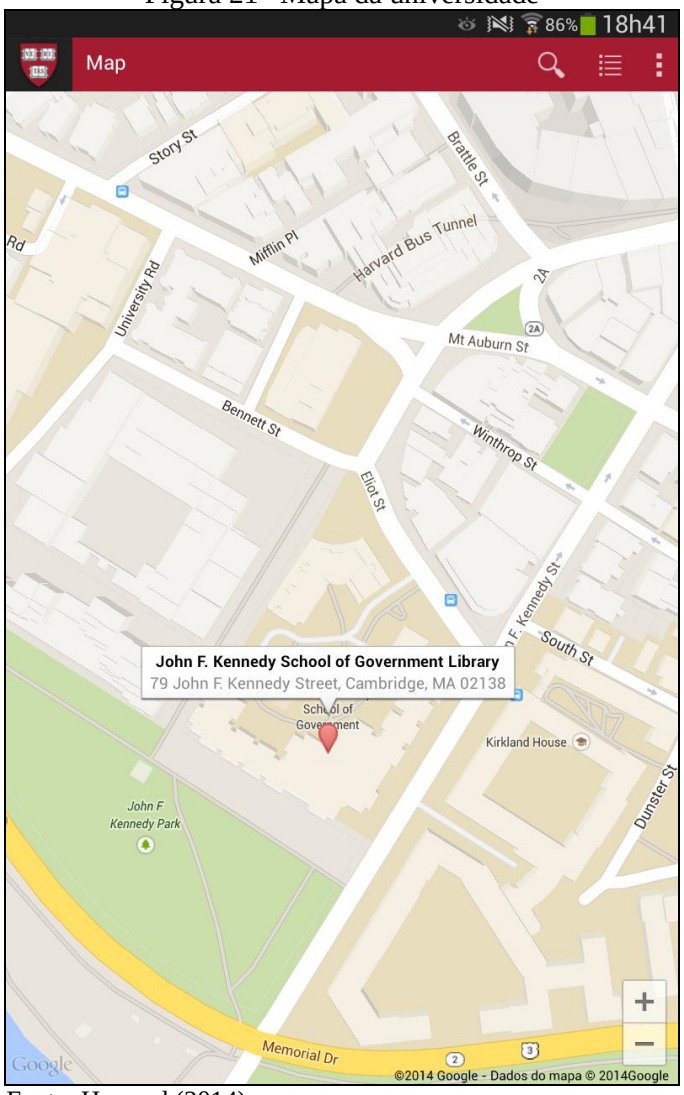
Fonte: Lima, Mendes Neto e Rodrigues (2012).

2.5.2 Harvard Mobile

Harvard Mobile é uma iniciativa para entregar um conteúdo móvel para toda a comunidade da Universidade de Harvard, a nível local e mundial (HARVARD, 2014). Segundo Harvard (2014), a iniciativa busca ampliar e melhorar a experiência móvel dos alunos, professores, visitantes, funcionários e pessoas que interagem de alguma forma com o câmpus e a comunidade da universidade. O aplicativo disponibiliza diversas funcionalidades como: notícias e novidades sobre as diversas áreas da universidade, informações acerca de cursos, realizar buscas nos acervos da biblioteca, mapas de ônibus e rotas para chegar até a universidade, informações de serviços e horários de funcionamento, entre outras funcionalidades.

A aplicação permite fazer um caminho em torno do câmpus através da visualização de mapas de ônibus e rotas em tempo real, visualização de horários de ônibus, calendários, listas de telefones, e ainda caso houver mudanças nos serviços o usuário é notificado (HARVARD, 2014). Outra funcionalidade referente a mapas é a possibilidade de navegar pelo câmpus da universidade, prédios, salas de aulas, casas, escritório, entre outros. Podendo mostrar a localização através de um mapa 3D, onde neste é possível dar zoom, rolar em qualquer direção, além de pesquisar por locais (HARVARD, 2014). A função de mapa da universidade pode ser visualizada na Figura 21. Com esta função, a locomoção dos alunos dentro e fora da universidade se torna mais rápida e segura, podendo contar com diversas informações centralizadas em um único aplicativo.

Figura 21– Mapa da universidade



Fonte: Harvard (2014).

Conforme pode ser observado na Figura 21, o mapa da universidade está disponível na aplicação e nele é possível efetuar buscas, traçar rotas e obter informações. Os mapas são integrados com a plataforma Google Maps, possibilitando assim efetuar zoom, e se posicionar

no mapa através de sua localização atual. A aplicação está disponível para iPhone/iPad/iPod Touch que tenham o iOS na versão 5.0 ou superior e para Android com versão superior à 2.2 (HARVARD, 2014).

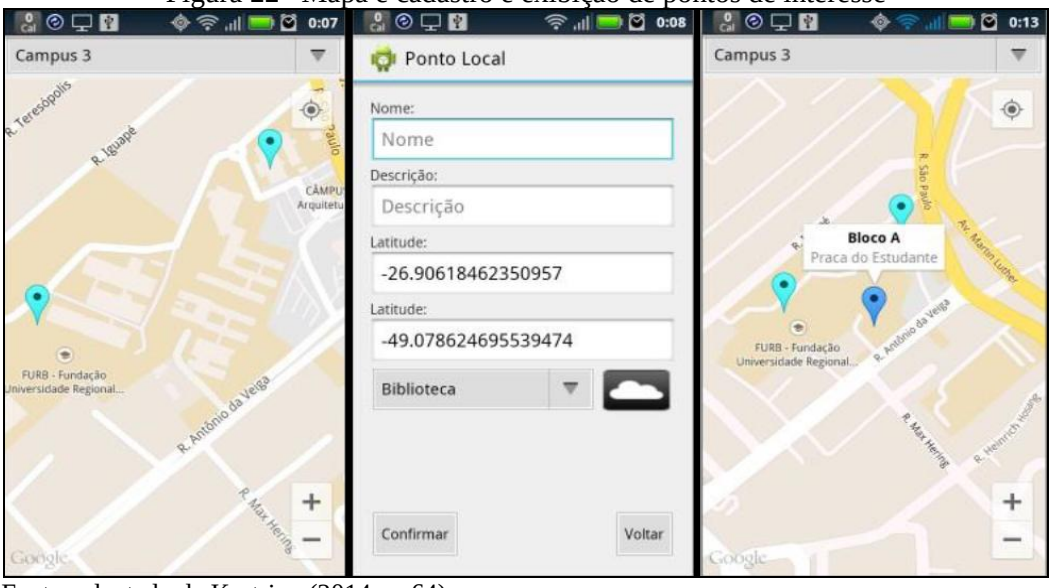
2.5.3 Compartilhamento de geolocalização utilizando mapas e notificações

Segundo Kestring (2014), o trabalho consiste em um aplicativo para a plataforma Android, que através de um mapa interativo e recursos de geolocalização, apresenta a posição do usuário, a posição de outros usuários logados e pontos de interesse cadastrados pelo usuário ou pela universidade. O aplicativo Android utiliza arquitetura cliente e se comunica com um servidor (*Web service*) responsável por suprir as informações utilizadas.

Utilizou-se da nova versão do Google Maps no Android, o Google Maps Android API v2 e para notificações implementou-se um servidor GCM Cloud Connection Server para que os dispositivos móveis também possam enviar notificações ao Google Cloud Message da Google (KESTRING, 2014).

Conforme Kestring (2014), a Figura 22 demonstra primeiramente a esquerda o mapa do aplicativo de acordo com a posição atual do usuário no mapa (balão azul escuro). A imagem ao centro demonstra o cadastro de um novo ponto, que pode ser realizado através de um duplo clique no mapa ou selecionando a opção novo ponto, a qual se baseia na posição atual do usuário. Na última imagem mostra-se o novo ponto cadastrado e sua descrição no mapa.

Figura 22– Mapa e cadastro e exibição de pontos de interesse



Fonte: adaptado de Kestring (2014, p. 64).

Por fim, de acordo com Kestring (2014), o trabalho se propôs a auxiliar estudantes, visitantes e pessoas com algum tipo de dificuldade de locomoção nas regiões da FURB. Como

fonte de estudo e aplicação do trabalho, foi escolhido o evento Interação FURB, onde os participantes têm poucos conhecimentos das localidades físicas da FURB. Neste caso o aplicativo disponibiliza o cadastro de qualquer pessoa sem nenhum vínculo com a universidade. Ainda de acordo com Kestring (2014), a integração entre as várias tecnologias empregadas se mostrou de grande valia devido às possibilidades que as ferramentas trazem.

2.5.4 Kinect e smartphones para rastreamento de movimento do pulso

Segundo Baião e Lima (2014), o trabalho consiste em o uso de um dispositivo móvel que possua sensores acelerômetro e giroscópio, para que em conjunto com o Kinect, efetue o rastreamento de movimentos dos pulsos para aplicações e jogos. Para isto, foi utilizado um conjunto de bibliotecas para desenvolvimento do Kinect e estabelecido uma comunicação via *Wi-Fi* com o dispositivo móvel. Para utilizar a detecção de movimentos do pulso, foi criado um jogo de espadas para dois jogadores utilizando o *framework* Unity 3D.

Ainda segundo Baião e Lima (2014), a movimentação corporal dos personagens do jogo de luta criado é feita de modo a copiar os movimentos dos jogadores, para isso foi utilizado o *plugin* do Kinect ZigFu. A movimentação da arma segundo o movimento de pulso de cada jogador é calculada de acordo com os dados dos sensores dos dispositivos móveis, utilizando o *plugin* GyroDroid.

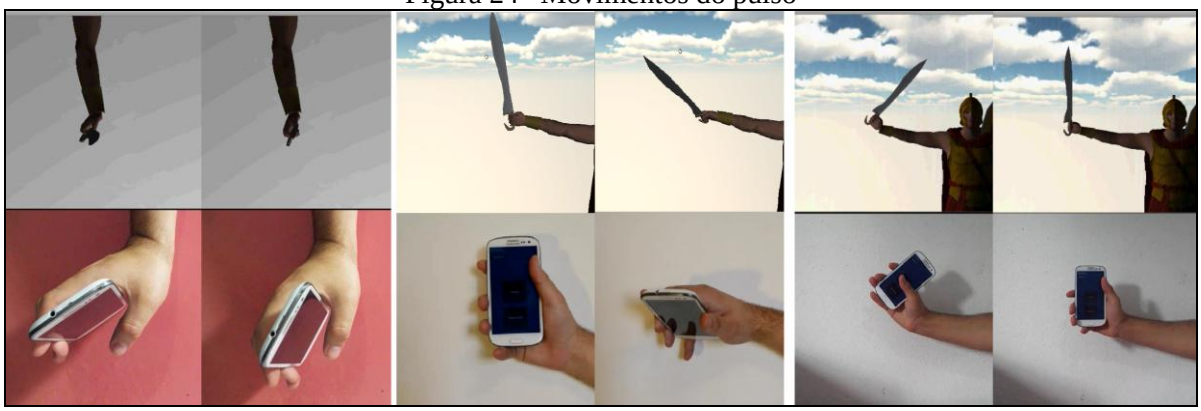
Conforme Baião e Lima (2014), a Figura 23 demonstra o jogo em execução, onde os dispositivos móveis estão conectados a uma rede *Wi-Fi* e se comunicando com um *notebook*, e este *notebook* está conectado ao Kinect. Os dois jogadores estão se combatendo no jogo, onde cada um possui seu *avatar* e estes estão voltados um a frente do outro, imitando os movimentos executados pelos jogadores.

Figura 23– Jogo em execução

Fonte: Baião e Lima (2014).

Conforme Baião e Lima (2014), a Figura 24 demonstra o resultado dos movimentos copiados através dos sensores acelerômetro e giroscópio do dispositivo móvel e o mapeamento realizado para o personagem do jogo.

Figura 24– Movimentos do pulso



Fonte: adaptado de Baião e Lima (2014, p. 40-41).

Por fim, ainda conforme Baião e Lima (2014), a Figura 25 demonstra o uso do Kinect para mapear os movimentos realizados pelo corpo do jogador e aplicados no modelo do jogo. O uso conjunto do Kinect e sensores proposto neste trabalho permite o rastreamento do corpo e um detalhamento preciso dos movimentos do pulso, fundamentais para a simulação de uma luta com espadas (BAIÃO; LIMA, 2014).

Figura 25– Movimentos do corpo copiado pelo Kinect

Fonte: adaptado de Baião e Lima (2014, p. 42).

2.5.5 Comparativo entre os trabalhos correlatos

Com base nas informações obtidas a partir dos trabalhos descritos acima, montou-se o Quadro 1, onde estão relacionadas as principais características de cada trabalho.

Quadro 1 – Comparativo entre trabalhos

Características dos trabalhos relacionados	Lima, Mendes Neto e Rodrigues (2012)	Harvard (2014)	Kestring (2014)	Baião e Lima (2014)
geolocalização	Sim	Sim	Sim	Não
framework Unity 3D	Não	Não	Não	Sim
uso de mapas	Sim	Sim	Sim	Não
plataformas	Web	Web, IOS e Android	Android	Mac e Windows
tridimensional	Sim	Não	Não	Sim
uso de sensores	Sim, GPS	Sim, GPS	Sim, GPS	Sim, acelerômetro e giroscópio
aplicação/jogo sério	Sim, jogo interativo para auxílio na aprendizagem de alunos	Sim, mapa interativo para auxílio de alunos da universidade de Harvard	Sim, mapa interativo para auxílio de alunos da universidade FURB	Não

O Quadro 1, demonstra que todos os trabalhos utilizam algum sensor no auxílio e obtenção de resultados e valores, como por exemplo, o GPS que foi utilizado em todos os

trabalhos exceto o trabalho de Baião e Lima (2014). O mesmo ocorre para o uso de geolocalização, onde somente o trabalho de Baião e Lima (2014) não o utilizam.

Tanto o trabalho de Lima, Mendes Neto e Rodrigues (2012) quanto o trabalho de Baião e Lima (2014), são trabalhos construídos na plataforma tridimensional, porém, ambos utilizam ferramentas e bibliotecas diferentes, o primeiro utiliza a biblioteca Open GL para o desenvolvimento tridimensional, já o segundo utiliza o *framework* Unity 3D. O trabalho de Harvard (2014) e Kestring (2014) são mapas interativos com o principal objetivo de auxiliar os alunos e pessoas que frequentam a universidade a se locomoverem e obter informações relevantes.

3 DESENVOLVIMENTO

Neste capítulo são apresentadas as etapas de desenvolvimento do aplicativo proposto. Na seção 3.1 são apresentados os principais requisitos. A seção 3.2 apresenta a especificação. A seção 3.3 apresenta de forma detalhada a implementação do aplicativo e, por fim, a seção 3.4 apresenta os resultados obtidos, discussões e sugestões de melhoria.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

Os Requisitos Funcionais (RF) e Requisitos Não Funcionais (RNF) deste trabalho são:

a) disponibilizar um mapa da FURB 3D dos blocos R,S e T do câmpus 1 (Requisito Funcional – RF);
b) permitir utilizar a posição atual do usuário para posicionar o mesmo em relação a FURB (RF);
c) permitir que o usuário utilize o mapa sem estar presente fisicamente na FURB (RF);
d) o aplicativo deve permitir que o usuário utilize um *avatar* para representá-lo no mundo virtual da FURB (RF);
e) permitir ser executado em sistema operacional Android 4.0 ou versões superiores (Requisito Não Funcional – RNF);
f) ser desenvolvido utilizando o *framework* Unity na versão 4.6 (RNF);
g) possuir os seus modelos tridimensionais construídos utilizando o *framework* tridimensional Blender na versão 2.7.4 (RNF);
h) utilizar o software Adobe Photoshop CS6 para criação das texturas (RNF);
i) utilizar o periférico GPS para obter a geolocalização do usuário (RNF);
j) utilizar o sensor de bussola para obter informações de troca de direção (RNF).

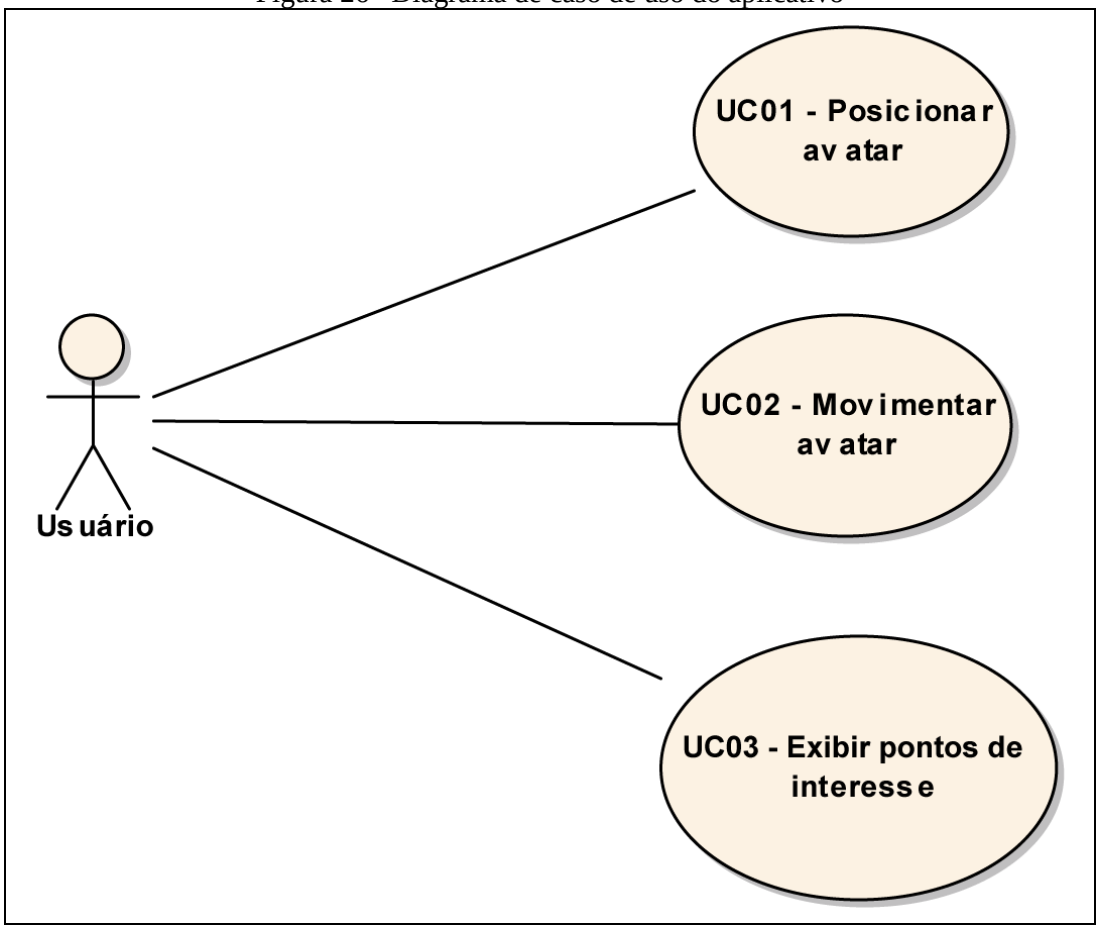
3.2 ESPECIFICAÇÃO

A especificação do aplicativo foi realizada seguindo a análise orientada a objetos, utilizando a *Unified Modeling Language* (UML), modelada através da ferramenta *Enterprise Architect*. Neste trabalho foram elaborados os diagramas de casos de uso, de classes e de atividades.

3.2.1 Diagrama de casos de uso

Nesta seção serão apresentados os casos de uso do aplicativo, ilustrados na Figura 26. Identificou-se somente um ator, denominado como `Usuário`, o qual utiliza o aplicativo.

Figura 26– Diagrama de caso de uso do aplicativo



Quadro 2– Caso de uso UC01-Posicionar *avatar*

Descrição	Este caso de uso tem por objetivo posicionar o *avatar* no mapa tridimensional, através da localização real do Usuário.
Pré-Condição	Estar com o *GPS* ativado, possuir conexão com a internet e estar com o aplicativo aberto.
Cenário Principal	1. O Usuário se movimenta fisicamente; 2. O aplicativo busca as coordenadas geográficas do Usuário através do *GPS* e atualiza a posição do *avatar* no mapa.
Fluxo Alternativo 1	A qualquer momento o Usuário pode efetuar um zoom, através da movimentação de dois dedos na tela do dispositivo.
Pós-Condição	O *avatar* deve ser movimentado pelo mapa, de acordo com o movimento real do Usuário.

Quadro 3– Caso de uso UC02-Movimentar *avatar*

Descrição	Este caso de uso tem por objetivo movimentar o *avatar* no mapa tridimensional através do movimento do dedo do Usuário na tela do dispositivo.
Pré-Condição	Possuir o *GPS* desativado e estar com o aplicativo aberto.
Cenário Principal	1. O Usuário aciona o *touch* do dispositivo, deslizando o dedo na tela do dispositivo; 2. O aplicativo movimenta o *avatar* no mapa utilizando a direção do *touch* a uma velocidade média de caminhada.
Fluxo Alternativo 1	A qualquer momento o Usuário pode efetuar um zoom, através da movimentação de dois dedos na tela do dispositivo.
Pós-Condição	O *avatar* deve ser movimentado na direção que o Usuário arrastou o dedo na tela do dispositivo.

Quadro 4– Caso de uso UC03-Exibir pontos de interesse

Descrição	Este caso de uso tem por objetivo exibir um ponto de interesse no mapa.
Pré-Condição	Estar com o aplicativo aberto e movimentar o *avatar* através dos casos de uso UC02 ou UC01.
Cenário Principal	1. O aplicativo verifica se o *avatar* esta dentro da área do ponto de interesse; 2. O aplicativo exibe o ponto de interesse e sua descrição; 3. O Usuário visualiza o ponto de interesse no mapa e sua descrição.
Exceção 1	O ponto de interesse não será exibido, caso o *avatar* não esteja dentro de sua área de abrangência, neste caso, nada acontece e o caso de uso é finalizado.

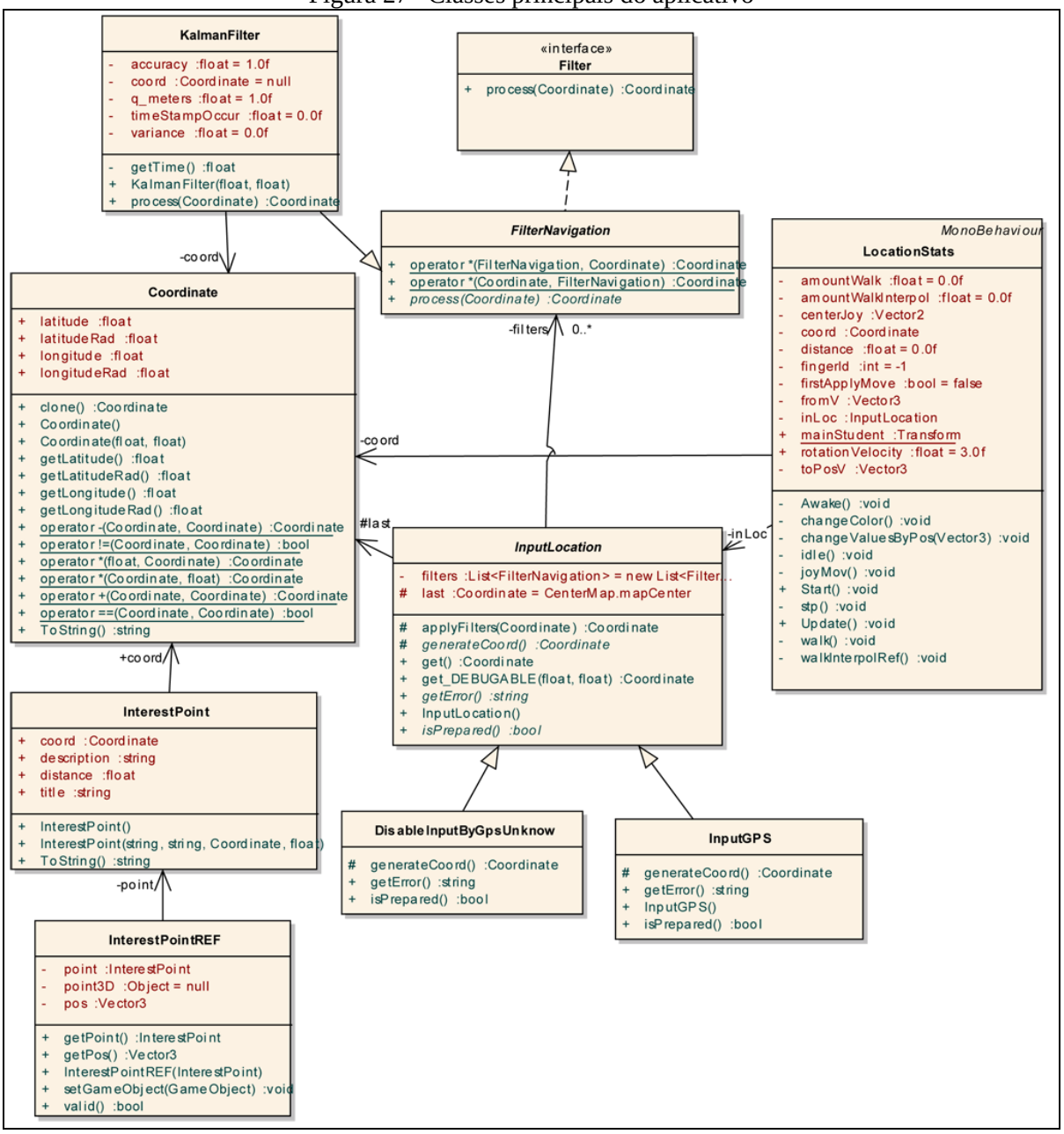
3.2.2 Diagramas de classes

Nesta seção serão descritas as classes utilizadas no aplicativo, suas relações e estruturas. A seção 3.2.2.1 apresenta as principais classes utilizadas no aplicativo. Por fim, a seção 3.2.2.2 apresenta as classes utilitárias do aplicativo.

3.2.2.1 Classes principais

O diagrama de classes da Figura 27 apresenta as principais classes do aplicativo, seus relacionamentos e estruturas.

Figura 27– Classes principais do aplicativo



A classe `InputLocation` provê a geração de coordenadas e aplicação de filtros (`Filter`) para eliminação de ruídos, valores fora do padrão e possíveis erros. Todas as consultas de valores de latitude e longitude realizados no GPS passam pelo tratamento desta classe. Existem duas implementações para esta classe `DisableInputByGpsUnknow` e `InputGPS`.

A classe `DisableInputByGpsUnknow` é uma implementação da classe `InputLocation`. Esta classe provê uma implementação “vazia”, ou seja, não retorna valores de coordenadas que representam algum movimento, sendo instanciada somente quando o GPS está desativado ou retornando algum erro para o dispositivo.

A classe `InputGPS` também é uma implementação da classe `InputLocation`. Esta classe retorna valores de coordenadas geográficas através de instâncias da classe `Coordinate`, sendo utilizados na classe base `InputLocation`, para aperfeiçoamento de seus valores. Os valores de coordenadas latitude e longitude são consultados no GPS, e ao contrário da classe `DisableInputByGpsUnknow`, esta classe é instanciada somente quando o GPS está ativado e sem gerar nenhum erro ao ler seus dados.

A classe `Coordinate` representa os valores de coordenada geográfica latitude e longitude utilizados em todo o aplicativo. Esta classe provê operações básicas entre coordenadas e conversão de valores geométricos.

A classe `FilterNavigation` é uma implementação de filtro (interface `Filter`), que disponibiliza um relacionamento entre um filtro e uma coordenada, respectivamente `FilterNavigation` e `Coordinate`. Esta classe é encapsulada na classe `InputLocation`, que aplica o produto de todos os seus filtros em sua coordenada, determinando assim a coordenada final retornada pelo método `InputLocation.get`.

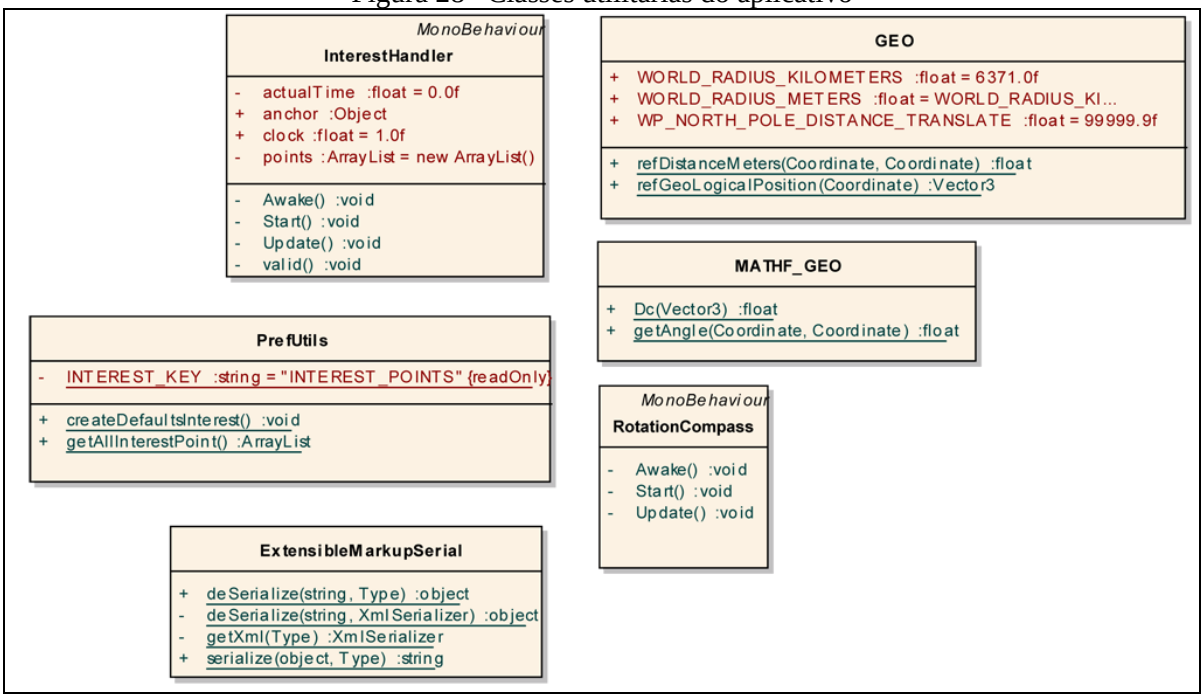
A classe `KalmanFilter` representa uma implementação de `FilterNavigation`. Esta classe provê o filtro de `Kalman`, que por sua vez atua na eliminação de ruídos e valores fora do padrão para coordenadas geográficas.

A classe `InterestPoint` representa os pontos de interesse existentes no aplicativo, contendo sua descrição, coordenada geográfica e abrangência, respectivamente representados pelos campos `description`, `coord` e `distance`. A representação do ponto de interesse em objeto tridimensional se dá pela classe `InterestPointREF`, esta por sua vez efetua o georreferenciamento, transformando a coordenada geográfica da instância de `InterestPoint` em coordenada tridimensional.

3.2.2.2 Classes utilitárias

O diagrama da Figura 28 apresenta as classes utilitárias do aplicativo, seus relacionamentos e estruturas.

Figura 28– Classes utilitárias do aplicativo



A classe `RotationCompass` é responsável por tratar as rotações do *avatar* no mapa, de acordo com a direção da bússola do dispositivo. Esta classe somente aplica a direção da bússola no *avatar* caso a classe `InputGPS` esteja instanciada como entrada de coordenadas padrão.

A classe `InterestHandler` é responsável por controlar a exibição dos pontos de interesse no mapa, levando em conta a distância entre o ponto e o *avatar*. Para carregar os pontos de interesse existentes, é utilizado o método `getAllInterestPoint` da classe `PrefUtils`, este por sua vez irá consultar *eXtensible Markup Language* (XML) na base de dados e efetuar a conversão para uma coleção de `InterestPoint`. A conversão é realizada pela classe `ExtensibleMarkupSerial`, que é responsável pelas operações de conversão do formato XML para uma instância de classe e instância de classe para o formato XML.

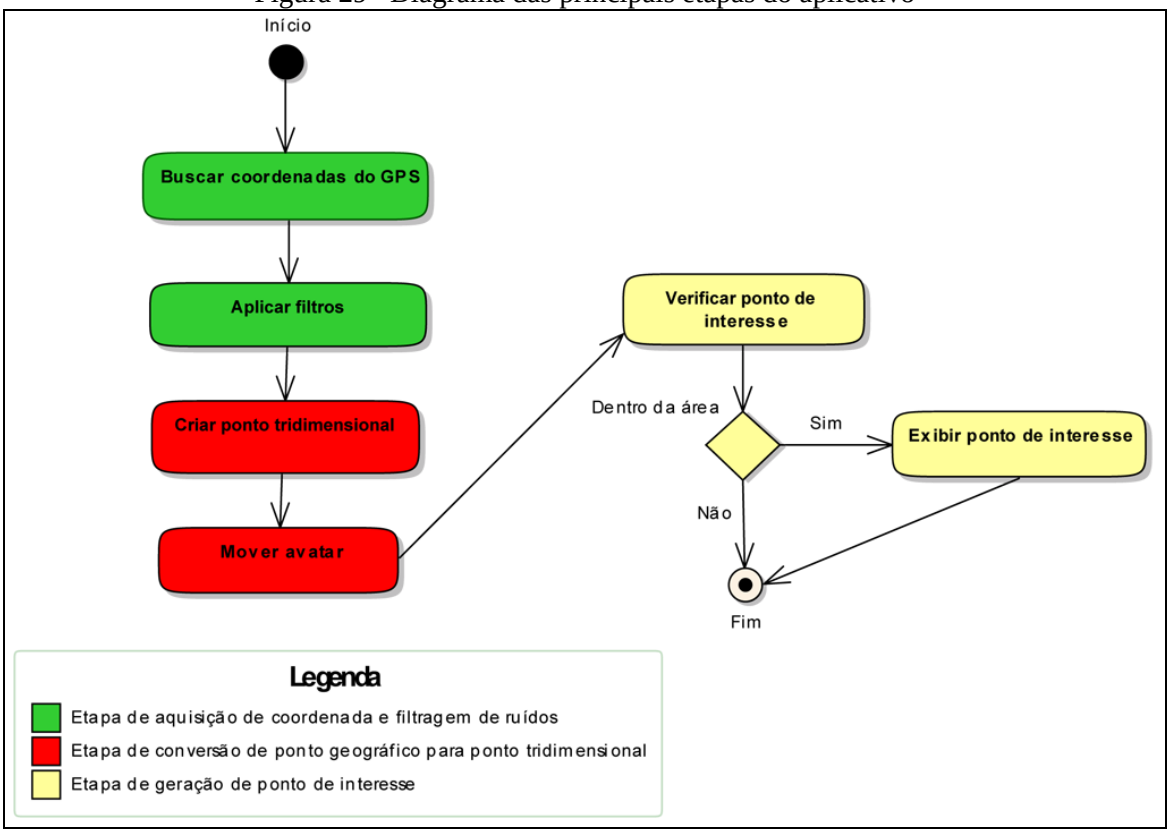
A classe `GEO` é responsável pelas operações de georreferenciamento, transformando valores de coordenada em coordenadas tridimensionais, operação representada pelo método `refGeoLogicalPosition`. Esta classe conta com a implementação da fórmula de Haversine, responsável por retornar a distância entres dois pontos de coordenada geográfica, representado pelo método `refDistanceMeters`.

A classe `MATHF_GEO` auxilia a classe `GEO`, disponibilizando operações para determinar o ângulo entre coordenadas geográficas.

3.2.3 Diagrama de atividade

O diagrama de atividades da Figura 29 representa as principais etapas do aplicativo: etapa de aquisição de coordenada, etapa de georreferenciamento e etapa de exibição de pontos de interesse.

Figura 29– Diagrama das principais etapas do aplicativo



Na primeira etapa, as coordenadas são capturadas pelo *GPS* em formato de latitude e longitude, estes valores são recalculados através de um filtro que inibe possíveis ruídos, como valores fora do padrão aceitável, como por exemplo, valores muito distantes em pouco tempo. Ao final desta etapa, as coordenadas já estarão filtradas e seus valores serão representados através de uma instância da classe `Coordinate`.

Na segunda etapa, a coordenada geográfica é transformada em coordenada tridimensional (georreferenciamento), representando a posição que o *avatar* deve ser movido no mapa. Após determinar o ponto tridimensional na qual o *avatar* deve ser movido, o mesmo irá "caminhar" até este ponto no mapa.

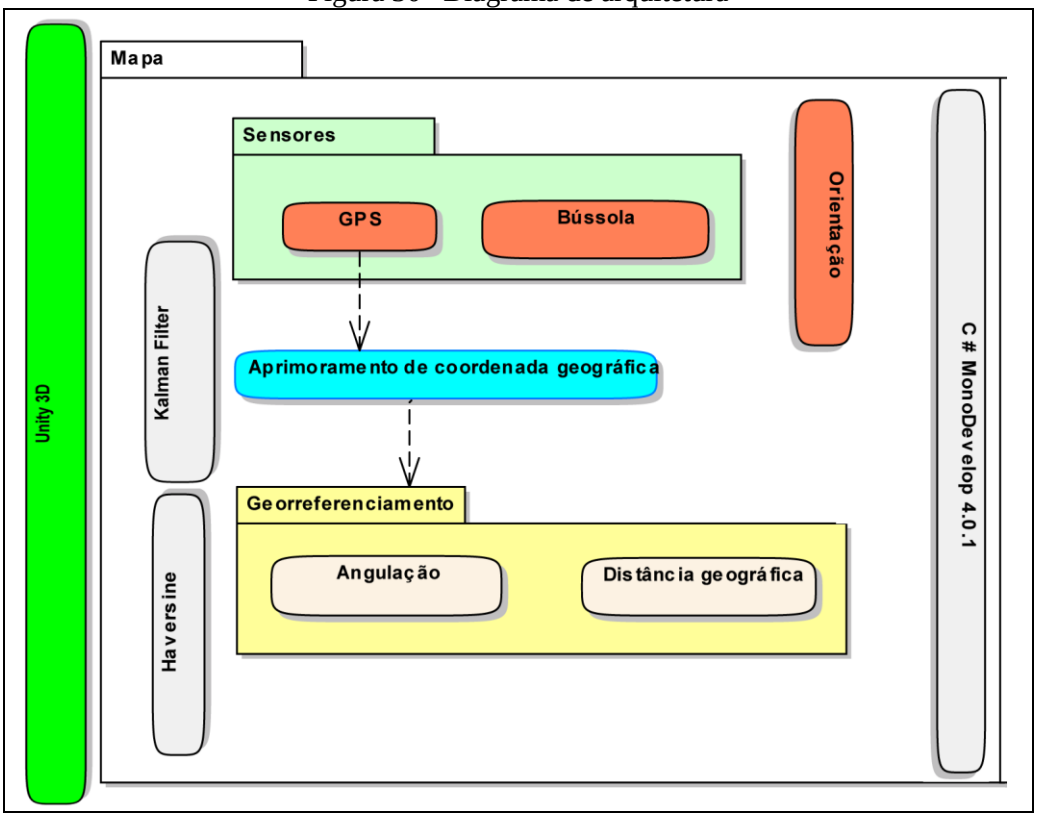
Na última etapa, é verificado se o *avatar* se encontra dentro da área de algum ponto de interesse existente no mapa, para isto, é necessário saber a distância do *avatar* até o ponto verificado e calcular se esta distância é menor que distância definida para o ponto de

interesse, caso a distância seja menor o ponto de interesse e sua descrição são exibidos no mapa na sua localização.

3.2.4 Diagrama de arquitetura

O diagrama de arquitetura é representado pela Figura 30. Nela estão contidas as principais tecnologias e técnicas utilizadas.

Figura 30– Diagrama de arquitetura



A arquitetura demonstrada na Figura 30 se dá em torno do *framework* Unity 3D utilizando a linguagem C Sharp e ambiente de desenvolvimento integrado (*Integrated Development Environment* - IDE) MonoDevelop 4.0.1. Nele são executadas as principais etapas do aplicativo, como a obtenção de coordenadas geográficas através do sensor GPS, a determinação da orientação baseado nas valores da bússola, aprimoramento das coordenadas geográficas aplicando filtros de ruído e a etapa de georreferenciamento.

3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação, respectivamente seção 3.3.1 e seção 3.3.2.

3.3.1 Técnicas e ferramentas utilizadas

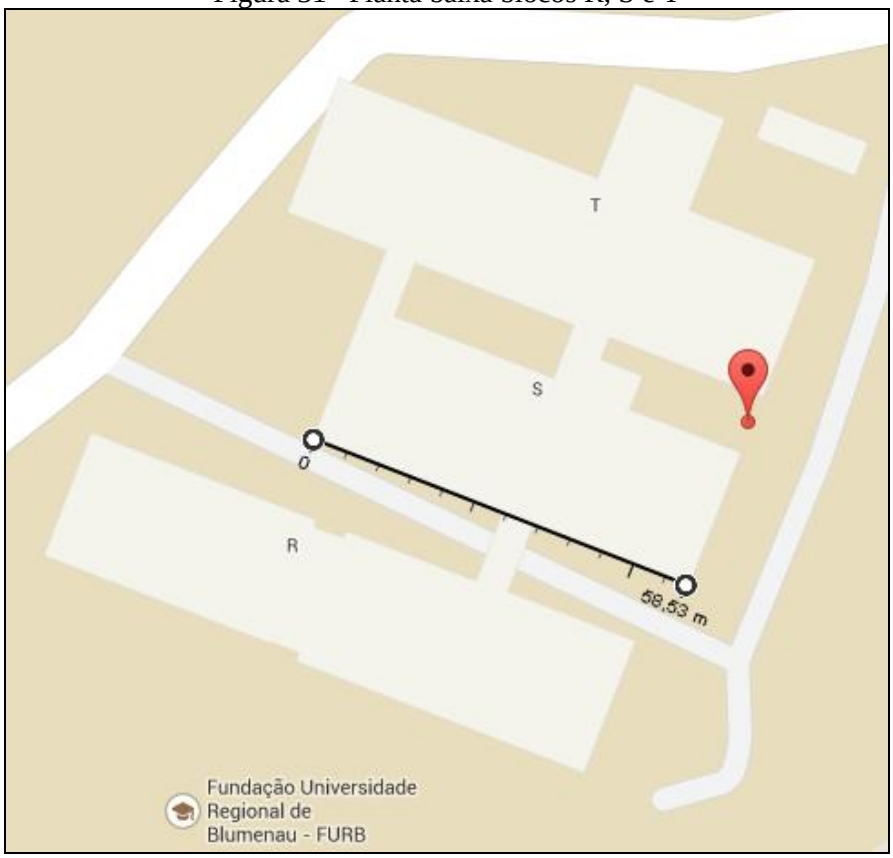
O aplicativo foi desenvolvido no *framework* Unity 3D em conjunto com a IDE MonoDevelop 4.0.1, utilizando a linguagem de programação C Sharp. Foram utilizados também os softwares Blender 2.7.4 para modelagem tridimensional e criação de animações do *avatar*, Adobe Photoshop CS6 para texturização dos modelos tridimensionais e o Google Maps para obtenção das plantas baixa dos blocos da FURB, através de sua ferramenta de mapas *indoor*. O *framework* Unity 3D permite a criação de aplicações multiplataforma, disponibilizando componentes para criação de físicas de corpos rígidos, animações, iluminação e sombreamento, sistema de partículas, entre outros recursos. Deste *framework* foram utilizados os componentes para animação do *avatar*, iluminação e sombreamento da cena onde está contido o mapa, além de recursos básicos como transformações geométricas e matemáticas trigonométricas. O acesso aos valores dos *hardwares* da bússola e GPS, também foram efetuados através do *framework* Unity 3D, onde é disponibilizada uma interface para acesso aos dados.

As seções a seguir detalham como foi implementado o aplicativo, relatando as etapas de criação do mapa da FURB, etapa de obtenção e ajuste das coordenadas geográficas, etapa de georreferenciamento e por fim a etapa de gerenciamento dos pontos de interesses.

3.3.1.1 Criação do mapa

A primeira etapa do desenvolvimento do aplicativo é a criação do mapa, a partir de uma região da FURB. Nesta etapa foram escolhidos os blocos R, S e T do câmpus um da FURB como região a ser trabalhada. A planta baixa, medidas utilizadas e ponto pivô (balão vermelho) foram obtidos através do Google Mapas, conforme Figura 31 abaixo.

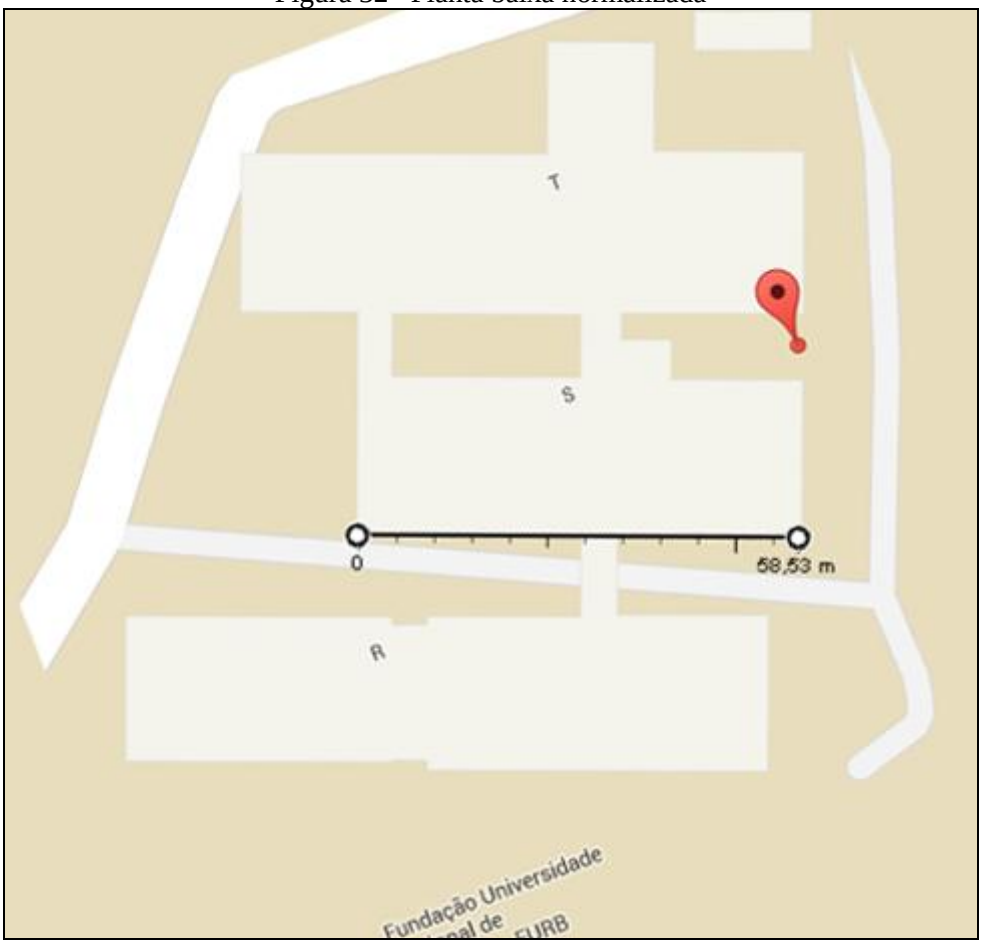
Figura 31– Planta baixa blocos R, S e T



Fonte: Google (2015).

Esta planta baixa contém as medidas do bloco S, e o ponto pivô utilizado no mapa. O ponto pivô é utilizado na etapa de georreferenciamento, que será explicada na seção 3.1.1.3. A planta baixa foi normalizada para servir como base para modelagem, pois, os blocos se encontram “inclinados” em relação ao polo norte geográfico, conforme Figura 32.

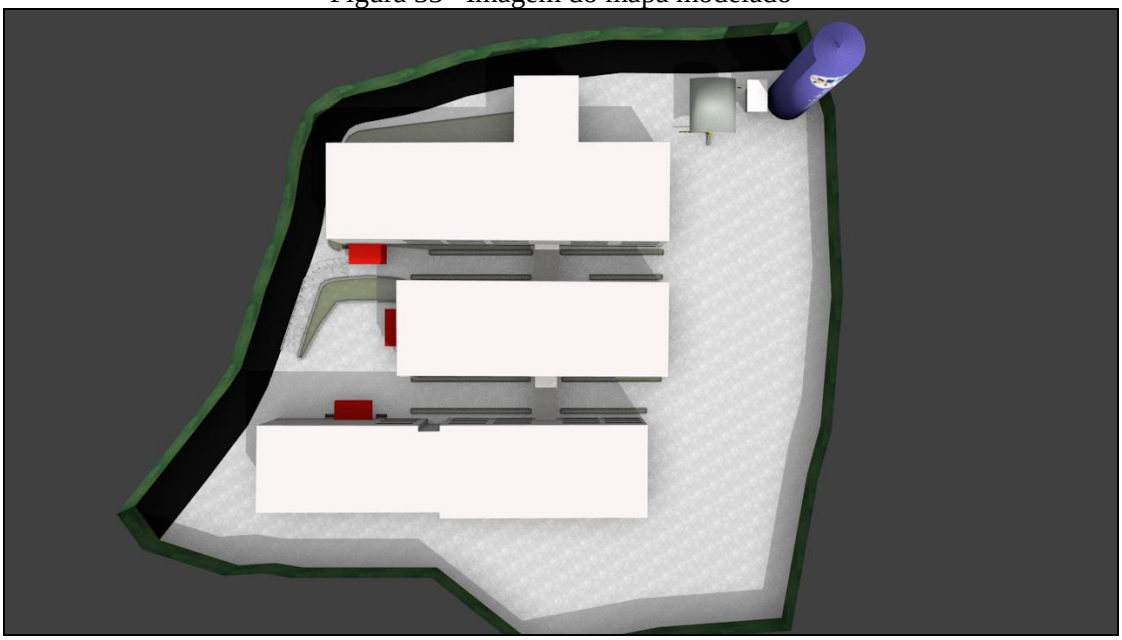
Figura 32– Planta baixa normalizada



Fonte: adaptado de Google (2015).

Esta planta sem “inclinação” foi utilizada como base para modelagem no software Blender 2.7.4. Os blocos foram construídos conforme a topologia da planta, e ao final receberam suas escalas conforme os valores das medidas da planta. Outros objetos pertencentes ao mapa foram modelados com base de pesquisa em campo, onde foram coletas fotos e medidas.

Figura 33– Imagem do mapa modelado



A Figura 33 ilustra o mapa modelado conforme a planta da Figura 32. O mapa foi importado para o *framework* Unity, onde será utilizado como base de terreno para o aplicativo. Nesta etapa são incorporados ao mapa as iluminações e sombreamentos utilizando *shaders* da Unity.

3.3.1.2 Obtenção e ajustes de coordenadas geográficas

Nesta etapa é demonstrado como são obtidas as coordenadas geográficas latitude e longitude através do GPS, e os filtros aplicados para melhorar a precisão dos valores. No Quadro 5 é demonstrado o código fonte responsável por capturar as coordenadas.

Quadro 5 – Captura de coordenadas geográficas

```
01.  protected override Coordinate generateCoord() {
02.              return new
03.  Coordinate(Input.location.lastData.latitude,
04.  Input.location.lastData.longitude);
05.        }
```

A função `generateCoord` da classe `InputGPS` retorna uma instância da classe `Coordinate`, passando como parâmetro os valores do GPS obtidos através do comando `Input.location.lastData.latitude` e `Input.location.lastData.longitude`, este comando retorna os valores em graus de latitude e longitude respectivamente.

Após obter a instância de `Coordinate` com os valores do GPS, são aplicados os filtros que corrigem os valores de coordenadas e aumentam sua precisão, conforme pode ser observado no Quadro 6.

Quadro 6 – Aplicação dos filtros nas coordenadas

```
01.  protected virtual Coordinate applyFilters(Coordinate coord) {
02.        Coordinate c = coord;
03.        foreach(FilterNavigation f in filters) {
04.              c = c * f;
05.        }
06.        return c;
07.  }
```

No método `applyFilters` da classe `InputLocation`, são multiplicados todos os filtros existentes pela coordenada obtida. Neste caso existe somente um filtro que é aplicado sobre as coordenadas de GPS, chamado de Filtro de Kalman. O Quadro 7 apresenta a implementação do filtro de Kalman para tratar valores de coordenada latitude e longitude.

Quadro 7 – Implementação do filtro de Kalman

```
01.  public KalmanFilter(float accuracy, float velocity) {
02.        this.accuracy = accuracy;
03.        this.q_meters = velocity;
04.  }
05.  public override Coordinate process(Coordinate c) {
06.        if (timeStampOccur == 0.0f) {
07.              //INITIALIZE
08.              timeStampOccur = getTime();
09.              variance = this.accuracy * this.accuracy;
10.              this.coord = c.clone();
11.        } else {
12.              //PROCESS
13.              float timeInc_milliseconds = getTime() -
14.  this.timeStampOccur;
15.              if (timeInc_milliseconds > 0.0f) {
16.                    this.variance += timeInc_milliseconds * q_meters
17.  * q_meters;
18.                    this.timeStampOccur = getTime();
19.              }
20.              float K = variance / (variance + accuracy * accuracy);
21.
22.              this.coord = this.coord + ( K * ( c - this.coord ) );
23.              variance = (1 - K) * variance;
24.        }
25.        return coord;
26.  }
```

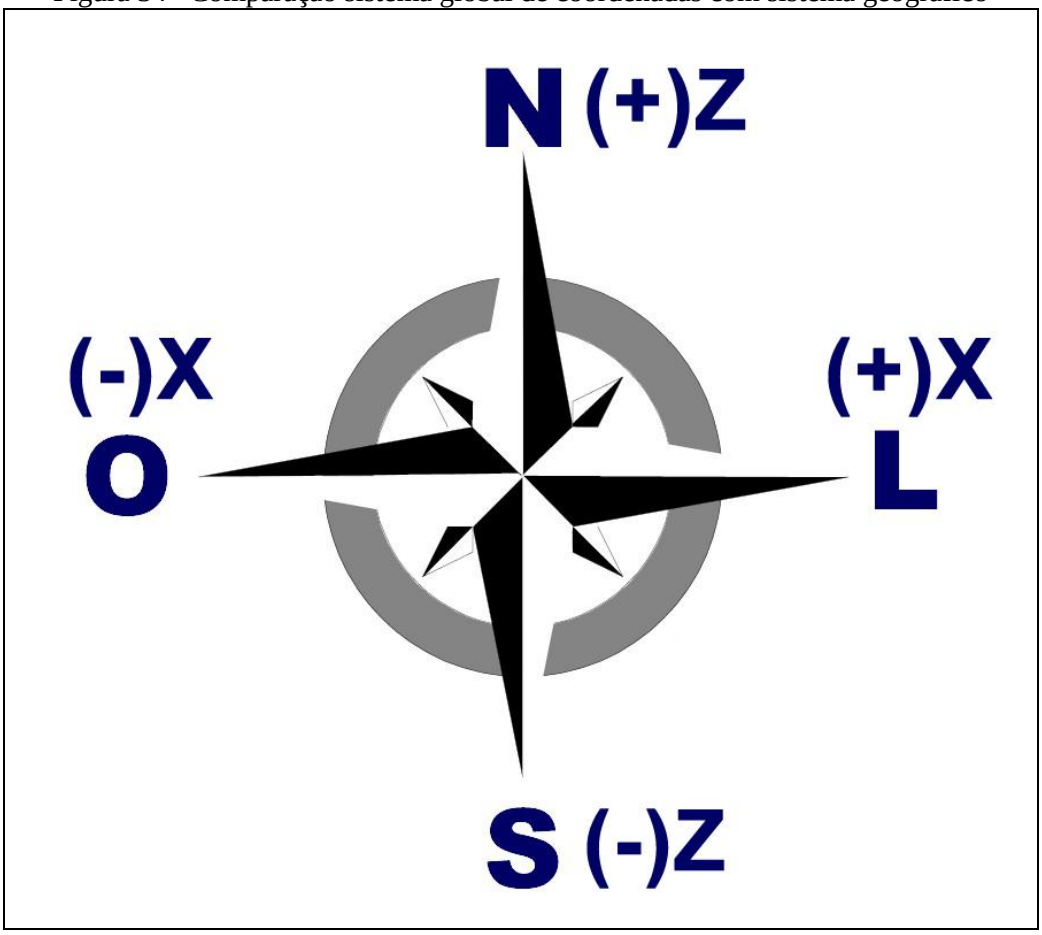
Primeiramente o filtro é instanciado passando dois valores como parâmetro, `accuracy` que significa o quão exato e preciso serão os valores retornados, e `velocity` que será a velocidade ou montante que pode ser atualizado em um segundo de tempo. O método `process` é responsável por processar a coordenada e retornar seu valor corrigido, para isso o algoritmo se baseia nos valores já definidos de `accuracy`, `velocity` e o tempo. Com essas variáveis é descoberta a `variance` (valor de dispersão máximo) possível para aquela coordenada baseada na `velocity`. Depois é calculado o coeficiente `K` baseado na `accuracy` em função da `variance`. A nova coordenada será calculada levando em conta o coeficiente `K` multiplicado pela diferença da coordenada atual subtraído pela última coordenada gerada. Ao

final do processo, uma nova `variance` é calculada, e o valor de tempo do último cálculo é computado na variável `timeStampOccur`, para assim ambos serem utilizados em um próximo cálculo.

3.3.1.3 Georreferenciamento

A etapa de georreferenciamento consiste em transformar os valores de coordenada geográfica latitude e longitude, para valores de coordenadas tridimensionais. Para isto, primeiramente é necessário normalizar a rotação do mapa de acordo com o sistema global de coordenadas utilizado no desenvolvimento do aplicativo no *framework* Unity.

Figura 34– Comparação sistema global de coordenadas com sistema geográfico



A Figura 34 apresenta uma comparação do sistema de coordenadas dos vetores de direção e o sistema de coordenadas geográfico. Quando a direção é totalmente voltada ao norte geográfico, as movimentações e translações através dos vetores de direção ocorrem somente em torno da coordenada z, quando voltado para leste ocorre somente em x, quando voltado para sul a movimentação é retrógada em z e finalmente quando voltado para oeste a movimentação é retrógada em x.

Conforme demonstrado na seção 3.1.1.1, o mapa do aplicativo foi modelado com o bloco T paralelo ao polo norte geográfico, ou paralelo a coordenada z do vetor direcional. Essa associação entre polo norte geográfico e coordenada z, é necessária para haver uma norma entre ambos e assim efetuar as transformações e movimentações de forma igual. Neste caso a norma diz que o polo norte geográfico é representado pelo movimento somente realizado pela coordenada tridimensional z.

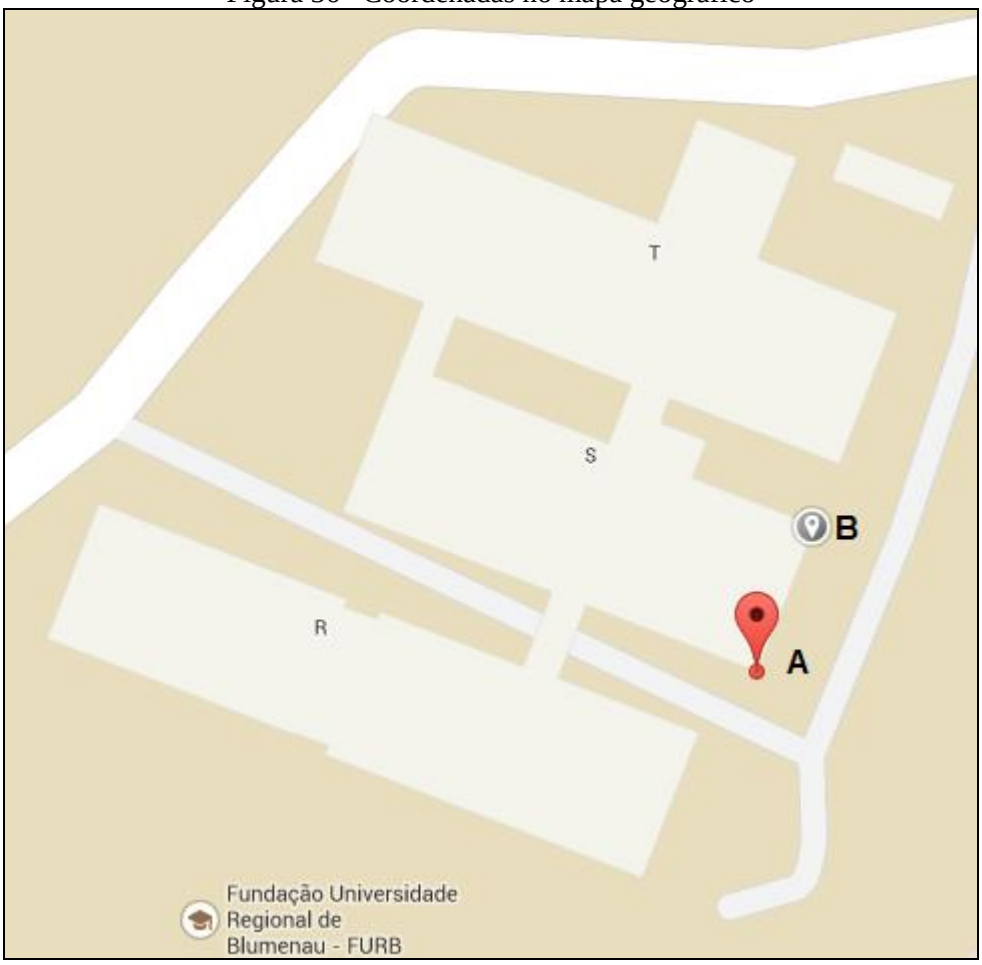
Após definir a norma acima, foi necessário posicionar o modelo do mapa corretamente de acordo com sua transformação geográfica. Para isto, foram retiradas duas coordenadas de latitude e longitude paralelas e retas no mapa. Após é retirado o ângulo entre estas coordenadas e aplicado a rotação deste ângulo no mapa modelado. Na Figura 35 são representadas as coordenadas A e B, que estão paralelas e retas na planta baixa normalizada do mapa.

Figura 35– Coordenadas paralelas e retas no mapa normalizado

Fonte: adaptado de Google (2015).

As duas coordenadas estão retas e paralelas em relação a planta baixa normalizada, demonstrando que ambas têm sua direção voltada ao polo norte ou coordenada de referência z, porém, seus valores de latitude e longitude são baseados no mapa geográfico, onde existe uma inclinação ou rotação, conforme demonstrado na Figura 36.

Figura 36– Coordenadas no mapa geográfico



Fonte: adaptado de Google (2015).

As duas coordenadas representam a inclinação da planta baixa em relação ao polo norte. Como já mencionado, para revelar esta inclinação (rotação) é necessário utilizar o ângulo entre estas coordenadas. Para descobrir o ângulo entre as coordenadas é utilizado o teorema de Pitágoras.

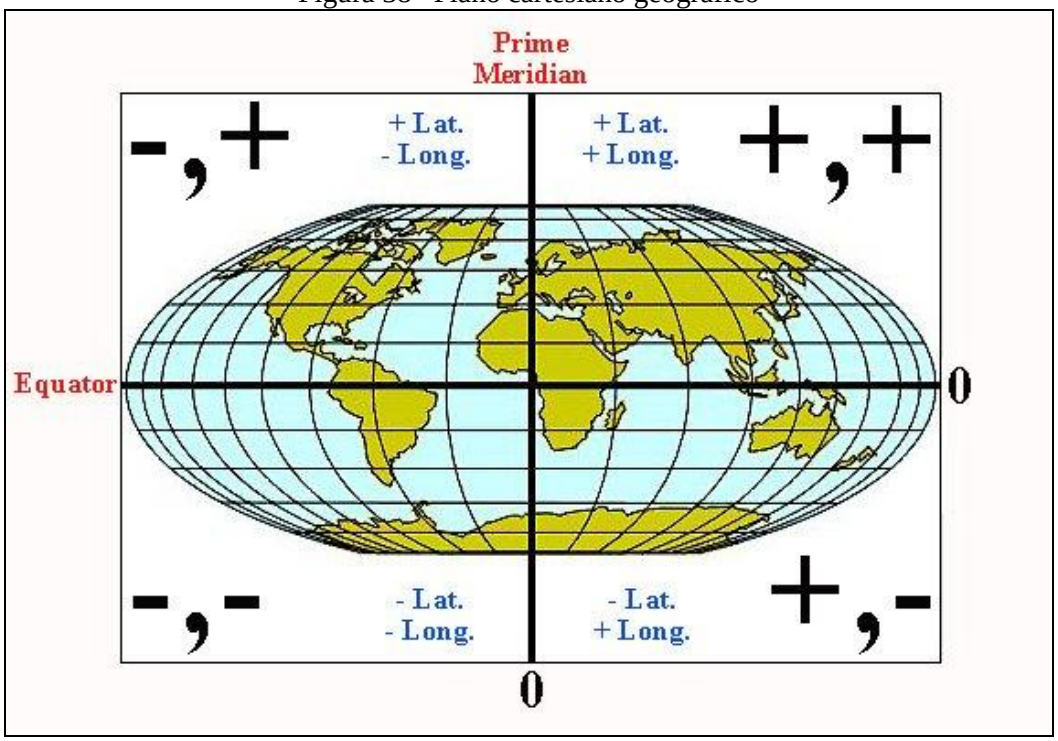
Na Figura 37 são demonstradas as razões básicas do teorema de Pitágoras, `CO` é o cateto oposto ao ângulo, `CA` cateto adjacente ao ângulo, `H` hipotenusa ou medida de distância entre `CO`, `CA` e `θ theta` que representa o ângulo a ser descoberto. Para aplicar as coordenadas geográficas ao teorema é necessário tratar o sistema de coordenadas geográficas como um sistema cartesiano, onde neste caso, latitude representa a coordenada `z` e longitude representa a coordenada `x`. Completando a equação, os valores de `A` e `B` são respectivamente a latitude da coordenada `A` e a latitude da coordenada `B`. Os valores de `C` e `D` são respectivamente a longitude da coordenada `A` e a longitude da coordenada `B`. Ao substituir os valores a `hipotenusa` será descoberta e assim poderá ser feito o inverso do cosseno ou arco cosseno a partir do cosseno representado pela relação entre `CA` e `H`, retornando o ângulo (`θ`).

Figura 37– Teorema de Pitágoras

$$CO = A - B$$
$$CA = C - D$$
$$H = \sqrt{CO^2 + CA^2}$$
$$\theta = A\cos\left(\frac{CA}{H}\right)$$

A Figura 38 mostra a associação entre plano cartesiano e coordenadas geográfica, a latitude representa os valores em direção aos polos, e também pode ser demonstrada como sendo a altura ou coordenada y do plano cartesiano. A longitude representa os valores em direção aos quadrantes leste e oeste, sendo demonstrado também como a coordenada x do plano cartesiano. Esta associação permite o uso de razões trigonométricas como o teorema de Pitágoras já demonstrado. O próximo passo é aplicar o ângulo calculado no mapa, através de uma rotação. Após este passo o mapa está geograficamente orientado em relação ao polo norte, conforme demonstrado na Figura 39, onde o mapa se encontra transformado na cena do aplicativo.

Figura 38– Plano cartesiano geográfico

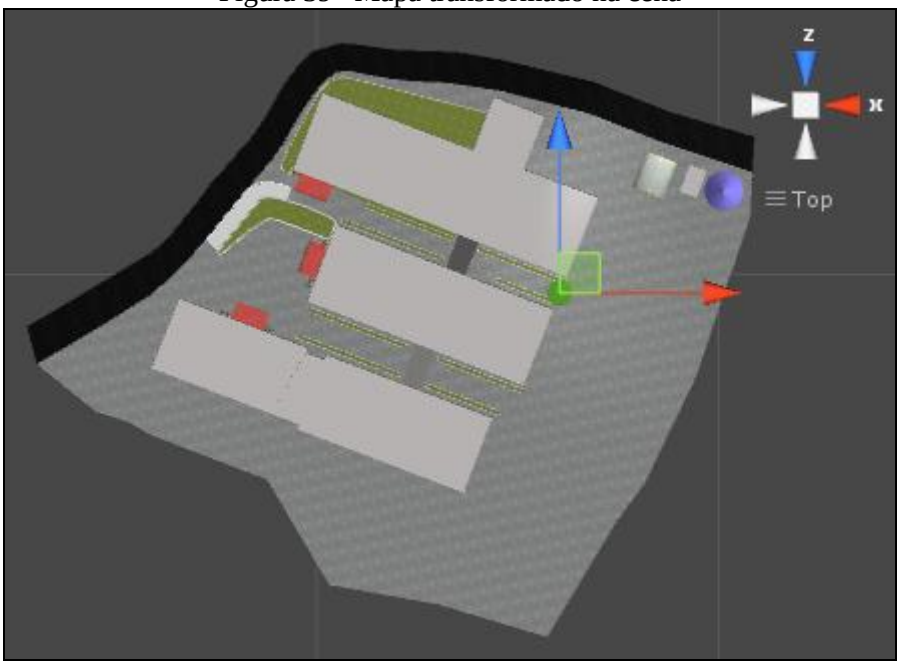


Fonte: Gold Ridge (2015).

Na Figura 39 além do mapa orientado, é demonstrado ponto pivô selecionado entre os dois blocos e como objeto filho do mapa na cena, recebendo todas as transformações do mapa (rotação, translação e escala). O ponto pivô é utilizado como ponto inicial para a descoberta de novos pontos no mapa, possuindo duas informações, sua coordenada/posição em relação

ao mapa (coordenadas x, y e z do plano tridimensional) e sua coordenada geográfica (latitude e longitude).

Figura 39– Mapa transformado na cena



Outra etapa necessária para o georreferenciamento é o cálculo de distância entre coordenadas geográficas. Esta distância é necessária para determinar quanto o *avatar* deve ser movimentado dentro do mapa, sendo que este movimento é executado com base nas mesmas medidas definidas na seção 3.3.1.1. Para solucionar este problema é utilizada a fórmula de Haversine. A fórmula de Haversine trata a terra como uma esfera perfeita, porém, como a terra não é uma esfera perfeita, é necessário utilizar o valor médio de seu raio, 6371 quilômetros (BALDI, 2011).

Conforme demonstrado no Quadro 8, a fórmula de Haversine utiliza as razões trigonométrica cosseno e seno tratando a terra como uma esfera, e a partir de seu raio calcula a distância entre duas coordenadas geográficas (latitude e longitude). Na implementação da fórmula, a variável `WORLD_RADIUS_METERS` tem o valor médio do raio da terra em metros, 6.371.000 metros.

Quadro 8 – Implementação da fórmula de Haversine

```
01.  public static float refDistanceMeters(Coordinate from, Coordinate
02.  to) {
03.              float step1 = Mathf.Sqrt( 1.0f - (
04.  Mathf.Pow(Mathf.Sin( (to.getLatitudeRad() -
05.  from.getLatitudeRad())/2.0f ),2.0f)
06.                                                  +
07.  Mathf.Cos(from.getLatitudeRad()) * Mathf.Cos(to.getLatitudeRad())
08.                                                  *
09.  Mathf.Pow(Mathf.Sin( (to.getLongitudeRad() -
10.  from.getLongitudeRad())/2.0f ),2.0f) ) );
11.
12.              float step2 = Mathf.Sqrt( ( Mathf.Pow(Mathf.Sin(
13.  (to.getLatitudeRad() - from.getLatitudeRad())/2.0f ),2.0f)
14.                                          +
15.  Mathf.Cos(from.getLatitudeRad()) * Mathf.Cos(to.getLatitudeRad())
16.                                          * Mathf.Pow(Mathf.Sin(
17.  (to.getLongitudeRad() - from.getLongitudeRad())/2.0f ),2.0f) ) );
18.
19.              float step3 = Mathf.Atan(step2/step1);
20.
21.              return WORLD_RADIUS_METERS * 2.0f * step3;
22.  }
```

Para finalizar é necessário utilizar a angulação e distanciamento propostos nesta seção para transformar a coordenada geográfica em um ponto no plano tridimensional. O Quadro 9 mostra o método `refGeoLogicalPosition`, responsável por criar e retornar o ponto tridimensional a partir da coordenada geográfica.

Quadro 9 – Método que efetua o georreferenciamento

```
01.  public static Vector3 refGeoLogicalPosition(Coordinate toGeoCoord)
02.  {
03.        return refGeoLogicalPosition(CenterMap.mapCenter,
04.  toGeoCoord, CenterMap.centerCoord.position);
05.  }
06.
07.  public static Vector3 refGeoLogicalPosition(Coordinate
08.  actualGeoCoord, Coordinate toGeoCoord, Vector3 actualLogicalPos) {
09.        float angle = MATHF_GEO.getAngle(actualGeoCoord,
10.  toGeoCoord);
11.        float distanceAmount = GEO.refDistanceMeters(actualGeoCoord,
12.  toGeoCoord);
13.        float sin = Mathf.Sin(angle * Mathf.Deg2Rad);
14.        float cos = Mathf.Cos(angle * Mathf.Deg2Rad);
15.        Vector3 forwardDir = new Vector3( sin, 0.0f, cos);
16.        Vector3 result = actualLogicalPos + (forwardDir *
17.  distanceAmount);
18.        return result;
19.  }
```

O método `refGeoLogicalPosition` é chamado passando a coordenada geográfica e coordenada tridimensional do ponto pivô como ponto de origem para o georreferenciamento, representado respectivamente por `CenterMap.mapCenter` e `CenterMap.centerCoord.position`. Também é passado como parâmetro a coordenada geográfica a ser calculada, representada por `toGeoCoord`. Primeiramente é calculado o ângulo

entre as duas coordenadas geográficas e armazenado na variável `angle`, depois é calculada a distância em metros entre essas coordenadas e armazenado na variável `distanceAmount`. O próximo passo é utilizar o ângulo presente na variável `angle` e calcular o seno e cosseno para criar o vetor direcional `forwardDir`. Como último passo é somado a posição do ponto pivô no plano tridimensional (`actualLogicalPos`) com a multiplicação da distância entre as coordenadas (`distanceAmout`) com a direção (`forwardDir`). O resultado será uma coordenada tridimensional representando a coordenada geográfica que foi georreferenciada.

3.3.1.4 Ponto de interesse

Os pontos de interesse são pontos que aparecem de acordo com a posição do `usuário` no mapa, exibindo informações relevantes a respeito da localização deste ponto, como por exemplo, salas de aula, blocos e restaurantes. Um ponto de interesse contém a sua localização através de uma coordenada geográfica, contém uma descrição informativa e a distância em metros que ele deve ser ativado de acordo com a posição do `usuário`. O Quadro 10 demonstra a criação dos pontos de interesse existentes no aplicativo.

Quadro 10 – Criação dos pontos de interesse

```
01.  public static void createDefaultsInterest() {
02.        string content = PlayerPrefs.GetString(INTEREST_KEY, "");
03.  if (content.Length == 0) {
04.        ArrayList interest = new ArrayList();
05.
06.        //CANTINA A 10 METROS
07.        interest.Add( new InterestPoint("Cantina", "Cantina do bloco
08.  T", new Coordinate(-26.905439f, -49.079955f), 10.0f) );
09.
10.        //SALA S-410
11.        interest.Add( new InterestPoint("S-410", "Sala S-410, bloco
12.  S", new Coordinate(-26.905768f, -49.079674f), 15.0f) );
13.
14.        //BLOCO R
15.        interest.Add( new InterestPoint("Bloco R", "Bloco R, campus
16.  1", new Coordinate(-26.905870f, -49.079992f), 20.0f) );
17.
18.        //Entrada bloco I
19.        interest.Add( new InterestPoint("Bloco I", "Entrada de bloco
20.  I", new Coordinate(-26.905669f, -49.078987f), 12.0f) );
21.
22.        //SALVA OS DADOS
23.        string contentData =
24.  ExtensibleMarkupSerial.serialize(interest, typeof(ArrayList));
25.        PlayerPrefs.SetString(INTEREST_KEY, contentData);
26.        PlayerPrefs.Save();
27.  }
28.  }
```

Os pontos de interesse são salvos nos dados embarcados ao aplicativo, chamado de `PlayerPrefs` (ver seção 2.4.6), estes dados são novamente carregados ao abrir o aplicativo.

O ponto de interesse é criado com seu título, descrição, coordenada geográfica e distância, ao abrir o aplicativo, os pontos são georreferenciados utilizando a mesma técnica de georreferenciamento do `usuário` (descrita na seção 3.3.1.3), e depois são salvos em outra estrutura na memória chamada `InterestPointREF`.

No Quadro 11 é demonstrado a estrutura presente na memória, que representa um ponto de interesse já georreferenciado. A estrutura `InterestPointREF` contém o ponto de interesse e a sua posição no mapa através de coordenadas tridimensionais (`pos`). Ao criar um `InterestPointREF` na memória, é calculado sua posição no mapa através do georreferenciamento `GEO.refGeoLogicalPosition`, sendo assim, esta estrutura já está preparada para exibir as informações do ponto de interesse no mapa.

Quadro 11 – Estrutura do ponto de interesse georreferenciado

```
01.  public class InterestPointREF {
02.        private InterestPoint point;
03.        private Vector3 pos;
04.        private GameObject point3D = null;
05.
06.        public InterestPointREF(InterestPoint point) {
07.              this.point = point;
08.              this.pos =
09.  GEO.refGeoLogicalPosition(this.point.coord);
10.        }
```

Já a classe `InterestPointHandler` é responsável por controlar a exibição dos pontos de interesse no mapa. Todos os pontos são avaliados a cada ciclo de um segundo, calculando a distância entre o `usuário` (*avatar* no mapa) e a posição tridimensional do ponto de interesse. No Quadro 12 é demonstrado o método `InterestPointREF.valid`, responsável por validar se o ponto de interesse deve ser exibido.

Quadro 12 – Método responsável por validar a exibição do ponto de interesse

```
01.  public bool valid() {
02.        if (MATHF_GEO.Dc(pos) > point.distance) {
03.              //clear
04.              if (point3D != null) {
05.                    MonoBehaviour.Destroy(point3D);
06.                    point3D = null;
07.              }
08.              return false;
09.        } else {
10.              if (point3D == null) {
11.                    //Instantiate
12.                    return true;
13.              }
14.              return false;
15.        }
16.  }
```

Primeiramente é verificado se a distância entre o ponto de interesse e o usuário é maior que a distância do ponto de interesse, se for maior, o ponto não será exibido, retornando

falso no método `valid` e destruindo o ponto de interesse caso ele exista no mapa `MonoBehavior.Destroy`. Caso a distância seja menor é retornado verdadeiro, indicando que o ponto de interesse deve ser criado no mapa. O método `valid` é chamado pela classe InterestPointHandler, que controla os pontos de interesse do aplicativo, conforme Quadro 13.

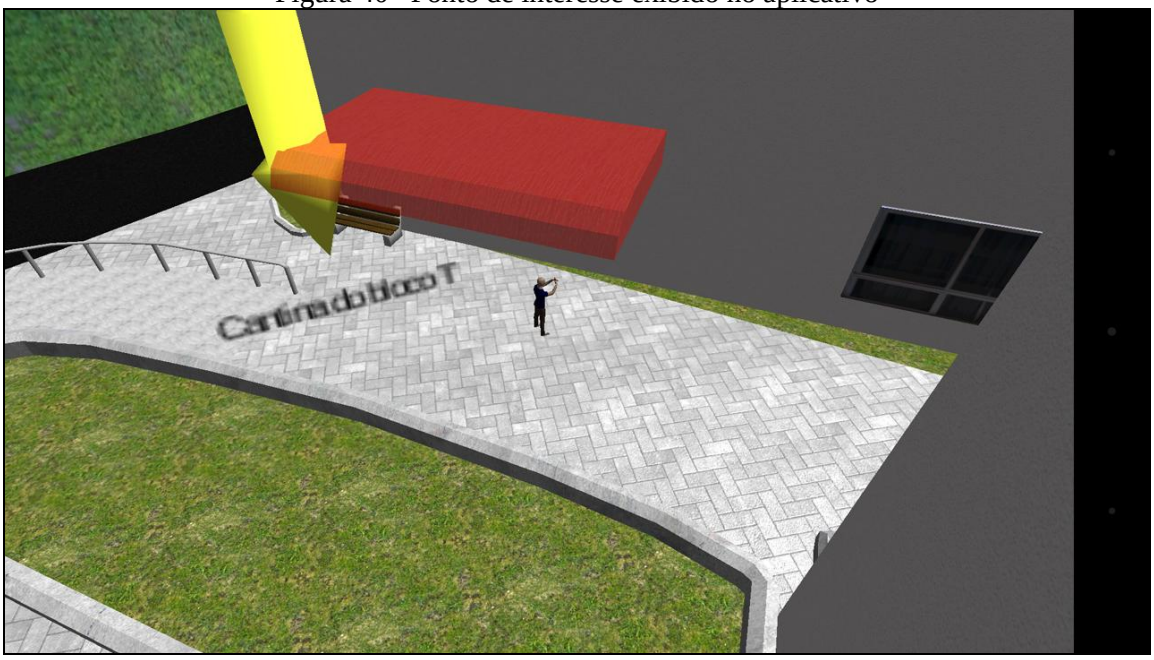
Quadro 13 – Validação dos pontos de interesse

```
01.  void Update () {
02.        if (actualTime >= clock) {
03.              valid();
04.              actualTime = 0.0f;
05.        } else {
06.              actualTime = actualTime + Time.deltaTime;
07.        }
08.  }
09.
10.  private void valid() {
11.        for (int i = 0; i < points.Count ; i++) {
12.              InterestPointREF refP = (InterestPointREF) points[i];
13.              if (refP.valid()) {
14.                    GameObject gb = (GameObject)Instantiate(anchor,
15.  refP.getPos(), Quaternion.Euler(0.0f, 0.0f, 0.0f));
16.
17.                    (gb.GetComponentsInChildren<TextMesh>())[0].text
18.  = refP.getPoint().description;
19.
20.                    refP.setGameObject(gb);
21.              }
22.        }
23.  }
```

O método `Update` que é chamado a cada *frame* desenhado, é responsável por efetuar a validação do ponto de interesse a cada um segundo (variável `clock` armazena o tempo de verificação), chamando o método `valid`. O método `valid` percorre todos os pontos de interesse do aplicativo, e para cada ponto chama o método `InterestPointREF.valid`, que verifica se deve ser exibido este ponto de interesse, o ponto de interesse será exibido na sua posição já georreferenciada no mapa, se e somente se o método `InterestPointREF.valid` retorne verdadeiro. Por fim, o ponto de interesse será removido do mapa quando o usuário estiver a uma distância maior que a distância do ponto de interesse, sendo que esta validação também é realizada pelo método `InterestPointREF.valid` já mencionado acima.

A Figura 40 demonstra o ponto de interesse Cantina do bloco T sendo exibido no aplicativo. O ponto de interesse é composto por uma âncora em amarelo que fica subindo e descendo, enfatizando a presença de um ponto importante a ser observado no mapa. Além da âncora, é exibida também uma descrição do ponto de interesse, neste caso a descrição `cantina do bloco T`. A descrição sempre é voltada para a tela do aplicativo, permitindo a sua visualização pelo usuário.

Figura 40– Ponto de interesse exibido no aplicativo

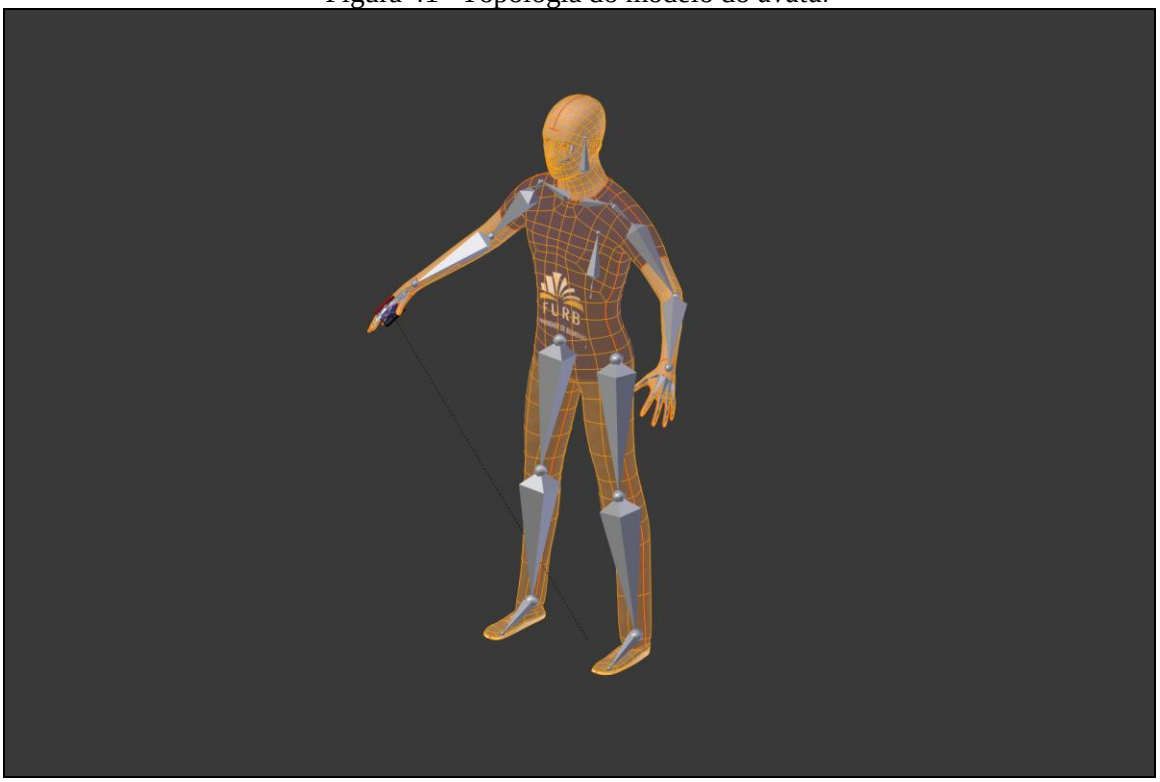


3.3.1.5 *Avatar* do usuário

Esta seção apresenta as etapas do desenvolvimento e características do *avatar* que representa o usuário no mapa virtual. O *avatar* é um modelo tridimensional orgânico, que imita a forma física e locomotora de um ser humano. Um modelo orgânico, pois, representa a forma de um ser vivo suas características e topologias únicas, como altura, forma da cabeça, olhos, mãos, nariz e outros. O modelo do *avatar* possui sua escala baseada em um ser humano, cuja altura, é de aproximadamente um metro e setenta e cinco centímetros.

A Figura 41 demonstra o modelo e suas topologias, através de uma imagem retirada do software Blender 2.7.4. O modelo possui aproximadamente dois mil vértices e setenta e três ossos. Os ossos são utilizados para efetuar as transformações de membros e partes separadas do modelo, sendo que essas transformações são utilizadas para gerar as animações do modelo, que neste caso são duas, animação parado e caminhando. A animação parado, é executada quando o usuário não está se movimentando pelo mapa, sendo que esta executa ações como mexer os braços, cabeça e mãos. A animação caminhando é executada toda vez que o usuário está se movimentando pelo mapa. Este movimento de caminhada é realizado em uma velocidade média de um metro e meio.

Figura 41– Topologia do modelo do *avatar*



O Quadro 14 demonstra os métodos `walkInterpolRef, walk e stp`. O método `walkInterpolRef` é responsável por executar o movimento de caminhada do *avatar* quando uma nova coordenada for processada pela etapa de georreferenciamento (ver seção 3.3.1.3). Para realizar o movimento de caminhada o método `walkInterpolRef` irá mover o *avatar* até a nova posição tridimensional a uma velocidade constante de dois metros por segundo, realizando para isto uma translação em duas unidades por segundo na direção da nova posição. Enquanto o *avatar* é movimentado a animação `caminhada` será executada através do método `walk`. Quando o *avatar* chegar à nova posição, a animação `caminhada` será pausada através do método `stp`.

Quadro 14 – Métodos responsáveis pela movimentação do *avatar*

```
01.  private void walkInterpolRef() {
02.        if (this.amountWalk < 1.0f) {
03.              this.amountWalk = this.amountWalk +
04.  (this.amountWalkInterpol * Time.deltaTime);
05.              walk ();
06.        } else {
07.              stp();
08.        }
09.        this.transform.position = Vector3.Slerp(this.fromV
10.                                                , this.toPosV
11.                                                , amountWalk);
12.        this.transform.position = new Vector3
13.  (this.transform.position.x, yGambiBefore,
14.  this.transform.position.z);
15.
16.        this.transform.rotation =
17.  Quaternion.Lerp(this.transform.rotation,
18.  Quaternion.LookRotation(toPosV - fromV), rotationVelocity *
19.  Time.deltaTime);
20.  }
21.
22.  private void stp() {
23.        if(this.gameObject.animation.IsPlaying("walk")) {
24.              this.gameObject.animation.Stop("walk");
25.        }
26.  }
27.
28.  private void walk() {
29.        if(this.gameObject.animation.IsPlaying("idle")) {
30.              this.gameObject.animation.Stop("idle");
31.        }
32.        this.gameObject.animation.Play("walk");
33.  }
```

Quando o *avatar* não está se movimentando a animação `parado` é executada. Na animação `parado`, são executados movimentos com a cabeça do *avatar* indo de um lado ao outro e movimentos simulando o uso do dispositivo móvel. O Quadro 15 demonstra o método `idle` que é responsável por executar a animação `parado` quando o *avatar* não estiver mais em movimento. Para isso será verificado se a animação `caminhada` está sendo executada (representada pela função `this.gameObject.animation.IsPlaying("walk"))`, caso não esteja em execução será executada a animação `parado` (representada pela função `this.gameObject.animation.Play("idle")`).

Quadro 15 – Execução da animação `parado`

```
01.  private void idle() {
02.        if(!this.gameObject.animation.IsPlaying("walk")) {
03.              this.gameObject.animation.Play("idle");
04.        }
05.  }
```

A Figura 42 demonstra uma imagem do modelo utilizado no *avatar* do aplicativo. Esta imagem já possui as texturas utilizadas no modelo, que foram criadas utilizando o software

Photoshop CS6. Na mão direita do modelo existe um modelo que representa um dispositivo móvel, simulando o usuário real com seu dispositivo.

Figura 42– Modelo utilizado no aplicativo

3.3.1.6 Orientação

Esta seção apresenta a etapa de desenvolvimento da orientação do aplicativo quando está utilizando o GPS como movimentação. A orientação consiste em posicionar a visão do aplicativo de acordo com a visão do usuário, baseado no sensor da bússola.

O Quadro 16 demonstra o método `update` da classe `RotationCompass`, este método é responsável por atualizar a rotação da câmera de acordo com os valores retornados pelo sensor da bússola, fazendo com que o aplicativo sempre possua sua orientação de acordo com a orientação do usuário e seu dispositivo. O sensor da bússola representado pela interface `Input.compass` retorna a diferença de ângulo em graus entre o dispositivo e polo norte geográfico, através do método `Input.compass.magneticHeading`.

Quadro 16 – Orientação baseada nos valores da bússola

```
01.  void Update () {
02.        if (CONFIG_VALUES.canCompass) {
03.              float compassRot = MATHF_GEO.normalizeEulerDegress(
04.  Input.compass.magneticHeading );
05.              Quaternion to = Quaternion.Euler(0.0f
06.                                               , compassRot
07.                                               , 0.0f);
08.              this.transform.rotation =
09.  Quaternion.Slerp(this.transform.rotation, to, Time.deltaTime);
10.
11.        }
12.  }
```

A Figura 43 demonstra o dispositivo com a orientação de tela utilizada no aplicativo, deitado e com sua parte superior voltada para o lado esquerdo. Neste caso o sensor da bússola retorna que o dispositivo está totalmente voltado ao polo norte geográfico, retornando zero graus de diferença entre a orientação do dispositivo e o polo norte, representado na imagem pela flecha vermelha, que indica o ponto de partida para a orientação do dispositivo.

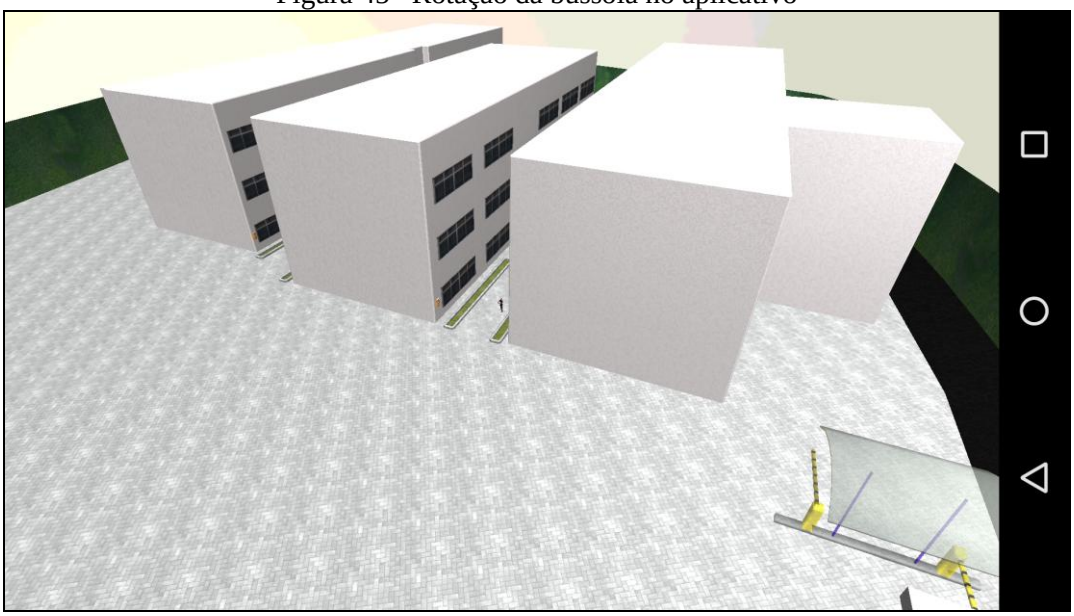
Figura 43– Rotação da bússola do dispositivo

A Figura 44 demonstra o dispositivo com outra orientação, inclinado para o lado esquerdo. Neste caso a diferença entre a orientação do dispositivo e o polo norte geográfico é de noventa graus, retornados pelo sensor da bússola.

Figura 44– Rotação da bússola do dispositivo com outra orientação

Na Figura 45 é demonstrada a aplicação dos valores de ângulos em graus retornados pela bússola na câmera do aplicativo. A câmera representa toda a visualização da aplicação, simulando a visualização do usuário no mundo real para o aplicativo.

Figura 45– Rotação da bússola no aplicativo



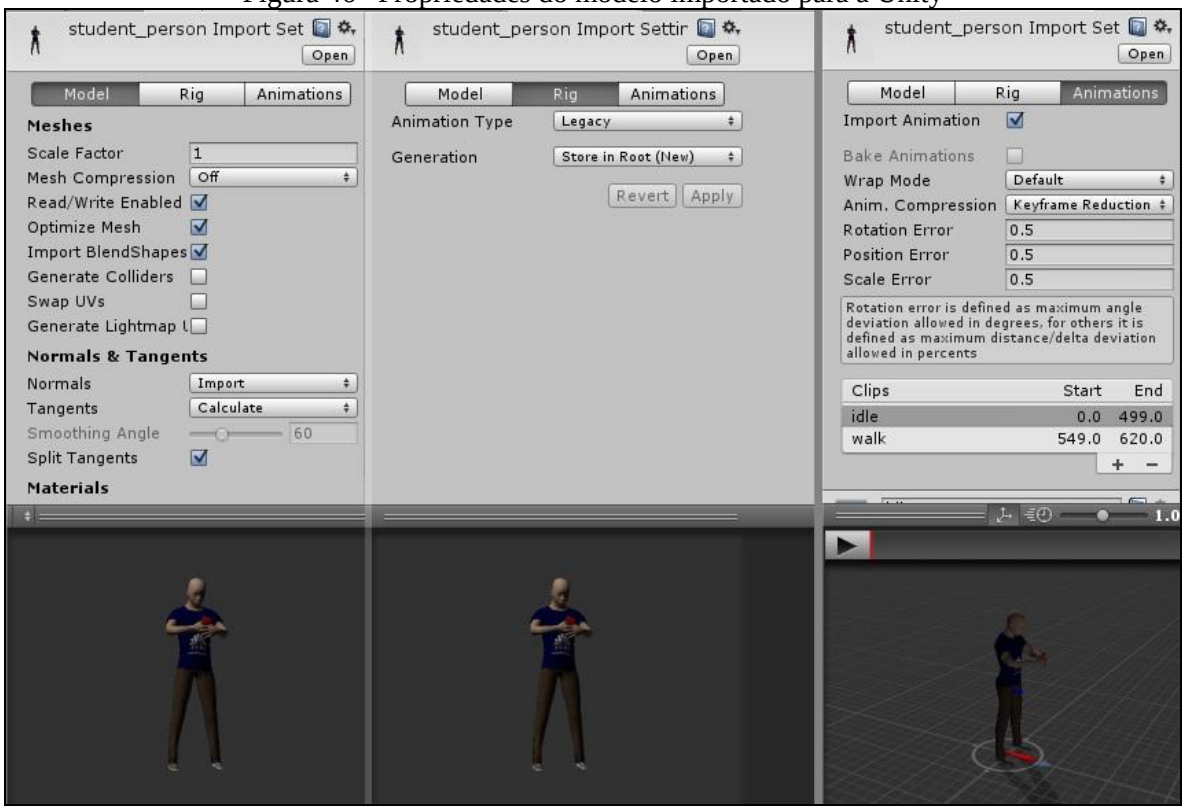
3.3.1.7 Integração Blender e Unity

Esta seção apresenta a etapa de importação dos modelos tridimensionais desenvolvidos no software Blender 2.7.4 para o *framework* Unity 3D, demonstrando todas as características e informações importadas e como configurá-las para o uso no aplicativo.

Os modelos tridimensionais desenvolvidos no Blender são exportados individualmente utilizando o formato padrão `blend`. Segundo Blender (2015) o formato `blend`, é um formato de arquivo nativo do software Blender. Este formato armazena objetos 3D, cenas inteiras, texto, imagens, vídeos, sons e configurações do Blender. Possuindo suporte a compressão, encriptação e multiplataforma.

Ao importar o modelo para o *framework* Unity 3D é gerado um novo *asset* (ver seção 2.4.3). A Figura 46 demonstra o modelo do *avatar* importado para dentro da Unity e suas propriedades de modelo, esqueleto (estrutura óssea) e animação. Nas propriedades do modelo representado por `Model` podem ser configurados tipos de compressão, escala do modelo, coeficientes para iluminação e materiais utilizados. Nas propriedades de esqueleto representado por `Rig` é possível escolher o método utilizado para animação esquelética, neste caso `Legacy` que constitui uma estrutura óssea com hierarquias, e também a opção para escolher o osso raiz representada por `Generation`. Por fim nas propriedades de animação representado por `Animations` são configurados o modo de execução das animações e também quais as animações existentes para o modelo e seu nome. Ainda nas propriedades de animação, as animações do modelo ficam todas juntas em um intervalo de *frames*, estes *frames* possuem as transformações que fazem o movimento dos ossos do modelo. Para criar uma nova animação deve ser selecionado o seu início e fim e dar a um nome a esta. Neste caso o modelo do *avatar* possui as animações `idle` (parado) que inicia no *frame* 0 a 499 e `walk` (caminhando) que inicia no *frame* 549 a 620.

Figura 46– Propriedades do modelo importado para a Unity



3.3.2 Operacionalidade da implementação

Para a utilização do aplicativo, o usuário deve optar entre usar o GPS ou a tela do dispositivo para as movimentações do *avatar*. Para utilizar o aplicativo com GPS, basta estar com o GPS do dispositivo ativado, caso contrário o aplicativo irá utilizar o *touch* da tela para as movimentações. A troca do uso do GPS ou *touch* da tela pode ser feita a qualquer momento, basta ativar ou desativar o GPS do dispositivo no decorrer do uso do aplicativo.

A Figura 47 demonstra o aplicativo em execução com o GPS desativado, neste caso será habilitada a movimentação pelo *touch* da tela, conforme exibido em vermelho na mensagem da tela do aplicativo. Ao habilitar o GPS, o aplicativo irá desabilitar a movimentação pelo *touch* e se movimentar baseado nas coordenadas do GPS.

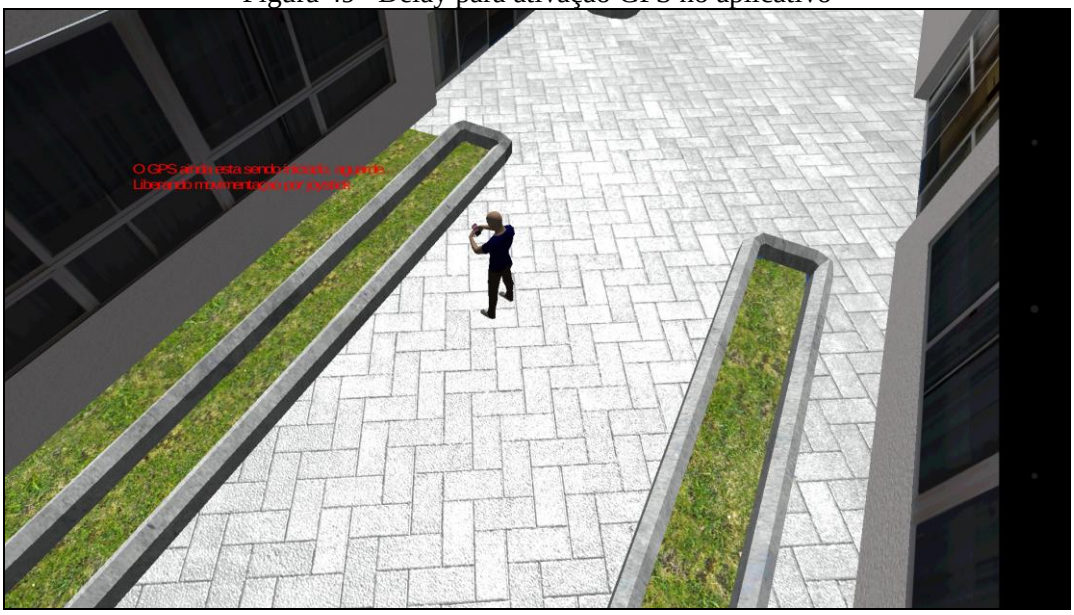
Figura 47– Aplicativo com GPS desativado

A Figura 48 demonstra o aplicativo em execução com o GPS ativado. Ao habilitar o GPS, ainda pode ocorrer um *delay* (tempo de espera) até o aplicativo desabilitar a movimentação pelo *touch* e habilitar a movimentação pelo GPS. Este *delay* pode ocorrer, pois, o hardware do GPS do dispositivo ainda não enviou as coordenadas para o aplicativo, ou ainda está sendo iniciado. Caso ocorra algum erro ao obter as coordenadas pelo GPS, o aplicativo irá habilitar o *touch*.

Figura 48– Aplicativo com GPS ativado

A Figura 49 demonstra o aplicativo em execução com o GPS sendo iniciado. Neste momento a movimentação por *touch* ainda está ativada. Isto pode ocorrer por causa do *delay* explicado anteriormente.

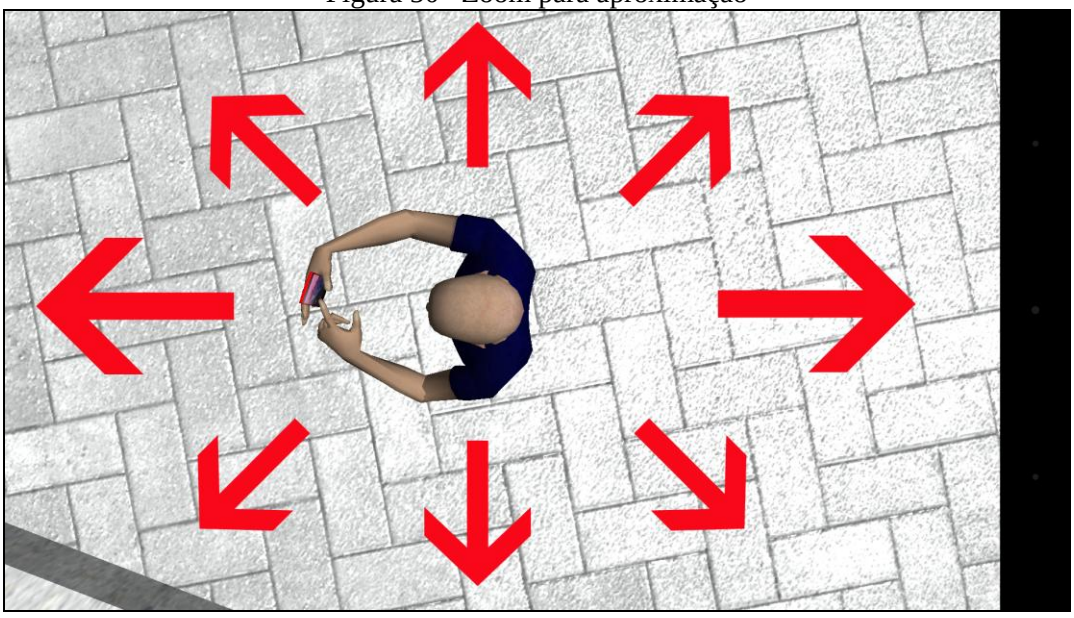
Figura 49– Delay para ativação GPS no aplicativo



No decorrer da execução do aplicativo, o usuário pode efetuar *zoom* para se aproximar do *avatar* ou se distanciar. Para efetuar o *zoom*, o usuário deve utilizar dois dedos na tela do dispositivo, ao aproximar os dois dedos o usuário está efetuando um *zoom* para se distanciar do *avatar*. Já ao distanciar os dois dedos o usuário está efetuando um zoom para se aproximar do *avatar*.

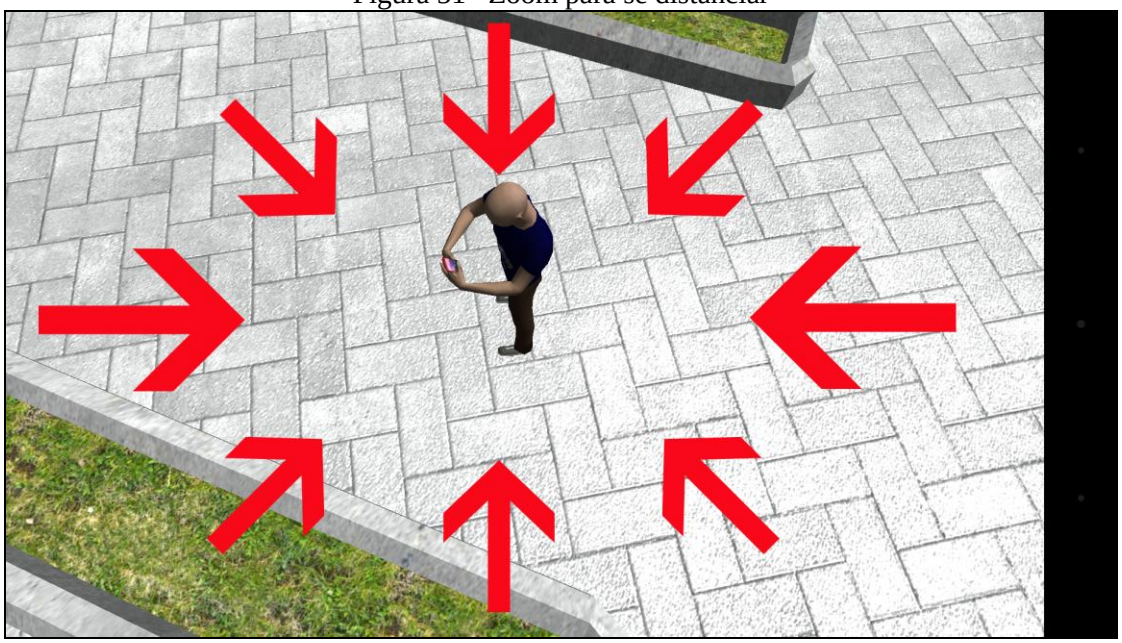
A Figura 50 demonstra o *zoom* para aproximação do *avatar*. Neste caso o usuário deve distanciar os dedos em qualquer direção na tela do dispositivo, sem soltar os dedos. As flechas vermelhas demonstram a possibilidade de distanciar os dedos em qualquer direção.

Figura 50– Zoom para aproximação



A Figura 51 demonstra o *zoom* para se distanciar do *avatar*. Neste caso o usuário deve aproximar os dedos em qualquer direção na tela do dispositivo, sem soltar os dedos. As flechas vermelhas demonstram a possibilidade de aproximar os dedos em qualquer direção.

Figura 51– Zoom para se distanciar



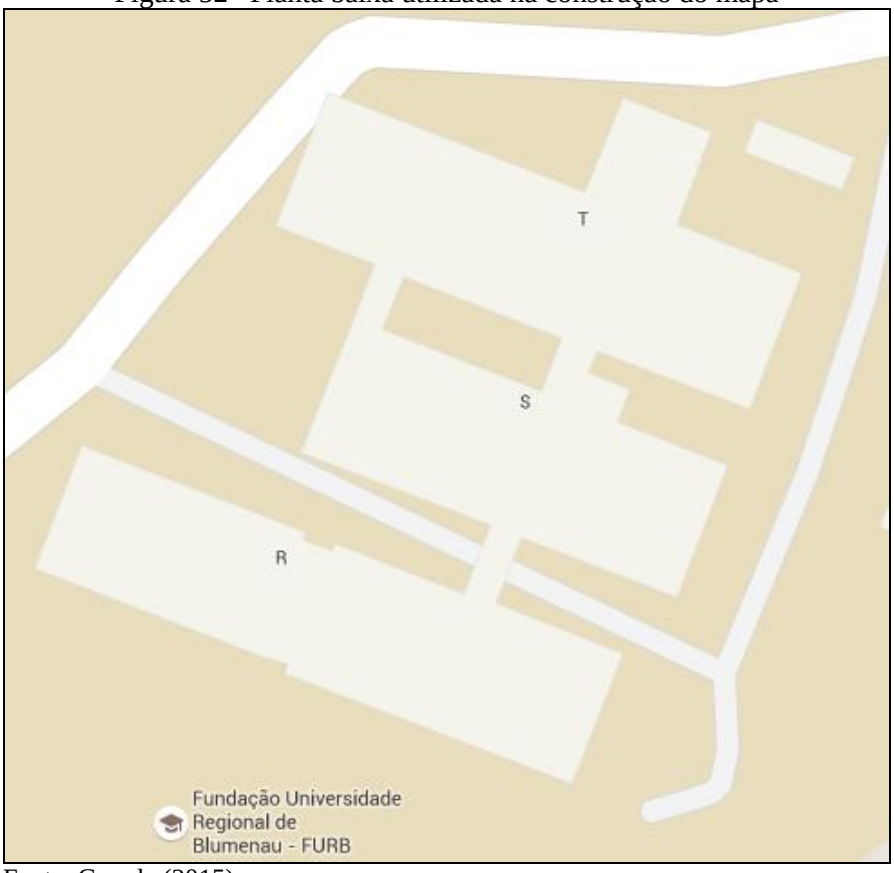
Mais detalhes do uso do aplicativo pode ser visto no vídeo Pitz (2015).

3.4 RESULTADOS E DISCUSSÕES

O presente trabalho teve como objetivo inicial a construção de um aplicativo para a plataforma Android que possui um mapa tridimensional de uma região da FURB, e através dos sensores GPS, bússola e acelerômetro permita geolocalizar o usuário e exibir informações relevantes de acordo com sua localização. Para isto, inicialmente foi necessário optar por uma região da FURB e iniciar o levantamento das informações necessárias para a construção do mapa. Foi decidido então utilizar os blocos R, S e T do câmpus um da FURB, e utilizar uma planta baixa obtida através do aplicativo Google Maps.

A planta baixa apresentada na Figura 52 contém todas as informações necessárias para a modelagem tridimensional do mapa no software Blender 2.7.4. Foi extraída desta planta baixa a localização do ponto pivô utilizado na geolocalização do usuário, medidas em metros dos blocos e construções, e orientação do mapa.

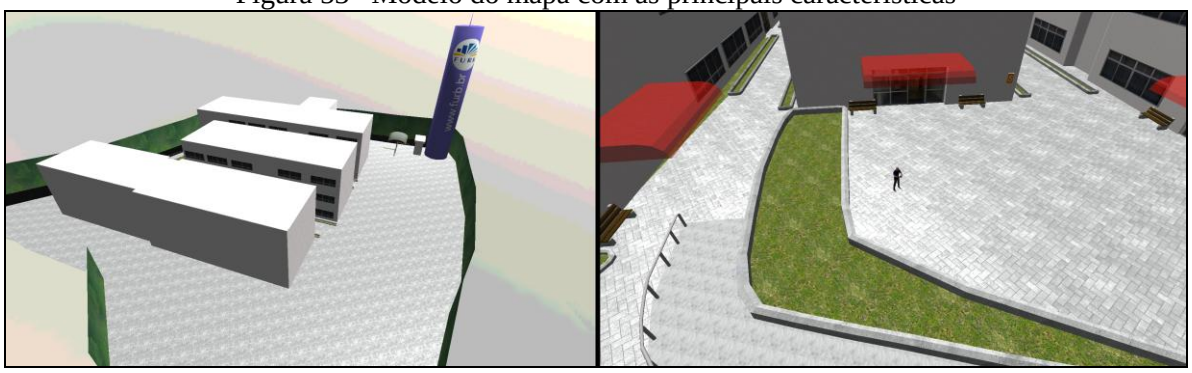
Figura 52– Planta baixa utilizada na construção do mapa



Fonte: Google (2015).

Os outros detalhes dos blocos R, S e T foram obtidos através de uma pesquisa em campo, onde foram analisados os principais pontos que deveriam ser modelados para representar com mais realismo esta região da FURB. Nesta pesquisa foram capturadas imagens da região dos blocos R, S e T. Na Figura 53 é demonstrado o mapa modelado com os principais pontos dos blocos. Foram escolhidos os detalhes que mais servem de referência visual, como a torre azul em frente ao bloco I, placas com os nomes dos blocos, cancela com guarita na entrada dos blocos, bancos, toldos nas portas de entrada e o estacionamento atrás do bloco S.

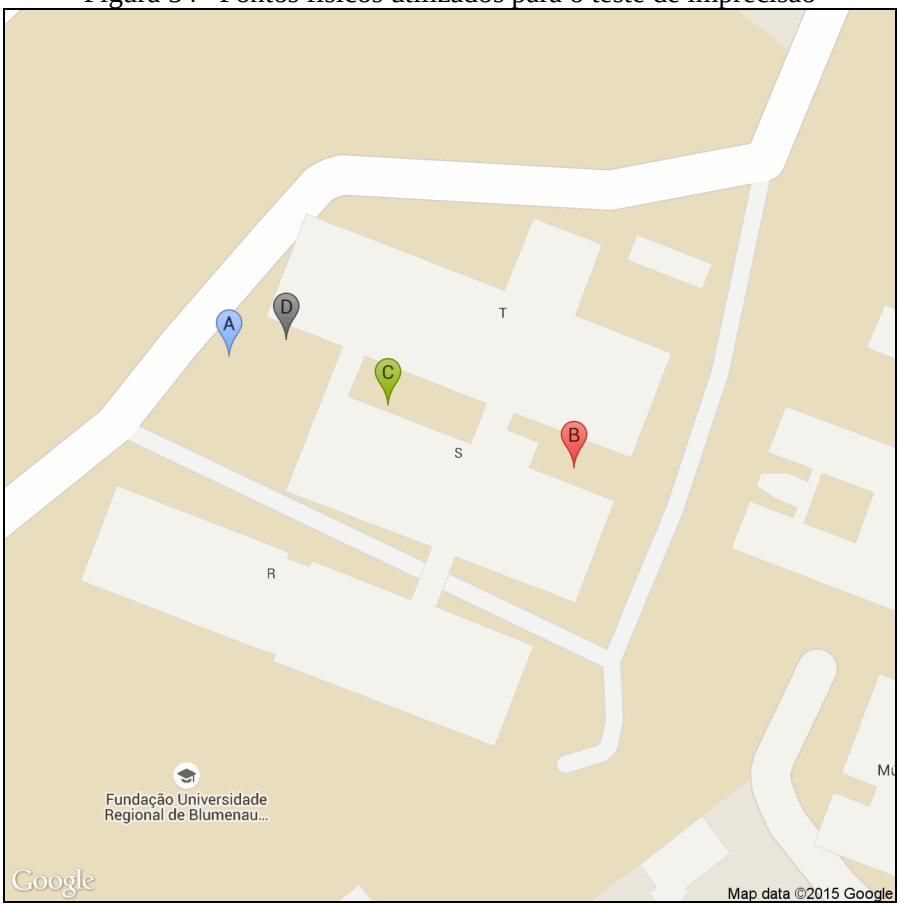
Figura 53– Modelo do mapa com as principais características



Inicialmente havia cogitado a utilização do sensor acelerômetro para auxiliar na geolocalização do usuário no mapa, porém, não houve a necessidade do uso deste sensor no trabalho, sendo utilizado somente os sensores bússola e GPS. Também foi realizado o estudo da API do Google Maps para auxiliar nas técnicas de geolocalização, porém, devido a limitações no arquivo de configurações `manifest.xml` em conjunto com a Unity, foi descartado o seu uso. Ainda inicialmente foi descrito como principais requisitos do trabalho o uso do *framework* Unity 4.6 para a construção do aplicativo, compatibilidade do aplicativo com o sistema operacional Android 4.0 ou superior e construção dos modelos tridimensionais no software Blender 2.7.4, ambos os requisitos foram atendidos no decorrer do desenvolvimento.

Durante a fase de desenvolvimento do trabalho, na etapa de georreferenciamento do usuário, encontrou-se a necessidade de descobrir a medida entre duas coordenadas geográficas (latitude e longitude) para saber o quanto o *avatar* deve caminhar no mapa de acordo com as novas coordenadas obtidas. Para isso, foi utilizada a fórmula de Haversine, que permite descobrir a distância em metros de duas coordenadas. Na fase de geolocalização foi utilizado o sensor GPS do dispositivo para obter as coordenadas geográficas e utilizá-las como localização do usuário no mapa tridimensional. Porém, ao iniciar a implementação foram detectadas inconsistências nos valores retornados pelo GPS, como, imprecisão na localização e variação de valores mesmo não estando em movimento. A imprecisão consiste em retornar valores de coordenadas que não representam a posição atual do usuário, já a variação consiste em retornar valores de coordenadas diferentes, em um período curto de tempo, mesmo estando no mesmo local fisicamente. Um fato que auxilia na imprecisão das coordenadas retornadas pelo GPS, é o uso de A-GPS (*Assisted Global Positioning System*) pelos dispositivos móveis. O A-GPS consiste em utilizar o sensor do GPS em conjunto com sinal da rede Wifi, com o objetivo de melhorar a precisão da coordenada retornada pelo GPS. Neste caso como o A-GPS depende da Wifi, quando ocorre perda no sinal desta rede somente o sensor do GPS passa a atuar, assim as coordenadas retornadas passam a ficar com taxas mais lentas de atualização e maior perda na precisão. Quanto aos valores imprecisos retornados pelo GPS, foi efetuado um teste no ambiente utilizado para desenvolvimento (blocos R, S e T da FURB), a fim de verificar os valores imprecisos.

Figura 54– Pontos físicos utilizados para o teste de imprecisão



Fonte: Google (2015).

Para o teste de imprecisão foram escolhidos os locais físicos A, B, C e D, demonstrados na Figura 54. Para obter os valores, foi posicionado o dispositivo com GPS nos locais escolhidos acima e iniciada a leitura de trinta coordenadas com intervalo de dois segundos entre cada uma. Após o término da leitura, foi retirada a média aritmética das trinta coordenadas (latitude e longitude) obtidas, e por fim foi utilizada a fórmula de Haversine para retirar a distância em metros entre o valor real dos locais A, B, C e D e os valores obtidos através das médias. Os resultados são demonstrados na Tabela 1.

Tabela 1 – Comparação de coordenadas geográficas

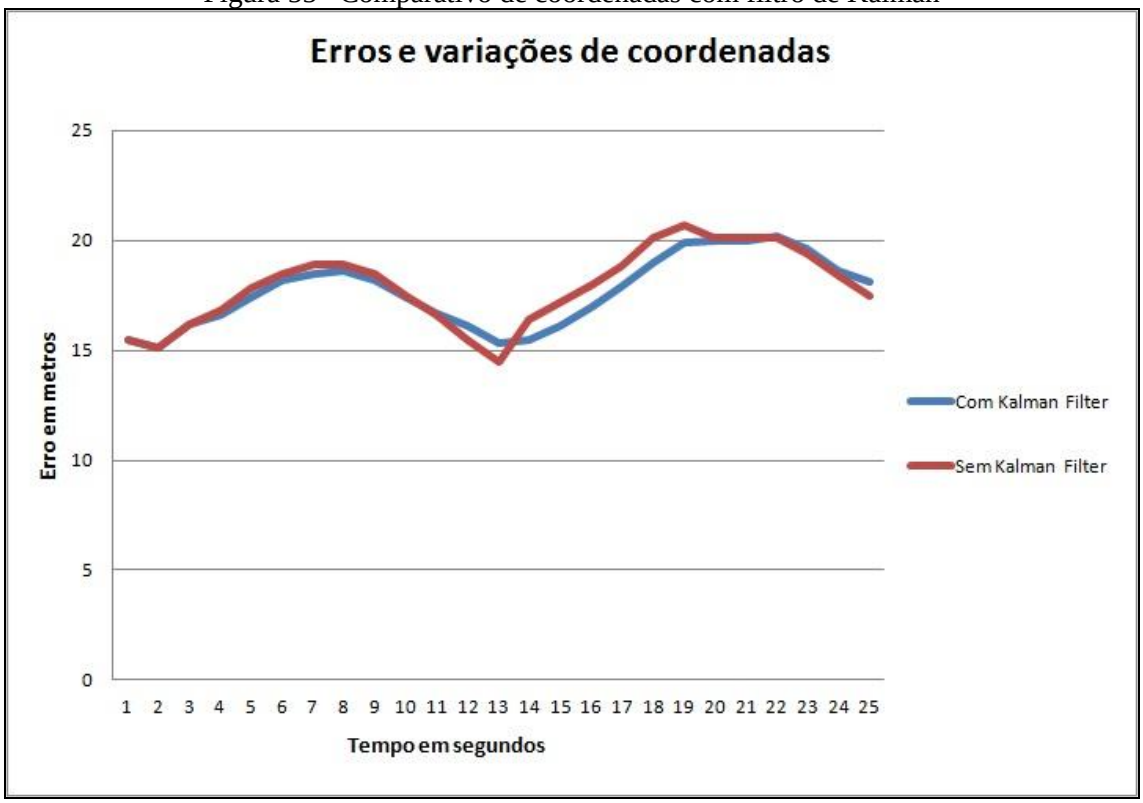
Local	Coordenada real	Coordenada obtida	Diferença em metros (real X obtido)
A	-26.905494, -49.080062	-26.905591, -49.080071	10,8
B	-26.905686, -49.079399	-26.905713, -49.079287	11,5
C	-26.905578, -49.079757	-26.905602, -49.079662	9,4
D	-26.905465, -49.079953	-26.905471, -49.079934	2

Além dos valores imprecisos retornados pelo GPS, foram encontrados valores com variação. Conforme já mencionado anteriormente, os valores com variação representam valores de coordenadas diferentes retornados para um mesmo local físico. Para tratar estes

valores com variação, foi implementado um filtro chamado Filtro de Kalman. Este filtro auxilia na correção de valores fora do padrão, evitando grandes variações, como possíveis erros. Para o teste da aplicação do filtro nas coordenadas de GPS, foi utilizada uma coordenada fixa como sendo uma origem, e posicionado o dispositivo com GPS a uma distância de dez metros desta coordenada. Após isto, foi iniciada a leitura de vinte e cinco coordenadas em um intervalo de um segundo para cada coordenada.

A Figura 55 demonstra o resultado da aplicação do filtro de Kalman nos testes de valores imprecisos. Em vermelho são representados os valores de distância da coordenada gerada pelo filtro de Kalman até a coordenada de origem, e em azul são representados os valores de distância das coordenadas sem o filtro de Kalman até a coordenada de origem. É possível notar que os valores gerados pelo filtro admitem variações menores e mais suaves, ao contrário do que ocorrer quando não utilizado o filtro. Esta correção nos valores através do filtro de Kalman permite a movimentação mais precisa do *avatar* no aplicativo, e evita grandes erros de movimentação e variações.

Figura 55– Comparativo de coordenadas com filtro de Kalman



A Tabela 2 demonstra o resultado de testes de erro e taxa de atualização das coordenadas obtidas pelo sensor GPS. Os testes foram realizados levando em conta o ambiente onde o usuário está localizado, ambiente aberto, com obstáculos (blocos, árvores, entre outros) e ambiente fechado (dentro dos blocos). Os valores de coordenadas foram

obtidos durante um intervalo de trinta segundos se deslocando no ambiente. O menor e maior tempo para atualização da coordenada do GPS é descrito em segundos. O erro médio é descrito em metros, sendo obtido no decorrer dos trinta segundos do teste, obtendo os valores de distância através da fórmula de Haversine.

Tabela 2 – Teste de ambiente

Ambiente	Menor tempo para atualização da coordenada	Maior tempo para atualização da coordenada	Erro médio
aberto	1	5	7
fechado	12	30	34
com obstáculos	5	23	17

Para o teste de performance do aplicativo foram utilizados três dispositivos com versões do sistema operacional Android diferentes e configurações de hardware diferentes. Os dispositivos utilizados foram o *smartphone* Motorola Moto X XT1097, smartphone LG Nexus 5 D821 e o *tablet* Samsung Galaxy Note 8 GT N5110. Para obter os resultados do teste foi iniciado o aplicativo em ambos os dispositivos com o GPS e Wifi ligados. Foi gerada uma versão do aplicativo que mede os *Frames Per Second* (FPS), que são os quadros desenhados a cada segundo, possibilitando entender o desempenho geral do aplicativo. Os valores de consumo de memória e uso do processador foram obtidos através do aplicativo de monitoramento System Tuner (System Tuner, 2015). A Tabela 3 demonstra os resultados do teste de performance realizado com os dispositivos listados acima.

Tabela 3 – Teste de performance

Dispositivo	Memória utilizada em megabytes	Uso total do processador em %	Quantidade de quadros desenhados por segundo	Versão do Android
Moto X	46,52	2,87	60	5.1
Nexus 5	57	3,58	60	5.1.1
Note 8	51,68	8	61	4.2

A Tabela 4 demonstra as características tridimensionais do *avatar* e mapa virtual utilizados no aplicativo. Para obter as informações foi utilizado o software Blender na qual os modelos foram desenvolvidos. As informações listadas são a quantidade de vértices, quantidade de faces, quantidade de ossos, quantidade de animações e quantidade de objetos tridimensionais. Neste caso, o *avatar* é composto do modelo que representa o usuário e o modelo do dispositivo móvel presente em sua mão. O mapa é composto pelos modelos dos blocos R, S e T, bancos, detalhes da rua, janelas, torre e portaria com cancela.

Tabela 4 – Características tridimensionais do aplicativo

Modelo	Quantidade de vértices	Quantidade de faces	Quantidade de ossos	Quantidade de animações	Quantidade de objetos 3D
Avatar	1893	1852	73	2	2
Mapa	3543	3305	0	0	11

No decorrer da implementação notou-se que um mapa virtual tridimensional possui alguns pontos positivos e outros negativos em relação a um mapa virtual bidimensional comum. Um ponto positivo do mapa tridimensional é a maior imersão do usuário com um mapa mais rico em detalhes e características que se assemelham ao real, como por exemplo, o ponto de interesse que informa ao usuário detalhes de algum local no mapa baseado em sua localização atual. Outro ponto positivo referente a imersão do usuário no mapa, é a possibilidade de cruzar referências visuais do mapa virtual com o mundo real. Quanto aos pontos negativos, um mapa bidimensional se torna mais simples em relação a um tridimensional, tanto na construção quanto ao uso pelo usuário, sendo um mapa mais leve para ser processado e mais limpo visualmente. Para efetuar consultas rápidas, por exemplo, o mapa bidimensional se torna melhor, pois, irá carregar mais rápido e requer uma análise menor dos detalhes do mapa perante o usuário. Baseado nos pontos positivos e negativos é possível concluir que um mapa tridimensional possui a sua construção mais complexa, pois, requer um estudo maior da região a ser representa, podendo haver muitos detalhes que devem ser criados. O mapa tridimensional se tornou viável para o trabalho aqui descrito, pois, o objetivo do trabalho é disponibilizar ao usuário um ambiente virtual rico em detalhes e através de técnicas de geolocalização auxiliá-lo em sua locomoção.

3.5 COMPARATIVO DOS TRABALHOS CORRELATOS

Esta seção é responsável por realizar a comparação dos trabalhos correlatos descritos na seção anterior e este trabalho. O Quadro 17 apresenta a relação de existência de características nos trabalhos apresentados. O presente trabalho está identificado como Pitz (2015).

Quadro 17 – Comparativo dos trabalhos correlatos e este trabalho

Características dos trabalhos relacionados	Lima, Mendes Neto e Rodrigues (2012)	Harvard (2014)	Kestring (2014)	Baião e Lima (2014)	Pitz (2015)
geolocalização	Sim	Sim	Sim	Não	Sim
framework Unity3D	Não	Não	Não	Sim	Sim
uso de mapas	Sim	Sim	Sim	Não	Sim
plataformas	Web	Web, IOS e Android	Android	Mac e Windows	Android
tridimensional	Sim	Não	Não	Sim	Sim
uso de sensores	Sim, GPS	Sim, GPS	Sim, GPS	Sim, acelerômetro e giroscópio	Sim, GPS e bússola
aplicação/jogo sério	Sim, jogo interativo para auxílio na aprendizagem de alunos	Sim, mapa interativo para auxílio de alunos da universidade de Harvard	Sim, mapa interativo para auxílio de alunos da universidade FURB	Não	Sim, mapa virtual para auxílio na localização e obtenção de informações acerca de regiões da universidade FURB

Como pode ser visto no Quadro 17, os trabalhos possuem características distintas, onde nem todos possuem sua construção tridimensional ou compartilham as mesmas plataformas, porém, todos os trabalhos utilizam algum sensor. Os trabalhos que utilizam o sensor GPS, também utilizam mapas e realizam a geolocalização do usuário dentro desse mapa. No trabalho de Harvard (2014) os usuários, que na sua maioria são alunos, navegam pelo mapa da Universidade de Harvard e através de suas posições obtém informações voltadas aos locais onde estão inseridos. O trabalho de Baião e Lima (2014) é o único trabalho que não utiliza o sensor GPS e técnicas de geolocalização, porém, a sua construção tem grandes semelhanças com este trabalho, pois, através do uso do *framework* Unity 3D construiu um aplicativo/jogo tridimensional que simula movimentos do usuário em um mundo virtual. Exceto o trabalho de Baião e Lima (2014), os outros trabalhos e este são aplicativos/jogos sérios, onde o objetivo principal é auxiliar o usuário de alguma forma na vida real, como o trabalho de Lima, Mendes Neto e Rodrigues (2012), que se trata de um jogo sério para auxílio na aprendizagem de alunos. O trabalho de Kestring (2014) é o trabalho que mais se assemelha a este trabalho, onde, o objetivo do aplicativo criado é auxiliar alunos da FURB a se locomoverem pela universidade, utilizando a sua posição atual através do sensor GPS e geolocalizando em um mapa, onde é possível obter informações através de pontos de

interesses cadastrados no mapa. Porém, o trabalho de Kestring (2014) não foi desenvolvido no *framework* Unity3D, não utiliza mapas tridimensionais, não utiliza *avatares* que representam o usuário, não utiliza o sensor bússola para obter a orientação do usuário em relação ao mapa, não utiliza cálculos de distância e georreferenciamento para um plano tridimensional.

4 CONCLUSÕES

O presente trabalho apresenta a criação de um aplicativo para a plataforma Android que possui um mapa virtual tridimensional da universidade FURB. Os principais objetivos do foram criar o mapa dos blocos R, S e T da FURB e utilizar os sensores GPS e bússola para auxiliar na geolocalização do usuário no mapa. Para alcançar os objetivos do trabalho foi utilizado o *framework* Unity 3D para a construção da interface gráfica do mapa, onde constam as animações, os modelos tridimensionais, cálculos para georreferenciamento, acesso aos sensores e outros recursos necessários para o desenvolvimento. Para a construção dos modelos tridimensionais foi utilizado o software Blender.

O estudo na qual se baseou a construção do trabalho foi a necessidade de auxiliar pessoas na locomoção dentro do ambiente da FURB, disponibilizando pontos de interesse que contém informações do local demarcado e utilizando a localização física para se posicionar no mapa. Para isto, o aplicativo desenvolvido permite o uso por qualquer pessoa, e esta pessoa pode utilizá-lo estando presente na FURB, utilizando o GPS do seu dispositivo, ou não estando presente, onde se pode navegar pelo mapa livremente.

Quanto a implementação, foi encontrado dificuldade no uso dos valores retornados pelo sensor do GPS, onde os valores retornados continham erros de precisão, retornando coordenadas divergentes quanto a posição real, e ainda valores com grande variação em pequenos períodos de tempo. Para tratar estes erros foi necessário implementar um filtro de ruídos, o filtro de Kalman, que inibe a grande variação e permite predizer valores mais reais e lineares. Outro problema encontrado com o uso do GPS foram os obstáculos como prédios, árvores e a parte interna dos blocos (ambiente fechado) que atrapalham o sinal do sensor e diminuem sua precisão e sua taxa de atualização. Nos testes realizados no aplicativo com GPS, em locais abertos, onde não existem obstáculos em um raio de vinte metros, o GPS possuía uma taxa de atualização por segundo e com um erro de cinco a dez metros. Já em locais com obstáculos e mais fechados a taxa de atualização pode ter uma lentidão de até quinze segundos, e o erro pode chegar a trinta metros.

A implementação do georreferenciamento demonstrou-se um pouco complexa quanto ao uso de matemáticas trigonométricas, para transformar valores de coordenadas geográficas em pontos tridimensionais. Para isso, foi necessário construir todos os modelos tridimensionais utilizando uma unidade de media comum, neste caso o metro. Para os cálculos foi necessário a utilização da fórmula de Haversine, para a obtenção do distanciamento entre

duas coordenadas geográficas, e o uso do teorema de Pitágoras, para obtenção da orientação entre coordenadas geográficas.

Por fim, o trabalho se mostrou de grande valia, pois, utiliza diversas tecnologias, ferramentas e técnicas em conjunto, para obter um melhor resultado. A região dos blocos R, S e T escolhido para o desenvolvimento do mapa foi ideal, pois, possuía uma grande variedade de ambientes para testes do aplicativo, ambientes abertos, com obstáculos e variações no sinal da WiFi e GPS. Auxiliando assim na pesquisa e aperfeiçoamento das técnicas utilizadas. Apesar de algumas dificuldades e limitações, foi possível implementar todos os objetivos propostos inicialmente.

4.1 EXTENSÕES

Como sugestão de extensões para a continuidade do presente trabalho tem-se:

a) expandir o mapa construindo outras regiões da FURB;
b) realizar a construção de mapas internos, para visualizar e acessar salas de aulas e blocos;
c) realizar integração com os serviços da FURB para obter informações como horários de aula, salas e eventos;
d) criar um ambiente virtual, na qual os usuários do mapa possam se visualizar e trocar informações;
e) permitir o cadastro de pontos de interesses, ou obter os mesmo através de alguma integração com outros sistemas da FURB;
f) buscar hardwares que possam melhorar e/ou auxiliar o sinal do GPS.

REFERÊNCIAS

AIRES, Fabio J. R.; HAHN, Eliza C. Um estudo da API de geolocalização do HTML5: como desenvolver aplicativos de geolocalização para internet. **A Revista Eletrônica da Faculdade de Ciências Exatas da Terra Produção/Construção e Tecnologia**, Dourados, v. 3, n. 4, 2014. Disponível em:<http://www.unigran.br/ciencias_exatas/conteudo/ed4/artigos/06.pdf>. Acesso em: 27 fev 2015.

AQUINO, André Teixeira de.; ARAÚJO, André Luiz Carneiro de. Aplicação do filtro de Kalman a um sistema de posicionamento de veículo aquático. In: CONGRESSO NORTE-NORDESTE DE PESQUISA E INOVAÇÃO. 17., 2010, Maceió. **Anais eletrônicos...** Maceió: SETEC, 2010. Disponível em:<http://connepi.ifal.edu.br/ocs/index.php/connepi/CONNEPI2010/paper/viewFile/1607/870>. Acesso em: 21 maio 2015.

BAIÃO, Alexandre Henrique Aben-Athar Sousa.; LIMA, Diego Bichara de. **Combinando Kinect e smartphones para rastreamento do movimento do pulso**. 2014. p 58. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) – Departamento de Ciência da Computação, Universidade de Brasília, Brasília.

BAKKER, Paul. **GPS/dead reckoning navigation with Kalman filter integration**. 2012. p 25. Disponível em:<https://www.calvin.edu>. Acesso em: 03 maio 2015.

BALDI, Luiz Marcelo. **Desenvolvimento de uma aplicação para localização de fast food especializado em cachorros quentes**. 2011. 52 p. Monografia de Especialização (Especialização em Tecnologia Java) - Universidade Tecnológica Federal do Paraná, Paraná.

BLENDER ORG. **Blender**. 2015. Disponível em: <http://www.blender.org/>. Acesso em 04 jun. 2015.

CARVALHO, Diego Eduardo. **Desenvolvimento do jogo Robogol para navegadores web utilizando o motor de jogos Unity 3D**. 2012. 32 p. Trabalho de Conclusão de Curso (Bacharelado em Ciências de Computação) – Instituto de Ciências Matemáticas e Computação, Universidade de São Paulo, São Paulo.

CARVALHO, Samuel de Oliveira. **Aplicação do filtro de Kalman a um sistema de posicionamento de veículo aquático**. 2013. 79 p. Trabalho de Conclusão de Curso (Bacharelado em Engenharia Elétrica) – Departamento de Engenharia Elétrica, Universidade Federal do Rio Grande do Sul, Porto Alegre.

EASTMAN, Dick. **Converter um endereço de latitude e longitude**. 2014. Disponível em: <http://blog.eogn.com/2014/09/16/convert-an-address-to-latitude-and-longitude/>. Acesso em 19 out. 2014.

GEOREFERENCE. **Sistema de coordenadas UTM**. [S.l.], 2014. Disponível em: <http://georeference.blogspot.com.br/2010/02/sistema-de-coordenadas-utm.html>. Acesso em: 09 maio 2015.

GOLD RIDGE. **Latitude and longitude**. 2015. Disponível em:<http://www.goldridge08.com/latlong.htm>. Acesso em: 02 mar 2015.

GOOGLE. **Google mapas**. 2015. Disponível em: <https://www.google.com/intl/pt-BR/maps/about/explore/>. Acesso em: 17 maio 2015.

GRUPO DE PESQUISA EM COMPUTAÇÃO GRÁFICA. **FURB-Mobile**. 2014. Disponível em: <http://gcg.inf.furb.br/?page_id=3117>. Acesso em: 12 set 2014.

HARVARD. **Harvard mobile**. 2014. Disponível em: <http://m.harvard.edu>. Acesso em: 7 abr 2015.

IBGE (INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTÁTISTICA). **Noções básicas de cartografia**. Rio de Janeiro, 1998. Disponível em: <ftp://geoftp.ibge.gov.br/documentos/cartografia/nocoes_basicas_cartografia.pdf>. Acesso em 10 maio 2015.

KESTRING, Bruno A. **Sistema móvel na plataforma Android para compartilhamento de geolocalização usando mapas e notificações da Google**. 2014. 73 p. Trabalho de Conclusão de Curso (Bacharelado em Ciências da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

LIMA, Rodrigo Monteiro de.; MENDES NETO, Francisco Milton.; RODRIGUES, Rafael de Almeida. **Um jogo sério em 3D para apoiar a aprendizagem sensível ao contexto dos estudantes**. 2012. 10 p. Artigo (Bacharelado em Ciência da Computação) – Departamento de Ciências Exatas e Naturais, Universidade Federal Rural do Semi-Árido, Mossoró.

MONO. **Mono project**. 2014. Disponível em: <http://www.mono-project.com/>. Acesso em: 14 set 2014.

OLIVEIRA, Adilson de. **Das estrelas ao GPS**. 2011. Disponível em: <http://cienciahoje.uol.com.br/colunas/fisica-sem-misterio/das-estrelas-ao-gps>. Acesso em: 02 jul 2015.

PASSOS, Erick Baptista et al. Tutorial**:** Desenvolvimento de jogos com Unity 3D. In: SIMPÓSIO BRASILEIRO DE JOGOS E ENTRETERIMENTO DIGITAL, 8., 2014, Porto Alegre. **Anais eletrônicos...** Porto Alegre: PUCRS, 2014. Disponível em:<http://sbgames.org/papers/sbgames09/computing/tutorialComputing2.pdf>. Acesso em 02 maio 2015.

PITZ, Marcus Vinícius. **TCC2015 1 23 VF MarcusVPitz**. Blumenau, 2015. Disponível em:<https://www.youtube.com/watch?v=9pGjGue27Y0>. Acesso em: 03 jun 2015.

RAMOS, Cristhiane da Silva. **Visualização cartográfica e cartografia multimídia**. São Paulo: Unesp Editora, 2003. 179 p.

RIBEIRO JÚNIOR, José Geraldo et al. Sistema para monitoramento descentralizado de trânsito baseado em redes veiculares infraestruturadas. In: 31º SIMPÓSIO BRASILEIRO DE REDES DE COMPUTADORES E SISTEMAS DISTRIBUÍDOS, 10., 2013, Brasília. **Anais eletrônicos...** Brasília: Royal Tulip Brasilia Alvorada, 2010. Disponível em:<http://www.gta.ufrj.br/ftp/gta/TechReports/JCC13.pdf>. Acesso em 12 maio 2015.

SEUBERT, Curtis. **Definição de erro quadrático médio (EQM)**. [S.l.], 2015. Disponível em:<http://www.ehow.com.br/definicao-erro-quadratico-medio-eqm-fatos_6235/>. Acesso em: 04 jul 2015.

SIGNIFICADOS. **Latitude e longitude**. 2014. Disponível em: <http://www.significados.com.br/latitude-e-longitude/>. Acesso em: 13 set. 2014.

SYSTEM TUNER. **System Tuner:** Aplication and tools for Android. 2015. Disponível em:<http://www.3c71.com/android>. Acesso em: 06 jun 2015.

UNITY TECHNOLOGIES. **Unity editor**. 2014a. Disponível em: <http://unity3d.com/unity/workflow/integrated-editor >. Acesso em: 04 set. 2014.

_____. **Unity plataformas**. 2014b. Disponível em: <http://unity3d.com/unity/multiplatform>. Acesso em: 04 set. 2014.

_____. **Unity script reference: PlayerPrefs**. 2014c. Disponível em: <http://docs.unity3d.com/ScriptReference/PlayerPrefs.html>. Acesso em: 02 jun. 2015.

UNIVERSIDADE REGIONAL DE BLUMENAU. **Interação FURB**. 2014b. Disponível em: <http://www.furb.br/web/1368/relacao-com-a-comunidade/interacao-furb>. Acesso em 11 set 2014.

_____. **Localização**. 2014c. Disponível em: < http://www.furb.br/web/1740/institucional/localizacao/acesso-a-blumenau>. Acesso em 11 set 2014.

_____. **Relação com a comunidade**. 2014a. Disponível em: <http://www.furb.br/web/1003/relacao-com-a-comunidade>. Acesso em 11 set 2014.