

UNIVERSIDADE REGIONAL DE BLUMENAU
CENTRO DE CIÊNCIAS EXATAS E NATURAIS
CURSO DE CIÊNCIA DA COMPUTAÇÃO – BACHARELADO

JOGO DE LETRAS/NÚMEROS VOLTADO PARA TECNOLOGIA
ASSISTIVA NO ANDROID

WAGNER JEAN REETZ

BLUMENAU
2013

2013/2-21

WAGNER JEAN REETZ

JOGO DE LETRAS/NÚMEROS VOLTADO PARA TECNOLOGIA ASSISTIVA NO ANDROID

Trabalho de Conclusão de Curso submetido à
Universidade Regional de Blumenau para a
obtenção dos créditos na disciplina Trabalho
de Conclusão de Curso II do curso de Ciência
da Computação — Bacharelado.

Prof. Dalton Solano dos Reis, Mestre - Orientador

BLUMENAU
2013

2013/2-21

JOGO DE LETRAS/NÚMEROS VOLTADO PARA TECNOLOGIA ASSISTIVA NO ANDROID

Por

WAGNER JEAN REETZ

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

Presidente:	<hr/> Prof. Dalton Solano dos Reis, M. Sc. – Orientador – FURB
-------------	--

Membro:	<hr/> Prof. Alexander Roberto Valdameri, M. Sc. – FURB
---------	--

Membro:	<hr/> Prof. Maurício Capobianco Lopes, M. Sc. – FURB
---------	--

Blumenau, 10 de dezembro de 2013

Dedico este trabalho a minha família, que sempre me apoiou nas minhas decisões e a todos os meus amigos, especialmente aqueles que me ajudaram diretamente na realização deste.

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.

Talento é dom, é graça. E sucesso nada tem a ver com sorte, mas com determinação e trabalho.

Augusto Branco

RESUMO

Este trabalho apresenta a especificação e implementação de um protótipo de jogo 2D para plataforma Android voltado para tecnologia assistiva, onde é explorado o aspecto pedagógico/lúdico em computação aplicada. Neste jogo o usuário terá a possibilidade de aprender a escrever todos as letras do alfabeto e também os números. Estas possibilidades de aprendizagem estão separadas em planos, que são: letras maiúsculas, letras minúsculas e números. Os planos além de servirem para agrupar as pranchas, também serviram para permitir integrar o material desenvolvido neste trabalho com o projeto Tagarela. O jogo também possibilita ao usuário a criação de planos customizados, onde usuário deverá informar um texto qualquer e o jogo irá montar o plano com base nos símbolos dos planos padrões da aplicação. A avaliação deste trabalho foi realizada com base em testes qualitativos por uma professora de apoio pedagógico a crianças com necessidades especiais. Neste jogo também são aplicados conceitos e técnicas de modelagem orientada a componentes, onde o objeto gráfico desenvolvido é independente e capaz de ser acoplado em outros locais do cenário da aplicação.

Palavras-chave: Android. Jogo. Assistiva.

ABSTRACT

This work presents the specification and implementation of a prototype of a 2D game for the Android platform focused on assistive technology, which is exploring the aspect pedagogical/playful in applied computing. In this game, the user will be able to learn how to write all the letters of the alphabet and numbers. These learning opportunities are separate plans, which are: uppercase letters, lowercase letters and numbers. Plans as well as serving to group the boards, also served to allow the integration of the material developed in this work with the Jabber project. The game also enables the user to create customized plans where user has to enter some text and the game will set the plan based on the symbols of the plans application standards. The evaluation of this work was based on qualitative testing by a professor of educational support to children with special needs. In this game are also applied concepts and techniques of component-oriented modeling, where the object graph is developed independently and able to be engaged elsewhere in the application scenario.

Key-words: Android. Game. Assistive.

LISTA DE ILUSTRAÇÕES

Figura 1 - Símbolos que compõem o plano escolhido	18
Figura 2 - Cenário de um dos níveis do jogo.....	19
Figura 3 - Aplicativo Desenhe e Aprenda a Escrever	20
Figura 4 - Diagrama de casos de uso	22
Quadro 1 - Caso de uso selecionar plano	23
Quadro 2 - Jogar plano	24
Quadro 3 - Criar plano customizado	25
Quadro 4 - Alterar plano customizado	25
Quadro 5 - Excluir plano customizado.....	26
Figura 5 - Diagrama de pacotes do jogo de letras/números	26
Figura 6 - Diagrama de classes do pacote controler.....	27
Figura 7 - Diagrama de classes do pacote view.....	28
Figura 8 - Diagrama de classes do pacote model	30
Figura 9 - Diagrama de classes do pacote util.....	31
Figura 10 - Classes desenvolvidas por Marco (2013) que foram utilizadas no jogo	32
Figura 11 - Diagrama de sequência do jogo	34
Quadro 6 - Método onCreate da classe PrincipalJogo.....	36
Quadro 7 - Método getInstance da classe Gerenciador	36
Quadro 8 - Método prepararJogo da classe Gerenciador.....	37
Quadro 9 - Método CarregarPlanoBD da classe Gerenciador.....	37
Quadro 10 – Continuação do método CarregarPlanoBD da classe Gerenciador	38
Quadro 11 - Método getNextServerID da classe Gerenciador	38
Quadro 12 - Método gravarPlano da classe PlanoBanco.....	39
Quadro 13 - Método onCreate da classe Jogo	39
Quadro 14 - Método gerarPreVisualizacao da classe Jogo	40
Quadro 15 - Método gerarHistorico da classe Jogo.....	41
Quadro 16 - Método gerarViewHistorico da classe Jogo	42
Quadro 17 - Método aplicarPrancha da classe SimboloView.....	43
Quadro 18 - Método playAnimationOut da classe SimboloView	43
Quadro 19 – Continuação do método playAnimationOut da classe SimboloView....	44

Quadro 20 - Método playSound da classe SimboloView.....	44
Quadro 21 – Continuação do método playSound da classe SimboloView.....	45
Quadro 22 - Método recarregarImagens da classe SimboloView.....	45
Quadro 23 - Método getSimboloBmp da classe SimboloBanco.....	46
Quadro 24 - Método onTouch da classe SimboloView.....	47
Quadro 25 - Método onDraw da classe SimboloView	48
Figura 12 - Tela inicial do jogo	49
Figura 13 - Incluir plano customizado.....	50
Figura 14 - Alterar plano customizado.....	51
Figura 15 - Tela jogar plano.....	52
Figura 16 - Desenhando símbolo.....	52
Figura 17 - Plano concluído.....	53
Figura 18 - Gráfico consumo de memória do método CarregarPlanosBD	55
Quadro 26 - Consumo médio de memória por símbolo	55
Figura 19 - Consumo de memória do plano números	56
Figura 20 - Desempenho do método recarregarImagens.....	56
Quadro 27 - Comparação com os trabalhos correlatos.....	57
Figura 21 - Questionário de avaliação	63
Figura 22 - Jogo sendo testado por uma criança com necessidades especiais	64

SUMÁRIO

1 INTRODUÇÃO.....	12
1.1 OBJETIVOS DO TRABALHO	13
1.2 ESTRUTURA DO TRABALHO	13
2 FUNDAMENTAÇÃO TEÓRICA	14
2.1 TECNOLOGIA ASSISTIVA E JOGOS EDUCACIONAIS	14
2.2 MODELAGEM ORIENTADA A COMPONENTES	15
2.3 ANDROID.....	15
2.4 TRABALHOS CORRELATOS	17
2.4.1 Tagarela: Aplicativo para Comunicação Alternativa no iOS.....	17
2.4.2 Dibugrama.....	18
2.4.3 Desenhe e Aprenda a Escrever.....	19
3 DESENVOLVIMENTO	21
3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO.....	21
3.2 ESPECIFICAÇÃO	21
3.2.1 Diagrama de casos de uso	22
3.2.1.1 Selecionar plano	22
3.2.1.2 Jogar plano.....	23
3.2.1.3 Criar plano customizado	24
3.2.1.4 Alterar plano customizado	25
3.2.1.5 Excluir plano customizado	25
3.2.2 Diagrama de classes	26
3.2.2.1 Pacote br.com.furb.tagarela.game.controler.....	26
3.2.2.2 Pacote br.com.furb.tagarela.game.view.....	27
3.2.2.3 Pacote br.com.furb.tagarela.game.model	29
3.2.2.4 Pacote br.com.furb.tagarela.game.util.....	31
3.2.2.5 Pacote br.com.furb.tagarela.model	32
3.2.3 Diagrama de sequência	34
3.3 IMPLEMENTAÇÃO	34
3.3.1 Técnicas e ferramentas utilizadas.....	35
3.3.2 Implementação do jogo 2D de letras/números.....	35

3.3.2.1 Classe PrincipalJogo.....	35
3.3.2.2 Classe Gerenciador.....	36
3.3.2.3 Plano customizado	38
3.3.2.4 Tela do jogo	39
3.3.3 Operacionalidade da implementação	48
3.3.3.1 Tela inicial	49
3.3.3.2 Incluir plano customizado	49
3.3.3.3 Alterar plano customizado.....	50
3.3.3.4 Jogando plano	51
3.4 RESULTADOS E DISCUSSÃO	53
3.4.1 Usabilidade.....	54
3.4.2 Memória e desempenho	54
3.4.2.1 Consumo de memória	54
3.4.2.2 Desempenho do jogo	56
3.4.3 Comparativo entre o trabalho desenvolvido e os trabalhos correlatos.....	57
4 CONCLUSÕES.....	58
4.1 EXTENSÕES	58
REFERÊNCIAS BIBLIOGRÁFICAS	60
APÊNDICE A – Questionário de avaliação	63
APÊNDICE B – Testes com criança com necessidades especiais.....	64

1 INTRODUÇÃO

Atualmente, com a evolução dos *smartphones* e *tablets* juntamente com seus sistemas operacionais, é possível realizar atividades as quais só eram possíveis serem realizadas em computadores. Isso se deve a dois fatores: evolução constante dos *hardwares* dos aparelhos e o crescimento no número de aplicativos desenvolvidos.

Com esta evolução observasse que no mercado de sistemas operacionais, o grande destaque é o sistema operacional Android desenvolvido pela Google. Segundo o estudo realizado pela ABI Research, no final de 2013, existirá cerca de 1,4 bilhões de *smartphones* e 268 milhões de *tablets* em uso no mundo, desses 57% dos *smartphones* e 28% dos *tablets*, serão equipados com plataforma Android (ABI RESEARCH, 2013).

Este crescimento reflete também no desenvolvimento de aplicativos para dispositivos móveis. Um dos tipos de aplicativos que está evoluindo em todos os aspectos são os jogos. Esta crescente evolução tende a tornar os jogos de alta qualidade mais comuns. Desta forma, muitas instituições de ensino estão voltando seus olhares para esta tendência, onde visam maximizar seus resultados utilizando aplicativos e jogos para estimularem o raciocínio e o desenvolvimento de crianças e de pessoas com necessidades especiais. Existem as instituições que utilizam tecnologia assistiva para desenvolver a autonomia e independência de pessoas com necessidades especiais.

De acordo com Melo (2006, p. 62), “Tecnologia assistiva são recursos e serviços que visam facilitar o desenvolvimento de atividades diárias por pessoas com deficiência. Procuram aumentar as capacidades funcionais e assim promover a independência e a autonomia de quem as utiliza”. Hoje em dia existem poucos aplicativos e jogos sendo desenvolvidos utilizando a tecnologias assistivas como forma de metodologia de ensino, ou seja, muitas das atividades a serem realizadas por pessoas com necessidades especiais ainda são manuais.

Diante do exposto, este trabalho desenvolveu um protótipo de jogo 2D para plataforma Android voltado para tecnologia assistiva, onde explorasse o aspecto pedagógico/lúdico em computação aplicada, tendo como cenário um jogo 2D que manipula letras e números. A arquitetura do desenvolvimento do jogo 2D utiliza uma modelagem orientada a componentes, permitindo um baixo acoplamento e reutilização dos componentes entre diversos objetos deste jogo.

1.1 OBJETIVOS DO TRABALHO

O objetivo deste trabalho é disponibilizar um protótipo de um jogo 2D voltado para tecnologia assistiva utilizando os recursos disponíveis da plataforma Android.

Os objetivos específicos do trabalho são:

- a) disponibilizar um jogo 2D que manipula letras e números;
- b) especificar uma arquitetura com modelagem orientada a componentes, onde deverá ter baixo custo de acoplamento para futuros jogos a serem desenvolvidos;
- c) propiciar um ambiente que auxilie o desenvolvimento e evolução de pessoas com necessidades especiais e crianças em fase de alfabetização;
- d) disponibilizar diversos cenários utilizando uma variedade de imagens e áudios.

1.2 ESTRUTURA DO TRABALHO

Este trabalho está dividido em quatro capítulos. O segundo capítulo aborda a fundamentação teórica necessária para compreensão deste trabalho.

No terceiro capítulo são apresentadas as etapas de desenvolvimento do protótipo do jogo 2D. Primeiramente são apresentados os requisitos da aplicação. Posteriormente é mostrada a especificação do protótipo do jogo 2D com os diagramas desenvolvidos. No passo seguinte são descritas as ferramentas e técnicas utilizadas na implementação e também a operacionalidade do protótipo do jogo 2D. Por fim são apresentados os resultados e discussão.

No quarto capítulo são apresentadas as conclusões do presente trabalho e também sugestões para trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

A seção 2.1 trata sobre os conceitos relacionados a tecnologia assistiva e a jogos educacionais. A seção 2.2 procura esclarecer os conceitos relacionados a modelagem orientada a componentes. A seção 2.3 procura esclarecer os conceitos relacionados ao sistema operacional Android e seus recursos que serão utilizados neste trabalho. Por fim, a seção 2.4 descreve os três trabalhos correlatos referentes ao jogo 2D que será desenvolvido.

2.1 TECNOLOGIA ASSISTIVA E JOGOS EDUCACIONAIS

Tecnologia assistiva é uma área do conhecimento de característica interdisciplinar que objetiva promover a funcionalidade, relacionada à atividade e participação de pessoas com necessidades especiais, visando sua autonomia, independência, qualidade de vida e inclusão social (COMITÊ DE AJUDAS TÉCNICAS, 2007).

Segundo Manzini (2005, p. 82), os recursos de tecnologia assistiva estão muito próximos do dia-a-dia. Ora eles causam impactos devido à tecnologia que apresentam, ora passam quase despercebidos. Ou seja, pode-se chamar de tecnologia assistiva uma bengala, utilizada para proporcionar conforto e segurança no momento de caminhar, bem como um aparelho de amplificação utilizado por uma pessoa com surdez moderada ou mesmo veículo adaptado para uma pessoa com deficiência.

Num sentido amplo percebe-se que a evolução tecnológica caminha na direção de tornar a vida mais fácil. Atualmente, diversas ferramentas foram especialmente desenvolvidas para favorecer e simplificar as atividades pretendidas, tais como talheres, canetas, computadores, controle remoto, automóveis, telefones celulares e relógios (SANTOS et al., 2012). “Para as pessoas sem deficiência, a tecnologia torna as coisas mais fáceis. Para as pessoas com deficiência, a tecnologia torna as coisas possíveis” (RADABAUGH, 1993).

Hoje a tecnologia proporciona inúmeros recursos que podem ser utilizados no aprendizado de crianças e pessoas com necessidades especiais. Entre elas destaca-se os jogos educacionais com finalidades pedagógicas, pois promovem situações de aprendizagem e aumentam a construção do conhecimento, introduzindo atividades lúdicas e prazerosas (IAVORSKI; JUNIOR, 2008). “A estimulação, a variedade, o interesse, a concentração e a motivação são igualmente proporcionados pela situação lúdica...” (MOYLES, 2002, p. 21).

Jogos educacionais permitem que crianças em fase de alfabetização assimilem e familiarizem com as letras, números, cores, formas geométricas e desenvolvem a percepção de que existe uma lógica. A presença dos jogos proporcionam uma boa brincadeira e estímulo ao desenvolvimento, tornando o aprendizado algo interessante (VINHAS, 2013).

O Tagarela (2013) é um projeto que surgiu com intuito de disponibilizar uma forma de comunicação alternativa aos pacientes, principalmente em crianças com necessidades especiais e limitações fonoarticulatórias (TAGARELA, 2013). Neste projeto foram elaborados e desenvolvidos os conceitos de planos, pranchas e símbolos.

Os símbolos são compostos por uma imagem que representa uma ação, local, evento ou figura qualquer, e também um áudio para assimilar a imagem (TAGARELA, 2013). As pranchas são agrupamentos de símbolos que possibilitam o tutor e o paciente interagem de forma que haja uma evolução na capacidade de comunicação do paciente (TAGARELA, 2013). Os planos são agrupamentos de pranchas que possuem mesma finalidade de aprendizado (TAGARELA, 2013).

2.2 MODELAGEM ORIENTADA A COMPONENTES

Modelagem orientada em componentes é a utilização de técnicas de Desenvolvimento Baseado em Componentes (DBC) como metodologia de desenvolvimento. O objetivo final é que o *software* seja capaz de acoplar e fazer funcionar, integrada e harmoniosamente, componentes diversos, que podem ser ou ter sido construído com as mais diversas tecnologias e ferramentas, criando-se uma interface padrão para conexão destes componentes.

O DBC apresenta-se como uma técnica para sistematizar o desenvolvimento de software a partir de partes pré-construídas. Esta abordagem permite que *softwares* bem definidos tenham características de fácil atualização, onde apenas os componentes são adicionados, removidos ou substituídos, sem que haja a necessidade da manutenção completa (SPAGNOLI; BECKER, 2003, p. 5).

Um componente é definido como uma unidade de *software* independente, que encapsula dentro de si seu projeto e implementação, e oferece interfaces bem definidas para o meio externo. Para Brown e Wallnau (1998, p. 38), “um componente é uma parte não trivial, quase independente, e substituível de um sistema, que cumpre uma função clara dentro do contexto de uma arquitetura bem definida”.

2.3 ANDROID

Android é um conjunto de *software open-source* para telefones móveis que inclui um sistema operacional, *middleware* e aplicativos chave. Android foi originado por um grupo de empresas conhecido como *Open Handset Alliance* (OHA), liderada pelo Google. Hoje o grupo OHA é composto por mais de 40 empresas, das quais são encarregadas da manutenção e desenvolvimento do Android (ANDROID OPEN SOURCE PROJECT, 2013). A versão

mais recente existente no mercado é 4.2 Jelly Bean que esta sendo divulgada como a versão mais rápida e simplificada (ANDROID, 2013).

O *Android Software Development Kit* (SDK) é o kit de desenvolvimento que disponibiliza as ferramentas e *Application Programming Interface* (API) necessárias para desenvolver aplicações para a plataforma Android, utilizando a linguagem programação Java. Um recurso que é bem explorado no desenvolvimento de jogos é o *framework* multimídia do Android, que inclui suporte para reprodução de diversos tipos de mídias comuns, de modo que vídeos, áudios e imagens podem ser facilmente integrados nas aplicações (ANDROID DEVELOPERS, 2013a).

`MediaPlayer` é umas das API mais importantes do *framework* multimídia do Android, onde uma instância deste objeto pode facilmente buscar, decodificar e reproduzir áudios e vídeos com configuração mínima. Com este recurso é possível reproduzir arquivos oriundos das seguintes fontes: interno do aparelho, *Uniform Resource Locator* (URL) interna e URL externa (ANDROID DEVELOPERS, 2013b). Para imagens o Android disponibiliza a API *Graphics*, que dispõem de vários recursos que possibilitam a manipulação de imagens, das quais destaca-se a classe *Bitmap* que possui diversos métodos ágeis para carregar, editar e exibir imagens em qualquer aplicativo (ANDROID DEVELOPERS, 2013c).

A classe `Base64` é uma classe utilitária do pacote `util` do Android. Ela é responsável por codificar e decodificar as representações em `Base64` de dados binários (ANDROID DEVELOPERS, 2013f). Esta codificação geralmente é utilizada quando se necessita realizar transferência de dados binários por meios que lidam apenas com textos.

Hoje o Android possui suporte para os seguintes formatos de mídia (ANDROID DEVELOPERS, 2013d):

- a) áudio: *Advanced Audio Coding Low Complexity* (AAC LC), *High-Efficiency Advanced Audio Coding* (HE-AAC), *Advanced Audio Coding Enhanced Low Delay* (AAC-ELD), *Adaptive Multi-Rate* (AMR), *Free Lossless Audio Codec* (FLAC), *Moving Picture experts group-1/2 audio layer 3* (MP3), *Musical Instrument Digital Interface* (MIDI) e *Pulse-Code Modulation* (PCM);
- b) vídeo: *Moving Picture Experts Group 4* (MPEG-4) e *Moving Picture Experts Group 4 Simple Profile* (MPEG-4 SP);
- c) imagem: *Joint Photographic Experts Group* (JPEG) e *Graphics Interchange Format* (GIF), *Portable Network Graphics* (PNG) e *Bitmap*.

2.4 TRABALHOS CORRELATOS

Esta seção apresenta trabalhos relacionados com o tema de pesquisa aqui proposto. Serão destacados as principais características do trabalho do Fabeni (2012), da ferramenta Dibugrama (GLOBANT LABS, 2013) e do jogo Desenhe e Aprenda a Escrever (FIZZBRAIN 2013).

2.4.1 Tagarela: Aplicativo para Comunicação Alternativa no iOS

Fabeni (2012) desenvolveu um aplicativo para comunicação alternativa para a plataforma iOS. O aplicativo tem como principal objetivo criar um ambiente onde o fonoaudiólogo, o seu paciente e o tutor deste paciente possam interagir de forma que haja uma evolução na capacidade de comunicação. Tudo isso através de planos de atividades elaborados pelo fonoaudiólogo em conjunto com o tutor do paciente. Estes planos tem como objetivo estimular a capacidade de comunicação através da utilização dos recursos multimídias presentes na plataforma iOS (FABENI, 2012, p. 16).

Esta ferramenta possui diversos recursos que foram diagnosticadas a partir de dificuldades encontradas no tratamento de pessoas com necessidades especiais, onde a maioria das atividades executadas nos pacientes eram manuais. Entre estas características destaca-se as seguintes: possibilidade de criar símbolos personalizados, associar áudio aos símbolos, trocar mensagens entre as pessoas envolvidas, criar plano de atividades, possibilidade de impressão das pranchas, histórico de observações do paciente, entre outras. O aplicativo pode ser visto na Figura 1, onde é apresenta a tela com os símbolos que compõem um plano escolhido.

Figura 1 - Símbolos que compõem o plano escolhido



Fonte: Fabeni (2012, p. 79).

Este trabalho, o de Fabeni (2012), se transformou num projeto que continua em desenvolvimento (TAGARELA, 2013).

2.4.2 Dibugrama

Dibugrama é um jogo desenvolvido pela Globant Labs para a Associação de Síndrome de *Down* da República Argentina (2013). Ele visa estimular o desenvolvimento de crianças e pessoas com necessidades especiais. Com cenários coloridos e divertidos, a criança deve localizar os objetos que desaparecem do cenário, selecionando e movendo os objetos da área inicial para a sua posição final, posicionando todos os objetos é concluído corretamente o nível (GLOBANT LABS, 2013).

O aplicativo foi desenvolvido para plataforma Android com a linguagem de programação Java, onde foi utilizado a biblioteca `libgdx`, uma plataforma que envolve todas as chamadas para `OpenGL` de acordo com o *hardware* que o jogo está sendo executado. O aplicativo pode ser visto na Figura 2, onde é apresenta a tela com um dos cenários que o jogo possui.

Figura 2 - Cenário de um dos níveis do jogo



Fonte: Globant Labs (2013).

2.4.3 Desenhe e Aprenda a Escrever

Desenhe e Aprenda a Escrever é um jogo educativo desenvolvido por FizzBrain (2013). O aplicativo surgiu de dois americanos com quase 50 anos de experiência em educação infantil, onde empregaram as técnicas educacionais mais avançadas, usadas diariamente nas melhores escolas americanas, para ensinar crianças de todas as idades a escrever (FIZZBRAIN, 2013).

O aplicativo tem como principal objetivo propiciar um ambiente onde o usuário possa se entreter e ao mesmo tempo aprender a escrever as letras, números, animais, insetos e até mesmo textos livres. Ele também possibilita a escolha de diversas texturas para serem utilizadas durante o desenho do objeto presente na tela.

O aplicativo foi desenvolvido para plataforma iOS com a linguagem de programação Objective-C. O aplicativo pode ser visto na Figura 3, onde é apresentada a tela principal e um dos cenários do aplicativo.

Figura 3 - Aplicativo Desenhe e Aprenda a Escrever



Fonte: FizzBrain (2013).

3 DESENVOLVIMENTO

Neste capítulo são detalhadas as etapas do desenvolvimento do jogo 2D. São apresentados os principais requisitos, a especificação, a implementação e ao final são apresentados os resultados e discussão. Os conceitos relacionados com `planos`, `pranchas` e `símbolos` serão utilizados do Tagarela (2013), seção 2.4.1, e seguindo os conceitos descritos em Tecnologias Assistiva e Jogos Educacionais, seção 2.1. Os termos `caçador` e `presa` representam respectivamente a relação entres a imagem apresentada quando o `Usuário` toca na tela (`caçador`) e os pontos de interesse existentes no conteúdo do `símbolo` (`presa`) utilizados nos jogos nas `pranchas`.

3.1 REQUISITOS PRINCIPAIS DO PROBLEMA A SER TRABALHADO

O jogo 2D letras/números voltado para tecnologia assistiva deverá:

- a) utilizar vários cenários e atividades, que permitam trabalhar no mínimo com todas as letras e números da língua portuguesa (Requisito Funcional - RF);
- b) permitir que o usuário interaja com as atividades através do toque (RF);
- c) permitir que o usuário crie `planos` customizados (RF);
- d) permitir que o usuário altere `planos` customizados (RF);
- e) permitir que o usuário exclua `planos` customizados (RF);
- f) apresentar ao usuário o áudio do `símbolo` quando a `prancha` for concluída (RF);
- g) ser implementado utilizando a linguagem de programação Java (Requisito Não Funcional - RNF);
- h) ser implementado utilizando o ambiente de desenvolvimento Eclipse (RNF);
- i) ser um módulo integrado do aplicativo Tagerela (RNF);
- j) utilizar os `planos` presentes na base de dados local do aplicativo Tagarela (RNF);
- k) rodar no sistema operacional Android 4.0 e em suas versões superiores (RNF);
- l) seguir o guia de boas práticas de desenvolvimento definido pela Google (ANDROID DEVELOPERS, 2013e) (RNF);

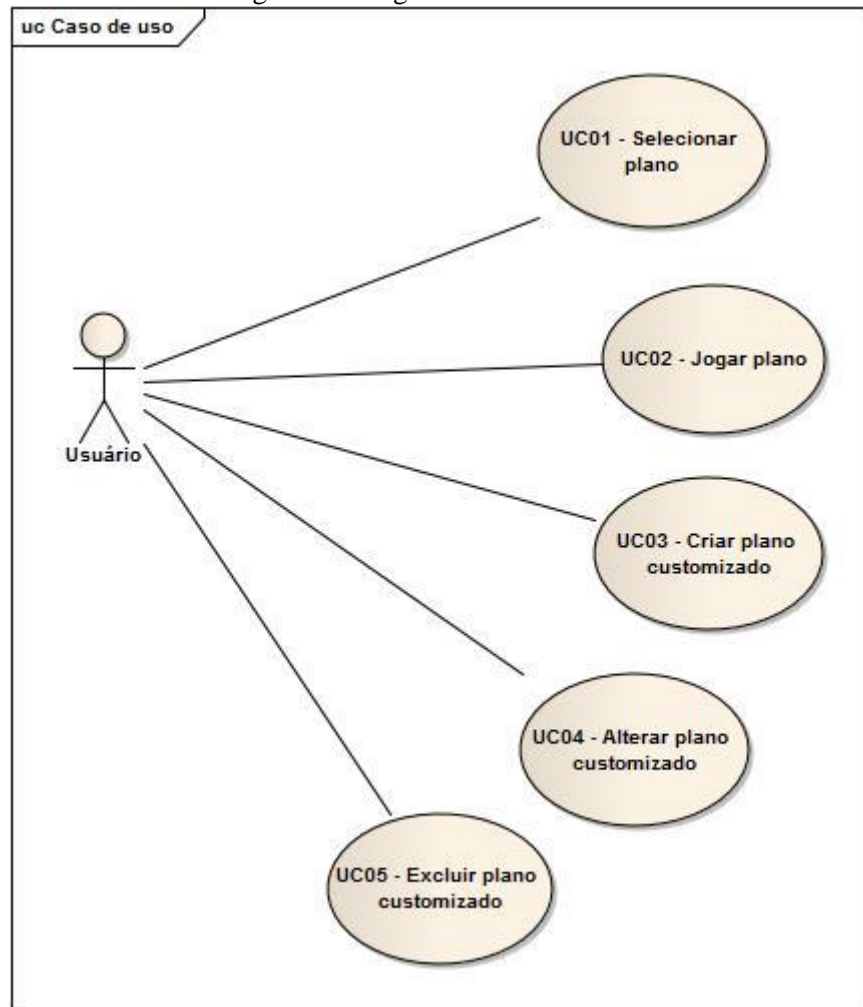
3.2 ESPECIFICAÇÃO

A especificação deste trabalho foi feita utilizando modelagem de diagrama de casos de uso, diagrama de classes e diagrama de sequência, todos da *Unified Modeling Language* (UML). A ferramenta Enterprise Architect foi utilizada na especificação. A seguir são apresentados os diagramas.

3.2.1 Diagrama de casos de uso

Nesta seção são descritos os casos de uso de todas as funcionalidades do jogo 2D. Foi identificado apenas um ator que terá contato com o jogo, o ator *Usuário*. Na Figura 4 pode-se observar o diagrama de casos de uso com o ator e os casos de uso.

Figura 4 - Diagrama de casos de uso



3.2.1.1 Selecionar plano

Este caso de uso descreve qual é a relação entre o *Usuário* e a funcionalidade que possibilita selecionar um *plano*. O Quadro 1 descreve em detalhes este caso de uso.

Quadro 1 - Caso de uso selecionar plano

UC01 – Selecionar plano	
Pré-condições	Usuário estar logado na aplicação Tagarela (MARCO, 2013) e possuir base de dados sincronizada.
Cenário Principal	<ol style="list-style-type: none"> 1. A aplicação Tagarela (MARCO, 2013) apresenta os planos disponíveis para o Usuário. 2. O Usuário seleciona o plano desejado. 3. O Usuário clica no botão Jogar Plano.
Fluxo Alternativo	<ol style="list-style-type: none"> 1. O Usuário clica no botão Jogar Plano. 2. Abre-se a tela principal do jogo com todos os planos disponíveis para serem jogados. 3. O Usuário seleciona o plano desejado. 4. O Usuário clica no botão Escrever.
Pós-condições	A tela de jogar plano é aberta para o Usuário.

3.2.1.2 Jogar plano

Este caso de uso descreve qual é a relação entre o Usuário e a funcionalidade de jogar plano. O Quadro 2 descreve em detalhes este caso de uso.

Quadro 2 - Jogar plano

UC02 – Jogar plano	
Pré-condições	Usuário deve ter executado o caso de uso UC01.
Cenário Principal	<ol style="list-style-type: none"> 1. O jogo exibe na parte superior da tela a pré-visualização de todos os símbolos contidos nas pranchas do plano. 2. O jogo aplica efeito de entrada, rotacionando de forma crescente símbolo contido na prancha seguinte. 3. O jogo mantém destacado em vermelho o símbolo atual na pré-visualização de pranchas contido na parte superior da tela. 4. O jogo exibe todos os pontos a qual o Usuário deverá passar para concluir a prancha. 5. O Usuário toca na tela sobre qualquer parte do símbolo e o jogo começa desenhar a trajetória efetuada. 6. O Usuário completa todas os pontos existentes dentro do símbolo. 7. O jogo reproduz o áudio do símbolo desenhado pelo Usuário. 8. O jogo aplica efeito de rotação onde símbolo irá diminuindo de tamanho até que não seja mais possível sua visualização. 9. O jogo exibe símbolo escrito pelo Usuário na parte inferior da tela. 10. Se ainda houver pranchas para serem jogadas, volta ao passo 2 do cenário principal. 11. O Usuário conclui todos as pranchas contidas no plano.
Fluxo Alternativo	Caso o Usuário não deseje concluir o plano, este deve clicar no botão voltar.
Exceção	No passo 4 do cenário principal, caso o Usuário toque em uma área fora das limitações do símbolo, o jogo irá fazer com que o dispositivo móvel vibre.
Pós-condições	O Jogo irá exibir uma mensagem parabenizando o Usuário pelo conclusão do plano.

3.2.1.3 Criar plano customizado

Este caso de uso descreve qual é a relação entre o Usuário e a funcionalidade de manter plano. O Quadro 3 descreve em detalhes este caso de uso.

Quadro 3 - Criar plano customizado

UC03 – Criar plano customizado	
Pré-condições	Usuário deverá estar com a tela principal do jogo aberta.
Cenário Principal	<ol style="list-style-type: none"> 1. O Usuário clica no botão Criar Plano. 2. O jogo exibe a tela de manutenção de planos customizados no modo de inclusão, mantendo o botão Excluir desabilitado. 3. O Usuário informa o nome do novo plano. 4. O Usuário informa os números e letras desejados para o plano. 5. O Usuário clica no botão Gravar. 6. O jogo grava o plano customizado na base de dados local da aplicação Tagarela (MARCO, 2013).
Fluxo Alternativo	Caso o Usuário deseje cancelar o cadastramento do novo plano, este deve clicar no botão Cancelar.
Exceção	No passo 4 do cenário principal, caso o Usuário informe conteúdos inválidos, o jogo ao gravar irá ignorá-los.
Pós-condições	A tela de manutenção de planos customizados é fechada.

3.2.1.4 Alterar plano customizado

Este caso de uso descreve qual é a relação entre o Usuário e a funcionalidade de manter plano. O Quadro 4 descreve em detalhes este caso de uso.

Quadro 4 - Alterar plano customizado

UC04 – Alterar plano customizado	
Pré-condições	Usuário deverá estar com a tela principal do jogo aberta.
Cenário Principal	<ol style="list-style-type: none"> 1. O Usuário clica no botão Alterar Plano. 2. O jogo exibe a tela de manutenção de planos customizados no modo de edição. 3. O Usuário altera nome do plano. 4. O Usuário altera os números e letras do plano. 5. O Usuário clica no botão Gravar. 6. O jogo atualiza os dados do plano customizado na base de dados local da aplicação Tagarela (MARCO, 2013).
Fluxo Alternativo	Caso o Usuário deseje cancelar a alteração do plano, este deve clicar no botão Cancelar.
Exceção	No passo 4 do cenário principal, caso o Usuário informe conteúdos inválidos, o jogo ao atualizar as informações do plano irá ignorá-los.
Pós-condições	A tela de manutenção de planos customizados é fechada.

3.2.1.5 Excluir plano customizado

Este caso de uso descreve qual é a relação entre o Usuário e a funcionalidade de manter plano. O Quadro 5 descreve em detalhes este caso de uso.

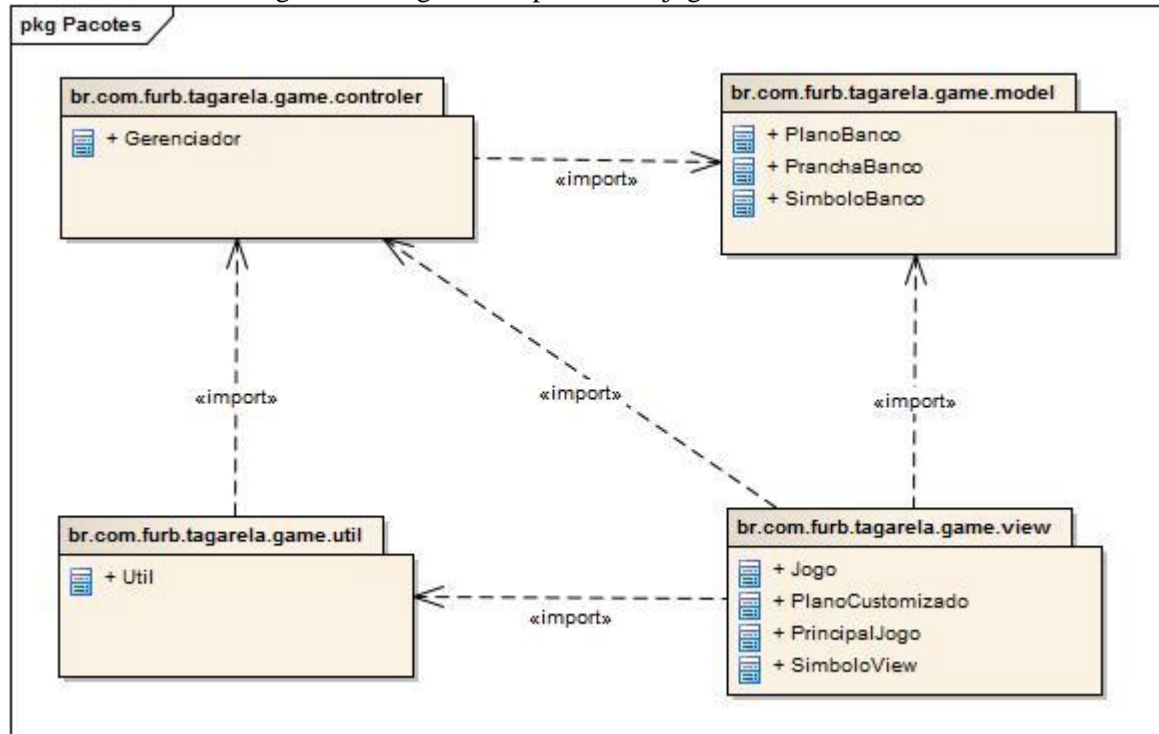
Quadro 5 - Excluir plano customizado

UC05 – Excluir plano customizado	
Pré-condições	Usuário deverá estar com a tela principal do jogo aberta.
Cenário Principal	<ol style="list-style-type: none"> 1. O Usuário clica no botão Alterar Plano. 2. O jogo exibe a tela de manutenção de planos customizados no modo de edição. 3. O Usuário clica no botão Excluir. 4. O jogo exclui o plano customizado da base de dados local da aplicação Tagarela (MARCO, 2013).
Fluxo Alternativo	Caso o Usuário deseje cancelar a alteração do plano, este deve clicar no botão Cancelar.
Pós-condições	A tela de manutenção de planos customizados é fechada.

3.2.2 Diagrama de classes

O diagrama de classes apresenta uma visão de como as classes estão estruturadas e relacionadas. Nesta seção são descritas as classes necessárias para o desenvolvimento desta aplicação, e também as classes necessárias para a integração com o projeto Tagarela desenvolvidas por Marco (2013), descritas na seção 3.2.2.5. A Figura 5 demonstra o diagrama de pacotes da aplicação, bem como as classes envolvidas.

Figura 5 - Diagrama de pacotes do jogo de letras/números



3.2.2.1 Pacote br.com.furb.tagarela.game.controller

O pacote br.com.furb.tagarela.game.controller possui somente a classe Gerenciador. Esta classe é inicializada logo quando o jogo inicia sua execução. Ela é

responsável por carregar e centralizar todos os planos, pranchas e símbolos utilizados no jogo. Desta forma, tendo um papel de gerenciador esta classe irá fornecer as demais classes do jogo mecanismos de iteração e busca para as informações desejadas. Veja a ilustração da classe Gerenciador na Figura 6.

Figura 6 - Diagrama de classes do pacote controller



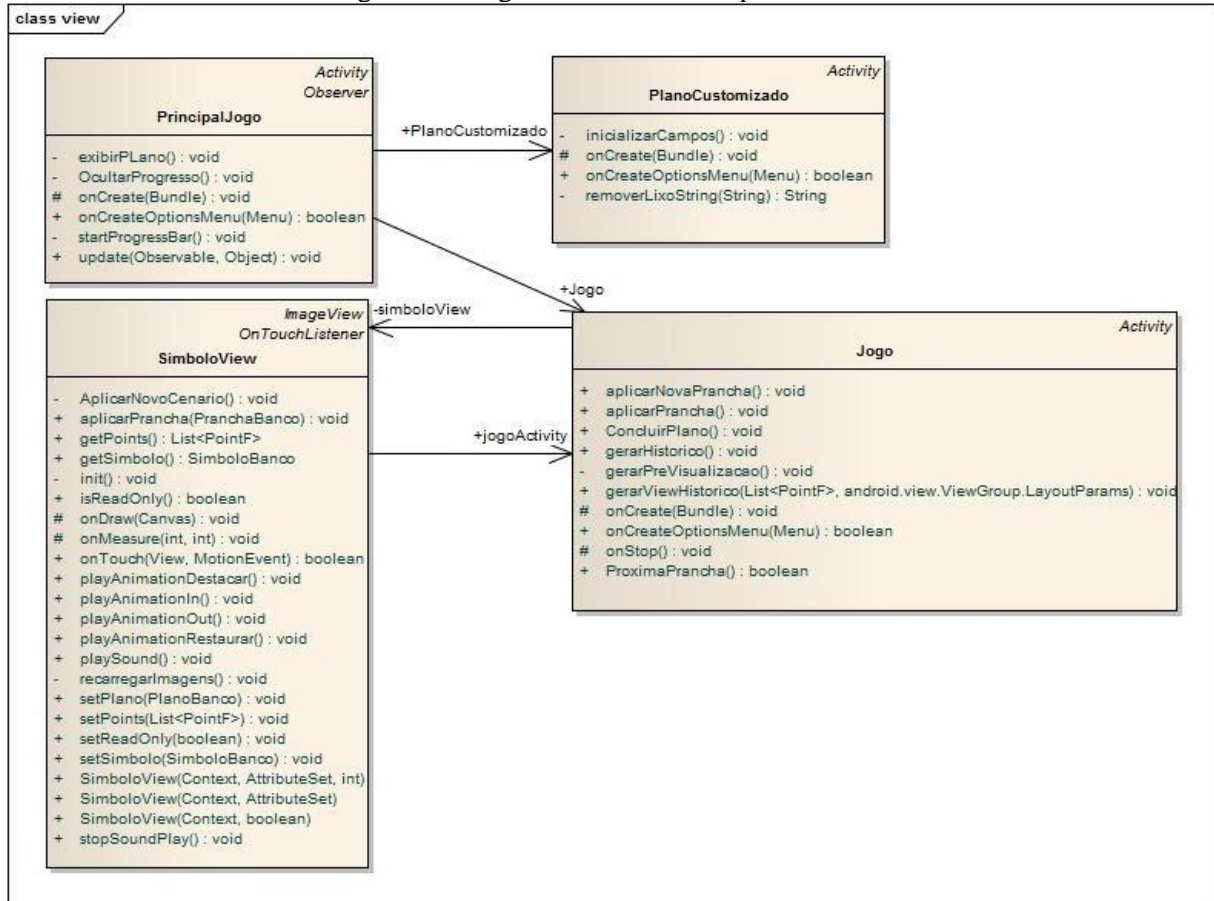
O método `prepararJogo` é responsável por preparar todas as informações necessárias para que o jogo possa ser executado pelo Usuário, informações a quais são extraídas da base de dados do Tagarela desenvolvidas no trabalho de Marco (2013). Este método é chamado pelo método `onCreate` da classe `PrincipalJogo` a qual esta contido no pacote `br.com.furb.tagarela.game.view`. Ao termino deste processo a classe `PrincipalJogo` é notificada sobre sua conclusão.

3.2.2.2 Pacote `br.com.furb.tagarela.game.view`

O pacote `br.com.furb.tagarela.game.view` é o principal pacote do jogo. Nele estão contidas as principais classes do jogo. Por padrão, as classes deste pacote herdam as

funcionalidades da classe `Activity` da API do Android, que gerencia o ciclo de vida da aplicação. A Figura 7 representa o diagrama de classes desse pacote. Todos os atributos privados foram removidos para uma melhor visualização.

Figura 7 - Diagrama de classes do pacote `view`



A classe `PrincipalJogo` apresenta a tela inicial para o Usuário, é a classe que inicia o fluxo do jogo.

A classe `PlanoCustomizado` é responsável pela criação e manutenção dos planos customizados. Após a conclusão das operações realizadas pelo Usuário a classe irá retornar a chamada para a classe `PrincipalJogo`, onde plano customizado já estará disponível para a seleção do Usuário.

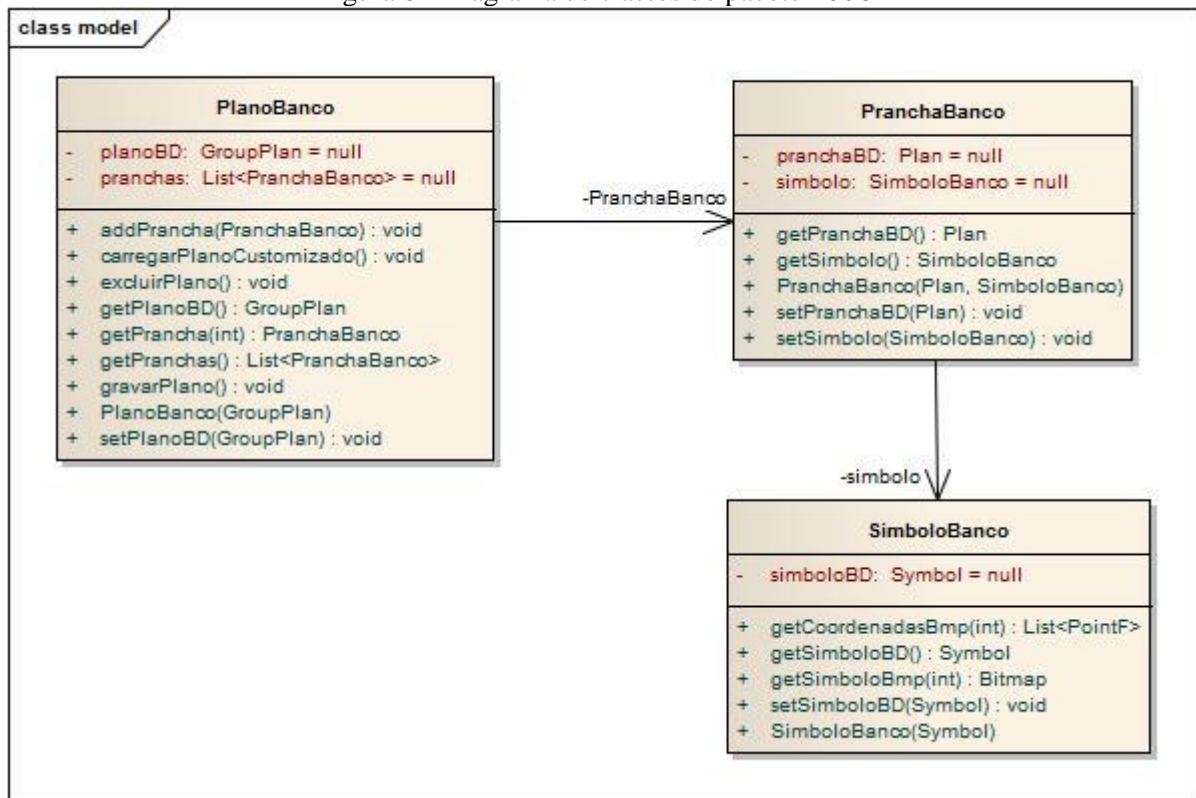
A classe `Jogo` é responsável por controlar as pranchas a serem utilizadas e gerar o histórico das pranchas jogadas pelo Usuário. A classe possui atributos para controlar o plano e a prancha atual, onde, no método `onCreate` recebe o índice do plano selecionado pelo Usuário na classe `PrincipalJogo`, e pelo método `aplicarPrancha` prepara a primeira prancha. Quando uma prancha é finalizada pelo Usuário, a classe chama o método `aplicarNovaPrancha` a qual será reponsável por gerar histórico e preparar a nova prancha.

A classe `SimboloView` é a principal classe do jogo, nela o jogo é efetivamente iniciado. Esta classe estende de `ImageView`, facilitando assim a manipulação de imagens e suas renderizações na tela. Essa classe também implementa a interface `OnTouchListener` para que seja possível capturar o evento de toque na tela efetuado pelo `Usuário`, disponível no método `onTouch`. A classe `SimboloView` recebe via parâmetro do método `aplicarPrancha` a instância da classe `Prancha` a ser jogada pelo `Usuário`, este método é invocado pela classe `Jogo`. A classe `Prancha` irá conter o símbolo a qual será renderizado no centro da tela pelo evento `onDraw`. Após o símbolo ser renderizado o método `buildDrawingCache` é chamado para que o mapa de *pixels* do símbolo seja realizado e armazenado em memória. Já após a execução deste método, este mapa de *pixels* é atribuído ao atributo `drawingCache`. Em cada *pixel* do `Buffer drawingCache` está contido as informações `ARGB`, das quais a principal informação é o `alpha` (no caso o `A`), pois com ele será determinado se o ponto de toque do `Usuário` na tela é válido ou não. A propriedade `alpha` do *pixel* também servirá para determinar os pontos de passagem pelo `Usuário`, pontos das quais serão armazenados no atributo `WayPoints`. Cada toque efetuado na tela pelo `Usuário` é processado no evento `onTouch`, onde nesse evento será validado o *pixel* citado anteriormente, e as coordenadas `x` e `y` da tela serão armazenadas no atributo `points`. No evento `onDraw` a lista do atributo `points` será lida e desenhada na tela, causando assim a impressão de caminho sobre o símbolo. Toda entrada e saída de prancha será animada utilizando a classe `AnimatorSet` e `ObjectAnimator`, das quais permitem aplicar efeitos de rotação e escala sobre os eixos `x` e `y` do objeto, que no caso é a própria classe `SimboloView`. Quando o `Usuário` percorre todos os pontos de passagem, a classe `SimboloView` emite o áudio correspondente do símbolo desenhado pelo método `playSound`.

3.2.2.3 Pacote `br.com.furb.tagarela.game.model`

O pacote `br.com.furb.tagarela.game.model` contém as classes responsáveis por gerenciar e interpretar as informações extraídas do banco de dados do Tagarela (MARCO, 2013). A Figura 8 representa o diagrama de classes desse pacote.

Figura 8 - Diagrama de classes do pacote model



A classe `PlanoBanco` é a classe responsável por controlar e manter o plano. Essa classe recebe na sua criação a referência da classe `GroupPlan`, classe a qual foi importada da aplicação Tagarela (MARCO, 2013), e nela irá conter todas as informações do plano. O atributo `pranchas` conterá a lista de todas as pranchas contidas no plano, a classe `PranchaBanco` é a representação da prancha nesta lista. O método `addPrancha` é responsável por adicionar pranchas a lista `pranchas`. O método `carregarPlanoCustomizado` é responsável por percorrer todos os caracteres do texto customizado e carregar da base de dados do Tagarela (MARCO, 2013) a prancha e o símbolo correspondente. Os métodos `excluirPlano` e `gravarPlano` são responsáveis por excluir e gravar os planos customizados na base, eles são chamados pela classe `PlanoCustomizado` do pacote `br.com.furb.tagarela.game.view`.

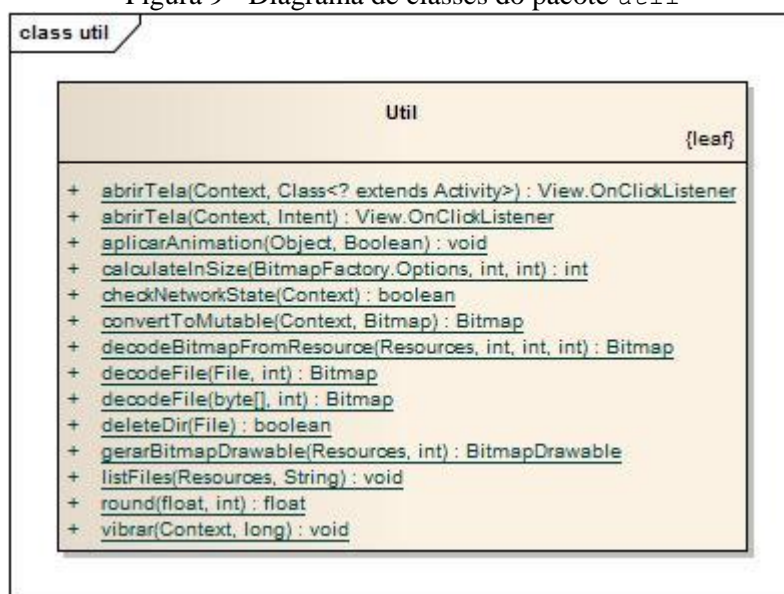
A classe `PranchaBanco` é a classe responsável por controlar e manter a prancha. Essa classe recebe na sua criação a referência da classe `Plan`, classe a qual foi importada da aplicação Tagarela (MARCO, 2013), e nela irá conter todas as informações da prancha. Também receberá uma referência de `SimboloBanco`, tendo assim a referência direta para seu único símbolo.

A classe `SimboloBanco` é a classe responsável por controlar e manter o símbolo. Essa classe recebe na sua criação a referência da classe `Symbol`, classe a qual foi importada da aplicação Tagarela (MARCO, 2013), e nela irá conter todas as informações do símbolo. O método `getSimboloBmp` é responsável por decodificar a imagem em Base64 para o formato `Bitmap`, e também recebe como parâmetro o tamanho final desejado para a imagem. O método `getCoordenadasBmp` é responsável por percorrer todos os *pixels* da imagem e retornar todos que tiverem alpha entre 1 e 254. Os valores 0 e 255 são valores reservados do jogo, pois são utilizados para identificar a área interna (no caso o 255) ou a externa (no caso o 0) da imagem, ou seja, quando o `Usuário` efetuar o toque na tela, o jogo saberá identificar se a área tocada compreende o interior ou o exterior da imagem.

3.2.2.4 Pacote `br.com.furb.tagarela.game.util`

O pacote `br.com.furb.tagarela.game.util` possui somente a classe `Util`. Esta classe é responsável por disponibilizar recursos de utilização comum entre as classes existentes no jogo. Veja a ilustração da classe `Util` na Figura 9.

Figura 9 - Diagrama de classes do pacote `util`

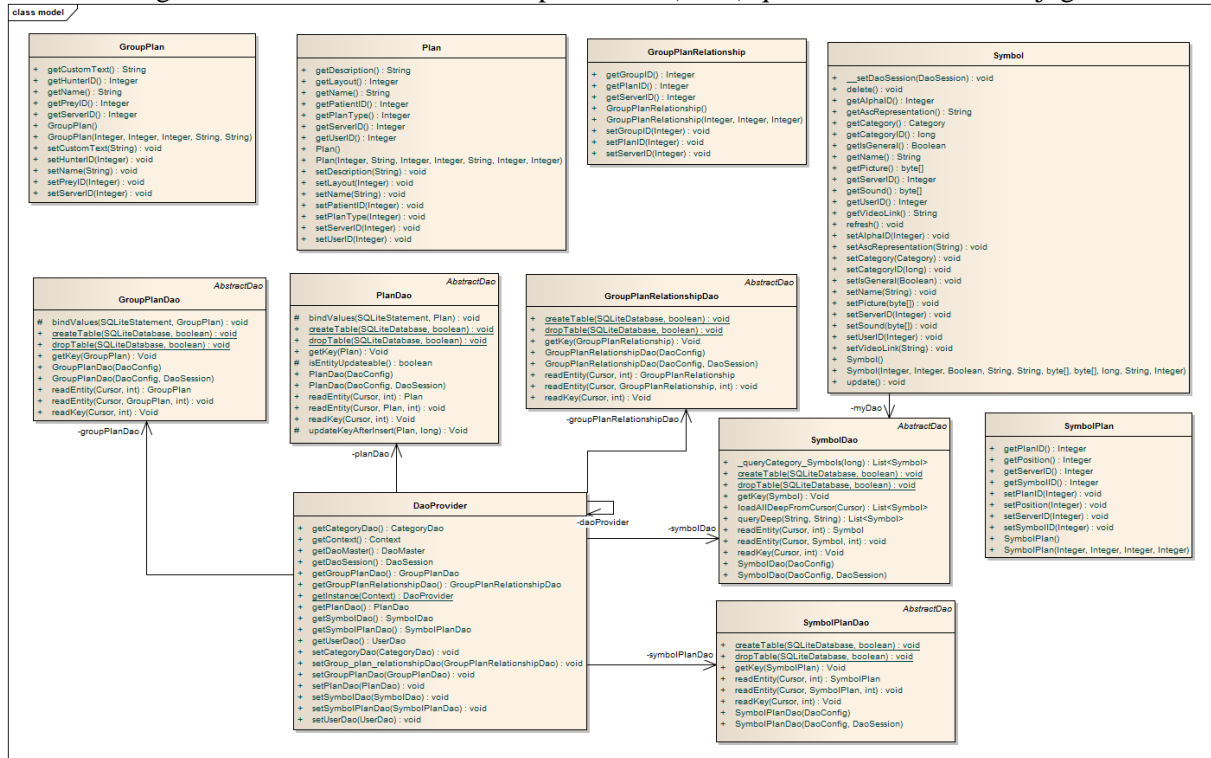


O método `abrirTela` é responsável por abrir qualquer tela da aplicação passando por parâmetro a referência do `Context` que é o contexto da aplicação e uma classe que seja herdada de `Activity`. O método `vibrar` é responsável por vibrar o dispositivo móvel por um determinado período de tempo, tempo a qual é determinado pelo parâmetro `long`.

3.2.2.5 Pacote `br.com.furb.tagarela.model`

Para comunicação com o banco de dados do Tagarela (MARCO, 2013) optou-se pela utilização das classes do pacote `br.com.furb.tagarela.model` desenvolvidas por Marco (2013). Na Figura 10 são apresentadas as classes que foram utilizadas deste pacote. Todos os métodos e atributos privados foram removidos para uma melhor visualização.

Figura 10 - Classes desenvolvidas por Marco (2013) que foram utilizadas no jogo



A classe `daoProvider` é responsável por iniciar a sessão com o banco de dados através do atributo `tagarelaDB` que é do tipo `SQLiteDatabase`. Nesta classe também são criadas todas as tabelas da aplicação no banco de dados. Esta classe é *singleton*, portanto será criada apenas uma vez por execução na aplicação Tagarela (MARCO, 2013).

A classe `GroupPlan` é responsável por possuir as informações da tabela `GROUP_PLAN` existente no bando de dados do Tagarela (MARCO, 2013). Para este trabalho esta classe representará as informações do plano. O atributo `serverID` represente o código único do plano no banco de dados. Os atributos `hunterID` e `preyID` possuem os códigos dos símbolos que representam respectivamente o caçador e a presa.

A classe `GroupPlanDao` é responsável por manter as informações da tabela `GROUP_PLAN` existente no banco de dados do Tagarela (MARCO, 2013). Com esta classe é possível realizar operações de *insert*, *update* e *delete* sobre uma instância da classe `GroupPlan`.

A classe `Plan` é responsável por possuir as informações da tabela `PLAN` existente no bando de dados do Tagarela (MARCO, 2013). Para este trabalho esta classe representará as informações da prancha. O atributo `serverID` represente o código único da prancha no banco de dados.

A classe `PlanDao` é responsável por manter as informações da tabela `PLAN` existente no banco de dados do Tagarela (MARCO, 2013). Com esta classe é possível realizar operações de *insert*, *update* e *delete* sobre uma instância da classe `Plan`.

A classe `GroupPlanRelationship` é responsável por possuir as informações da tabela `GROUP_PLAN_RELATIONSHIP` existente no bando de dados do Tagarela (MARCO, 2013). Nesta classe estará contida a ligação entre o plano e a prancha. O atributo `serverID` representa o código único desta tabela no banco de dados. Os atributos `groupID` e `planID` representam respectivamente o código do plano e da prancha.

A classe `GroupPlanRelationshipDao` é responsável por manter as informações da tabela `GROUP_PLAN_RELATIONSHIP` existente no banco de dados do Tagarela (MARCO, 2013). Com esta classe é possível realizar operações de *insert*, *update* e *delete* sobre uma instância da classe `GroupPlanRelationship`.

A classe `Symbol` é responsável por possuir as informações da tabela `SYMBOL` existente no bando de dados do Tagarela (MARCO, 2013). Para este trabalho esta classe representará as informações do símbolo. O atributo `serverID` represente o código único do símbolo no banco de dados.

A classe `SymbolDao` é responsável por manter as informações da tabela `SYMBOL` existente no banco de dados do Tagarela (MARCO, 2013). Com esta classe é possível realizar operações de *insert*, *update* e *delete* sobre uma instância da classe `Symbol`.

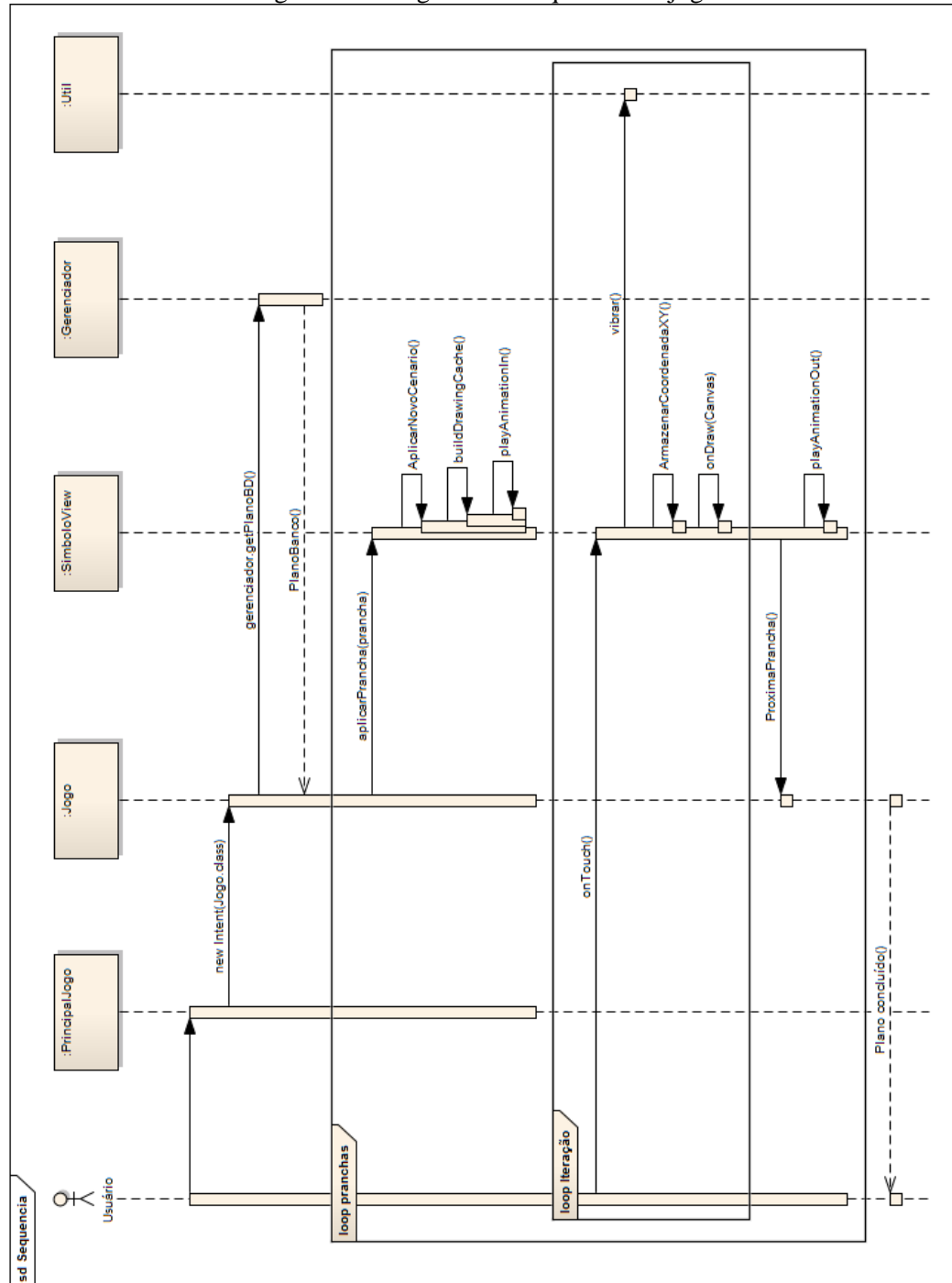
A classe `SymbolPlan` é responsável por possuir as informações da tabela `SYMBOL_PLAN` existente no bando de dados do Tagarela (MARCO, 2013). Nesta classe estará contida a ligação entre a prancha e o símbolo. O atributo `serverID` representa o código único desta tabela no banco de dados. Os atributos `planID` e `symbolID` representam respectivamente o código da prancha e do símbolo.

A classe `SymbolPlanDao` é responsável por manter as informações da tabela `SYMBOL_PLAN` existente no banco de dados do Tagarela (MARCO, 2013). Com esta classe é possível realizar operações de *insert*, *update* e *delete* sobre uma instância da classe `SymbolPlan`.

3.2.3 Diagrama de sequência

O diagrama de sequência demonstra como o `Usuário` interage com o jogo relatado neste trabalho. Veja a ilustração do diagrama de sequência na Figura 11.

Figura 11 - Diagrama de sequência do jogo



3.3 IMPLEMENTAÇÃO

A seguir são mostradas as técnicas e ferramentas utilizadas e a operacionalidade da implementação, assim como detalhes das classes e rotinas.

3.3.1 Técnicas e ferramentas utilizadas

A linguagem de programação utilizada no desenvolvimento do jogo 2D foi o Java. O jogo trabalha com a API de desenvolvimento 4.2.2 do Android, que também é conhecida como Jelly Bean. O ambiente de desenvolvimento utilizado foi o Eclipse 4.2.1 em conjunto com os *plugins* do *Android Development Tools* (ADT). Estas ferramentas Android oferecem os recursos necessários para desenvolver, depurar e executar aplicações diretamente no dispositivo móvel ou em emuladores. Para execução e depuração do jogo foi utilizado também um dispositivo Samsung Galaxy Note 10.1. Para criação das imagens do alfabeto e dos números foi utilizado a ferramenta de edição de imagens Adobe Photoshop CS5.

O depurador disponibilizado pelo ADT, permite que o desenvolvedor execute e depure o código diretamente no dispositivo móvel, aumentando assim a produtividade e a percepção dos testes efetuados. Uma das limitações encontradas foi a lentidão encontrada no uso do simulador, que ainda está longe do desempenho dos dispositivos móveis.

O Samsung Galaxy Note 10.1 possui processador ARM Cortex-A9 Quad Core de 1.4 *Giga Hertz* (GHz), com GPU ARM Mali-400. Possui memória de 2 *Giga Bytes* (GB) de RAM e memória interna de 16GB. Sua resolução é de 800 x 1.280 e 10.1 polegadas, a versão do Android é a 4.0.

3.3.2 Implementação do jogo 2D de letras/números

Abaixo são apresentados as principais partes do código fonte do jogo.

3.3.2.1 Classe `PrincipalJogo`

A classe `PrincipalJogo` do pacote `br.com.furb.tagarela.game.view` é a classe responsável por iniciar a execução do jogo. O primeiro código a ser apresentado, é do método `onCreate`, neste método é carregado o *eXtensible Markup Language* (XML) com o *layout* da tela e aplicado no método `setContentView(R.layout.principal_jogo)`, tornando-o o *layout* o ativo da tela. A classe `PrincipalJogo` foi desenvolvida para ser um *observer* da classe `Gerenciador` do pacote `br.com.furb.tagarela.game.controler`, portanto quando a classe `Gerenciador` inicia o método `prepararJogo` a classe `PrincipalJogo` exibe uma barra de progresso na tela para o *Usuário* até que o método seja concluído. O Quadro 6 apresenta os principais trechos de código do método `onCreate`.

Quadro 6 - Método onCreate da classe PrincipalJogo

37	@Override
38	protected void onCreate(Bundle savedInstanceState) {
39	super.onCreate(savedInstanceState);
40	
41	requestWindowFeature(Window.FEATURE_NO_TITLE);
42	getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,
43	WindowManager.LayoutParams.FLAG_FULLSCREEN);
44	
45	setContentView(R.layout.principal_jogo);
46	
47	progresso = new ProgressDialog(this);
48	handler = new Handler();
49	gerenciador = Gerenciador.getInstance();
50	gerenciador.setContext(this);
51	gerenciador.addObserver(this);
..	...
130	startProgressBar();
131	gerenciador.prepararJogo();
132	}

3.3.2.2 Classe Gerenciador

A classe Gerenciador contida no pacote `br.com.furb.tagarela.game.controller` foi desenvolvida para ser um *singleton*, portanto não pode ser instanciada, para utilizá-la deve ser utilizado o seu método estático `getInstance` que pode ser visto no Quadro 7.

Quadro 7 - Método getInstance da classe Gerenciador

69	public static Gerenciador getInstance(){
70	if (instance == null) {
71	instance = new Gerenciador();
72	}
73	return instance;
74	}

O método `prepararJogo` é responsável por chamar os principais métodos de preparação de artefatos antes que o jogo seja iniciado. Este método utiliza uma `Thread` exclusiva para não consumir a `Thread` principal, evitando assim a sensação de aplicação travada. Ao final o método `notifyObservers` é chamado para que as classes que observam esta classe sejam notificadas e tomem as ações desejadas. Neste caso a classe `PrincipalJogo` será notificada e a barra de progresso até então exibida para o Usuário será ocultada. No Quadro 8 pode ser visto o método `prepararJogo`.

Quadro 8 - Método `prepararJogo` da classe `Gerenciador`

122	<code>public void prepararJogo(){</code>
123	<code> new Thread() {</code>
124	<code> public void run() {</code>
125	<code> inicializarPlanos();</code>
126	<code> CarregarPlanos();</code>
127	<code> CarregarCheckPoints();</code>
128	<code> InicializarBanco();</code>
129	<code> CarregarPlanosBD();</code>
130	<code> setChanged();</code>
131	<code> notifyObservers();</code>
132	<code> }</code>
133	<code> }.start();</code>
134	<code>}</code>

No Quadro 9 e no Quadro 10 é apresentado o método `CarregarPlanosBD`, onde é possível visualizar a leitura da base de dados do Tagarela (MARCO, 2013). Este método também é responsável por instanciar as classes `PlanoBanco`, `PranchaBanco` e `SimboloBanco`, das quais são armazenadas na lista `planos` para posteriormente serem utilizadas durante o jogo.

Quadro 9 - Método `CarregarPlanoBD` da classe `Gerenciador`

299	<code>private void CarregarPlanosBD() {</code>
300	<code> GroupPlanDao planoDAO =</code>
301	<code> DaoProvider.getInstance(null).getGroupPlanDao();</code>
302	
303	<code> GroupPlanRelationshipDao</code>
304	<code> planoXPranchaDAO = DaoProvider.getInstance(null).</code>
305	<code> getGroupPlanRelationshipDao();</code>
306	<code> PlanDao pranchaDAO =</code>
307	<code> DaoProvider.getInstance(null).getPlanDao();</code>
308	
309	<code> SymbolPlanDao</code>
310	<code> pranchaXSimboloDAO = DaoProvider.getInstance(null).</code>
311	<code> getSymbolPlanDao();</code>
312	
313	<code> SymbolDao</code>
314	<code> simboloDAO = DaoProvider.getInstance(null).getSymbolDao();</code>
315	
316	<code> for (GroupPlan planoBD : planoDAO.loadAll()) {</code>
317	<code> PlanoBanco plano = new PlanoBanco(planoBD);</code>
318	
319	<code> for (GroupPlanRelationship planoXPranchaBD :</code>
320	<code> planoXPranchaDAO.queryRaw("where group_ID = ?",</code>
321	<code> planoBD.getServerID().toString())) {</code>
322	
323	<code> for (Plan pranchaBD :</code>
324	<code> pranchaDAO.queryRaw("where server_ID = ?",</code>
325	<code> planoXPranchaBD.getPlanID().toString())) {</code>
326	
327	<code> for (SymbolPlan pranchaXSimboloBD :</code>
328	<code> pranchaXSimboloDAO.</code>
329	<code> queryRaw("where plan_ID = ?",</code>
330	<code> pranchaBD.getServerID().toString())) {</code>

Quadro 10 – Continuação do método `CarregarPlanoBD` da classe `Gerenciador`

331	<code>for (Symbol simboloBD :</code>
332	<code> simboloDAO.</code>
333	<code> queryRaw("where server_ID = ?",</code>
334	<code> pranchaXSimboloBD.</code>
335	<code> getSymbolID().toString())) {</code>
336	
337	<code> SimboloBanco</code>
338	<code> simbolo = new SimboloBanco(simboloBD);</code>
339	
340	<code> PranchaBanco</code>
341	<code> prancha = new PranchaBanco(pranchaBD,</code>
342	<code> simbolo);</code>
343	
344	<code> plano.addPrancha(prancha);</code>
345	<code> }</code>
346	<code>}</code>
347	
348	<code> }</code>
349	<code>}</code>
350	<code>if (!planoBD.getCustomText().equals("")) {</code>
351	<code> plano.carregarPlanoCustomizado();</code>
352	<code>}</code>
353	<code>planosBD.add(plano);</code>
354	<code>}</code>
355	<code>for (Symbol simboloBD :</code>
356	<code> simboloDAO.queryRaw("where alpha_ID > ?",</code>
357	<code> String.valueOf(0))) {</code>
358	
359	<code> SimboloBanco simbolo = new SimboloBanco(simboloBD);</code>
360	<code> checkPointsBD.add(simbolo);</code>
361	<code>}</code>
362	<code>}</code>

O método `getNextServerID` é responsável por buscar da base do Tagarela (MARCO, 2013) o maior `SERVER_ID` existente para a tabela em questão e incrementá-lo para que assim seja retornado. Este método recebe como parâmetro qualquer classe que estenda de `AbstractDao<?, ?>`. No Quadro 11 pode visualizado o código do método `getNextServerID`.

Quadro 11 - Método `getNextServerID` da classe `Gerenciador`

541	<code>public int getNextServerID(AbstractDao<?, ?> tableDAO) {</code>
542	<code> SQLiteDatabase db = DaoProvider.getInstance(null).</code>
543	<code> getDaoMaster().getDatabase();</code>
544	<code> Cursor c = db.rawQuery("SELECT MAX(SERVER_ID) AS MAIOR FROM " +</code>
545	<code> tableDAO.getTablename(), null);</code>
546	<code> c.moveToNext();</code>
547	<code> return c.getInt(c.getColumnIndex("MAIOR")) + 1;</code>
548	<code>}</code>

3.3.2.3 Plano customizado

A classe `PlanoCustomizado` representa a tela que o `Usuário` interage para manter os planos customizados. Esta classe por sua vez interage diretamente com a classe `PlanoBanco`

para atualizar as informações do plano customizado e persisti-lo no banco de dados do Tagarela (MARCO, 2013). Esta ação é realizada pelo método `gravarPlano` da classe `PlanoBanco`, conforme pode ser visualizado no Quadro 12.

Quadro 12 - Método `gravarPlano` da classe `PlanoBanco`

52	public void <code>gravarPlano()</code> {
53	<code>GroupPlanDao planoDAO = DaoProvider.getInstance(null).</code>
54	<code>getGroupPlanDao();</code>
55	<code>GroupPlanRelationshipDao planoXPranchaDAO =</code>
56	<code>DaoProvider.getInstance(null).getGroupPlanRelationshipDao();</code>
57	
58	<code>SQLiteDatabase db = planoXPranchaDAO.getDatabase();</code>
59	
60	<code>ContentValues values = new ContentValues();</code>
61	<code>values.put("Name", planoBD.getName());</code>
62	<code>values.put("Hunter_ID", planoBD.getHunterID());</code>
63	<code>values.put("Prey_ID", planoBD.getPreyID());</code>
64	<code>values.put("Custom_Text", planoBD.getCustomText());</code>
65	
66	<code>db.update(planoDAO.getTablename(), values, "Server_ID = ?",</code>
67	<code>new String[] {planoBD.getServerID().toString()});</code>
68	
69	<code>carregarPlanoCustomizado();</code>
70	}

3.3.2.4 Tela do jogo

A classe `Jogo` é a classe que efetivamente inicia as atividades das pranchas do plano com o Usuário. O Quadro 13 apresenta o trecho de código do método `onCreate`, onde pode-se observar na linha 62 que a classe recebe por parâmetro o `planoindex` contendo o índice do plano a ser jogado. Parâmetro a qual foi previamente estipulado pela classe `PrincipalJogo` após o Usuário ter selecionado o plano na tela principal.

Quadro 13 - Método `onCreate` da classe `Jogo`

51	<code>@Override</code>
52	protected void <code>onCreate(Bundle savedInstanceState)</code> {
53	<code>super.onCreate(savedInstanceState);</code>
54	
55	<code>requestWindowFeature(Window.FEATURE_NO_TITLE);</code>
56	<code>getWindow().setFlags(WindowManager.LayoutParams.FLAG_FULLSCREEN,</code>
57	<code>WindowManager.LayoutParams.FLAG_FULLSCREEN);</code>
58	
59	<code>setContentView(R.layout.jogo);</code>
60	<code>gerenciador = Gerenciador.getInstance();</code>
61	<code>Bundle extras = getIntent().getExtras();</code>
62	<code>plano = gerenciador.getPlanoBD(extras.getInt("planoindex"));</code>

No Quadro 14 é demonstrado o método `gerarPreVisualizacao` da classe `jogo` que tem como objetivo gerar a pré-visualização das pranchas a serem jogadas pelo Usuário. Cada prancha possui uma instância da classe `SimboloBanco` que por sua vez possui a imagem e o caractere do alfabeto ou número a qual está sendo representando. Desta forma

conforme pode observar na linha 223 um `TextView` é criado com um *layout* proporcional a área reservada para pré-visualização. Na linha 231 o valor do caractere é atribuído ao elemento criado e linha 240 adicionado ao *layout* responsável pela área superior da tela.

Quadro 14 - Método `gerarPreVisualizacao` da classe `Jogo`

```

219     private void gerarPreVisualizacao() {
220         jogoLayoutText.removeAllViewsInLayout();
221         int index = 0;
222         for (PranchaBanco p : plano.getPranchas()) {
223             TextView text = new TextView(getApplicationContext());
224
225             android.view.ViewGroup.LayoutParams
226             lp = new LayoutParams(android.view.ViewGroup.
227                                     LayoutParams.WRAP_CONTENT,
228                                     android.view.ViewGroup.LayoutParams.MATCH_PARENT);
229
230             text.setLayoutParams(new LayoutParams(lp));
231             text.setText(p.getSimbolo().getSimboloBD().getName());
232
233             text.setTextSize(60.f);
234
235             if (index == pranchaIndex) {
236                 text.setTextColor(Color.RED);
237             }
238             else
239                 text.setTextColor(Color.BLACK);
240             jogoLayoutText.addView(text);
241             index++;
242         }
243     }

```

O método `gerarPreVisualizacao` além de gerar a pré-visualização das pranchas ele também é responsável manter o Usuário informado da prancha atual, destacando-a em vermelho, conforme pode ser observado nas linhas 235 e 236.

No Quadro 15 é demonstrado o método `gerarHistorico` da classe `Jogo` que tem como objetivo gerar o histórico das pranchas jogadas pelo Usuário. Sua principal funcionalidade é pegar todos os pontos *x* e *y* do caminho efetuado pelo Usuário sobre o símbolo da prancha e redesenha-lo em proporção menor na área inferior da tela.

Ainda no Quadro 15 na linha 278 pode ser verificado a iteração realizada entre todos os pontos do caminho e logo em seguida a lógica aplicada para gerar o valor *x* e *y* proporcional ao *layout* inferior da tela.

No Quadro 15 na linha 288 pode se observar a chamada para o método `gerarViewHistorico` que efetivamente irá criar o novo elemento na área inferior da tela. Os parâmetros enviados foram *points* contendo a lista de pontos *x* e *y* ajustados proporcionalmente para o *layout* e *lParams* que possui a definição de *layout* do novo elemento.

Quadro 15 - Método gerarHistorico da classe Jogo

```

261 public void gerarHistorico(){
262     if (simboloView.getPoints().size() > 0) {
263
264         android.view.ViewGroup.LayoutParams
265         lParams = new LayoutParams(lParamsImg);
266
267         char c = simboloView.getSimbolo().getSimboloBD().
268             getName().toCharArray()[0];
269
270         // Letras Maiúsculas ou Números
271         if (((c >= 65) && ((c <= 90))) ||
272             ((c >= 48) && (c <= 57))) {
273             lParams.width = 70;
274             lParams.height = 70;
275         }
276
277         List<PointF> points = new ArrayList<PointF>();
278         for (PointF pointF : simboloView.getPoints()) {
279             pointF.x = pointF.x * ((float) lParams.width /
280                 simboloView.getWidth());
281
282             pointF.y = pointF.y * ((float) lParams.height /
283                 simboloView.getHeight());
284
285             points.add(pointF);
286         }
287
288         gerarViewHistorico(points, lParams);
289
290         if ((pranchaIndex > 0) &&
291             (plano.getPrancha(pranchaIndex-1).
292                 getSimbolo().getSimboloBD().getName().
293                 equals(" "))) {
294
295             gerarViewHistorico(new ArrayList<PointF>(),
296                 new android.view.ViewGroup.LayoutParams(20, 50));
297         }
298     }
299 }

```

No Quadro 16 pode ser observado o código do método gerarViewHistorico. Na linha 304 pode se observar que é criado um novo objeto gráfico SimboloView e logo em seguida a ele é atribuído os parâmetros lParams com as informações do *layout* e points contendo os pontos x e y do caminho desenhado pelo Usuário. A classe SimboloView foi desenvolvido totalmente orientado a componente, ou seja, pode ser acoplado a qualquer view que implemente o padrão adapter. Na linha 308 o método setReadOnly da classe SimboloView foi chamado recebendo o valor true de parâmetro, indicando assim que não haverá iteração com o Usuário na área em que este componente for adicionado.

Quadro 16 - Método `gerarViewHistorico` da classe `Jogo`

```

301 @SuppressWarnings("NewApi")
302 public void gerarViewHistorico(List<PointF> points,
303     android.view.ViewGroup.LayoutParams lParams){
304     SimboloView img = new SimboloView(getApplicationContext(), true);
305
306     img.setLayoutParams(new LayoutParams(lParams));
307     img.setPoints(points);
308     img.setReadOnly(true);
309     img.setOnTouchListener(null);
310     img.setVisibility(View.VISIBLE);
311
312     if (lParams.width == 70) {
313         jogoLayoutHistorico.setGravity(Gravity.CENTER | Gravity.BOTTOM);
314     }
315     else
316         jogoLayoutHistorico.setGravity(Gravity.CENTER);
317
318     jogoLayoutHistorico.addView(img);
319
320     ObjectAnimator
321     scaleXIn = ObjectAnimator.ofFloat(img, "scaleX", 0f, 1f);
322
323     ObjectAnimator
324     scaleYIn = ObjectAnimator.ofFloat(img, "scaleY", 0f, 1f);
325
326     ObjectAnimator
327     rotateClockWise = ObjectAnimator.ofFloat(img, "rotation", 0f, 360f);
328
329     AnimatorSet set = new AnimatorSet();
330     set.play(scaleXIn).with(rotateClockWise).with(scaleYIn);
331     set.setDuration(1000);
332     set.setStartDelay(0);
333     set.start();
334 }

```

No Quadro 16 também pode se observar a utilização das classes `ObjectAnimator` e `AnimatorSet` responsáveis por realizar a animação de entrada sobre a classe `SimboloView`. A animação aplicada parte do princípio rotacionar em 360 graus o objeto `view` aparentemente inexistente no cenário até que ele se restabeleça com o seu tamanho real.

A classe `SimboloView` é a principal classe do jogo, nela o jogo é efetivamente iniciado. Ela é responsável por exibir o símbolo da prancha no centro da tela, desenhar os pontos de interesse do símbolo com base na propriedade `alpha` dos *pixels*, por capturar o evento de toque na tela efetuado pelo Usuário, por realizar animações com os símbolos da prancha, e por fim exibir o áudio do símbolo desenhado pelo Usuário.

A classe `SimboloView` estende de `ImageView` que por sua vez estende de `View`, portanto ela possui recursos nativos da API do Android para trabalhar com imagens e seguindo a linha de modelagem orientada a componentes ela pode ser acoplada a qualquer `View` que implemente `adapter`. Desta forma, esta classe é criada e acoplada em dois *layouts*

do tipo `LinearLayout` na classe `Jogo`, no *layout* central onde o `Usuário` terá interação direta com o componente, e no *layout* inferior onde `Usuário` não poderá interagir, pois a classe será criada em estado de leitura.

No Quadro 17 é apresentado o método `aplicarPrancha`. Este método é responsável receber uma instancia da classe `PranchaBanco` e aplica-lo no cenário. Quando a `prancha` for a primeira do plano a ser utilizada no cenário invocasse diretamente o método `AplicarNovoCenário`, caso contrário o método `playAnimationOut` é chamado para realizar animação de saída do símbolo anterior.

Quadro 17 - Método `aplicarPrancha` da classe `SimboloView`

```

114 public void aplicarPrancha(PranchaBanco prancha) {
115     boolean primeiraPrancha = this.prancha == null;
116
117     this.prancha = prancha;
118     this.simbolo = prancha.getSimbolo();
119
120     if (primeiraPrancha)
121         AplicarNovoCenario();
122     else
123         playAnimationOut();
124 }
```

No Quadro 18 e no Quadro 19 é apresentado o método `playAnimationOut`. Este método é responsável por realizar a animação de saída do símbolo finalizado pelo `Usuário` na tela. Semelhante ao tratamento existente para o histórico de `pranchas` da classe `Jogo`, porém com diferencial de que ao finalizar sua execução o evento `onAnimationEnd` é disparado e o método `playAnimationIn` é chamado para realizar a animação de entrada da próxima `prancha`.

Quadro 18 - Método `playAnimationOut` da classe `SimboloView`

```

184 @SuppressWarnings("NewApi")
185 public void playAnimationOut() {
186     this.clearAnimation();
187
188     ObjectAnimator
189     scaleYOut = ObjectAnimator.ofFloat(this, "scaleY", 1f, 0f);
190
191     ObjectAnimator
192     scaleXOut = ObjectAnimator.ofFloat(this, "scaleX", 1f, 0f);
193
194     ObjectAnimator
195     rotateCounterClockwise =
196     ObjectAnimator.ofFloat(this, "rotation", 0f, -360f);
197
198     AnimatorSet set = new AnimatorSet();
```


Quadro 21 – Continuação do método `playSound` da classe `SimboloView`

```

340     File file = new File(getContext().
341         getExternalFilesDir(null) + "/atemp.m4a");
342
343     if (!file.exists()) {
344         file.createNewFile();
345     }
346     FileUtils.writeByteArrayToFile(file, b);
347     Uri uri = Uri.fromFile(file);
348     mPlayer.setDataSource(getContext(), uri);
349     mPlayer.setOnCompletionListener(new OnCompletionListener() {
350
351         @Override
352         public void onCompletion(MediaPlayer mp) {
353             // TODO Auto-generated method stub
354             mp.release();
355         }
356     });
357     mPlayer.prepare();
358     mPlayer.start();
359
360 } catch (IOException e) {
361     // TODO Auto-generated catch block
362     e.printStackTrace();
363 }
364 }

```

No Quadro 22 é apresentado o método `recarregarImagens`. Este método é responsável por buscar a imagem do símbolo, transformá-la em `Bitmap`, aplicar a imagem a classe `SimboloView`, buscar os pontos de interesse existentes na imagem, buscar a imagem que irá representar o caçador, e por fim a presa que será exibida nos pontos de interesse na imagem do símbolo. A definição dos termos `caçador` e `presa` podem ser vistos no início do capítulo 3.

Quadro 22 - Método `recarregarImagens` da classe `SimboloView`

```

436 private void recarregarImagens(){
437     if (!simboloCarregado && getWidth() > 0) {
438         this.simboloCarregado = true;
439         if (fimDeJogo) {
440             this.setImageResource(R.drawable.fim);
441             readOnly = true;
442         }
443         else this.setImageBitmap(simbolo.getSimboloBmp(getWidth()));
444         if (!readOnly) {
445             wayPoints = simbolo.getCoordenadasBmp(getWidth());
446             SimboloBanco s = Gerenciador.getInstance().
447                 getCheckPointServerID(plano.getPlanoBD().getHunterID());
448             hunter = s.getSimboloBmp((int) dimWayPoint);
449             s = Gerenciador.getInstance().
450                 getCheckPointServerID(plano.getPlanoBD().getPreyID());
451             prey = s.getSimboloBmp((int) dimWayPoint);
452         }
453     }
}

```

No Quadro 23 é apresentado a implementação do método `getCoordenadasBmp`. Este método é responsável por retornar uma lista de pontos `x` e `y`, a qual indicará o local exato das coordenadas na imagem do símbolo (no caso as presas). Estas coordenadas são identificadas pela propriedade `alpha` do ARBG existente em cada *pixel* da imagem. O método `getPixel` da classe `Bitmap` irá retornar o valor do *pixel* e o método `alpha` da classe `Color` irá interpretar este valor e extrair a informação do `alpha`. A informação estando entre 1 e 254 considera-se uma coordenada válida para gerar o ponto de referência. Os valores 0 e 255 são valores reservados do jogo, pois são utilizados para identificar a área interna (no caso o 255) ou a externa (no caso o 0) da imagem, ou seja, quando o `Usuário` efetuar o toque na tela, o jogo saberá identificar se a área tocada compreende o interior ou o exterior da imagem.

O parâmetro `tamanho` indica qual é o tamanho da classe `SimboloView` na classe `Jogo`. Esta informação será utilizada para realizar o redimensionamento das coordenadas gravadas em resolução 1.000 x 1.000 nas imagens.

Quadro 23 - Método `getSimboloBmp` da classe `SimboloBanco`

```

34 public List<PointF> getCoordenadasBmp(int tamanho){
35     Bitmap bmp = getSimboloBmp(1000);
36
37     List<PointF> points = new ArrayList<PointF>();
38
39     for (int i = 0; i < 1000; i++) {
40         for (int j = 0; j < 1000; j++) {
41             int color = bmp.getPixel(i, j);
42             int alpha = Color.alpha(color);
43
44             if ((alpha > 0) && (alpha < 255)) {
45                 PointF p = new PointF(i, j);
46                 p.x = Util.round(((float) tamanho /
47                               1000f) * p.x, 0);
48
49                 p.y = Util.round(((float) tamanho /
50                               1000f) * p.y, 0);
51
52                 points.add(p);
53             }
54         }
55     }
56     return points;
57 }

```

No Quadro 24 é apresentado o método `onTouch`. Este método é responsável por interceptar o evento de toque na tela efetuado pelo `Usuário` e realizar tratamentos de validação e armazenamento do local tocado na tela. As informações referentes a ação realizada pelo `Usuário` estão contidas no parâmetro `event` da classe `MotionEvent`.

Quadro 24 - Método onTouch da classe SimboloView

```

467 @Override
468 public boolean onTouch(View v, MotionEvent event) {
469     if (readOnly) {
470         return true;
471     }
472     // garante que evento não ultrapasse limites da view
473     if ((event.getX() > 0 && event.getX() <= getWidth()) &&
474         (event.getY() > 0 && event.getY() <= getHeight())) {
475         PointF p = new PointF();
476
477         p.x = event.getX();
478         p.y = event.getY();
479
480         edPointX.setText(String.valueOf(Util.round(p.x, 2)));
481         edPointY.setText(String.valueOf(Util.round(p.y, 2)));
482
483         if (drawingCache == null) {
484             drawingCache = this.getDrawingCache();
485         }
486
487         if (drawingCache != null) {
488             int aRGB = drawingCache.getPixel((int) p.x, (int) p.y);
489             int alpha = Color.alpha(aRGB);
490             if (aRGB != 0) {
491                 points.add(p);
492                 int remove = -1;
493                 for (int i = 0; i < waypoints.size(); i++) {
494                     PointF wayP = waypoints.get(i);
495
496                     // Testa a Bounding Box do waypoint
497                     if (((p.x >= wayP.x-(dimWayPoint/2)) &&
498                         (p.x <= wayP.x+(dimWayPoint/2))) &&
499                         ((p.y >= wayP.y-(dimWayPoint/2)) &&
500                         (p.y <= wayP.y+(dimWayPoint/2)))) {
501                         remove = i;
502                     }
503                 }
504                 if (remove >= 0) {
505                     waypoints.remove(remove);
506                 }
507
508                 invalidate();
509                 if ((remove >= 0) && (waypoints.size() == 0)) {
510                     playSound();
511                     playAnimationDestacar();
512                 }
513             }
514             else
515                 Util.vibrar(getContext(), 100);
516         }
517     }
518     return true;
519 }

```

No Quadro 25 é apresentado o método `onDraw`. Este método é responsável por desenhar todas informações gráficas contidas na instância da classe `Canvas` obtida via parâmetro. O método `super.onDraw(canvas)` irá se encarregar de executar o evento `onDraw`

da classe `ImageView`. A partir deste ponto a rotina começa a desenhar seus controles na tela, tais como o caminho realizado pelo `Usuário` no símbolo, caminho a qual está contido na lista `points`. O caçador e a presa também serão desenhados na tela conforme definição do cenário.

Quadro 25 - Método `onDraw` da classe `SimboloView`

```

375 @Override
376 protected void onDraw(Canvas canvas) {
377     super.onDraw(canvas);
378
379     if ((!simboloCarregado) && (!readOnly))
380         return;
381
382     paint.setColor(Color.RED);
383     PointF pOld = null;
384     float x = 0;
385     float y = 0;
386
387     for (PointF p : points) {
388         if (pOld == null)
389             pOld = p;
390
391         canvas.drawCircle(p.x, p.y, dimPincel, paint);
392         x = p.x;
393         y = p.y;
394         pOld = p;
395     }
396
397     if (!readOnly) {
398         if (points.size() > 0) {
399             canvas.drawBitmap(hunter, x -
400                 ((float) hunter.getWidth()/2f),
401                 y - ((float) hunter.getHeight()/2f), paint);
402         }
403
404         for (PointF p : wayPoints) {
405             canvas.drawBitmap(preY, p.x -
406                 ((float) preY.getWidth()/2f),
407                 p.y - ((float) preY.getHeight()/2f), paint);
408         }
409     }
410 }
411 }

```

3.3.3 Operacionalidade da implementação

Nesta seção é apresentado a utilização do jogo. São demonstrados o funcionamento das telas e a usabilidade. Serão demonstrados os passos necessário que o `Usuário` deverá seguir para interagir com o sistema. As imagens demonstradas nesta seção são do jogo funcionando no Galaxy Note 10.1.

3.3.3.1 Tela inicial

A tela inicial do jogo apresenta ao `Usuário` os planos disponíveis para serem jogados, e também os botões `Escrever` e `Criar Plano`. Nesta tela o `Usuário` poderá selecionar um plano já existente na base, ou criar um novo clicando em `Criar Plano`. Após `Usuário` ter selecionado o plano desejado ele deverá clicar em `Escrever` para iniciar efetivamente o jogo. A Figura 12 apresenta a tela inicial do jogo.

Figura 12 - Tela inicial do jogo



3.3.3.2 Incluir plano customizado

A tela de manutenção de planos customizados é exibida em estado de inclusão para o `Usuário` quando ele clicar no botão `Criar Plano` da tela inicial do jogo. Nesta tela o `Usuário` poderá criar um plano customizado informando o nome do plano e um texto customizado para ser utilizado durante o jogo. Ao final do processo o `Usuário` deverá clicar no botão `Gravar` para concluir a inclusão e gravar as informações do plano customizado no banco de dados do Tagarela (MARCO, 2013). `Usuário` também poderá cancelar a inclusão clicando em `Cancelar`. Após a inclusão o plano customizado criado passará a ser exibido na tela inicial do jogo. A Figura 13 apresenta a tela de manutenção de planos customizados em estado de inclusão.

Figura 13 - Incluir plano customizado



The image shows a software interface for adding a custom plan. It has a blue background with a white title bar at the top. The title 'Incluir Plano' is centered in a large, bold, black font. Below the title, there are two input fields. The first is labeled 'Nome do plano:' and contains the text 'Wagner'. The second is labeled 'Texto customizado:' and contains the text 'Wagner Jean Reetz'. At the bottom of the form, there are three buttons: 'Gravar' (Save), 'Remover' (Remove), and 'Cancelar' (Cancel). The 'Gravar' button is highlighted with a yellow border.

3.3.3.3 Alterar plano customizado

A tela de manutenção de planos customizados é exibida em estado de edição para o Usuário quando ele clicar no botão **Alterar Plano** da tela inicial do jogo. Nesta tela o Usuário poderá editar as informações do nome do plano e do texto customizado. Ao final do processo o Usuário deverá clicar no botão **Gravar** para concluir as alterações e atualizar as informações do plano customizado no banco de dados do Tagarela (MARCO, 2013). Usuário também poderá cancelar a alteração clicando no botão **Cancelar** ou até mesmo excluir o plano clicando no botão **Remover**. Ao voltar para tela inicial do jogo o plano será atualizado na lista de planos. A Figura 14 apresenta a sequência cronologia para realizar uma alteração de plano customizado.

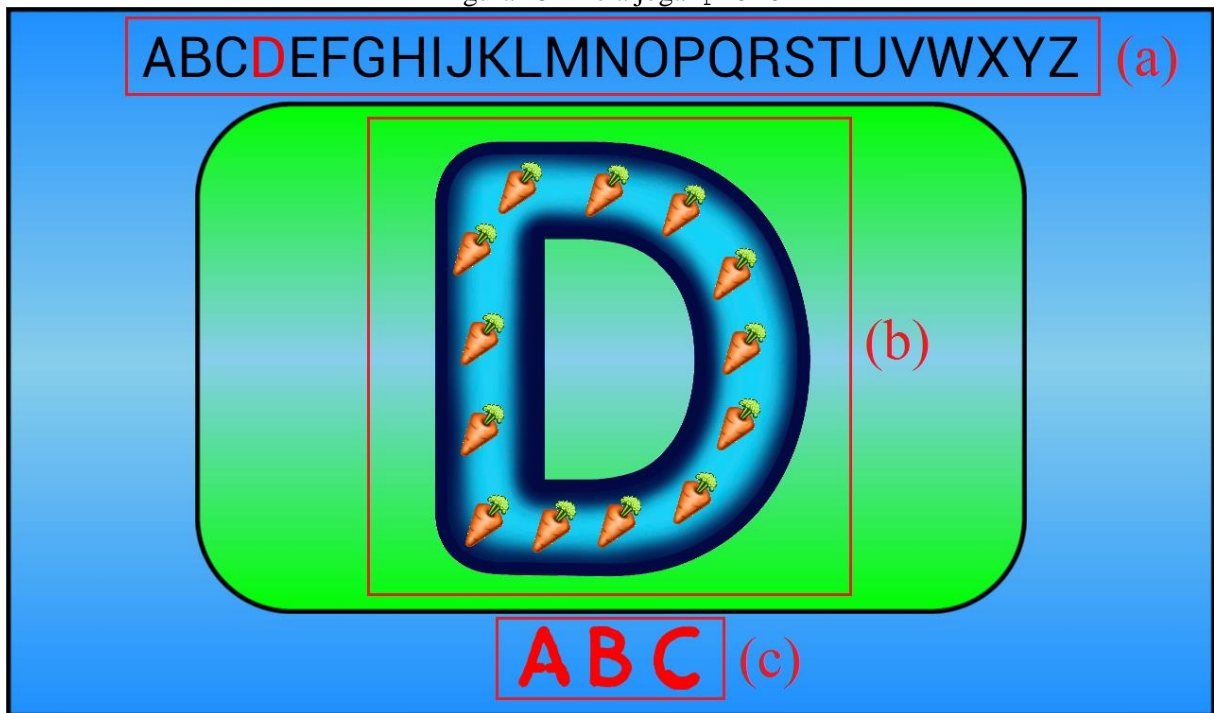
Figura 14 - Alterar plano customizado



3.3.3.4 Jogando plano

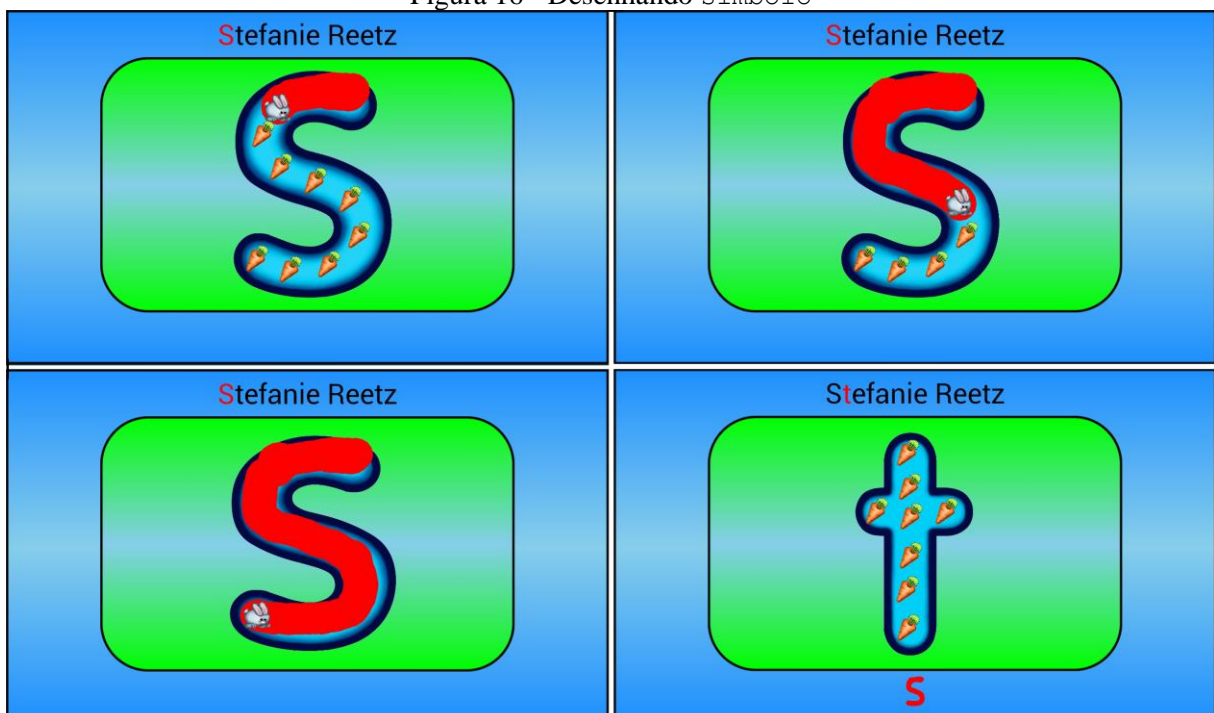
Ao clicar no botão **Escrever** da tela inicial, a tela de jogar plano será exibida para o Usuário. Esta tela é dividida em 3 principais partes, que são: pré-visualização de pranchas, símbolo principal e histórico de pranchas. A pré-visualização de pranchas localizada no canto superior da tela (Figura 15a) é a área reservada para o jogo listar todas as pranchas existentes no plano, e também terá função de destacar em vermelho a prancha atual. O símbolo principal localizado no centro da tela (Figura 15b) é a área reservada para o Usuário realizar a interação com o símbolo e consequentemente desenhá-lo. O histórico de pranchas localizada no canto inferior da tela (Figura 15c) é a área reservada para o jogo listar as pranchas já concluídas pelo Usuário.

Figura 15 - Tela jogar plano



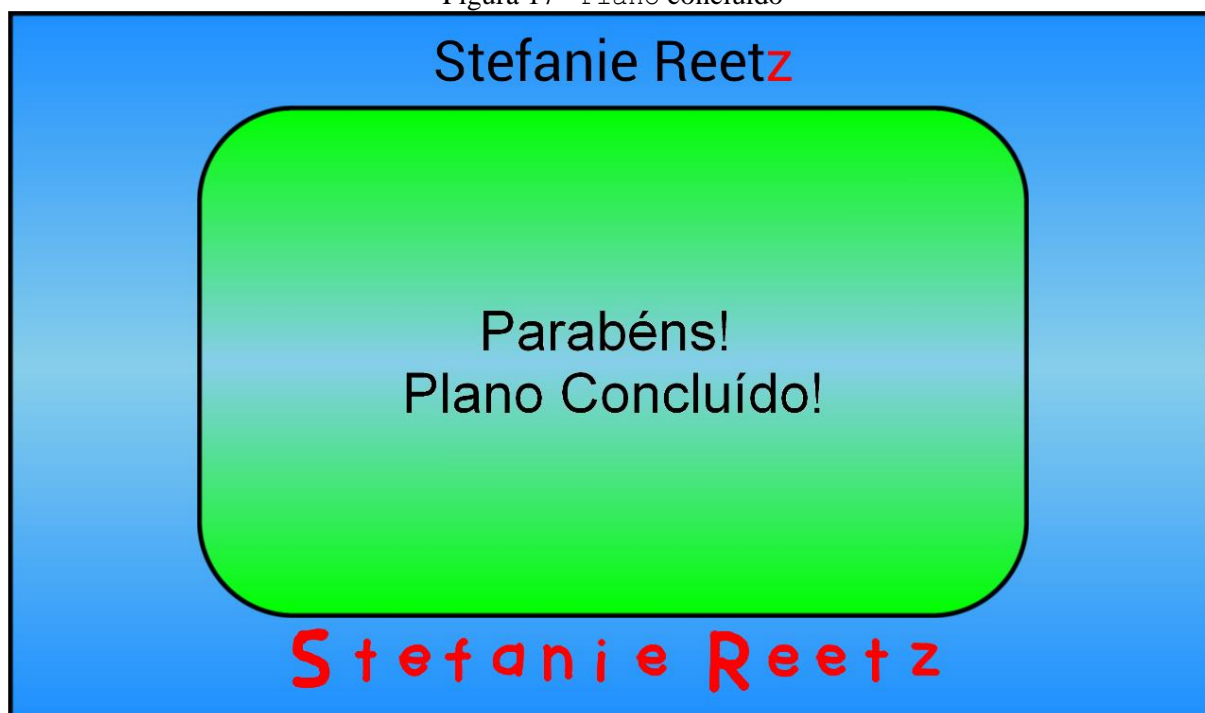
Com todas as pranchas carregadas na pré-visualização e primeiro símbolo posicionado no centro da tela, o Usuário deverá iniciar a interação tocando na tela sobre o conteúdo do símbolo. O Usuário terá que passar por todos os pontos de interesse para que o símbolo seja concluído com sucesso e a próxima prancha seja carregada. Não existe um ponto de partida, portanto usuário poderá iniciar da forma que desejar. A Figura 16 apresenta o símbolo sendo desenhado e por fim o jogo posicionando o próximo símbolo.

Figura 16 - Desenhando símbolo



O jogo só irá avançar para a próxima prancha quando o Usuário passar por todos os pontos de interesse. No momento que o Usuário concluir o desenho do símbolo o áudio correspondente ao símbolo será tocado e a próxima prancha será carregada. Quando todos os símbolos de todas as pranchas forem desenhadas corretamente o Usuário concluirá o plano. A Figura 17 apresenta a mensagem exibida para o Usuário ao concluir as atividades do plano.

Figura 17 - Plano concluído



3.4 RESULTADOS E DISCUSSÃO

Este trabalho apresenta uma protótipo de um jogo 2D de letras/números voltado para tecnologia assistiva na plataforma Android.

A aplicação foi desenvolvida utilizando os conceitos de modelagem orientada a componentes. Optou-se por utilizar este conceito pois proporciona o desenvolvimento de um componente independente capaz de ser acoplado e funcionar integrado e harmoniosamente no local utilizado. A utilização do componente criado neste trabalho está presente no cenário principal, onde o Usuário realizava as atividades, e no histórico de pranchas, onde cada símbolo presente no histórico é um componente identificados ao desenhado pelo Usuário. Concluiu-se que este conceito foi bem aplicado neste trabalho, pois facilitou a centralização dos principais recursos no componente desenvolvido.

Nas seções seguintes serão mostrados os resultados de usabilidade da aplicação. Posteriormente são realizados diagnósticos de consumo de memória e desempenho da aplicação. Por fim são comparados os resultados deste trabalho com os trabalhos correlatos.

3.4.1 Usabilidade

Atualmente na rede municipal de Blumenau existem cerca de 380 alunos distribuídos em 50 escolas e 77 Centros de Educação Infantil (CEIs) contando com 147 Professores de Apoio Pedagógico (PAPs), onde toda criança com necessidade **especial** é acompanhada por **um ou dois professores**.

Para realizar a análise de usabilidade desta aplicação foram realizados testes qualitativos com a professora Luscimar Rech Berkenbrock, onde ela respondeu uma entrevista usando o questionário do apêndice A. A professora Luscimar Rech Berkenbrock trabalha na rede municipal de ensino da cidade de Blumenau como PAP, e desde do início deste ano letivo (2013) trabalha com uma criança com necessidades especiais do 4^a ano na Escola Básica Municipal Machado de Assis.

Os resultados obtidos na entrevista foram excelentes levando em consideração os aspectos avaliados. No apêndice B apresentasse o jogo desenvolvido sendo testado por uma criança com necessidades especiais da professora Luscimar Rech Berkenbrock.

3.4.2 Memória e desempenho

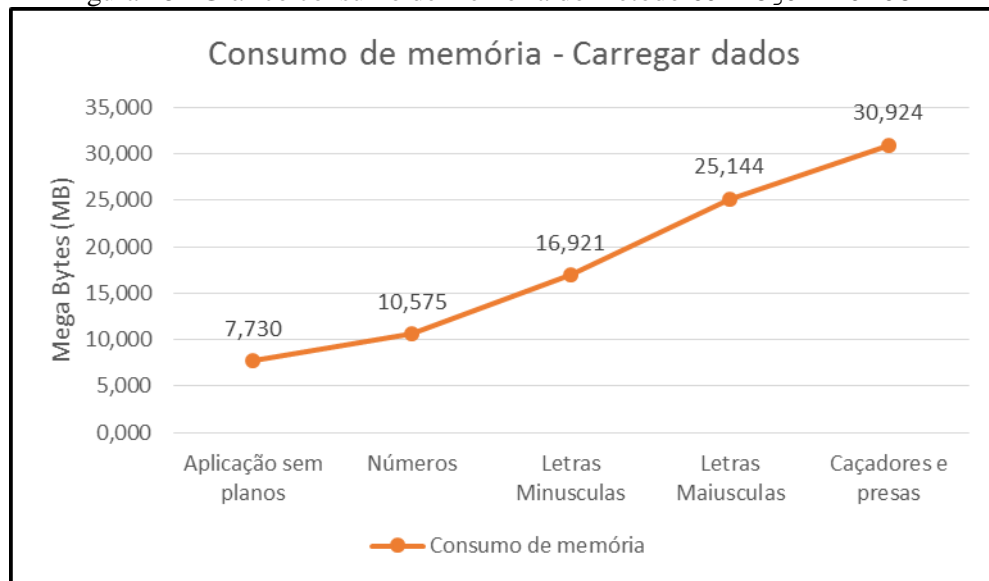
Esta seção analisa a memória e o desempenho nos testes realizados durante a utilização da aplicação. Serão avaliados o consumo de memória no momento em que os recursos são carregados para memória e também o tempo (em milissegundos) que a `símbolo` é carregado para o cenário.

Os testes foram realizados utilizando o aparelho Samsung Note 10.1 (descrito na seção 3.3.1) e não foram realizados testes utilizando o simulador do Android.

3.4.2.1 Consumo de memória

Primeiramente realizou-se o monitoramento do método `CarregarPlanosBD` da classe `Gerenciador`. Neste método é carregado todos os planos, pranchas e símbolos da base de dados para a memória da aplicação. A Figura 18 apresenta o gráfico do aumento do consumo de memória da aplicação, onde é possível visualizar o consumo inicial da aplicação até o final do método `CarregarPlanosBD`, onde todos os planos e imagens já foram carregados para memória.

Figura 18 - Gráfico consumo de memória do método CarregarPlanosBD



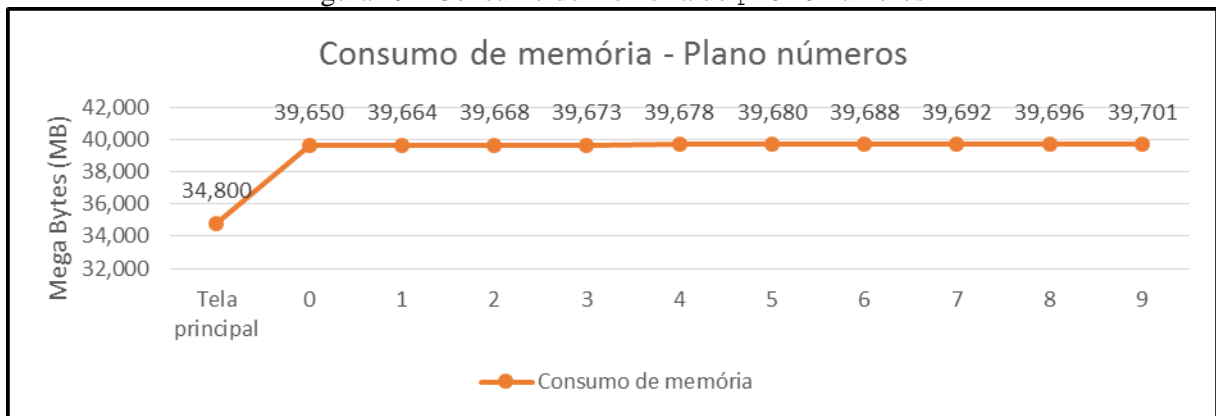
Já o Quadro 26 apresenta o consumo de memória utilizada para cada plano e agrupamento de caçadores e presas. Neste quadro também é apresentado a média de memória consumida para cada símbolo.

Quadro 26 - Consumo médio de memória por símbolo

	Memória utilizada	Quantidade de símbolos	Média por símbolo
Números	2,845MB	10	0,284MB
Letras minúsculas	6,346MB	26	0,244MB
Letras maiúsculas	8,223MB	26	0,316MB
Caçadores e presas	5,780MB	10	0,578MB

Para finalizar a análise do consumo de memória realizou-se o monitoramento dos planos sendo jogados, para que assim seja possível verificar a estabilização da memória durante a troca das pranchas. Na Figura 19 apresenta-se o gráfico de consumo de memória do plano de números, onde é constatado que a partir da primeira prancha (zero) até a última prancha (nove) o consumo de memória apresentou-se estável.

Figura 19 - Consumo de memória do plano números

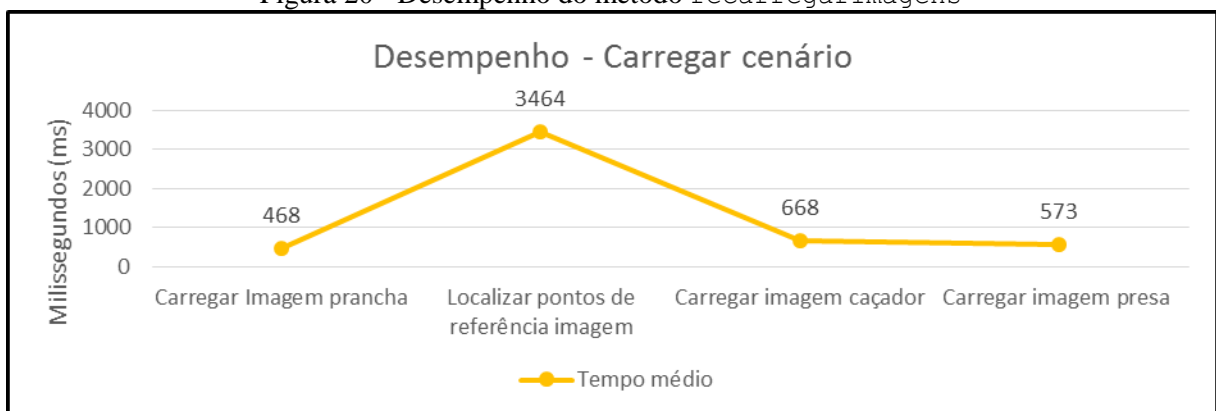


3.4.2.2 Desempenho do jogo

Para avaliar o desempenho do jogo, realizou-se o monitoramento do método `recarregarImagens` da classe `SimboloView`. Neste método são realizados os seguintes procedimentos: imagem da prancha é carregada, pontos de referência contidos na imagem são localizados, e por fim imagem do caçador e da presa são carregados, formando assim o novo cenário para ser jogado pelo Usuário. Na Figura 20 apresenta-se o gráfico de desempenho de cada procedimento realizado no método `recarregarImagens`.

Observa-se que o procedimento localizar pontos de referência imagem é o que consome o maior tempo deste método. Isto ocorre devido ao fato de que os pontos de referência estão contidos na propriedade `alpha` de cada *pixel* da imagem, portanto para localizar estes pontos de referência a rotina tem que varrer todos os *pixels* para localiza-los. As imagens utilizadas nos testes possuem dimensões de 1.000 x 1.000 *pixels*.

O tempo médio apresentado na Figura 20 foi extraído com base nos testes realizados com todas as pranchas dos planos: números, letras maiúsculas e letras minúsculas.

Figura 20 - Desempenho do método `recarregarImagens`

3.4.3 Comparativo entre o trabalho desenvolvido e os trabalhos correlatos

Nesta seção são apresentados a comparação entre as principais características deste trabalho com as dos trabalhos correlatos. No Quadro 27 é apresentado um comparativo entre este trabalho e os trabalhos correlatos.

Quadro 27 - Comparação com os trabalhos correlatos

	Jogo 2D desenvolvido	Fabeni (2012)	Dibugrama	Desenhe e Aprenda a Escrever
Apelo pedagógico	X	X	X	X
Possui vários cenários	X	X	X	X
Possibilita criar planos/cenários	X	X		X
Utiliza áudio	X	X	X	X
Aplicativo gratuito	X	X	X	
Possui planos/cenários nativos	X		X	X
Trabalha letras e números	X			X
Possibilita acentuação nas letras				X

Ao analisar o quadro, conclui-se que o jogo Desenhe e Aprenda a Escrever desenvolvido por FizzBrain (2013) para plataforma iOS é a aplicação mais próxima do trabalho desenvolvido, possuindo muitas características semelhantes e até sendo mais completo e robusto. Porém, o jogo 2D desenvolvido neste trabalho destaca-se por ser gratuito para plataforma Android e ainda por estar incorporado ao projeto Tagarela (2013), podendo assim ser melhor aproveitado pelos pacientes, tutores e especialista. Pois o projeto Tagarela (2013) possibilita além de usar os planos também a troca de informações entre estas pessoas.

4 CONCLUSÕES

Este trabalho apresentou o desenvolvimento de um protótipo de jogo 2D voltado para tecnologia assistiva na plataforma Android. O estudo mais aprofundado dos recursos disponível no SDK do Android, possibilitaram o desenvolvimento de rotinas e classes mais adequadas para cada ocasião. Em consequência disto um dos requisitos que era a utilização do guia de boas práticas de desenvolvimento do Android foi atingido.

Outro objetivo atendido refere-se à utilização de vários cenários e atividades, que permitam trabalhar com todas as letras e números da língua portuguesa. Para que este objetivo fosse atendido, foi necessário obter conhecimento sobre o formato de cada letra e número, a fim de criar formatos novos sem perder a característica da letra ou número. Para criação destes novos formatos utilizou-se a ferramenta de edição e criação de imagens Adobe Photoshop CS5, ferramenta na qual obteve-se o resultado esperado.

Durante o desenvolvimento da aplicação, encontraram-se alguns problemas de *out of memory* ao realizar o mapeamento de imagens em um `Bitmap`, isto ocorria quando as dimensões mapeadas no objeto ultrapassavam o total de memória disponível para a aplicação. Foi necessário encontrar outras soluções para resolver este problema, que no caso foram limitados as dimensões das imagens para 1.000 x 1.000 *pixels*.

O protótipo de jogo 2D desenvolvido apresentou algumas limitações, como por exemplo o tempo médio para a `prancha` ser exibida na tela, em torno de 5 segundos, o que pode causar certo desconforto para o usuário durante a sua utilização. Além do problema de desempenho, o jogo não possui tratamento para vogais com acentuação, ou seja, palavras ou nomes pessoais que possuem acentuação não podem ser usadas nos `planos` customizados, como por exemplo o nome `João` que pode ser utilizado no jogo apenas como `Joao`. Entretanto, estas situações não influenciaram no comprimento dos objetivos propostos.

4.1 EXTENSÕES

Sugerem-se as seguintes extensões para trabalhos futuros:

- a) implementar um editor de pontos de referências nas imagens dos `símbolos`;
- b) implementar variedade de texturas para o caminho do `símbolo` e possibilitar ao usuário selecioná-las;
- c) permitir que o usuário crie `planos` customizados com acentuações nas letras;
- d) analisar a possibilidade de utilizar um sintetizador de voz para reproduzir o significado da palavra ou frase dos `planos` customizados;

- e) analisar a possibilidade de incluir músicas de ambiente nos cenários;
- f) adaptar o aplicativo para todas as dimensões de tela disponível para plataforma Android;
- g) disponibilizar o aplicativo no Play Store da Google.

REFERÊNCIAS BIBLIOGRÁFICAS

ABI RESEARCH. **45 million Windows Phone and 20 million BlackBerry 10 smartphones in active use at year-end; enough to keep developers interested.** [S.l.], 2013. Disponível em: <<http://www.abiresearch.com/press/45-million-windows-phone-and-20-million-blackberry>>. Acesso em: 23 mar. 2013.

ANDROID. **What's new.** [S. l.], 2013. Disponível em: <<http://www.android.com/whatsnew>>. Acesso em: 13 abr. 2013.

ANDROID DEVELOPERS. **Developer tools.** [S. l.], 2013a. Disponível em: <<http://developer.android.com/tools/index.html>>. Acesso em: 13 abr. 2013.

_____. **Media playback.** [S. l.], 2013b. Disponível em: <<http://developer.android.com/guide/topics/media/mediaplayer.html>>. Acesso em: 13 abr. 2013.

_____. **Android.graphics.** [S. l.], 2013c. Disponível em: <<http://developer.android.com/reference/android/graphics/package-summary.html>>. Acesso em: 13 abr. 2013.

_____. **Android supported media formats.** [S. l.], 2013d. Disponível em: <<http://developer.android.com/guide/appendix/media-formats.html>>. Acesso em: 13 abr. 2013.

_____. **Android design.** [S. l.], 2013e. Disponível em: <<http://developer.android.com/design/index.html>>. Acesso em: 31 maio 2013.

_____. **Base64.** [S. l.], 2013f. Disponível em: <<http://developer.android.com/reference/android/util/Base64.html>>. Acesso em: 21 set. 2013.

ANDROID OPEN SOURCE PROJECT. **Philosophy and goals.** [S. l.], 2013. Disponível em: <<http://source.android.com/about/philosophy.html>>. Acesso em: 13 abr. 2013.

ASSOCIAÇÃO DE SÍNDROME DE DOWN DA REPÚBLICA ARGENTINA. [S. l.], 2013. Disponível em: <<http://www.asdra.org.ar>>. Acesso em: 31 mar. 2013.

BROWN, Alan W.; WALLNAU, Kurt C. **The current state of CBS.** [S. l.], 1998. Disponível em: <<http://www.idt.mdh.se/kurser/cd5490/2008/lectures/CBSE-Kurt.pdf>>. Acesso em: 21 maio 2013.

COMITÊ DE AJUDAS TÉCNICAS. **VII reunião do comitê de ajudas técnicas.** [S.l.], 2007. Disponível em: <http://www.comunicacaoalternativa.com.br/artigos-cientificos/Ata_VII_Reuni%C3%A3o_do_Comite_de_Ajudas_T%C3%A9cnicas.pdf?attredirects=0&d=1>. Acesso em: 31 mar. 2013.

FABENI, Alan F. C. **Tagarela:** aplicativo para comunicação alternativa no iOS. 2012. 107 f. Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Centro de Ciências Exatas e Naturais, Universidade Regional de Blumenau, Blumenau.

FIZZBRAIN. **Desenhe e Aprenda a Escrever.** [S. l.], 2013. Disponível em: <<https://itunes.apple.com/us/app/desenhe-e-aprenda-a-escrever!/id545187337?mt=8&ign-mpt=uo%3D4>> Acesso em: 20 set. 2013.

GLOBANT LABS. **Globant Labs.** [S. l.], 2013. Disponível em: <<http://labs.globant.com>>. Acesso em: 31 mar. 2013.

IAVORSKI, Joyce; VENDITTI, Rubens J. **A ludicidade no desenvolvimento e aprendizado da criança na escola:** reflexões sobre a educação física, jogo e inteligências múltiplas. [S. l.], 2008. Disponível em: <<http://www.efdeportes.com/efd119/a-ludicidade-no-desenvolvimento-e-aprendizado-da-crianca-na-escola.htm>>. Acesso em: 31 mar. 2013.

MANZINI, Eduardo J. **Tecnologia assistiva para educação:** recursos pedagógicos adaptados. Brasília: SEESP/MEC, 2005.

MARCO, Darlan D. de. **Tagarela:** Aplicativo de comunicação alternativa na plataforma Android. Blumenau, 2013. Disponível em: <https://bitbucket.org/daltonreis/tcc_darlanmarco>. Acesso em: 16 set. 2013.

MELO, Amanda M. **Acessibilidade:** discurso e prática no cotidiano das bibliotecas. Campinas: UNICAMP, 2006.

MOYLES, Janet R. **Só brincar?** O papel do brincar na educação infantil. Porto Alegre: Artmed, 2002.

RADABAUGH, Mary P. **Study on the financing of assistive technology devices of services for individuals with disabilities.** [S. l.], 1993. Disponível em: <<http://www.infoesp.net/recursos/recurso1.htm>>. Acesso em: 07 abr. 2013.

SANTOS, Cristina P. et al. **Projeto Infoacesso** - informática para portadores de deficiência. [S. l.], 2012. Disponível em: <http://www.reitoria.uri.br/~vivencias/Numero_014/artigos/artigos_vivencias_14/n14_19.pd>. Acesso em: 31 mar. 2013.

SPAGNOLI, Luciana A.; BECKER, Karin. **Um estudo sobre o desenvolvimento baseado em componentes.** Porto Alegre, 2003. Disponível em: <<http://www3.pucrs.br/pucrs/files/uni/poa/facin/pos/relatoriostec/tr026.pdf>>. Acesso em: 30 maio 2013.


TAGARELA. **Tagarela:** Plataforma de Comunicação Alternativa. Blumenau, 2013. Disponível em: <<http://gcg.inf.furb.br/tagarela>>. Acesso em: 16 set. 2013.

VINHAS, Laisa A. **A importância dos jogos educativos.** [S. l.], 2013. Disponível em: <http://jogos-educativos.info/mos/view/A_import%C3%A2ncia_dos_jogos_educativos>. Acesso em: 31 mar. 2013.

APÊNDICE A – Questionário de avaliação

Na Figura 21 é possível visualizar o questionário de avaliação realizado pela professora Luscimar Rech Berkenbrock.

Figura 21 - Questionário de avaliação



FURB - Universidade Regional de Blumenau
 Curso de Ciência da Computação – Bacharelado
 Projeto Tagarela – gcg.inf.furb.br/tagarela
 TCC - Jogo de letras/números voltado para tecnologia assistiva no Android

Questionário de Avaliação

1. Nome: Luscimar Rech Berkenbrock
2. A utilização de imagens para representar os pontos a serem percorridos pelo aluno ajudam a estimular o aprendizado?
 1 - ☐ Sim
 2 - ☐ Não
3. A utilização do áudio para representar o símbolo ajuda no reconhecimento da letra/número?
 1 - ☐ Não, ajudou
 2 - ☐ Ajudou pouco
 3 - ☐ Ajudou
 4 - ☐ Ajudou muito
4. Como você avalia a utilização sequencial de pranchas durante o jogo?
 1 - ☐ Ruim
 2 - ☐ Regular
 3 - ☐ Bom
 4 - ☒ Ótimo
5. Com base nas respostas anteriores, quais seriam os principais "Pontos Positivos" no uso do aplicativo?

+ estimulação visual do aplicativo incentiva, motiva a aluna.
6. Com base nas respostas anteriores, quais seriam os principais "Pontos Negativos" no uso do aplicativo?

Nada observado.
7. Com base nas respostas anteriores, quais seriam as suas "Sugestões" de melhoria para o aplicativo?

Os aplicativos estão bem elaborados. Acredito que uma sugestão seria elaborar aplicativos diversos, contribuindo cada vez mais para o conhecimento e crescimento da aluna.

Obs.: caso precise de espaço, favor usar o verso desta folha.

APÊNDICE B – Testes com criança com necessidades especiais

Na Figura 22 pode ser observado a criança com necessidades especiais utilizando o jogo desenvolvido neste trabalho.

Figura 22 - Jogo sendo testado por uma criança com necessidades especiais

