UNIVERSIDADE REGIONAL DE BLUMENAU

CENTRO DE CIÊNCIAS EXATAS E NATURAIS

CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

TECNOLOGIA ASSISTIVA: TORNANDO JOGO DE MESA ACESSÍVEL PARA CEGOS COM AUXÍLIO DE APLICATIVO MÓVEL DE RECONHECIMENTO DE IMAGEM

RONAN GUIMARÃES KRAEMER

BLUMENAU
2017

RONAN GUIMARÃES KRAEMER

TECNOLOGIA ASSISTIVA: TORNANDO JOGO DE MESA ACESSÍVEL PARA CEGOS COM AUXÍLIO DE APLICATIVO MÓVEL DE RECONHECIMENTO DE IMAGEM

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.

Dalton Solano dos Reis - Orientador

BLUMENAU
2017

TECNOLOGIA ASSISTIVA: TORNANDO JOGO DE MESA ACESSÍVEL PARA CEGOS COM AUXÍLIO DE APLICATIVO MÓVEL DE RECONHECIMENTO DE IMAGEM

Por

RONAN GUIMARÃES KRAEMER

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(a). Dalton Solano dos Reis, M.Sc. – Orientador, FURB
Membro:	Prof(a). Joyce Martins, M.Sc. – FURB
Membro:	Prof(a). Gilvan Justino, M.Sc. – FURB

Blumenau, 06 de julho de 2017

Dedico este trabalho para todos aqueles que ainda sofrem e buscam por algum tipo de inclusão social. A todos que, mesmo com suas limitações, não desistem da vida e batalham dia após dia para superar seus obstáculos com muita garra e perseverança.

AGRADECIMENTOS

A Deus, pelo seu imenso amor e graça.

À minha família, que me apoiou e sempre esteve presente.

Aos meus amigos, pelos empurrões e cobranças.

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

Você perde 100% das chances que não aproveita.

Wayne Gretzky

RESUMO

Este trabalho apresenta o desenvolvimento de uma tecnologia assistiva para garantir a acessibilidade de pessoas cegas ao jogo de mesa Munchkin. Os usuários poderão utilizar o aplicativo desenvolvido com tecnologia de multiplataformas para dispositivos móveis e disponível para Android e iOS. Com isso, poderão usufruir e participar do jogo, garantindo que tenham a oportunidade de interagir e serem incluídas de forma igualitária no grupo de jogadores. O aplicativo permite que seja tirada uma foto da carta que o jogador estiver segurando e, através do uso da internet, irá reconhecer e retornará ao usuário, na forma de voz, a descrição da mesma. O desenvolvimento do aplicativo também contou com uma pesquisa de tecnologias atuais para o uso em conjunto do *framework* Ionic para extrair e garantir o melhor desempenho e precisão ao usuário final. Além do Ionic, o aplicativo também foi construído utilizando as tecnologias web (HTML, CSS e Javascript), auxílio da biblioteca AngularJS e o serviço Google Cloud Vision API (2017). Ao final deste trabalho, constatou-se que os resultados foram muito satisfatórios com relação ao uso das tecnologias envolvidas e a precisão do aplicativo ao efetuar o reconhecimento das cartas.

Palavras-chave: Ionic. Munchkin. Multiplataformas. Acessibilidade. Google Cloud.

ABSTRACT

This work presents an assistive technology development to grant accessibility for people with vision impairment to the Munchkin board game. The users will have the possibility of using the app developed with multiplatform technology for mobile devices and available for Android and iOS. With this, they will have the opportunity to enjoy, participate, interact and being included equally in the player's party. The application allows a photo to be taken of the card that the player is holding and, using the internet, will recognize and return to the user, in voice form, the description of the same. This project also featured current technology research to jointly use the Ionic framework to extract and ensure the best performance and accuracy to the end user. Besides the Ionic, the application was also built using web technologies (HTML, CSS and Javascript), help from the AngularJS library and the Google Cloud Vision API (2017) service. At the end of this work, it was found that the results were very satisfactory in relation to the use of the technologies involved and the set accuracy plus performance of the application when performing the cards recognition.

Key-words: Ionic. Munchkin. Multiplatform. Accessibility. Google Cloud.

LISTA DE FIGURAS

LISTA DE QUADROS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

API – Application Programming Interface (Interface de Programação de Aplicações)

CAT – Comitê de Ajudas Técnicas

CES2017 – Consumer Electronic Show 2017

DNs – Digital Numbers

HOG – Histogram of Oriented Gradients

HTML5 – Hypertext Markup Language 5

IA – Inteligência Artificial

OCR – Optical Character Recognition (Reconhecimento óptico de caracteres)

RPG – Rolling-Playing Game

SVM – Support Vector Machines

TA – Tecnologia Assistiva

TTS – Text-To-Speech

SUMÁRIO

1 INTRODUÇÃO

Já é de domínio público que a tecnologia do mundo está crescendo de forma muito rápida. Visivelmente este crescimento acelerado tem sido bastante consistente, ou seja, é bem possível que se passem vários anos até que haja alguma desaceleração neste ponto. Da mesma forma como a tecnologia vêm crescendo, a inclusão social também tem tido sua importância mais valorizada e discutida. “Nos dias atuais, a inclusão é, sem dúvida, uma questão central em todos os ambientes em que vivemos: nas famílias, nas escolas, no mercado de trabalho, nos espaços de lazer, enfim, em todas as situações da vida do ser humano” (AMIRALIAN, 2009 apud FERRONI; GASPARETTO, 2011, p. 1092).

Existe uma grande preocupação no mundo quanto à acessibilidade e, no Brasil, este assunto é levado tão a sério que existe um comitê unicamente para os assuntos de tecnologia e acessibilidade. Em 16 de novembro de 2006 foi instituído, pela Portaria nº 142, o Comitê de Ajudas Técnicas (CAT), estabelecido pelo Decreto nº 5.296/2004 para aperfeiçoar, dar transparência e legitimidade ao desenvolvimento da Tecnologia Assistiva (TA) (BRASIL - SDHPR, 2009, p. 9).

O CAT define TA como uma área do conhecimento que engloba produtos, recursos, metodologias, estratégias, práticas e serviços. Tem como objetivo promover a funcionalidade de pessoas com deficiência, incapacidades ou mobilidade reduzida, visando sua autonomia, independência, qualidade de vida e inclusão social (BRASIL - SDHPR, 2009). Dentre os tipos de ferramentas mais utilizadas para a criação de tecnologias assistivas, é possível identificar as ferramentas de reconhecimento de imagem e texto. Tais ferramentas são utilizadas para o desenvolvimento de sistemas para auxílio de pessoas com deficiência visual no dia a dia, desde a leitura de uma placa até para promover maior segurança em um trajeto de caminhada.

Com o crescimento destas tecnologias, uma maior abrangência de soluções de acessibilidade deve ser considerada, como, por exemplo, jogos competitivos ou de grupo que envolvem cartas, tabuleiros e interação com objetos físicos. Estes tipos de jogos normalmente são denominados como “jogos de mesa” (vulgo *board games*). Um exemplo de jogo de mesa é o Munchkin, onde os jogadores utilizam um baralho para simular uma exploração em cavernas atrás de tesouros e enfrentam “monstros” com o objetivo de chegar no último nível. As cartas têm suas descrições e efeitos diversos e o jogador deve saber quando guardá-las e/ou utilizá-las.

Criar tecnologias capazes de dar maior acessibilidade para estes tipos de jogos pode ser algo complexo. Em especial para pessoas cegas, onde a abstração de situações é muito maior

e a necessidade de balanceamento, para evitar que algum jogador tenha vantagens sobre outros, deve ser considerada. Neste contexto, o foco deste trabalho foi criar uma solução para pessoas com deficiência visual, que tinha como objetivo a pesquisa e utilização de tecnologias atuais de identificação de imagens e textos, para implementar um aplicativo de celular capaz de permitir que uma pessoa com dificuldades visuais possa jogar um jogo de mesa.

1.1 OBJETIVOS

Este trabalho objetiva criar uma aplicação móvel para permitir que pessoas cegas possam participar de uma partida de Munchkin.

Os objetivos específicos são:

a) desenvolver o aplicativo de forma que possa ser utilizado ao menos em duas plataformas diferentes (Android e iOS) utilizando a ferramenta Ionic;
b) adaptar o jogo físico para aumentar a jogabilidade para o deficiente visual sem alterar as características do próprio jogo;
c) fazer com que a aplicação seja de acesso gratuito.

1.2 ESTRUTURA

Para melhor organização e entendimento do leitor, este trabalho está separado em quatro capítulos. Iniciando o primeiro capítulo com a introdução ao tema do trabalho, seguido do segundo capítulo no qual estão descritos os fundamentos básicos para o entendimento das técnicas e ferramentas utilizadas. O terceiro capítulo apresenta informações sobre a construção do aplicativo como diagramas, arquitetura, serviços utilizados e o código fonte. As últimas seções trazem as telas do aplicativo demonstrando sua operacionalidade, os resultados obtidos após o desenvolvimento e comparação entre os trabalhos correlatos. No quarto e último capítulo estão as conclusões finais e sugestões para o desenvolvimento de trabalhos futuros.

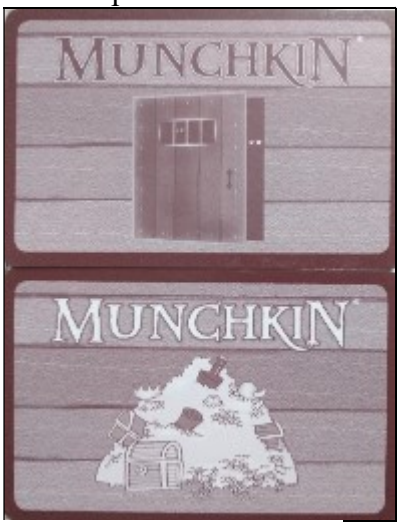
2 FUNDAMENTAÇÃO TEÓRICA

Na seção 2.1 é apresentada uma explicação sobre o jogo Munchkin. Na seção 2.2, é apresentado o conceito de funcionamento dos algoritmos de OCR. A seção 2.3 demonstra a importância do tratamento de imagem para o melhor funcionamento do OCR. A seção 2.4 faz uma apresentação ao Ionic *framework* e alguns de seus *plugins* utilizados neste trabalho. E, por fim, a seção 2.5 apresenta três trabalhos correlatos.

2.1 MUNCHKIN

Munchkin é um jogo de cartas orginalmente criado pela empresa americana Steve Jackson Games e hoje é traduzido e distribuído no Brasil pela Galápagos Jogos. De acordo com a Galápagos Jogos (2012), este jogo contém a essência da experiência de jogos de exploração de cavernas, sem toda a complexidade do Rolling-Playing Game (RPG). Embora a base do jogo esteja nas cartas, Munchkin conta muito com a capacidade de comunicação e negociação dos jogadores onde todos têm liberdade para ajudar ou atrapalhar o colega. As cartas são divididas em dois tipos, sendo denominados “portas” e “tesouros” (Figura 1).

Figura 1 – Tipos de cartas de Munchkin



Fonte: Galápagos Jogos (2012).

No início do jogo, cada jogador recebe quatro cartas de tesouro e quatro cartas de porta, sendo este m dos únicos momentos em que ninguém irá saber as cartas que o outro tem em mãos, porém todos têm a liberdade de abrir o jogo se assim desejarem. A maior proposta deste jogo é o incentivo de comunicação, onde o jogador precisa usar, além da interpretação, habilidades de persuasão, negociação e lógica para garantir que todos os eventos do jogo sirvam para direcioná-lo a vitória. O jogo segue uma lógica de turnos, onde cada jogador tem sua chance de entrar em uma caverna e descobrir o que existe dentro dela. Existe um único

detalhe que permite a todos os outros jogadores a opção de interferir no turno do jogador da vez, seja de forma positiva ou negativa:

a) positiva: ajudando-o a derrotar um monstro muito forte ou lhe dando um bônus para facilitar passar pela caverna em troca de uma compensação pela ajuda;
b) negativa: impedindo-o de derrotar um monstro que lhe daria muita vantagem com relação aos outros jogadores e diminuindo as chances dele de vitória.

Dentro da divisão de cartas de porta e tesouro, existem mais subdivisões de tipos de cartas, como cartas de maldição, classes, raças, equipamentos, poções, efeitos, entre outros, porém todas as cartas do jogo têm o mesmo padrão visual representado na Figura 2. As cartas de porta servem para simular a entrada nas cavernas onde o jogador pode encontrar um monstro, ou uma maldição, ou uma classe, ou uma raça ou simplesmente uma habilidade que poderá ser utilizada no decorrer do jogo. Quando o jogador encontra um monstro ele tem as opções de enfrentá-lo ou fugir. Ao derrotar o monstro, com ou sem ajuda de outra pessoa, o jogador tem direito a acessar os tesouros que o monstro estava guardando, sendo normalmente uma carta do tipo tesouro. Cada carta tem seu efeito e usabilidade, cabendo ao jogador decidir quais manter e quando utilizá-las. Diante de tanta variedade de cartas, é possível identificar um padrão entre todas elas. Todas as cartas do jogo têm o mesmo layout variando apenas em alguns modelos de cores e detalhes.

Figura 2 – Modelo de carta de Munchkin

Fonte: Galápagos Jogos (2012).

Todas as cartas possuem dois losangos presentes na parte superior logo acima da região onde se encontra o título, que é o que se deseja reconhecer com o algoritmo de OCR. Assim, este padrão poderá ser utilizado para definir as características para identificação e mapeamento das cartas. Utilizando estes losangos como ponto de referência, será possível

identificar a posição da carta e efetuar os devidos tratamentos na imagem para que a aplicação tenha o melhor desempenho possível.

2.2 RECONHECIMENTO ÓPTICO DE CARACTERES (OCR)

"Reconhecimento óptico de caracteres pertence à família de técnicas de execução automática de identificação" (EIKVIL, 1993, p. 5, tradução nossa). Segundo Fujisawa (2007), a execução do OCR consiste no processo de identificação de caracteres existentes em imagens e conversão destes caracteres para o formato de texto, podendo ser utilizado posteriormente em diversas aplicações.

Segundo Eikvil (1993, p. 7) o OCR é necessário quando a informação precisa ser legível tanto por humanos quanto por máquinas e as entradas alternativas para solução deste problema não são pré-definíveis. O reconhecimento de caracteres poderá ser realizado para casos de impressões ou até mesmo escrita à mão, porém a performance dependerá diretamente da qualidade do documento de entrada para o processamento. O OCR pode ser processado de duas formas diferentes:

a) *on-line:* quando a aplicação é capaz de efetuar o processamento dos caracteres enquanto são desenhados ou impressos;
b) *off-line:* quando o processamento de OCR é realizado após o objeto desejado ter sido identificado (foto ou escaneamento).

Eikvil (1993, p. 11) explica e demonstra (Figura 3) que um sistema de OCR consiste em vários componentes:

a) escaneamento óptico: é a etapa de aquisição de entrada para o processamento do OCR, normalmente adquiridas com o uso de dispositivos fotográficos. Quando executando o OCR, é comum a prática de converter uma imagem com diversos níveis de cores em uma imagem com apenas dois níveis de cores (preto e branco);
b) localização e segmentação: é o processo que determina as áreas de texto da imagem e serve para diferenciá-las das áreas de figuras e gráficos. Aplicado em textos, é a isolação de caracteres ou palavras. Esta etapa pode apresentar alguns problemas que são divididos em quatro grupos:
    - extração de caracteres unidos e fragmentados, fazem com que uma parte de um caractere ou dois caracteres muito unidos sejam interpretados como um símbolo completo,
    - distinção de ruídos do texto, onde pontos e acentos podem ser marcados como ruído, assim como ruídos podem ser interpretados como pontos e acentos,

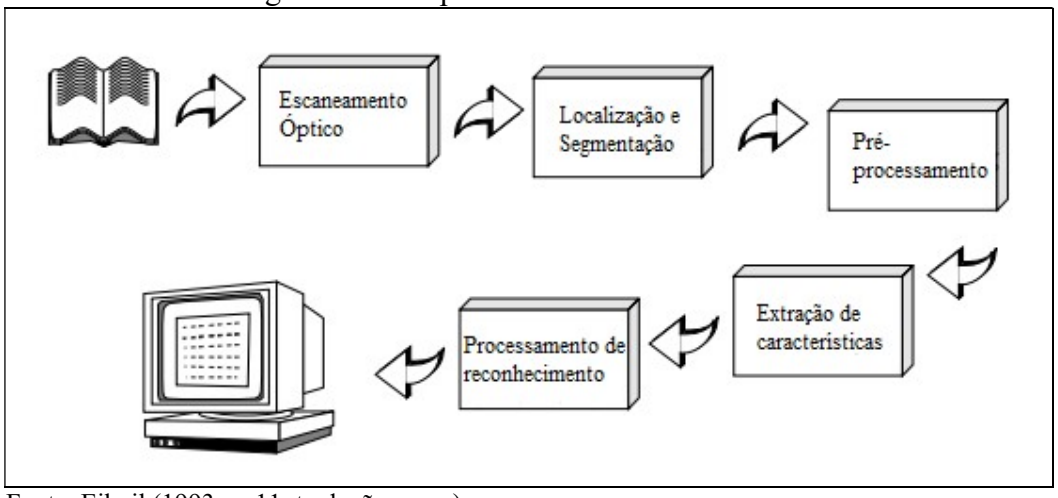
- interpretação de partes gráficas ou geométricas como texto, acontece com partes sem texto sendo reconhecidas como texto pelo algoritmo,
- interpretação de texto como partes gráficas ou geométricas, normalmente acontece com caracteres que estão conectados com partes gráficas da imagem que acabam sendo ignorados pelo algoritmo;

c) pré-processamento*:* neste ponto, a imagem resultante pode conter alguns ruídos e para evitar a baixa qualidade de reconhecimento é realizado um processo de suavização dos caracteres. Normalmente, além da suavização, também é realizada a normalização dos caracteres para que todos tenham um tamanho uniforme;

d) extração de características: consiste em extrair as características essenciais dos símbolos reconhecidos. "A forma mais direta de descrever o caractere é pela leitura dos pixels que a compõem" (FERREIRA, 2014, p. 7). Outra forma de executar esta etapa é realizando a extração de algumas características mais importantes que determinam o caractere e deixando de lado os atributos menos importantes;

e) processamento de reconhecimento: neste ponto o algoritmo possui um conjunto de símbolos individuais e é necessário realizar um agrupamento para que sejam identificadas as palavras e números. Este agrupamento é realizado com base na localização dos símbolos no documento onde símbolos que estão suficientemente perto são juntamente agrupados. Nesta etapa também é executado o processo de detecção e correção de erros, onde os erros gramaticais são detectados e corrigidos com base nas regras gramaticais da linguagem em uso.

Figura 3 – Componentes de um sistema OCR



Fonte: Eikvil (1993, p. 11, tradução nossa).

Ferreira (2014, p. 8) acrescenta que “o texto obtido pode ser utilizado para impressão, para introdução em outros documentos ou num processo de sintetização de voz [...]. Esta última utilização possibilita [...] o acesso à informação sem a necessidade de intervenção de outras pessoas”.

2.3 PROCESSAMENTO DE IMAGEM

Processamento de imagem é qualquer forma de processamento de dados, onde a entrada e saída são imagens digitais oriundas de qualquer tipo de dispositivo fotográfico ou digitalizador. Uma parte importante para este trabalho é a definição de um ponto de interesse (mapeamento) nas cartas de Munchkin e, segundo UFRGS (2008), este ponto é caracterizado “em função das propriedades dos objetos ou padrões que compõem a imagem. Portanto, extrair informação da imagem requer reconhecimento de objetos ou padrões”.

Dentre as diversas formas de se realizar o reconhecimento de padrões, existe uma abordagem denominada casamento de padrões que, segundo Araújo (2009, p. 40), consiste em determinar a similaridade entre duas entidades do mesmo tipo comparando a imagem que está sendo analisada com a imagem que contém o padrão de referência (*template*). Esta abordagem tem grande potencial, pois, como demonstrado anteriormente, as cartas de Munchkin possuem um padrão bastante visível, contendo até mesmo um modelo em branco das mesmas que poderá ser utilizado como referência.

No que diz respeito ao pré-processamento da imagem para execução de OCR, Silva (2001 apud UFRGS, 2008) explica que, em processamento de imagens, comumente trabalha-se sempre com tons de cinzas (Digital Numbers - DNs) atribuídos aos pixels de uma imagem. O histograma é uma das formas mais comuns de se representar a distribuição dos DNs de uma imagem e possivelmente a mais útil em processamento digital de imagens.

> O histograma de uma imagem é simplesmente um conjunto de números indicando o percentual de pixels naquela imagem que apresentam um determinado nível de cinza. Estes valores são normalmente representados por um gráfico de barras que fornece para cada nível de cinza o número (ou percentual) de pixels correspondentes na imagem. Através da visualização do histograma de uma imagem obtemos uma indicação de sua qualidade quanto ao nível de contraste e quanto ao seu brilho médio (FILHO; NETO, 1999, p. 55).

Saber a predominância de tonalidade da imagem é bastante importante para definir o tipo de tratamento que a imagem irá receber afim de melhorar ao máximo a precisão do algoritmo de OCR. Os tratamentos de imagem mais comuns utilizados, através do histograma, são:

a) negativo: “é uma função de mapeamento linear inversa, ou seja, o contraste ocorre

de modo que as áreas escuras (baixos valores de nível de cinza) se tornam claras (altos valores de nível de cinza) e vice-versa" (DPI, 2009, p. 1);

b) binarização: "determina-se um valor de limiar e todos os valores dos pixels menores ou iguais a esse valor são mapeados em 0, enquanto os demais são mapeados em 1" (FILHO; NETO, 1999, p. 303);
c) brilho: "adicionando-se um valor constante a todos os níveis de cinza consegue-se deslocar sua média, tornando-a mais clara" (CENTENO, 2004 apud SILVA; RIBEIRO, 2009);
d) expansão: "o histograma original de uma imagem é modificado de tal forma que parte dele é expandida para ocupar toda a faixa de cinza da imagem" (FILHO; NETO, 1999, p. 70);
e) compressão: "modifica o histograma original de uma imagem de tal forma que suas raias passam a ocupar apenas um trecho da faixa total de cinza, produzindo como resultado uma redução de contraste na imagem" (FILHO; NETO, 1999, p. 70).

As técnicas de processamento de imagem são muito importantes para o projeto, pois interferem diretamente na performance do algoritmo de OCR, sendo necessário realizar tratamentos suficientes para garantir a melhor tonalidade nas áreas de interesse e a maior precisão possível para a aplicação.

2.4 IONIC *FRAMEWORK* E *PLUGINS*

Ionic é um *framework* de desenvolvimento de aplicativos para dispositivos móveis baseados em Hypertext Markup Language 5 (HTML5) e visa a criação de aplicações híbridas para dispositivos móveis. Estas aplicações híbridas são, essencialmente, pequenas páginas da web rodando em um *browser shell* dentro de um aplicativo que tem acesso à camada nativa da plataforma (IONICFRAMEWORK, 2017a). O Ionic permite o uso de mais de 120 características nativas dos aparelhos como Bluetooth, HealthKit, FingerPrintAuth e mais com *plugins* de Cordova/PhoneGap e extensões TypeScript (IONICFRAMEWORK, 2017b, tradução nossa).

O uso de *plugins* no desenvolvimento com o Ionic é fundamental, pois existe a necessidade de acessar os recursos nativos do dispositivo móvel. Alguns exemplos de recursos são câmera, *text to speech*, *speech to text* e *file transfer*. O Quadro 1 mostra uma relação de *plugins* disponíveis para cada recurso mencionado.

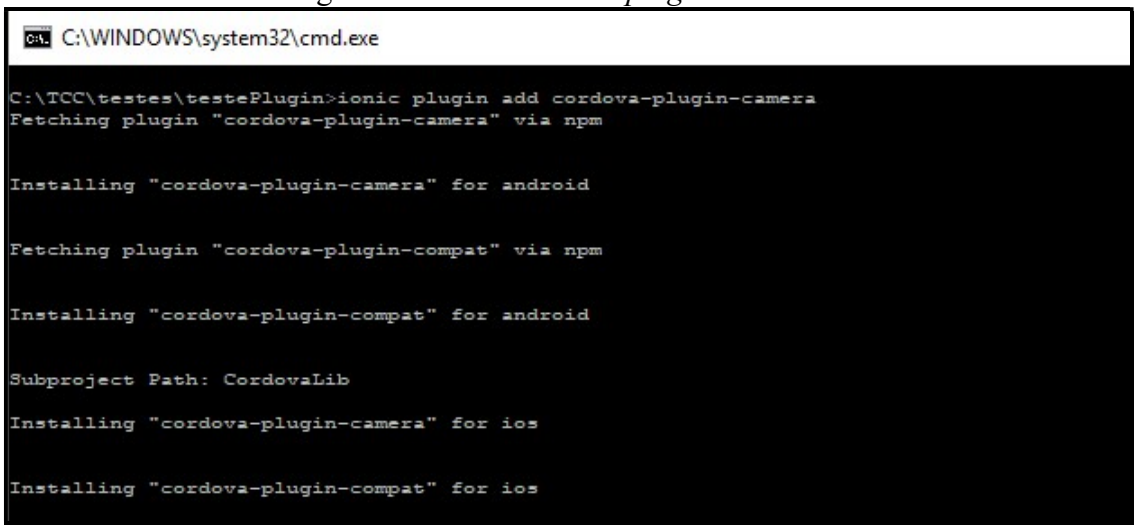
Quadro 1 – Lista de recursos e *plugins*

Recurso	Plugin	Link
câmera	cordova-plugin-camera	github.com/apache/cordova-plugin-camera
text to speech	cordova-plugin-tts	github.com/vilic/cordova-plugin-tts
speech to text	speech-recognition-plugin	github.com/macdonst/SpeechRecognitionPlugin
file transfer	cordova-plugin-file-transfer	github.com/apache/cordova-plugin-file-transfer

Fonte: adaptado de Apache Cordova (2013).

O Ionic possui uma interface via linha de comando, Ionic CLI, que possibilita que os *plugins* sejam adicionados ao projeto. Por exemplo, para adicionar o *plugin* que permite a manipulação da câmera do dispositivo, basta executar o comando `ionic plugin add cordova-plugin-camera` conforme a Figura 4.

Figura 4 – Adicionando o *plugin* da câmera



Fonte: Elaborado pelo autor.

> É possível a criação de *plugins* customizados para que os desenvolvedores possam utilizar todos os recursos do desenvolvimento nativo em uma plataforma híbrida. Um exemplo de uso de *plugin* customizado seria a criação de uma interface que comunica nativamente com o *back-end* da aplicação, na tentativa de proteger os dados de acesso ao serviço. Esses *plugins* devem ser escritos na linguagem nativa de cada plataforma (Adão, 2016, p. 18).

2.5 TRABALHOS CORRELATOS

Entre os trabalhos correlatos está o trabalho de Sousa (2013); o aplicativo Read4Blind (FERREIRA, 2014) e o aplicativo Aipoly (2016).

2.5.1 Uso de visão computacional em dispositivos móveis para reconhecimento de faixas de pedestre

Sousa (2013, p. 12) teve como objetivo a "criação de um sistema que auxilie o deficiente visual na detecção de faixas de pedestres para travessia de ruas usando aparelhos

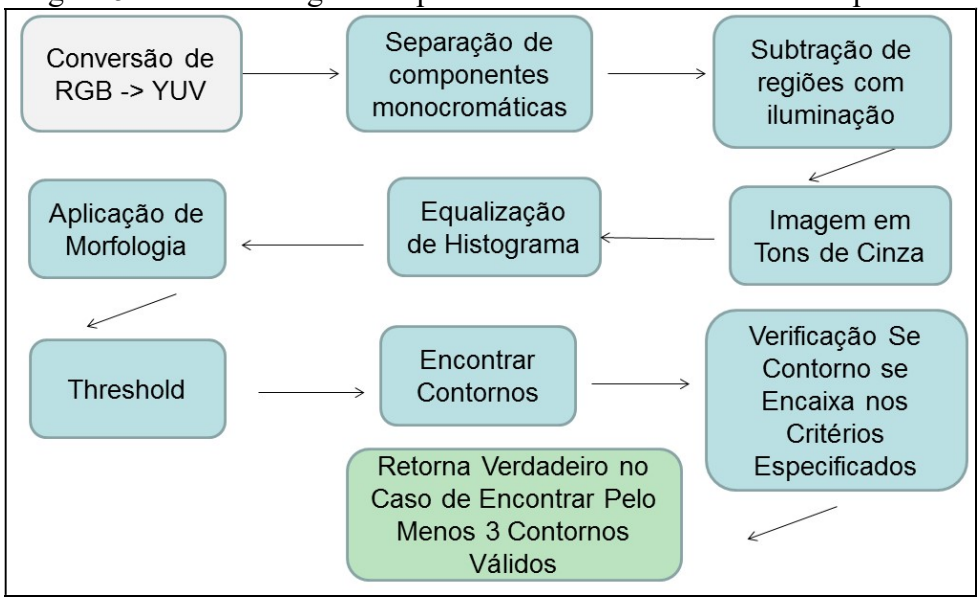
celulares que utilizem a plataforma Android. Adicionalmente, o sistema efetua o reconhecimento de pessoas de forma a aumentar a confiabilidade da informação".

Para realizar o de tratamento de imagens e implementação dos algoritmos, Sousa (2013, p. 46) optou pelo uso da biblioteca OpenCV que já fornecia suporte para a plataforma Android. Para a definição das técnicas de identificação de faixa de pedestre e algoritmos de identificação de pessoas, a autora realizou diversos testes entre algumas soluções já existentes para ambas as situações.

Ao final de suas pesquisas para os algoritmos de reconhecimento de pessoas, Sousa (2013, p. 55) constatou que o algoritmo Histogram of Oriented Gradients (HOG), que tem como objetivo a extração de características em imagens, em conjunto com um algoritmo baseado no modelo Support Vector Machines (SVM), que consiste em supervisionar as tarefas de regressão e classificação, obteve melhor performance, chegando a uma taxa de acerto de 91,8% com tempo de processamento de 1500 milissegundos. No que diz respeito ao reconhecimento de faixas de pedestres, Sousa (2013, p. 52-54) definiu uma rotina com diversas técnicas de trabalho com imagem conforme demonstrado na Figura 5.

Sousa (2013, p. 53) explica que o sistema começa convertendo a imagem para o padrão YUV, um modelo de representação da cor dedicado ao vídeo analógico, para que o algoritmo funcione em ambientes noturnos. Após isto, a imagem passa por uma sequência de seis tratamentos e filtros para facilitar o processo de encontro de contornos que definem as faixas que vai desde a subtração de regiões com iluminação até a fase de *threshold*. Então o algoritmo de busca de contornos é executado juntamente com as validações especificadas para definição de uma faixa de pedestre e por fim retorna ao usuário verdadeiro caso tenha encontrado no mínimo três contornos que se encaixem nas características necessárias.

Figura 5 – Fluxo do algoritmo para reconhecimento de faixas de pedestre



Fonte: Sousa (2013, p. 52).

Para que exista uma forma de comunicação efetiva entre o aplicativo e o usuário, foi utilizada a síntese de voz, onde, para tal, Sousa (2013, p. 56) destaca que o Android não possui um sintetizador de voz para língua portuguesa. Isto levou a autora a utilizar a biblioteca SVOX para que o sistema pudesse funcionar em português.

Com o projeto finalizado, Sousa (2013, p. 59-64) aponta que foram realizados diversos testes em cenários a luz do dia e durante a noite para fins de coletar informações sobre o aplicativo em funcionamento. A autora demonstra os cenários mais propícios ao sucesso em identificar a faixa de pedestre e pessoas, como em ambientes mais iluminados, com menor tráfego de pessoas. Também demonstra os cenários em que o sistema teve maiores problemas, como testes em avenidas escuras durante a noite e sem a devida iluminação. O aplicativo teve resultados bastante satisfatórios quanto ao reconhecimento de imagens e, no que diz respeito ao tempo de processamento. A autora aponta ainda que a aplicação teve importantes ganhos de desempenho durante o desenvolvimento, mantendo o tempo médio menor que um segundo.

2.5.2 Read4blind

O trabalho de Ferreira (2014, p. 3) tinha como objetivo “o desenvolvimento de uma aplicação que permita as pessoas cegas escutar a leitura de textos escritos em papel, sinais, paredes ou outros suportes impressos”. Utilizando uma tecnologia de reconhecimento óptico de caracteres (Optical Character Recognition - OCR), o aplicativo exclusivo para iOS foi desenvolvido para utilizar a câmera do celular para identificar textos em diversos locais.

Ferreira (2014, p. 6-8) explica o funcionamento do algoritmo de OCR de forma simples. Primeiro é realizada a captura do documento ou texto desejado, seguido da localização e segmentação das regiões de texto para a extração dos símbolos. Logo após é realizado um pré-processamento nos símbolos extraídos, afim de eliminar os possíveis ruídos existentes. Então, o algoritmo realiza a extração de características dos símbolos reconhecidos. Em seguida é realizada a classificação destes símbolos comparando suas características com as classes de símbolos obtidas através de uma fase de aprendizagem. Por fim, é realizado o pós-processamento onde os resultados obtidos são processados a fim de adquirir o texto com a maior precisão possível.

Após o reconhecimento e processamento dos textos, é gerada uma saída em forma de voz humana através de uma biblioteca de sintetização de voz (Text-To-Speech – TTS). Para poder desenvolver o produto, o autor precisou realizar diversos testes para diferentes bibliotecas de OCR e sintetização de voz.

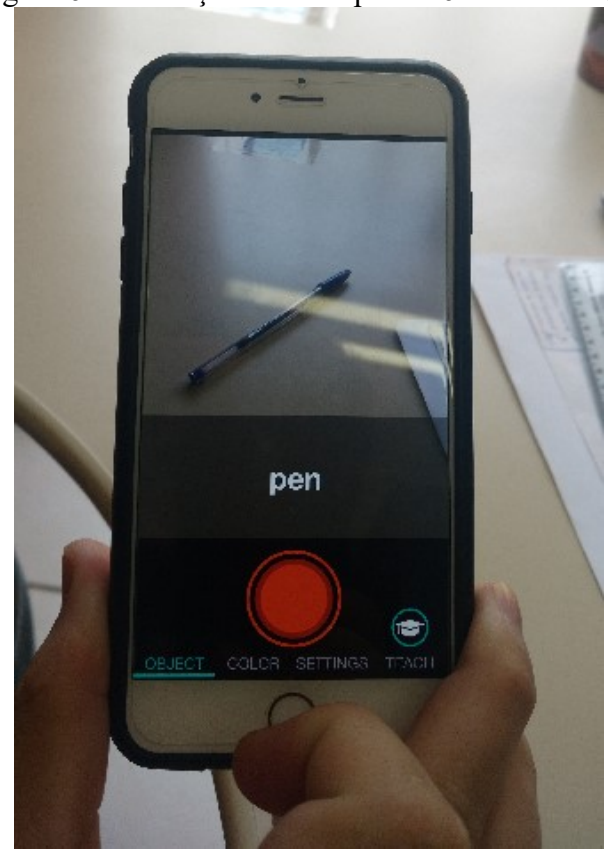
Ao que diz respeito à biblioteca de OCR, Ferreira (2014, p. 36-37) decidiu que a biblioteca a ser utilizada no projeto seria o Tesseract, pois esta obteve um resultado satisfatório nos testes, além de ser gratuita. Para a biblioteca de TTS, após realizar os testes apenas com ferramentas gratuitas, Ferreira (2014, p. 37) verificou que os resultados eram muito semelhantes e acabou por optar pelo GoogleTTS que apresenta maior facilidade de utilização e implementação.

Ao final, após a utilização conjunta das duas tecnologias e a implementação de algoritmos de otimização, o aplicativo foi finalizado em sua proposta e obteve resultados satisfatórios. O autor não disponibilizou o aplicativo para uso, porém deu margem para trabalhos futuros como melhorias nos algoritmos e até mesmo a substituição de bibliotecas.

2.5.3 Aipoly

Aipoly é um aplicativo desenvolvido para iOS e Android que é capaz de identificar objetos em geral através da câmera do celular do usuário. Aipoly (2016a) foi desenvolvido com uma Inteligência Artificial (IA), ou mais especificamente uma Rede Neural de Múltiplas Camadas, com a proposta de aprender e entender todo o universo a sua volta, ou para onde a câmera do dispositivo está sendo apontada, retornando as devidas descrições dos mesmos em forma de voz. A Figura 6 mostra um exemplo do aplicativo em funcionamento. Aipoly (2016b) ressalta que já ajudou mais de 350.000 pessoas com deficiências visuais a explorarem o mundo através do aplicativo, ganhou 12 premios nacionais e internacionais incluindo o CES2017 Best of Innovation Award e está concorrendo ao $5M AI Xprize.

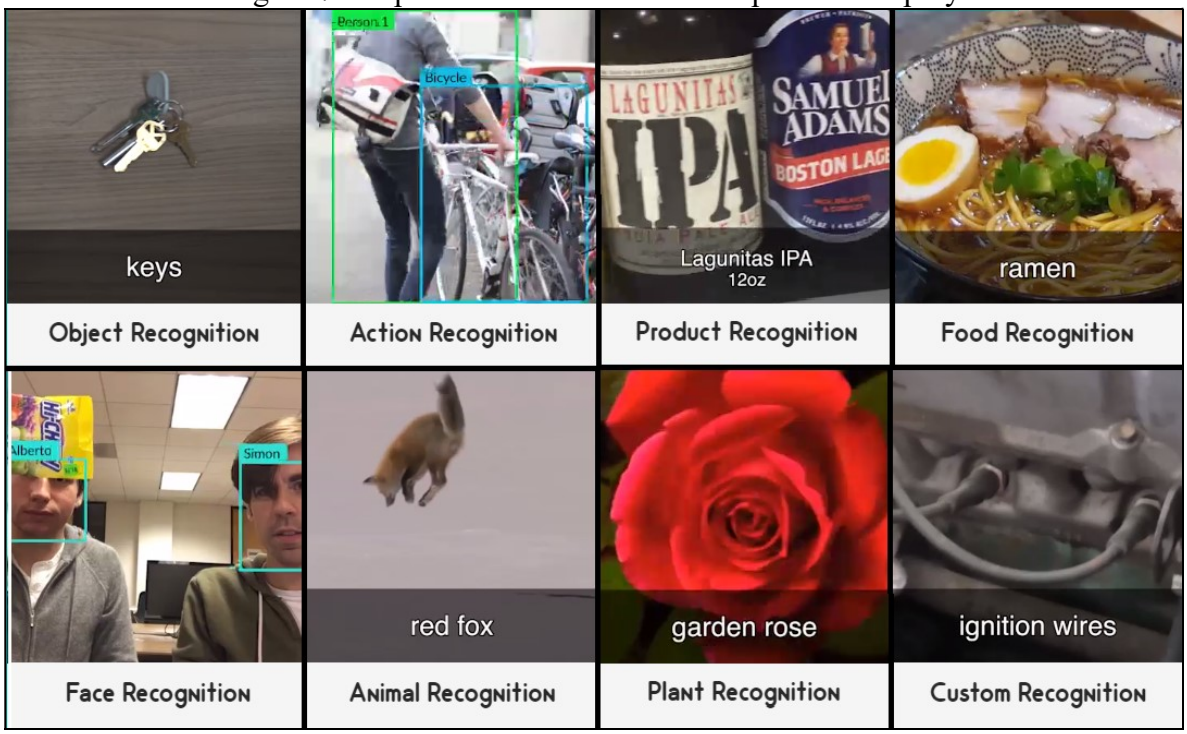
Figura 6 – Execução em um Iphone 6 com iOS 10.3



Fonte: Digitalizado pelo autor.

Conforme Aipoly (2016c) apresenta, suas propriedades de aprendizado fazem do aplicativo até dez vezes mais rápido que soluções na nuvem para reconhecimento de imagem e faz isso sem usar nenhuma conexão com a internet. Também garante que a aplicação tenha um grande poder de diversificação com relação aos tipos de reconhecimentos que é capaz de realizar, como demonstrado na Figura 7. Aipoly (2016c) explica que a IA é executada localmente, fazendo a aplicação funcionar em qualquer ocasião e também explica que nenhum usuário será forçado a enviar as imagens para o serviço de processamento na nuvem da empresa. Aipoly (2016a) destaca que a parte importante de ser uma rede neural é que, conforme a comunidade de pessoas com deficiência ou até mesmo usuários comuns vão utilizando, maior se tornará o poder de reconhecimento do aplicativo.

Figura 7 – Tipos de reconhecimento do aplicativo Aipoly



Fonte: Aipoly (2016c).

3 DESENVOLVIMENTO

Este capítulo faz uma abordagem com relação à construção do aplicativo denominado Munchkin Recognizer. Na seção 3.1 é possível visualizar todos os requisitos funcionais e não funcionais. Na Seção 3.2 são apresentados os diagramas de caso e uso e classes, detalhando o funcionamento e especificando o aplicativo. Ainda nesta seção, são apresentados os principais serviços que integram a solução, mostrando a comunicação entre todos os componentes. A seção 3.3 retrata todas as ferramentas, bibliotecas e serviços utilizados durante o desenvolvimento. Os principais trechos de código e utilização das aplicações também são explicados nesta seção. Para encerrar o capítulo, a seção 3.4 expõe os resultados obtidos após o desenvolvimento do aplicativo e realizada a comparação entre trabalhos correlatos apresentados na seção 2.4.

3.1 REQUISITOS

Os requisitos funcionais e não funcionais do sistema desenvolvido estão descritos na lista abaixo:

a) o aplicativo deve permitir que o usuário tire foto da carta (Requisito Funcional - RF);
b) o aplicativo deve efetuar o reconhecimento do texto existente na foto da carta (RF);
c) o aplicativo deve utilizar o texto reconhecido da foto para encontrar a carta correta (RF);
d) o aplicativo deve sintetizar a descrição da carta de forma clara e em português (RF);
e) o aplicativo deve permitir que o usuário possa repetir a descrição da carta (RF);
f) o aplicativo deve aceitar entradas de comando de voz (RF);
g) o aplicativo deve ser implementado utilizando a ferramenta multiplataformas Ionic (Requisito Não Funcional - RNF);
h) o aplicativo deve ser desenvolvido com tecnologias web (HTML5, CSS e Javascript) no ambiente de desenvolvimento Visual Studio Code (RNF);
i) o aplicativo deve utilizar a Interface de Programação de Aplicações (API) Cloud Vision da Google para realizar o reconhecimento de texto (RNF).

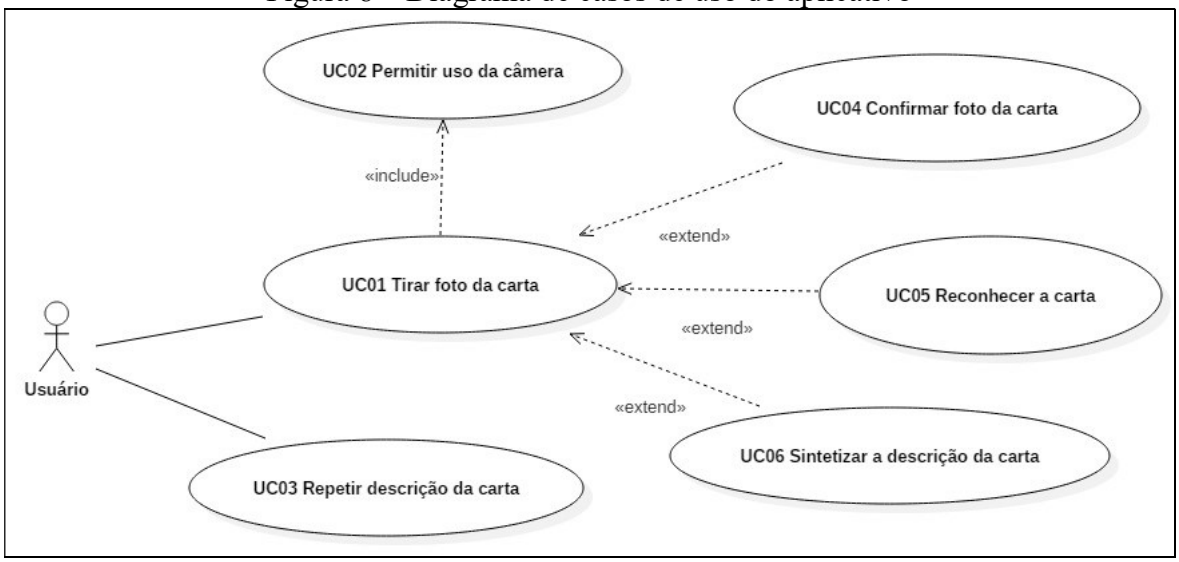
3.2 ESPECIFICAÇÃO

Para o entendimento das funcionalidades e arquiteturas do aplicativo, esta seção apresenta os diagramas de caso de uso, classe e atividade, elaborados com as ferramentas StarUML e Draw.io. No final desta seção é apresentada a arquitetura de funcionamento do aplicativo com seus respectivos serviços.

3.2.1 Diagrama de caso de Uso

A figura 8 mostra os casos de uso que representam as funcionalidades para os usuários do aplicativo.

Figura 8 – Diagrama de casos de uso do aplicativo



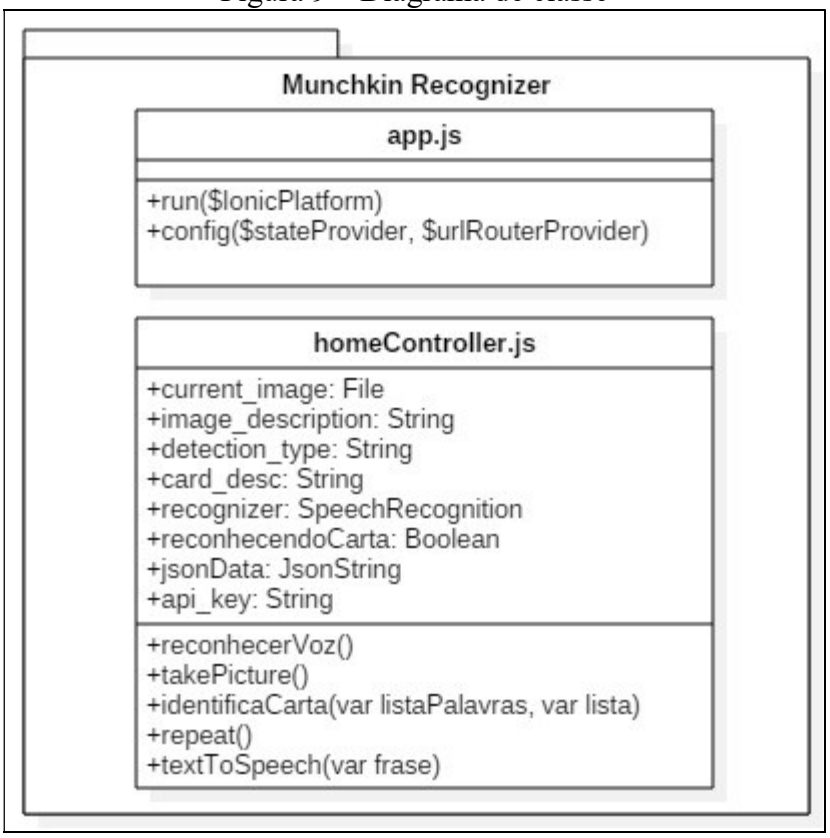
Fonte: Elaborado pelo autor.

O caso de uso `UC01 Tirar foto da carta` permite ao usuário tirar uma foto da carta que deseja reconhecer, porém isso somente será possível após a autorização para o uso da câmera do dispositivo através do caso de uso `UC02 Permitir uso da câmera`. No caso de uso `UC04 Confirmar foto da carta`, o usuário deverá confirmar se a foto que foi tirada será realmente utilizada para o reconhecimento, caso contrário o usuário poderá tirar outra foto ou apenas cancelar a ação, voltando para a tela inicial do aplicativo. Depois de confirmada a foto, o sistema então irá realizar o reconhecimento da carta, como mostrado no caso de uso `UC05 Reconhecer a carta`. O caso de uso `UC06 Sintetizar a descrição da carta` representa a sintetização das informações da carta que foi reconhecida. Após ter sido realizado o reconhecimento da carta o usuário terá a opção de ouvir novamente a descrição da mesma através do caso de uso `UC03 Repetir Descrição da carta`.

3.2.2 Diagrama de classe

Para a implementação do aplicativo Munchkin Recognizer, foi utilizado como base o projeto disponibilizado pelo usuário anchetaWern no serviço de hospedagem de projetos GitHub (2016a), o qual possui uma demonstração do uso do Google Cloud Vision API (2017) e foi reaproveitada a estrutura desse projeto para criar o Munchkin Recognizer. A Figura 9 apresenta o diagrama de classes da aplicação.

Figura 9 – Diagrama de classe



Fonte: Elaborado pelo autor.

A classe `app.js` é usada exclusivamente para iniciar a aplicação independente do dispositivo, realizando as configurações necessárias para garantir que todos os componentes nativos sejam inicializados e executados sem erros. A classe `homeController.js` possui todas as funcionalidades necessárias para o funcionamento da aplicação. O caminho da foto tirada fica armazenado no parâmetro `current_image`, a descrição da última carta reconhecida é salva na variável `card_desc`, o parâmetro `recognizer` é utilizado para realizar o reconhecimento de voz, a variável `jsonData` contém o registro de todas as cartas catalogadas e a variável `api_key` guarda a chave de acesso ao serviço externo da Google.

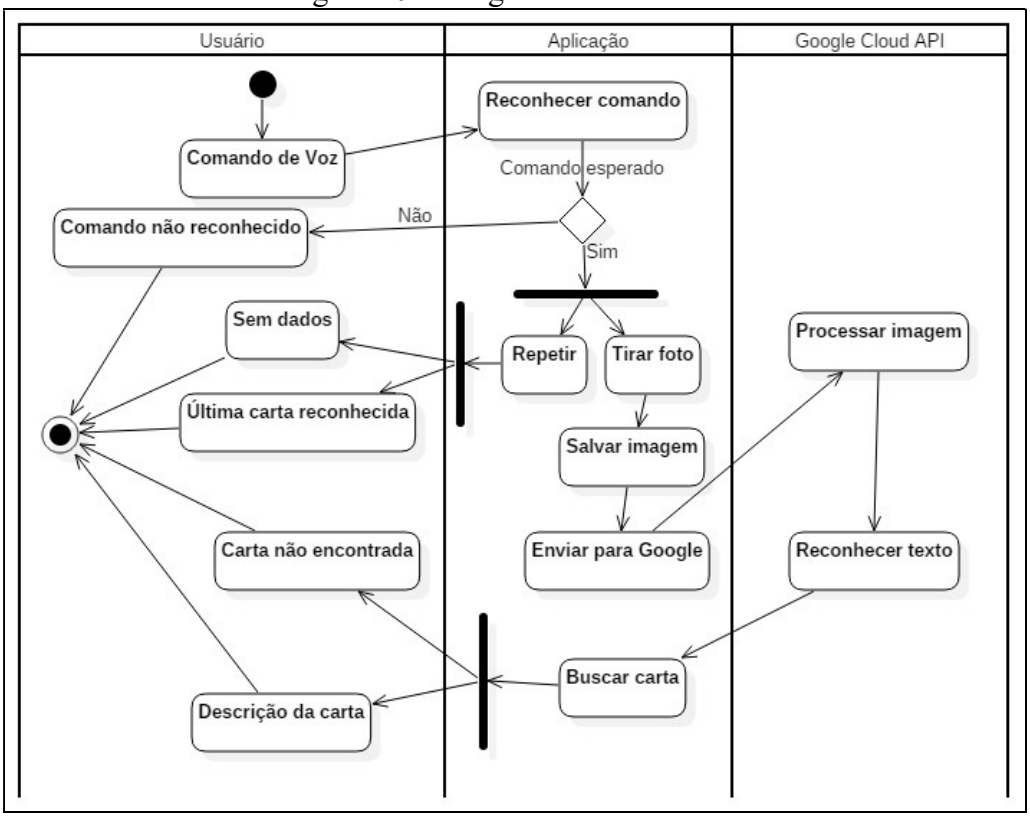
Com relação aos métodos, `reconhecerVoz` realiza o reconhecimento de voz para executar os comandos da aplicação; `takePicture` faz toda a parte de tirar, salvar e enviar a foto para o serviço externo executar o algoritmo de OCR; `identificarCarta` executa o

algoritmo de aproximação com as informações retornadas da Google para identificar a carta alvo; `repeat` repete a descrição da última carta reconhecida pelo usuário e `textToSpeech` tem o objetivo de substituir os retornos visuais, normalmente utilizados para usuários comuns, por retornos de voz.

3.2.3 Diagrama de atividade

A Figura 10 apresenta um diagrama de atividade onde são apresentadas as etapas e ações do usuário na utilização do aplicativo. Quando o aplicativo é aberto o usuário tem a sua disposição uma interface simples com apenas um botão grande o suficiente para que o usuário consiga pressioná-lo sem dificuldades. Ao pressionar este botão o usuário pode executar um comando de voz para o aplicativo que, por sua vez, irá realizar o reconhecimento do comando e, caso o comando esteja entre os comandos aceitos pela aplicação, o mesmo será executado. Do contrário, será retornado ao usuário, em forma de voz, que o comando não pode ser reconhecido.

Figura 10 – Diagrama de atividades



Fonte: Elaborado pelo autor.

Caso o comando tenha sido o comando de tirar foto, o aplicativo irá abrir a câmera para que o usuário possa tirar uma foto da carta que deseja reconhecer. Após a captura da imagem o sistema então irá efetuar o arquivamento da imagem em um diretório temporário e enviará esta mesma imagem para o serviço Google Cloud Vision API (2017). Neste serviço

serão realizados os tratamentos de imagem necessários e, por fim, será feita a execução do algoritmo de OCR e o texto reconhecido será retornado para a aplicação. Por último, será executado o algoritmo de aproximação para tentar identificar a carta em questão. Se obtiver sucesso, a descrição da carta será retornada ao usuário, se não, será retornado um erro indicando que não foi possível identificar a carta e solicitando uma nova foto. Ambos retornos são em forma de voz.

Caso o comando tenha sido o comando para repetir a última carta reconhecida, a aplicação irá verificar se existem dados do último reconhecimento e então efetuará a sintetização dessas informações ao usuário. Do contrário irá apenas retornar uma mensagem, em forma de voz, indicando que não existem dados disponíveis.

3.2.4 Arquitetura, *plugins* e serviços do aplicativo

Para o funcionamento deste trabalho, vários *plugins* e um serviço de processamento na nuvem da Google foram utilizados. A Figura 11 mostra a arquitetura macro criada com a integração destes componentes.

Figura 11 – Arquitetura do aplicativo

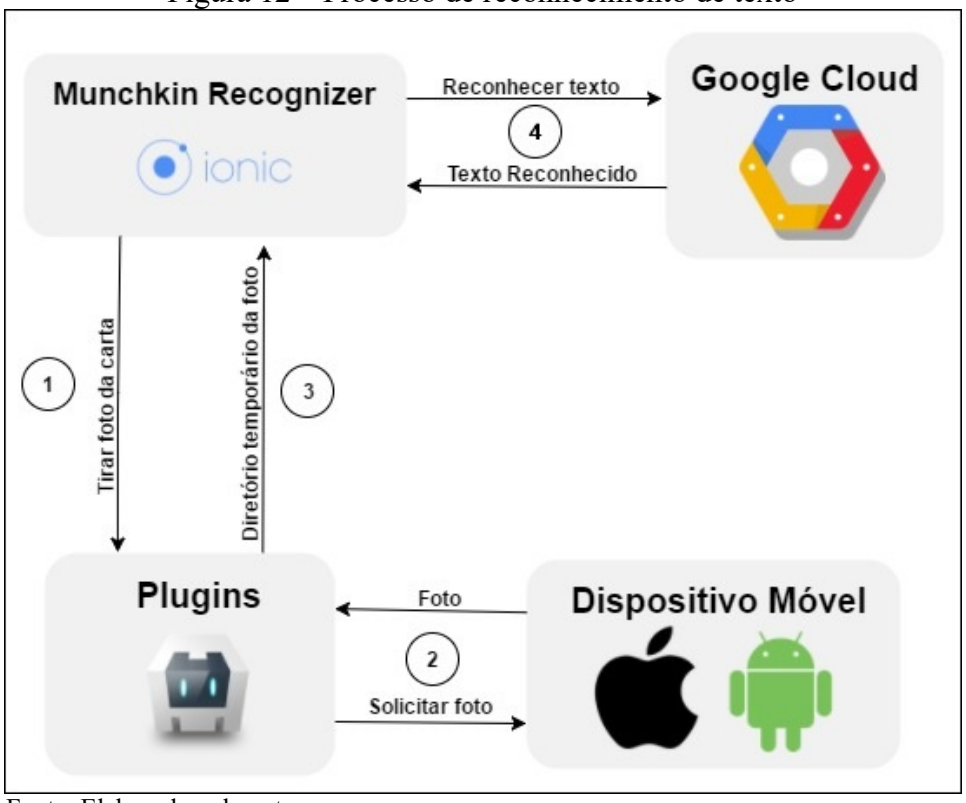
Fonte: Elaborado pelo autor.

A camada `Dispositivo Móvel` representa o dispositivo móvel do usuário com o aplicativo Munchkin Recognizer instalado. Este aplicativo é um `Hybrid Web Container` desenvolvido na plataforma Ionic e que pode ser executado em sistemas operacionais Android e iOS.

O aplicativo possui diversos `Plugins` que permitem a manipulação dos periféricos como câmera e microfone, e também permite fazer o envio de um arquivo para uma aplicação externa via upload. O serviço da `Google` chamado de Google Cloud Vision API (2017) é uma tecnologia que engloba diversas funcionalidades de processamento de imagem, uma delas é o

reconhecimento de texto que é utilizado pelo aplicativo. A Figura 12 demonstra o funcionamento da aplicação com relação aos *plugins* e o serviço externo.

Figura 12 – Processo de reconhecimento de texto



Fonte: Elaborado pelo autor.

Para que o reconhecimento do texto na foto seja realizado é necessário executar o passo 1, realizando a chamada do *plugin* de controle da câmera. No passo 2 é realizado uma solicitação para que a foto seja tirada e retornada para ser salva pelo controle de sistema de arquivos. O passo 3 se resume em devolver o diretório temporário da foto para a aplicação. Por fim, no passo 4, o aplicativo faz o envio da imagem para o serviço da Google que faz todo o processo de OCR e retorna esta informação para que sejam feitos os devidos tratamentos de reconhecimento da carta e sintetização das informações da mesma.

3.3 IMPLEMENTAÇÃO

Nesta seção são apresentadas as técnicas e ferramentas utilizadas no trabalho e as etapas seguintes para o desenvolvimento do aplicativo Munchkin Recognizer. Por fim, são apresentadas as telas e funcionalidades do protótipo desenvolvido.

3.3.1 Técnicas e ferramentas utilizadas

O desenvolvimento deste trabalho está divido em três partes. A primeira delas é a parte do aprendizado do uso da plataforma Ionic. Utilizando o ambiente de desenvolvimento do

Visual Studio Code e as plataformas Android e iOS, foram criados dois projetos pequenos afim entender aos poucos a estrutura de funcionamento desta plataforma e de testar os *plugins* para utilização da câmera e sintetização de voz.

A segunda etapa foi dedicada à pesquisa e teste de técnicas de reconhecimento de imagens e de OCR. A primeira parte da pesquisa foi focada em encontrar ferramentas e bibliotecas que pudessem ser utilizadas sem o uso de internet para processamento, tendo apenas a necessidade de um dispositivo e a aplicação instalada. O primeiro objeto de pesquisa foi a utilização da biblioteca do Tesseract para realizar o processamento de OCR em imagens. Foram encontrados alguns modelos para a utilização desta ferramenta em conjunto da plataforma Ionic, como, por exemplo, o disponibilizado pelo usuário gustavomazzoni na plataforma GitHub (2016b). Após diversas tentativas de desenvolvimento de uma aplicação de teste sem sucesso ao tentar realizar a execução das mesmas, foi concluído que a pouca quantidade de documentação para o uso destas tecnologias em conjunto e a alta necessidade de tempo para desenvolver um aplicativo de testes funcional eram inviáveis para este trabalho.

Durante as pesquisas foi encontrado um projeto de teste utilizando uma biblioteca chamada GNU Ocrad. Esta biblioteca funciona como uma extensão do Ionic e se mostrou de fácil uso e aplicação. Foi executada uma bateria de testes com esta tecnologia e foi concluído que esta biblioteca funciona muito bem com imagens de textos bem uniformes e formatados como, por exemplo, um *print screen* de um documento no computador. Porém, para textos com formatações incomuns, fotos de documentos e as cartas de Munchkin, o GNU Ocrad não obteve resultados satisfatórios.

Após diversas tentativas em construir a base do aplicativo em uma solução off-line, a estratégia de pesquisa foi alterada para abordar soluções on-line. A busca foi realizada tentando manter o foco na praticidade e menor uso de banda possível.

Uma das opções encontradas foi a de utilizar uma IA para efetuar o reconhecimento completo das imagens. Foram realizados testes com uma extensão do IBM Watson chamada de Visual Recognition. Este serviço é pago com um mês de teste grátis. Com uma interface bastante moderna e visualmente fácil de se entender, preparar o material e treinar a máquina para reconhecer as cartas foi uma tarefa bastante simples. Contudo, não se pôde dizer o mesmo da parte de comunicação externa com o IBM Watson. Por conta de falta de documentação e também uma abordagem bastante difícil no retorno de erros dos servidores, integrar esta solução ao Ionic se tornou uma tarefa difícil e sem visão de sucesso.

Outra solução on-line encontrada foi o Google Cloud Vision API (2017), uma ferramenta de processamento de imagem com diversas funções, inclusive OCR. Se trata de um serviço também pago que oferece 300 dólares de uso da plataforma de nuvem da Google para testes e avaliações. Esta ferramenta se mostrou de fácil uso e a grande quantidade de exemplos e informações disponibilizadas para os usuários através das comunidades facilitaram ainda mais na construção dos testes de validação. Foram realizados testes com as cartas de Munchkin e os resultados foram satisfatórios o suficiente para optar por esta ferramenta para a execução de OCR na aplicação final.

Por fim, a terceira etapa se resume a criação do aplicativo para dispositivos móveis multiplataformas Munchkin Recognizer, construído utilizando o *framework* Ionic. Realizando a junção de todos os componentes já testados e validados em uma única solução. Para a construção do código fonte, utilizaram-se as ferramentas padrões de desenvolvimento web HTML5, CSS, Javascript e AngularJS, com o auxílio do ambiente de desenvolvimento Visual Studio Code.

Complementando o desenvolvimento do aplicativo, foram utilizados os seguintes *plugins*: um para utilização da câmera do dispositivo móvel (`cordova-plugin-camera`); um para realizar a sintetização de voz quando necessário (`cordova-plugin-tts`); um para realizar o reconhecimento de voz (`SpeechRecognitionPlugin`); um para acessar a câmera do dispositivo móvel (`cordova-plugin-camera`) e um para realizar o envio da imagem para processamento nos servidores da Google (`cordova-plugin-file-transfer`). Foi utilizado o módulo ngCordova para acessar os plugins no formato de injeção de dependências utilizando módulos do AngularJS.

3.3.2 Trechos de código

Nessa seção, encontram-se trechos de código desenvolvidos para realizar a integração com o serviço de OCR da Google, câmera, transferência de arquivos, reconhecimento e sintetização de voz. Os principais métodos do aplicativo são detalhados para o entendimento do comportamento de sua execução.

3.3.2.1 Acessando a câmera do dispositivo móvel

Para que o aplicativo realize o reconhecimento da carta de Munchkin, o usuário deve tirar uma foto da mesma com a câmera do seu dispositivo móvel. Foi utilizado o *plugin* `cordova-plugin-camera` com a extensão `ngCordova` para acessar o recurso da câmera do dispositivo. O Quadro 2 apresenta o código fonte utilizado, onde na linha 2 é definido o

objeto `options` com as opções disponíveis no *plugin* para controle da câmera e imagem. Os parâmetros `targetWidth` e `targetHeight` foram definidos como 500 para limitar o tamanho da imagem para envio para os servidores da Google. Na linha 14, o método `getPicture` da classe `$cordovaCamera`, abre o aplicativo de câmera nativo do sistema operacional do dispositivo móvel. Após o usuário tirar a foto, na linha 16, a imagem é passada para o formato `base64`, na linha 19 é criado o `vision_api_json` e por seguinte, na linha 35 é declarado o `file_contents` que será utilizado na comunicação com o Google Cloud Vision API (2017).

Quadro 2 – Utilizando a câmera do dispositivo móvel

```
 1  $scope.takePicture = function(){
 2     var options = {
 3        destinationType: Camera.DestinationType.DATA_URL,
 4        sourceType: Camera.PictureSourceType.CAMERA,
 5        targetWidth: 500,
 6        targetHeight: 500,
 7        correctOrientation: true,
 8        cameraDirection: 0,
 9        encodingType: Camera.EncodingType.JPEG
10     };
11
12     $scope.textToSpeech("Abrindo a câmera. Tire uma foto da carta para
                           realizar o reconhecimento.");
13
14     $cordovaCamera.getPicture(options).then(function(imagedata){
15        me.reconhecendo = true;
16        me.current_image = "" + imagedata;
17        me.image_description = '';
18
19        var vision_api_json = {
20           "requests":[
21              {
22                 "image":{
23                    "content": imagedata
24                 },
25                 "features":[
26                    {
27                       "type": me.detection_type,
28                       "maxResults": 1
29                    }
30                 ]
31              }
32           ]
33        };
34
35        var file_contents = JSON.stringify(vision_api_json);
36
37        // Rotina de arquivamento e envio de imagem
38        // ...
39
40     }, function(err){
42       $scope.textToSpeech('Ocorreu um erro ao tirar a foto da carta');
43     });
44  }
```

Fonte: Elaborado pelo autor.

3.3.2.2 Utilizando a sintetização de voz do dispositivo

Para facilitar a comunicação do aplicativo com o usuário final, fez-se necessário a adição da sintetização de voz. Sendo este recurso bastante utilizado em todo o aplicativo, foi desenvolvido o método `textToSpeech` demonstrado no Quadro 3. Na linha 2 é realizada a chamada do método `speak` do *plugin* `cordova-plugin-tts`, onde o parâmetro `text` é o texto que que será sintetizado, `locale` se refere à linguagem do sintetizador e `rate` é a velocidade em que o texto será falado.

Quadro 3 – Utilizando a sintetização de voz

```
1  $scope.textToSpeech = function(frase) {
2     TTS.speak({
3        text: frase,
4        locale: 'pt-BR',
5        rate: 1.2
6     }, function () {
7        // Do Something after success
8     }, function (reason) {
9        // Handle the error case
10    });
11 }
```

Fonte: Elaborado pelo autor.

3.3.2.3 Enviando a imagem para processamento

Após a foto da carta ter sido tirada, o aplicativo precisa salvar a carta em um diretório temporário e então enviar a imagem para o serviço externo da Google. O Quadro 4 demonstra o algoritmo para salvar a imagem no dispositivo e realizar o upload da mesma. Na linha 9 é utilizado o método `writeFile` onde `cordova.file.cacheDirectory` é o diretório onde a imagem será salva, o segundo parâmetro se trata do nome do arquivo, `file_contents` são as informações do arquivo mostradas no Quadro 2 que servirão de orientação para o Google Cloud Vision API (2017) realizar os devidos processamentos. Após realizar o arquivamento da foto, na linha 21 é declarada a variável `server` que recebe o caminho da aplicação da Google e também a `api_key` que foi gerada pela interface de uso deste serviço e serve como uma chave de autorização para o uso dele. Na linha 25 é realizado o `upload` da imagem para o serviço externo através do *plugin* `cordova-plugin-file-transfer`. Após o processamento da imagem, é então feita uma conversão do retorno enviado dos servidores da Google na linha 30 para que possa ser analisado e usado para identificar a carta. Na linha 32 é realizado um tratamento de remoção de caracteres de formatação e pontuação para que a função de reconhecimento da carta funcione com maior precisão.

Quadro 4 – Envio de imagem para Google

```
$scope.takePicture = function(){
   // Quadro 1
   // ...

   $cordovaCamera.getPicture(options).then(function(imagedata){
      // Quadro 1
      // ...

      $cordovaFile.writeFile(
         cordova.file.cacheDirectory,
         'file.json',
         file_contents,
         true
      ).then(function(result){
         var headers = {
            'Content-Type': 'application/json'
         };

         options.headers = headers;

         var server =
           'https://vision.googleapis.com/v1/images:annotate?key='
           + api_key;

         var filePath = cordova.file.cacheDirectory + 'file.json';

         $cordovaFileTransfer.upload(server, filePath, options,
         true)
        .then(function(result){

          var res = JSON.parse(result.response);
          var key = me.detection_types[me.detection_type] +
                    'Annotations';

          me.image_description =res.responses[0][key][0].description.
                                  replace(/\r\n/g, " ").
                                  replace(/\n/g, " ").
                                  replace(/\n/g, " ").
                                  replace(/\t/g, " ").
                                  replace(".", "").
                                  replace("!", "").split(" ");

          $scope.identificaCarta(me.image_description,
                                 me.jsonData.CartasArray);


        }, function(err){
           $scope.textToSpeech('Ocorreu um erro ao fazer o upload da
                                imagem');
        });
      }, function(err){
         $scope.textToSpeech('Ocorreu um erro ao salvar a imagem.');
      });

   }, function(err){
      $scope.textToSpeech('Ocorreu um erro ao tirar a foto da carta');
   });
}
```

Fonte: elaborado pelo autor.

3.3.2.4 Realizando o reconhecimento de voz

Para melhorar a usabilidade do aplicativo pelo público alvo, se faz necessário o uso de uma alternativa de comunicação do usuário com o dispositivo móvel e vice-versa. Utilizar o reconhecimento de voz para realizar as funções da aplicação é uma solução bastante viável e possível com o *plugin* `SpeechRecognitionPlugin` disponibilizado pelo usuário macdonst no serviço de hospedagem de projetos GitHub (2016c). O Quadro 5 demonstra o algoritmo do método `reconhecerVoz` que é acionado pelo botão mestre da aplicação. Quando executado, é criada uma instância do *plugin* como pode ser visto na linha 3 e então é feita a configuração de linguagem para o mesmo na linha 4. Na linha 7 é feita a sobreposição do método `onresult` para que sejam realizados os tratamentos do resultado do reconhecimento de acordo com a aplicação e não do *plugin*. Enquanto a aplicação não ouvir nenhuma entrada de voz, o *plugin* manterá a instância do reconhecedor por um período de tempo até que o usuário realize um comando. Assim que o comando for reconhecido, o método sobrescrito tentará compará-lo com os dois possíveis comandos demonstrados nas linhas 12 e 15, onde, caso obtenha sucesso, será disparado o método correspondente da linha 13 ou 16 respectivamente.

Quadro 5 – Algoritmo de reconhecimento de voz

```
1  $scope.reconhecerVoz = function() {
2     var texto = "";
3     me.recognizer = new window.SpeechRecognition();
4     me.recognizer.lang = "pt-BR";
5     me.recognizer.continuous = true;
6
7     me.recognizer.onresult = function(event) {
8        if (event.results.length > 0) {
9           texto = event.results[0][0].transcript;
10
11          if (!me.reconhecendo) {
12             if (texto == "tirar foto") {
13                $scope.takePicture();
14
15             } else if (texto == "repetir") {
16                $scope.repeat()
17
18             } else {
19                $scope.textToSpeech('Comando inválido.');
20             }
21          }
22
23       }
24    };
25
26    me.recognizer.start();
27 }
```

Fonte: Elaborado pelo autor

3.3.2.5 Efetuando o reconhecimento da carta

Para que o reconhecimento da carta seja possível foi criado um arquivo JSON com uma lista de todas as cartas do jogo cadastradas com duas chaves, sendo elas o nome e a descrição. Este arquivo foi hospedado por um serviço on-line chamado MyJson (2017) afim de não ocupar espaço de memória dos dispositivos. A aquisição do arquivo é realizada através do *request* apresentado no Quadro 6. Na linha 1 é executado o *request* através do método `get` e, caso este tenha sucesso, os dados retornados são atribuídos à variável `jsonData` que será utilizada durante a busca da carta.

Quadro 6 – Aquisição da lista de cartas

```
1  $http.get("https://api.myjson.com/bins/1ewky5")
2     .success(function(response) {
3        me.jsonData = response;
4     })
5
6     .error(function(response) {
7        $scope.textToSpeech("Erro ao buscar informações das cartas.
8                             Verifique sua conexão.");
9  });
```

Fonte: Elaborado pelo autor

Ao receber os resultados do reconhecimento enviados dos servidores da Google, a aplicação executa o método `identificaCarta` que é demonstrado no Quadro 7. Este método possui um algoritmo, aqui denominado de algoritmo de aproximação, que tem como objetivo percorrer a lista de palavras (`listaPalavras`) reconhecidas da carta e filtrar as cartas com as mesmas palavras até que reste apenas uma que é, teoricamente, a carta alvo.

Na linha 13 é iniciado um loop que percorrerá todas as palavras da variável `listaPalavras`. Cada palavra será analisada no segundo loop iniciado na linha 14 que tem objetivo de percorrer todas as cartas salvas na variável `lista`. Na linha 15 é criada uma lista temporária, chamada de `tempList`, com as palavras contidas no nome da carta. Na linha 17 esta lista será percorrida em um terceiro loop que irá comparar as palavras do título da carta com a palavra contida na variável `listaPalavras`. Caso exista esta palavra no título da carta, o objeto que representa a carta é passado para uma segunda variável temporária (`novaLista`) que ao final do segundo loop terá filtrado todas as cartas que possuem em comum a mesma palavra do primeiro loop. Ao final da primeira interação é verificada a quantidade de cartas filtradas. Caso seja apenas uma, o método é finalizado e, na linha 38, a aplicação retorna ao usuário a descrição da carta encontrada, caso contrário, a variável `lista` recebe os valores da variável `novaLista` que tem seu conteúdo apagado em seguida para que o ciclo se repita até que exista apenas uma carta, como demonstrado no intervalo de linhas do número 27 a 33.

Caso todo o ciclo de filtragem termine sem encontrar a carta, então, na linha 41, é lançada uma mensagem ao usuário solicitando uma nova foto.

Quadro 7 – Algoritmo de aproximação

```
1  $scope.identificaCarta = function(listaPalavras, lista) {
2     var tempList = [];
3     var novaLista = [];
4
5     $ionicLoading.show({
6        content: 'Loading',
7        animation: 'fade-in',
8        showBackdrop: true,
9        maxWidth: 200,
10       showDelay: 0
11    });
12
13    for (i=0; i < listaPalavras.length; i++) {
14       for (j=0; j < lista.length; j++) {
15          tempList = lista[j].nome.split(" ");
16
17          for (k=0; k < tempList.length; k++) {
18             if ($filter('uppercase')(tempList[k]) ==
                   $filter('uppercase')(listaPalavras[i])) {

19                novaLista.push(lista[j]);
20                break;
21             }
22          }
23       }
24
25       if (novaLista.length > 0) {
26          if (novaLista.length == 1) {
27             break;
28
29          } else {
30             lista = novaLista;
31             novaLista = [];
32          }
33       }
34    }
35
36    if (novaLista.length > 0 && novaLista.length == 1) {
37       me.card_desc = novaLista[0].desc;
38       $scope.textToSpeech(novaLista[0].desc);
39
40    } else {
41       $scope.textToSpeech("Foto com baixa resolução ou fora de foco.
42                            Por favor. Tire outra foto da carta.");
43    }
44
45    $ionicLoading.hide();
46    me.reconhecendo = false;
47 }
```

Fonte: Elaborado pelo autor.

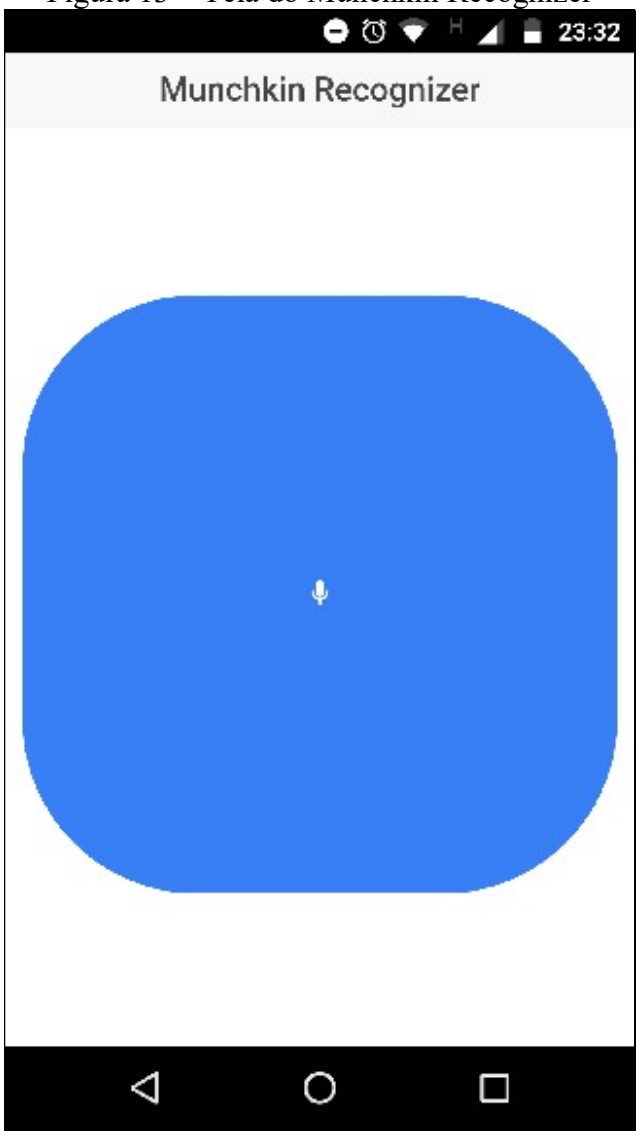
3.3.3 Operacionalidade da Implementação

Nesta seção é apresentada a operacionalidade da aplicação através da interface gráfica disponível para o usuário. As imagens abaixo foram obtidas utilizando câmera do dispositivo móvel Moto Play X com Android 6.0.1.

3.3.3.1 Reconhecimento da carta

Por se tratar de uma aplicação voltada para pessoas com deficiência visual, o Munchking Recognizer possui uma interface bastante simples, como demonstrado na Figura 13, tendo apenas um botão central que irá ativar o reconhecimento de voz para que o usuário possa dar o comando desejado.

Figura 13 – Tela do Munchkin Recognizer

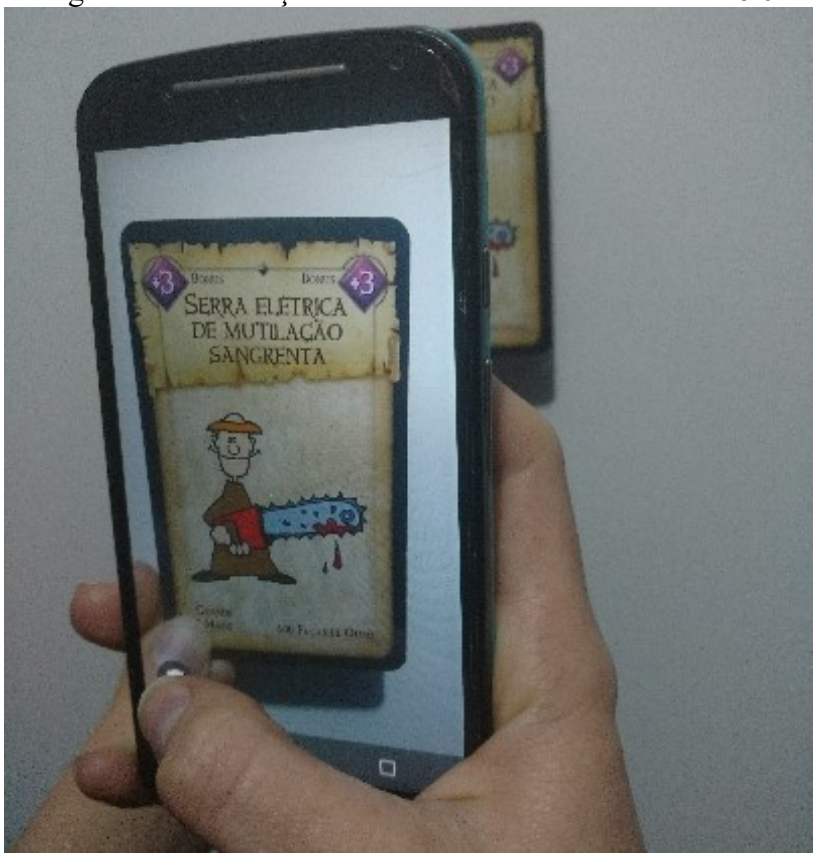


Fonte: Elaborado pelo Autor.

Ao clicar no botão, a aplicação ativa o reconhecimento de voz e o mantém ativo por um período de tempo de aproximadamente cinco segundos, esperando por uma entrada de

algum comando qualquer. Os comandos reconhecíveis pela aplicação são `tirar foto` e `repetir`. Todo e qualquer comando diferente será considerado inválido para o aplicativo e, caso o usuário utilize o comando de repetir sem ter realizado nenhum reconhecimento de carta, o sistema retornará uma mensagem de voz informando que não existem dados para executar o comando. Ao dar entrada com o primeiro comando de tirar foto, o sistema então realiza a abertura da câmera e a prepara para tirar a foto da carta. Com a câmera aberta, o usuário pode então tirar uma foto da carta que deseja realizar o reconhecimento (Figura 14). A foto pode ser feita de forma padrão e manual ou, caso o dispositivo tenha configurado os comandos de voz nativos, por comando de voz.

Figura 14 – Execução em um Moto G 2 com Android 6.0



Fonte: Digitalizado pelo autor.

Depois que a foto é tirada o usuário precisa fazer uma confirmação da mesma ou, caso queira uma foto melhor, descartar a foto e então tentar mais uma vez (Figura 15).

Figura 15 – Execução em um Moto G 2 com Android 6.0

Fonte: Digitalizado pelo autor.

Quando confirmada a foto, o sistema então volta para a tela principal da aplicação enquanto executa os algoritmos de reconhecimento da carta. O resultado será retornado ao usuário na forma de voz.

3.4 ANÁLISE DOS RESULTADOS

Esta seção é dedicada a mostrar os experimentos realizados com o Munchkin Recognizer. Na seção 3.4.1 é apresentada a metodologia utilizada para a realização do experimento. A seção 3.4.2 apresenta os experimentos e resultados da aplicação. Os experimentos da aplicação foram realizados individualmente e em grupo com duas pessoas conhecedoras do jogo. A seção 3.4.3 apresenta uma análise geral sobre o trabalho desenvolvido como um todo. Por fim, a seção 3.4.4 apresenta uma comparação entre os trabalhos correlatos e a aplicação.

3.4.1 Metodologia

Os testes do aplicativo foram realizados durante o mês de junho de 2017, com testes individuais e em grupo acompanhados do autor deste trabalho. No caso dos testes individuais, foi utilizado um dispositivo móvel Moto Play X com Android 6.0.1. Já para os testes em

grupo, foram utilizados um dispositivo móvel Galaxy Note 3 com Android 5.0 e um dispositivo móvel Moto G com Android 5.1. No experimento em grupo, foi realizada uma breve explicação do trabalho, como utilizar a aplicação e de seu objetivo como uma aplicação de auxílio a deficientes visuais para que possam participar de uma partida de Munchkin. As imagens dos testes em grupo se encontram no Apêndice A.

3.4.2 Experimentos e resultados da aplicação

Esta seção está dividida em duas partes, a seção 3.4.2.1 apresenta os testes individuais e mais estatísticos do aplicativo e a seção 3.4.2.2 apresenta os testes em grupo e seus resultados.

3.4.2.1 Testes individuais

Os testes individuais foram focados em analisar e adquirir dados para avaliar a performance da aplicação nos quesitos: tempo de execução do OCR (processamento da imagem enviada para o serviço externo), tempo de processamento do algoritmo de aproximação (com ou sem sucesso) e precisão dos reconhecimentos realizados. Para que possam adquirir maior abrangência, todos os testes que envolveram a captura de imagens foram realizados em três etapas, sendo elas em curta (entre 3 e 5 cm), média (entre 5 e 12 cm) e longa (maior que 12 cm) distância (Figura 16) em ambientes de alta e baixa iluminação (Figura 17). Todas as capturas de imagem foram feitas com o intuito de garantir a melhor imagem possível em todos os casos de testes individuais.

Figura 16 – Etapas de testes com captura de imagem

Fonte: Elaborado pelo autor.

Figura 17 – Ambientes de alta e baixa iluminação

Fonte: elaborado pelo autor.

Para facilitar a mensuração das informações necessárias para os testes, foi criado um "modo de desenvolvedor" na aplicação que tem como objetivo retornar dados relevantes e de forma visual para o usuário (Figura 18).

Figura 18 – Modo desenvolvedor

Fonte: Elaborado pelo autor.

3.4.2.1.1 Tempo de resposta do serviço MyJson

Estes testes tiveram o objetivo de avaliar o tempo de conexão com o serviço MyJson (2017) e a execução do `get` da lista de cartas catalogadas do jogo. Estes testes foram realizados em um ambiente com conexão via wi-fi e outro ambiente com conexão via dados móveis (3G). Para cada ambiente foram realizados 20 testes que se resumem em abrir a aplicação e verificar o tempo que foi necessário para que o aplicativo consiga realizar o *request* e as informações sejam retornadas. A Figura 19 mostra a execução de um dos testes realizados.

Figura 19 – `Get` da lista de cartas

Fonte: Elaborado pelo autor.

Como resultado, foi verificado que os dois ambientes de testes obtiveram uma média de resultados bastante próxima sendo 2027 milissegundos para a conexão via wi-fi e 2711 milissegundos para a conexão via dados móveis. Com isso, é possível concluir que o serviço de hospedagem do arquivo que lista todas as cartas é bastante viável, pois ajuda a aplicação em manter um tamanho de armazenamento baixo e possui um ótimo tempo de resposta e retorno para a mesma.

3.4.2.1.2 Tempo de execução do OCR

Com o objetivo de validar o tempo de execução dos algoritmos de OCR, foram realizados testes em dois ambientes, sendo eles um com conexão via wi-fi e outro com conexão via dados móveis (3G). Os testes se basearam na execução do reconhecimento (independentemente do resultado) de 10 cartas diferentes. Para cada carta, foi utilizado o padrão de captura de curta, média e longa distância com alta e baixa iluminação, totalizando

60 execuções para cada ambiente. A Figura 20 mostra a execução de um dos 120 testes realizados ao todo. Para realizar a validação de cada caso, optou-se pelo uso do cálculo da mediana, pois alguns dos testes apresentaram variações de tempo muito grandes, quando o padrão de tempo era visivelmente menor.

Figura 20 – Teste de tempo de execução do OCR

Fonte: Elaborado pelo Autor.

A Tabela 1 apresenta as medianas resultantes para cada caso no ambiente iluminado e a Tabela 2 apresenta as medianas adquiridas para os testes no ambiente com baixa iluminação.

Tabela 1 – Mediana de tempo de execução em ambiente iluminado

Conexão	Curta distância(ms)	Média distância(ms)	Longa distância(ms)
Wi-fi	2024	2237	2199
Dados móveis	7010	6324	7271

Fonte: Elaborado pelo autor.

Tabela 2 – Mediana de tempo de execução com baixa iluminação

Conexão	Curta distância(ms)	Média distância(ms)	Longa distância(ms)
Wi-fi	1670	1546	1645
Dados móveis	3084	4463	4173

Fonte: elaborado pelo autor.

Com os resultados demonstrados nas Tabelas 1 e 2, observando apenas os valores referentes às medianas em cada tipo de conexão, é possível concluir de forma direta que o uso de uma rede de internet com conexão via wi-fi é mais recomendada para o uso desta aplicação, tendo em vista que o uso de dados móveis deixa a experiência de uso até aproximadamente três vezes mais lenta.

Com relação aos casos de diferentes distâncias, observando-se as informações de forma geral, os testes demonstraram que existe pouca influência no tempo de processamento, dada a variação de melhor tempo entre cada caso de ambas as tabelas.

Ao observar as medianas de cada caso e comparando com seus equivalentes entre as Tabelas 1 e 2, é possível verificar que, dentre todos os casos, a performance de tempo de execução do processamento de imagem do Google Cloud Vision API (2017) foi melhor nos casos em que a imagem estava submetida à um ambiente de baixa iluminação.

3.4.2.1.3 Tempo de execução do algoritmo de aproximação

Este teste tem como objetivo validar o tempo de execução do algoritmo criado para realizar a filtragem e a localização da descrição da carta que está sendo reconhecida, utilizando as informações retornadas dos servidores da Google, independentemente de sucesso ou falha. Este teste foi realizado em paralelo com os testes de tempo de execução do OCR e baseou-se em apenas calcular a média de tempo de execução de todas as execuções de reconhecimento. A Figura 21 apresenta um dos 120 testes realizados.

Ao final dos testes, chegou-se no valor médio de aproximadamente 75 milissegundos para a execução do algoritmo de aproximação. Este resultado demonstra que é um algoritmo bastante performático e eficiente no quesito de tempo de execução, dada a quantidade de dados existente para o processamento das 149 cartas catalogadas, dando bastante margem para o aumento de tempo que tende a existir com a adição da coleção completa das cartas do jogo de Munchkin, podendo alcançar mais de 600 cartas. É possível concluir também que, ao observarmos o processamento da carta como um todo, a parte de execução do algoritmo de aproximação (com ou sem sucesso) tem influência mínima no tempo de execução total.

Figura 21 – Tempo de execução do algoritmo de aproximação

Fonte: Elaborado pelo autor.

3.4.2.1.4 Precisão da aplicação

Também realizado em paralelo com os testes de tempo de execução do OCR, este teste teve o objetivo de avaliar a precisão do aplicativo executado como um todo através da quantidade de cartas que foram reconhecidas ou não. Foram avaliados os mesmos casos de curta, média e longa distância em ambientes com alta e baixa iluminação. A Tabela 3 apresenta a relação de acertos por número de processamentos para cada um dos casos.

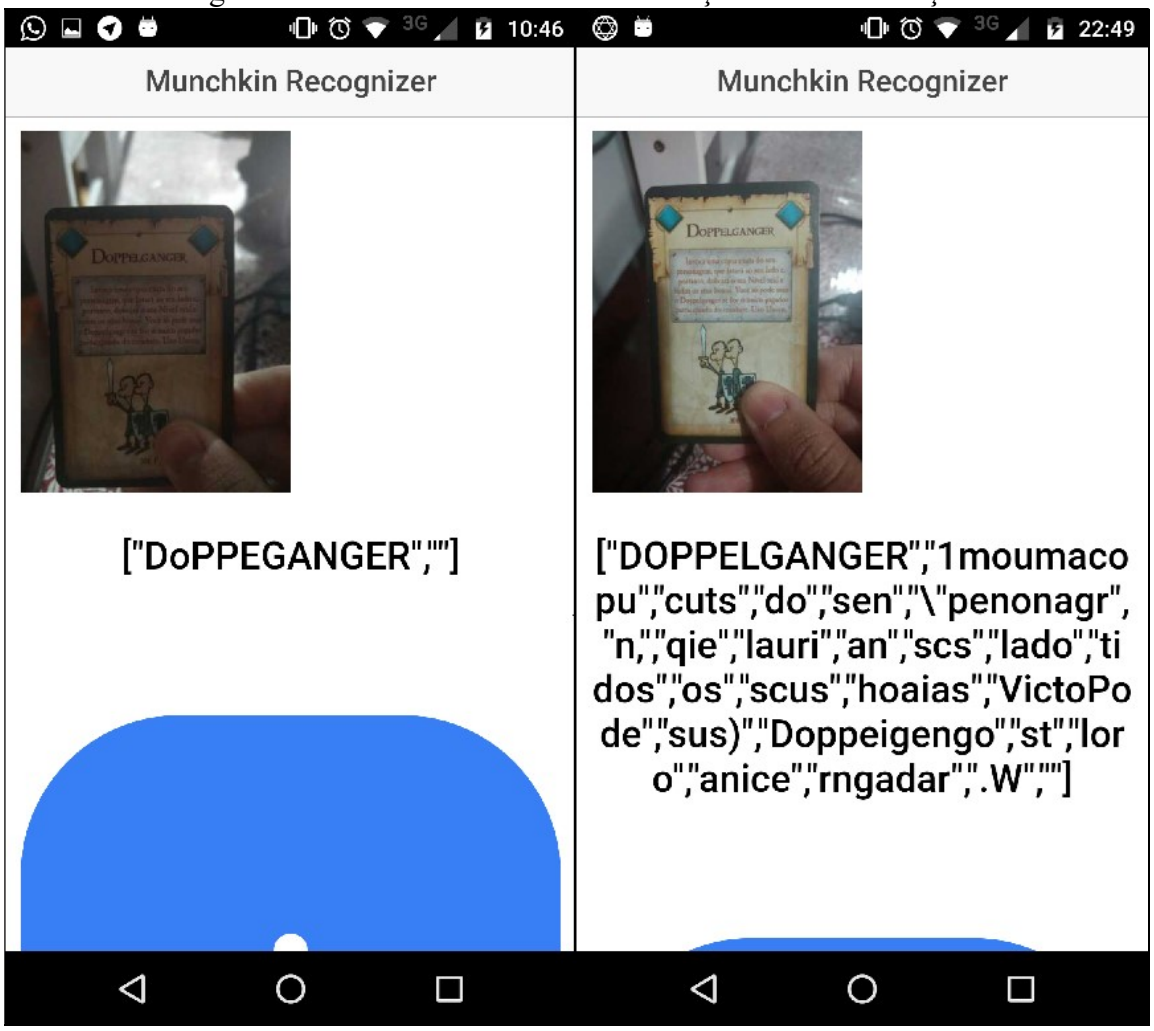
Tabela 3 – Precisão da aplicação

Iluminação	Curta distância	Média distância	Longa distância
Alta	20/20	18/20	16/20
Baixa	15/20	11/20	5/20

Fonte: Elaborado pelo autor.

Com base na Tabela 3 e observando-as com as informações adquiridas nas Tabelas 1 e 2, é possível concluir que a precisão do aplicativo nos casos de baixa iluminação, por mais que o processamento tenda a ser mais rápido, acaba sendo muito inferior aos casos executados em alta iluminação. Os testes demonstraram que, em baixa iluminação, o processamento de OCR tem maiores problemas em reconhecer todos os caracteres do título da carta, necessário para o sucesso da execução do algoritmo de aproximação, diferentemente do que acontece nos casos de alta iluminação (Figura 22).

Figura 22 – Resultados de baixa iluminação e alta iluminação



Fonte: Elaborado pelo autor.

Pôde-se observar que, mesmo com a precisão menor, a baixa iluminação não torna o uso, nestas condições, impossível, porém afeta diretamente a fluidez e usabilidade do aplicativo. Pois isto pode tornar uma partida muito lenta e até mesmo pode acabar irritando os participantes da partida pela falta de precisão e a quantidade de repetições necessárias. Um ambiente com bastante iluminação é o mais indicado para garantir uma precisão bastante confortável para os usuários usufruírem do aplicativo de forma mais responsiva.

3.4.2.2 Testes em grupo

Os testes em grupo foram focados em validar a usabilidade da aplicação em uma partida de Munchkin. Para a realização dos testes, foi escolhido o cenário mais propício para o melhor desempenho do mesmo, tendo boa iluminação e utilizando uma conexão via wi-fi. Pelo fato dos participantes serem conhecedores do jogo, não foi necessário explicar nenhuma das regras do mesmo, assim como as formas de se jogá-lo. Apenas foram passados os objetivos dos testes e a única exigência seria de que deveriam jogar de olhos fechados ou vendados. Os testes duraram cerca de 30 minutos e iniciaram na distribuição de cartas e foram até o final da primeira rodada.

No que diz respeito à interface da aplicação, a mesma se mostrou bastante simples e fácil de uso. Os comandos de voz de `tirar foto` e `repetir` foram entendidos com bastante facilidade em ambos os dispositivos, porém foi verificado que em certos níveis de ruído a aplicação passou a distorcer os comandos e necessitar de novas tentativas. O problema dos ruídos teve significante diminuição ao serem adicionados fones de ouvido com microfone embutido, dando um pouco mais de precisão ao reconhecimento de voz. A real utilização da aplicação se deu início na parte de identificação das cartas que foram dadas para cada usuário no começo do jogo. Esta tarefa se mostrou bastante longa com ambos os jogadores tendo que tentar realizar o reconhecimento de cada carta mais de uma vez e tendo que memorizá-las.

Analisando cada execução de reconhecimento realizada, foi possível identificar uma limitação com relação ao posicionamento do dispositivo (câmera) e da carta, onde grande parte dos erros de reconhecimento aconteceram por conta da falta de informação com relação ao foco da carta. Este problema foi parcialmente contornado no dispositivo Galaxy Note 3 onde foram configuradas as opções de acessibilidade. O dispositivo passou a dar um sinal de aviso sempre que o foco fosse realizado, porém, em alguns casos, isto levou bastante tempo para acontecer e nem sempre o foco realmente focou a carta por conta do posicionamento da mesma com relação a câmera.

Tirar a foto (independentemente do foco) se mostrou uma tarefa simples, a qual, com as devidas configurações do dispositivo, pode até mesmo ser realizada por comando de voz. Porém, logo após a captura da foto, é necessário realizar uma confirmação da imagem e, dado o público alvo da aplicação, não existiria necessidade para tal. Esta confirmação se mostrou um empecilho na usabilidade do aplicativo, pois se trata de um botão na parte direita inferior da tela, em uma área bastante fácil de se errar e que não aceita comando de voz para executar

esta função. Ao estudar o *plugin* da câmera, foi verificado que não existe opção para remover esta confirmação através do mesmo.

Os jogadores tiveram certa dificuldade na parte de mapeamento das cartas em suas mãos, onde muitas falhas de reconhecimento aconteceram e decorar cada carta que estavam em posse se mostrou algo bastante complicado, fazendo com que necessitassem de confirmação mais de uma vez. De forma geral, a aplicação foi muito bem recebida como uma ferramenta que auxiliará os deficientes visuais na inclusão ao jogo e demonstra bastante potencial com relação ao poder de processamento. Porém só o aplicativo ainda não é o suficiente para uma integração completa, ou seja, um grupo composto apenas por pessoas com deficiência visual, por exemplo, não conseguiria jogar o jogo sem ao menos uma pessoa sem deficiência auxiliando-as.

3.4.3 Análise geral sobre o desenvolvimento da aplicação

Nesta seção são apresentadas análises sobre como foi o processo de desenvolvimento da aplicação e suas implicações neste trabalho.

Pelo fato da ideia inicial da aplicação ser totalmente off-line, logo no início do desenvolvimento foi realizado um esforço em mantê-lo totalmente dentro do *framework* Ionic, ou seja, sem a necessidade de aplicações externas e muito complexas. A parte de pesquisas realizadas e já relatadas tomou bastante tempo do desenvolvimento pois tentou-se ao máximo encontrar formas de manter a ideia do aplicativo não necessitar de internet para realizar o reconhecimento das cartas. Em certa altura das pesquisas, foi cogitada a criação de uma aplicação nativa que se comunicaria com a aplicação do Ionic para realizar as tarefas de processamento enquanto o Ionic apenas gerenciaria os dados de envio e o recebimento de resultados. Após algumas pesquisas foi verificado que isso seria possível e provavelmente seria a solução mais eficiente, visto que possibilitaria o uso de ferramentas como o Tesseract, ou até mesmo o OpenCV, de forma nativa. Porém, a curva de aprendizado e o tempo necessário para produzir esta solução eram muito grandes e a ideia teve que ser deixada para trás.

Infelizmente o tempo estava se tornando apertado e ao menos uma solução deveria ser buscada e desenvolvida. Portanto a ideia da aplicação off-line foi abandonada e, após alguns testes com algumas plataformas como o IBM Watson e o Google Cloud Vision API (2017), novas ideias surgiram e o serviço da Google foi escolhido e adicionado ao projeto. Com isso, o objetivo de fazer com que a aplicação seja de acesso gratuito não pôde mais ser atingido, pois este serviço tem um custo para cada vez que é acessado através da chave de acesso,

deixando apenas a opção de disponibilizar os arquivos fontes de forma gratuita. A evolução do aplicativo aconteceu de forma rápida e prática após alguns testes realizados com a plataforma externa de processamento de imagens e reconhecimento de textos. Durante a implementação do que seria a base da aplicação final, foram encontrados alguns problemas de execução na plataforma iOS e como o autor deste trabalho não tinha nenhuma forma de executar o aplicativo nesta plataforma, fez-se necessário a locomoção até as dependências da FURB. Foram realizados alguns testes utilizando um macOS e um iPAD e logo descobriu-se que se tratava de um problema na distribuição de diretórios do iOS que possui algumas diferenças com relação ao Android. Para resolver este problema foi realizada uma rápida pesquisa sobre os diretórios em comum das duas plataformas e o diretório que estava dando problema foi alterado, fazendo com que a aplicação voltasse a funcionar normalmente.

De início a aplicação foi desenvolvida de forma bastante visual e com algumas interações com a tela através do toque. Contudo, após conversar com o professor Dalton Solano dos Reis (orientador deste trabalho), foi adicionada a funcionalidade que permite a comunicação com a aplicação na forma de comandos de voz. Tentou-se fazer com que este recurso ficasse 100% do tempo ativo e sempre pronto para receber um comando. Todavia a única forma encontrada de fazer isto era com a aplicação a todo momento emitindo o som de ativação do microfone sem parar, o que se mostrou inviável. O layout do aplicativo foi alterado para que funcionasse o máximo possível com apenas comandos de voz. Os botões de ação foram substituídos por apenas um único botão no centro da tela, grande o suficiente para que não possa ser errado pelo usuário e que habilita o microfone do dispositivo para que seja realizado o reconhecimento de voz.

A ideia de mapeamento das cartas existia desde o início do projeto, porém a forma de mapeamento foi mudada drasticamente, onde, ao invés de mapear a imagem da carta, foram mapeados os títulos de cada carta do jogo original (sem nenhuma das extensões) assim como suas devidas descrições. A princípio este mapeamento permaneceria dentro da aplicação no formato de um arquivo Json, entretanto o orientador deste trabalho deu a ideia de já deixar a aplicação preparada para no futuro receber estas informações de um servidor externo. Para tal, foi utilizado o serviço online MyJson (2017) para hospedar o arquivo com as informações das cartas e na aplicação foi adicionado um *request* para o serviço de hospedagem de Json.

3.4.4 Comparação com os trabalhos correlatos

Esta seção faz uma comparação da ferramenta desenvolvida com os trabalhos correlatos apresentados na seção 2.5. O Quadro 8 apresenta a comparação entre as principais características.

Quadro 8 – Quadro comparativo entre os trabalhos correlatos

	Sousa (2013)	Ferreira (2014)	Aipoly (2016c)	Munchkin Recognizer
utiliza internet	Não	Não	Não/Sim	Sim
reconhecimento de texto	Não	Sim	Não	Sim
reconhecimento de imagem	Sim	Não	Sim	Não
sintetização de voz	Sim	Sim	Sim	Sim
reconhecimento de voz	Não	Não	Não	Sim
tecnologia assistiva	Sim	Sim	Sim	Sim
disponível para uso	Não	Não	Sim	Não
plataforma suportada	Android	iOS	Hibrido	Hibrido
framework mobile	Nativo	Nativo	Nativo	Ionic

Fonte: Elaborado pelo autor.

Através das informações demonstradas no Quadro 8, percebe-se que todos os trabalhos são tecnologias assistivas voltadas para a inclusão social, assim como todos fazem a utilização da sintetização de voz como forma de comunicação com seus usuários, porém apenas o Munchkin Recognizer faz o reconhecimento de voz. É possível observar que o aplicativo Munchkin Recognizer necessita de uma conexão com a internet para que possa realizar seus processamentos e alcançar seus objetivos. O Aipoly (2016a) utiliza internet também, porém não para o processamento de imagem, mas sim para o envio de dados para o aprendizado e evolução da IA, enquanto as aplicações de Sousa (2013) e Ferreira (2014) conseguem realizar todas as suas tarefas sem a necessidade de estarem conectados à internet. Percebe-se que o trabalho de Ferreira (2014) e o Munchkin Recognizer possuem a mesma base de trabalho que é o reconhecimento de texto, enquanto o trabalho de Sousa (2013) e a aplicação Aipoly trabalham com reconhecimento de imagens. Verifica-se que dentre todos, a única aplicação disponível para uso é o Aipoly, enquanto todos os outros não foram disponibilizados. Sousa (2013) e Ferreira (2014) foram desenvolvidos para plataformas individuais e nativas, sendo elas Android e iOS respectivamente. Aipoly e Munchkin Recognizer são aplicações híbridas, porém o aplicativo Aipoly foi desenvolvido de forma nativa para Android e iOS, enquanto o Munchkin Recognizer foi desenvolvido para estas mesmas plataformas através do *framework* Ionic.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um aplicativo para incluir e auxiliar as pessoas com deficiência visual em um jogo de mesa chamado Munchkin utilizando o *framework* Ionic para que o mesmo possa ser utilizado em múltiplas plataformas. Com este aplicativo é possível realizar o reconhecimento de qualquer carta do jogo que estiver mapeada através do processamento da imagem e reconhecimento do título da mesma. Tendo como identificar e conhecer a carta através do auxílio da aplicação, o usuário já consegue ter uma base que o permite se aprofundar ainda mais nos conhecimentos do jogo, permitindo que o mesmo possa participar de uma partida de forma igualitária.

Inicialmente tentou-se desenvolver uma aplicação totalmente off-line para que os jogadores pudessem usufruir da ferramenta em qualquer local, bastando apenas o dispositivo com o aplicativo instalado. Porém, devido aos fatores da curva de aprendizado, das tecnologias já disponíveis e suas devidas documentações, seria necessário um tempo maior que o utilizado para o desenvolvimento deste trabalho. Assim, foi optado pela criação de uma solução que pudesse fazer o uso de serviços on-line e atingir o objetivo de realizar o reconhecimento das cartas de Munchkin.

O objetivo de criar uma aplicação de inclusão que auxilie uma pessoa com deficiência visual em uma partida de Munchkin foi cumprido. Também foi cumprido o objetivo de desenvolver o aplicativo utilizando o *framework* Ionic. Os *plugins* utilizados no aplicativo foram compatíveis e possibilitaram a distribuição para as plataformas Android e iOS. A aplicação não foi suficientemente bem desenvolvida para que possa ser disponibilizada nas lojas virtuais de suas respectivas plataformas assim como não houve tempo de ir atrás de uma forma simples de adaptação das cartas para que pudessem ter seus títulos impressos em *braile* na parte de trás das cartas.

A aplicação não pôde ser testada por pessoas com deficiência visual. Porém os poucos testes realizados com pessoas vendadas tentando jogar com o auxílio do aplicativo demonstraram que ainda é necessário melhorar usabilidade do mesmo para que se torne mais fluído, mais responsivo e mais preciso em ambientes adversos. Com relação ao tempo de execução e resposta do aplicativo, os testes resultaram em informações bastante satisfatórias e mostraram que é possível utilizar as tecnologias presentes hoje para pesquisar e investir em aplicações desse tipo.

Por fim, este trabalho deixa uma contribuição social, pois é diretamente direcionada para a inclusão social de pessoas com deficiência visual e auxiliá-las em participar de todos os

momentos de descontração e diversão que um jogo de mesa pode lhes oferecer. Quanto à contribuição cientifica, este trabalho deixa as soluções encontradas para integrar o *framework* Ionic com os *plugins* e o serviço Google Cloud Vision API (2017), além da fundamentação teórica sobre os temas relacionados.

4.1 EXTENSÕES

Para trabalhos futuros são sugeridos:

a) criar uma aplicação nativa da câmera, afim de eliminar o problema da confirmação da foto;
b) eliminar a necessidade de um botão para realizar os comandos de voz;
c) realizar a implementação de um aplicativo nativo para utilizar as ferramentas de processamento de imagem de forma off-line;
d) criar uma página web para que a comunidade de jogadores alimente o mapeamento das cartas de forma orgânica;
e) adaptar a ideia para diferentes tipos de jogos de mesa;
f) implementar o reconhecimento em tempo real, eliminando a necessidade de tirar uma foto da carta.

REFERÊNCIAS

ADÃO, Rafael. **FURB mobile gincanas**: Sistema móvel na plataforma Phonegap para gincanas virtuais. 2016. 79 f. TCC (Graduação) - Curso de Bacharelado em Ciências da Computação, Universidade Regional de Blumenau, Blumenau, 2016. Disponível em: <http://dsc.inf.furb.br/tcc/index.php?cd=6&tcc=1747>. Acesso em: 30 maio 2017.

AIPOLY. **We help the blind and visually impaired quickly identify objects using affordable, cutting-edge technology**. [S.I.], 2016a. Disponível em: <http://aipoly.com/aipoly-vision.html/>. Acesso em: 02 abr. 2016.

_____. **Vision through atificial inteligence**. [S.I.], 2016b. Disponível em: <http://aipoly.com/about.html/>. Acesso em: 02 abr. 2016.

_____. **Real-time deep learning SDK**. [S.I.], 2016c. Disponível em: <http://aipoly.com/index.html>. Acesso em: 02 abr. 2016.

AMIRALIAN, M.L.T.M. apud FERRONI, Marília C. C.; GASPARETTO, Maria E. R. F. A influência das relações sociais no desenvolvimento de escolares com baixa visão. In: VII ENCONTRO DA ASSOCIAÇÃO BRASILEIRA DE PESQUISADORES EM EDUCAÇÃO ESPECIAL, 7., 2011, Londrina. **Família, sociedade e deficiência**. Londrina: ISSN, 2011. p. 1092-1103. Disponível em: <http://www.uel.br/eventos/congressomultidisciplinar/pages/arquivos/anais/2011/familia/103-2011.pdf>. Acesso em: 13 mar. 2017.

APACHE CORDOVA. **Cordova Plugins**. [S.I.], 2013. Disponível em: <http://cordova.apache.org/plugins/>. Acesso em: 25 abr. 2017.

ARAÚJO, Sidnei Alves de. **Casamento de padrões em imagens digitais livre de segmentação e invariante sob transformações de similaridade**. 2009. 142 f. Tese (Doutorado em Engenharia) - Escola Politécnica, Universidade de São Paulo, São Paulo - Brasil.

BRASIL - SDHPR. Governo Brasileiro. Secretaria Especial dos Direitos Humanos (SEDH). **Tecnologia Assistiva**. Brasília: SEDH, 2009. 140 p.

CENTENO, J. A. S. apud SILVA, Ricardo K; RIBEIRO, Selma R.A.R. Importância da alteração do Histograma de Imagem de Alta Resolução (PAN) para fusão de imagens digitais pelo método de componentes principais. **Ambiência**, v. 5, n. 1, 2009. Disponível em:<http://revistas.unicentro.br/index.php/ambiencia/article/view/226>. Acesso em: 07 nov. 2016.

DPI. **Teoria:** Processamento de Imagens**.** [S.I.], 2009. Disponível em:<http://www.dpi.inpe.br/spring/teoria/realce/realce.htm>. Acesso em 05 nov. 2016.

EIKVIL, Line. **OCR Optical Character Recognition**. Oslo: Insular, 1993. 35 p.

FERREIRA, Roberto. **Camera reading for blind people**. 2014. 177 f. Relatório de Projeto (Mestrado em Engenharia Informática – Computação móvel) - Escola Superior de Tecnologia e Gestão, Instituto Politécnico de Leiria, distrito de Leiria - Portugal.

FIJISAWA, Hiromichi. **A view on the past and future of character and document recognition.** [S.I.], 2007. Disponível em:<https://www.researchgate.net/publication/4288403>. Acesso em 05 nov. 2016.

FILHO, Ogê Marques; NETO, Hugo Vieira. **Processamento digital de imagens**. ed. Atual. Rio de Janeiro: Brasport, 1999. 311 p.

GALAPAGOS JOGOS. **Apunhale seus amigos em Munchkin**. [S.I.], 2012. Disponível em: <https://www.galapagosjogos.com.br/jogos/munchkin>. Acesso em: 06 nov. 2016.

GITHUB. **anchetaWern:** Ionic vision. [S.I.], 2016a. Disponível em <https://github.com/anchetaWern/ionic-vision>. Acesso em: 28 maio 2017.

_____. **Cordova Plugin for OCR process using Tesseract**. [S.I], 2016b. Disponível em <https://github.com/gustavomazzoni/cordova-plugin-tesseract>. Acesso em: 04 abr. 2017.

_____. **W3C Web Speech API - Speech Recognition plugin for PhoneGap**. [S.I.], 2016c. Disponível em <https://github.com/macdonst/SpeechRecognitionPlugin>. Acesso em: 28 maio 2017.

GOOGLE CLOUD VISION API. **Google Cloud Vision API Documentation**. [S.I.], 2017. Disponível em <https://cloud.google.com/vision/docs/>. Acesso em: 26 maio 2017.

IONICFRAMEWORK, Ionic. **Welcome to Ionic**. [S.I.], 2016a. Disponível em: <http://ionicframework.com/docs/v1/guide/preface.html>. Acesso em: 02 abr. 2017.

_____. **Create mobile apps with the web technologies you love**. [S.I.], 2016b. Disponível em: < http://ionicframework.com >. Acesso em: 02 abr. 2017.

MYJSON. **A simple JSON store for your web or mobile app**. [S.I.], 2017. Disponível em <http://myjson.com/>. Acesso em: 01 jun. 2017.

SOUSA, Kelly A. O. **Uso de visão computacional em dispositivos móveis para auxílio à travessia de pedestres com deficiência visual**. 2013. 85 f. Dissertação (Mestrado em Engenharia Elétrica) – Programa de Pós Graduação em Engenharia Elétrica, Universidade Presbiteriana Mackenzie – São Paulo.

UFRGS. **Página dinâmica para aprendizado do sensoriamento remoto**. [S.I.], 2008. Disponível em: <http://www.ufrgs.br/engcart/PDASR/>. Acesso em: 06 nov. 2016.

APÊNDICE A – Teste da aplicação no dia 17 de junho de 2017

Neste apêndice são apresentadas fotos do teste realizado no dia 17 de junho de 2017 onde duas pessoas conhecedoras do jogo Munchkin fizeram uma simulação de uma partida de olhos vendados. As Figuras 23 e 24 mostram um dos usuários realizando o reconhecimento de suas cartas no início da partida. As Figuras 25 e 26 mostram o segundo usuário em seu turno realizando o reconhecimento de uma das cartas em jogo.

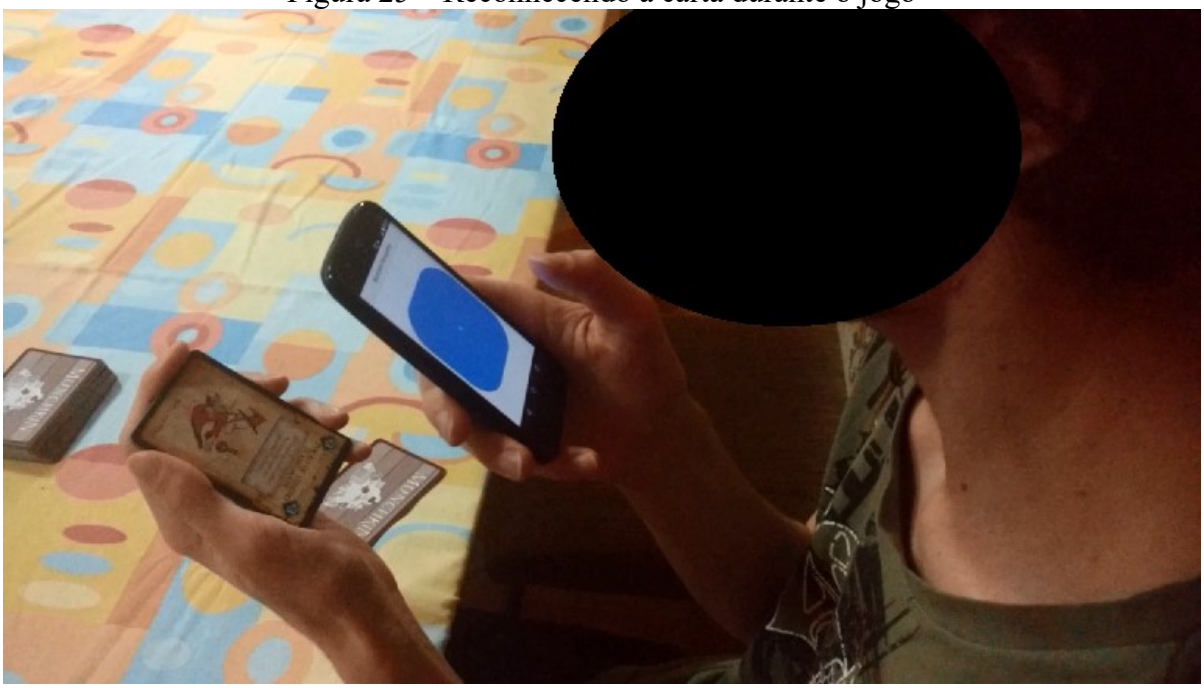
Figura 23 – Reconhecendo as cartas no início do jogo

Fonte: Elaborado pelo autor.

Figura 24 – Reconhecendo as cartas no início do jogo

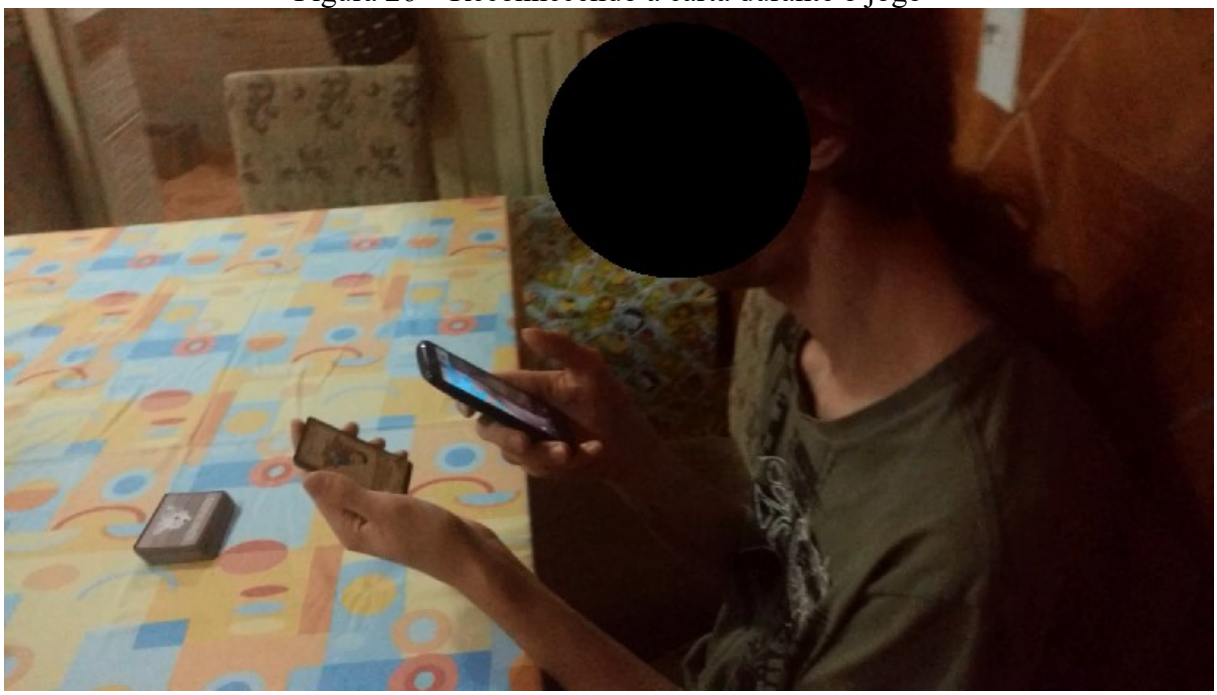
Fonte: Elaborado pelo autor.

Figura 25 – Reconhecendo a carta durante o jogo



Fonte: Elaborado pelo autor.

Figura 26 – Reconhecendo a carta durante o jogo



Fonte: Elaborado pelo autor.