UNIVERSIDADE DE SÃO PAULO
Escola de Engenharia de São Carlos
Departamento de Engenharia Elétrica e de Computação

Aplicação Móvel iOS com Realidade Aumentada Aplicada a uma Plataforma de Ensino de Programação para Crianças

Ivo Gimenes Dutra

São Carlos - SP

Ivo Gimenes Dutra

Aplicação Móvel iOS com Realidade Aumentada Aplicada a uma Plataforma de Ensino de Programação para Crianças

Trabalho de Conclusão de Curso apresentado
à Escola de Engenharia de São Carlos, da
Universidade de São Paulo

Curso de Engenharia Elétrica
com ênfase em Eletrônica

ORIENTADORA: Prof. Dra. Kalinka Regina L. J. C. Branco

USP – São Carlos

2019

Ficha catalográfica elaborada pela Biblioteca Prof. Dr. Sérgio Rodrigues Fontes da EESC/USP com os dados inseridos pelo(a) autor(a).

Dutra, Ivo Gimenes
D975a Aplicação Móvel iOS com Realidade Aumentada Aplicada a uma Plataforma de Ensino de Programação para Crianças / Ivo Gimenes Dutra; orientadora Kalinka Regina L. J. C. Branco. São Carlos, 2019.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2019.

1. Realidade Aumentada. 2. Aplicação Móvel iOS. 3. Ensino de Programação para Crianças. I. Título.

Eduardo Graziosi Silva - CRB - 8/8907

FOLHA DE APROVAÇÃO

Nome: Ivo Gimenes Dutra

Título: "Aplicação Móvel iOS com Realidade Aumentada Aplicada a uma Plataforma de Ensino de Programação para Crianças"

Trabalho de Conclusão de Curso defendido e aprovado em 22/11/2019,

com NOTA 10,0 (Dez), pela Comissão Julgadora:

Profa. Associada Kalinka Regina Lucas Jaquie Castelo Branco - Orientadora - SSC/ICMC/USP

Prof. Associado Rogério Andrade Flauzino - SEL/EESC/USP

Mestre Mariana Rodrigues - Doutoranda - ICMC/USP

Coordenador da CoC-Engenharia Elétrica - EESC/USP:
Prof. Associado Rogério Andrade Flauzino

Aos meus pais, família, e a todos que compartilharam comigo esta jornada...

"Stay Hungry,

Stay Foolish"

— Steve Jobs

Agradecimentos

Aos meus pais, minha família e minha namorada por todo o amor e por me guiarem na vida.

Agradecimentos especiais à Profa. Dra. Kalinka R. C. Branco e ao Kaique Lupo Leite pela confiança e pelo apoio no projeto.

A todos os meus amigos da Apple Developer Academy pela paciência, aprendizado e amizades.

A esta universidade por ter tornado possível inúmeras experiências vividas, profissionais e pessoais, no Brasil e no exterior.

E, é claro, a todos os amigos que São Carlos me proporcionou que fizeram parte desta jornada.

Ivo Gimenes Dutra.

Resumo

Realidade aumentada, que trata da sobreposição de objetos virtuais ao mundo real, é um conceito que remonta à década de 1960. Muito se avançou nesta tecnologia desde então e, ao que tudo indica, devido a popularização de dispositivos móveis, ela se tornará mais presente no cotidiano nos próximos anos. Inserido neste contexto, este trabalho tem o intuito de mostrar as etapas do desenvolvimento de uma aplicação em realidade aumentada para iOS aplicada a uma plataforma de ensino de programação para crianças, o FEPSE. O projeto detalha o funcionamento da plataforma, bem como aspectos da arquitetura de um projeto iOS, como o modelo MVC (*Model-View-Controller*). O *framework* adotado, ARKit, também foi discutido em detalhe. Por fim, os resultados obtidos podem ser vistos como uma prova de conceito para uma possível integração da plataforma com a aplicação.

Palavras-Chave: Realidade Aumentada, Aplicação Móvel iOS, Ensino de Programação para Crianças.

Abstract

Augmented reality, which deals with the composition of virtual objects in the real world, is a concept that goes back to the 1960s. Since then a lot of progress has been made in this technology and due to the popularization of mobile devices, it will probably become more and more present in the next years. Whithin this context, this paper shows the development steps of an augmented reality application for iOS applied to a children's programming learning platform, FEPSE. The project reveals details of this platform, as well as architectural aspects of an iOS project, such as the MVC (Model-View-Controller) model. The adopted framework, ARKit, was also discussed in details. Finally, the results can be seen as a proof of concept for a possible integration of the platform with the application.

Key-Words: Augmented reality, iOS Mobile Applications, Programming Learning for Children.

Lista de Figuras

2.1 Sistema de RA por Sutherland 24

2.2 *Touring Machine* de Steve Feiner 25

2.3 Sistema de marcação *CyberCode* 26

2.4 Primeira linha amarela em um jogo de futebol americano 27

2.5 ARToolKit: calibração do HMD por meio do marcador de referência . 28

2.6 Jogo ARquake 29

2.7 HoloLens 2 permite interação com objetos virtuais 30

2.8 Pokémon GO 31

2.9 GeoGebra AR auxilia no aprendizado de matemática 32

2.10 Camadas do iOS 33

2.11 Estrutura de um aplicativo móvel baseado no modelo MVC 36

3.1 Carrinho FEPSE 40

3.2 Crianças posicionam obstáculos para o carrinho 41

3.3 Exemplo de programa utilizando a linguagem desenvolvida pelo FEPSE 42

3.4 *Feature points* em amarelo: quanto mais contraste e detalhes uma imagem tiver, mais pontos serão detectados 43

3.5 Estrutura das aplicações do ARKit 45

4.1 Organização do projeto de acordo com a arquitetura MVC 49

4.2 Configuração do ARKit 50

4.3 Código para detecção do plan 51

4.4 Etapa 1. Animação de marcas de pneu apontam superfície plana detectada. *Feature points* em amarelo 52

4.5 Etapa 2, criança posiciona seus próprios obstáculos 54

4.6 Etapa 3, objeto *car* percorre caminho descrito pelo programa 56

Siglas

RA	*Realidade Aumentada*
AR	*Augmented Reality*
FEPSE	*Framework de Ensino de Programação e Sistemas Embarcados*
HDM	*Head Mounted Display*
CRT	*Cathode Ray Tube*
NFL	*National Football League*
GPS	*Global Positioning System*
MVC	*Model-View-Controller*
VANTs	*Veículos Aéreos Não Tripulados*
WWDC	*Worldwide Developers Conference*

Sumário

1 Introdução **19**

1.1 Motivação 20

1.2 Objetivos do Trabalho 20

1.3 Organização do Trabalho 21

2 Embasamento Teórico **23**

2.1 Considerações Iniciais 23

2.2 Realidade aumentada (RA) 23

2.2.1 Histórico 24

2.2.2 Aplicações atuais da RA 29

2.3 Aplicativos móveis iOS 33

2.3.1 Arquitetura do iOS 33

2.3.2 MVC - *Model-View-Controller* 35

2.4 Considerações Finais 37

3 Materiais e Métodos **39**

3.1 Considerações iniciais 39

3.2 FEPSE - *Framework* de Ensino de Programação e Sistemas Embarcados 39

3.3 ARKit 42

3.4 Considerações finais 46

4 Resultados e Discussões **47**

4.1 Considerações Iniciais 47

4.2 Organização da Aplicação 47

4.2.1 Organização das etapas 47

4.2.2 Organização do projeto 48

4.3 Detecção do plano 49

4.4 Posicionamento dos obstáculos 52

4.5 Visualização da execução do programa 54

4.6 Considerações Finais 56

5 Conclusões **57**

5.1 Trabalhos Futuros 58

Capítulo 1

Introdução

É raro no Brasil que se encontre escolas de ensino fundamental onde haja aulas de programação. Esta disciplina não só oferece um aprendizado de diversas habilidades e conhecimentos para crianças e jovens, mas também os insere na atual sociedade informatizada em que vivemos.

Contudo, este ensino deve ser realizado de forma especial e adequado ao seu público-alvo. De acordo com [1], as linguagens de programação mais tradicionais não são as mais indicadas para estas aulas, já que estas não possuem uma forma fácil e efetiva de serem ensinadas para crianças de 8 e 9 anos.

Neste contexto, o **FEPSE** (*Framework* de Ensino de Programação e Sistemas Embarcados) [2] tem como principal objetivo ensinar programação e lógica para crianças de 5 a 12 anos de um modo divertido, interativo, prático e colaborativo com a preocupação de ser acessível a todas as classes sociais. Esta plataforma, que nasceu como um projeto de iniciação científica do então aluno Kaique Lupo Leite sob orientação da Prof. Dra. Kalinka Regina L. J. C. Branco, possui a sua própria linguagem de programação, a qual utiliza figuras e não texto. Cada figura é impressa em um cartão contendo um código de barras, de forma que cada figura possa ser lida

por um celular. Os cartões representam, por fim, instruções básicas de lógica [3].

1.1 Motivação

O FEPSE funciona hoje utilizando sistemas embarcados para que a criança ou jovem veja seu programa em execução. Entretanto, haverá situações nas quais o sistema embarcado pode não se fazer presente, seja por motivos de manutenção, seja pelo fato da pessoa estar fora do ambiente de sala de aula.

Neste cenário, a realidade aumentada (RA, ou AR - *augumented reality* em inglês) pode ser uma solução. RA pode ser definido como sendo a tecnologia a qual sobrepõe objetos virtuais (considerados objetos aumentados) no mundo real. Esta tecnologia pode então facilitar o aprendizado por ajudar os alunos a engajarem em explorações no mundo real, aumentando a motivação dos mesmos bem como melhorando suas habilidades investigativas. [4]

1.2 Objetivos do Trabalho

Pelas razões apresentadas anteriormente, o objetivo deste trabalho é estender o atual FEPSE para o campo da realidade aumentada por meio de uma simulação apresentada pelo aplicativo móvel. Isto permitirá que a criança de continuidade ou inicie uma nova atividade fora do ambiente da sala de aula ou na ausência do sistema embarcado.

Para tanto são apresentados os conceitos de RA e da arquitetura de uma aplicação iOS, plataforma adotada para o trabalho.

Tem-se, como resultado, uma prova de conceito da aplicação de RA para o FEPSE, dado o estágio em que a tecnologia de RA se encontra no presente momento.

1.3 Organização do Trabalho

A monografia está dividida da seguinte forma: o capítulo 2 apresenta todo o conhecimento teórico necessário para o desenvolvimento do trabalho, em especial, conceitos de RA e da arquitetura de uma aplicação móvel iOS. A seguir, no capítulo 3, são apresentados a plataforma FEPSE e o funcionamento do *framework* ARKit. O capítulo 4 apresenta os resultados do trabalho como um todo. Por fim, no capítulo 5, as conclusões finais do trabalho e os trabalhos futuros são apresentados.

Capítulo 2

Embasamento Teórico

2.1 Considerações Iniciais

Este capítulo compila a bibliografia necessária para o entendimento e o desenvolvimento deste projeto, abordando o conceito de RA, seu histórico e principais aplicações, além do desenvolvimento de um aplicativo móvel iOS.

2.2 Realidade aumentada (RA)

Há na comunidade científica diversas definições para RA. A definição mais tradicional desta tecnologia traz qualquer sistema o qual satisfaça três características [5]:

- Combina elementos do mundo real e do virtual;
- Possui interação em tempo real;
- Ajusta objetos virtuais no espaço tridimensional.

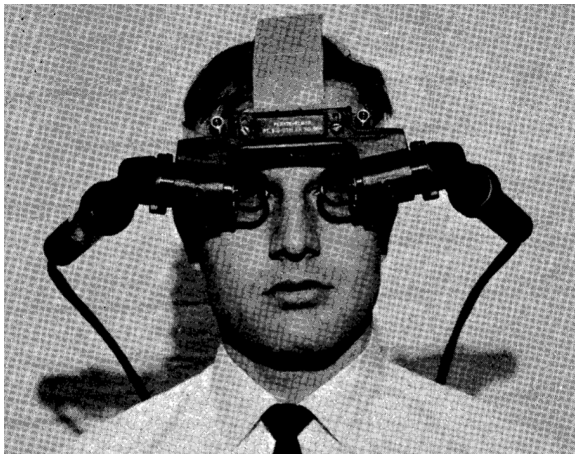
Uma definição mais ampla traz RA como sendo qualquer tecnologia que mistura informação real com virtual de uma forma significativa. Ou seja, também seria considerado RA como sendo qualquer contexto do mundo real o qual é sobreposto dinamicamente com informações virtuais relativas ao mesmo [6].

2.2.1 Histórico

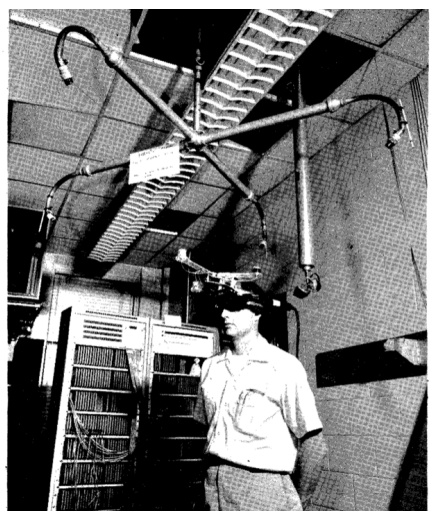
O primórdio da RA muito se confunde com o da realidade virtual. Ao traçar uma linha do tempo, muitos especialistas começam por experiências com o cinema no começo da década de 1960, por Morton Heiling. Mas a primeira experiência considerada realidade aumentada veio em 1968 por Ivan Sutherland, utilizando um "óculos", ou dispositivo de *display*, **HDM** (*Head Mounted Display*) [7]. O dispositivo era composto por dois *displays* **CRT** (*Cathode Ray Tube*) e, como ilustrado na Figura 2.1, dispunha de conexões diretas ao computador e no teto. A ideia básica por trás deste experimento era projetar uma imagem na retina do usuário a qual se deslocava de acordo com o movimento de sua cabeça. Criava-se, assim, uma ilusão de um objeto em três dimensões.

Figura 2.1: Sistema de RA por Sutherland

(a) Sua versão do dispositivo HDM



(b) Conexões do equipamento



Fonte: [8]

Entretanto, este dispositivo foi inventado antes mesmo do próprio termo "realidade aumentada ser cunhado. Isso aconteceu em 1992, quando dois pesquisadores, Dave Mizell e Tom Caudell, escreveram um artigo sobre o desenvolvimento de um dispositivo para auxiliar a montagem de cabos em aeronaves na Boeing cujo objetivo era substituir os manuais de instruções [9].

Após a divulgação deste artigo, muitas companhias começaram a investir nesta tecnologia para aplicações industriais. Em 1993 Louis Rosenberg desenvolveu para um laboratório militar americano a primeira experiência de RA interativa, *Virtual Fixtures*. Nela, o usuário controlava braços robóticos a distância com sobreposição de informações virtuais que auxiliavam a execução de tarefas específicas [10].

Neste mesmo ano, um grupo do pesquisador Steve Feiner na Universidade de Columbia, Estados Unidos, desenvolveu um sistema com óculos de RA capaz de mostrar como realizar a manutenção em uma impressora. O projeto ficou conhecido como KARMA [11]. Alguns anos depois, em 1999, Feiner e seu grupo de pesquisa desenvolveram um outro protótipo de óculos, desta vez móvel, chamado de *Touring Machine* (Figura 2.2).

Figura 2.2: *Touring Machine* de Steve Feiner

(a) Equipamento com mochila

(b) Informações sobre os prédios

Fonte: [12]

Este óculos fornecia informações sobre os prédios do campus da sua universidade com base na sua posição pelo **GPS** (*Global Positioning System*) [13]. Apesar do protótipo oferecer certo grau de mobilidade, ainda era preciso que o usuário utilizasse uma mochila para transportar os circuitos necessários, como ilustrado na Figura 2.2.

Foi em 1996 que o primeiro sistema de marcação por referência (*fiducial markers*) para RA foi desenvolvido dentro da Sony por Jun Rekimoto, chamado de *CyberCode*. Tratava-se de identificador retangular constituído de tons de branco e preto os quais formavam um padrão que poderia ser facilmente reconhecido pelas câmeras de *notebooks* [14]. Este identificador era então utilizado como ponto de referência para um modelo 3D da realidade aumentada, como ilustrado na Figura 2.3. Esta técnica permitia que, caso a imagem de referência no mundo real se mexesse, o objeto virtual a acompanharia. Muitos trabalhos em RA utilizam marcadores de referência desde então.

Figura 2.3: Sistema de marcação *CyberCode*

Fonte: [15]

Como hoje realidade aumentada possui um conceito mais amplo, pode-se identificar seu uso também na televisão. Em 1998 uma empresa chamada Sportvision desenvolveu a primeira linha computadorizada do *1st & Ten* (Figura 2.4) e

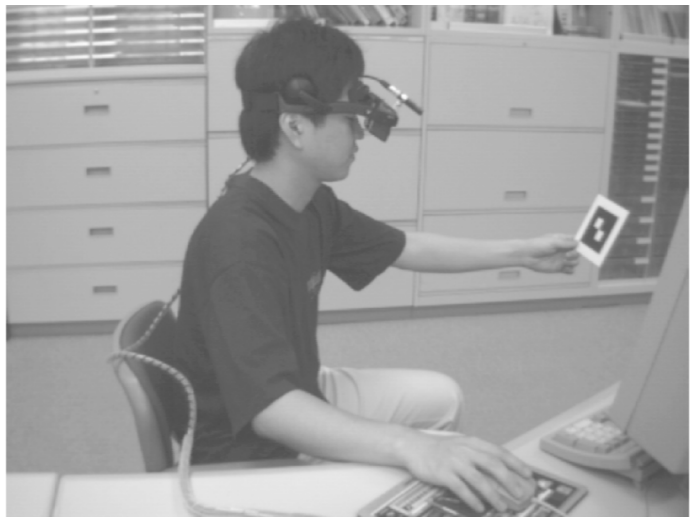
transmitiu em um jogo ao vivo de futebol americano da **NFL** (*National Football League*) [16]. Para a época isto foi um grande feito pelo fato da linha aparecer embaixo dos jogadores, além de ajudar a aproximar o público da RA.

Figura 2.4: Primeira linha amarela em um jogo de futebol americano

Fonte: [17]

A primeira biblioteca *open source* (código aberto) de desenvolvimento para RA veio em 1999, chamada de ARToolKit [18]. Foi desenvolvida por Hirokazu Kato e Mark Billinghurst e inicialmente era focada em oferecer um método para reconhecer os marcadores de referência (parecidos com o *CyberCode*) bem como calibrar os dispositivos HMD (Figura 2.5). Esta biblioteca é utilizada até os dias de hoje, sendo uma das primeiras a estar disponível para aplicações móveis. Seu código está disponível no GitHub [19].

Figura 2.5: ARToolKit: calibração do HMD por meio do marcador de referência



Fonte: [18]

Considerado o primeiro jogo em realidade aumentada, Bruce Thomas desenvolveu em 2000 o ARquake [20]. O jogo se propunha a ser uma adaptação do jogo para *desktop* Quake, um jogo em primeira pessoa. Ele poderia ser jogado tanto em áreas internas quanto externas e funcionava utilizando uma bússola digital, sistema GPS e marcadores de referência. A Figura 2.6 ilustra a visão em primeira pessoa do jogo e também todo o equipamento que era acoplado aos óculos.

Alguns anos depois, em 2008, a fabricante de automóveis BMW foi a primeira empresa a utilizar RA como instrumento para o *marketing* na campanha do MINI [21]. Na propaganda, ao invés de conter a foto do novo modelo, vinha apenas um marcador de referência com a frase: *The MINI Convertible in your hands* (O conversível MINI em suas mãos). Ao entrar na página *web* da empresa era possível então apontar uma *webcam* normal para o marcador e o modelo 3D do novo MINI aparecia na tela.

Figura 2.6: Jogo ARquake

(a) Visualização de objetos do jogo

(b) Equipamento necessário para realização do jogo

Fonte: [20]

Já em 2010, em meio ao crescimento da popularidade dos *smartphones*, é fundada uma *startup* dentro da Google, chamada Niantic [22]. Ela foi formada com o objetivo de explorar como dispositivos móveis e o conhecimento sobre mapas poderia ser utilizado para promover uma exploração de novos lugares bem como a interação do mundo real com as pessoas. Pouco tempo depois da sua fundação, a empresa começou a pesquisar como criar jogos em realidade aumentada. Cinco anos depois esta empresa se torna independente da Alphabet (*holding* da Google).

2.2.2 Aplicações atuais da RA

A partir desse histórico, percebe-se que as aplicações para RA são das mais variadas, indo do *marketing* para entretenimento, bem como educação e ferramenta de produtividade. Abaixo são listadas algumas das principais aplicações atuais para RA, além do que algumas das maiores empresas do mundo estão desenvolvendo a respeito desta tecnologia.

Óculos

A Google lançou o dispositivo Goggle Glass em 2012 e abriu sua venda apenas para um público seleto em 2014. O aparelho era semelhante a um par de óculos, que fixados em um dos olhos, disponibiliza uma pequena tela acima do campo de visão. A divulgação do novo produto na época foi dada por um vídeo no qual o usuário era capaz de verificar eventos na sua agenda, acessar a previsão do tempo, tirar foto e tocar música, todos comandos acionados por voz. Apesar do seu caráter disruptivo, debates sobre seu impacto na privacidade das pessoas ao redor do usuário, além do seu design futurista pouco atraente e preço não acessível, culminaram no encerramento das vendas em 2015 [23]. Entretanto, a Google anunciou em 2019 a nova versão do projeto, Glass Enterprise, voltada única e exclusivamente para o ambiente de trabalho, podendo ser utilizado por médicos até operadores de máquinas [24].

Já o dispositivo HoloLens (Figura 2.7), da Microsoft, busca cunhar o conceito de realidade misturada (*mixed reality*).

Figura 2.7: HoloLens 2 permite interação com objetos virtuais

Fonte: [25]

O projeto deste aparelho surgiu através de uma tentativa de transportar a tecnologia do Kinect (sensor de movimentos da Microsoft) para um dispositivo HMD. A primeira versão foi lançada em 2016 e a versão mais atual, HoloLens 2, foi anunciada em fevereiro de 2019 [26]. Ao contrário do Glass, HoloLens 2 ocupa todo o campo de visão e permite a visualização de objetos 3D sobrepostos no mundo real. No evento do lançamento também foi demonstrado que os óculos reconhecem as mãos do usuário, o que permite a interação em tempo real com objetos virtuais.

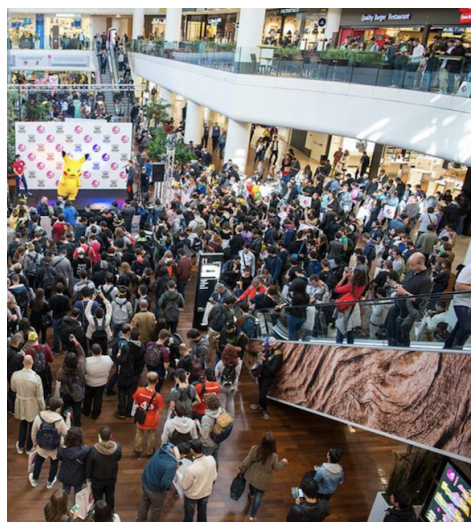
Jogos

Lançado em 2016, Pokémon GO (Figura 2.8) é até hoje o aplicativo de realidade aumentada mais baixado do mundo, e também o mais rentável. Até 2018, o jogo rendeu quase 2 bilhões de dólares e foi baixado mais de 800 milhões de vezes [22][27]. Desenvolvido pela Niantic em colaboração com o *The Pokémon Company* e com a Nintendo, o jogo utiliza a localização GPS real do usuário para capturar e batalhar as criaturas Pokémon, em localizações do mundo real.

Figura 2.8: Pokémon GO

(a) Visualização dos pokemóns no ambiente

(b) Evento organizado pelo *app*



Fonte: [22]

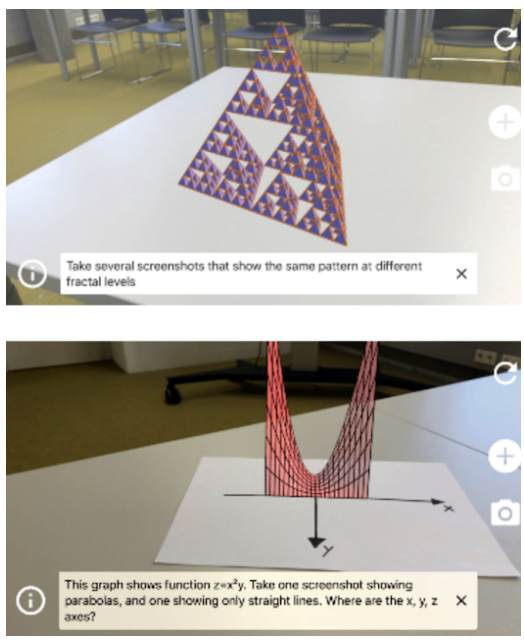
Um fenômeno de popularidade, até estudos sobre o impacto do uso do Pokémon GO na saúde das pessoas foram realizados, concluindo que, no curto prazo, houve melhora no nível de atividade física do jogador [28].

Educação

Muitas aplicações notáveis no campo da educação auxiliam o aluno na compreensão da dimensão espacial. Este é o caso do aplicativo *Complete Anatomy Platform 2020*, da 3D4Medical [29]. Ele permite que o usuário visualize, em detalhes e em tamanho real, o interior do corpo humano. Modelos 3D sobre anatomia do corpo foram refinados ao longo de 15 anos de pesquisa.

De forma similar, o aplicativo *GeoGebra Augmented Reality* (Figura 2.9) tem o intuito de projetar gráficos matemáticos e objetos geométricos em 3D em realidade aumentada de forma que seja possível caminhar por eles, permitindo uma interatividade que não seria possível apenas por uma simulação. O *app* também conta com aulas guiadas para ensinar alunos sobre os mais diversos assuntos de geometria.

Figura 2.9: GeoGebra AR auxilia no aprendizado de matemática



Fonte: [30]

Investimentos

Hoje há um grande investimento por parte das maiores companhias em *software mobile* que existem, havendo *frameworks* tanto para iOS quanto para Android, o que permite desenvolvedores ao redor do mundo criarem as mais diversas soluções [31].

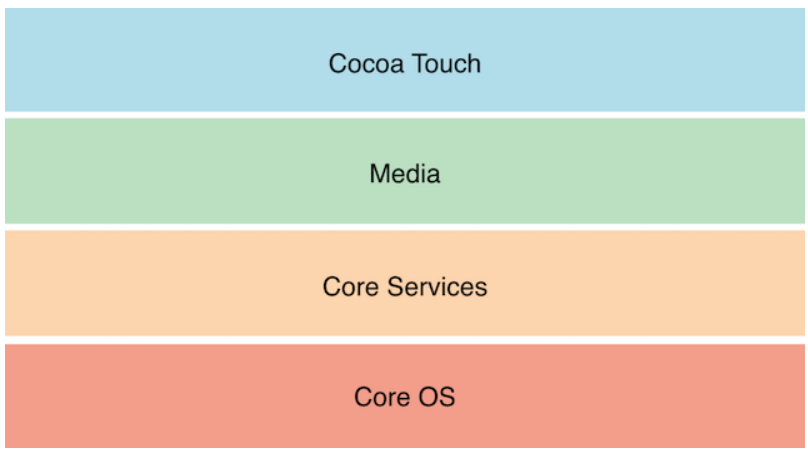
2.3 Aplicativos móveis iOS

2.3.1 Arquitetura do iOS

O sistema operacional iOS é dividido em camadas. Normalmente, aplicativos desenvolvidos para esta plataforma "conversam" com o hardware do dispositivo por meio de um sistema de interfaces desenvolvido para proteger o aplicativo de alterações súbitas, sem acesso direto [32].

Esse sistema de interfaces é ilustrado na Figura 2.10. As camadas inferiores representam serviços e tecnologias fundamentais para o funcionamento do dispositivo. Já as camadas mais altas representam serviços e tecnologias mais sofisticados [32].

Figura 2.10: Camadas do iOS



Fonte: [32]

Core OS

A camada mais fundamental, *Core* OS, é a camada de mais baixo nível que contém todas as funcionalidades sobre as quais as demais camadas são construídas. Qualquer situação que envolva algo relativo à segurança ou à comunicação com outros dispositivos externos é tratada por esta camada. Aqui destacam-se os seguintes *frameworks* e suas funcionalidades:

- *Accelerate Framework*: processamento digital de sinais;
- *Core Bluetooth Framework*: comunicação via *Bluetooth Low-Energy*;
- *Security Framework*: segurança de dados, certificados e criptografia;
- *System*: *kernel*, *drivers*, e todas as interfaces UNIX de baixo nível.

Core Services

Core Services contém os serviços fundamentais que as aplicações irão utilizar. Mesmo que não se utilize diretamente estes serviços, muitas partes do sistema são construídas a partir deles. Aqui destacam-se os seguintes *frameworks* e suas funcionalidades:

- *Core Foundation Framework*: conjunto de interfaces em C, para tratamento básico de dados;
- *Core Location Framework*: obtém a latitude e longitude do dispositivo;
- *Core Data Framework*: cuida do banco de dados do usuário.

Media

Esta camada é responsável por controlar os dados multimídia do dispositivo, ou seja, é responsável pelo tratamento dos dados gráficos e de áudio. Aqui destacam-se os seguintes *frameworks* e suas funcionalidades:

- *Core Graphics Framework*: renderiza imagens vetorizadas 2D;
- *Core Animation Framework*: suporte avançado para animações;
- *AV Foundation Framework*: processamento de áudio, recursos audiovisuais.

Cocoa Touch

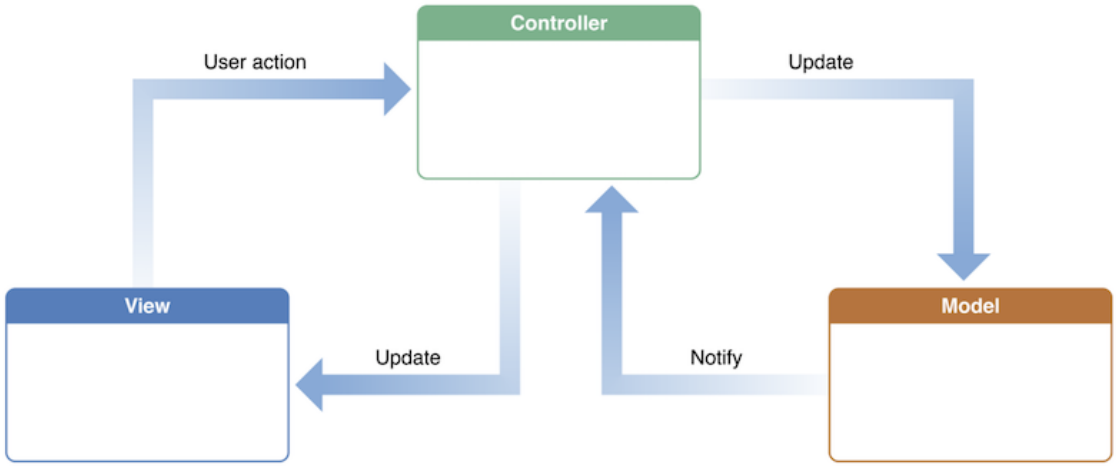
O termo Cocoa refere-se ao conjunto de *frameworks* e tecnologias principais utilizadas para desenvolver aplicações iOS. Aqui se fornece uma infraestrutura básica para funções como *multitasking*, toque na tela, notificações e outros serviços considerados de alto nível. Aqui destacam-se os seguintes *frameworks* e tecnologias:

- *UIKit Framework*: principal biblioteca para implementar aplicativos gráficos no iOS. Apresenta suporte para reconhecimento de toque, *multitasking*, notificações, entre outros;
- *Address Book UI Framework*: interface para criar e adicionar novos contatos;
- *Auto Layout*: torna o *design* das telas responsivo ao tamanho do aparelho.

2.3.2 MVC - *Model-View-Controller*

O padrão de arquitetura de *software* **MVC** (*Model-View-Controller*, ou Modelo-Visualização-Controle) é o recomendado pela Apple para ser utilizado no desenvolvimento de aplicações iOS [33]. Este padrão não só agrupa os diferentes objetos da aplicação de acordo com a funcionalidade que desempenham, mas também define como é a comunicação entre eles. Esta arquitetura de código é ilustrada na Figura 2.11.

Figura 2.11: Estrutura de um aplicativo móvel baseado no modelo MVC



Fonte: [34]

Model

A camada *Model* encapsula todos os objetos relativos ao tratamento de dados do *app* bem como toda a lógica que processa esses dados. Estes objetos não só expressam uma lógica específica para resolver determinados problemas, mas também são construídos de tal forma que possam ser reutilizados para resolver problemas similares. Isso implica no fato de, idealmente, objetos desta camada não se preocupam com a interface do usuário ou na forma a que os dados são apresentados.

Por isso a sua comunicação é estrita com objetos da camada *Controller*: quando um objeto *Model* é alterado (por exemplo, novos dados são recebidos por uma conexão de rede), ele notifica um objeto *Controller*, que por sua vez atualiza os objetos de exibição apropriados.

View

A camada *View*, ou Visualização, concentra todos os objetos da aplicação que o usuário enxerga. Eles sabem, portanto, como devem ser desenhados e como devem tratar as interações do usuário. A principal função deles é mostrar os dados da

camada *Model* e permitir que os mesmos possam ser editados. Apesar desta suposta conexão, objetos desta camada são desassociados dos da camada *Model*.

Os objetos da *View* exibem informações sobre alterações nos dados da *Model* por meio dos objetos *Controller* do aplicativo, e vice-versa. Por exemplo, se um texto for inserido em um campo de texto, os objetos da *View* comunicam alterações iniciadas pelo usuário aos objetos *Model*, por meio dos objetos *Controller*.

Controller

Um objeto *Controller* atua como intermediário entre um (ou mais) objetos da *View* e um (ou mais) objetos da *Model*. Ele atua, portanto, como um canal por meio do qual os objetos *View* se informam sobre mudanças nos objetos *Model* e vice-versa. Os objetos *Controller* também podem executar tarefas de configuração e coordenação da aplicação móvel e gerenciar os ciclos de vida de outros objetos.

A comunicação, portanto, é com as duas camadas. Um objeto *Controller* interpreta as ações do usuário feitas nos objetos da *View* e comunica dados novos ou alterados à camada *Model*. Quando os objetos *Model* são alterados, um objeto *Controller* comunica esses novos dados aos objetos da *View* para que eles possam exibi-los.

2.4 Considerações Finais

Este capítulo apresentou o embasamento e as informações necessárias para a realização de um aplicativo móvel iOS o qual se faz uso da realidade aumentada.

A tecnologia da realidade aumentada, descrita neste capítulo, é um campo muito extenso e não é algo de exclusividade de aplicações iOS. No entanto, a Apple fornece ferramentas próprias as quais tornam o seu desenvolvimento mais fácil, de forma que desenvolvedores possam focar mais no conteúdo de suas aplicações.

O próximo capítulo demonstra como esta tecnologia foi empregada.

Capítulo 3

Materiais e Métodos

3.1 Considerações iniciais

Para construir a aplicação móvel iOS deste projeto, utilizou-se o ambiente de desenvolvimento da Apple Xcode, o *framework* para realidade aumentada da Apple ARKit, um celular iPhone XR e a estrutura do FEPSE. Este capítulo detalha o que é e como o projeto se relaciona com o FEPSE, bem como aspectos e requisitos do *framework* adotado.

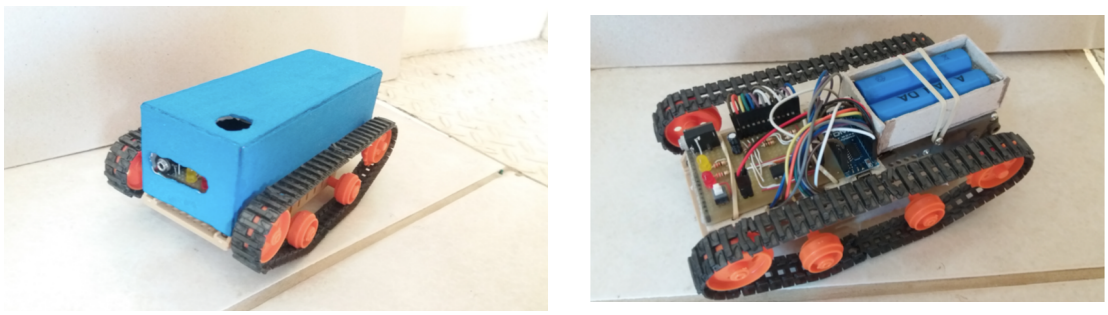
3.2 FEPSE - *Framework* de Ensino de Programação e Sistemas Embarcados

Desenvolvido em 2017, o FEPSE nasceu como um projeto de iniciação científica do então aluno Kaique Lupo Leite sob orientação da Prof. Dra. Kalinka Regina L. J. C. Branco. É uma plataforma de ensino de baixo custo voltada ao ensino de programação e lógica para crianças de 5 a 12 anos. Ela tem como objetivo

ensinar conceitos de sistemas embarcados, computação tangível e programação visual de uma forma divertida, prática, interativa e acessível a todas as classes sociais [3].

A ideia central do FEPSE está na solução de problemas (ou atividades) propostos pelo educador por meio de algoritmos de computação, utilizando a sua própria linguagem de programação. Estes algoritmos são então executados em sistemas embarcados como **VANTs** (Veículos Aéreos Não Tripulados), carros de controle remoto, barcos de controle remoto entre outros. Para efeito deste trabalho em realidade aumentada, o carrinho FEPSE, ilustrado na Figura 3.1, foi modelado em 3D.

Figura 3.1: Carrinho FEPSE



(a) Vista completa

(b) Vista do interior

Fonte: [3]

As atividades propostas são práticas pois não se necessita de equipamentos além do próprio carrinho. Como ilustrado na Figura 3.2, as próprias crianças desenvolvem seus desafios, ou seja, são elas que posicionam os obstáculos (objetos disponíveis em sala de aula, como caderno e lápis) e na sequência programam o caminho pelo qual o carrinho deverá percorrer para concluir ou sair de tal labirinto.

A computação tangível (interação com um sistema digital por meio de objetos físicos) é utilizada na forma da linguagem de programação. Ela representa de modo direto e simples suas instruções por meio de peças impressas em papel comum e encaixadas umas as outras como um quebra cabeça. Nestes cartões, instruções como início e término do programa, ir para frente ou para trás, para o lado, instruções de repetição, entre outras, são representadas por símbolos. Desta forma, a criança

Figura 3.2: Crianças posicionam obstáculos para o carrinho

Fonte: Dados do arquivo do Laboratório de Sistemas Embarcados Críticos

utiliza uma interface amigável sem utilizar periféricos como teclado ou *mouse*, além do fato dela não precisar ser alfabetizada para reconhecer as figuras.

Para estas peças serem identificadas pelo computador, foi utilizado código de barras devido a sua simplicidade e baixo custo. Estes códigos são então identificados por visão computacional por meio de uma câmera, traduzindo para o computador qual instrução deve ser executada. Uma vez lidas e compreendidas as instruções, o programa é então transmitido para o sistema embarcado por meio de um cabo p2. Um exemplo de programa, utilizando os cartões de instruções e seus respectivos códigos de barra é ilustrado na Figura 3.3.

Dado que toda a parte de leitura e compreensão das instruções já foi implementada pelo FEPSE, este trabalho de aplicação móvel não contempla esta parte de visão computacional, já implementada e validada. Como o escopo deste projeto

é estudar a realidade aumentada, os dados que deveriam ser obtidos por meio desta ponte foram adquiridos por um *mock* (simulação). Por isso, este projeto tem em seu diretório raiz o arquivo *Input.txt*, o qual representa a sequência de instruções previamente programadas.

Figura 3.3: Exemplo de programa utilizando a linguagem desenvolvida pelo FEPSE

Fonte: [3]

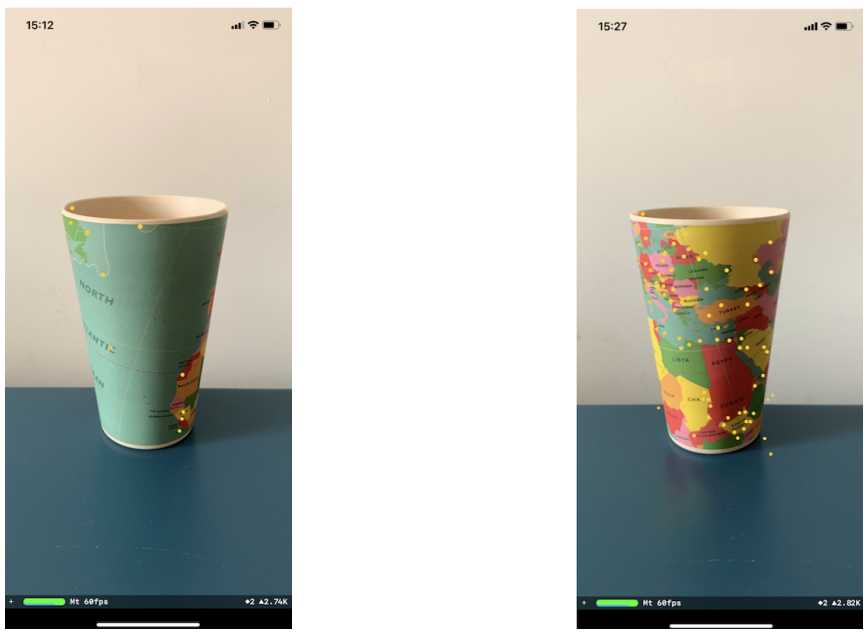
3.3 ARKit

ARKit é o *framework* da Apple para o desenvolvimento de aplicações móveis iOS com realidade aumentada. Para isso, ARKit integra a câmera do dispositivo a sensores de movimento e a processamentos de imagem avançados para projetar objetos virtuais no mundo visto pela tela do dispositivo, de forma que eles pareçam

de fato pertencerem ao mundo real. Ele permite a projeção de objetos tanto 2D quanto 3D, tanto na câmera traseira quanto na frontal do iPhone [35].

Para que as imagens obtidas em tempo real pela câmera possam ser processadas e cálculos como a distância da câmera até objetos possam ser realizados, o ARKit utiliza os chamados *feature points*. *Feature points* são pontos detectados na imagem da câmera os quais representam características notáveis de objetos, tais como: cantos, linhas de estrutura, características do tecido, gradientes, alterações na cor, forma ou bordas de objetos. Observa-se que quanto mais contraste e detalhes uma imagem tiver, mais *feature points* serão detectados. Na Figura 3.4 é ilustrado como a aplicação detecta estes pontos: a Figura 3.4a quase não apresenta *feature points* (representados pelos pontos amarelos) devido a sua predominância monocromática. O mesmo não ocorre na Figura 3.4b, a qual possui uma variedade de cores.

Figura 3.4: *Feature points* em amarelo: quanto mais contraste e detalhes uma imagem tiver, mais pontos serão detectados



(a) Imagem com poucos detalhes (b) Imagem com mais detalhes

Fonte: O autor

Uma das funcionalidades mais importantes do ARKit está na sua habilidade

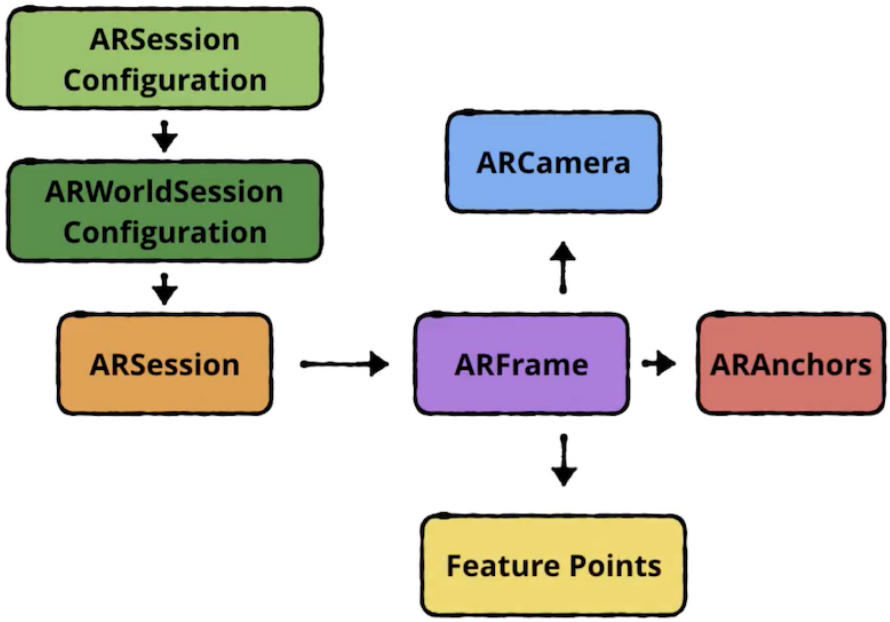
de rastrear a posição de um objeto virtual enquanto o celular se move. Ou seja, manter o objeto virtual fixo em relação ao seu sistema de coordenadas, também chamado de *world tracking*. Para que isto ocorra, o ARKit utiliza uma técnica chamada de *visual-inertial odometry*. Esse processo combina informações do *hardware* de detecção de movimento do iPhone com métodos de visão computacional da cena visível pela câmera do dispositivo. O ARKit detecta os *feature points* na imagem da cena, rastreia diferenças nas posições desses pontos nos *frames* de vídeo e compara essas informações com dados de detecção de movimento coletados pelos sensores do dispositivo. O resultado é um modelo de alta precisão da posição e movimento do aparelho móvel [36].

Código

Em termos de código, a primeira coisa que se deve fazer é criar um *ARSession*, algo como o cerne de toda aplicação deste tipo. Objetos do tipo *ARSession* são responsáveis por interpretar todos os dados captados pelos sensores de movimento, controlar a câmera do dispositivo e realizar a análise da imagem. Estes objetos são então configurados pela classe *ARWorldSessionConfiguration*. Lá é possível optar por habilitar funções como detecção de plano ou capturar a distância entre objetos. Iniciado o processamento de imagem, ARKit retorna um *ARFrame*, o que representa um *frame* do vídeo atual da câmera com informações adicionais. Uma delas é a posição da câmera em relação ao mundo real (*ARCamera*). Outra é dado pelas *ARAnchors*, que representam as posições e orientações de objetos em relação ao mundo real. Essas "âncoras"podem ser programaticamente adicionadas ou podem ser automaticamente criadas pelo *ARSession* quando este estiver detectando objetos (como um plano, por exemplo). Por último, *ARFrame* também carrega os *feature points* associados à imagem [37][38].

Este esquema de como as aplicações de ARKit são construídas é ilustrado na Figura 3.5.

Figura 3.5: Estrutura das aplicações do ARKit



Fonte: [38]

Limitações

Mesmo com ARKit oferecendo o estado da arte da tecnologia atual, o *world tracking* não é ciência exata. Por isso há um conjunto de três boas práticas para melhorar o resultado desses tipos de aplicações [36]:

- Utilizar iluminação adequada e objetos com alto contraste: a qualidade do rastreamento é reduzida quando a câmera não consegue ver detalhes, como quando a câmera está apontada para uma parede em branco ou a cena está muito escura;

- Mover o celular adequadamente: o ARKit desenvolve uma melhor compreensão da cena se o dispositivo estiver em movimento, mesmo se o dispositivo se mover apenas sutilmente;

- Dar tempo para que a detecção de plano possa produzir resultados claros e desativar a detecção de plano quando obter os resultados necessários;

Ressalta-se ainda que este *framework* só é possível de ser executado em dispositivos cujo iOS ou iPadOS seja superior ou igual ao iOS 11 e processador A9 [39]. Logo, esta aplicação pode ser executada a partir do iPhone 6s [40]. Para efeito deste projeto, foi utilizado um iPhone XR.

3.4 Considerações finais

Este capítulo apresentou os materiais e métodos utilizados para o desenvolvimento desta aplicação móvel iOS. Foi apresentado o FEPSE, a plataforma a qual este projeto se propõe a estender para o campo da realidade aumentada. Também foi apresentado o funcionamento do *framework* do ARKit, essencial para o desenvolvimento desta aplicação.

O próximo capítulo descreve detalhes do software desta aplicação e os respectivos resultados obtidos.

Capítulo 4

Resultados e Discussões

4.1 Considerações Iniciais

Como descrito na seção 1.2, o objetivo deste trabalho é o de estender o projeto do FEPSE para o campo da realidade aumentada. Para tanto, foi realizada uma aplicação móvel utilizando o sistema operacional iOS 12, arquitetura MVC e linguagem de programação Swift.

Nesse capítulo a organização da solução é apresentada, bem como o detalhe de cada uma das soluções implementadas em cada uma das etapas. Na seção 4.2, o fluxo do *app* e a organização do projeto são apresentados, enquanto que as demais seções trazem detalhes de cada uma das etapas.

4.2 Organização da Aplicação

4.2.1 Organização das etapas

Esta aplicação foi dividida em três etapas diferentes:

- **Detecção do plano:** posiciona a cena construída no mundo real;
- **Posicionamento dos obstáculos:** etapa interativa da aplicação;
- **Visualização da execução do programa:** carrinho FEPSE modelado em 3D executa o programa.

A detecção do plano é parte crucial da aplicação pois é a responsável por posicionar a cena construída no mundo real, ou seja, substitui os marcadores de referência. Já a segunda etapa visa reproduzir a Figura 3.2 em RA: primeiro o usuário constrói seu próprio labirinto e depois programa o caminho do carrinho FEPSE para que este então chegue na saída ou no objetivo final. Por último, o veículo executa o programa previamente construído pela criança utilizando os cartões da Figura 3.3.

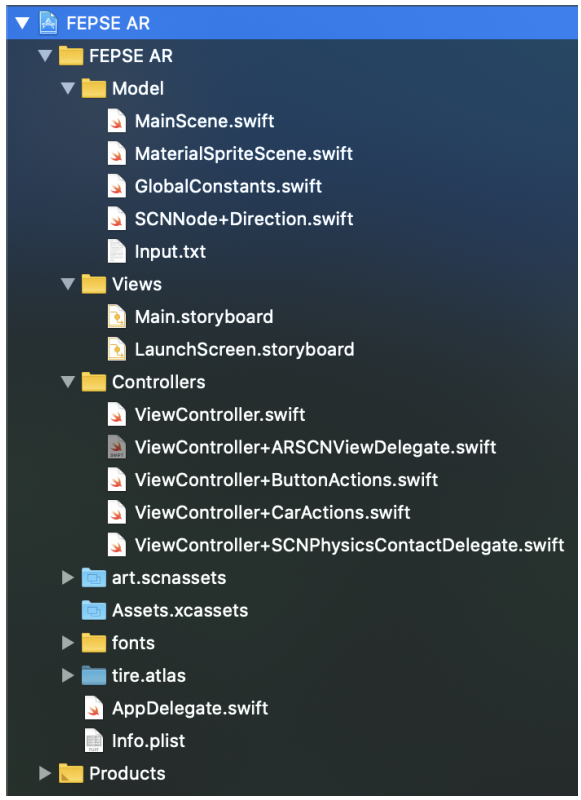
É importante frisar que como este trabalho visa o aprendizado em RA, optou-se por realizar a integração desta aplicação com a aplicação *web* do FEPSE no futuro. Desta forma, a leitura dos cartões (competência da aplicação *web* do FEPSE) foi dada por meio de um *mock*, ou seja, simulada pela leitura do arquivo *Input.txt*.

4.2.2 Organização do projeto

Como explicado na seção 2.3.2, este projeto procurou seguir a arquitetura MVC. A organização dos arquivos do projeto é ilustrada na Figura 4.1.

LaunchScreen.storyboard contém as *Views* que são chamadas apenas no momento quando o aplicativo está carregando. Todas as outras *Views* do projeto são organizadas pelo arquivo *Main.storyboard*. Este, por sua vez, contém apenas uma tela, controlada pela classe *ViewController*. Os demais arquivos *ViewController* são extensões da mesma, cada um com uma função determinada. Por exemplo, a principal *View* do projeto é uma do tipo *ARSCNView*, a qual é controlada com métodos que seguem o protocolo *ARSCNViewDelegate*. A rigor, o arquivo *MainScene.swift* não necessariamente seria um do tipo *Model*, no entanto ele contém todos os dados e as referências aos modelos 3D da cena a ser apresentada pela *ARSCNView*.

Figura 4.1: Organização do projeto de acordo com a arquitetura MVC



Fonte: O autor

4.3 Detecção do plano

A detecção do plano substitui os marcadores de referência neste projeto. Quando a câmera do celular detecta uma superfície plana (ou mais especificamente, um conjunto de *feature points* que remetem a uma superfície plana), é possível criar uma referência nesta posição no mundo real. Em outras palavras, no momento em que a aplicação é iniciada, cria-se um sistema de coordenadas com sua origem sendo a posição atual do dispositivo móvel. Ao detectar este conjunto específico de pontos, o dispositivo é capaz de medir a distância entre ele mesmo para com a superfície, criando assim uma referência. Nota-se que a referência permanece mesmo quando o celular se move.

Como descrito na seção 3.3, a primeira coisa que se deve fazer é criar um

objeto *ARSession*. Isto foi feito dentro da função *ViewWillAppear*, da classe *ViewController*, como ilustrado na Figura 4.2.

Figura 4.2: Configuração do ARKit

```
// Notifica o view controller que a view está prestes a ser adicionada
override func viewWillAppear(_ animated: Bool) {
    super.viewWillAppear(animated)

    // Inicia a configuração da ARKitSession
    let configuration = ARWorldTrackingConfiguration()
    // Habilita a visualização de Feature Points
    sceneView.debugOptions = [ARSCNDebugOptions.showFeaturePoints]
    // Habilita a detecção de planos horizontais
    configuration.planeDetection = .horizontal
    // Inicia a Session
    sceneView.session.run(configuration)
}
```

Fonte: O autor

Iniciado a *ARSession*, é possível utilizar os métodos do protocolo *ARSCNViewDelegate*. A função $renderer(\_, updateAtTime)$ é chamada constantemente a intervalos de milissegundos e tem o dever de realizar qualquer tipo de atualização na cena. Como ela é chamada constantemente, implementou-se uma máquina de estados para controlar esse método de acordo com a etapa da aplicação. Logo, o primeiro ciclo deste método é a própria detecção do plano horizontal, chamado de *placeGrid()*. *Grid* remete ao campo quadriculado da cena.

Este ciclo começa com a função intrínseca ao ARKit, chamada de *hitTest*, a qual procura por objetos ou superfícies do mundo real detectados por meio do processamento da imagem da câmera. No caso, procura por *feature points* que representem um plano horizontal. Esta função retorna um vetor contendo diversas informações sobre o ponto, porém foi utilizada apenas a matriz 4x4 *worldTransform*, a qual contém as informações de translação e orientação do objeto ou superfície detectada. Dada a origem do sistema de coordenadas como a posição inicial do celular, a própria translação é a posição do objeto, dado pelos elementos M_{41}, M_{42} e M_{43}. Em seguida utiliza-se um *SCNNode* (objeto que representa um elemento gráfico da cena)

apelidado de *trackerNode*, para apontar para o usuário onde exatamente a superfície se encontra. Por fim, se atualiza então a posição do *trackerNode* para a posição da superfície quando houver um conjunto maior do que 45 *feature points*, número empiricamente testado que oferece uma estabilidade relativamente boa. Esta parte do código é ilustrada na Figura 4.3.

Figura 4.3: Código para detecção do plan

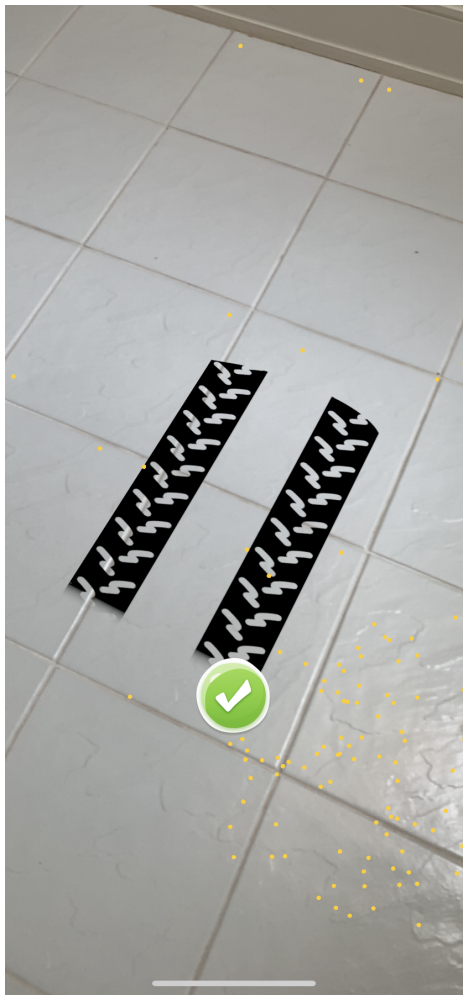
```
/// Primeiro ciclo: só executado enquanto a grid não foi posicionada, há detecção de plano
func placeGrid () {
    // Me interessa apenas feature points que resultem em um plano horizontal - hitTest ARKit
    let hitTest = self.sceneView.hitTest(CGPoint(x: self.frame.midX, y: self.frame.midY), types:
        [.estimatedHorizontalPlane])
    // Se não houver feature points de plano horizontal, sai da função
    guard let result = hitTest.last else { return }

    // Pega a posição do feature points em relação a coordenada do celular
    let translation = SCNMatrix4(result.worldTransform)
    let position = SCNVector3Make(translation.m41, translation.m42, translation.m43)

    // Só considero que houve detecção se houve mais do que 35 feature points
    if (sceneView.session.currentFrame?.rawFeaturePoints?.points.count)! >= 45 {
        // Tracker node e accept btn aparece. Pode-se posicionar a grid.
        scene.trackerNode.isHidden = false
        changeAcceptBtnIsHiddenStatus(to: false)
        // Atualizo a âncora da grid
        scene.gridAnchor = position
    } else {
        // Como a função é chamada várias vezes, aqui eu retorno o tracker para estado de procurando
            e escondo o botão
        scene.trackerNode.isHidden = true
        changeAcceptBtnIsHiddenStatus(to: true)
    }

    self.scene.trackerNode.position = position
}
```

Fonte: O autor

Importante ressaltar que diferentemente de outros *apps* de RA tais como *GeoGebra Augmented Reality*, este projeto não utiliza nenhuma forma de texto para apontar para o usuário onde o plano se encontra, pois o projeto tem também como objetivo a inclusão de crianças analfabetas. Portanto, ao invés de textos, utilizou-se uma animação de marcas de pneu para apontar a localização no mundo real do carrinho FEPSE. Tal animação é definida pela classe *MaterialSpriteScene*, a qual cria uma cena utilizando o SpriteKit, *framework* da Apple para criação de jogos 2D. A animação se dá por uma sequência de fotos. Esta cena é então utilizada como material para criação do objeto *trackerNode*, em detrimento de uma imagem estática.

O resultado final é ilustrado na Figura 4.4.

Figura 4.4: Etapa 1. Animação de marcas de pneu apontam superfície plana detectada. *Feature points* em amarelo



Fonte: O autor

4.4 Posicionamento dos obstáculos

Uma vez posicionada a cena, o aplicativo entra em seu próximo ciclo: posicionamento de obstáculos. Uma vez estabelecida a posição dos obstáculos, estes permanecem "parados" até o próximo ciclo, quando o carrinho finalmente se movimenta.

Assim que ocorre o posicionamento, aparecem 3 objetos: *grid*, *car* e *flag*. O objeto *grid* representa todo o campo pelo qual o carrinho pode se movimentar. A linguagem de programação do FEPSE restringe a movimentação do carrinho para movimentos nos 4 sentidos, logo *grid* possui formato quadriculado. Já *car* e *flag* representam, respectivamente, o carrinho FEPSE e uma bandeira, apontada como "linha de chegada". Na tela são ainda apresentadas outras *Views*, como a mira para o posicionamento, contador de quantos blocos restam e um botão de *reset*.

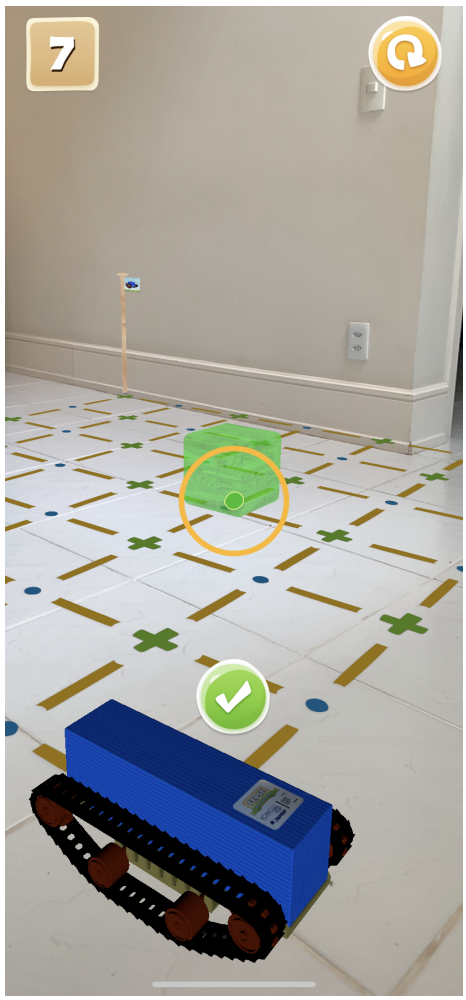
Este ciclo também utiliza a função *hitTest*, porém desta vez com outros parâmetros e objeto de retorno. Desta vez a aplicação projeta o seu ponto central da tela na cena e retorna, caso exista, o objeto pelo qual a projeção atravessa. Sendo assim, verifica-se continuamente se o objeto detectado é o próprio *grid*. Utilizando a propriedade *localCoordinates* obtém-se a posição da projeção em relação ao próprio *grid*. Desta forma é possível analisar se a projeção aponta para algum ponto do campo quadriculado e, em caso positivo, o obstáculo pode ser posicionado.

Quando houver a possibilidade de posicionamento, o objeto *blockPreviewNode* aparece, o qual tem o mesmo formato e tamanho do obstáculo (*blockNode*), porém tem uma leve transparência e é esverdeado. Caso a criança aperte o botão *acceptBtn*, um objeto do tipo *blockNode* é criado na posição do *blockPreviewNode*. Além disso, se o objeto detectado for um obstáculo já posicionado, o próximo obstáculo aparece acima do mesmo.

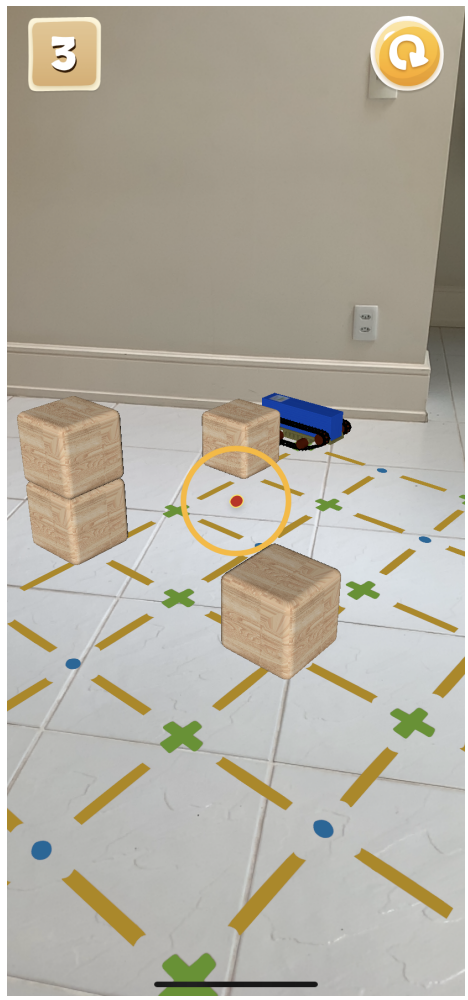
O resultado final é ilustrado na Figura 4.5.

Figura 4.5: Etapa 2, criança posiciona seus próprios obstáculos

(a) Objeto *blockPreviewNode* aparece



(b) Posição não válida



Fonte: O autor

4.5 Visualização da execução do programa

Nesta etapa do programa, com os obstáculos agora posicionados, o carrinho FEPSE irá por fim executar o programa construído a partir da leitura dos cartões ilustrados na Figura 3.3 no momento em que o botão *playBtn* for acionado. Importante ressaltar que futuramente haverá uma etapa prévia à execução do programa, que seria a integração deste projeto com a aplicação *web* do próprio FEPSE, a qual é responsável pela leitura e interpretação dos cartões. Como o escopo do projeto era

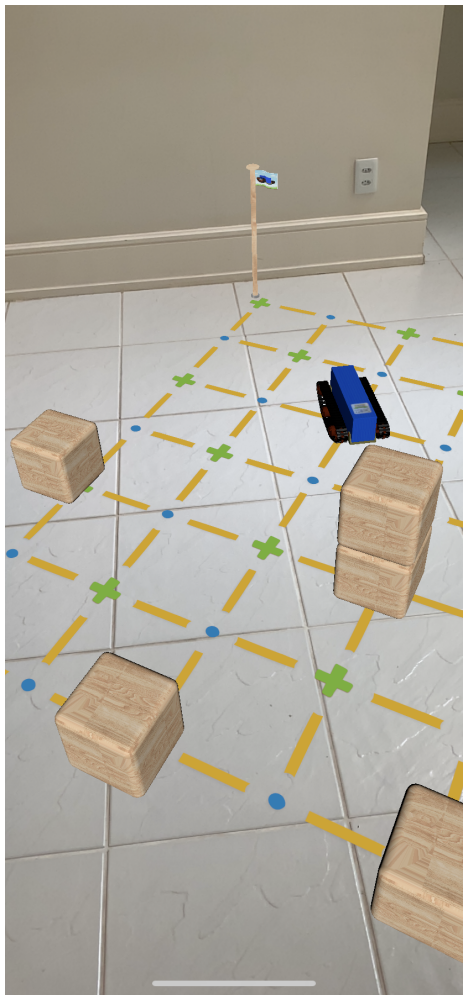
justamente o aprendizado da tecnologia RA, a programação do carrinho foi feita por meio de um *mock* contido no arquivo *Input.txt*.

Neste arquivo há quatro tipos de instruções possíveis: f, b, l, r, que são, respectivamente, *forward, backward, left, right* (frente, trás, direita, esquerda). Cada um destes comandos, quando lidos, acionam funções diferentes as quais fazem o carrinho se mover no sentido desejado independente da sua direção e sentido atual. Diferentemente do carrinho FEPSE real, o objeto deste carro possui uma posição e sentido no sistema de coordenadas iniciado pela *ARSession*. Portanto, instruções como ir para frente ou para trás mudam conforme a sua orientação no sistema de coordenadas XYZ. Utilizando a propriedade *simdWorldFront*, no entanto, é possível obter a orientação do objeto utilizando com referência para frente o eixo $-Z$. A partir desta propriedade, inicia-se uma *custom action* do tipo *SCNAction* para mover o carro no sentido desejado pelo comando. Animações extras também foram adicionadas ao movimento com o intuito de aproximá-lo ao máximo da realidade.

Caso o objeto carro colida com algum obstáculo, o carro irá parar devido a implementação dos métodos do protocolo *SCNPhysicsContactDelegate*. Foi atribuída uma máscara binária de contato diferente para os objetos *car*, *flag* e *blockNode*. Caso qualquer tipo de contato seja detectado, o botão *resetBtn* aparece novamente possibilitando a renovação do ciclo do aplicativo.

O resultado final da última etapa é ilustrado na Figura 4.6.

Figura 4.6: Etapa 3, objeto *car* percorre caminho descrito pelo programa



Fonte: O autor

4.6 Considerações Finais

Neste capítulo, foram apresentadas as três etapas da aplicação FEPSE AR: detecção de plano, posicionamento dos obstáculos e a visualização da execução do programa. Crédito pelos elementos gráficos e tipografia para [41] [42] [43].

O próximo capítulo exibe as conclusões e os possíveis trabalhos a serem realizados no futuro.

Capítulo 5

Conclusões

Este projeto estudou e implementou uma aplicação móvel iOS utilizando uma tecnologia de realidade aumentada, o qual estendeu uma plataforma de ensino de programação para crianças, o FEPSE.

A realidade aumenta foi utilizada com o intuito de auxiliar a visualização das atividades na ausência dos sistemas embarcados. Para garantir familiaridade com a plataforma, o carrinho FEPSE foi modelado em 3D e a mesma identidade visual foi adotada.

Os resultados obtidos na primeira etapa do projeto foram promissores, pois asseguraram a acessibilidade da aplicação para crianças analfabetas, um dos valores base da plataforma FEPSE. O posicionamento de obstáculos foi construído de forma a explorar a interatividade proporcionada pela tecnologia, permitindo que a criança se movimente pelo campo. Já as animações realizadas pelo carrinho dentro da plataforma visaram se aproximar ao máximo do movimento real.

A aplicação atua, portanto, como prova de conceito de que a realidade aumentada pode ser aplicada a esta plataforma. Terminada a integração deste aplicativo móvel com a sua aplicação *web*, a validação do projeto nas escolas parceiras poderá

ser realizada.

Por fim, espera-se que este trabalho contribua para aqueles que gostariam de ter mais contato com aplicações em realidade aumentada. Apesar do fato das aplicações em RA serem estudadas popularmente desde a década de 1990, não há ainda um volume substancial de estudos sobre sua aplicabilidade na área da educação. Há aqueles que defendem que RA diminui o esforço cognitivo, outros dizem que RA promove sobrecarga [4].

O que se sabe, entretanto, é que as maiores empresas de tecnologia estão realizando investimentos enormes na área, seja em realidade aumentada ou virtual. Facebook (Oculos), Sony (Playstation) e Samsung (Gear) possuem todas suas próprias versões de óculos de realidade virtual. Microsoft com HoloLens 2 e Google com Glass Enterprise entre outros são grandes nomes para os óculos de realidade aumentada, além do rumor que a Apple irá lançar a sua própria versão em breve [31].

De acordo com a empresa MarketsandMarkets, o valor do mercado de RA vai saltar dos 4,21 bilhões de dólares em 2017 para 60,55 bilhões em 2023 [44]. Definitivamente, muito se falará sobre realidade aumentada nos próximos anos.

5.1 Trabalhos Futuros

A primeira funcionalidade a ser desenvolvida futuramente seria a integração deste aplicativo móvel com a aplicação *web* FEPSE. Desta forma, este projeto estaria pronto para ser testado nas escolas parceiras da plataforma, pois as crianças seriam capazes de realizar a leitura dos cartões e ver em tempo real qual instrução está sendo executada.

Existem também outras funcionalidades do *framework* ARKit que não foram exploradas por este projeto que, futuramente, poderiam agregar valor à aplicação. A primeira delas seria o *multiplayer*, ou seja, quando dois dispositivos móveis compartilham a mesma cena. Isso possibilitará que duas crianças (ou mais) compartilhem a mesma instância do campo e posicionem, juntas, os obstáculos.

Na **WWDC19** (*Apple Worldwide Developers Conference 2019*) foi anunciada a nova versão do ARKit, o ARKit 3, e um novo *framework* voltado para RA, RealityKit. Além do ARKit 3 possuir melhorias internas no processamento da imagem (o que melhora a detecção do plano), foram adicionadas funcionalidades como oclusão de pessoas e captura de movimento de pessoas em tempo real. Já o RealityKit é uma ferramenta nativa para tornar os objetos virtuais mais próximos da realidade, principalmente através de animações em 3D.

Utilizar então o RealityKit neste projeto para criação dos objetos virtuais poderia tornar o *app* mais imersivo. Atualizar para a nova versão do ARKit poderia abrir novas possibilidades de interação: oclusão de pessoas contribuiria em aproximá-lo da realidade, enquanto que a captura dos movimentos poderia permitir que as crianças posicionassem ou criassem os obstáculos de forma mais lúdica.

Referências Bibliográficas

[1] KRALEVA, R.; KRALEV, V.; KOSTADINOVA, D. Investigating some programming languages for children to 8 years. In: . c2016. v. 5. p. 4–6.

[2] LEITE, K. L. Fepse - framework de ensino de programação e sistemas embarcados. https://fepse.com.br, 2019. Acesso em 01 de Outubro de 2019.

[3] LEITE, K. L. Plataforma de ensino de programação para crianças e jovens usando sistemas embarcados, storytelling, computação tangível e programação visual. https://fepse.com.br/docs/FAPESP_FEPSE_Kaique_Lupo.pdf, 2016. Acesso em 01 de Outubro de 2019.

[4] AKÇAYIR, M.; AKÇAYIR, G. Advantages and challenges associated with augmented reality for education: A systematic review of the literature. *Educational Research Review*, v. 20, p. 1–11, 2017.

[5] AZUMA, R. T. A survey of augmented reality. *Presence: Teleoperators & Virtual Environments*, v. 6, n. 4, p. 355–385, 1997.

[6] WU, H.-K.; LEE, S. W.-Y.; CHANG, H.-Y.; LIANG, J.-C. Current status, opportunities and challenges of augmented reality in education. *Computers & education*, v. 62, p. 41–49, 2013.

[7] BERRYMAN, D. R. Augmented reality: a review. *Medical reference services quarterly*, v. 31, n. 2, p. 212–218, 2012.

[8] SUTHERLAND, I. E. A head-mounted three dimensional display. In: . c1968. p. 757–764.

[9] CAUDELL, T. P.; MIZELL, D. W. Augmented reality: An application of heads-up display technology to manual manufacturing processes. In: . c1992. v. 2. p. 659–669.

[10] ROSENBERG, L. B. Virtual fixtures: Perceptual tools for telerobotic manipulation. In: . c1993. p. 76–82.

[11] FEINER, S.; MACINTYRE, B.; SELIGMANN, D. Knowledge-based augmented reality. *Communications of the ACM*, v. 36, n. 7, p. 53–62, 1993.

[12] COLUMBIA. Steven feiner and salvatore stolfo elevated to ieee fellow. https://www.cs.columbia.edu/2017/steven-feiner-and-salvatore-stolfo-elevated-to-ieee-fellow/, 2019. Acesso em 09 de Outubro de 2019.

[13] FEINER, S.; MACINTYRE, B.; HÖLLERER, T.; WEBSTER, A. A touring machine: Prototyping 3d mobile augmented reality systems for exploring the urban environment. *Personal Technologies*, v. 1, n. 4, p. 208–217, 1997.

[14] REKIMOTO, J.; AYATSUKA, Y. Cybercode: designing augmented reality environments with visual tags. In: . c2000. p. 1–10.

[15] SONY. Cybercode. https://www.sonycsl.co.jp/tokyo/320/, 2019. Acesso em 09 de Outubro de 2019.

[16] OLNEY, A. Augmented reality. *Beyond Reality: Augmented, Virtual, and Mixed Reality in the Library*, p. 1, 2019.

[17] HOFHEIMER, B. Virtual yellow 1st and ten line debuted on espn 15 years ago today. https://www.espnfrontrow.com/2013/09/virtual-yellow-1st-and-ten-line-debuted-on-espn-15-years-ago-today/, 2019. Acesso em 05 de Outubro de 2019.

[18] KATO, H.; BILLINGHURST, M. Marker tracking and hmd calibration for a video-based augmented reality conferencing system. In: . c1999. p. 85–94.

[19] ARTOOLKIT. Aartoolkit. https://github.com/artoolkit, 2019. Acesso em 3 de Setembro de 2019.

[20] PIEKARSKI, W.; THOMAS, B. Arquake: the outdoor augmented reality gaming system. *Communications of the ACM*, v. 45, n. 1, p. 36–38, 2002.

[21] BMW. The convertible in your hands! https://www.press.bmwgroup.com/middle-east/article/detail/T0048960EN, 2008. Acesso em 12 de Outubro de 2019.

[22] NIANTIC. A história da niantic. https://nianticlabs.com/pt_br/about/, 2019. Acesso em 12 de Outubro de 2019.

[23] GUARDIAN, T. The rebirth of google glass shows the merit of failure. https://www.theguardian.com/commentisfree/2017/jul/23/the-return-of-google-glass-surprising-merit-in-failure-enterprise-edition, 2019. Acesso em 10 de Outubro de 2019.

[24] GOOGLE. Glass enterprise edition 2: faster and more helpful. https://www.blog.google/products/hardware/glass-enterprise-edition-2/, 2019. Acesso em 10 de Outubro de 2019.

[25] TV, M. B. A. Microsoft hololens 2 demo - mobile world congress 2019 (julia schwarz). https://www.youtube.com/watch?v=8wHC2Rb46H4&t=2s, 2019. Acesso em 10 de Outubro de 2019.

[26] WIRED. Microsoft's hololens 2 puts a full-fledged computer on your face. https://www.wired.com/story/microsoft-hololens-2-headset/, 2019. Acesso em 10 de Outubro de 2019.

[27] CHAMARY, J. Why 'pokémon go' is the world's most important game. https://www.forbes.com/sites/jvchamary/2018/02/10/pokemon-go-

science-health-benefits/#3e3743153ab0, 2018. Acesso em 12 de Outubro de 2019.

[28] ALTHOFF, T.; WHITE, R. W.; HORVITZ, E. Influence of pokémon go on physical activity: study and implications. *Journal of medical Internet research*, v. 18, n. 12, p. e315, 2016.

[29] 3D4MEDICAL. Complete anatomy 2020. https://3d4medical.com, 2019. Acesso em 12 de Outubro de 2019.

[30] ARTOOLKIT. Bring math to life with apple arkit. https://www.geogebra.org/m/R8Qd7U8y, 2019. Acesso em 17 de Setembro de 2019.

[31] FINUCANE, B. The next technological revolution: Investing in augmented and virtual reality. https://financial-news-now.com/the-next-technological-revolution-investing-in-augmented-and-virtual-reality/, 2019. Acesso em 1 de Novembro de 2019.

[32] APPLE. ios technology overview. http://pooh.poly.asu.edu/Mobile/ClassNotes/Papers/MobilePlatforms/iOSTechnicalOverview.pdf, 2012. Acesso em 5 de Agosto de 2019.

[33] APPLE. About app development with uikit. https://developer.apple.com/documentation/uikit/about_app_development_with_uikit#3004320, 2019. Acesso em 5 de Agosto de 2019.

[34] APPLE. Model-view-controller. https://developer.apple.com/library/archive/documentation/General/Conceptual/DevPedia-CocoaCore/MVC.html#//apple_ref/doc/uid/TP40008195-CH32-SW1, 2018. Acesso em 6 de Agosto de 2019.

[35] APPLE. Arkit — apple developer documentation. https://developer.apple.com/documentation/arkit, 2019. Acesso em 01 de Julho de 2019.

[36] APPLE. Understanding world tracking. https://developer.apple.com/documentation/arkit/understanding_world_tracking, 2019. Acesso em 15 de Outubro de 2019.

[37] RAYWENDERLICH. Augmented reality and arkit tutorial. https://www.raywenderlich.com/378-augmented-reality-and-arkit-tutorial, 2017. Acesso em 15 de Outubro de 2019.

[38] RAYWENDERLICH. Getting started with arkit with swift 4 - xcode 9, ios 11 - augmented reality in swift. https://www.youtube.com/watch?v=zcwPnU-XVtQ, 2018. Acesso em 15 de Outubro de 2019.

[39] APPLE. Augmented reality. https://www.apple.com/lae/ios/augmented-reality/, 2019. Acesso em 6 de Agosto de 2019.

[40] APPLE. Apple introduces iphone 6s and iphone 6s plus. https://www.apple.com/newsroom/2015/09/09Apple-Introduces-iPhone-6s-iPhone-6s-Plus/, 2015. Acesso em 6 de Agosto de 2019.

[41] STARLINE. Background assets. https://www.freepik.com/free-photos-vectors/background, 2019. Acesso em 1 de Outubro de 2019.

[42] GRAPHICBURGER. Mobile game ui. https://graphicburger.com/mobile-game-gui/, 2019. Acesso em 1 de Outubro de 2019.

[43] ADAMS, V. Carter one. https://fonts.google.com/specimen/Carter+One, 2019. Acesso em 1 de Outubro de 2019.

[44] MARKETSANDMARKETS. Augmented reality market. https://www.marketsandmarkets.com/Market-Reports/augmented-reality-market-82758548.html, 2019. Acesso em 1 de Novembro de 2019.